

Dissecting APT21 samples using a step-by-step approach

cybergEEKs.tech/dissecting-apt21-samples-using-a-step-by-step-approach/

Summary

In this blog post we're presenting a detailed analysis of 2 malicious files (a backdoor known as "Travelnet") linked to an APT (Advanced Persistent Threat) actor called APT21.

APT21, also known as Zhenbao or Hammer Panda, is a group of suspected state sponsored hackers of Chinese origin.

According to multiple online sources, that we have referenced in the article, APT 21 historically targeted the Russian government and groups which seek greater autonomy or independence from China, such as those from Tibet or Xinjiang.

The first file is a dropper used to register a malicious DLL (NetTraveler trojan) as a service. The main purpose of the trojan is to gather information about the environment such as user name, host name, IP address of the host, Windows OS version, different configurations of the CPU, information about memory consumption, the list of processes. The malicious process is interested in .doc, .docx, .xls, .xlsx, .txt, .rtf, .pdf files on disk and also on USB drives and network shares in order to exfiltrate them. During the entire infection, multiple .ini configuration files are created and also the malware has the capability to download and execute additional files on the infected machine. The data is compressed using a custom Lempel-Ziv-based algorithm and encoded with a modified Base64 algorithm before it will be exfiltrated to the Command and Control server.

Technical analysis

Section I

Dropper

SHA256: FECA8DB35C0C0A901556EFF447C38614D14A7140496963DF2E613B206527B338

One of the first steps the malware is performing consists of creating a mutex called "INSTALL SERVICES NOW!" (note the space). The mutex is used to avoid reinfection of an already infected machine:

```
.text:00401000 push    ebp
.text:00401001 mov     ebp, esp
.text:00401003 sub     esp, 208h
.text:00401009 push    esi
.text:0040100A push    offset Name          ; "INSTALL SERVICES NOW!"
.text:0040100F push    1                    ; bInitialOwner
.text:00401011 push    0                    ; lpMutexAttributes
.text:00401013 call    ds:CreateMutexA
.text:00401019 mov     esi, eax
.text:0040101B call    ds:GetLastError
.text:00401021 cmp     eax, 0B7h ; '.'
.text:00401026 jz     loc_4010C2
```

Figure 1

The malicious process creates a configuration file at "C:\Windows\System\config_t.dat" which will be heavily used during the entire infection. The API call used to accomplish this task is CreateFileA and it's presented in figure 2:

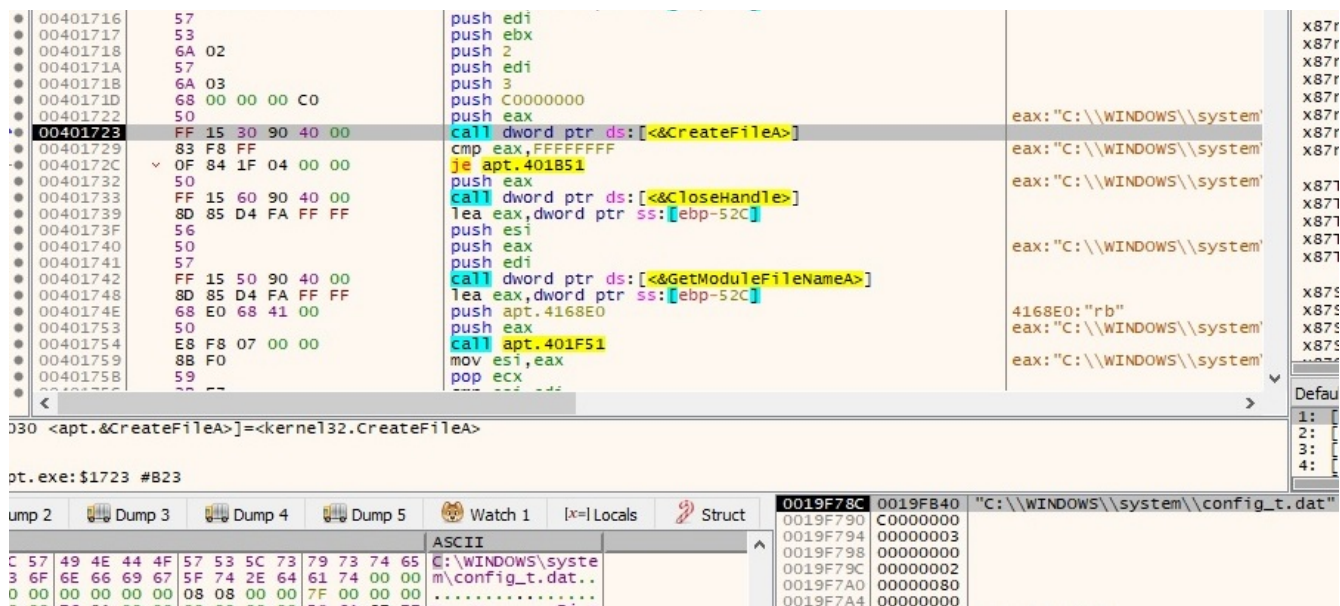


Figure 2

The following bytes found at a precise location in the malicious file are read in order to decrypt them:

Address	Hex	ASCII
0019FAC0	56 4A 4A 4E 04 11 11 49 49 49 10 48 57 4E 53 5F	VJJN...III.HWNS_
0019FAD0	57 52 4C 4B 10 5D 51 53 11 50 5B 49 4D 57 50 58	WRLK.]QS.P[IMWPX
0019FAE0	51 11 0F 0F 50 4A 11 50 5B 4A 4A 4C 5F 48 5B 52	Q...P].P[J]L_H[R
0019FAF0	5B 4C 10 5F 4D 4E 00 00 00 00 00 00 00 00 00	[L..MN.....

Figure 3

The decryption routine is shown in the next figure and consists of a XOR operation with 0x3E:

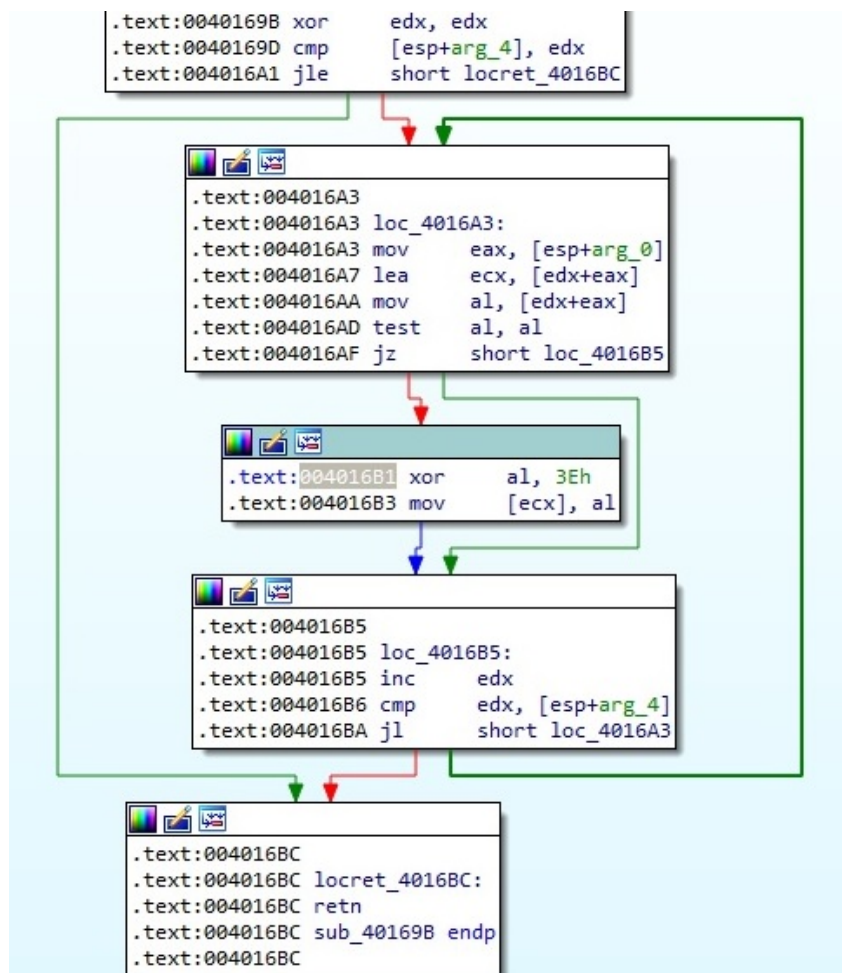


Figure 4

After the decryption is over the new string represents a URL which contains the C2 server as we'll see later on:

Address	Hex	ASCII
0019FAC0	68 74 74 70	http://www.vipma
0019FAD0	69 6C 72 75	ilru.com/newsinf
0019FAE0	6F 2F 31 31	o/1int/nettrave
0019FAF0	65 72 2E 61	er.asp.....

Figure 5

The configuration file is populated using WritePrivateProfileStringA API calls as shown below. Please note that WebPage is equal to the string decrypted above and the others options will be explained later on in a better context:

esi=<kernel32.WritePrivateProfileStringA> (763591A0)

.text:004019DF apt.exe:\$19DF #DDF

Address	Hex	ASCII
0019F798	00416640	"Option"
0019F79C	00416804	"WebPage"
0019F7A0	0019F9BC	"http://www.vipmailru.com/newsinfo/1int/nettraveller.asp"
0019F7A4	0019FB40	"C:\\WINDOWS\\system\\config_t.dat"

Figure 6

esi=<kernel32.WritePrivateProfileStringA> (763591A0)

.text:00401A15 apt.exe:\$1A15 #E15

Address	Hex	ASCII
0019F798	00416640	"Option"
0019F79C	004168C4	"DownCmdTime"
0019F7A0	0019FCDC	"10"
0019F7A4	0019FB40	"C:\\WINDOWS\\system\\config_t.dat"

Figure 7

esi=<kernel32.WritePrivateProfileStringA> (763591A0)

.text:00401A4B apt.exe:\$1A4B #E4B

Address	Hex	ASCII
0019F798	00416640	"Option"
0019F79C	004168B8	"UploadRate"
0019F7A0	0019FCDC	"128"
0019F7A4	0019FB40	"C:\\WINDOWS\\system\\config_t.dat"

Figure 8

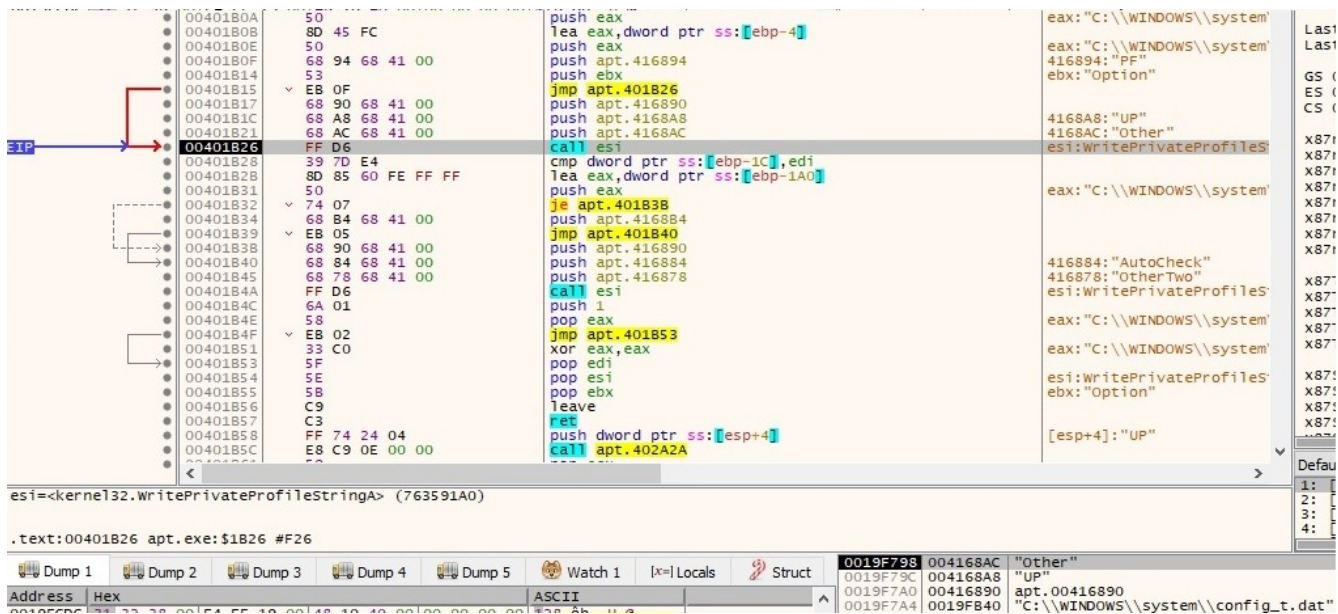


Figure 9

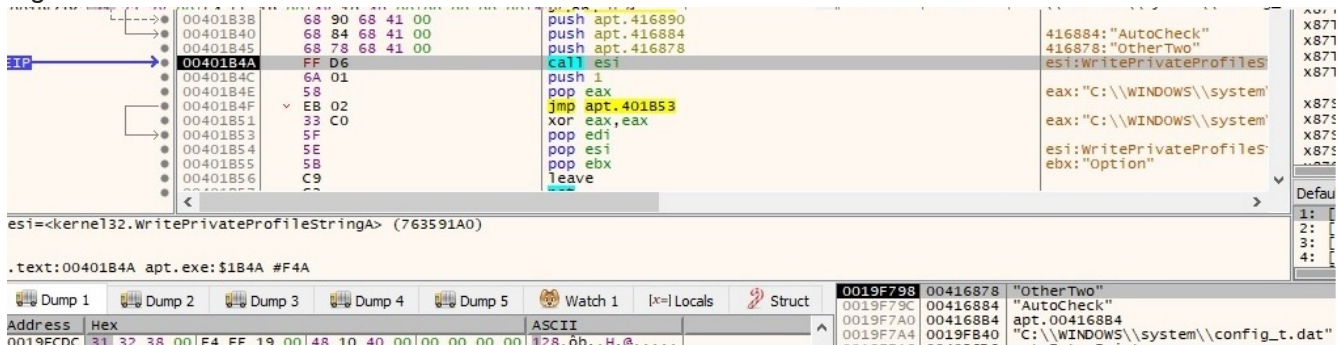


Figure 10

After all of these API calls the configuration file has the following schema:

```

config_t.dat
1 [Option]
2 WebPage=http://www.vipmailru.com/newsinfo/llnt/nettraveler.asp
3 DownCmdTime=10
4 UploadRate=128
5 ServiceName=FastUserSwitchingCompatibility
6 [Other]
7 UP=0
8 [OtherTwo]
9 AutoCheck=1

```

Figure 11

Now there is a byte at offset 0x334 in the file which indicates if the malicious process is supposed to use a proxy or not (by default this value is 0 and UP=0 means the malware is not using a proxy for network communications). If that byte is set to 1, the malware writes UP=1 in the configuration file and also 5 additional values: PS (proxy address), PP (proxy port), PU (proxy user), PW (proxy password) and PF (unknown). RegQueryValueExA API is used to retrieve the type and data for netsvcs (svchost.exe) associated with "HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Svchost":

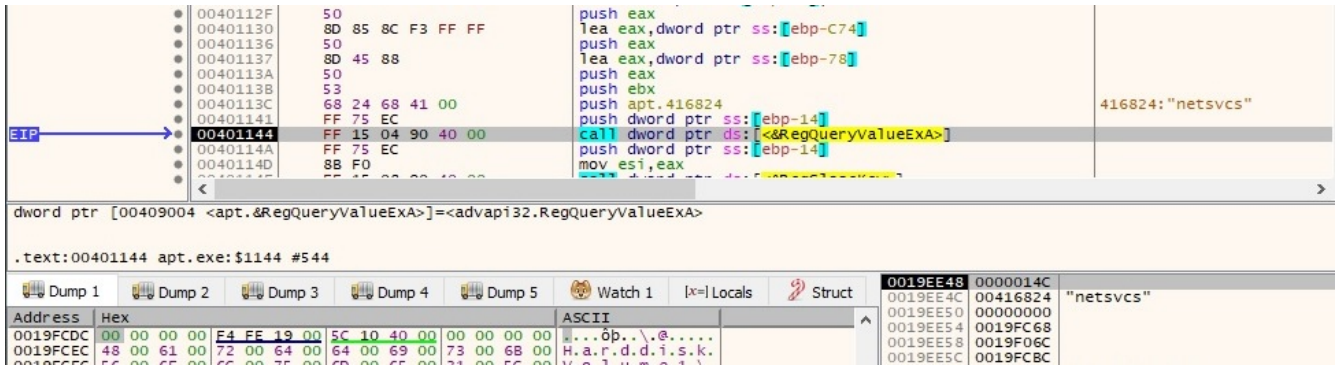


Figure 12

The malicious file enumerates all the available services on the host and compares them with a hardcoded list presented in figure 13. The first service which is not found on the system will be used for malicious purposes as we'll describe further.

