TokyoX: DLL side-loading an unknown artifact

lab52.io/blog/tokyox-dll-side-loading-an-unknown-artifact

ml10

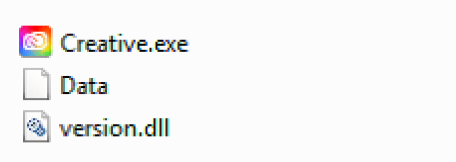
During Christmas holidays, Lab52 has been analyzing a sample which loads an artifact that we have decided to refer to as “TokyoX” since no similarities have been found as to any known malware, which we usually detect in open sources. However, we cannot confirm so far that it is indeed a new family of malware.

The first thing we identified was a DLL (382b3d3bb1be4f14dbc1e82a34946a52795288867ed86c6c43e4f981729be4fc) which had the following timestamps in VirusTotal at the time of the current analysis, and was uploaded from Russia via web site:

Creation Time 2021-12-09 02:46:43
First Submission 2021-12-09 08:48:20
Last Submission 2021-12-09 08:48:20
Last Analysis 2021-12-23 23:38:08

Some antivirus engines tagged the sample as PlugX, but it seems that the attribution might be due to the final payload’s loading mechanism: DLL sideloading with an encrypted payload in the same directory. After analyzing the final payload we could not find any similarities with other known samples from PlugX other than the loading TTPs.

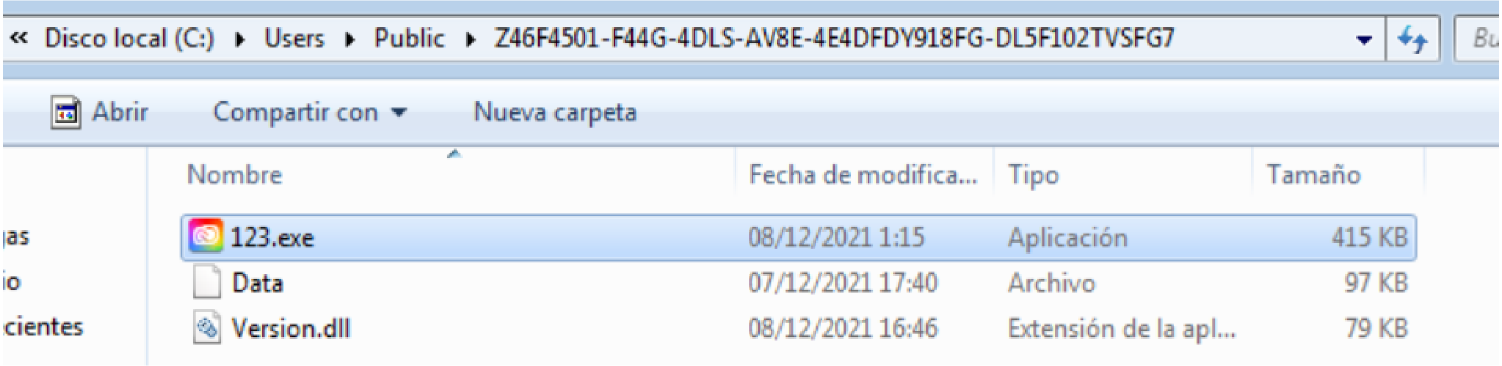
This DLL had a related .zip file with the name планирование.zip (translated to as planning.zip). When unzipping, the following files are observed:



Creative.exe
Data
version.dll

The legitimate file Creative.exe, an encrypted Data file and the version.dll DLL, which implements the loader function for the Data file, and therefore responsible of mapping the “TokyoX”.

If we execute it from a path which is not final or the expected by the malware, it replicates to another path and executes from there, which is something it does have in common with some PlugX dll loaders:



« Disco local (C:) ▸ Users ▸ Public ▸ Z46F4501-F44G-4DLS-AV8E-4E4DFDY918FG-DL5F102TVSFG7

