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:

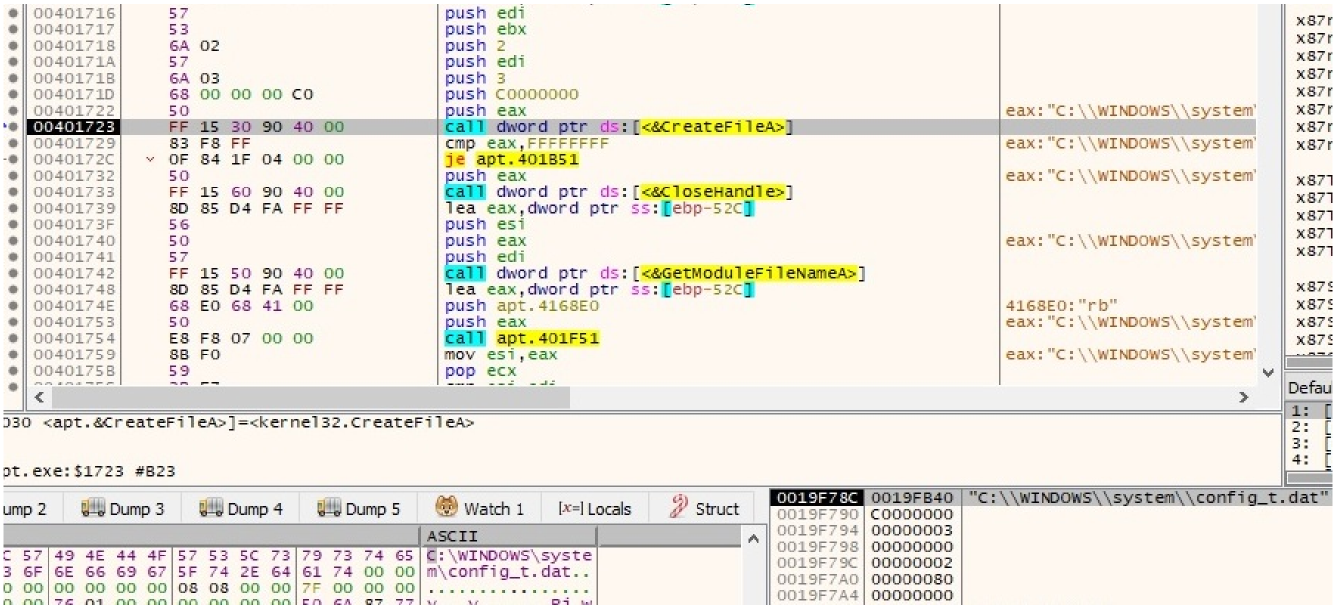


Figure 2

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

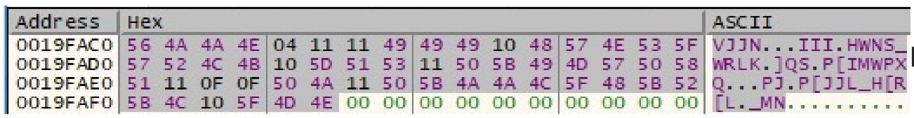


Figure 3

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

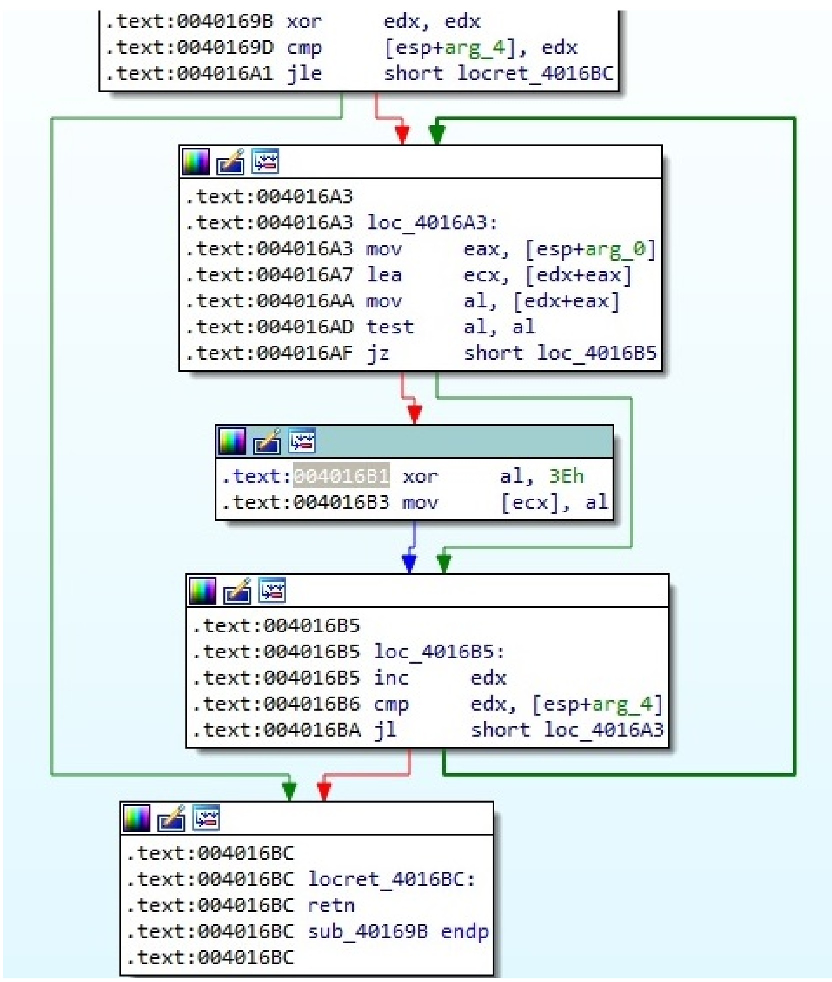


Figure 4

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 3A 2F 2F 77 77 77 2E 76 69 70 6D 61	http://www.vipma
0019FAD0	69 6C 72 75 2E 63 6F 6D 2F 6E 65 77 73 69 6E 66	ilru.com/newsinf
0019FAE0	6F 2F 31 31 6E 74 2F 6E 65 74 74 72 61 76 65 6C	o/11nt/nettravel
0019FAF0	65 72 2E 61 73 70 00 00 00 00 00 00 00 00 00 00	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:

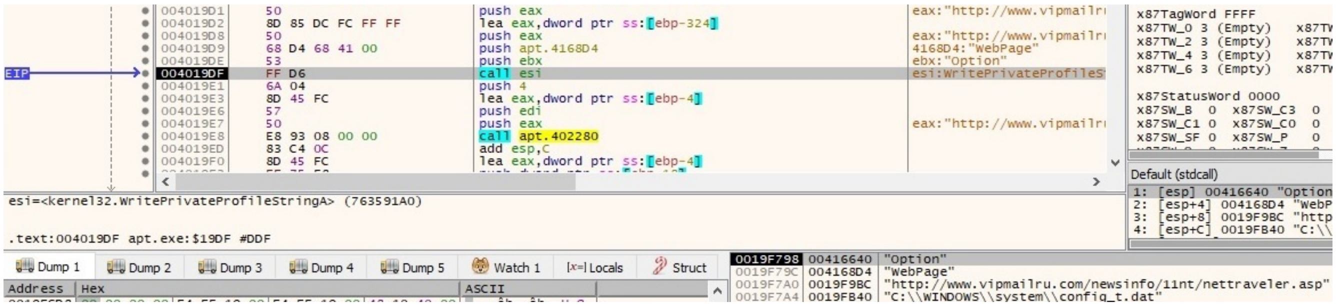


Figure 6

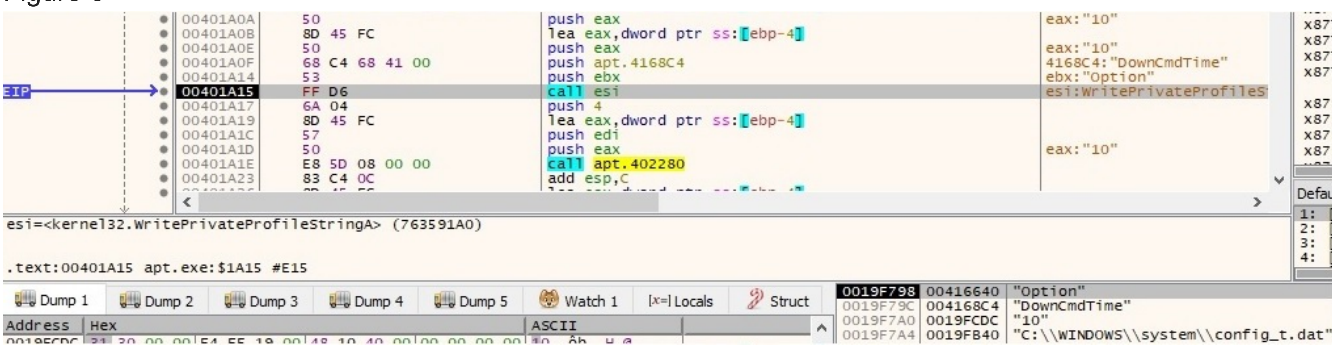


Figure 7

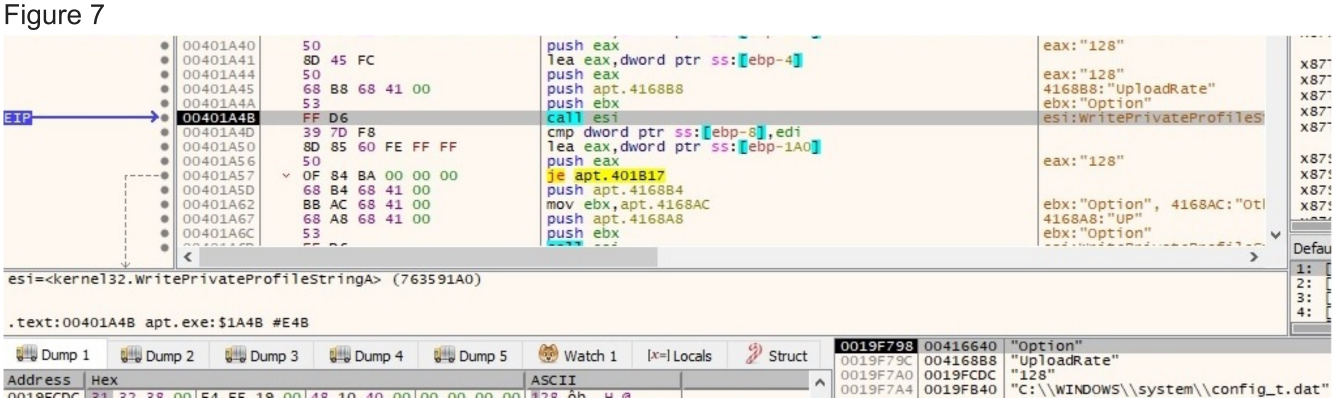


Figure 8

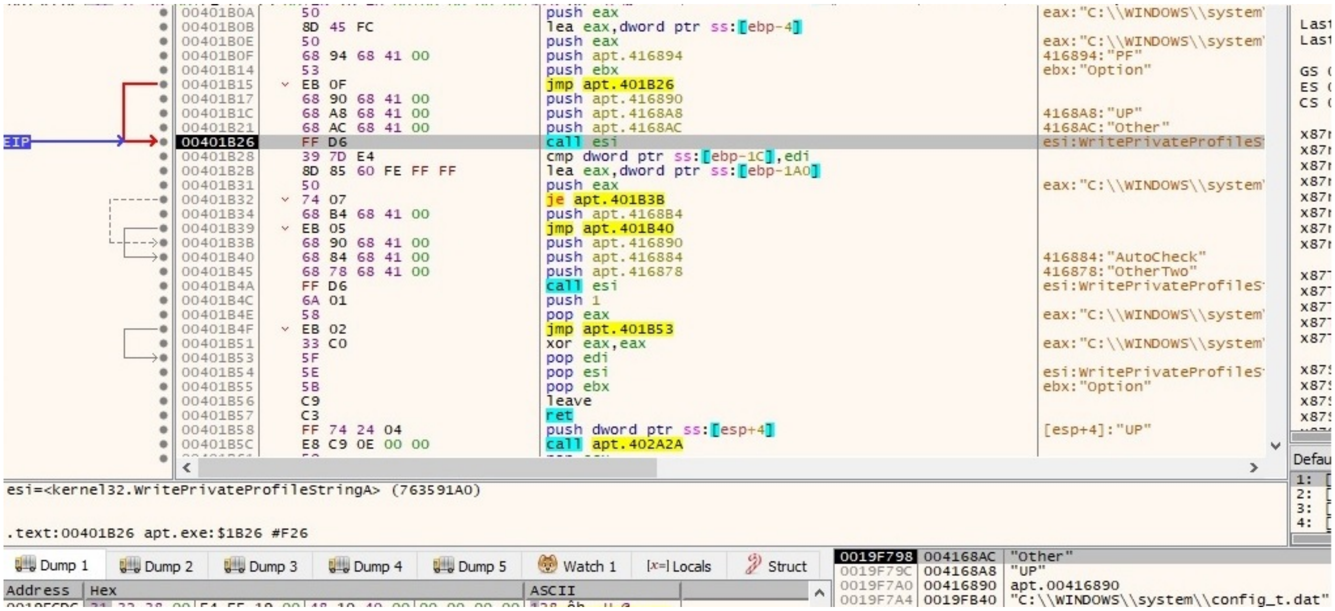


Figure 9

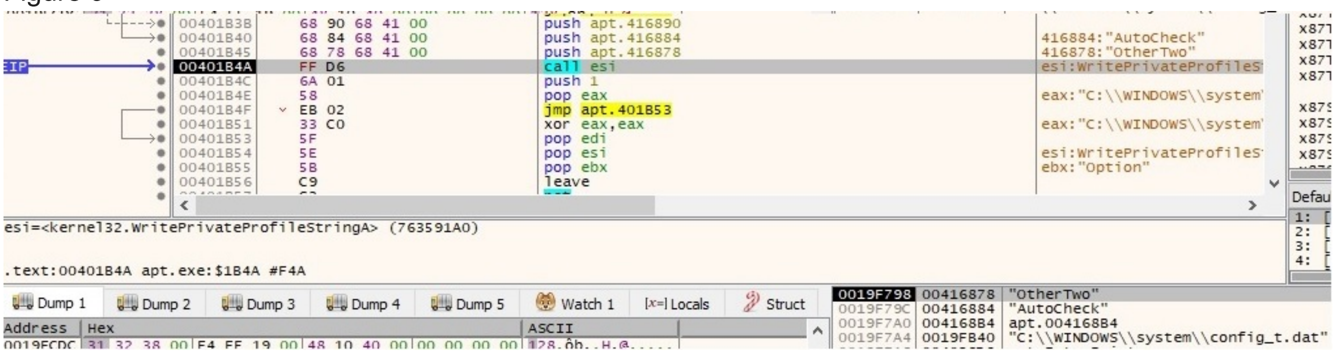
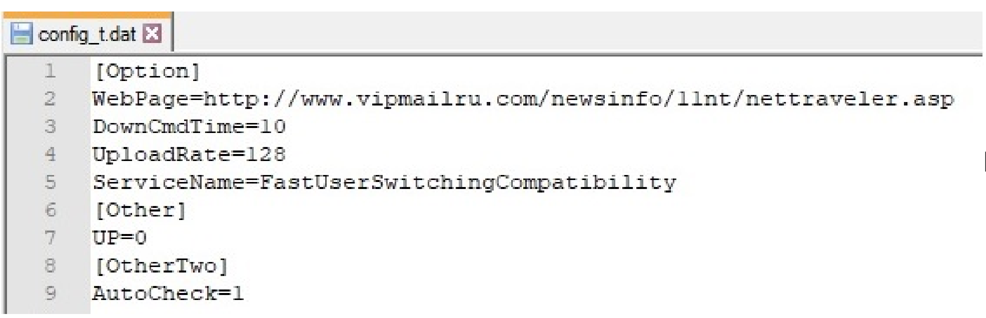