Address	Hex	ASCII
0019F06C	43 65 72 74 50 72 6F 70 53 76 63 00 53 43 50 6F	CertPropSvc.SCPo
0019F07C	6C 69 63 79 53 76 63 00 6C 61 6E 6D 61 6E 73 65	licySvc.Ianmanse
0019F08C	72 76 65 72 00 67 70 73 76 63 00 69 70 68 6C 70	rver.gpsvc.iphlp
0019F09C	73 76 63 00 6D 73 69 73 63 73 69 00 73 63 68 65	svc.msiscsi.sche
0019F0AC	64 75 6C 65 00 77 69 6E 6D 67 6D 74 00 53 65 73	dule.winmgmt.Ses
0019F0BC	73 69 6F 6E 45 6E 76 00 46 61 73 74 55 73 65 72	sionEnv.FastUser
0019F0CC	53 77 69 74 63 68 69 6E 67 43 6F 6D 70 61 74 69	SwitchingCompati
0019F0DC	62 69 6C 69 74 79 00 49 61 73 00 49 72 6D 6F 6E	bility.Ias.Irmon
0019F0EC	00 4E 6C 61 00 4E 74 6D 73 73 76 63 00 4E 57 43	.Nla.Ntmssvc.NWC
0019F0FC	57 6F 72 68 73 74 61 74 69 6F 6E 00 4E 77 73 61	Workstation.Nwsa
0019F10C	70 61 67 65 6E 74 00 52 61 73 61 75 74 6F 00 52	pagent.Rasauto.R
0019F11C	61 73 6D 61 6E 00 52 65 6D 6F 74 65 61 63 63 65	asman.Remoteacce
0019F12C	73 73 00 53 45 4E 53 00 53 68 61 72 65 64 61 63	ss.SENS.Sharedac
0019F13C	63 65 73 73 00 53 52 53 65 72 76 69 63 65 00 54	cess.SRService.T
0019F14C	61 70 69 73 72 76 00 57 6D 69 00 57 6D 64 6D 50	apisrv.Wmi.WmdmP
0019F15C	6D 53 70 00 77 75 61 75 73 65 72 76 00 42 49 54	mSp.wuauerv.BIT
0019F16C	53 00 53 68 65 6C 6C 48 57 44 65 74 65 63 74 69	S.ShellHWDetecti
0019F17C	6F 6E 00 4C 6F 67 6F 6E 48 6F 75 72 73 00 50 43	on.LogonHours.PC
0019F18C	41 75 64 69 74 00 68 65 6C 70 73 76 63 00 75 70	Audit.helpsvc.up
0019F19C	6C 6F 61 64 6D 67 72 00 54 6F 68 65 6E 42 72 6F	loadmgr.TokenBro
0019F1AC	68 65 72 00 55 73 65 72 4D 61 6E 61 67 65 72 00	ker.UserManager.
0019F1BC	41 70 70 4D 67 6D 74 00 00 00 6F 00 00 00 52 00	AppMgmt...o...R.

Figure 13

The strategy is as follows: it will enumerate the keys corresponding to a service like "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<ServiceName>" in order to see if the service is installed or not. The following services have been present on the analyzing machine: CertPropSvc, SCPolicySvc, lanmanserver, gpsvc, iphlpvc, msiscsi, schedule, winmgmt, SessionEnv and the first one which was missing is FastUserSwitchingCompatibility. RegOpenKeyExA API is utilized to check for the existence of the services, one such example is detailed in the figure below:

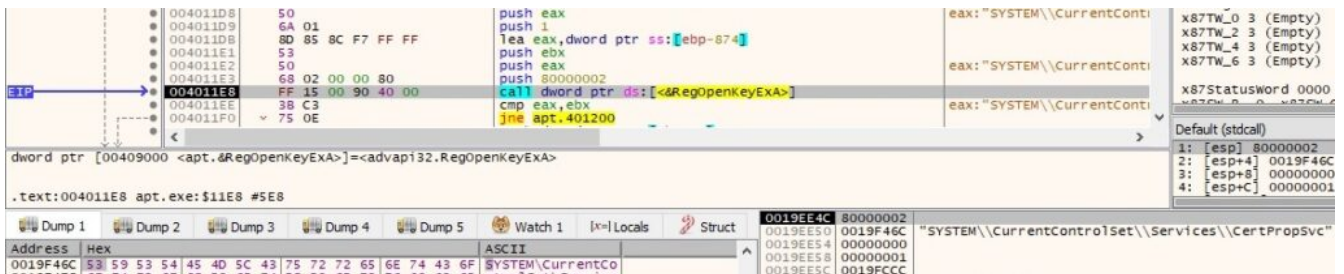


Figure 14

The file "C:\WINDOWS\system32\FastUserSwitchingCompatibilityex.dll" associated with FastUserSwitchingCompatibility service is supposed to be deleted by the running process (it doesn't exist on the machine):

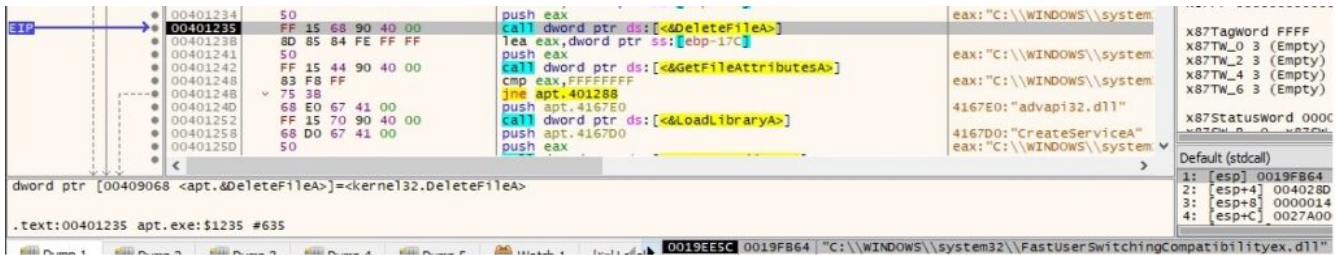


Figure 15

A new service called “FastUserSwitchingCompatibility” is created using CreateServiceA API function which tries to impersonate the legitimate service, the binary path of the service being %SystemRoot%\System32\svchost.exe -k netsvcs (legitimate process):

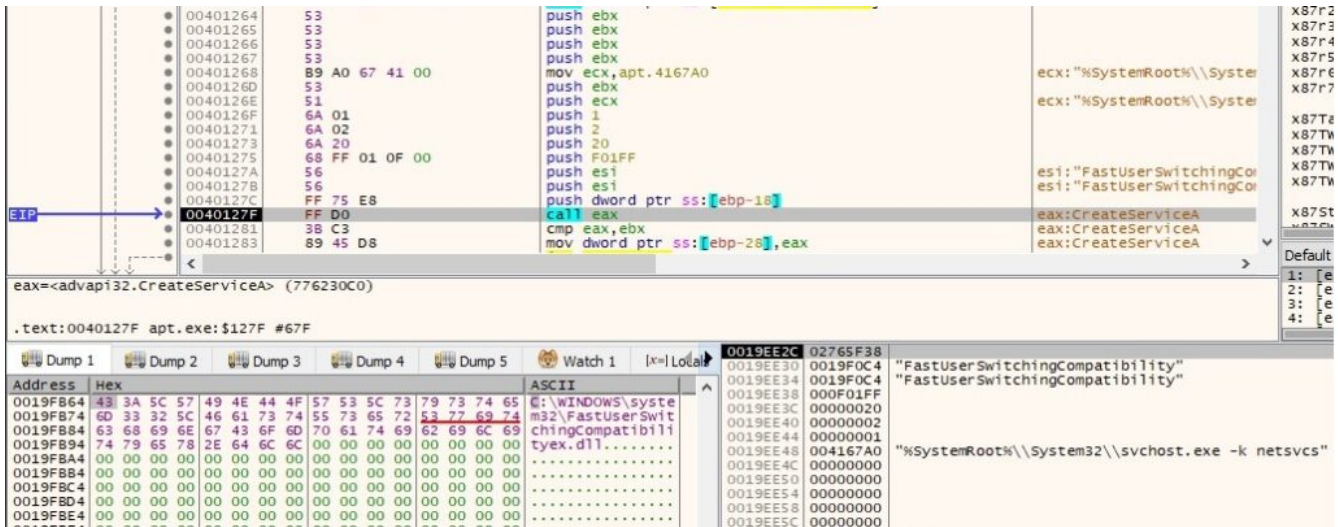


Figure 16

If the call is successful we’ll see a registry key like the one displayed in figure 17. (this technique is part of evasion techniques). Attackers will try to impersonate/use legitimate system binaries or libraries on the host to hide malicious activity. This will allow them to blend with regular activity and remain hidden. (you can find more details about lolbins at <https://lolbas-project.github.io/>).

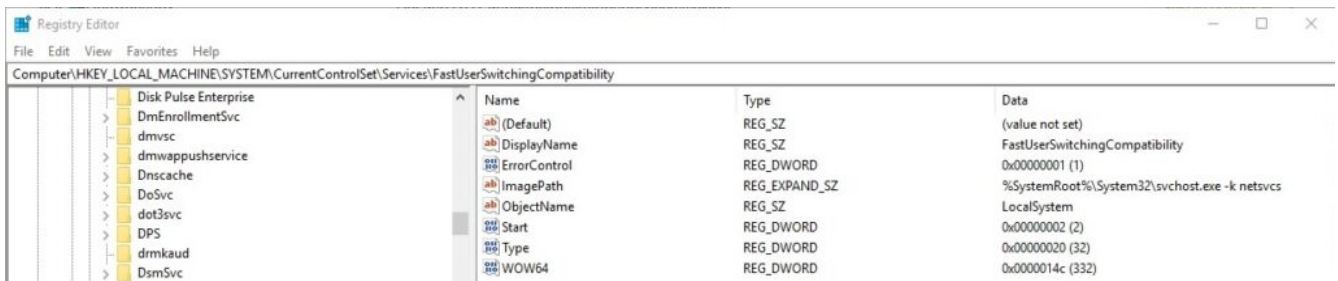


Figure 17

In order to verify that the service was successfully created the malicious process tries to open “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility” (now it exists because it corresponds to the newly created service):

Figure 18

A new key called “Parameters” is created under “HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\FastUserSwitchingCompatibility” using RegCreateKeyA API. This will be used to register a malicious DLL as a service:

Figure 19

The process creates an empty file called temp.bat in the same directory as the initial executable (in our case, Desktop). The content of the batch file is shown below:

Figure 20

The purpose of the batch file is to register the DLL found at “C:\\WINDOWS\\system32\\FastUserSwitchingCompatibilityex.dll” as a service by adding “ServiceDll” entry. File “C:\\WINDOWS\\system32\\FastUserSwitchingCompatibilityex.dll” doesn’t exist at this time, however it’s created by the malware using CreateFileA API as shown below (it will be populated with malicious code as we’ll see in a bit):

Figure 21

“Timestomping” is a technique used by a malicious actor to modify files’ timestamps (for example created/modified timestamps) in order not to raise any suspicions about the file. In our case the created and modified timestamps of the DLL file are set to Tuesday, August 17, 2004, 9:00:00 PM:

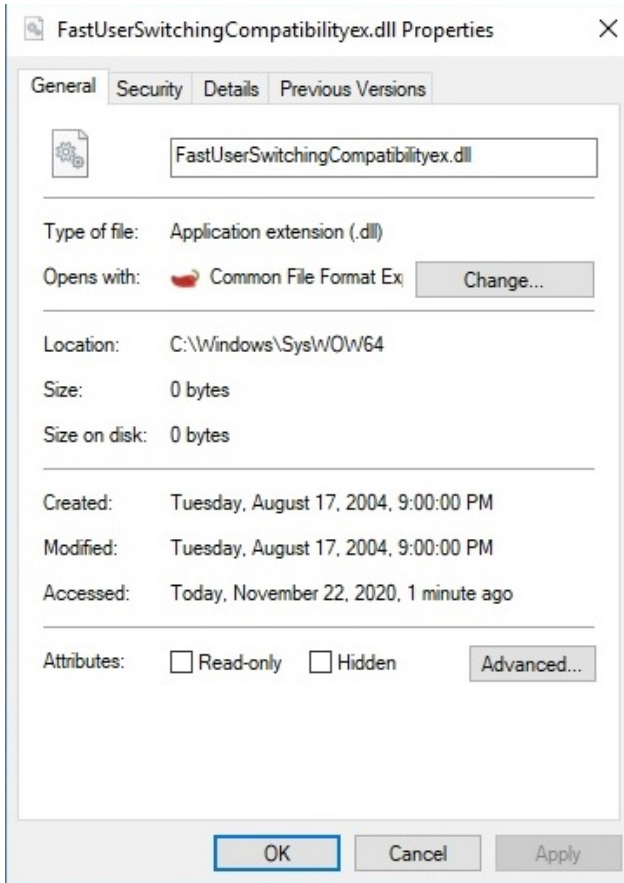


Figure 22

Now the DLL file created earlier is filled with malicious code using WriteFile API. Even if the path of the file looks legitimate (running from “C:\Windows\SysWOW64” directory), it’s just impersonating a legitimate service:

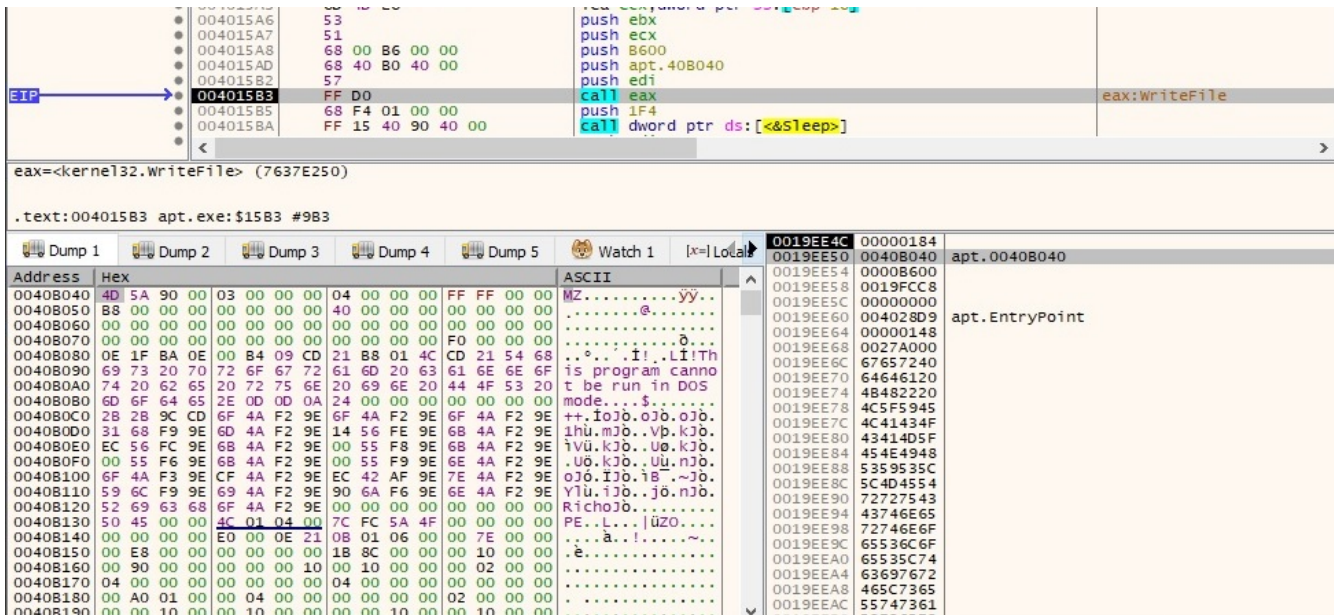


Figure 23

It’s worth mentioning that registering a DLL file as a service is a persistence mechanism. The newly created service is started using StartServiceA API and the flow of execution is passed to the DLL export function ServiceMain:

00401689	57	push edi	
0040168A	57	push edi	
0040168B	53	push ebx	
0040168C	FF D0	call eax	eax: StartServiceA
0040168E	53	push ebx	
0040168F	FF D6	call esi	esi: CloseServiceHandle
00401691	FF 75 FC	push dword ptr ss:[ebp-4]	

eax=<advapi32.StartServiceA> (77624600)

.text:0040168C apt.exe:\$168C #A8C

0019FC80	02765C00	
0019FC84	00000000	
0019FC88	00000000	

Figure 24

Section II

DLL file

SHA256: ED6AD64DAD85FE11F3CC786C8DE1F5B239115B94E30420860F02E820FFC53924

One of the first steps the malware is performing is to invoke GetProcessWindowStation API which returns a handle to the current window station and then it uses OpenWindowStationA API to open the interactive window station ("Winsta0"). The process assigns the specified window station ("Winsta0") which is the only interactive window station (the service is supposed to be interactive) to the calling process using the SetProcessWindowStation function:

10001C65	68 7F 03 00 00	push 37F	
10001C6A	56	push esi	
10001C6B	68 D8 B2 00 10	push fastuserswitchingcompatibilityex.100082D8	100082D8: "winsta0"
10001C70	8B F8	mov edi, eax	
10001C72	FF 15 D8 91 00 10	call dword ptr ds:[<&OpenWindowStationA>]	
10001C78	3B C6	cmp eax, esi	

dword ptr [100091D8 <fastuserswitchingcompatibilityex.&OpenWindowStationA>]=<user32.OpenWindowStationA>

.text:10001C72 fastuserswitchingcompatibilityex.dll:\$1C72 #1072

00B7F230	100082D8	"winsta0"
00B7F234	00000000	
00B7F238	0000037F	

Figure 25

As in the first example the process creates a different mutex called "NetTravler Is Running!". If it exists it will exit without reinfecting the machine:

10001C8A	68 C0 B2 00 10	push fastuserswitchingcompatibilityex.100082C0	100082C0: "NetTravler Is Running!"
10001C8F	6A 01	push 1	
10001C91	56	push esi	
10001C92	FF 15 88 90 00 10	call dword ptr ds:[<&CreateMutexA>]	
10001C98	A3 54 70 01 10	mov dword ptr ds:[10017084], eax	
10001C9D	FF 15 84 90 00 10	call dword ptr ds:[<&GetLastError>]	
10001CA3	3D B7 00 00 00	cmp eax, B7	

dword ptr [10009088 <fastuserswitchingcompatibilityex.&CreateMutexA>]=<kernel32.CreateMutexA>

.text:10001C92 fastuserswitchingcompatibilityex.dll:\$1C92 #1092

00B7F230	00000000	
00B7F234	00000001	
00B7F238	100082C0	"NetTravler Is Running!"

Figure 26

Now it retrieves a few elements from the configuration file config_t.dat created by the first process: WebPage, DownCmdTime, UploadRate, AutoCheck, UP and CheckedSuccess (it doesn't exist at this time, so the function returns 0). All of the values are extracted using GetPrivateProfileString and GetPrivateProfileInt APIs:

10004EDE	50	push eax	eax: "C:\\WINDOWS\\system\\config_t.
10004EDF	68 00 01 00 00	push 100	
10004EE4	68 70 68 01 10	push fastuserswitchingcompatibilityex.10016870	
10004EE9	68 98 72 01 10	push fastuserswitchingcompatibilityex.10017298	
10004EEE	68 98 B8 00 10	push fastuserswitchingcompatibilityex.10008898	10008898: "WebPage"
10004EF3	53	push ebx	ebx: "Option"
10004EF4	FF 15 A4 90 00 10	call dword ptr ds:[<&GetPrivateProfileStringA>]	
10004EF4	88 3D A8 90 00 10	mov edi, dword ptr ds:[<&GetPrivateProfileIntA>]	
10004F00	8D 44 24 10	lea eax, dword ptr ss:[esp+10]	

dword ptr [100090A4 <fastuserswitchingcompatibilityex.&GetPrivateProfileStringA>]=<kernel32.GetPrivateProfileStringA>

.text:10004EF4 fastuserswitchingcompatibilityex.dll:\$4EF4 #42F4

00B7F008	100088A0	"Option"
00B7F00C	10008898	"WebPage"
00B7F010	10017298	fastuserswitchingcompatibilityex.10017298
00B7F014	10016870	fastuserswitchingcompatibilityex.10016870
00B7F018	00000100	
00B7F01C	00B7F030	"C:\\WINDOWS\\system\\config_t.dat"

Address	Hex	ASCII
00B7F030	43 3A 5C 57 49 4E 44 4F 57 53 5C 73 79 73 74 65	C:\\WINDOWS\\system
00B7F040	6D 5C 63 6F 6E 66 69 67 5F 74 2E 64 61 74 00 00	m\\config_t.dat..

Figure 27

```

0010004F04 50          push eax
0010004F05 6A 00       push 0
0010004F07 68 8C 88 00 10 push fastuserswitchingcompatibilityex.1000B88C
0010004F0C 53         push ebx
EIP → 0010004F0F FF D7      call edi
0010004F11 A3 6C 68 01 10 mov dword ptr ds:[1001686C],eax

```

edi=<kernel32.GetPrivateProfileIntA> (76358FD0)

.text:10004F0D fastuserswitchingcompatibilityex.dll:\$F0D #430D

00B7F010	1000B8A0	"Option"
00B7F014	1000B88C	"DownCmdTime"
00B7F018	00000000	
00B7F01C	00B7F030	"C:\\WINDOWS\\system\\config_t.dat"

Figure 28

```

0010004F18 50          push eax
0010004F19 6A 00       push 0
0010004F1B 68 80 88 00 10 push fastuserswitchingcompatibilityex.1000B880
0010004F20 53         push ebx
EIP → 0010004F21 FF D7      call edi

```

edi=<kernel32.GetPrivateProfileIntA> (76358FD0)

.text:10004F21 fastuserswitchingcompatibilityex.dll:\$F21 #4321

00B7F010	1000B8A0	"Option"
00B7F014	1000B880	"UploadRate"
00B7F018	00000000	
00B7F01C	00B7F030	"C:\\WINDOWS\\system\\config_t.dat"

Figure 29

```

0010004F2C 50          push eax
0010004F2D 6A 00       push 0
0010004F2F 8B 74 88 00 10 mov ebx,fastuserswitchingcompatibilityex.1000B874
0010004F34 68 68 88 00 10 push fastuserswitchingcompatibilityex.1000B868
0010004F39 53         push ebx
0010004F3A C7 05 58 B1 00 10 00 28 00 mov dword ptr ds:[1000B158],2800
EIP → 0010004F44 FF D7      call edi

```

edi=<kernel32.GetPrivateProfileIntA> (76358FD0)

.text:10004F44 fastuserswitchingcompatibilityex.dll:\$F44 #4344

00B7F010	1000B874	"OtherTwo"
00B7F014	1000B868	"AutoCheck"
00B7F018	00000000	
00B7F01C	00B7F030	"C:\\WINDOWS\\system\\config_t.dat"

Figure 30

```

0010004F55 50          push eax
0010004F56 6A 00       push 0
0010004F58 68 58 88 00 10 push fastuserswitchingcompatibilityex.1000B858
0010004F5D 53         push ebx
EIP → 0010004F5E FF D7      call edi
0010004F60 48         dec eax

```

edi=<kernel32.GetPrivateProfileIntA> (76358FD0)

.text:10004F5E fastuserswitchingcompatibilityex.dll:\$F5E #435E

00B7F010	1000B874	"OtherTwo"
00B7F014	1000B858	"CheckedSuccess"
00B7F018	00000000	
00B7F01C	00B7F030	"C:\\WINDOWS\\system\\config_t.dat"

Figure 31

```

00100025C0 50          push eax
00100025C1 6A 00       push 0
00100025C3 68 6C B3 00 10 push fastuserswitchingcompatibilityex.1000B36C
00100025C8 56         push esi
EIP → 00100025C9 FF 15 A8 90 00 10 call dword ptr ds:[<&GetPrivateProfileIntA>]
00100025CF 6A 01       push 1
00100025D1 59         pop ecx

```

dword ptr [100090A8 <fastuserswitchingcompatibilityex.&GetPrivateProfileIntA>]=<kernel32.GetPrivateProfileIntA>

.text:100025C9 fastuserswitchingcompatibilityex.dll:\$25C9 #19C9

00B7EFD4	1000B3D0	"Other"
00B7EFD8	1000B36C	"up"
00B7EFD8	00000000	
00B7EFED	00B7F030	"C:\\WINDOWS\\system\\config_t.dat"

Figure 32

Because we're running the DLL using an executable used by x32dbg to debug the DLL files, the process name is similar to "DLLLoader32_58D1.exe" (in our case). The malicious process creates a .log file which has the same name as the executable ("DLLLoader32_58D1.log"):

```

0010005C8E 56          push esi
0010005C8F 68 80 00 00 00 push 80
0010005C94 6A 04       push 4
0010005C96 56         push esi
0010005C97 6A 03       push 3
0010005C99 68 00 00 00 40 push 40000000
0010005C9E 68 88 70 01 10 push fastuserswitchingcompatibilityex.10017088
EIP → 0010005CA3 FF 15 4C 90 00 10 call dword ptr ds:[<&CreateFileA>]

```

dword ptr [1000904C <fastuserswitchingcompatibilityex.&CreateFileA>]=<kernel32.CreateFileA>

.text:10005CA3 fastuserswitchingcompatibilityex.dll:\$5CA3 #50A3

00B7EE8C	10017088	"C:\\Users\\[redacted]\\Desktop\\DLLLoader32_58D1.log"
00B7EE90	40000000	
00B7EE94	00000003	
00B7EE98	00000000	
00B7EE9C	00000004	
00B7EEA0	00000080	
00B7EEA4	00000000	

Figure 33

The file enumerates the directories from “C:\Program File (x86)” and the output is copied to the newly created file:

```

DLLLoader32_58D1.log
1 |10-Strike Network File Search Pro\
2 |Adobe\
3 |AllowBlock\
4 |Any Sound Recorder\
5 |Application Verifier\
6 |AudioCoder\
7 |Common Files\
8 |desktop.ini
9 |Detect It Easy\
10 |Dev-Cpp\
11 |DeviceViewer\
12 |DICOMviewer demo\
13 |Disk Pulse Enterprise\
14 |DiskBoss\
15 |DiskBoss Enterprise\
16 |docPrint Pro v8.0\
17 |Dup Scout Enterprise\
18 |Easy Video to iPod Converter\
19 |Easy Video to PSP Converter\
20 |Easy WMV ASF ASX to DVD Burner\
21 |Entity Framework Tools\
22 |ExeinfoPe\
23 |Faleemi\
24 |Fiddler2\
25 |Flash Slideshow Maker Professional\
26 |Free MP3 CD Ripper\
27 |Frigate\
28 |Google\
29 |Graphviz2.38\
30 |HxD\
31 |Immunity Inc\
32 |Imports Fixer\
33 |InstallShield Installation Information\
34 |Internet Explorer\

```

Figure 34

RegOpenKeyExA API is used to open “HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders” registry key and the “History” value is extracted from it using RegQueryValueEx. The content of “History” value is “C:\Users\<Username>\AppData\Local\Microsoft\Windows\History”:

Figure 35

The malware is looking for a file called “C:\Users\<Username>\AppData\Local\Microsoft\Windows\History\History.IE5\index.dat” which contains Internet browsing history activity, including Internet based searches and opened files:

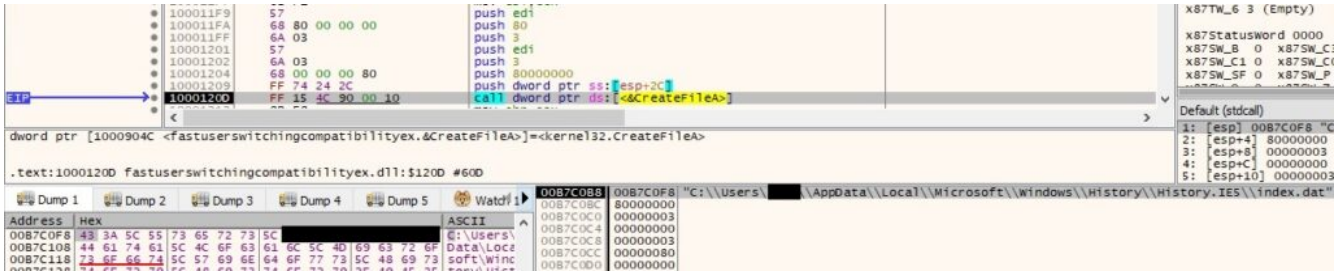


Figure 36

The process extracts “Version” value from “HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Internet Explorer” using RegQueryValueEx function:

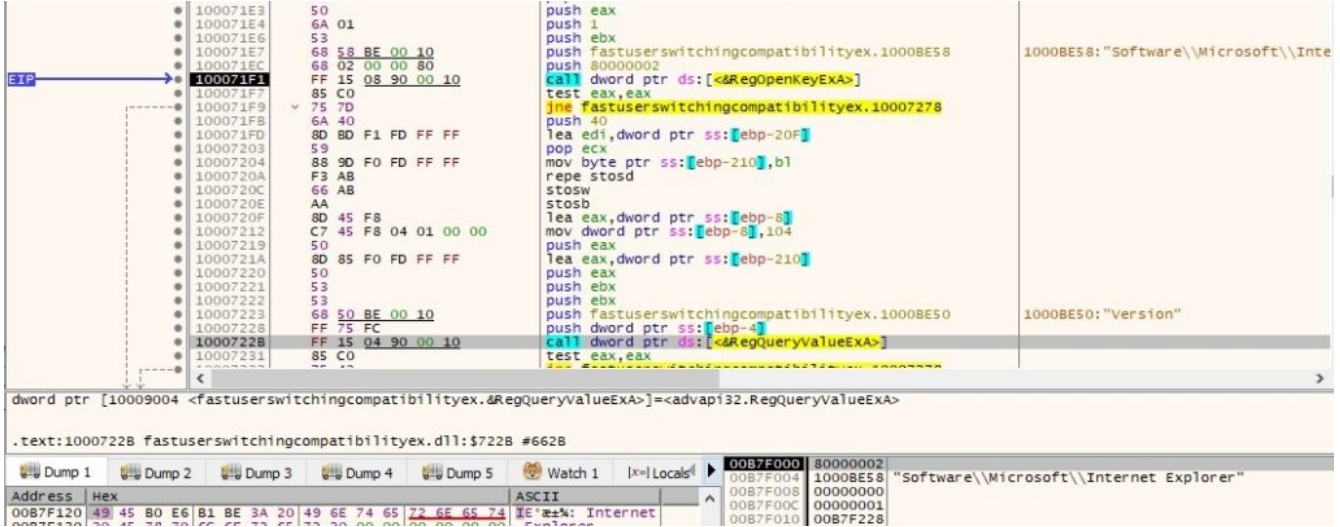


Figure 37

Window 10’s Internet Explorer is Build 916299, Version 9.11.16299.0 as shown in the figure below:

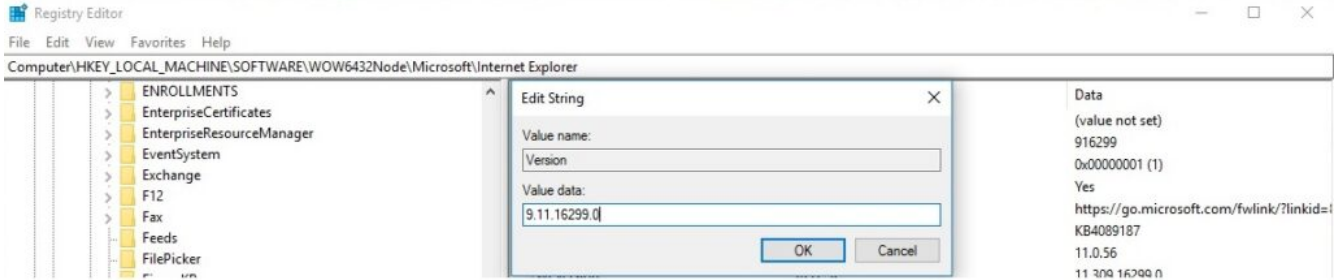


Figure 38

The following information is appended to the .log file: IE History is empty because that file is missing on Windows 10 and the IE version (note that “version” word is written in Chinese language “版本”):

```

74
75 ////////////////////////////////////////////////////////////////////
76
77
78 IE History:
79
80
81 ////////////////////////////////////////////////////////////////////
82
83 IE版本: Internet Explorer 9.11.16299.0
84

