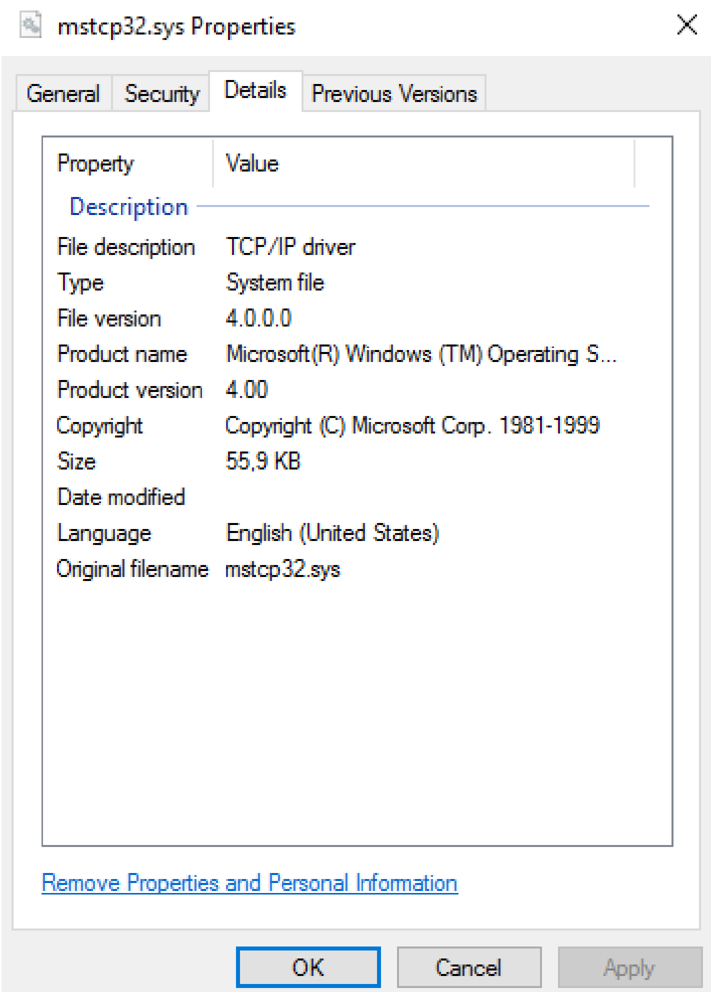
EquationDrug rootkit analysis (mstcp32.sys)

Malware arsenal that have been used by very sophisticated & so-called state-sponsored cyber group named "Equation Group" already was perfectly described by Kaspersky in their report. As always, it is hard to make an assumption about attribution of this malware as well as about origins of such elite cyber group. Anyway, it's obviously that code development and the cost of infrastructure for cyberattacks in such scale took enough human and money resources. As regular readers of my blog could notice, now I'm concentrating on research of rootkits allegedly belong to sophisticated/state-sponsored cyber actors. It is also interesting to assess skills of authors in driver development and compare it with code from another similar "products".

In the last year Equation Group group was hacked by another hacking group called Shadow Brokers, who claimed that got access to secret sources of NSA cyber toolkits. As we already know, SB released some exploits and backdoors for routers/network devices of some vendors that belong to EG. The last leak from SB was dedicated to set of PE-files, which used by Equation Group for cyberespionage and named EquationDrug. Analyzed driver mstcp32.sys was taken from this leak.

The driver mstcp32.sys (SHA256:26215BC56DC31D2466D72F1F4E1B6388E62606E9949BC41C28968FCB9A9D60A6) masked as "Microsoft TCP/IP driver".



Authors also took some steps to mask malicious purpose of this driver. For example, if you look to its imports or dump strings from file, you can't find something really suspicious. The driver imports API from NDIS kernel mode library called NDIS.SYS to work with network packets on physical level (that fully corresponds to its purpose). Actually, authors hid malicious indicators inside driver into encrypted data. Below you can see decrypted strings from driver's body.

```
System\CurrentControlSet\Services\
\Registry\Machine\System\CurrentControlSet\Services\
\Enum
Type
Start
ErrorControl
System\CurrentControlSet\Control\Class\{4D36E972-E325-11CE-BFC1-08002BE10318}\
Software\Microsoft\Windows NT\CurrentVersion\NetworkCards
\Linkage
\NDI\Interfaces
Group
Bind
Export
Route
ServiceName
UpperRange
LowerRange
RootDevice
UpperBind
PNP_TDI
ethernet
\Device\
services.exe
winlogon.exe
Processes
Options
ndiswanip
%SystemRoot%\System32\
<unknown>
Params
\Registry\Machine\System\CurrentControlSet\Control\Session Manager\Memory Management
ClearPageFileAtShutdown
\Registry\Machine\System\ControlSet
\Services\
\Enum\Root\LEGACY_
\SystemRoot\System32\Drivers\
```

