A step-by-step analysis of the new malware used by APT28/Sofacy called SkinnyBoy – CYBER GEEKS

Published: 2021-08-03 · Archived: 2026-04-05 16:48:19 UTC

Summary

The malware extracts configuration information about the machine that it infects using the systeminfo command, and then it retrieves the list of processes by spawning a tasklist process. The content of the following directories, along with the processes' output, is base64-encoded and exfiltrated to the C2 server updaterweb[.]com:

- Desktop folder
- C:\Program Files
- C:\Program Files (x86)
- C:\Users\<User>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Administrative Tools
- C:\Users\<User>\AppData\Roaming
- C:\Users\<User>\AppData\Roaming\Microsoft\Windows\Templates
- C:\WINDOWS
- C:\Users\<User>\AppData\Local\Temp

The user agent used during the network communication is set to "Opera", and the following is the structure of the POST request: "id=<hostname>#Username#<Serial number in decimal>¤t=1&total=1&data=<data to be exfiltrated>". The "cmd=y" command is used to download a DLL file from the C2 server, which is loaded using the LoadLibraryW API, and the first ordinal function is executed.

Analyst: @GeeksCyber

Technical analysis

SHA256: ae0bc3358fef0ca2a103e694aa556f55a3fed4e98ba57d16f5ae7ad4ad583698

The DLL has 2 exports (DllEntryPoint and RunMod). We have used rundll32.exe to run the DLL by calling the RunMod function:

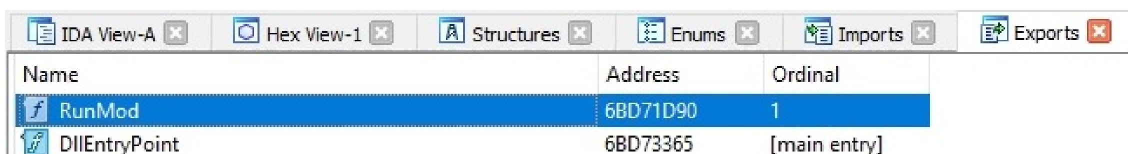

Figure 1

The malware creates an unnamed event object by calling the CreateEventW API:

Figure 2

Two new threads are created by the process using the CreateThread function:

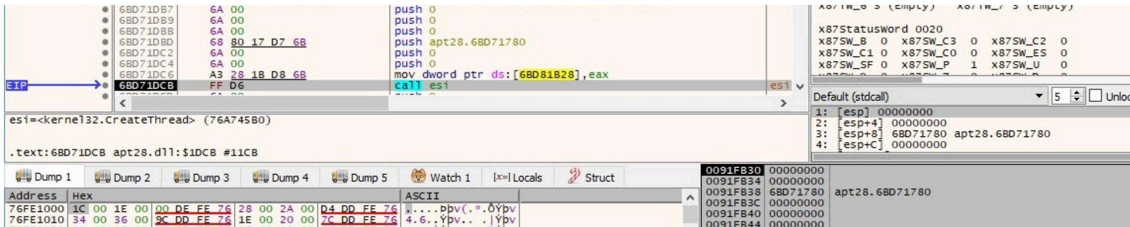

Figure 3

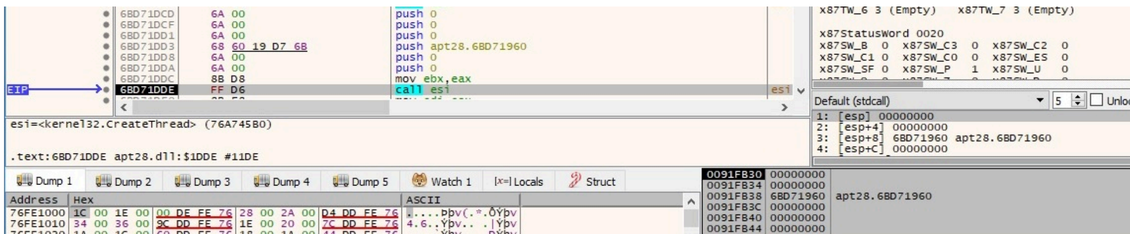

Figure 4

The GetMessage routine is utilized to retrieve a message from the thread's message queue:

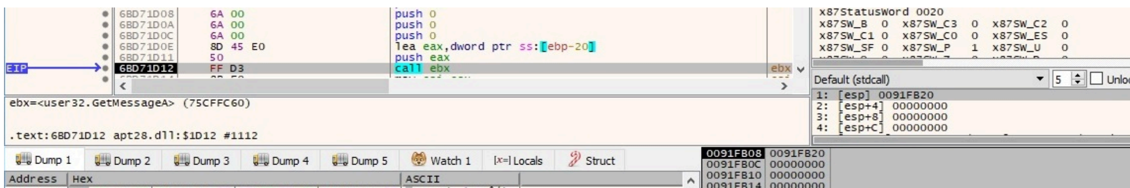

Figure 5

The malicious process enumerates all the messages, and it breaks the loop if the message is equal to 0x16 (**WM_ENDSESSION** – inform the application whether the session is ending):

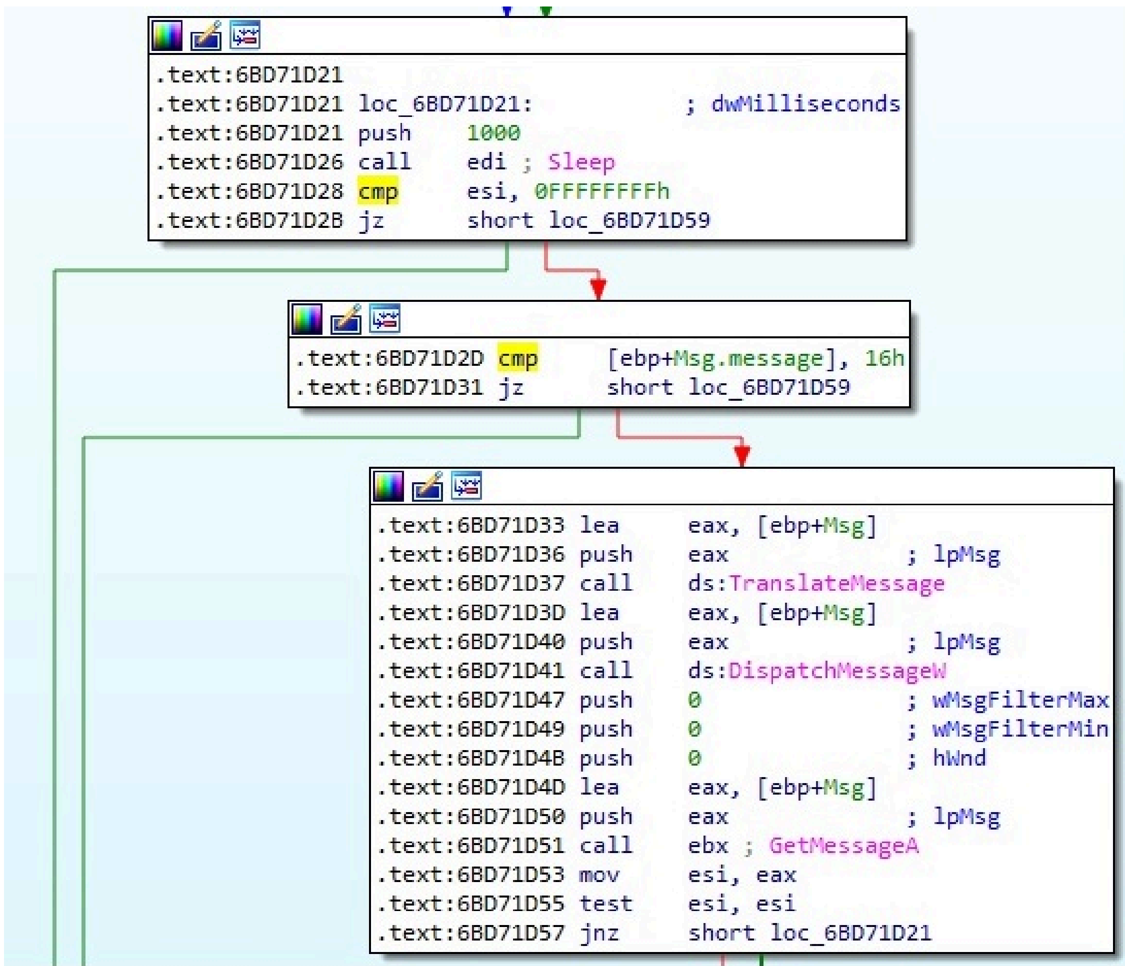

Figure 6

Thread activity – StartAddress function

The malware creates an anonymous pipe using the CreatePipe API:

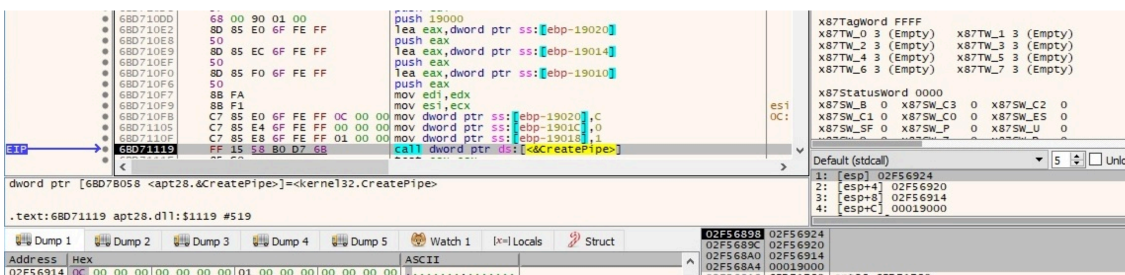

Figure 7

GetStartupInfoA is used to retrieve the content of the STARTUPINFO structure from when the calling process was created:

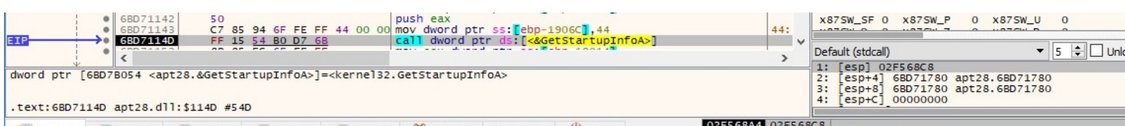

Figure 8

The binary creates a new process that runs the systeminfo command, which displays configuration information about the computer and its OS:

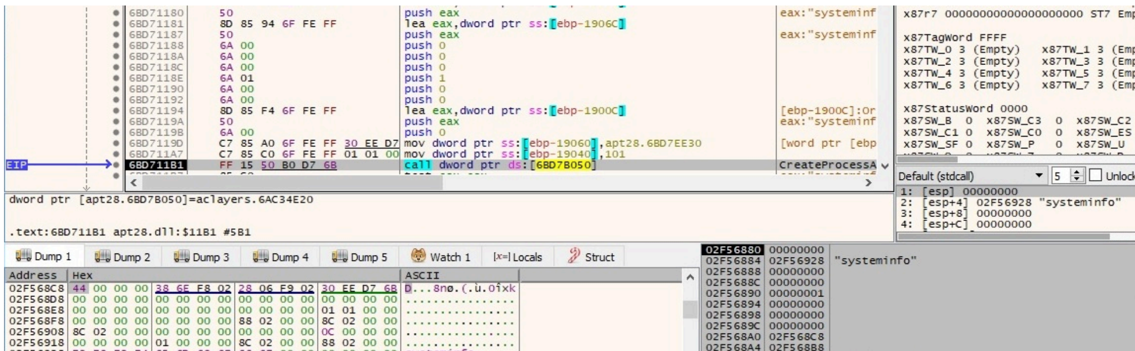

Figure 9

The pipe created earlier is used as an inter-process communication mechanism. The output of the systeminfo command is read via a ReadFile function call:

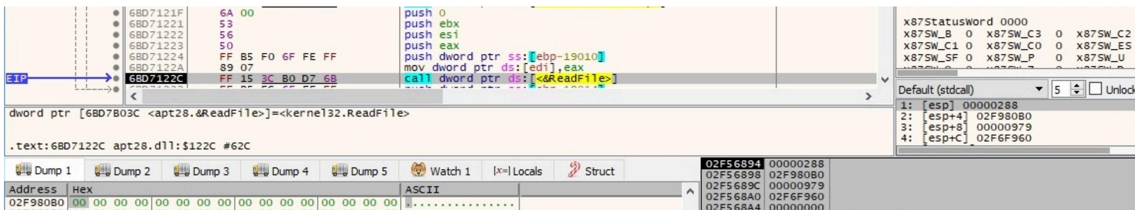

Figure 10

Address	Hex	ASCII
02F980B0	0D 0A 48 6F 73 74 20 4E 61 6D 65 3A 20 20 20 20	..Host Name:
02F980C0	20 20 20 20 20 20 20 20 20 20 20 20 20 44 45 53	DES
02F980D0	4B 54 4F 50 2D 32 43 [redacted] 0D 0A 4F 53	KTOP-2C[redacted]..OS
02F980E0	20 4E 61 6D 65 3A 20 20 20 20 20 20 20 20 20 20	Name:
02F980F0	20 20 20 20 20 20 20 20 20 4D 69 63 72 6F 73 6F	Microso
02F98100	66 74 20 57 69 6E 64 6F 77 73 20 31 30 20 45 6E	ft Windows 10 En
02F98110	74 65 72 70 72 69 73 65 0D 0A 4F 53 20 56 65 72	terprise..OS Ver
02F98120	73 69 6F 6E 3A 20 20 20 20 20 20 20 20 20 20 20	sion:
02F98130	20 20 20 20 20 31 30 2E 30 2E 31 36 32 39 39 20	10.0.16299
02F98140	4E 2F 41 20 42 75 69 6C 64 20 31 36 32 39 39 0D	N/A Build 16299.
02F98150	0A 4F 53 20 4D 61 6E 75 66 61 63 74 75 72 65 72	.OS Manufacturer
02F98160	3A 20 20 20 20 20 20 20 20 20 20 20 4D 69 63 72	: Micr
02F98170	6F 73 6F 66 74 20 43 6F 72 70 6F 72 61 74 69 6F	osoft Corporatio
02F98180	6E 0D 0A 4F 53 20 43 6F 6E 66 69 67 75 72 61 74	n..OS Configurat
02F98190	69 6F 6E 3A 20 20 20 20 20 20 20 20 20 20 53 74	ion: St
02F981A0	61 6E 64 61 6C 6F 6E 65 20 57 6F 72 6B 73 74 61	andalone Worksta
02F981B0	74 69 6F 6E 0D 0A 4F 53 20 42 75 69 6C 64 20 54	tion..OS Build T
02F981C0	79 70 65 3A 20 20 20 20 20 20 20 20 20 20 20 20	ype:
02F981D0	20 4D 75 6C 74 69 70 72 6F 63 65 73 73 6F 72 20	Multiprocessor

Figure 11

The list of processes is retrieved by creating a new process that runs the tasklist command:

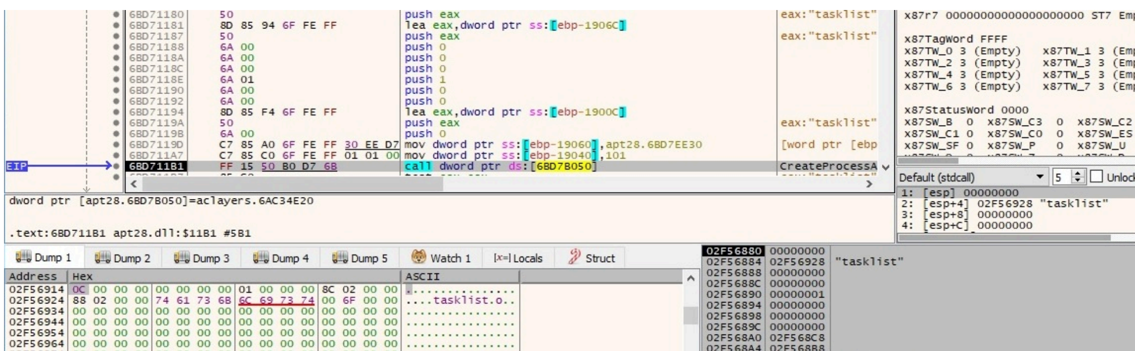

Figure 12

The output of the tasklist command is transmitted to the main process using the ReadFile API:

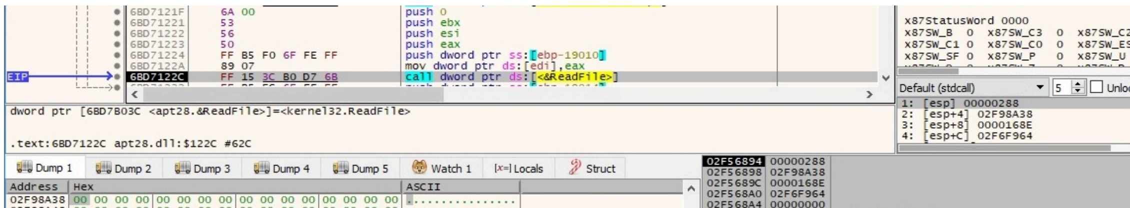

Figure 13

```
Address   Hex                                              ASCII
02F98A38  0D 0A 49 6D 61 67 65 20 4E 61 6D 65 20 20 20 20  ..Image Name
02F98A48  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
02F98A58  20 50 49 44 20 53 65 73 73 69 6F 6E 20 4E 61 6D   PID Session Nam
02F98A68  65 20 20 20 20 20 20 20 20 53 65 73 73 69 6F 6E  e        Session
02F98A78  23 20 20 20 20 4D 65 6D 20 55 73 61 67 65 0D 0A  #    Mem Usage..
02F98A88  3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D  ================
02F98A98  3D 3D 3D 3D 3D 3D 3D 3D 3D 20 3D 3D 3D 3D 3D 3D  ========= ======
02F98AA8  3D 3D 20 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D  == ==============
02F98AB8  3D 3D 3D 20 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 20  === ============
02F98AC8  3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 0D 0A 53 79  ============..Sy
02F98AD8  73 74 65 6D 20 49 64 6C 65 20 50 72 6F 63 65 73  stem Idle Proces
02F98AE8  73 20 20 20 20 20 20 20 20 20 20 20 20 20 20 30  s              0
02F98AF8  20 53 65 72 76 69 63 65 73 20 20 20 20 20 20 20   Services
02F98B08  20 20 20 20 20 20 20 20 20 20 20 20 30 20 20 20              0
02F98B18  20 20 20 20 20 20 20 38 20 4B 0D 0A 53 79 73 74         8 K..Syst
02F98B28  65 6D 20 20 20 20 20 20 20 20 20 20 20 20 20 20  em
02F98B38  20 20 20 20 20 20 20 20 20 20 20 20 20 34 20 53               4 S
02F98B48  65 72 76 69 63 65 73 20 20 20 20 20 20 20 20 20  ervices
02F98B58  20 20 20 20 20 20 20 20 20 20 30 20 20 20 20 20            0
```

Figure 14

The binary gets the path of the Desktop folder using the SHGetFolderPathW routine:

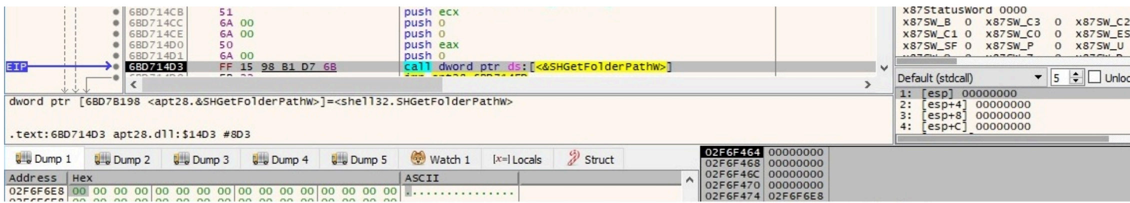

Figure 15

The process enumerates the files/directories from the Desktop directory using the FindFirstFileW and FindNextFileW functions:

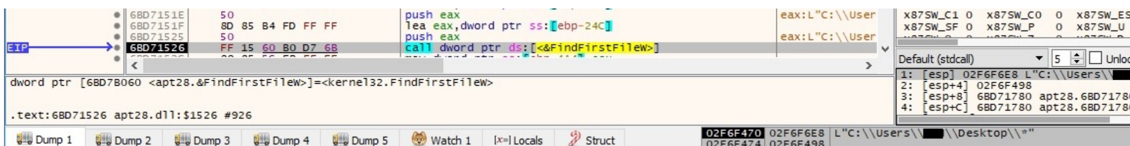

Figure 16

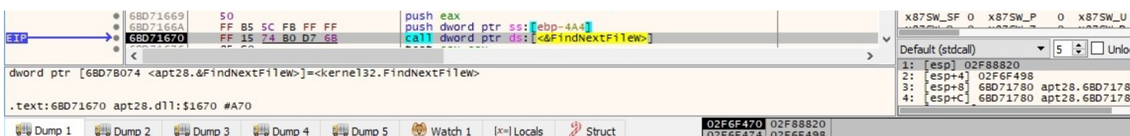

Figure 17

The binary adds 18 characters of “#” before and after the folder name, as following:

Address	Hex	ASCII
02F97880	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F97890	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F978A0	23 00 23 00 43 00 3A 00 5C 00 55 00 73 00 65 00	#.#.C.:.\.U.s.e.
02F978B0	72 00 73 00 5C 00 [redacted] 00 5C 00 44 00	r.s.\[redacted].\.D.
02F978C0	65 00 73 00 6B 00 74 00 6F 00 70 00 5C 00 2A 00	e.s.k.t.o.p.\.*.
02F978D0	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F978E0	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F978F0	23 00 23 00 0D 00 0A 00 0D 00 0A 00 00 00 00 00	#.#.............