```

Figure 39

GetVersionExA function is utilized to find the current operating system. The recognized versions are: Microsoft Windows 7, Microsoft Windows Vista, Microsoft Windows 2003, Microsoft Windows 2000, Microsoft Windows XP and Microsoft Windows NT:

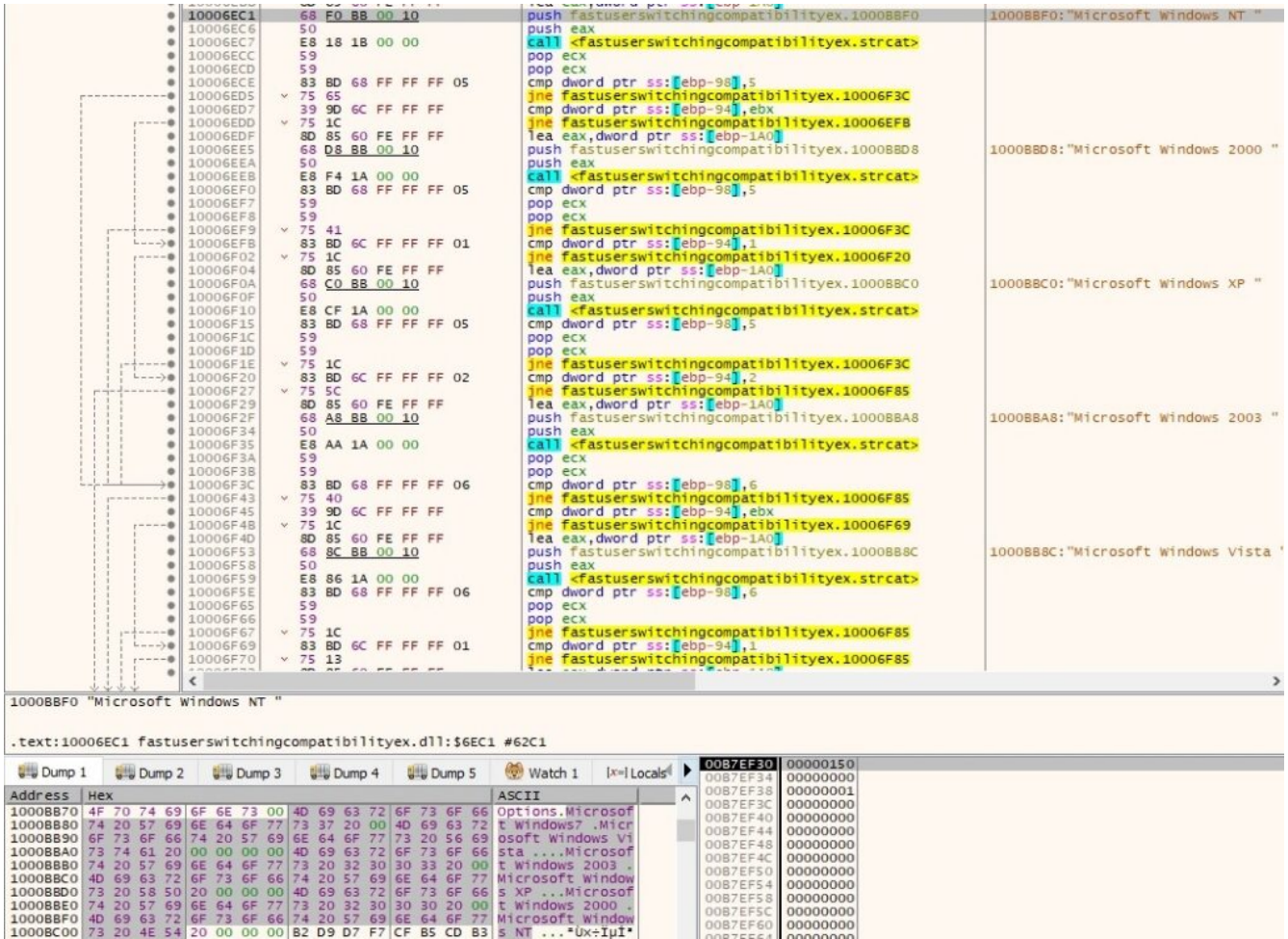


Figure 40

It also extracts the “ProductType” value from “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ProductOptions” registry key. On our system the value is equal to “WinNT”:

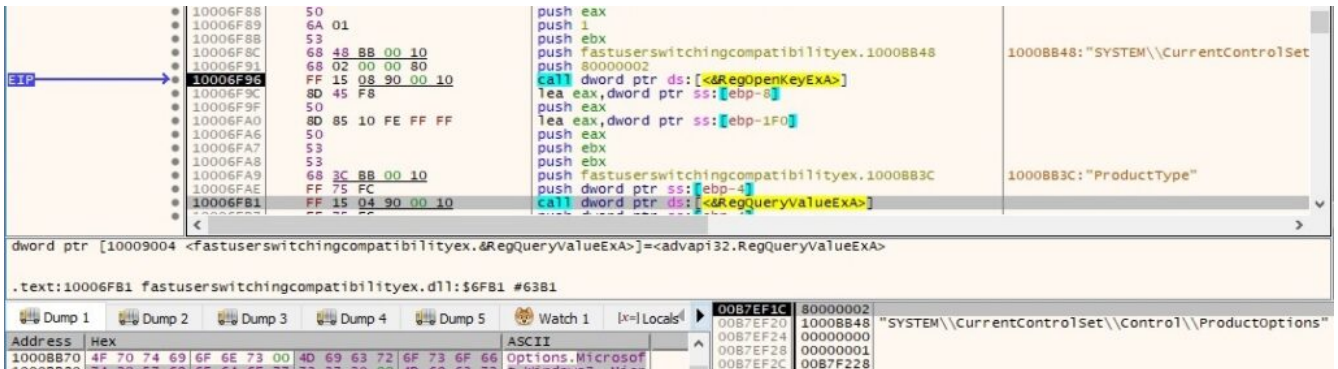


Figure 41

The following string is appended to the .log file: “操作系统版本” translates to “Operating system version”:

```

84
85 ////////////////////////////////////////////////////
86
87 操作系统版本: Professional, (Build 9200)
88

```

Figure 42

The user agent used in the network communications is always set to “Mozilla/4.0 (compatible; MSIE 6.0)”. There is also an Accept request HTTP header as shown below:

Figure 43

The process tries to connect to http://www.microsoft.com/info/privacy_security.htm (this URL used to be available in the past) in order to verify if there is an internet connection. The HTTP request is shown in figure 44:

Figure 44

If the connection is successful the following strings will be added at the end of the .log file:

```

89 method 1:
90
91 User: ██████████
92
93 <Response from the server>

```

Figure 45

Furthermore UP (use proxy indicator) is set to 0 and it adds a value called CheckedSuccess (set to 1) to config_t.dat using WritePrivateProfileStringA API:

```

0007382 50          push     eax
0007383 68 30 00 10 push     fastuserswitchingcompatibilityex.1000B330
0007388 68 30 00 10 push     fastuserswitchingcompatibilityex.1000B36C
0007389 68 30 00 10 push     fastuserswitchingcompatibilityex.1000B3D0
EIP → 0007392 FF 15 A0 90 00 10 call     dword ptr ds:[<&WritePrivateProfileStringA>]
; eax:"C:\\WINDOWS\\system\\co
; 1000B36C:"UP"
; 1000B3D0:"Other"

dword ptr [100090A0 <fastuserswitchingcompatibilityex.&WritePrivateProfileStringA>]=<kernel32.WritePrivateProfileStringA>
.text:10007392 fastuserswitchingcompatibilityex.dll:$7392 #6792

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x] Locals 00B7F014 1000B3D0 "Other"
00B7F018 1000B36C "UP"
Address Hex ASCII 00B7F01C 1000B330 fastuserswitchingcompatibilityex.1000B330
1000B330 30 00 00 00 31 30 00 00 50 46 00 00 50 57 00 00 |...10..FF..Pw..
00073D7 50          push     eax
00073D8 68 50 83 00 10 push     fastuserswitchingcompatibilityex.1000B350
00073DD 68 50 83 00 10 push     fastuserswitchingcompatibilityex.1000B358
00073E2 68 74 88 00 10 push     fastuserswitchingcompatibilityex.1000B874
EIP → 00073E7 FF 15 A0 90 00 10 call     dword ptr ds:[<&WritePrivateProfileStringA>]
; eax:"C:\\WINDOWS\\system\\co
; 1000B858:"CheckedSuccess"
; 1000B874:"OtherTwo"

dword ptr [100090A0 <fastuserswitchingcompatibilityex.&WritePrivateProfileStringA>]=<kernel32.WritePrivateProfileStringA>
.text:100073E7 fastuserswitchingcompatibilityex.dll:$73E7 #67E7

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x] Locals 00B7F01C 1000B874 "OtherTwo"
00B7F020 1000B858 "CheckedSuccess"
Address Hex ASCII 00B7F024 1000B350 fastuserswitchingcompatibilityex.1000B350
1000B350 31 00 00 00 25 73 5C 73 17 9 73 74 65 6D 5C 63 6F |...%s\\system\\co
00B7F028 00B7F02C "C:\\WINDOWS\\system\\config_t.dat"

```

Figure 46

Now, if the connection was unsuccessful, an “Method1 Fail!!!!” message is written to DLLLoader32_58D1.log. Process32First and Process32Next functions are used to find “EXPLORER.exe” process and then the process tries to open it using OpenProcess API:

```

000155E E8 75 74 00 00 call     <fastuserswitchingcompatibilityex.Process32First>
0001563 8B 3D 40 90 00 10 mov     edi,dword ptr ds:[<&CloseHandle>]
0001569 85 C0          test    eax,eax
000156B 74 63          je     fastuserswitchingcompatibilityex.100015D0
000156D 75 0C          jne   fastuserswitchingcompatibilityex.100015D0
0001570 8B 85 94 91 00 10 mov     esi,dword ptr ds:[<&strupr>]
0001576 FF D6          call   esi
0001578 59          pop    ecx
0001579 50          push   eax
000157A 8D 85 F8 FE FF FF lea    eax,dword ptr ss:[ebp-108]
0001580 50          push   eax
0001581 FF D6          call   esi
0001583 59          pop    ecx
0001584 50          push   eax
0001585 EC 8C 74 00 00 call     <fastuserswitchingcompatibilityex.strcmp>
000158A 59          pop    ecx
000158B 85 C0          test   eax,eax
000158D 59          pop    ecx
000158E 74 16          je     fastuserswitchingcompatibilityex.100015A6
0001590 8D 85 D4 FE FF FF lea    eax,dword ptr ss:[ebp-12C]
0001596 50          push   eax
0001597 53          push   ebx
0001598 E8 35 74 00 00 call     <fastuserswitchingcompatibilityex.Process32Next>
000159D 85 C0          test   eax,eax
000159F 74 2F          je     fastuserswitchingcompatibilityex.100015D0
00015A1 FF 75 0C          push  dword ptr ss:[ebp+C]
00015A4 EB D0          jmp    fastuserswitchingcompatibilityex.10001576
00015A6 FF B5 DC FE FF FF push  dword ptr ss:[ebp-124]
00015AC 6A 00          push  0
00015AE 68 00 04 00 00 push  400
EIP → 00015B3 FF 15 50 90 00 10 call     dword ptr ds:[<&OpenProcess>]
; edi:CloseHandle
; [ebp+C]:"EXPLORER.EXE"
; esi:strupr
; ecx:"EXPLORER.EXE"
; esi:strupr
; ecx:"EXPLORER.EXE"
; ecx:"EXPLORER.EXE"
; ecx:"EXPLORER.EXE"
; [ebp+C]:"EXPLORER.EXE"

dword ptr [10009050 <fastuserswitchingcompatibilityex.&OpenProcess>]=<kernel32.OpenProcess>
.text:100015B3 fastuserswitchingcompatibilityex.dll:$15B3 #983

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x] Locals StrUct 004FCDC0 00000400
004FCDC4 00000000
Address Hex ASCII 004FCDC8 000000E0

```

Figure 47

Basically the attacker’s purpose is to steal “explorer.exe” process’ token by calling OpenProcessToken in order to open the access token associated with “explorer.exe” and then it uses ImpersonateLoggedOnUser function to impersonate the security context of a user. The function calls are displayed in figure 48 and figure 49, respectively.

```

00015B9 FF 75 08          push  dword ptr ss:[ebp+8]
00015BC 8B F0          mov     esi,eax
00015BE 68 FF 01 0F 00 push  F01FF
00015C3 56          push   esi
EIP → 00015C4 FF 15 2C 90 00 10 call     dword ptr ds:[<&OpenProcessToken>]
; esi:Process
; ecx:Process

dword ptr [1000902C <fastuserswitchingcompatibilityex.&OpenProcessToken>]=<advapi32.OpenProcessToken>
.text:100015C4 fastuserswitchingcompatibilityex.dll:$15C4 #9C4

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x] Locals StrUct 004FCDC0 000002FC
004FCDC4 000F01FF
Address Hex ASCII 004FCDC8 004FF41C

```

Figure 48

```

0005F51 FF 75 E0          push  dword ptr ss:[ebp-20]
EIP → 0005F54 FF 15 10 90 00 10 call     dword ptr ds:[<&ImpersonateLoggedOnUser>]
; esi:Process
; ecx:Process

dword ptr [10009010 <fastuserswitchingcompatibilityex.&ImpersonateLoggedOnUser>]=<advapi32.ImpersonateLoggedOnUser>
.text:10005F54 fastuserswitchingcompatibilityex.dll:$5F54 #5354

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x] Locals StrUct 004FCF10 00000300

```

Figure 49

The process is using RegOpenKeyExA to open “HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Internet Settings” registry key and then it extracts “ProxyEnable” value to see if the computer uses a proxy server:

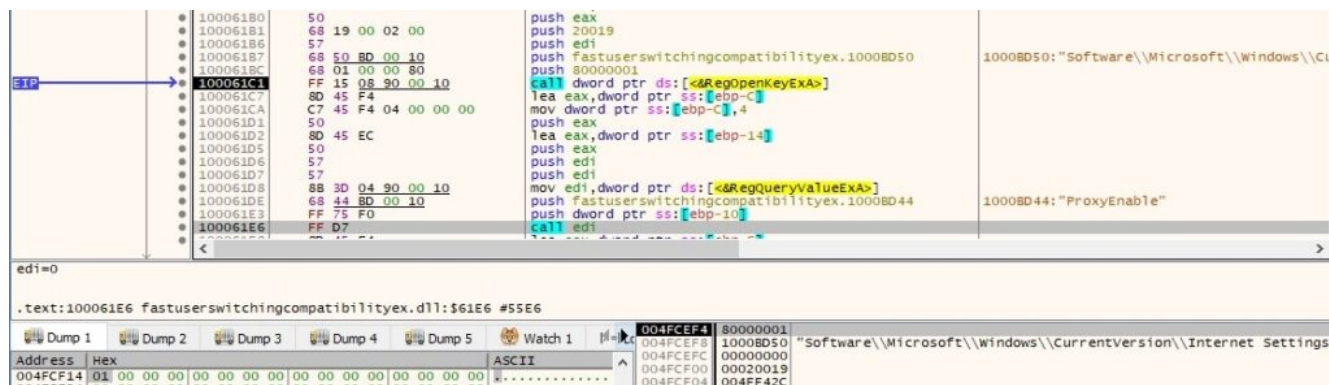


Figure 50

Also same function is used to get the “ProxyServer” (hostnames/IPs of the proxy server on the network) and “ProxyOverride” (hostnames/IPs that bypass the proxy server) values from the same registry key. The extraction of “ProxyServer” value is shown below:

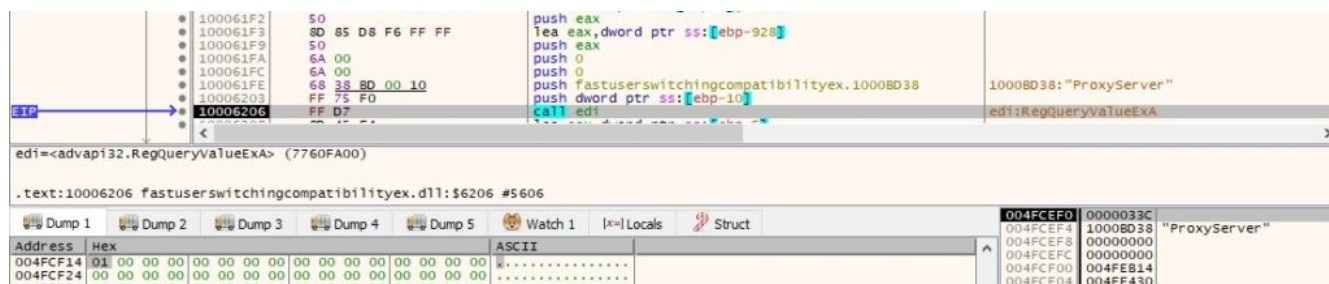


Figure 51

As in the first method, the attacker verifies if he’s able to connect to the same URL using the proxy settings he found in the registry. If the connection is successful it will append the content of that page to the .log file together with some new parameters:

```

95 ////////////////////////////////////////////////////////////////////
96
97
98 method 3:
99
100 User: ██████████
101
102 ProxyIP:
103 ProxyBypass:
104 User:
105 Pass:
106 <Response from the server>

```

Figure 52

Also, because the method works, the malicious process modifies the config_t.dat file by setting UP=1, PF=10 and then PS (proxy server), PP (proxy port), PU (proxy user), PW (proxy password) are set according to the settings found. If the connection fails, the message “Method3 Fail!!!!” is appended to the .log file. Method4 is pretty similar to Method3 presented above and will not be explained in details. One of the differences is that the “Method4 Fail!!!!” message is appended to the .log file if the network connection isn’t successful.