Figure 10

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



```
[Option]
WebPage=http://www.vipmailru.com/newsinfo/1lnt/nettraveler.asp
DownCmdTime=10
UploadRate=128
ServiceName=FastUserSwitchingCompatibility
[Other]
UP=0
[OtherTwo]
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”:

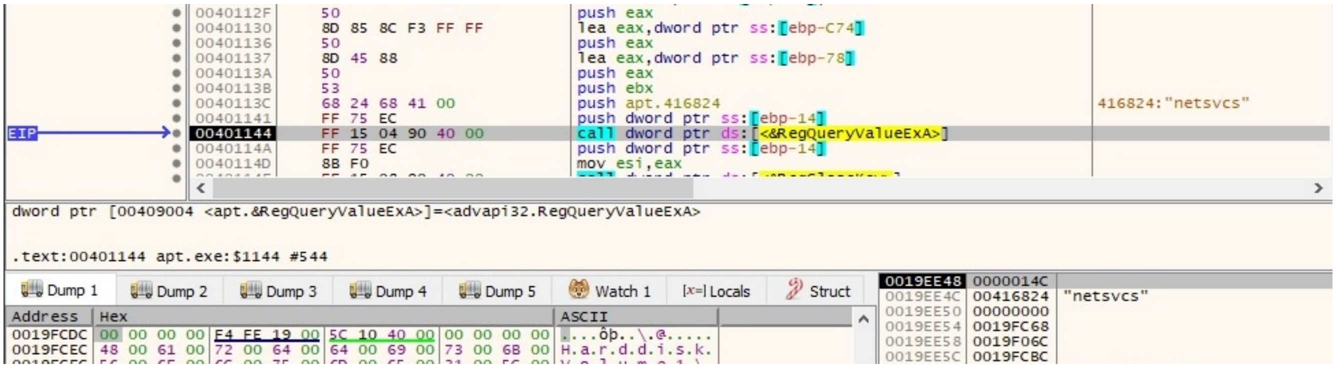


Figure 12

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.

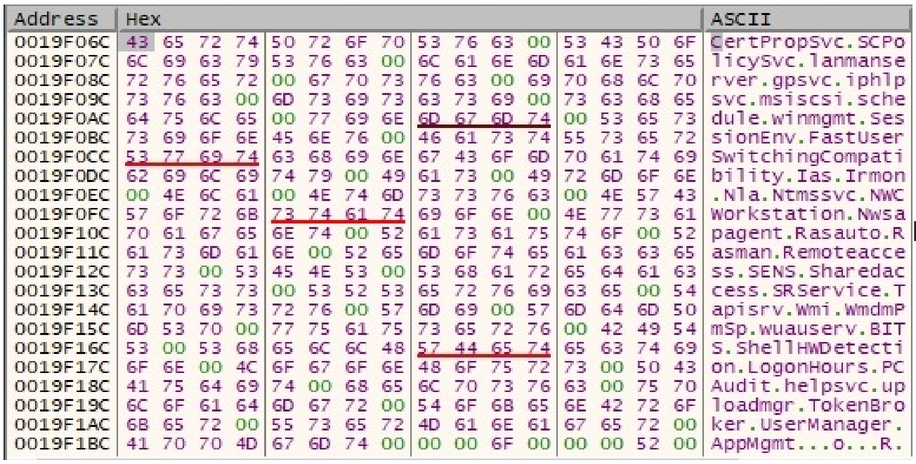


Figure 13

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, iphlpsvc, 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:

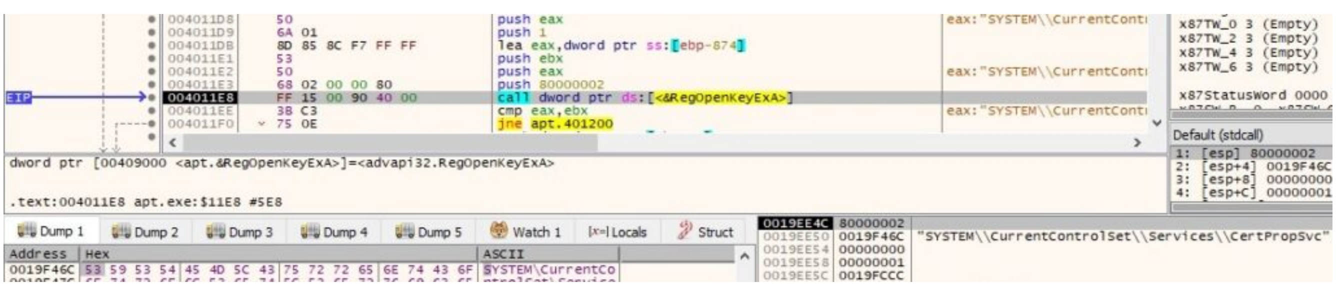


Figure 14

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):

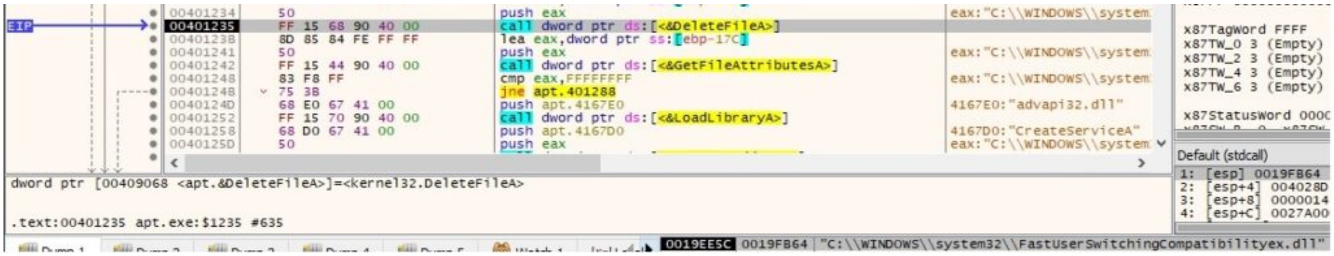


Figure 15
A new service called “FastUserSwitchingCompatibility” is created using CreateServiceA API function which tries to impersonate the legitimate service, the binary path of the service being %SystemRoot%\System32\svchost.exe -k netsvcs (legitimate process):

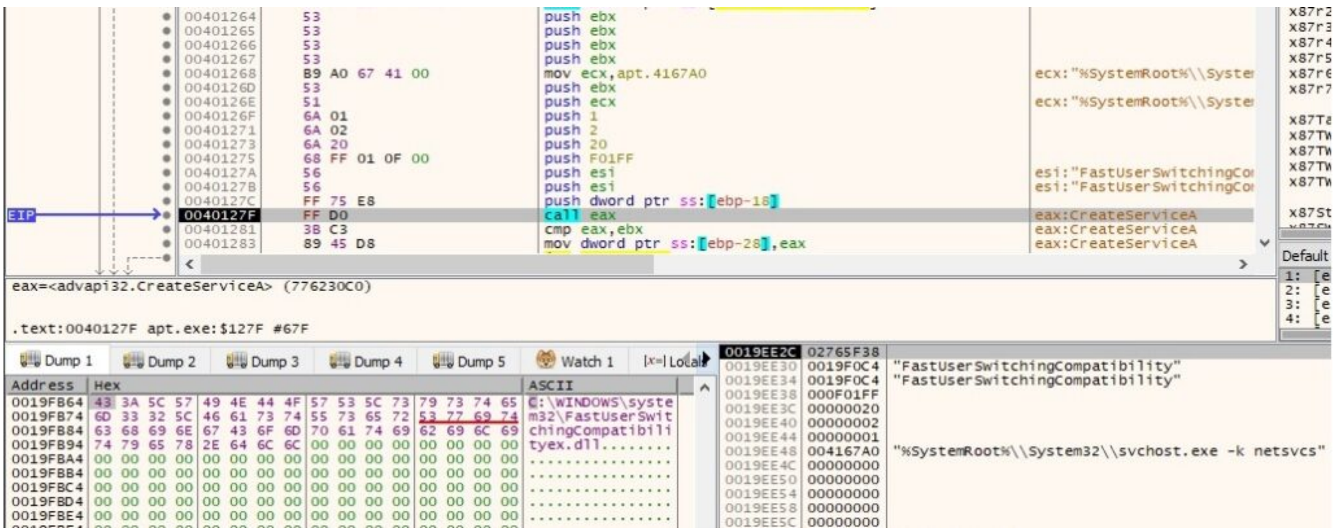


Figure 16
If the call is successful we’ll see a registry key like the one displayed in figure 17. (this technique is part of evasion techniques). Attackers will try to impersonate/use legitimate system binaries or libraries on the host to hide malicious activity. This will allow them to blend with regular activity and remain hidden. (you can find more details about lolbins at https://lolbas-project.github.io/).

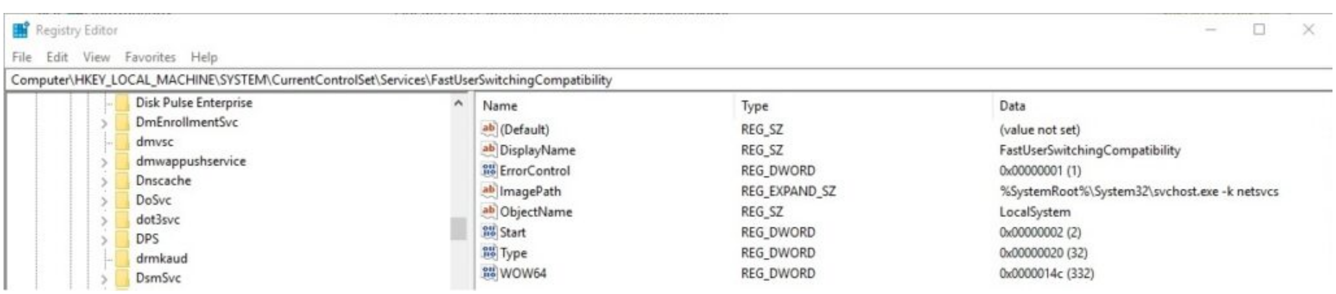


Figure 17