Figure 18

The list of files and directories extracted before is concatenated with the above string, as shown in figure 19:

Address	Hex	ASCII
02F971A8	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F971B8	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F971C8	23 00 23 00 43 00 3A 00 5C 00 55 00 73 00 65 00	#.#.C.:.\.U.s.e.
02F971D8	72 00 73 00 5C 00 [redacted] 00 5C 00 44 00	r.s.\[redacted].\.D.
02F971E8	65 00 73 00 6B 00 74 00 6F 00 70 00 5C 00 2A 00	e.s.k.t.o.p.\.*.
02F971F8	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F97208	23 00 23 00 23 00 23 00 23 00 23 00 23 00 23 00	#.#.#.#.#.#.#.#.
02F97218	23 00 23 00 0D 00 0A 00 0D 00 0A 00 30 00 61 00	#.#.......0.a.
02F97228	30 00 63 00 32 00 32 00 35 00 66 00 30 00 65 00	0.c.2.2.5.f.0.e.
02F97238	35 00 65 00 65 00 39 00 34 00 31 00 61 00 37 00	5.e.e.9.4.1.a.7.
02F97248	39 00 66 00 32 00 62 00 37 00 37 00 30 00 31 00	9.f.2.b.7.7.0.1.
02F97258	66 00 31 00 32 00 38 00 35 00 65 00 34 00 39 00	f.1.2.8.5.e.4.9.
02F97268	37 00 35 00 61 00 32 00 38 00 35 00 39 00 65 00	7.5.a.2.8.5.9.e.
02F97278	62 00 34 00 64 00 30 00 32 00 35 00 64 00 39 00	b.4.d.0.2.5.d.9.
02F97288	36 00 64 00 39 00 65 00 33 00 36 00 36 00 65 00	6.d.9.e.3.6.6.e.
02F97298	38 00 31 00 61 00 62 00 62 00 39 00 2E 00 62 00	8.1.a.b.b.9...b.
02F972A8	69 00 6E 00 2E 00 73 00 61 00 6D 00 70 00 6C 00	i.n...s.a.m.p.l.
02F972B8	65 00 2E 00 67 00 7A 00 0D 00 0A 00 00 00 00 00	e...g.z.........

Figure 19

The following directories are also targeted by the backdoor: "C:\Program Files", "C:\Program Files (x86)", "C:\Users\<User>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Administrative Tools", "C:\Users\<User>\AppData\Roaming", "C:\Users\<User>\AppData\Roaming\Microsoft\Windows\Templates", "C:\WINDOWS" and "C:\Users\<User>\AppData\Local\Temp". The SHGetFolderPathW function is utilized to obtain some of these folder names (0x2a = **CSIDL_PROGRAM_FILESX86**, 0x30 = **CSIDL_ADMINTOOLS**, 0x1a = **CSIDL_APPDATA**, 0x15 = **CSIDL_TEMPLATES** and 0x24 = **CSIDL_WINDOWS**):

Figure 20

Figure 21

Figure 22

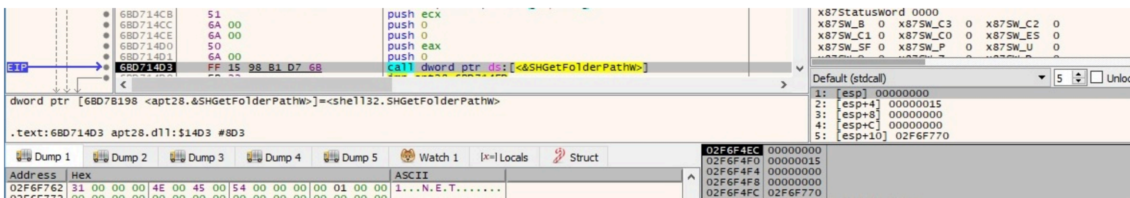

Figure 23

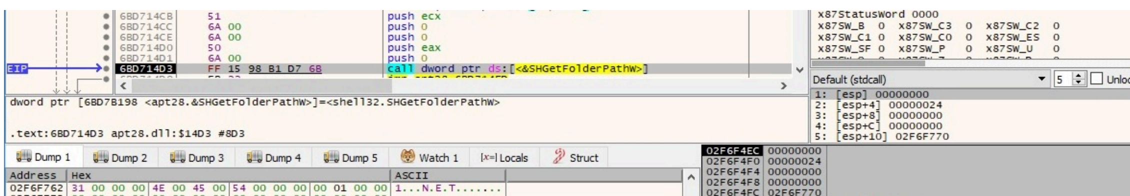

Figure 24

The GetTempPathW API is utilized to retrieve the path of the %TEMP% directory:

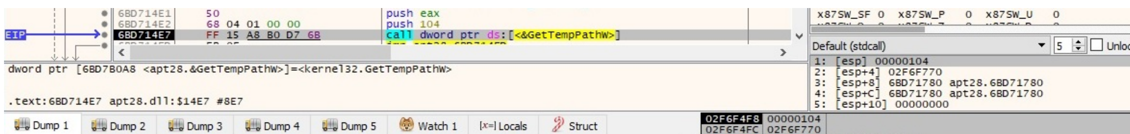

Figure 25

The file initializes the use of the WinINet functions using the InternetOpenW API (the user agent is hard-coded as "Opera"):

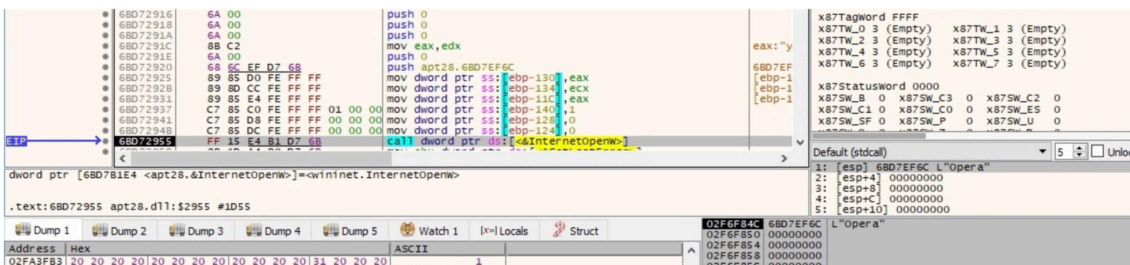

Figure 26

The send and receive timeouts are set to 600 seconds using the InternetSetOptionW routine (0x6 = **INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT** and 0x5 = **INTERNET_OPTION_CONTROL_SEND_TIMEOUT**):

Figure 27

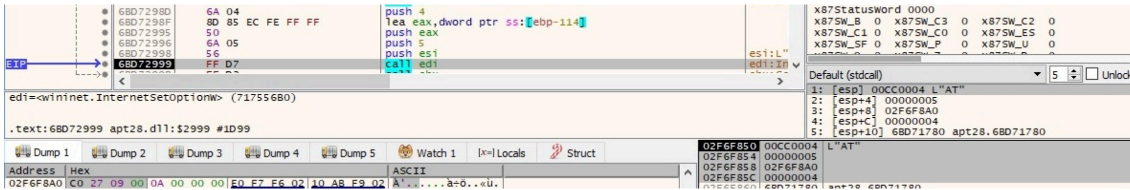

Figure 28

The malicious process establishes a connection to the C2 server updaterweb[.]com on port 443:

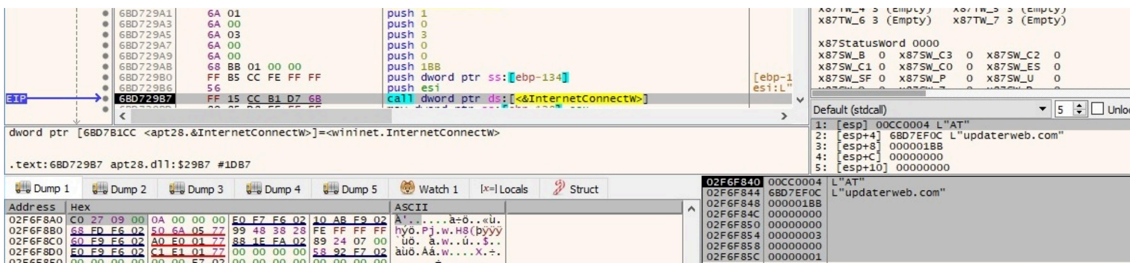

Figure 29

The NetBIOS name of the local computer is retrieved using the GetComputerNameA API:

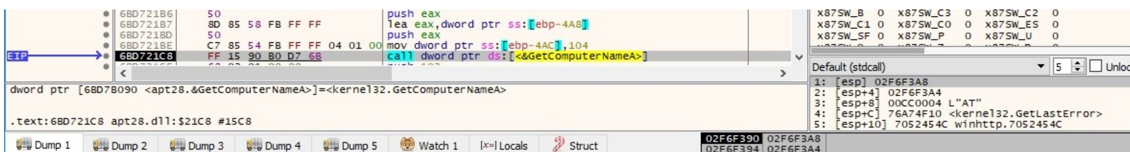

Figure 30

GetUserNameA is utilized to extract the name of the user associated with the current thread:

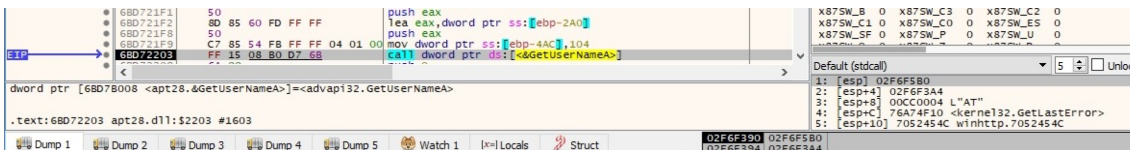

Figure 31

The malware extracts the volume serial number of the root of the current directory via a function call to GetVolumeInformationW:

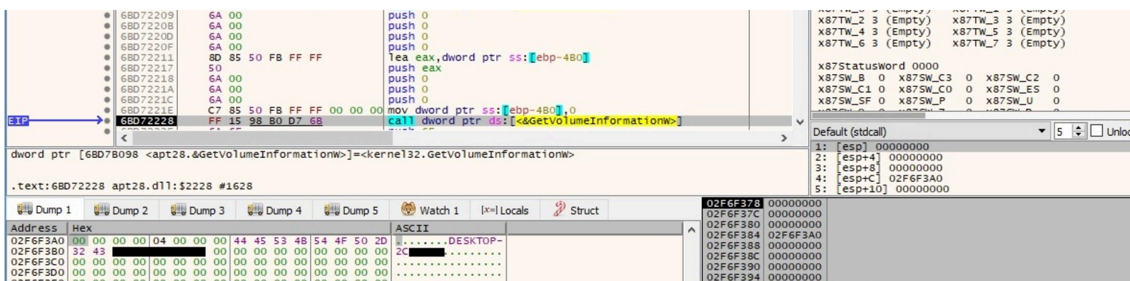

Figure 32

The process decrypts some important strings using the XOR algorithm, the keys being "CEJ&V%$84k839y92m" and "qpzoamxiendufbtbf3-#$*40fvnpwOPDwdkvn". The strings "id=%s#%s#%u&cmd=y" and "id=%s#%s#%u¤t=%s&total=%s&data=" have been computed:

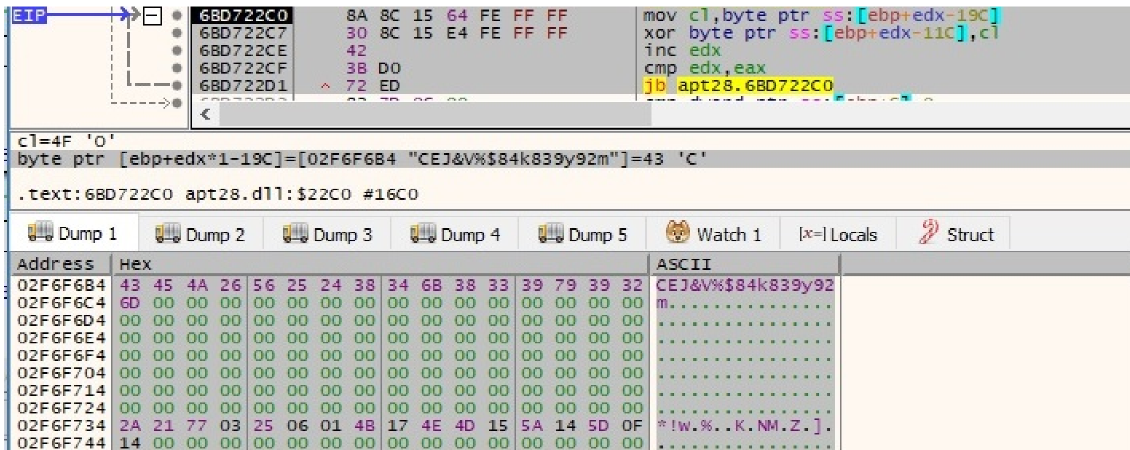

Figure 33

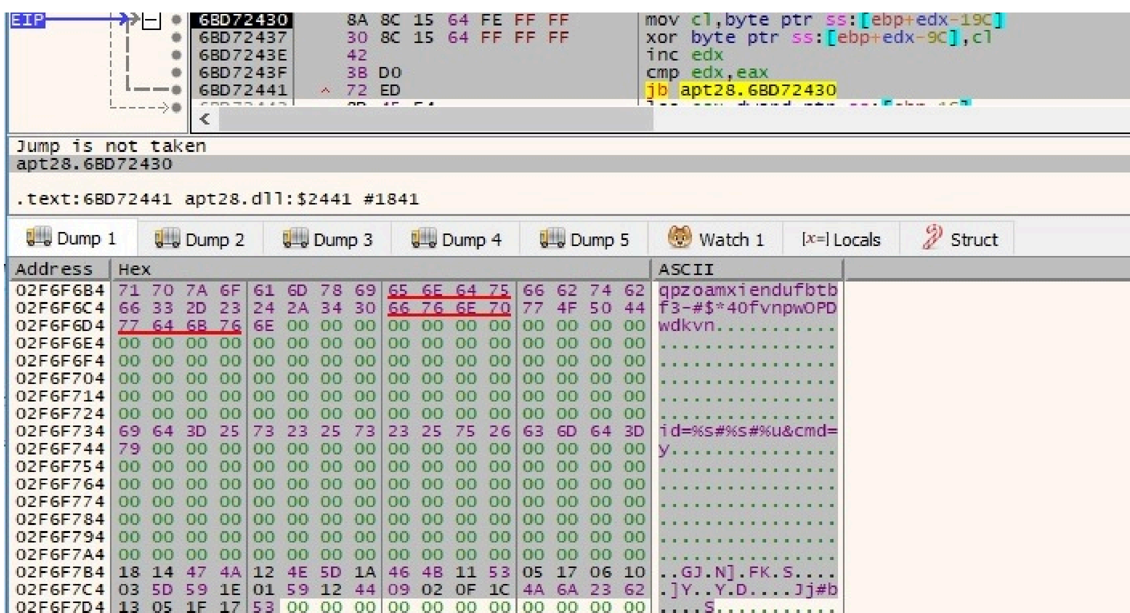

Figure 34

The output of the systeminfo command + output of the tasklist command + the list of targeted directories and their content are base-64 encoded using the CryptBinaryToStringA API (0x1 = **CRYPT_STRING_BASE64**):