If all methods fail, the infection will stop and the following operations are performed (self-deleting malware): “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility\Enum”, “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility\Parameters”,

“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility\Security” and “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility” registry keys are deleted using RegDeleteKeyA function. The following files are deleted as well: “C:\WINDOWS\system32\enumfs.ini”, “C:\WINDOWS\system32\dnlist.ini”, “C:\WINDOWS\system32\udidx.ini”, “C:\WINDOWS\system32\uenumfs.ini” and “C:\WINDOWS\system32\stat_t.ini” (some of them don’t exist at this time).

If one of the methods enumerated above works, the malicious process sleeps for 60 seconds and then creates another thread that we’ll call Thread1 , sleeps another 10 seconds, and creates Thread2. The main thread will enter into an infinite loop until the variable found at 0x100163E8 (absolute address) is set to 3:

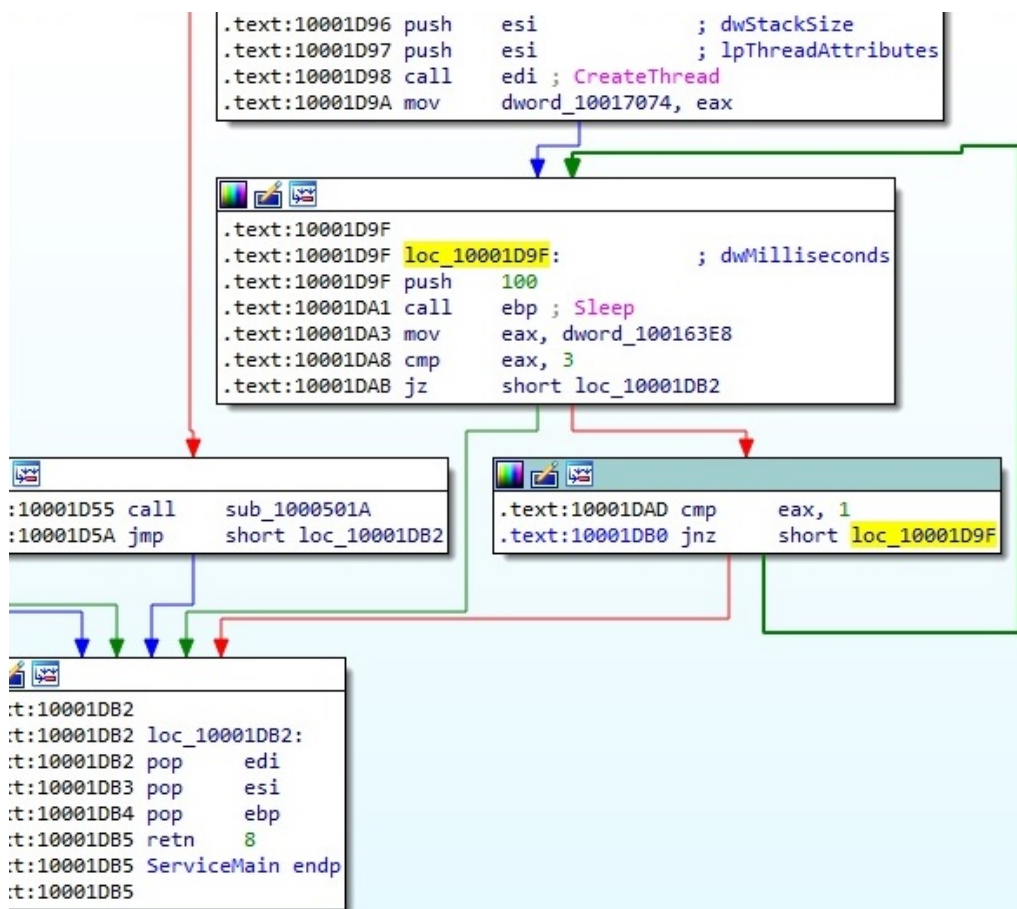


Figure 53

Thread1 activity

Firstly the thread retrieves the volume serial number (“A2C9-AD2F”) associated with “C:\” directory using GetVolumeInformationA function. This number will be used as a host id in the communication with the C2 server as we will see later on. Also it uses GetComputerNameA API to find the NETBIOS name of the computer, GetUserNameA API to find the username associated with the current thread, gethostname API to retrieve the host name for the computer and gethostbyname/inet_ntoa functions to print the IP address of the computer:

```

1000312C FF 75 08      push dword ptr ss:[ebp+8]
1000312F FF 15 34 92 00 10  call dword ptr ds:[<&gethostname>]
10003138 85 C0        test eax, eax
1000313A 75 47        jne fastuserswitchingcompatibilityex.10003183
1000313C FF 75 08      push dword ptr ss:[ebp+8]
1000313E FF 15 38 92 00 10  call dword ptr ds:[<&gethostbyname>]
10003145 8B F8        mov edi, eax
10003147 3B FE        cmp edi, esi
10003149 74 38        je fastuserswitchingcompatibilityex.10003183
1000314B 8B 47 0C     mov eax, dword ptr ds:[edi+C]
1000314E 8B 04 06     mov eax, dword ptr ds:[esi+eax]
10003151 85 C0        test eax, eax
10003153 74 2E        je fastuserswitchingcompatibilityex.10003183
10003155 FF 30        push dword ptr ds:[eax]
10003157 FF 15 3C 92 00 10  call dword ptr ds:[<&inet_ntoa>]
1000315D 50          push eax
1000315E 68 A4 B1 00 10  push fastuserswitchingcompatibilityex.1000B1A4
10003163 FF 75 10     push dword ptr ss:[ebp+10]
10003166 FF 15 90 91 00 10  call dword ptr ds:[<&sprintf>]
1000316C FF 75 10     push dword ptr ss:[ebp+10]
1000316F E8 88 58 00 00  call <fastuserswitchingcompatibilityex.strLen>

```

edi:CreateThread
edi:CreateThread

1000B1A4: "%s"

dwornd ptr [1000923C <fastuserswitchingcompatibilityex.&inet_ntoa>]=<ws2_32.inet_ntoa>

.text:10003157 fastuserswitchingcompatibilityex.dll:\$3157 #2557

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals 00BFF198 10016504 fastuserswitchingcompatibilityex.10016504
00BFF19C 00000104

Address	Hex	ASCII
100164F4	31 39 32 2E 31 36 38 2E 31 36 34 2E 31 32 38 00	192.168.164.128
10016504	44 45 53 4B 54 4F 50 2D 32	DESKTOP-2

Figure 54
One more time the "ProductType" value from "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ProductOptions" registry key is retrieved as shown in figure 55:

```

10005450 50          push eax
10005451 6A 01        push 1
10005453 53          push ebx
10005454 68 48 BB 00 10  push fastuserswitchingcompatibilityex.1000BB48
10005459 68 02 00 00 80  push 80000002
1000545E FF 15 08 90 00 10  call dword ptr ds:[<&RegOpenKeyExA>]
10005464 8D 45 DC     lea eax, dword ptr ss:[ebp-24]
10005467 50          push eax
10005468 8D 85 50 FF FF  lea eax, dword ptr ss:[ebp-180]
1000546E 50          push eax
1000546F 53          push ebx
10005470 53          push ebx
10005471 68 3C BB 00 10  push fastuserswitchingcompatibilityex.1000BB3C
10005476 FF 75 FC     push dword ptr ss:[ebp-4]
10005479 FF 15 04 90 00 10  call dword ptr ds:[<&RegQueryValueExA>]

```

eax: "bq"

1000BB48: "SYSTEM\\CurrentControlSet\\Control\\ProductOptions"

1000BB3C: "ProductType"

dwornd ptr [10009008 <fastuserswitchingcompatibilityex.&RegOpenKeyExA>]=<advapi32.RegOpenKeyExA>

.text:1000545E fastuserswitchingcompatibilityex.dll:\$545E #485E

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct 00BFE12C 80000002 "SYSTEM\\CurrentControlSet\\Control\\ProductOptions"
00BFE130 1000BB48
00BFE134 00000000
00BFE138 00000001
00BFE13C 00BFF344 "bq"

Address	Hex	ASCII
00BFF26C	94 00 00 00 06 00 00 00 02 00 00 00 F0 23 00 00Df..

Figure 55
The malicious process enumerates the available disks drives and it's interested in type 3 drives (DRIVE_FIXED) as shown in the screenshot below:

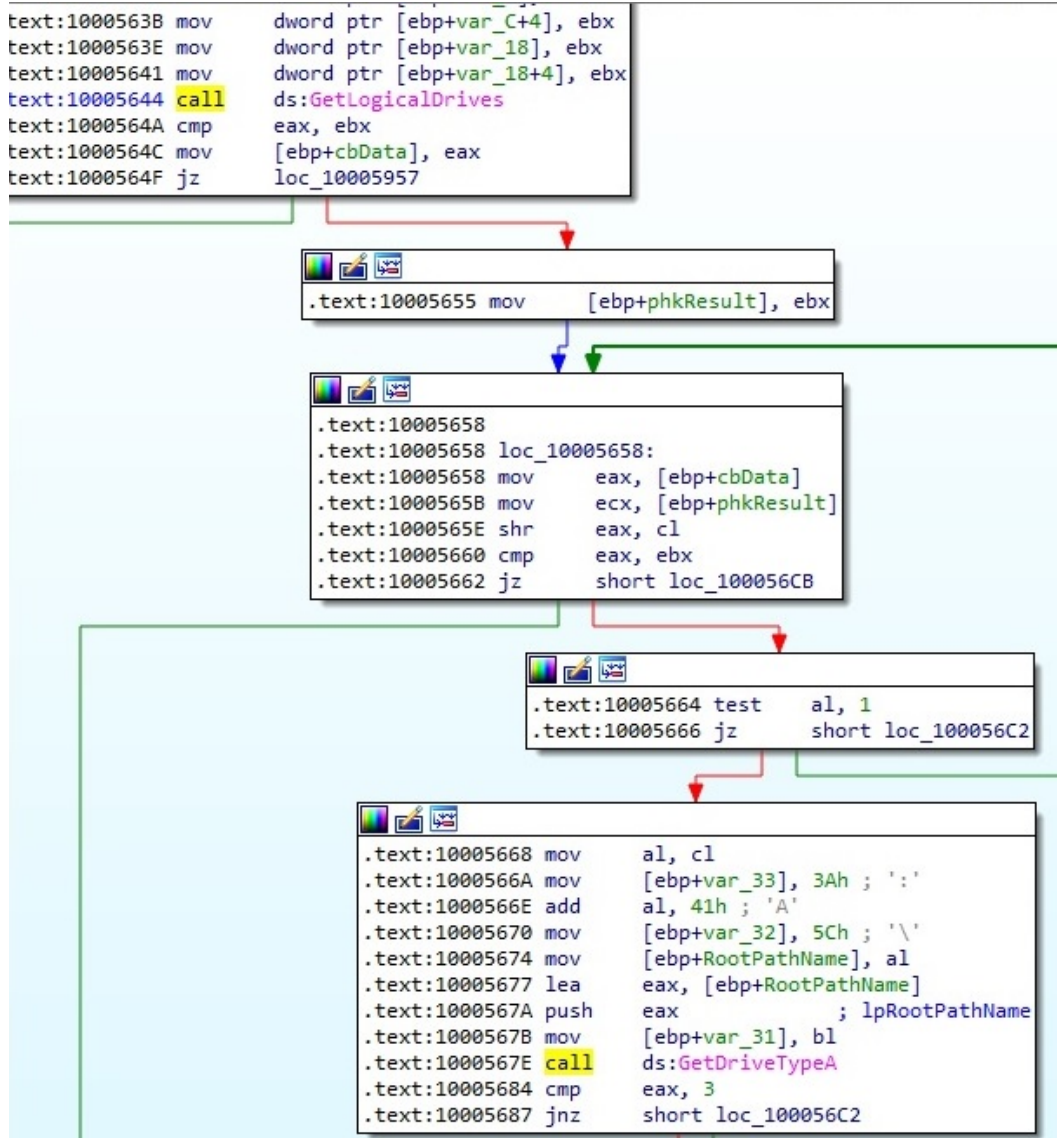


Figure 56

RegOpenKeyExA API is utilized to open “HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\0” registry key and then RegQueryValueEx is used to retrieve “VendorIdentifier”, “Identifier” and “~MHz” values:

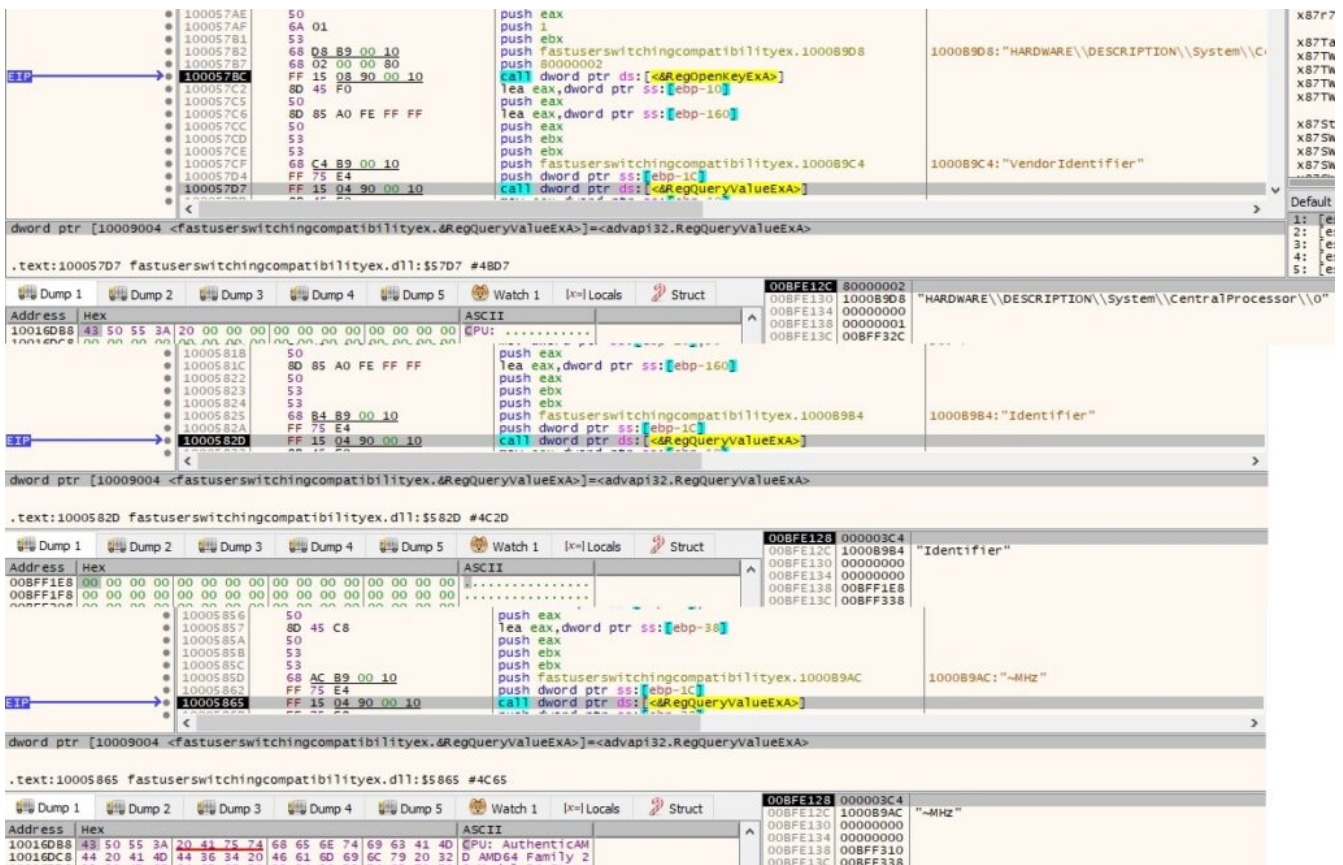


Figure 57

The process uses GlobalMemoryStatus function to get information about system's usage of physical and virtual memory. All the information extracted so far will be stored in a new file called "C:\Windows\SysWOW64\system_t.dll" in order to exfiltrate it. All translations from chinese to english are provided to better understand the content of the file: "计算机信息" translates to "computer information", "计算机" translates to "computer", "用户名" translates to username, "ip地址" translates to "Ip address", "操作系统" translates to "operating system", "磁盘空间" translates to "disk space", "总磁盘空间为" translates to "The total disk space is", "剩余磁盘空间为" translates to "The remaining disk space is", "占" translates to "take up", "物理内存" translates to "physical memory", "总物理内存" translates to "Total physical memory" and "可用内存" translates to "Available memory":