In order to verify that the service was successfully created the malicious process tries to open “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\FastUserSwitchingCompatibility” (now it exists because it corresponds to the newly created service):

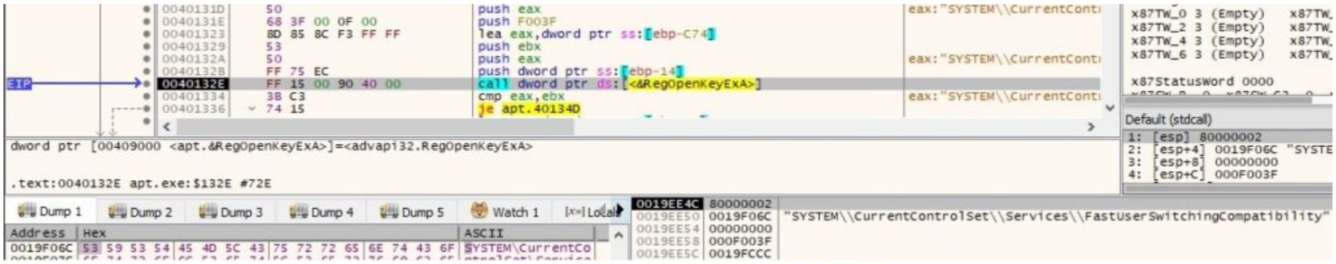


Figure 18

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:

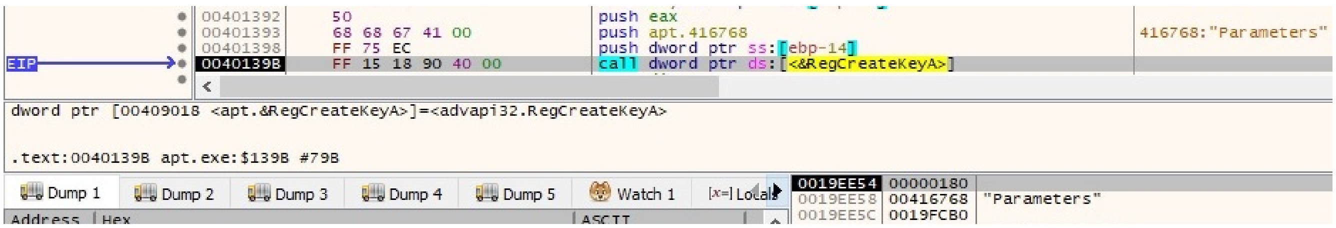


Figure 19

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:

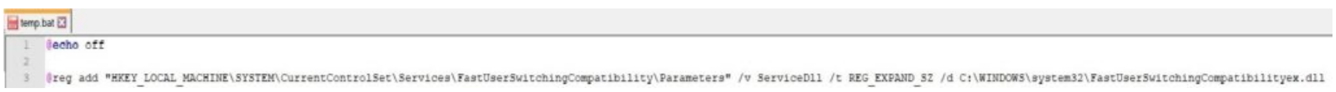


Figure 20

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):

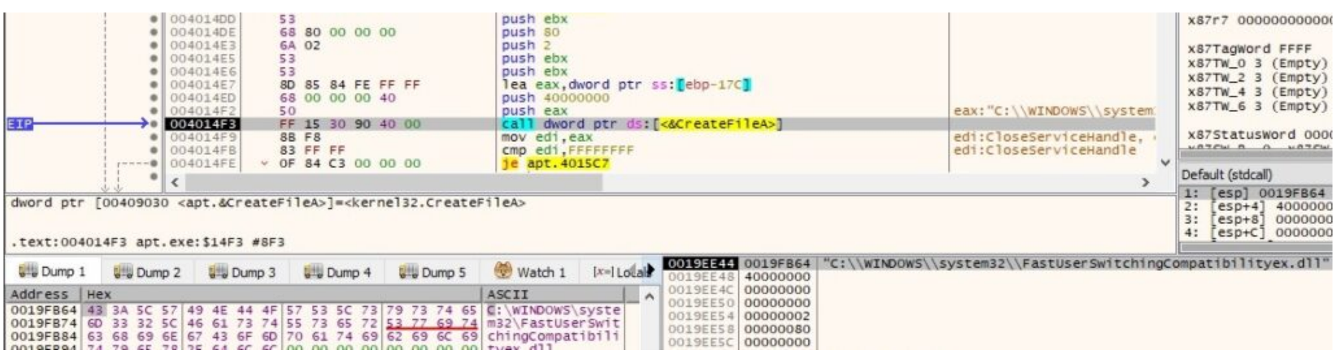


Figure 21

“Timestomping” is a tehnique 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:

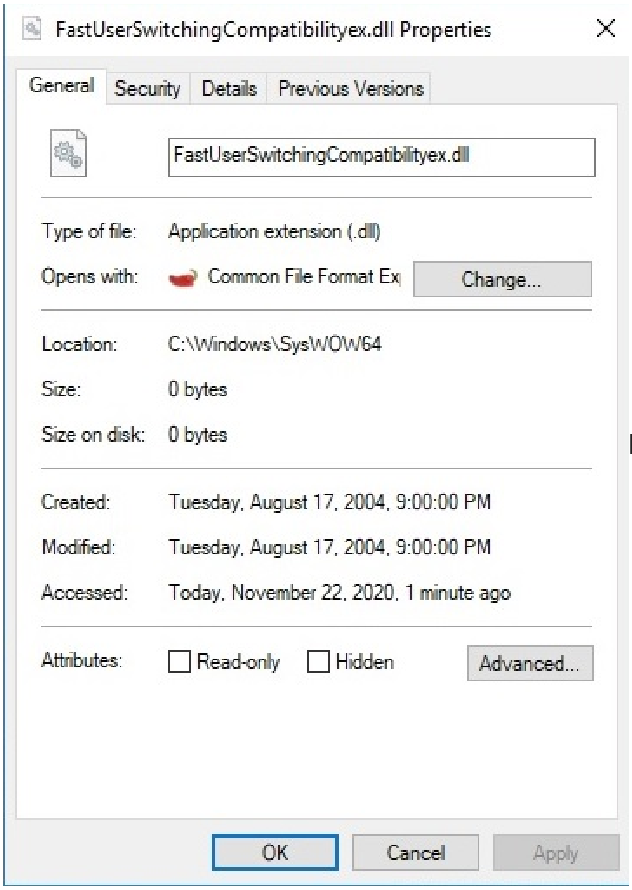


Figure 22

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:

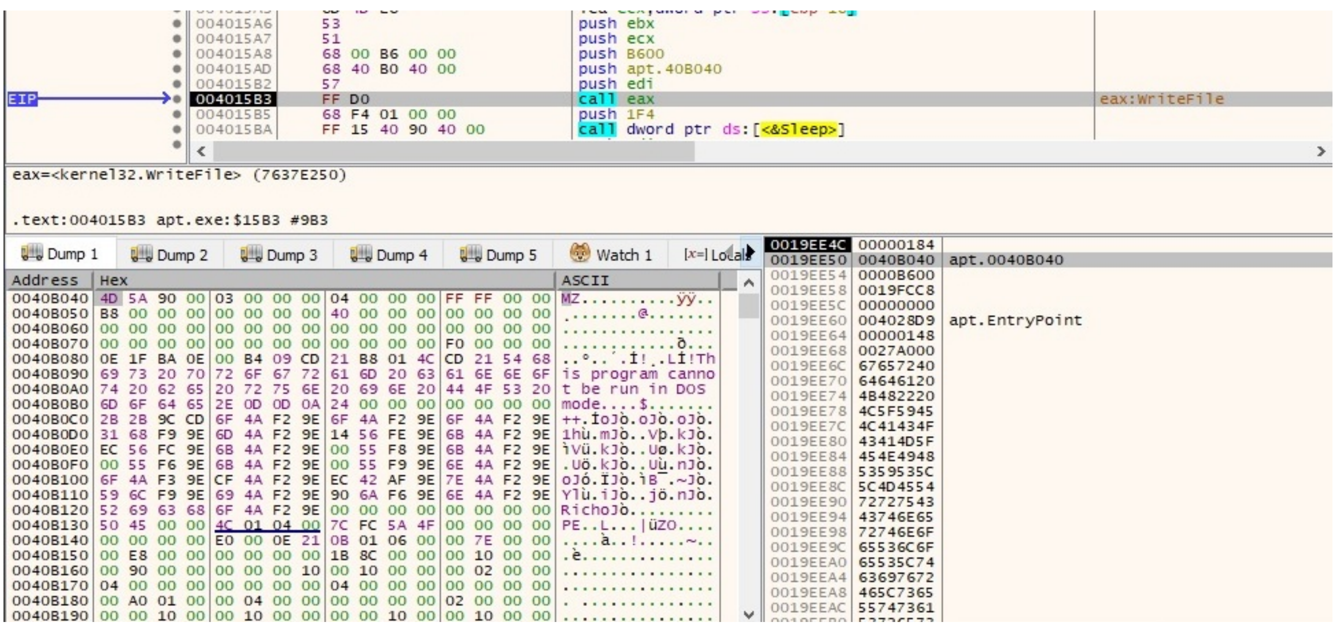


Figure 23

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:

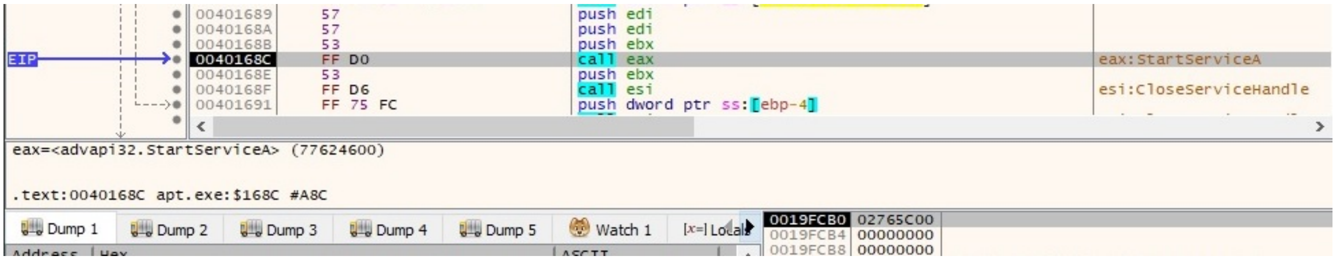


Figure 24

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:

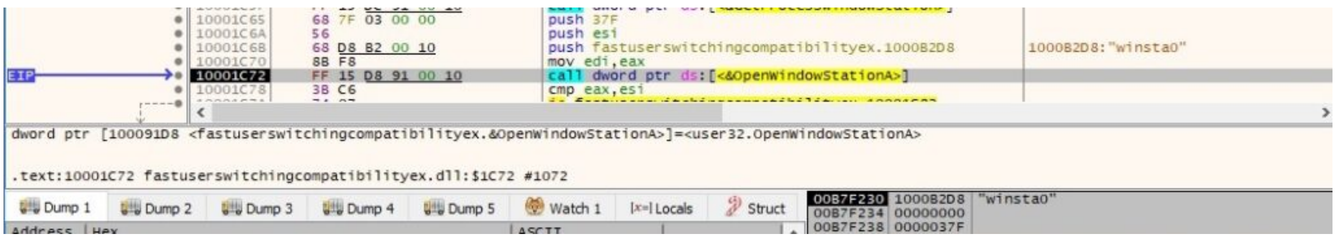


Figure 25

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:

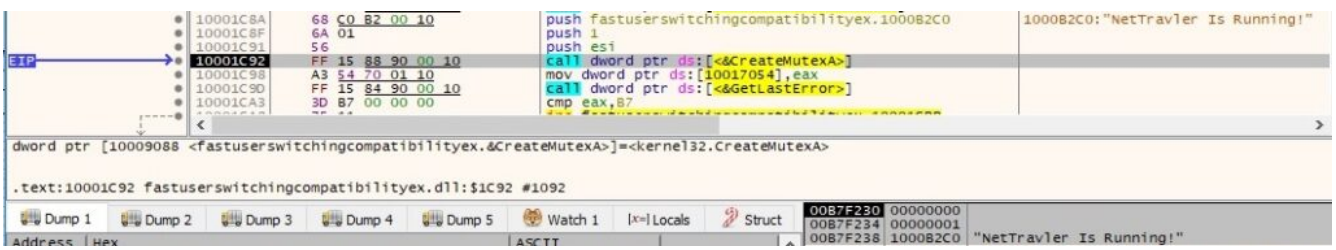


Figure 26

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:

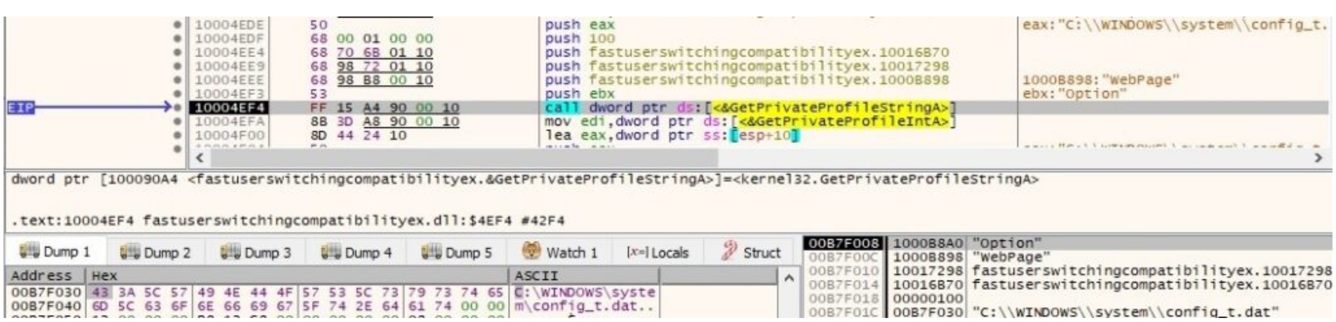


Figure 27

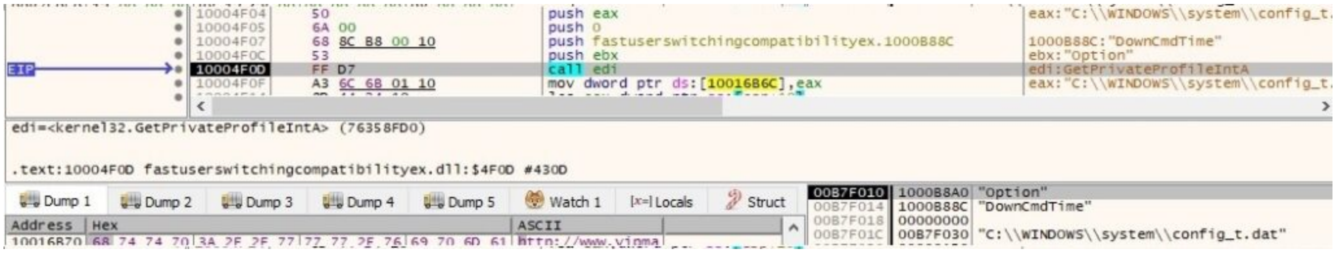


Figure 28

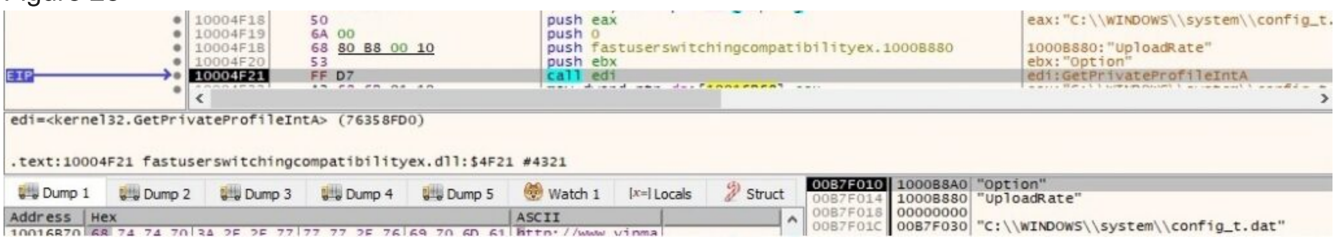


Figure 29

Figure 30

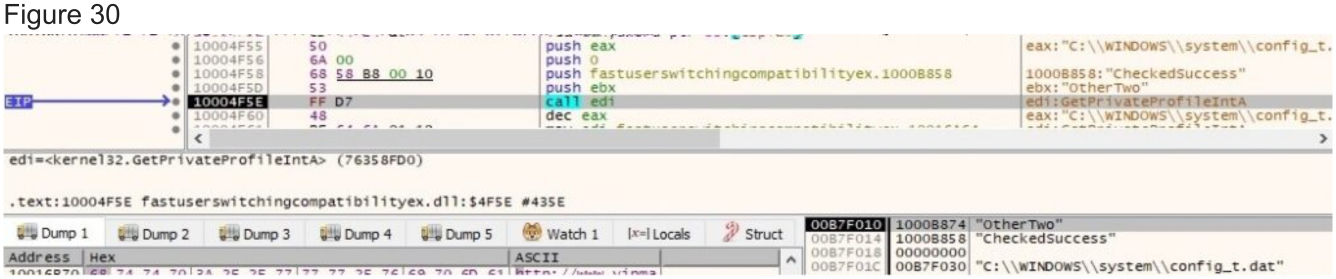


Figure 31

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"):

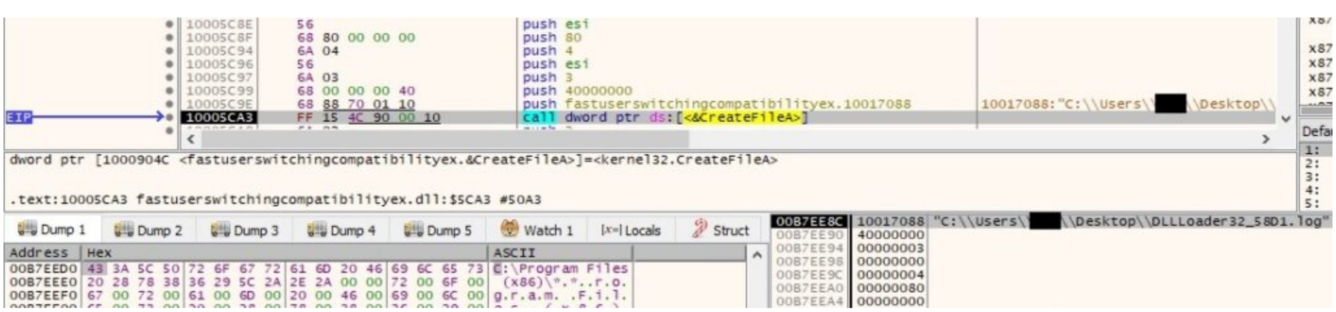