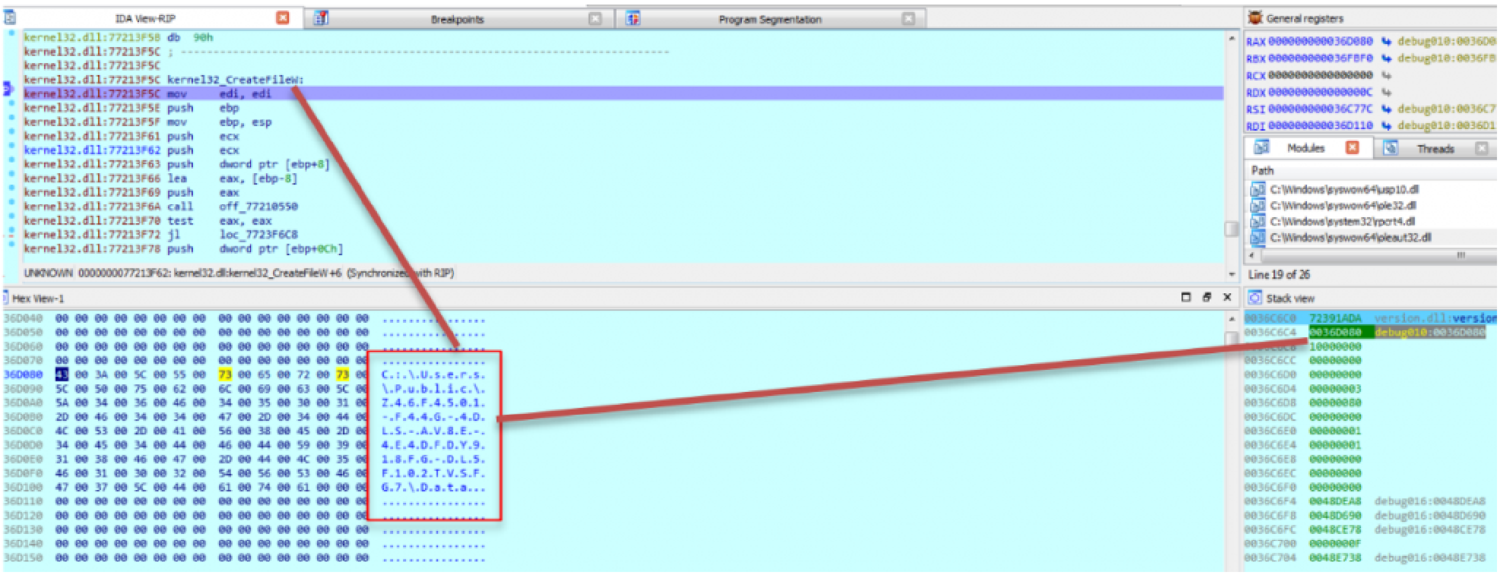
Nombre	Fecha de modifica...	Tipo	Tamaño
123.exe	08/12/2021 1:15	Aplicación	415 KB
Data	07/12/2021 17:40	Archivo	97 KB
Version.dll	08/12/2021 16:46	Extensión de la apl...	79 KB

Once executed, we observe how the netsh.exe process tries to establish connections with port 443 of the IP address 31.192.107[.]187.

In this analysis we will focus on different aspects about the process; from double-clicking the binary 123.exe process (which is a copy of Creative.exe but in another path) to the execution of “TokyoX” already decrypted in memory.

The first thing we observe within the process is how the version.dll library prepares the decryption and the final payload’s loading in the remote process:

In fact, we can see how the content of the Data file is read in the code section of version.dll:



```
kernel32.dll:77213F58 db  90h
kernel32.dll:77213F5C ; ---------------------------------------------------------------------------
kernel32.dll:77213F5C
kernel32.dll:77213F5C kernel32_CreateFileW:
kernel32.dll:77213F5C mov     edi, edi
kernel32.dll:77213F5E push    ebp
kernel32.dll:77213F5F mov     ebp, esp
kernel32.dll:77213F61 push    ecx
kernel32.dll:77213F62 push    ecx
kernel32.dll:77213F63 push    dword ptr [ebp+8]
kernel32.dll:77213F66 lea     eax, [ebp-8]
kernel32.dll:77213F69 push    eax
kernel32.dll:77213F6A call    off_77210550
kernel32.dll:77213F70 test    eax, eax
kernel32.dll:77213F72 jl      loc_7723F6C8
kernel32.dll:77213F78 push    dword ptr [ebp+0Ch]
UNKNOWN 000000077213F62: kernel32.dll:kernel32_CreateFileW+6 (Synchronized with RIP)
```