As you can see from dumped strings above, the rootkit attaches itself to Windows network stack for capturing packets on NDIS level. Also, it is clear that the rootkit implements injection of malicious code into trusted Windows processes - Services.exe (SCM) & Winlogon.

Below you can see compilation date of this driver, which indicates that it was compiled already almost 10 years ago. This means that cyber espionage group used the rootkit and was active already in 2007. Also authors were interested to make their operations stealthy from user eyes, putting code into Ring 0.

SHA-1 26E787997A338D8111D96C9A4C103CF8FF0201CE

Analysis [File Header]

Name	Offset	Size	Value	Description
Machine	000000D4	2	014C	Intel 386
NumberOfSections	000000D6	2	0005	
TimeDateStamp	000000D8	4	47023CA6	Tue Oct 2 15:42:14 2007
PointerToSymbolTable	000000DC	4	00000000	
NumberOfSymbols	000000E0	4	00000000	
SizeOfOptionalHeader	000000E4	2	00E0	
Characteristics	000000E6	2	030E	Click here

Timestamp from debug directory matches with its analog from IMAGE_FILE_HEADER.

SHA-1 26E787997A338D8111D96C9A4C103CF8FF0201CE

Analysis [Debug Directory]

Name	Offset	Size	Value	Description
Characteristics	00000460	4	00000000	
TimeDateStamp	00000464	4	47023CA6	Tue Oct 2 15:42:14 2007
MajorVersion	00000468	2	0000	
MinorVersion	0000046A	2	0000	
Type	0000046C	4	00000004	Misc
SizeOfData	00000470	4	00000110	
AddressOfRawData	00000474	4	00000000	
PointerToRawData	00000478	4	0000DEE0	[offset]

Below you can see screenshot of start rootkit code.

```
.text:00012D22
.text:00012D22                  ; int __stdcall fnInitDriver(int pDrvObj,int punDrvRegPath,int pFunc1,int pFunc2,int Flag)
.text:00012D22                  fnInitDriver    proc near               ; CODE XREF: DriverEntry+15↓p
.text:00012D22
.text:00012D22                  unDevName       = dword ptr -14h
.text:00012D22                  var_10          = dword ptr -10h
.text:00012D22                  var_C           = dword ptr -0Ch
.text:00012D22                  var_8           = dword ptr -8
.text:00012D22                  pDeviceObject   = dword ptr -4
.text:00012D22                  pDrvObj         = dword ptr  8
.text:00012D22                  punDrvRegPath   = dword ptr  0Ch
.text:00012D22                  pFunc1          = dword ptr  10h
.text:00012D22                  pFunc2          = dword ptr  14h
.text:00012D22                  Flag            = dword ptr  18h
.text:00012D22
.text:00012D22                  edit_DrvObj = edi
.text:00012D22 55                               push    ebp
.text:00012D23 8B EC                            mov     ebp, esp
.text:00012D25 83 EC 14                         sub     esp, 14h
.text:00012D28 53                               push    ebx
.text:00012D29 56                               push    esi
.text:00012D2A 57                               push    edit_DrvObj
.text:00012D2B 68 44 05 00 00                   push    1348
.text:00012D30 68 28 A7 01 00                   push    offset unk_1A728
.text:00012D35 E8 60 EC FF FF                   call    fnDecryptData
.text:00012D35
.text:00012D3A FF 75 0C                         push    [ebp+punDrvRegPath]
.text:00012D3D 8B 7D 08                         mov     edit_DrvObj, [ebp+pDrvObj]
.text:00012D40 8D 45 F4                         lea     eax, [ebp+var_C]
.text:00012D43 89 3D D8 C3 01+                  mov     pDrvObj, edit_DrvObj
.text:00012D49 50                               push    eax
.text:00012D4A E8 13 0C 00 00                   call    fnDissectPath
.text:00012D4A
.text:00012D4F 8D 45 F4                         lea     eax, [ebp+var_C]
.text:00012D52 50                               push    eax
.text:00012D53 8D 45 EC                         lea     eax, [ebp+unDevName]
.text:00012D56 50                               push    eax
.text:00012D57 E8 C0 0C 00 00                   call    fnGetDeviceName
```