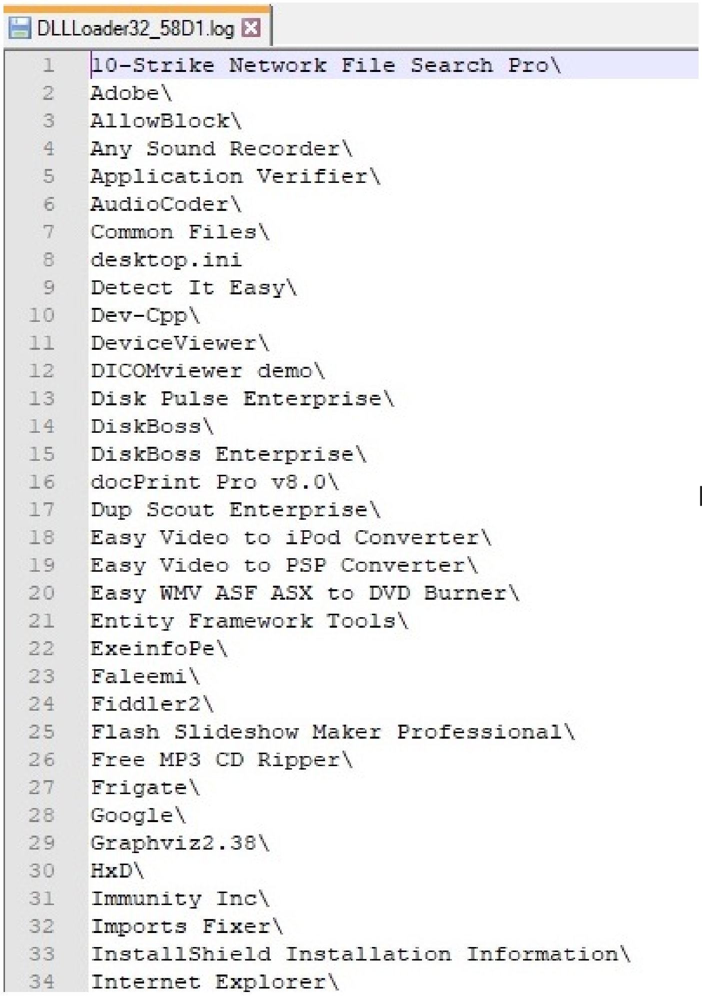


Figure 33

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


DLLLoader32_58D1.log

10-Strike Network File Search Pro\
Adobe\
AllowBlock\
Any Sound Recorder\
Application Verifier\
AudioCoder\
Common Files\
desktop.ini
Detect It Easy\
Dev-Cpp\
DeviceViewer\
DICOMviewer demo\
Disk Pulse Enterprise\
DiskBoss\
DiskBoss Enterprise\
docPrint Pro v8.0\
Dup Scout Enterprise\
Easy Video to iPod Converter\
Easy Video to PSP Converter\
Easy WMV ASF ASX to DVD Burner\
Entity Framework Tools\
ExeinfoPe\
Faleemi\
Fiddler2\
Flash Slideshow Maker Professional\
Free MP3 CD Ripper\
Frigate\
Google\
Graphviz2.38\
HxD\
Immunity Inc\
Imports Fixer\
InstallShield Installation Information\
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”:

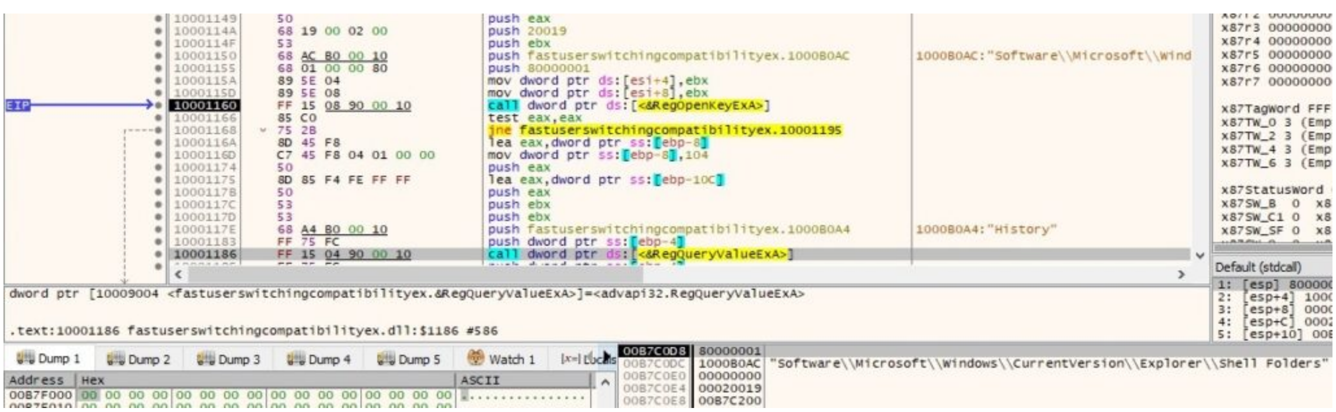


Figure 35

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:

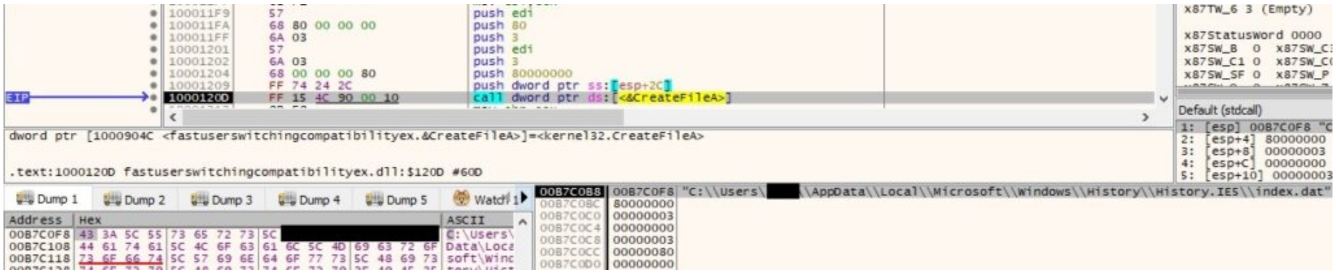


Figure 36

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

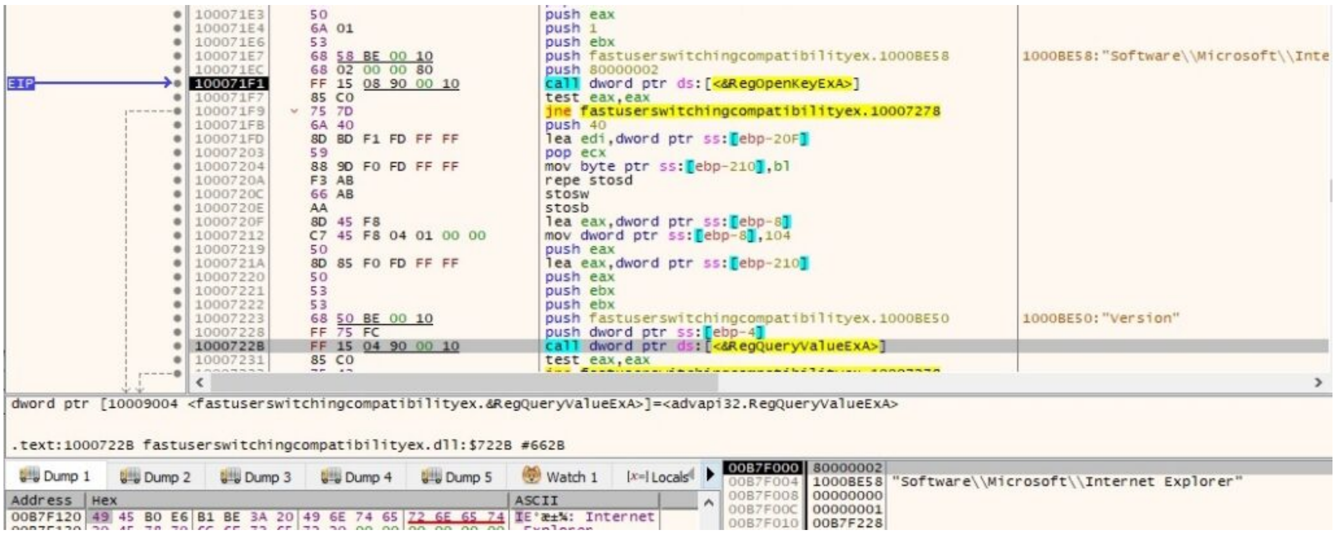


Figure 37

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

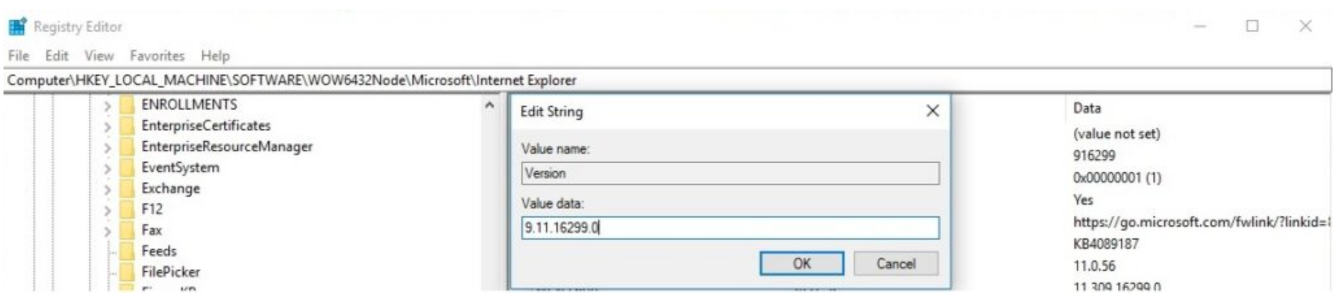


Figure 38

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”):

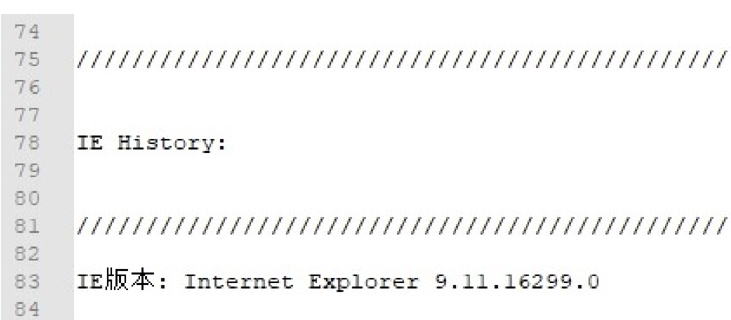


Figure 39

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:

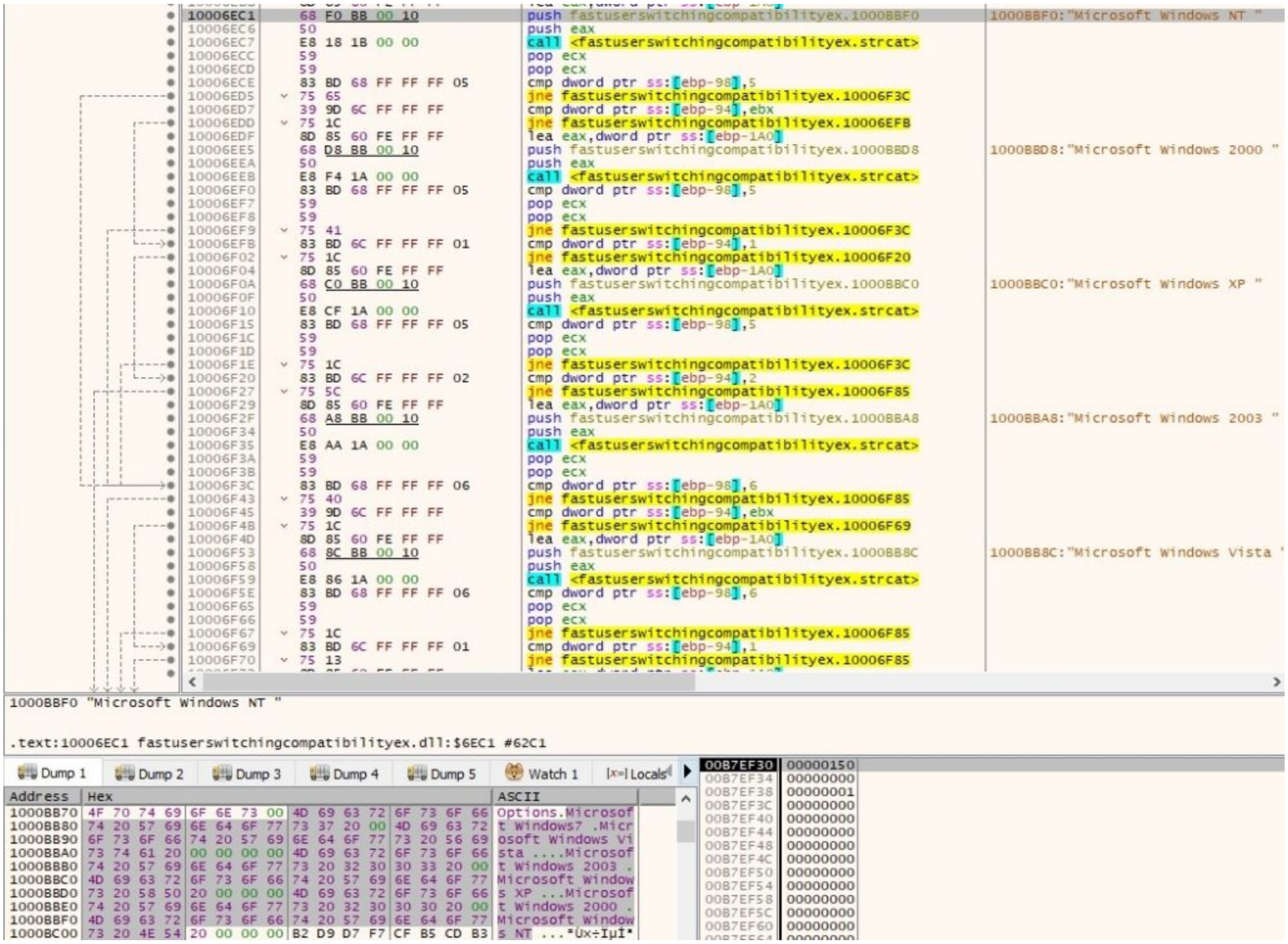


Figure 40

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":

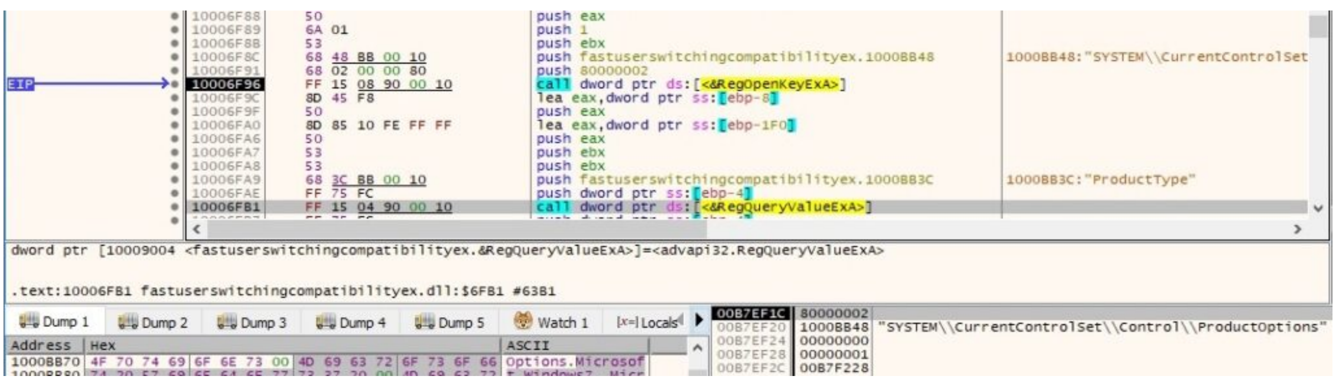


Figure 41

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

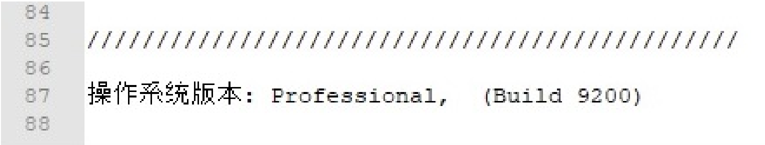