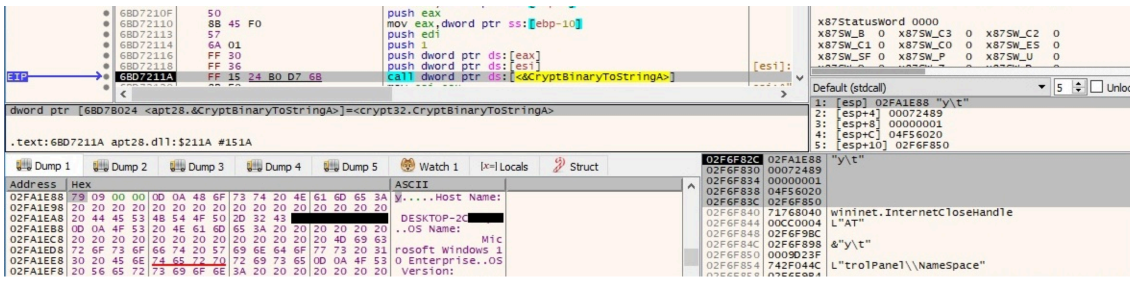

Figure 35

Figure 36

The HttpOpenRequestW routine is utilized to create an HTTP POST request handle:

Figure 37

The malware adds one HTTP request header ("application/x-www-form-urlencoded") to the HTTP request handle:

Figure 38

The request is sent to the HTTP server using the HttpSendRequestExW API, as displayed in figure 39:

Figure 39

In the case of failing to connect to the C2 server on port 443, the process tries to connect on port 80:

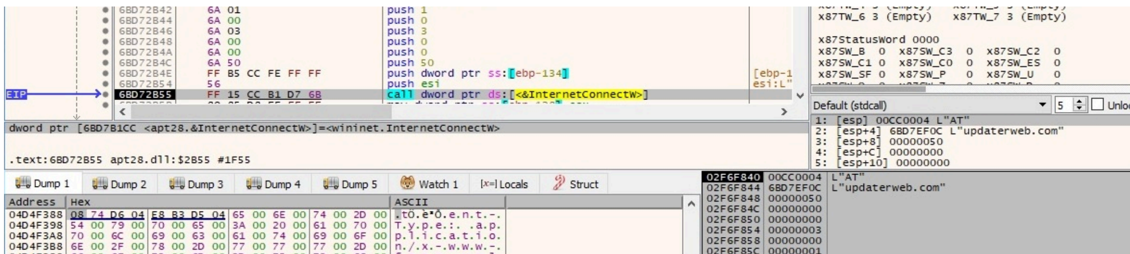

Figure 40

The information extracted before is exfiltrated to the C2 server (id=<hostname>#Username#<Serial number in decimal>¤t=1&total=1&data=<base-64 encoded data computed above>):

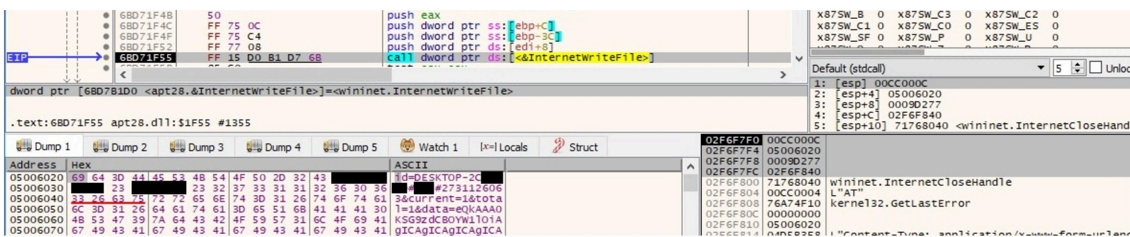

Figure 41

The thread sets the event created earlier to the signaled state:

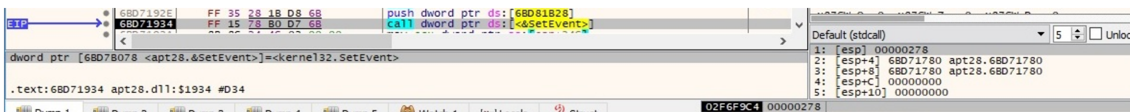

Figure 42

Thread activity – sub_6BD71960 function

This thread sets the event created earlier now to the nonsignaled state using the ResetEvent routine:

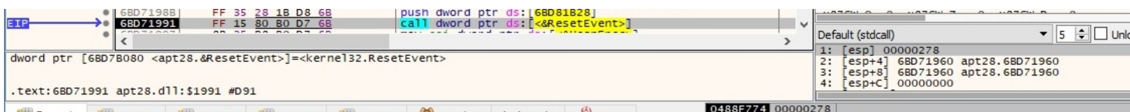

Figure 43

There is a similar workflow starting with calling the InternetOpenW function up until connecting to the C2 server on port 443 (or port 80 if the first one is unsuccessful). The POST request is different this time because it contains the “cmd=y” command that is used to download a DLL file:

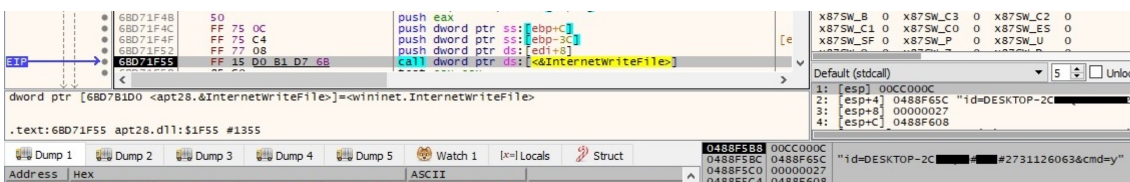

Figure 44

The malware queries the server to determine the amount of data available using the InternetQueryDataAvailable routine:

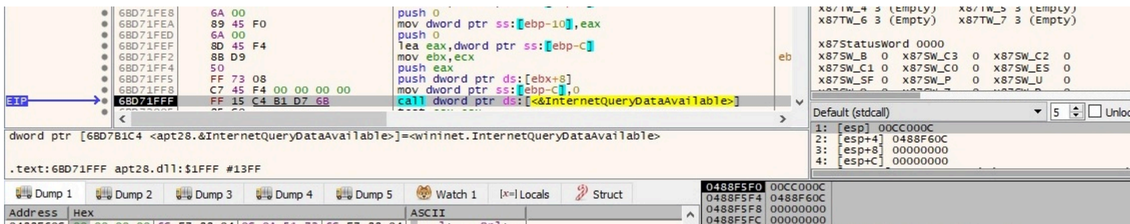

Figure 45

The potential DLL file is read from the handle using the InternetReadFile API (the first 4 bytes would represent the data size and there will also be 32 bytes that represent the SHA256 hash value of the content, as we'll describe in the upcoming paragraphs):

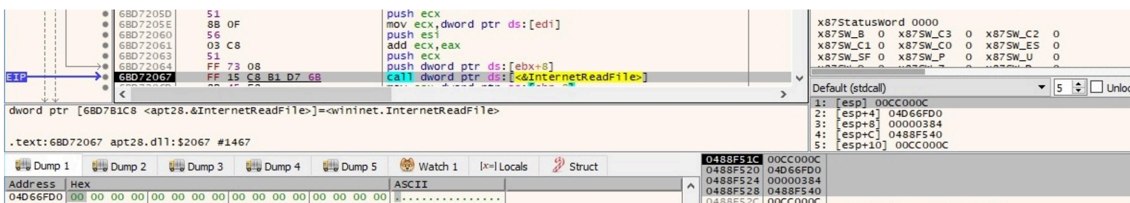

Figure 46

The expected DLL is base64-encoded because the process tries to decode it using the CryptStringToBinaryA function (0x1 = **CRYPT_STRING_BASE64**):