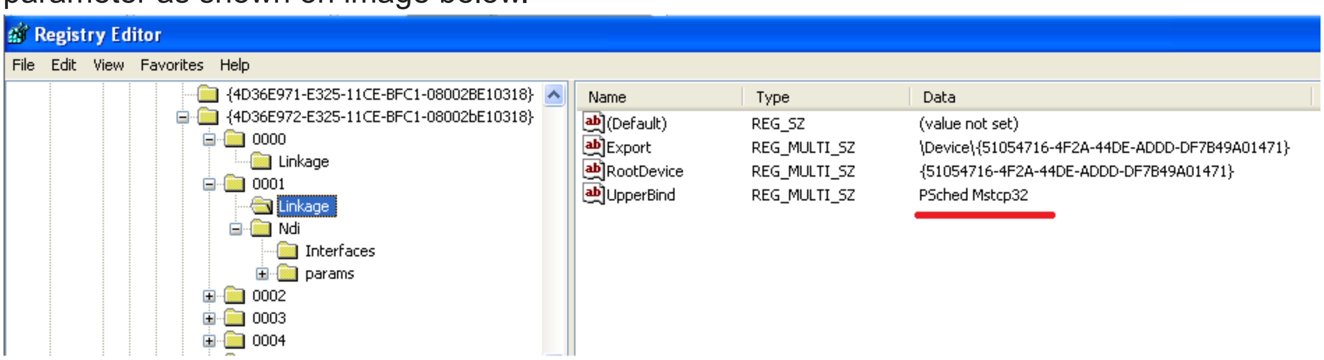
Malicious data decryption is a first step that takes the driver. After that it creates device object with name **\Device\Mstcp32** and performs initialization steps. The device name doesn't hard coded into driver's body, it forms on base of driver service name (**Mstcp32** as original name).

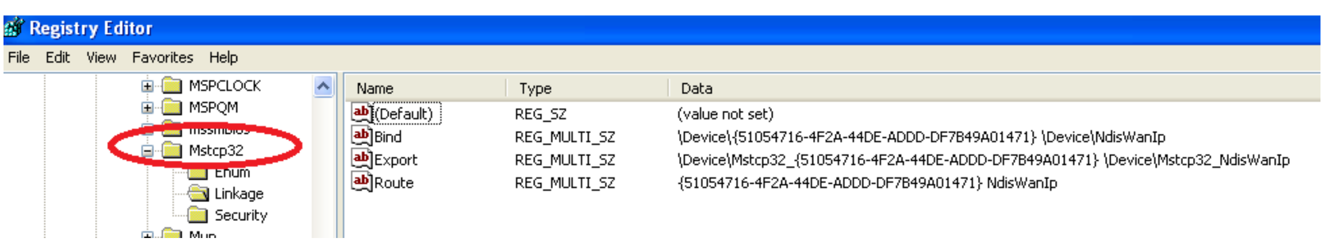
```
.text:00012DC1 53                          push    ebx
.text:00012DC2 FF 15 E0 02 01+             call    ds:NdisResetEvent
.text:00012DC8 8B 45 FC                    mov     eax, [ebp+pDeviceObject]
.text:00012DCB 83 48 1C 04                 or      dword ptr [eax+1Ch], 4
.text:00012DCF 8B 45 FC                    mov     eax, [ebp+pDeviceObject]
.text:00012DD2 A3 D4 C3 01 00              mov     dword_1C3D4, eax
.text:00012DD7 E8 FA EB FF FF              call    fnInitSpinLock
.text:00012DD7
.text:00012DDC C7 47 38 1E 30+             mov     dword ptr [edit_DrvObj+38h], offset fnDispatchIrpCreate
.text:00012DE3 C7 47 40 76 30+             mov     dword ptr [edit_DrvObj+40h], offset fnDispatchIrpClose
.text:00012DEA C7 47 44 AA 43+             mov     dword ptr [edit_DrvObj+44h], offset fnDispatchIrpReadWrite
.text:00012DF1 C7 47 48 AA 43+             mov     dword ptr [edit_DrvObj+48h], offset fnDispatchIrpReadWrite
.text:00012DF8 C7 87 80 00 00+             mov     dword ptr [edit_DrvObj+80h], offset fnDispatchIrpCleanup
.text:00012E02 C7 47 70 22 2F+             mov     dword ptr [edit_DrvObj+70h], offset fnDispatchIrpDeviceControl
.text:00012E09 C7 47 34 7A 2E+             mov     dword ptr [edit_DrvObj+34h], offset fnDriverUnload
.text:00012E10 E8 E7 0C 00 00              call    fn_IsNT4_WindowsVer
```