Figure 42

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:

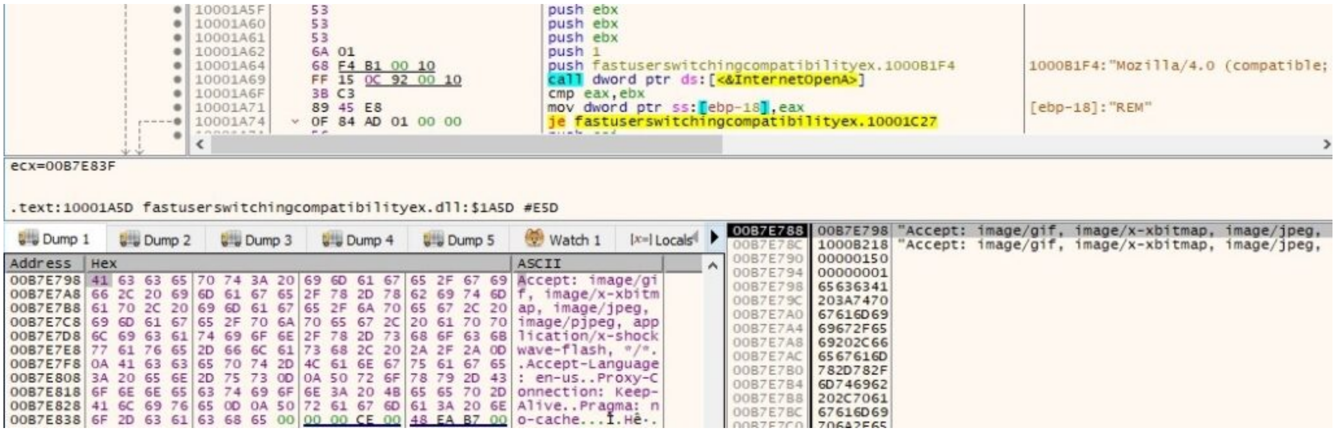


Figure 43

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:

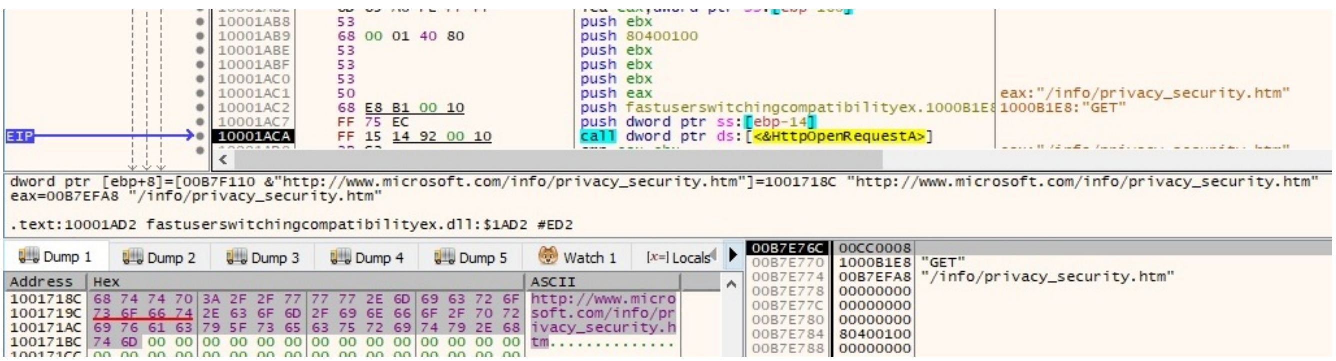


Figure 44

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

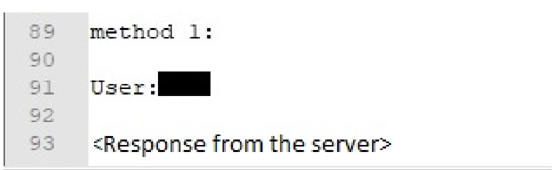


Figure 45

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:

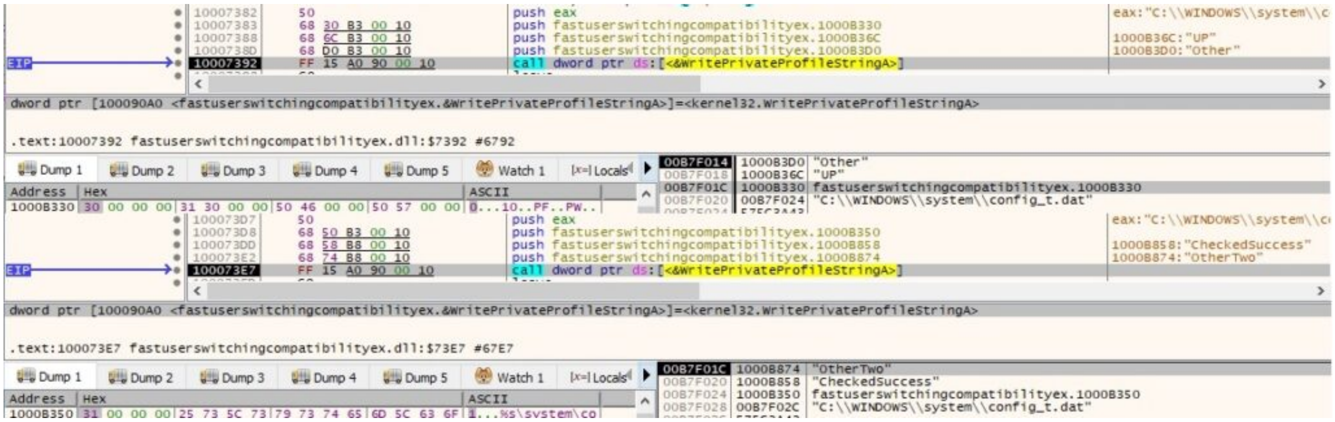


Figure 46

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:

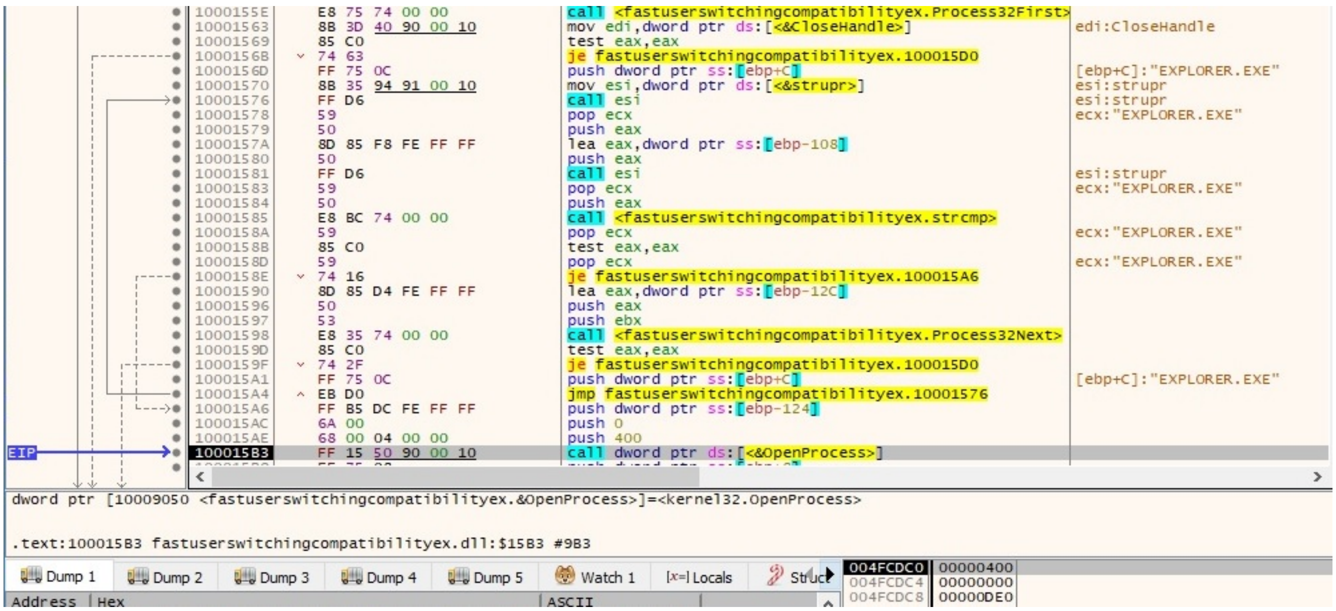


Figure 47

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.

Figure 48

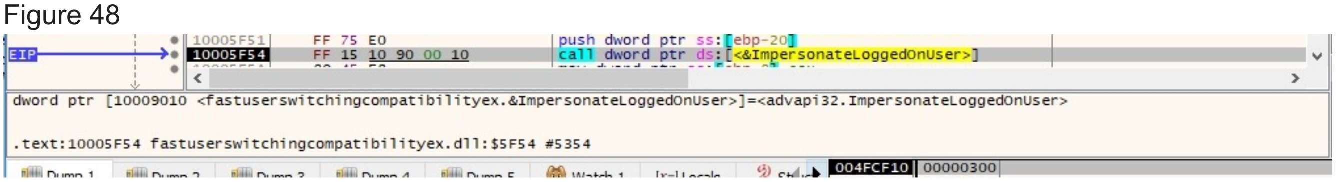


Figure 49

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:

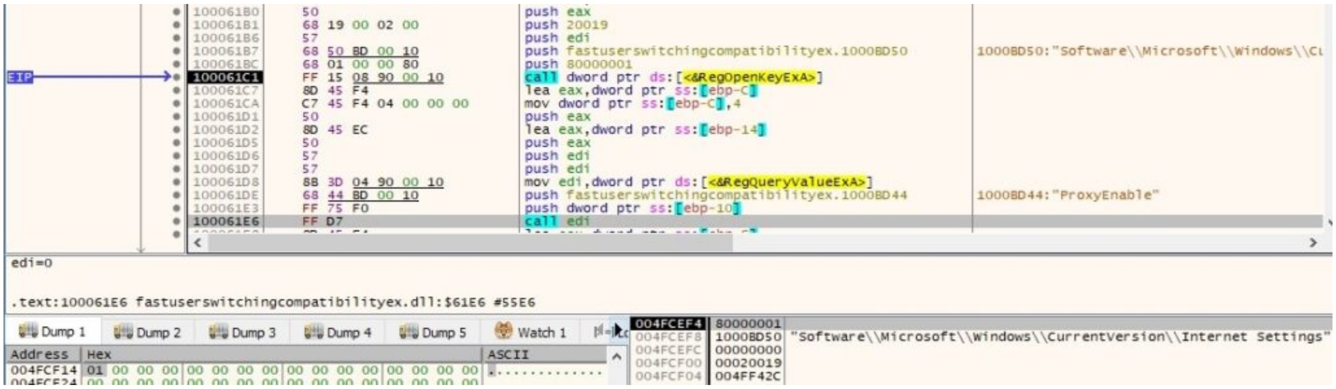


Figure 50

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:

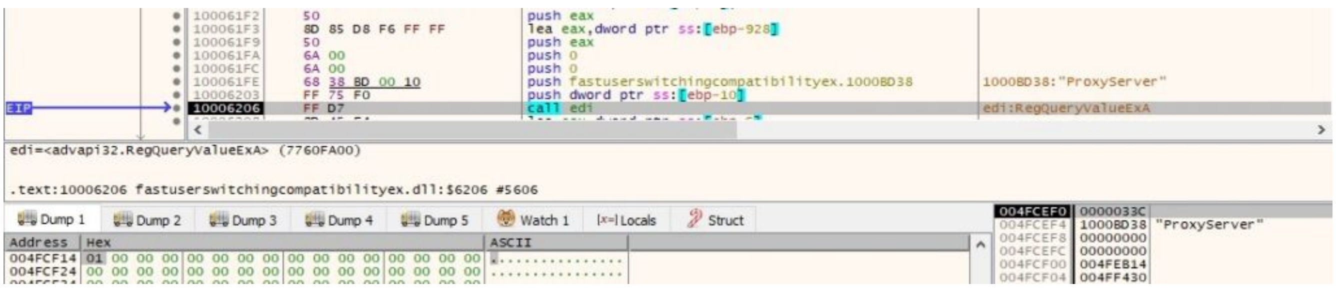


Figure 51

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:

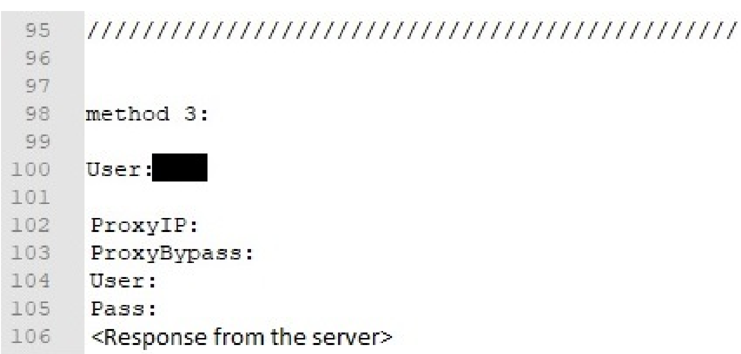


Figure 52

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:

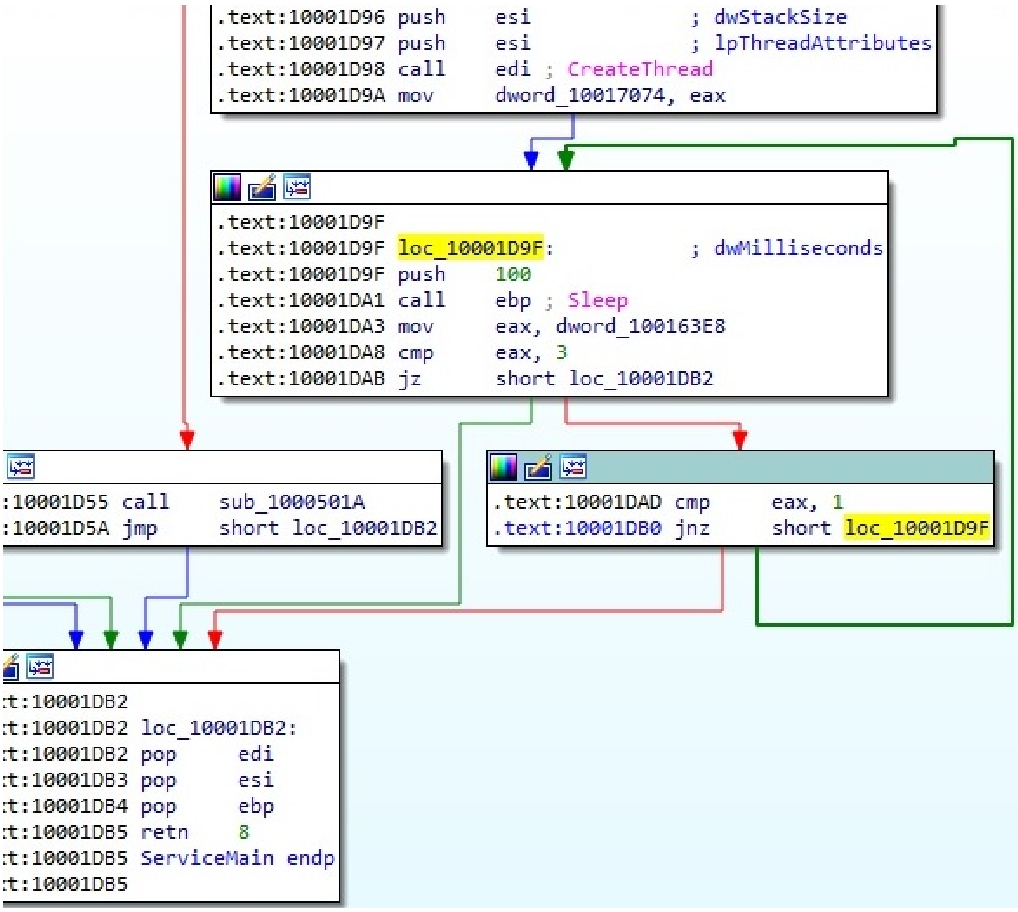


Figure 53

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:

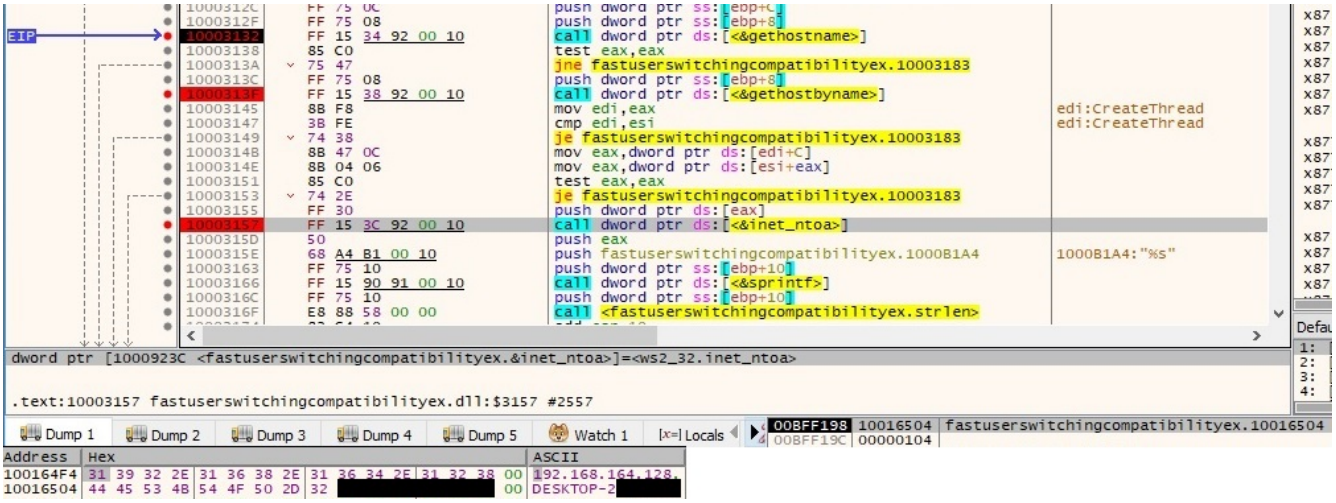


Figure 54

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

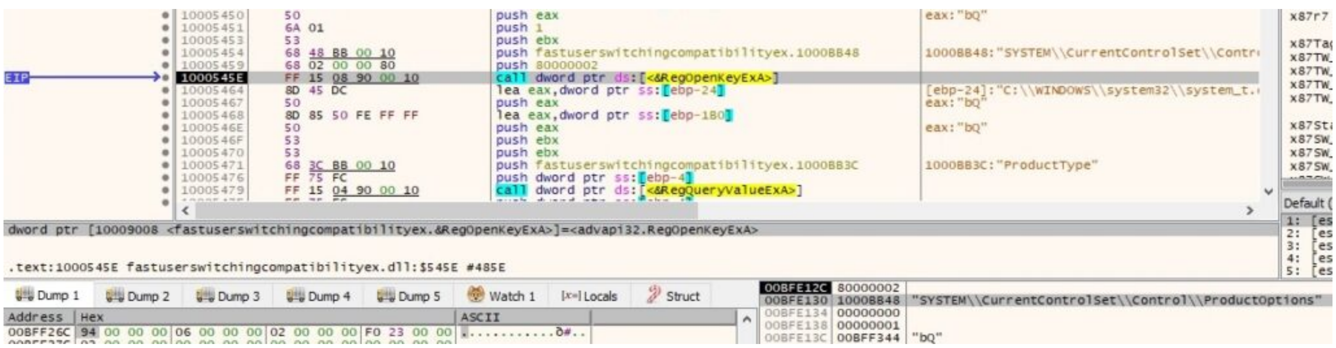


Figure 55

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

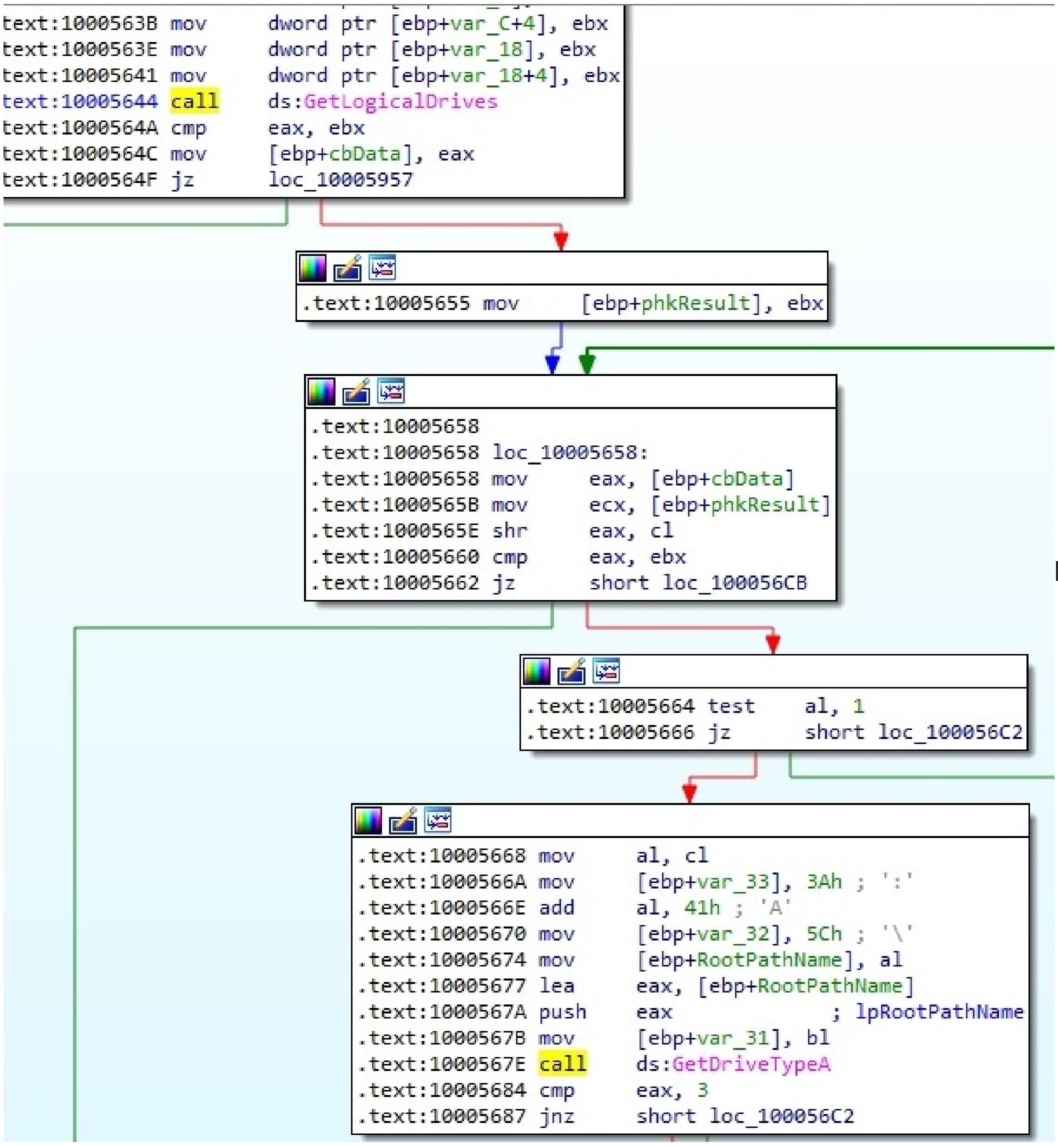


Figure 56

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:

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"):