Figure 58

A list of processes is retrieved using Process32First and Process32Next APIs as shown below:

0000A555	50	push eax	
0000A556	FF 75 FC	push dword ptr ss:[ebp-4]	
0000A55E	E8 7A 2F 00 00	call <fastuserswitchingcompatibilityex.Process32First>	
0000A560	85 C0	test eax, eax	
0000A561	0F 84 88 00 00 00	je fastuserswitchingcompatibilityex.1000A5EE	
0000A566	68 04 01 00 00	push 104	
0000A56B	8D 85 F4 FE FF FF	lea eax, dword ptr ss:[ebp-10C]	
0000A711	6A 00	push 0	
0000A713	50	push eax	
0000A714	E8 65 2F 00 00	call <fastuserswitchingcompatibilityex.memset>	
0000A719	83 C4 0C	add esp, c	
0000A7C	8D 85 F0 FD FF FF	lea eax, dword ptr ss:[ebp-210]	
0000A82	50	push eax	
0000A83	8D 85 F4 FE FF FF	lea eax, dword ptr ss:[ebp-10C]	
0000A89	FF 85 D4 FD FF FF	push dword ptr ss:[ebp-22C]	
0000A8F	68 4C BC 00 10	push fastuserswitchingcompatibilityex.1000BC4C	1000BC4C: "84d %s\r\n"
0000A94	50	push eax	
0000A95	FF D6	call esi	esi:sprintf
0000A97	8D 85 C8 FC FF FF	lea eax, dword ptr ss:[ebp-338]	1000BC58: "ab"
0000A9D	68 58 BC 00 10	push fastuserswitchingcompatibilityex.1000BC58	
0000AA2	50	push eax	
0000AA3	FF 15 70 91 00 10	call dword ptr ds:[<&fopen>]	
0000AA9	83 C4 18	add esp, 18	
0000AAC	89 45 F8	mov dword ptr ss:[ebp-8], eax	
0000AAF	85 C0	test eax, eax	
0000AB1	74 37	je fastuserswitchingcompatibilityex.1000AEA	
0000AB3	8D 85 F4 FE FF FF	lea eax, dword ptr ss:[ebp-10C]	
0000AB9	50	push eax	
0000ABA	E8 3D 2F 00 00	call <fastuserswitchingcompatibilityex.strlen>	
0000ABF	FF 75 F8	push dword ptr ss:[ebp-8]	
0000AC2	50	push eax	
0000AC3	8D 85 F4 FE FF FF	lea eax, dword ptr ss:[ebp-10C]	
0000AC9	6A 01	push 1	
0000ACB	50	push eax	
0000ACC	FF D7	call edi	edi:fwrite
0000ACE	FF 75 F8	push dword ptr ss:[ebp-8]	
0000AD1	FF D3	call ebx	ebx:fclose
0000AD3	83 C4 18	add esp, 18	
0000AD6	8D 85 CC FD FF FF	lea eax, dword ptr ss:[ebp-234]	
0000ADC	50	push eax	
0000ADD	FF 75 FC	push dword ptr ss:[ebp-4]	
0000ADD	E8 ED 2E 00 00	call <fastuserswitchingcompatibilityex.Process32Next>	

Figure 59

After the operation is complete and the malware obtains the list of processes, it will be appended to system_t.dll (“进程列表” translates to “Process list”):

```

11 [进程列表]
12 0 [System Process]
13 4 System
14 500 smss.exe
15 604 csrss.exe
16 672 wininit.exe
17 684 csrss.exe
18 732 winlogon.exe
19 796 services.exe
20 804 lsass.exe
21 884 svchost.exe
22 892 fontdrvhost.exe
23 900 fontdrvhost.exe
24 980 svchost.exe
25 8 dwm.exe
26 576 svchost.exe
27 1044 svchost.exe
28 1060 svchost.exe
29 1076 svchost.exe
30 1200 svchost.exe
31 1292 Memory Compression
32 1444 svchost.exe
33 1452 svchost.exe
34 1544 svchost.exe
35 1552 svchost.exe
36 1636 svchost.exe
37 1712 svchost.exe
38 1776 spoolsv.exe
39 2028 WmiPrvSE.exe
40 2080 armsvc.exe
41 2096 svchost.exe
42 2112 diskpls.exe
43 2136 dupsects.exe
44 2164 IpOverUsbSvc.exe
45 2220 TenorshareWinAdService.exe
46 2244 VGAuthService.exe

```

Figure 60

The next step is to create a pipe using CreatePipe API . This will be used as an inter-process communication mechanism. It will create a new process “ipconfig /all” which displays the full TCP/IP configuration for all adapters and the output will be transmitted back to the original process through pipes:

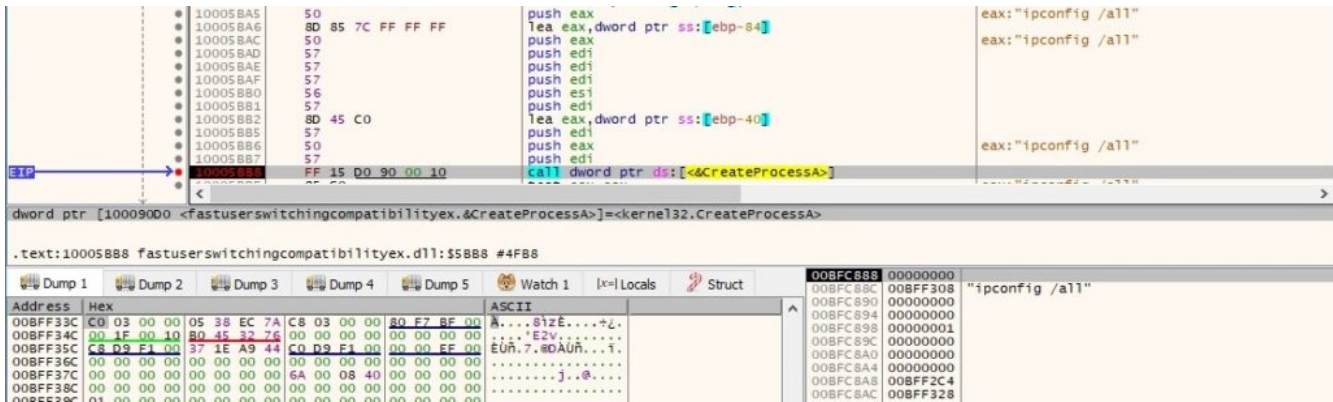


Figure 61

The output of the ipconfig process is saved to system_t.dll as shown in the figure below:

```

87  NUL
88  Windows IP Configuration
89
90  Host Name . . . . . : DESKTOP-2
91  Primary Dns Suffix . . . . . :
92  Node Type . . . . . : Hybrid
93  IP Routing Enabled. . . . . : No
94  WINS Proxy Enabled. . . . . : No
95  DNS Suffix Search List. . . . . : localdomain
96
97  Ethernet adapter Ethernet0:
98
99  Connection-specific DNS Suffix . : localdomain
100 Description . . . . . : Intel(R) 82574L Gigabit Network Connection
101 Physical Address. . . . . : 00-0C-29-25-66-15
102 DHCP Enabled. . . . . : Yes
103 Autoconfiguration Enabled . . . . : Yes
104 Link-local IPv6 Address . . . . . : fe80::bld6:3a45:ca5e:9bee%6 (Preferred)
105 IPv4 Address. . . . . : 192.168.164.128 (Preferred)
106 Subnet Mask . . . . . : 255.255.255.0
107 Lease Obtained. . . . . : Monday, November 9, 2020 5:16:45 AM
108 Lease Expires . . . . . : Monday, November 23, 2020 6:11:18 AM
109 Default Gateway . . . . . :
110 DHCP Server . . . . . : 192.168.164.254
111 DHCPv6 IAID . . . . . : 33557545
112 DHCPv6 Client DUID. . . . . : 00-01-00-01-26-A2-5A-CA-00-0C-29-25-66-15
113 DNS Servers . . . . . : 192.168.164.1
114 NetBIOS over Tcpip. . . . . : Enabled

```

Figure 62

The malware checks the UP value from config_t.dat using GetPrivateProfileInt function. According to the Kaspersky report, the content of system_t.dll file will be compressed using a custom Lempel-Ziv-based algorithm and encoded with a modified Base64 algorithm. The function responsible for this operation and the “modified Base64” alphabet is displayed in figure 63:

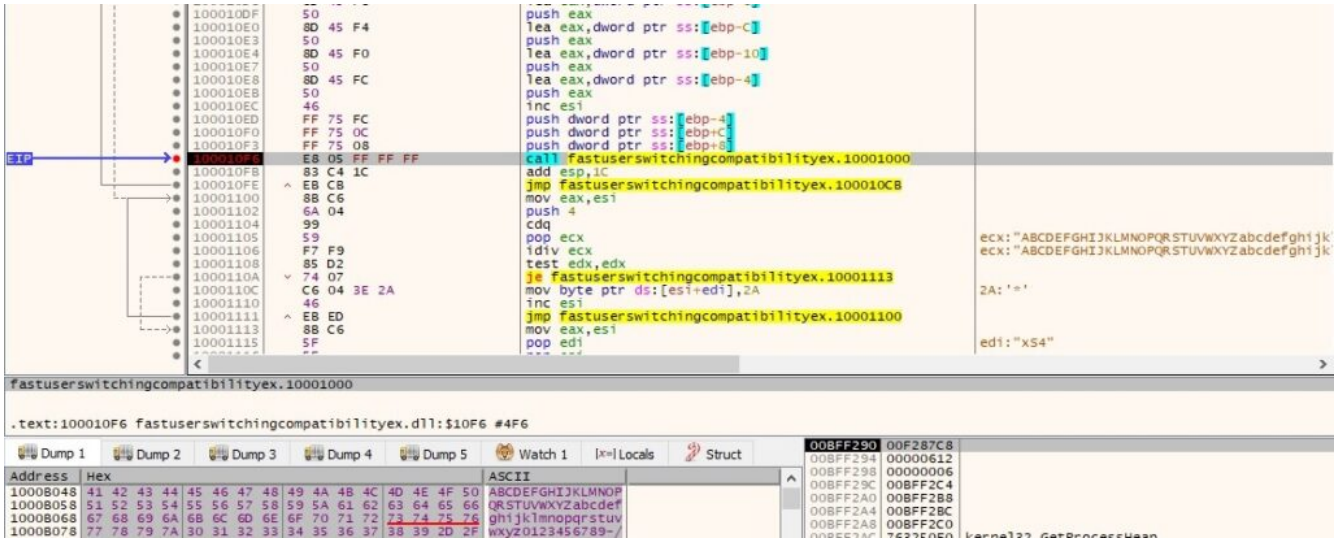


Figure 63

The encoded data is exfiltrated via a GET request to vipmailru[.]com (C2 server). The following parameters are provided in the URL: hostid = the serial number of current disk drive, hostname = hostname, hostip = IP of the machine, filename = "travlerbackinfo-<year>-<month>-<day>-<hour>-<minute>.dll":

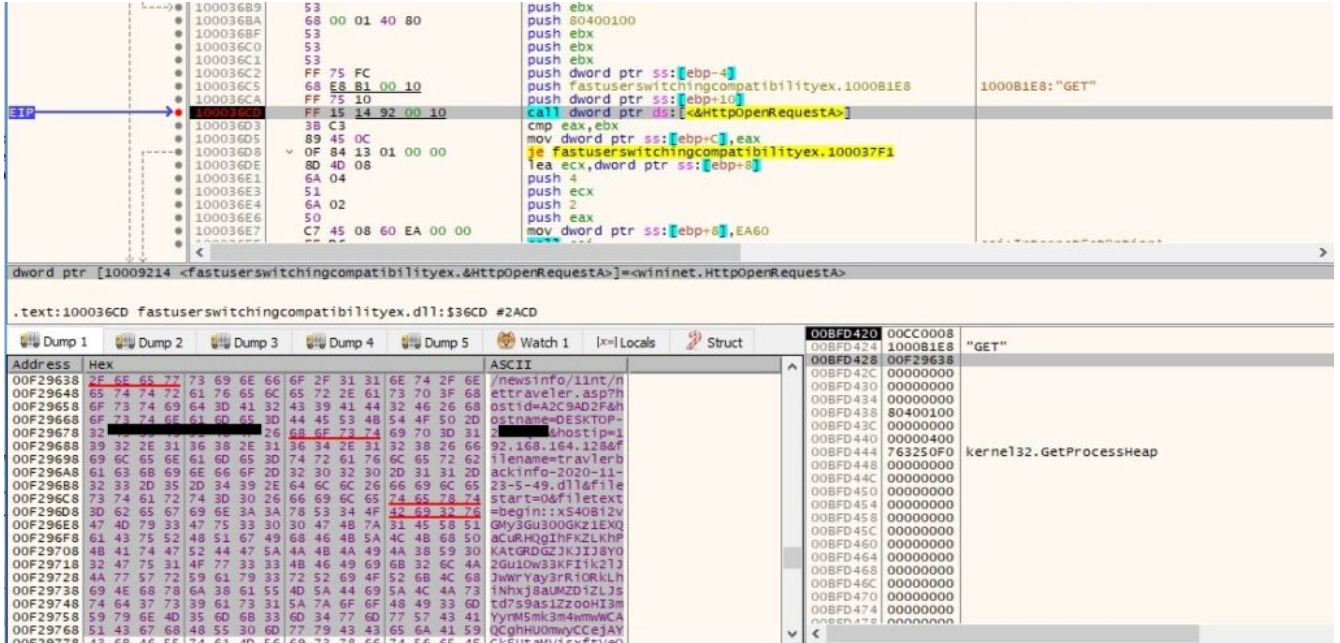


Figure 64

If the server response contains "Success:", the exfiltration was successful. The malicious user also deletes system_t.dll using DeleteFileA API. It performs another GET request (to the same C2 server) with the parameters including "action=getcmd" and others which were already explained above:



Figure 65

The result of the query must contain “[CmdBegin]\r\n” at the beginning of the message and “[CmdEnd]\r\n” at the end of it. The message between the “borders” is saved at “C:\Windows\System32\stat_t.ini” and then the process performs a GET request (same C2 server) with a modified parameter “action=gotcmd” and other parameters used before:

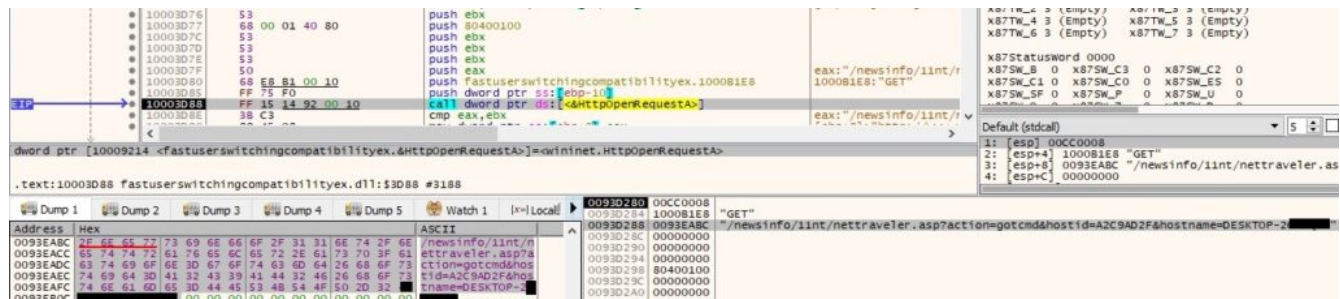


Figure 66

As before, if everything works fine the file expects an HTTP response which contains “Success” string. The process is looking to delete a file called “C:\Windows\SysWOW64\dnlist.ini” which doesn’t exist at this time. The file will be created and populated with the following data:

```
C:\windows\SysWOW64>type dnlist.ini
[Filelist]
filetotal=0
[ScanList]
dircount=0
ScanAll=True
[Other]
TypeLimit=True
USearch=True
GSearch=True
UTypeLimit=True
UAuto=False
Types=doc,docx,xls,xlsx,txt,rtf,pdf
```

Figure 67

File stat_t.ini is deleted using DeleteFileA function and then it calls GetACP API which returns the current Windows ANSI code page identifier for the operating system. Because the value of ScanAll is True in dnlist.ini, the malware scans for all available disk drives using GetLogicalDrives API and then compares the type of them with 3 (DRIVE_FIXED) or 4 (DRIVE_REMOTE) using GetDriveTypeA API:

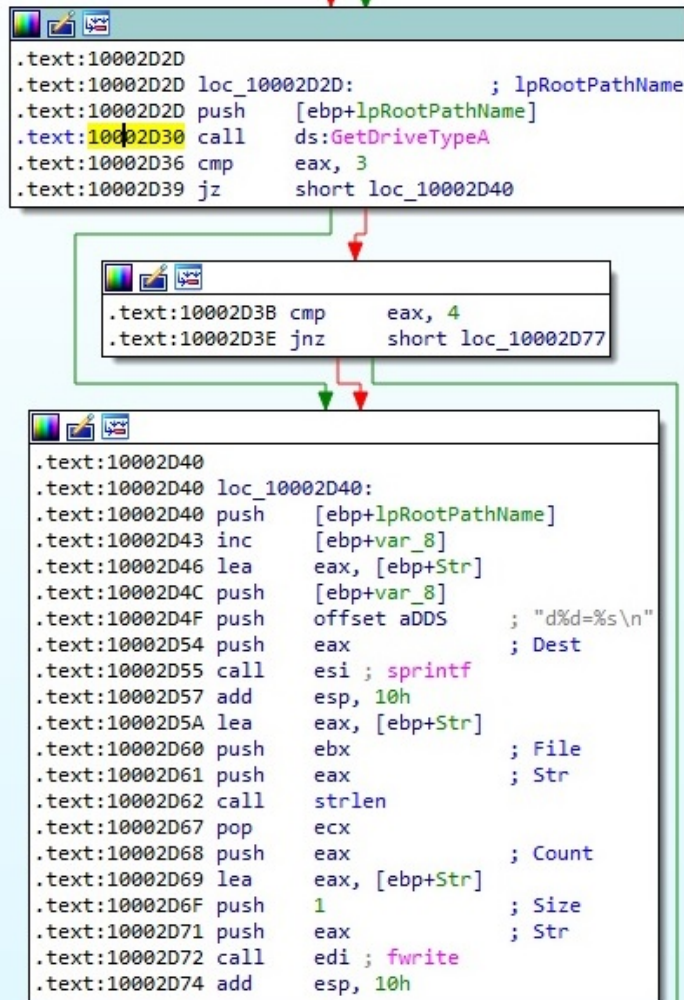


Figure 68

Let's suppose that "C:\\" is the first drive found by the process. The file will enumerate all files and directories from the "C:\\" drive and the directories name will be saved as dn (where n=1,2,3, ...) and the files name will be stored as fn (where n=1,2,3,...), together with filecount (total number of files) and dircount (total number of directories). All information described will be stored in a new file called "C:\Windows\SysWOW64\enumfs.ini":

```

enumfs.ini
1  [Computer]
2  Name=DESKTOP-2
3  Page=1252
4  [DESKTOP-2]
5  d1=C:\
6  dircount=1
7  [C:\]
8  d1=$Recycle.Bin
9  f1=$WINRE_BACKUP_PARTITION.MARKER
10 d2=706d7fdb2ede4d8e91
11 d3=Boot
12 f2=bootmgr
13 f3=BOOTNXT
14 f4=BOOTSECT.BAK
15 d4=Documents and Settings
16 d5=EFS Software
17 f5=eula.1028.txt
18 f6=eula.1031.txt
19 f7=eula.1033.txt
20 f8=eula.1036.txt
21 f9=eula.1040.txt
22 f10=eula.1041.txt
23 f11=eula.1042.txt
24 f12=eula.2052.txt
25 f13=eula.3082.txt
26 f14=globdata.ini
27 f15=install.exe
28 f16=install.ini
29 f17=install.res.1028.dll
30 f18=install.res.1031.dll
31 f19=install.res.1033.dll
32 f20=install.res.1036.dll
33 f21=install.res.1040.dll
34 f22=install.res.1041.dll
35 f23=install.res.1042.dll
36 f24=install.res.2052.dll
37 f25=install.res.3082.dll
38 d6=logs
39 f26=pagefile.sys
40 d7=PerfLogs
41 d8=Program Files
42 d9=Program Files (x86)
43 d10=ProgramData
44 d11=Python27
45 d12=python27-x64
46 d13=Recovery
47 f27=StarBurn.log
48 d14=Strawberry
49 f28=swapfile.sys
50 d15=System Volume Information
51 d16=temp
52 d17=Users
53 f29=vcredist.bmp
54 f30=VC_RED.cab
55 f31=VC_RED.MSI
56 d18=Windows
57 dircount=18
58 filecount=31
59

```

Figure 69

The operation applied to “C:\” drive is recursive and it’s applied to each directory (all information will be appended to enumfs.ini). The following information is added/modified in dnlist.ini:

```

[EnumTime]
DateTime = scan date
[ScanList]
ScanAll = False

```

The enumfs.ini file will be transferred to the C2 server via a GET request as before (compressed + encoded). The filename parameter has the following form: “FileList-<month><day>-<hour><minute><second>.ini”:

and "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility". Also the process deletes enumfs.ini, dnlst.ini, "C:\WINDOWS\system32\udidx.ini", uenumfs.ini and stat_t.ini. One such call is displayed below:

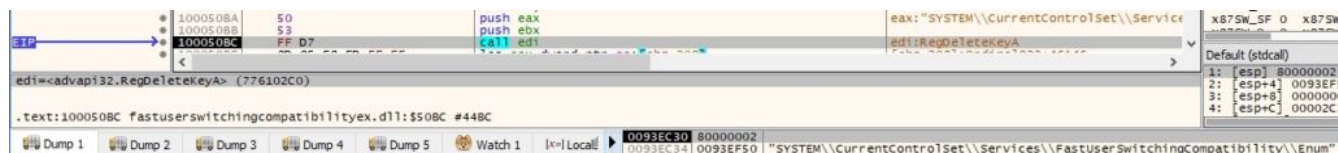


Figure 73

The C2 server is informed that the operation is complete by performing a GET request with "action=updated" parameter:

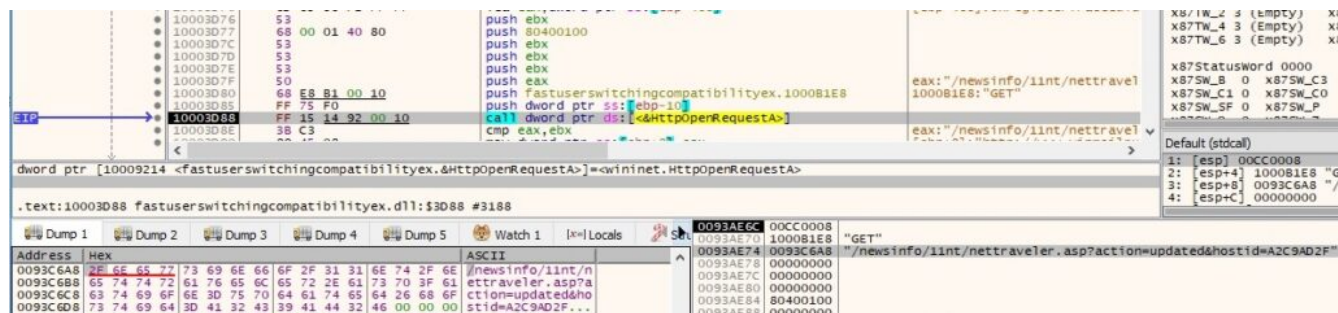


Figure 74

Case 2: (UPDATE)

Same registry keys and files are deleted as described above. Moreover, there is a GET request to the Command and Control server using "action=datsize" parameter and the HTTP response is supposed to include "Success:" if everything works smoothly:

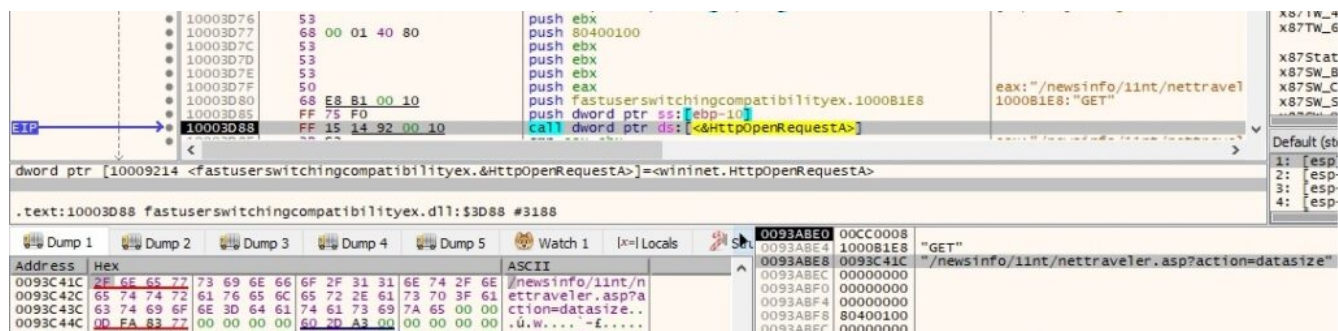


Figure 75

The malware is trying to download a file named updata.exe from the C2 server (this file not available for analysis, as the C2 server was down at the time of the analysis):

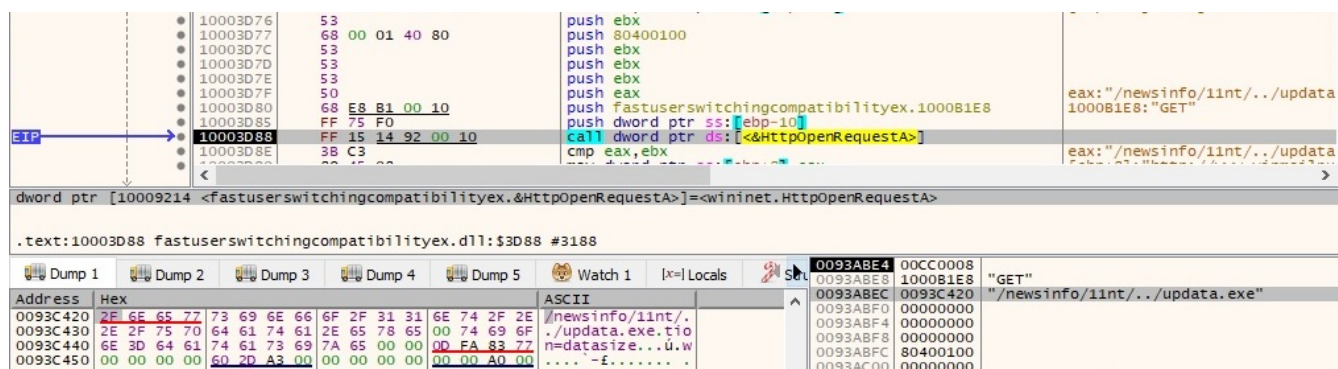


Figure 76

The magic bytes of the downloaded file are compared to “MZ” (the format for executable, DLL files in Windows) and also it’s looking for “PE” string at a specific offset as well. The downloaded file is saved as “C:\Windows\install.exe” and run by the malicious process:

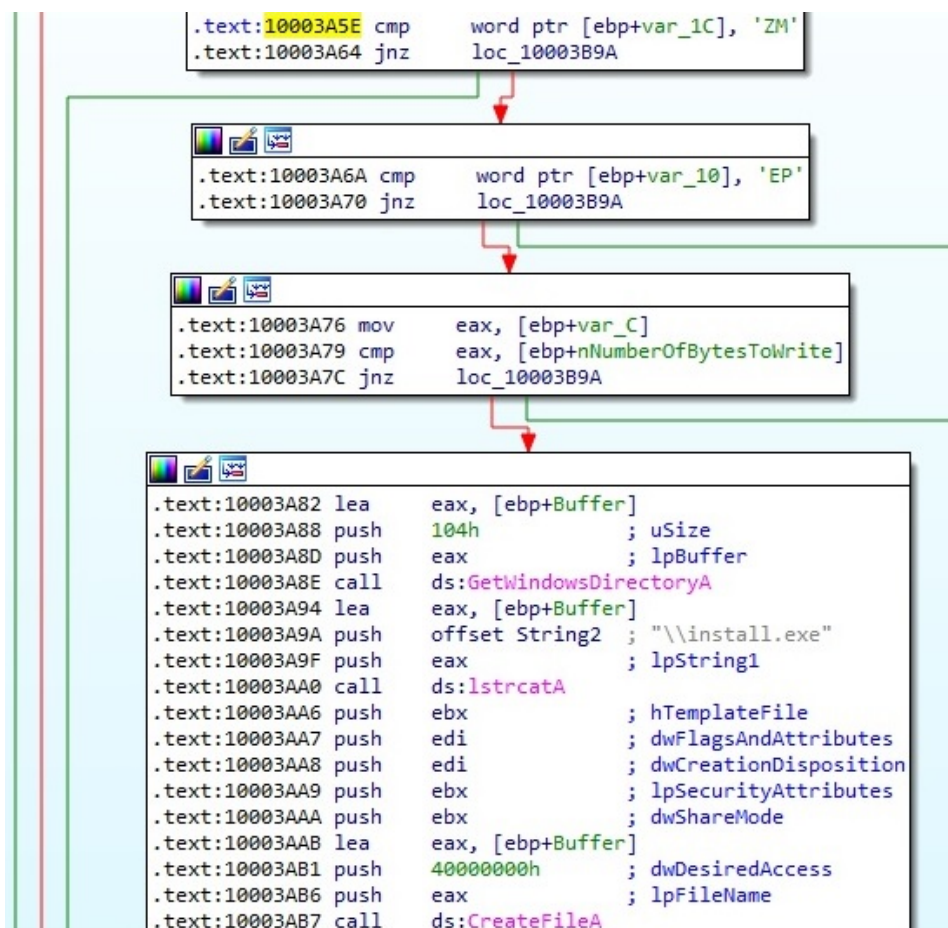


Figure 77

The same request as in Figure 74 is performed once more in order to keep the server in the loop for every new step.

Case 3: (RESET)

The following files are deleted: enumfs.ini, dnlist.ini, “C:\WINDOWS\system32\udidx.ini”, uenumfs.ini and stat_t.ini. Same request displayed in Figure 74 is used to contact the C2 server (this step is done in every case).

Case 4: (UPLOAD)

This case is identical to Case 2 (UPDATE) with the difference that no files/registry keys are deleted.

After the execution flow passes all cases, the process sleeps for 60 seconds and then it goes back in the loop.