As you can see from image above, driver dispatches following IRP requests:

- IRP_MJ_CREATE
- IRP_MJ_CLOSE
- IRP_MJ_READ
- IRP_MJ_WRITE
- IRP_MJ_DEVICE_CONTROL
- IRP_MJ_CLEANUP.

The driver registers itself as NDIS filter. It checks interface with GUID {4d36e972-e325-11ce-bfc1-08002be10318} (that located into encrypted part of data) and gets list of instances that already registered in Windows. It tries to find specific instance with value LowerRange == "ethernet" into HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\000X\Ndi\Interfaces. After driver code found it, it appends own value to this parameter as shown on image below.





As I already mentioned above, the rootkit was written by authors in 2007, so range of supported Windows versions is extremely small comparing with nowadays malware.

Moreover, like other rootkits authors in that time, they use a lot of undocumented fields in kernel mode objects for retrieving the data they need. Next Windows NT versions are supported by the rootkit.

- Windows NT 4.0 (1381)
- Windows 2000 (2195)
- Windows XP (2600)
- Windows Server 2003 (3790)

```
.text:0001600E                jIsWindowsXP:                          ; CODE XREF: fnGetUndocOffsetsOrFuncs+34↑j
.text:0001600E B8 B0 01 00 00         mov     eax, 1B0h
.text:00016013 C7 05 70 C1 01+        mov     offs_kthread_alertable, 164h
.text:0001601D C7 05 6C C1 01+        mov     offs_kprocess_threadlisthead, 50h
.text:00016027 A3 74 C1 01 00         mov     offs_kthread_threadlistentry, eax
.text:0001602C C7 05 68 C1 01+        mov     offs_eprocess_activeprocesslinks, 88h
.text:00016036 C7 05 78 C1 01+        mov     offs_eprocess_activeprocesslinksprev, 8Ch
.text:00016040 C7 05 50 C1 01+        mov     offs_eprocess_Pid, 84h
.text:0001604A C7 05 88 C1 01+        mov     offs_eprocess_imagefilename, 174h
.text:00016054 A3 48 C1 01 00         mov     offs_eprocess_peb, eax
.text:00016059 E9 A2 00 00 00         jmp     loc_16100
.text:00016059
.text:0001605E                ; ---------------------------------------------------------------------------
.text:0001605E
.text:0001605E                jIsW2k:                                ; CODE XREF: fnGetUndocOffsetsOrFuncs+29↑j
.text:0001605E C7 05 70 C1 01+        mov     offs_kthread_alertable, 158h
.text:00016068 C7 05 6C C1 01+        mov     offs_kprocess_threadlisthead, 50h
.text:00016072 C7 05 74 C1 01+        mov     offs_kthread_threadlistentry, 1A4h
.text:0001607C C7 05 68 C1 01+        mov     offs_eprocess_activeprocesslinks, 0A0h
.text:00016086 C7 05 78 C1 01+        mov     offs_eprocess_activeprocesslinksprev, 0A4h
.text:00016090 C7 05 50 C1 01+        mov     offs_eprocess_Pid, 9Ch
.text:0001609A C7 05 88 C1 01+        mov     offs_eprocess_imagefilename, 1FCh
.text:000160A4 C7 05 48 C1 01+        mov     offs_eprocess_peb, 1B0h
.text:000160AE EB 50                  jmp     short loc_16100
.text:000160AE
.text:000160B0                ; ---------------------------------------------------------------------------
.text:000160B0
.text:000160B0                jIsNT4:                                ; CODE XREF: fnGetUndocOffsetsOrFuncs+1E↑j
.text:000160B0 C7 05 70 C1 01+        mov     offs_kthread_alertable, 158h
.text:000160BA C7 05 6C C1 01+        mov     offs_kprocess_threadlisthead, 50h
.text:000160C4 C7 05 74 C1 01+        mov     offs_kthread_threadlistentry, 1A4h
.text:000160CE C7 05 68 C1 01+        mov     offs_eprocess_activeprocesslinks, 98h
.text:000160D8 C7 05 78 C1 01+        mov     offs_eprocess_activeprocesslinksprev, 9Ch
.text:000160E2 C7 05 50 C1 01+        mov     offs_eprocess_Pid, 94h
.text:000160EC C7 05 88 C1 01+        mov     offs_eprocess_imagefilename, 1DCh
.text:000160F6 C7 05 48 C1 01+        mov     offs_eprocess_peb, 18Ch
.text:000160F6 00 8C 01 00 00
```