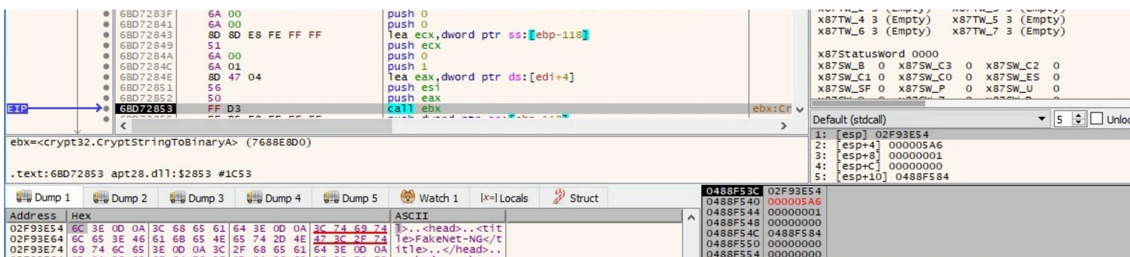

Figure 47

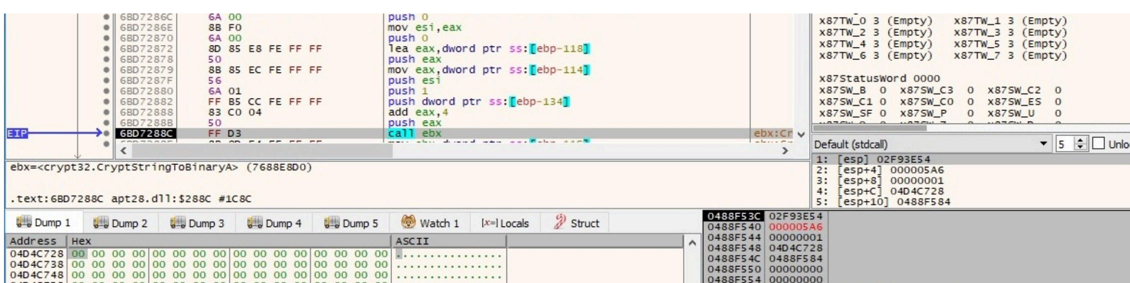

Figure 48

CryptAcquireContextA is utilized to acquire a handle to the Microsoft RSA and AES Cryptographic Provider (0x18 = **PROV_RSA_AES**):

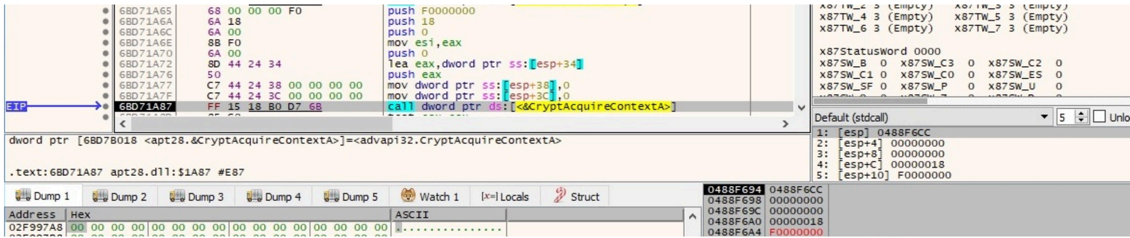

Figure 49

The CryptCreateHash routine is used to create a handle to a CSP (cryptographic service provider) hash object (0x800c = **CALG_SHA_256**):

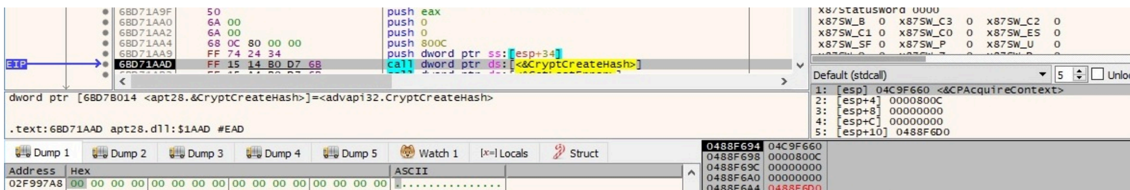

Figure 50

After the base64-encoded DLL file is decoded, then the malware hashes the buffer that is supposed to contain a DLL file using the SHA256 algorithm:

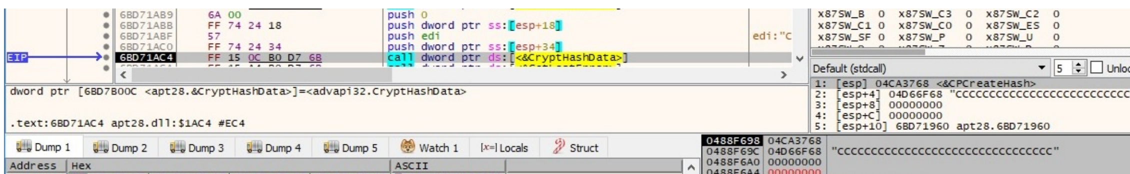

Figure 51

The hash value is extracted by calling the CryptGetHashParam API, as shown in figure 52 (0x2 = **HP_HASHVAL**):

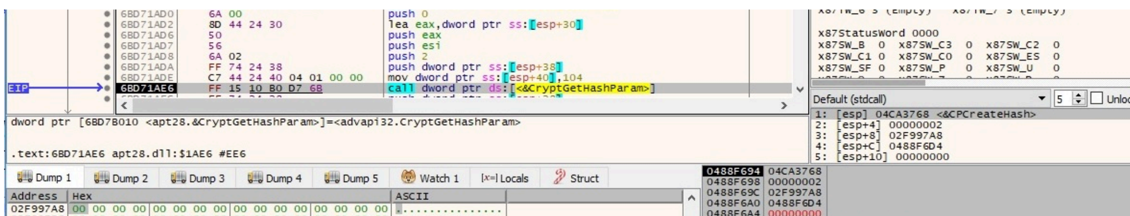

Figure 52

Address	Hex	ASCII
02F997A8	E3 B0 C4 42 98 FC 1C 14 9A FB F4 C8 99 6F B9 24	ã°ÄB.ü...ûôÈ.o¹$
02F997B8	27 AE 41 E4 64 9B 93 4C A4 95 99 1B 78 52 B8 55	'®Aäd..L¤...xR¸U

Figure 53

The malicious process verifies if the hash value computed above coincides with a 32-byte buffer that comes with the DLL file (of course that the response is emulated in our case, but we can adjust it to pass the comparison):

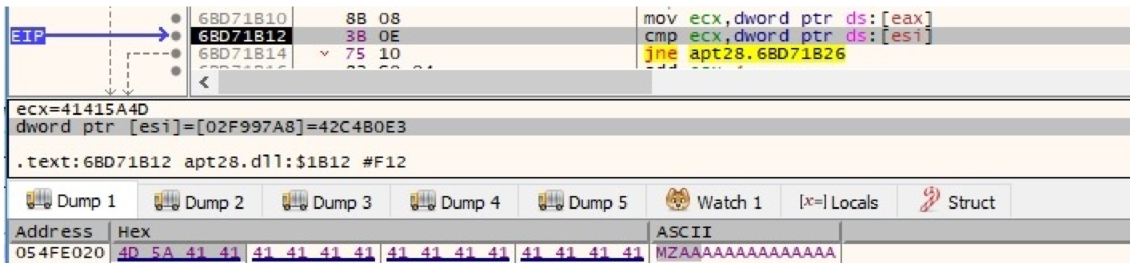

Figure 54

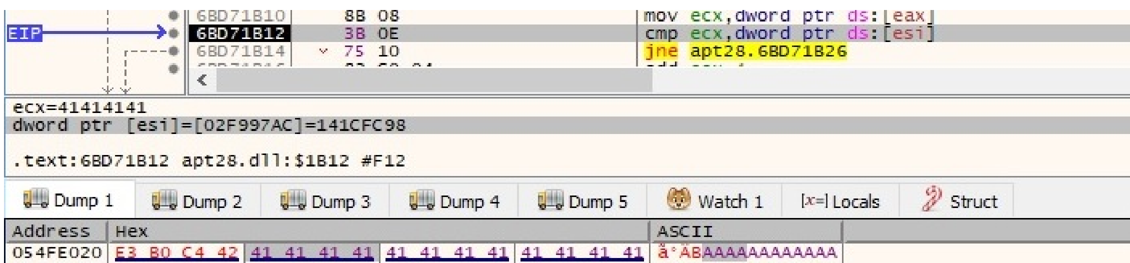

Figure 55

GetTempPathW is utilized to retrieve the path of the %TEMP% directory:

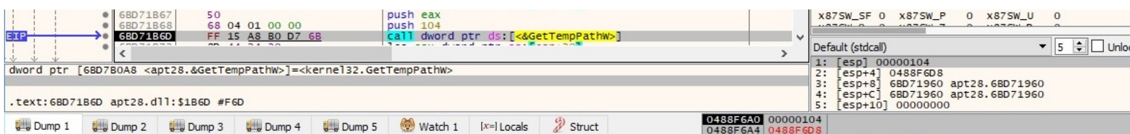

Figure 56

The malicious process creates a file called fvjoik.dll in the %TEMP% directory, as shown below:

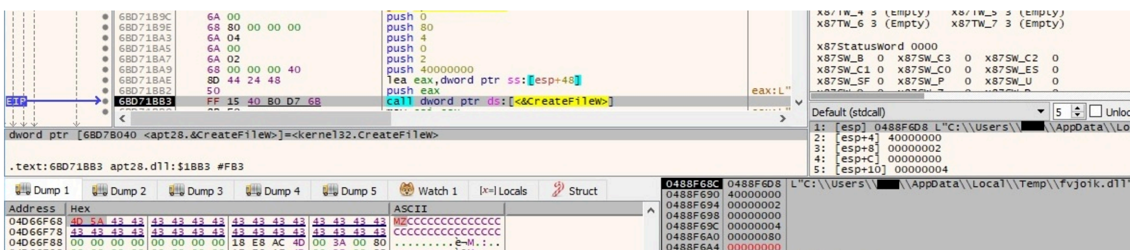

Figure 57

The newly created file is populated with the potential DLL downloaded from the C2 server:

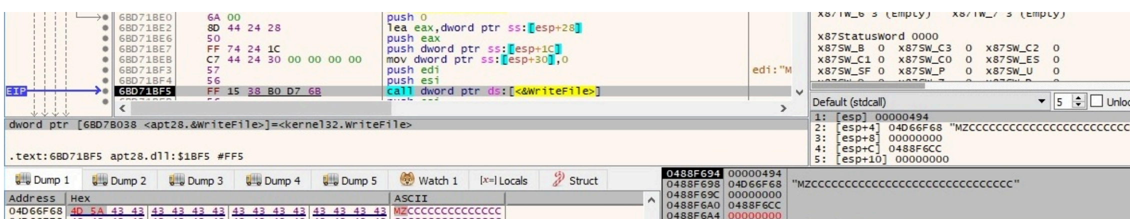

Figure 58

The DLL file is loaded into the address space of the current process using the LoadLibraryW routine:

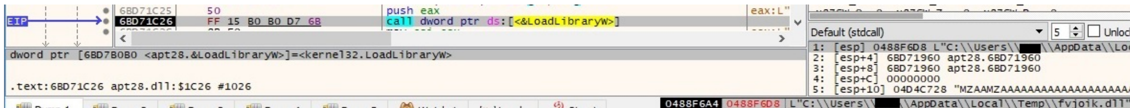

Figure 59

The malware will execute the exported function with ordinal 1, as highlighted in the next figure:

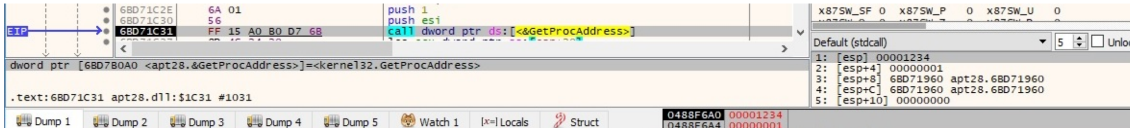

Figure 60

After the function finishes, there is a call to WinExec that deletes the DLL file created earlier:

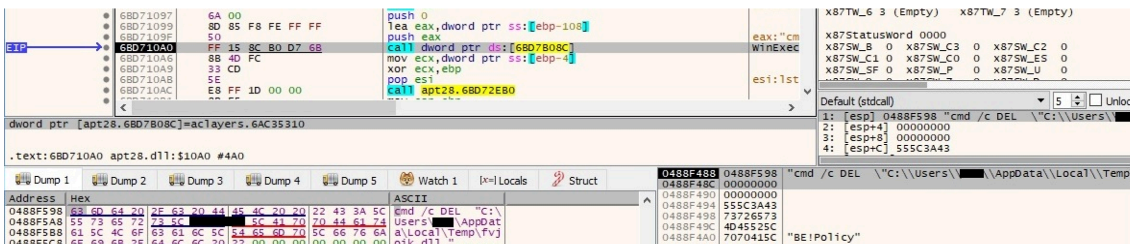

Figure 61

The process communicates again with the C2 server, and we believe that it transmits the result of the DLL execution (we won't go into too much details here because it's pretty much the same activity described so far). The parameters of the request are again as follows: "id=<hostname>#Username#<Serial number in decimal>¤t=1&total=1&data=<data to be transmitted>".

Main thread activity

The main thread sets the event created before to the signaled state:

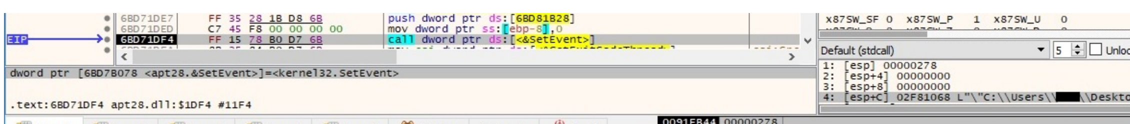

Figure 62

The malware retrieves the termination status of the 2 threads using the GetExitCodeThread API:

Figure 63

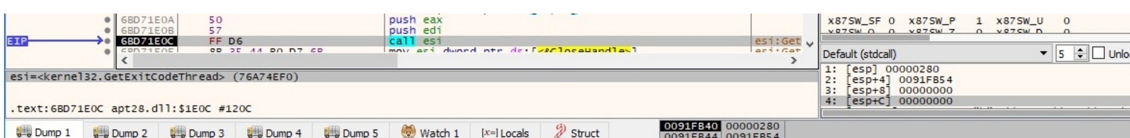

Figure 64

References

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

VirusTotal:
https://www.virustotal.com/gui/file/ae0bc3358fef0ca2a103e694aa556f55a3fed4e98ba57d16f5ae7ad4ad583698/detection

Fakenet: https://github.com/fireeye/flare-fakenet-ng

Cluster25: https://cluster25.io/wp-content/uploads/2021/05/2021-05_FancyBear.pdf

INDICATORS OF COMPROMISE

C2 server: updaterweb[.]com

SHA256: ae0bc3358fef0ca2a103e694aa556f55a3fed4e98ba57d16f5ae7ad4ad583698

User-Agent: Opera

Source: https://cybergeeks.tech/skinnyboy-apt28/