```
[进程列表]
   0 [System Process]
   4 System
 500 smss.exe
 604 csrss.exe
 672 wininit.exe
 684 csrss.exe
 732 winlogon.exe
 796 services.exe
 804 lsass.exe
 884 svchost.exe
 892 fontdrvhost.exe
 900 fontdrvhost.exe
 980 svchost.exe
   8 dwm.exe
 576 svchost.exe
1044 svchost.exe
1060 svchost.exe
1076 svchost.exe
1200 svchost.exe
1292 Memory Compression
1444 svchost.exe
1452 svchost.exe
1544 svchost.exe
1552 svchost.exe
1636 svchost.exe
1712 svchost.exe
1776 spoolsv.exe
2028 WmiPrvSE.exe
2080 armsvc.exe
2096 svchost.exe
2112 diskpls.exe
2136 dupscts.exe
2164 IpOverUsbSvc.exe
2220 TenorshareWinAdService.exe
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:

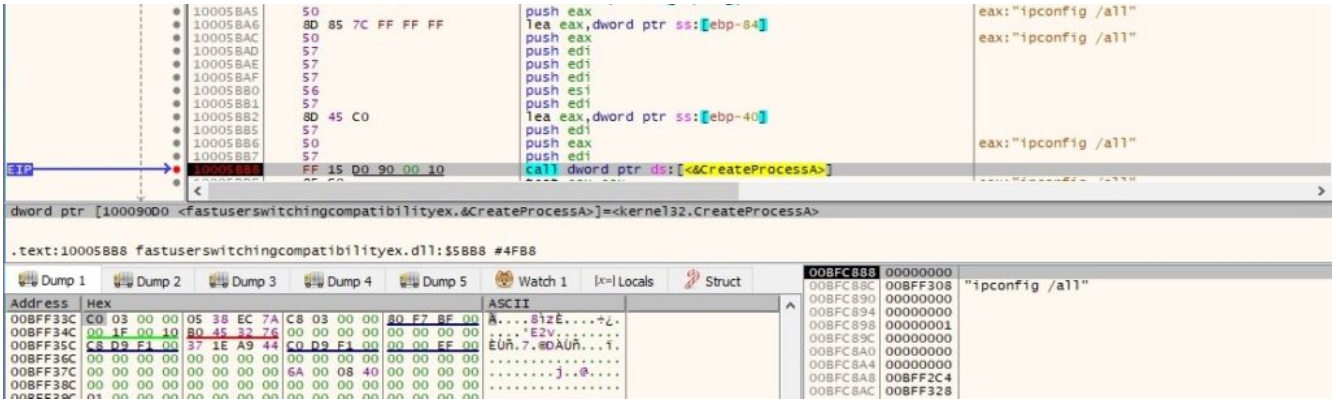


Figure 61

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

```
NUL
Windows IP Configuration

   Host Name . . . . . . . . . . . . : DESKTOP-2[redacted]
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : localdomain

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection
   Physical Address. . . . . . . . . : 00-0C-29-25-66-15
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::b1d6:3a45:ca5e:9bee%6(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.164.128(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Monday, November 9, 2020 5:16:45 AM
   Lease Expires . . . . . . . . . . : Monday, November 23, 2020 6:11:18 AM
   Default Gateway . . . . . . . . . :
   DHCP Server . . . . . . . . . . . : 192.168.164.254
   DHCPv6 IAID . . . . . . . . . . . : 33557545
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-26-A2-5A-CA-00-0C-29-25-66-15
   DNS Servers . . . . . . . . . . . : 192.168.164.1
   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:

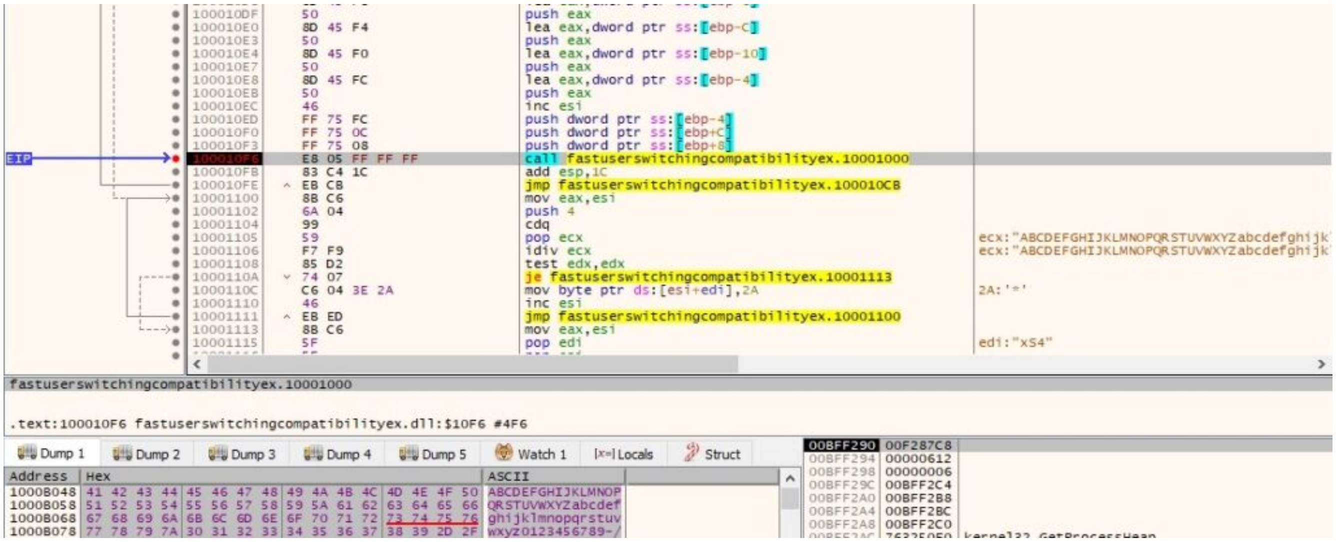


Figure 63

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”:

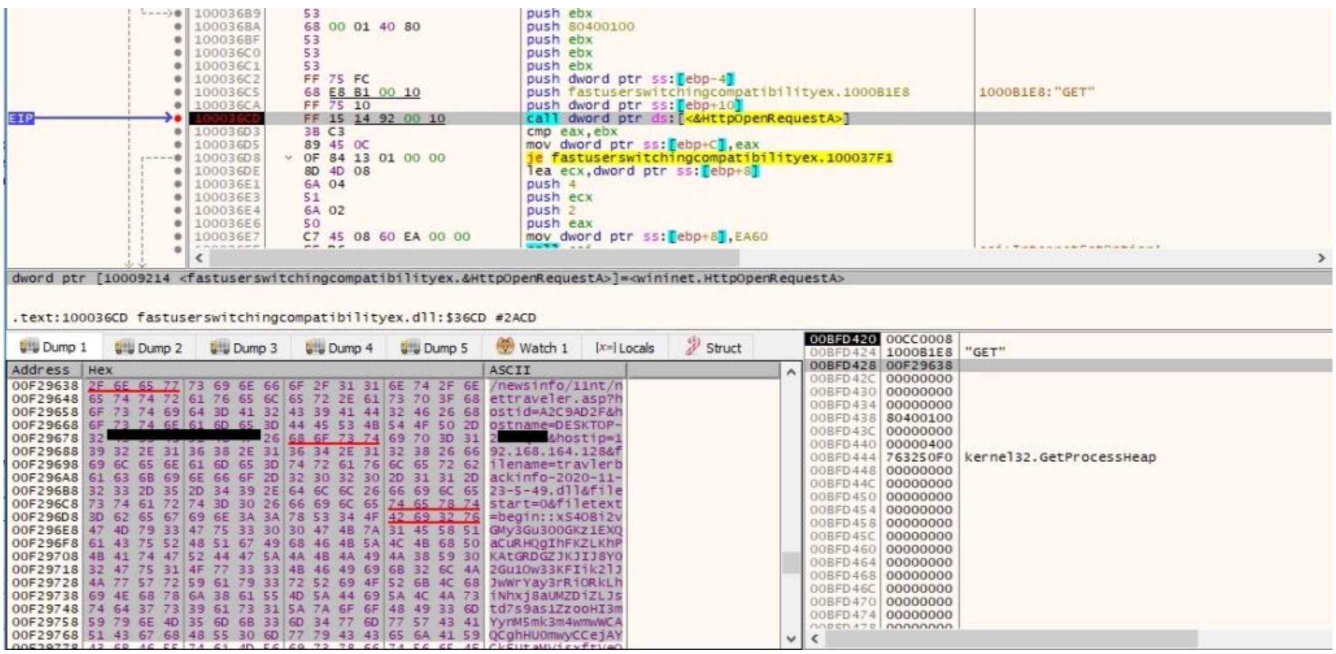


Figure 64

If the server response contains “Success:”, the exfiltration was successful. The malicious process 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:

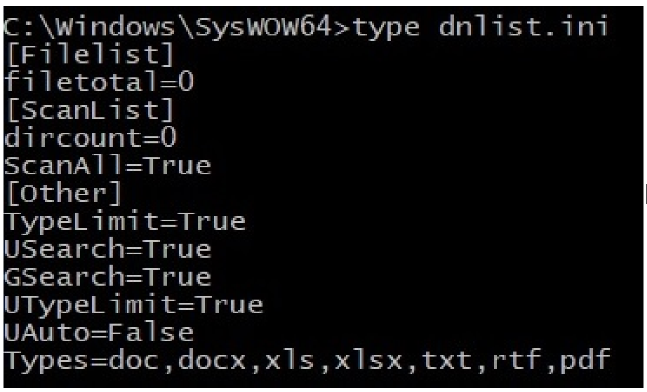


Figure 67

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:

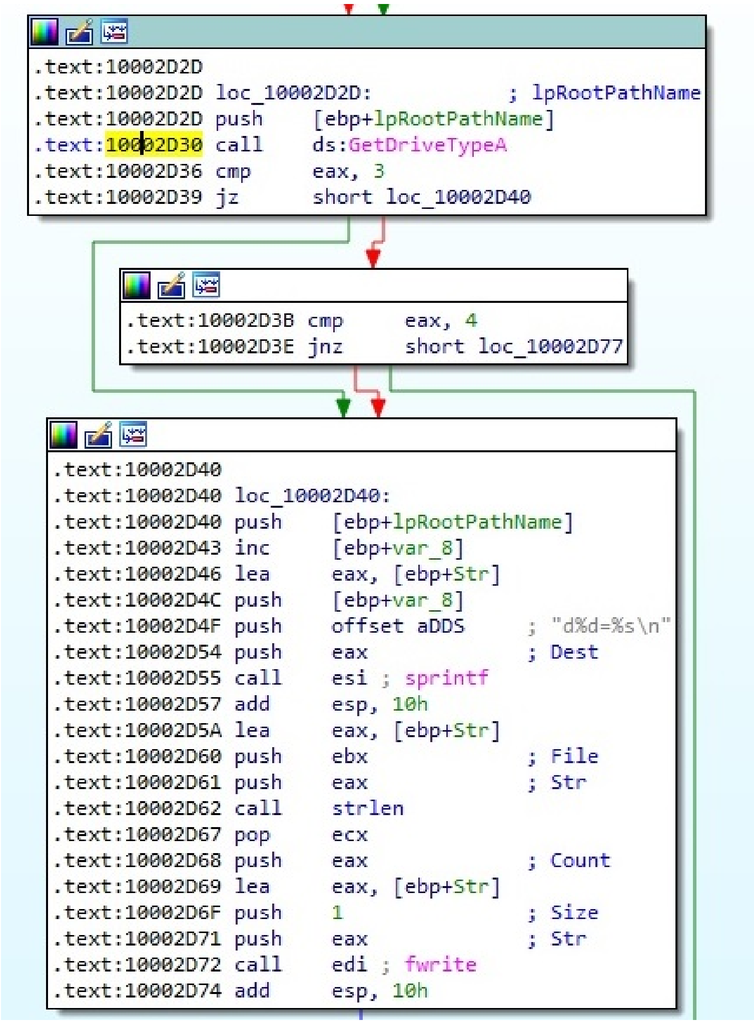


Figure 68

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