You can see that the rootkit uses various undocumented offsets in EPROCESS and ETHREAD kernel objects for some purposes, including, enumerating running processes and threads, checking thread alertable state, retrieving pointer to PEB and etc.

Injection of malicious code into processes is made in usual for such rootkits manner: Attach_To_Process->Allocate_Virtual_Memory->InsertApc.

```
.text:00016852
.text:00016854 53                          push    ebx
.text:00016855 55                          push    ebp
.text:00016856 FF 35 30 C1 01+             push    dword_1C130
.text:0001685C E8 83 3D 00 00              call    KeAttachProcess
.text:0001685C
.text:00016861 A1 04 C1 01 00              mov     eax, cbAllocatedUserModeCode
.text:00016866 6A 40                       push    40h
.text:00016868 BD 0C C1 01 00              mov     ebp, offset cbRegionSize
.text:0001686D 68 00 10 00 00              push    1000h
.text:00016872 55                          push    ebp
.text:00016873 BB 08 C1 01 00              mov     ebx, offset pApcRoutine_1
.text:00016878 6A 00                       push    0
.text:0001687A 53                          push    ebx
.text:0001687B 6A FF                       push    0FFFFFFFFh
.text:0001687D A3 0C C1 01 00              mov     cbRegionSize, eax
.text:00016882 FF 15 EC 03 01+             call    ds:ZwAllocateVirtualMemory
.text:00016888 8B 3D 08 C1 01+             mov     edi, pApcRoutine_1
.text:0001688E 85 FF                       test    edi, edi
.text:00016890 74 34                       jz      short loc_168C6
.text:00016890
.text:00016892 8B 0D 04 C1 01+             mov     ecx, cbAllocatedUserModeCode
.text:00016898 56                          push    esi
.text:00016899 8B 35 00 C1 01+             mov     esi, pAllocatedUserModeCode
.text:0001689F 8B C1                       mov     eax, ecx
.text:000168A1 C1 E9 02                    shr     ecx, 2
.text:000168A4 F3 A5                       rep movsd
.text:000168A6 8B C8                       mov     ecx, eax
.text:000168A8 83 E1 03                    and     ecx, 3
.text:000168AB F3 A4                       rep movsb
.text:000168AD E8 9C 02 00 00              call    fnInsertApc
.text:000168AD
```

```
.text:00018E7A
.text:00018E7A              fnPartOfUserModeCode proc near          ; DATA XREF: sub_18636+123↑o
.text:00018E7A                                                      ; sub_18636+12B↑o ...
.text:00018E7A
.text:00018E7A              var_10          = dword ptr -10h
.text:00018E7A              hProcess        = dword ptr -0Ch
.text:00018E7A              var_8           = dword ptr -8
.text:00018E7A              var_4           = dword ptr -4
.text:00018E7A
.text:00018E7A 55                   push    ebp
.text:00018E7B 8B EC                mov     ebp, esp
.text:00018E7D 83 EC 10             sub     esp, 10h
.text:00018E80 53                   push    ebx
.text:00018E81 56                   push    esi
.text:00018E82 57                   push    edi
.text:00018E83 89 55 FC             mov     [ebp+var_4], edx
.text:00018E86 8B CA                mov     ecx, edx
.text:00018E88 83 C1 1C             add     ecx, 1Ch
.text:00018E8B 89 4D F8             mov     [ebp+var_8], ecx
.text:00018E8E C7 45 F4 00 00+      mov     [ebp+hProcess], 0
.text:00018E95 C7 45 F0 00 00+      mov     [ebp+var_10], 0
.text:00018E9C 8B 4D F8             mov     ecx, [ebp+var_8]
.text:00018E9F 8B 59 04             mov     ebx, [ecx+4]
.text:00018EA2 53                   push    ebx             ; dwProcessId
.text:00018EA3 6A 00                push    0               ; bInheritHandle
.text:00018EA5 68 FF 0F 1F 00       push    1F0FFFh         ; dwDesiredAccess
.text:00018EAA 8B 55 FC             mov     edx, [ebp+var_4]
.text:00018EAD 8B 5A 0C             mov     ebx, [edx+0Ch]
.text:00018EB0 FF D3                call    ebx             ; OpenProcess
.text:00018EB2 89 45 F4             mov     [ebp+hProcess], eax
.text:00018EB5 83 F8 00             cmp     eax, 0
.text:00018EB8 0F 84 8A 00 00+      jz      jRet
.text:00018EB8 00
```