Thread2 activity

RegisterClassA function is used to register a window class for use in CreateWindow/CreateWindowEx calls, it creates a windows using CreateWindowExA (windows class name is “NTMainWndClass”, 0x80000000 – WS_POPUP style). Also, the window procedure used in RegisterClassA API call (sub_10004535) is called 5 times as follows (one for each type of message): 0x81 (WM_NCCREATE), 0x83 (WM_NCCREATE), 0x01 (WM_CREATE), 0x05 (WM_SIZE) and 0x03 (WM_SIZE). We should also mention the following calls:

ShowWindow (Sets the specified window's show state), UpdateWindow (it sends a WM_PAINT message to the window), GetMessage (gets a message from the calling thread's message queue) and TranslateMessage (translates messages into character messages):

```
.text:100045E2 mov     edi, offset ClassName ; "NTMainWndClass"
.text:100045E7 push    eax                ; lpWndClass
.text:100045E8 mov     [ebp+WndClass.style], esi
.text:100045EB mov     [ebp+WndClass.lpfWndProc], offset sub_10004535
.text:100045F2 mov     [ebp+WndClass.cbClsExtra], esi
.text:100045F5 mov     [ebp+WndClass.cbWndExtra], esi
.text:100045F8 mov     [ebp+WndClass.hInstance], esi
.text:100045FB mov     [ebp+WndClass.hIcon], esi
.text:100045FE mov     [ebp+WndClass.hCursor], esi
.text:10004601 mov     [ebp+WndClass.hbrBackground], esi
.text:10004604 mov     [ebp+WndClass.lpszMenuName], esi
.text:10004607 mov     [ebp+WndClass.lpszClassName], edi
.text:1000460A call    ds:RegisterClassA
.text:10004610 test    ax, ax
.text:10004613 jz     short loc_1000467E
```

```
.text:10004615 push    esi                ; lpParam
.text:10004616 push    esi                ; hInstance
.text:10004617 push    esi                ; hMenu
.text:10004618 mov     eax, 80000000h
.text:1000461D push    esi                ; hWndParent
.text:1000461E push    eax                ; nHeight
.text:1000461F push    eax                ; nWidth
.text:10004620 push    esi                ; Y
.text:10004621 push    esi                ; X
.text:10004622 push    eax                ; dwStyle
.text:10004623 push    esi                ; lpWindowName
.text:10004624 push    edi                ; lpClassName
.text:10004625 push    esi                ; dwExStyle
.text:10004626 call    ds:CreateWindowExA
.text:1000462C mov     edi, eax
.text:1000462E cmp     edi, esi
.text:10004630 jz     short loc_1000467E
```

```
.text:10004632 push    esi                ; nCmdShow
.text:10004633 push    edi                ; hWnd
.text:10004634 call    ds:ShowWindow
.text:1000463A push    edi                ; hWnd
.text:1000463B call    ds:UpdateWindow
.text:10004641 mov     edi, ds:GetMessageA
.text:10004647 push    esi                ; wParamFilterMax
.text:10004648 push    esi                ; wParamFilterMin
.text:10004649 lea    eax, [ebp+Msg]
.text:1000464C push    esi                ; hWnd
```

Figure 78

The malware is interested in WM_DEVICECHANGE (0x219) messages with a parameter of DBT_DEVICEARRIVAL (0x8000) which means that for example a new USB drive has been plugged in or a network shared folder is mounted on the system:

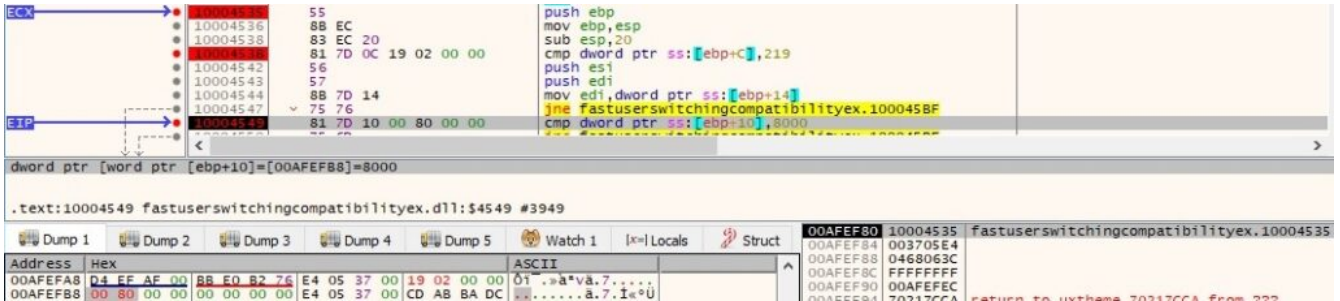


Figure 79

“USearch” and “UTypeLimit” values parsed from dnlist.ini are expected to be set to “True”, also “UAuto” value is “False” in dnlist.ini (this could indicate if the exfiltration of the targeted files should be automatically or not). The attacker is also interested in “Types” parameter (the targeted extensions) and we’ll see why in a bit. The idea is to scan each and every device inserted and also the network shares mounted on the host and create a “file system” structure in uenumfs.ini file (as it did in Thread1):

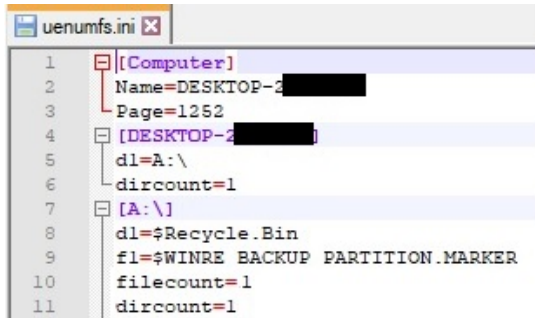


Figure 80

As in the first case, this search will apply for every directory found on the drive, recursively. The process creates a “C:\Users\\AppData\Local\Temp\ntvba00.tmp\” directory and its attribute is set to hidden. The following file is also created: “C:\Windows\SysWOW64\uenumfs.ini” (the content of it will be similar to enumfs.ini):

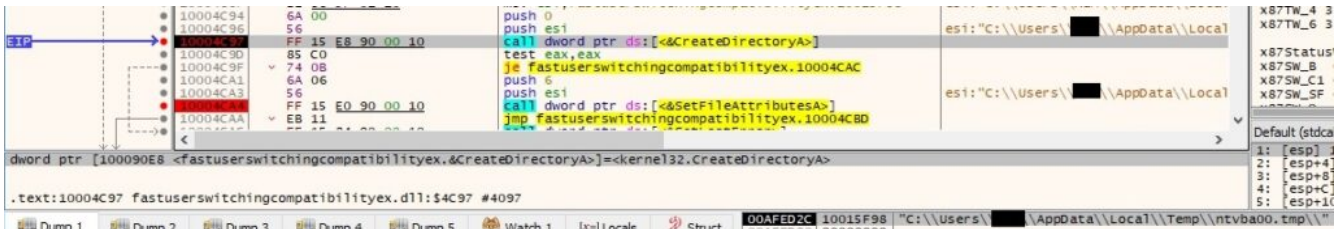


Figure 81

For each file found on the USB drive/network share, the process compares it’s extension with the list mentioned before: .doc, .docx, .xls, .xlsx, .txt, .rtf, .pdf:

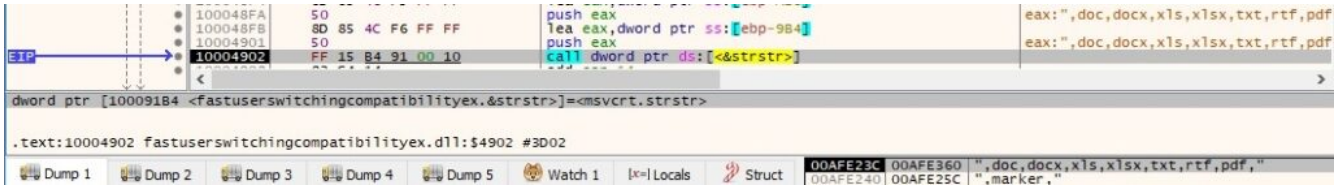


Figure 82

Let’s suppose that “C:\eula.1028.txt” (for the sake of simplicity) is a targeted file. The malware calculates a hash (MD5) of a combination between the filename and last modified timestamp (please note the initialize variables which correspond to MD5 algorithm):

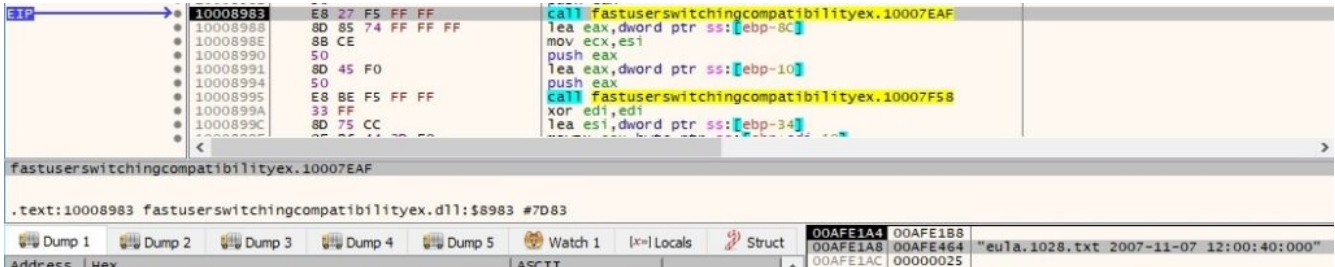


Figure 83

```

.text:10007E85
.text:10007E85
.text:10007E85
.text:10007E85 sub_10007E85 proc near
.text:10007E85
.text:10007E85 arg_0= dword ptr 4
.text:10007E85
.text:10007E85 mov     eax, [esp+arg_0]
.text:10007E89 and     dword ptr [eax+14h], 0
.text:10007E8D and     dword ptr [eax+10h], 0
.text:10007E91 mov     dword ptr [eax], 67452301h
.text:10007E97 mov     dword ptr [eax+4], 0EFCDA889h
.text:10007E9E mov     dword ptr [eax+8], 98BADCFEh
.text:10007EA5 mov     dword ptr [eax+0Ch], 10325476h
.text:10007EAC retn   4
.text:10007EAC sub_10007E85 endp
.text:10007EAC

```

Figure 84

After the function is finished the following result will represent the hash (unique identifier) corresponding to eula.1028.txt file:

Address	Hex	ASCII
00AFE210	35 66 37 61 37 38 65 37 39 32 37 35 33 32 62 61	5f7a78e7927532ba
00AFE220	32 61 39 33 30 65 63 38 64 34 37 65 32 35 32 61	2a930ec8d47e252a

Figure 85

Now "C:\eula.1028.txt" is copied to "C:\Users\\AppData\Local\Temp\ntvba00.tmp\U2007-11-07-12-00-5f7a78e7927532ba2a930ec8d47e252a.txt" (hidden file) – 2007 (year), 11 (month), 07 (day), 12 (hour), 00 (minute), 5f7a78e7927532ba2a930ec8d47e252a is the hash computed above (all values correspond to last modified timestamp):

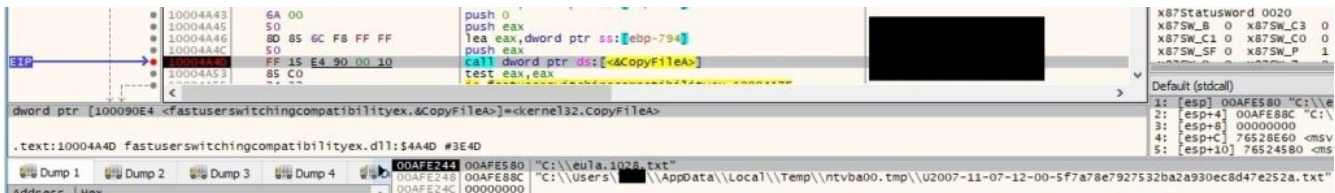


Figure 86

The process creates "C:\Windows\SysWOW64\udidx.ini" file and will add all hashes computed as explained before:



Figure 87

Last modified timestamp of the new file is set as the value extracted from the initial file:

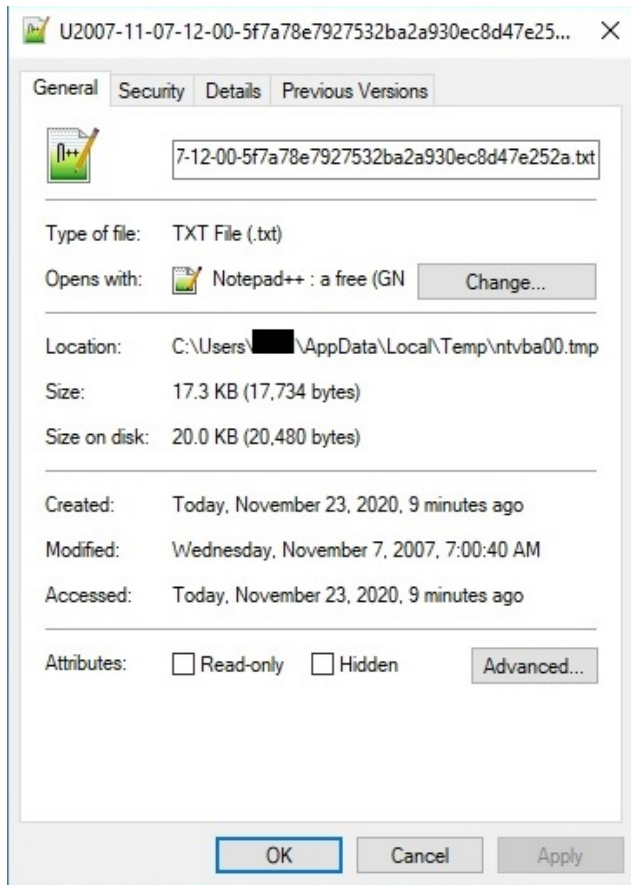


Figure 88

An example of udidx.ini file after copying all document-related files is shown in the figure below:



Figure 89

Finally the file is using DefWindowProcA API to ensure that window messages the application does not process have a default processing function (WM_DEVICECHANGE – 0x219, DBT_DEVICEARRIVAL – 0x8000):

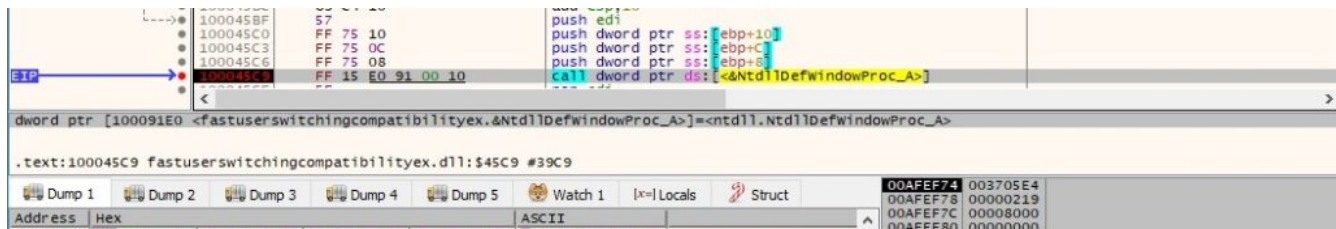


Figure 90

References

Kaspersky report: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08080841/kaspersky-the-net-traveler-part1-final.pdf>

VirusTotal link:

<https://www.virustotal.com/gui/file/feca8db35c0c0a901556eff447c38614d14a7140496963df2e613b206527b338/detection>

VirusTotal link:

<https://www.virustotal.com/gui/file/ed6ad64dad85fe11f3cc786c8de1f5b239115b94e30420860f02e820ffc53924/detection>

MSDN: <https://docs.microsoft.com/en-us/windows/win32/api/>

FireEye: [Advanced Persistent Threat Groups \(APT Groups\)](#)

DarkReading: [Chinese Cyberspies Pivot To Russia In Wake Of ... \(darkreading.com\)](#)

INDICATORS OF COMPROMISE

C2 domain: vipmailru[.]com

SHA256: FECA8DB35C0C0A901556EFF447C38614D14A7140496963DF2E613B206527B338

SHA256: ED6AD64DAD85FE11F3CC786C8DE1F5B239115B94E30420860F02E820FFC53924

Mutexes: "NetTravler Is Running!", "INSTALL SERVICES NOW!"

File names on disk:

%System%\config_t.dat

%windir%\system32\enumfs.ini

%windir%\system32\dnlist.ini

%windir%\system32\udidx.ini

%windir%\system32\uenumfs.ini

%windir%\system32\stat_t.ini

%windir%\system32\system_t.dll

%windir%\install.exe

%TEMP%\ntvba00.tmp\

temp.bat