```
[Computer]
Name=DESKTOP-2[redacted]
Page=1252
[DESKTOP-2[redacted]]
d1=C:\
dircount=1
[C:\]
d1=$Recycle.Bin
f1=$WINRE_BACKUP_PARTITION.MARKER
d2=706d7fdb2ede4d8e91
d3=Boot
f2=bootmgr
f3=BOOTNXT
f4=BOOTSECT.BAK
d4=Documents and Settings
d5=EFS Software
f5=eula.1028.txt
f6=eula.1031.txt
f7=eula.1033.txt
f8=eula.1036.txt
f9=eula.1040.txt
f10=eula.1041.txt
f11=eula.1042.txt
f12=eula.2052.txt
f13=eula.3082.txt
f14=globdata.ini
f15=install.exe
f16=install.ini
f17=install.res.1028.dll
f18=install.res.1031.dll
f19=install.res.1033.dll
f20=install.res.1036.dll
f21=install.res.1040.dll
f22=install.res.1041.dll
f23=install.res.1042.dll
f24=install.res.2052.dll
f25=install.res.3082.dll
d6=logs
f26=pagefile.sys
d7=PerfLogs
d8=Program Files
d9=Program Files (x86)
d10=ProgramData
d11=Python27
d12=python27-x64
d13=Recovery
f27=StarBurn.log
d14=Strawberry
f28=swapfile.sys
d15=System Volume Information
d16=temp
d17=Users
f29=vcredist.bmp
f30=VC_RED.cab
f31=VC_RED.MSI
d18=Windows
dircount=18
filecount=31
```

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”:

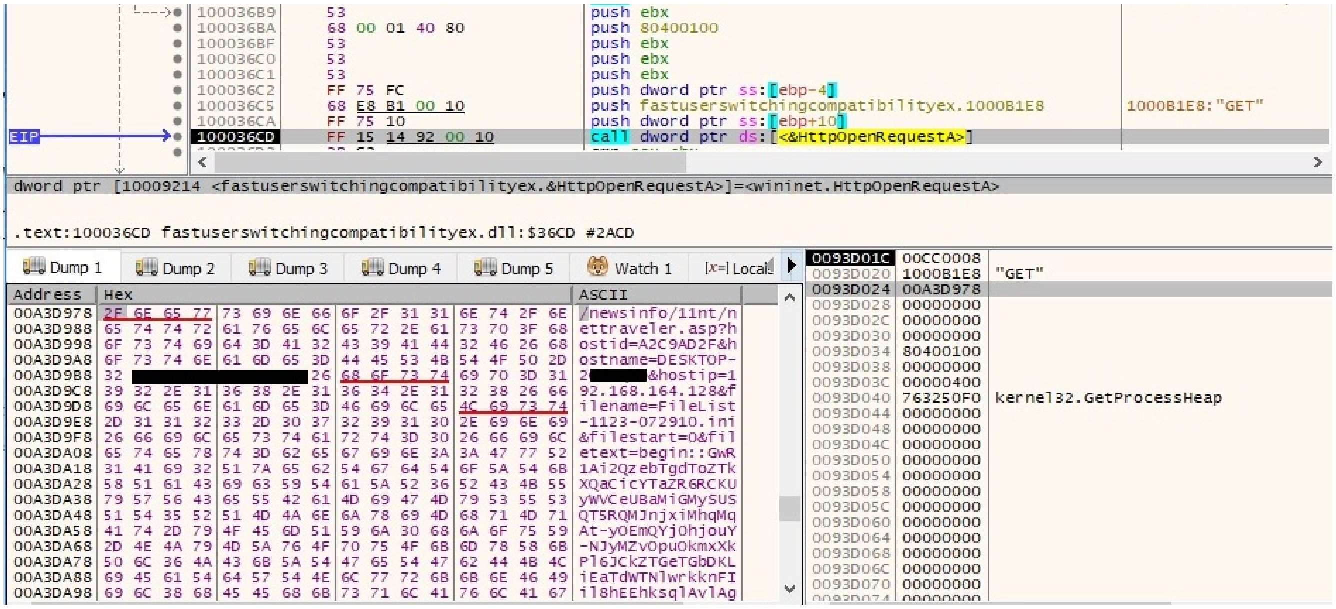


Figure 70

The server response is expected to contain “Success:”. The attacker is interested in the following types of files: .doc, .docx, .xls, .xlsx, .txt, .rtf, .pdf (Types parameter from dnlist.ini file). The malicious process tries to open uenumfs.ini which doesn’t exist at this moment, and then enumerates the files found in “C:\User<Username>\AppData\Local\Temp\ntvba00.tmp\”. This specific directory will be created by Thread2 and will contain all files which have been selected to be exfiltrated to the C2 server:

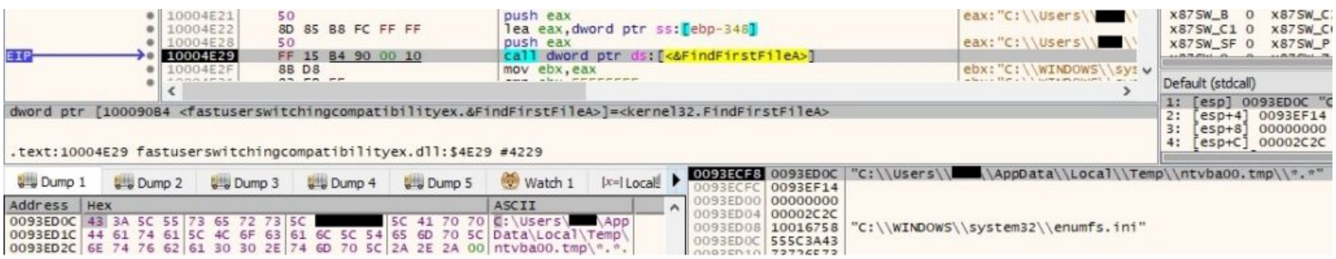


Figure 71

Now the process contacts the C2 server again with the parameter “action=getdata”. It expects one of the following responses: “A2C9AD2F:UNINSTALL”, “A2C9AD2F:UPDATE”, “A2C9AD2F:RESET” or “A2C9AD2F:UPLOAD” (note that “A2C9AD2F” is the volume serial number extracted a while ago):

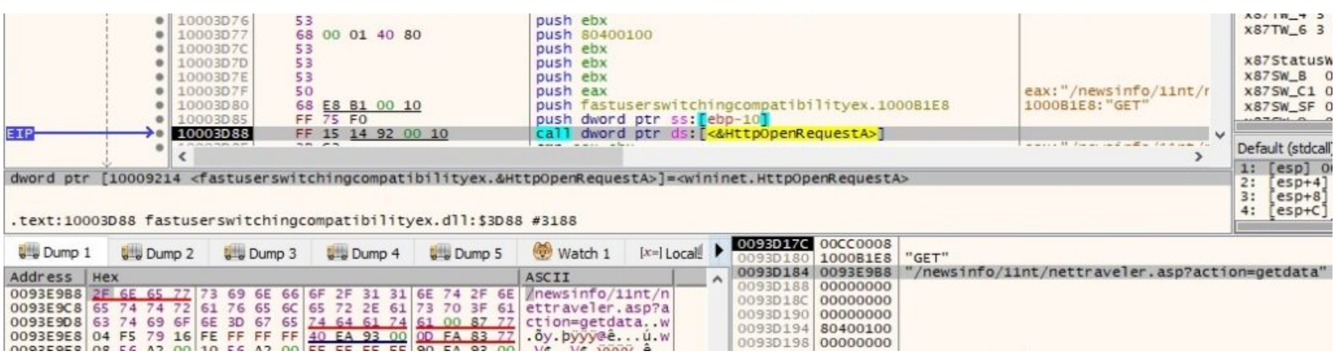


Figure 72

Case 1: (UNINSTALL)

The following registry keys are deleted using RegDeleteKeyA API:
“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”. Also the process deletes enumfs.ini, dnlist.ini, “C:\WINDOWS\system32\udidx.ini”, uenumfs.ini and stat_t.ini. One such call is displayed below:

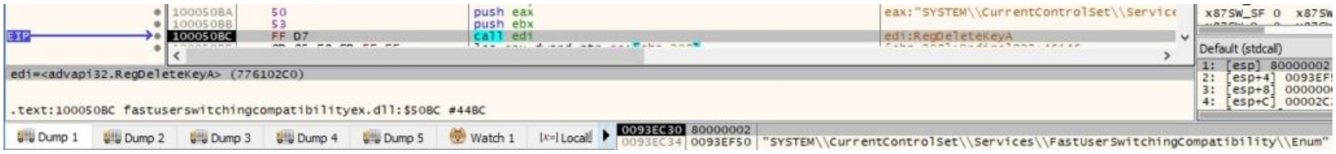


Figure 73

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

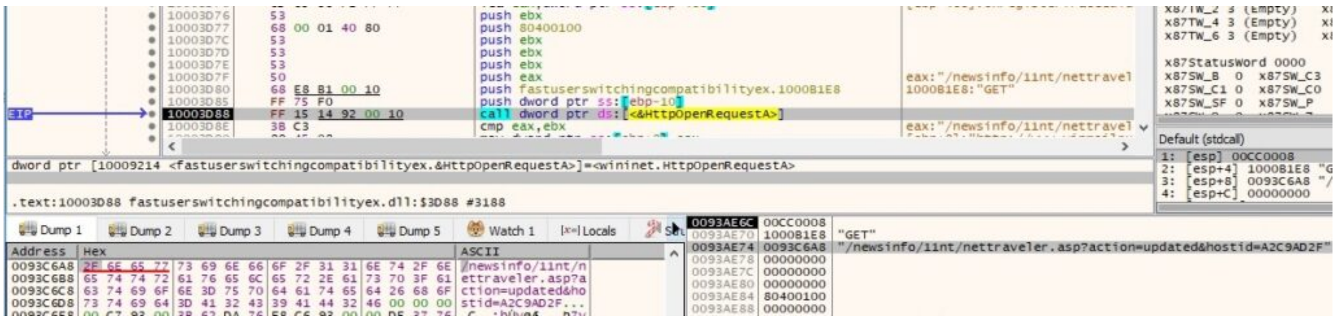


Figure 74

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=datasize” parameter and the HTTP response is supposed to include “Success:” if everything works smoothly:

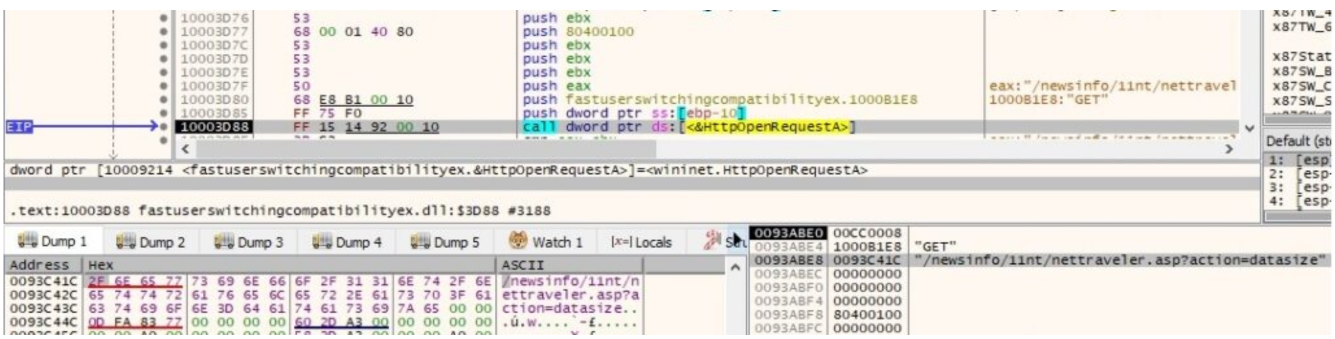


Figure 75

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):

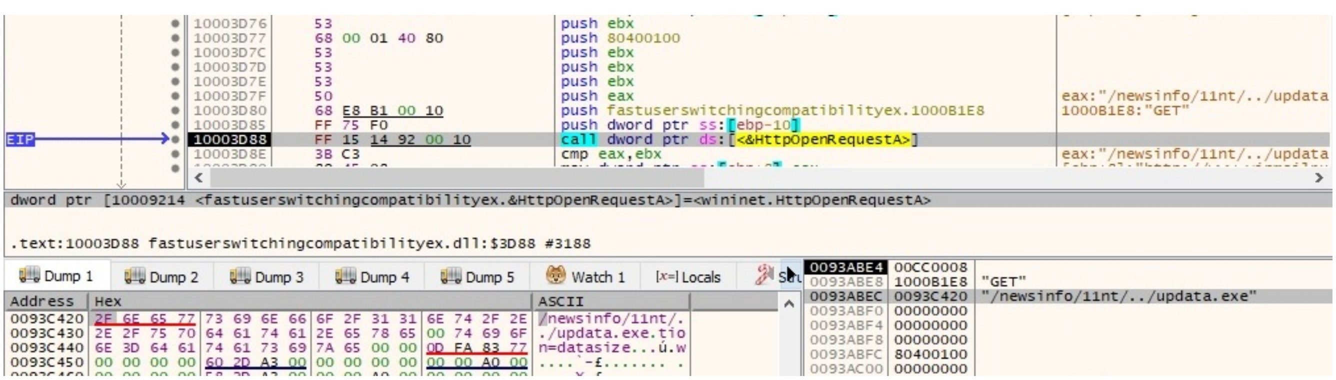


Figure 76

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):

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:

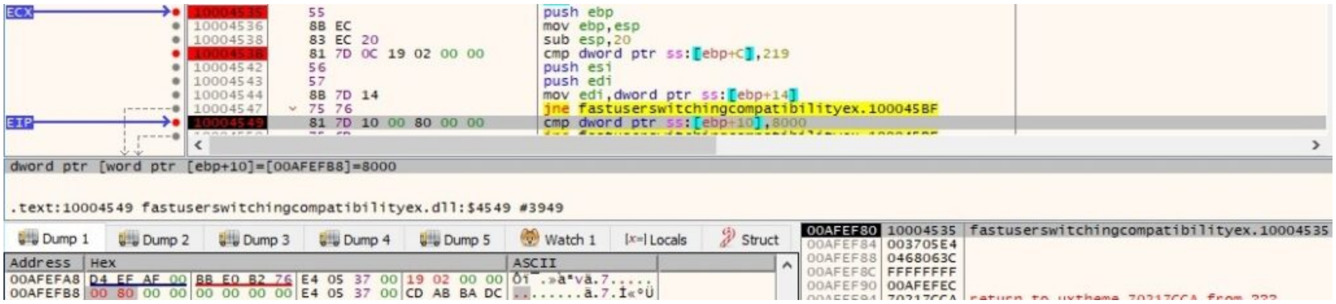


Figure 79

“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):

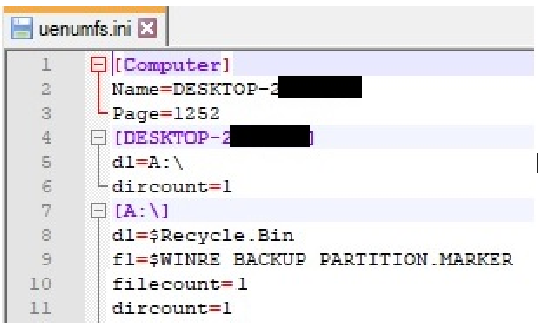


Figure 80

As in the first case, this search will apply for every directory found on the drive, recursively. The process creates a “C:\Users\<Username>\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):

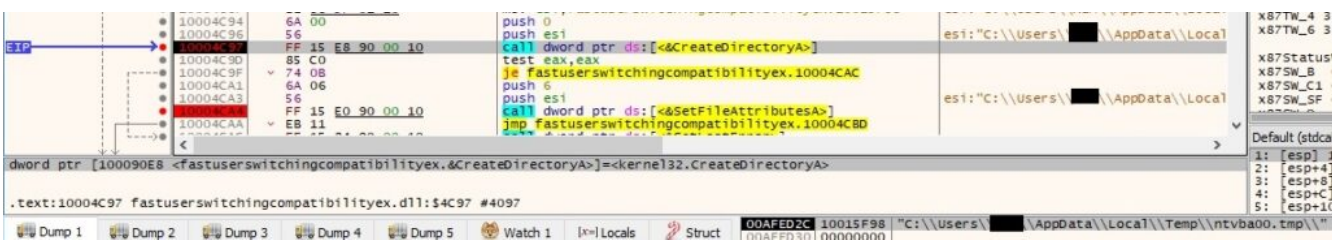


Figure 81

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:

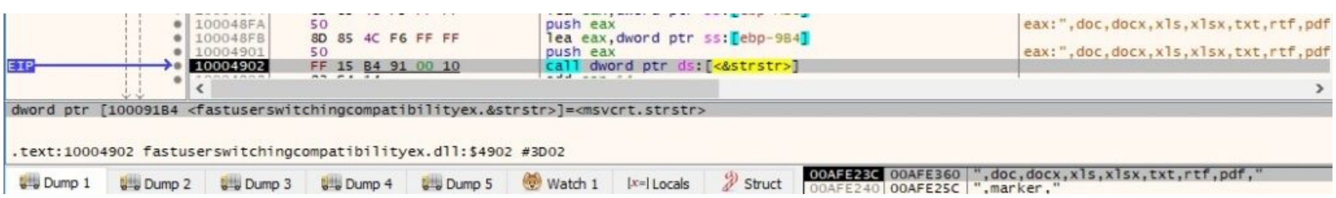


Figure 82

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):

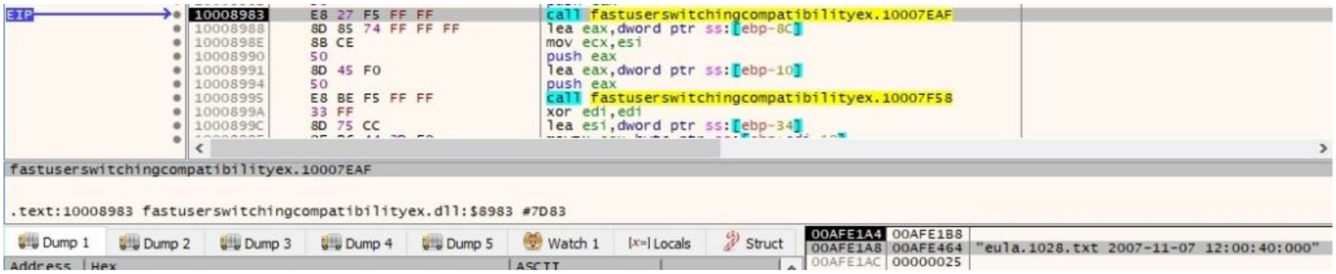
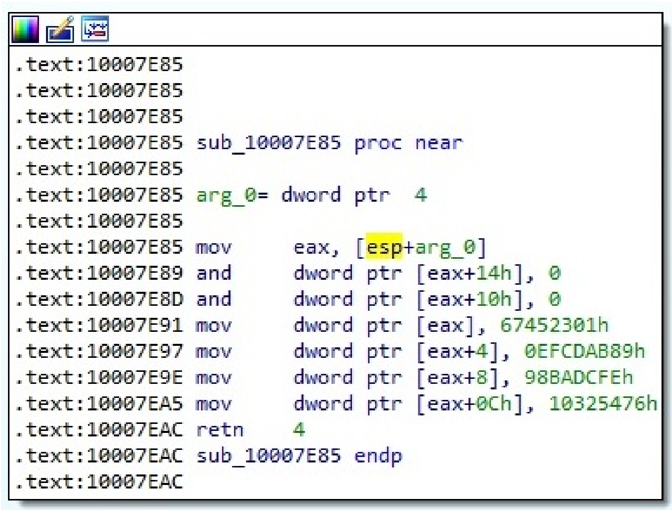


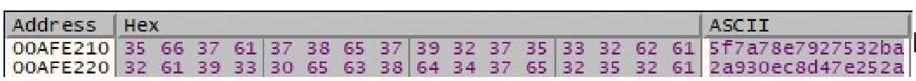
Figure 83



```
.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], 0EFCDAB89h
.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 37 65 32 35 32 61	2a930ec8d47e252a

Figure 85

Now "C:\eula.1028.txt" is copied to "C:\Users\<Username>\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:

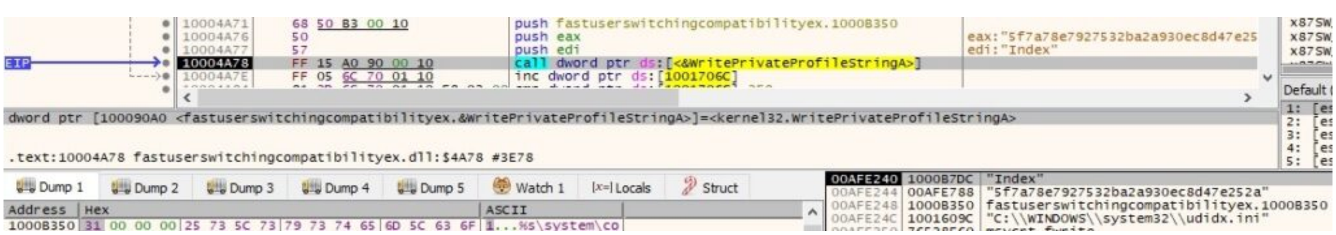


Figure 87

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

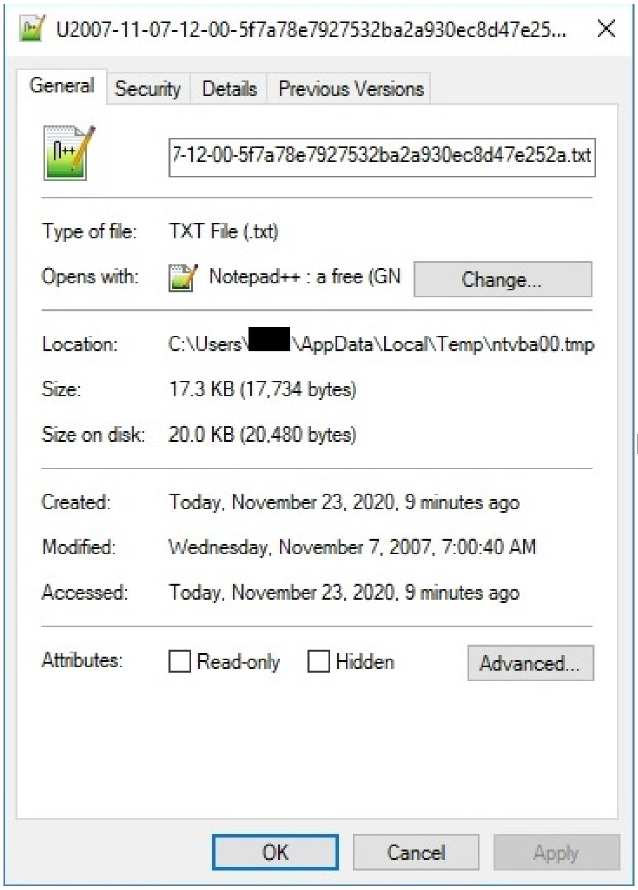


Figure 88

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):

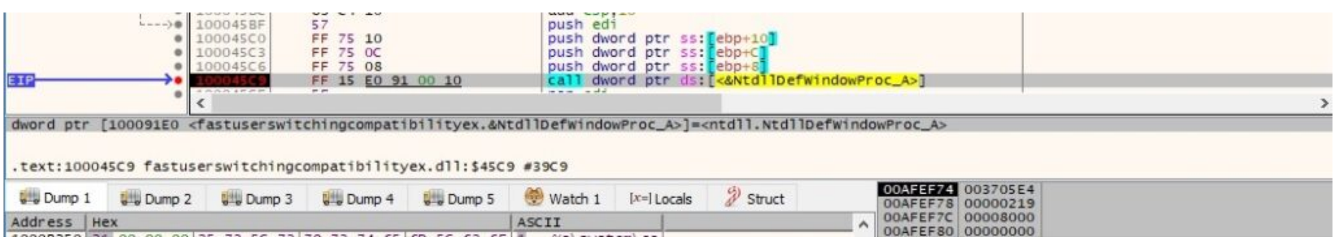


Figure 90

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