Conclusion

Unlike authors of other state-sponsored rootkits that were already mentioned in my blog, authors of mstcp32.sys don't rely on Windows native API for performing some operations, for example, for enumeration processes and threads. Instead this, they use undocumented kernel objects offsets for retrieving some data mentioned above. A significant portion of code in rootkit body is NDIS-oriented and dedicated to communication with network. There are a lot of Windows kernel rules for correctly organizing communication between NDIS driver and other parts of OS.

The rootkit driver supports IOCTL for sending data over network on NDIS level. This means that network logic of communicating with remote host is located into user mode part that use driver for this purpose.

```
.text:00012FB9                      jDispatchIOCTLSendData:                 ; CODE XREF: fnDispatchIrpDeviceControl+61↑j
.text:00012FB9 8B 7E 0C                         mov     edi, [esi+0Ch]  ; ->AssociatedIrp.SystemBuffer
.text:00012FBC 8D 45 08                         lea     eax, [ebp+DeviceObject_later_zero]
.text:00012FBF 53                               push    ebx_later_zero
.text:00012FC0 50                               push    eax
.text:00012FC1 53                               push    ebx_later_zero
.text:00012FC2 53                               push    ebx_later_zero
.text:00012FC3 68 01 00 0F 00                   push    0F0001h
.text:00012FC8 89 5D 08                         mov     [ebp+DeviceObject_later_zero], ebx_later_zero
.text:00012FCB FF 37                            push    dword ptr [edi]
.text:00012FCD FF 15 74 03 01+                  call    ds:ObReferenceObjectByHandle
.text:00012FD3 8B D8                            mov     ebx_later_zero, eax
.text:00012FD5 85 DB                            test    ebx_later_zero, ebx_later_zero
.text:00012FD7 75 1E                            jnz     short loc_12FF7 ; IRP->IoStatus.Info
.text:00012FD7
.text:00012FD9 FF 77 04                         push    dword ptr [edi+4] ; pBuffer2
.text:00012FDC 83 C7 08                         add     edi, 8
.text:00012FDF 57                               push    edi             ; pBuffer
.text:00012FE0 FF 75 08                         push    [ebp+DeviceObject_later_zero] ; Zero
.text:00012FE3 E8 74 05 00 00                   call    fnSendData
.text:00012FE3
.text:00012FE8 8B D8                            mov     ebx_later_zero, eax
.text:00012FEA 85 DB                            test    ebx_later_zero, ebx_later_zero
.text:00012FEC 7D 09                            jge     short loc_12FF7 ; IRP->IoStatus.Info
.text:00012FEC
.text:00012FEE 8B 4D 08                         mov     ecx, [ebp+DeviceObject_later_zero]
.text:00012FEE
.text:00012FF1
.text:00012FF1                      loc_12FF1:                              ; CODE XREF: fnDispatchIrpDeviceControl+95↑j
.text:00012FF1 FF 15 78 03 01+                  call    ds:ObfDereferenceObject
```