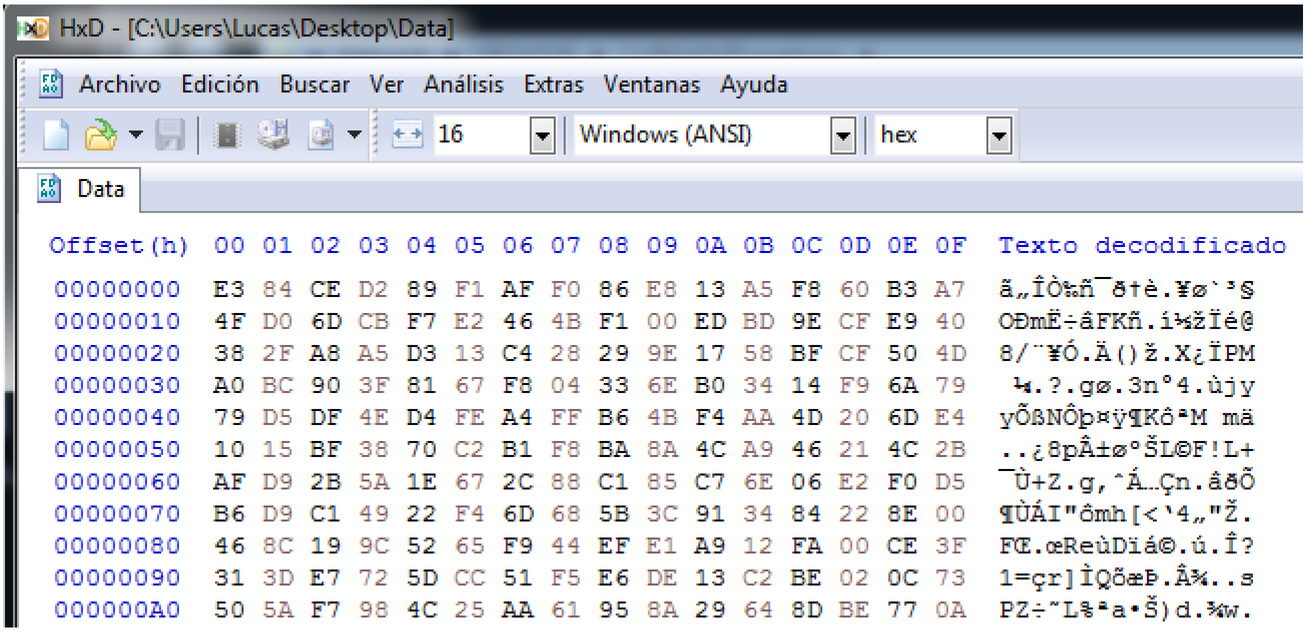
```
36D040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D080  43 00 3A 00 5C 00 55 00  73 00 65 00 72 00 73 00  C.:.\.U.s.e.r.s.
36D090  5C 00 50 00 75 00 62 00  6C 00 69 00 63 00 5C 00  \.P.u.b.l.i.c.\.
36D0A0  5A 00 34 00 36 00 46 00  34 00 35 00 30 00 31 00  Z.4.6.F.4.5.0.1.
36D0B0  2D 00 46 00 34 00 34 00  47 00 2D 00 34 00 44 00  -.F.4.4.G.-.4.D.
36D0C0  4C 00 53 00 2D 00 41 00  56 00 38 00 45 00 2D 00  L.S.-.A.V.8.E.-.
36D0D0  34 00 45 00 34 00 44 00  46 00 44 00 59 00 39 00  4.E.4.D.F.D.Y.9.
36D0E0  31 00 38 00 46 00 47 00  2D 00 44 00 4C 00 35 00  1.8.F.G.-.D.L.5.
36D0F0  46 00 31 00 30 00 32 00  54 00 56 00 53 00 46 00  F.1.0.2.T.V.S.F.
36D100  47 00 37 00 5C 00 44 00  61 00 74 00 61 00 00 00  G.7.\.D.a.t.a...
36D110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36D150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
RAX 000000000036D080 ↳ debug010:00360D0
RBX 000000000036FBF0 ↳ debug010:0036FB
RCX 0000000000000000 ↳
RDX 000000000000000C ↳
RSI 000000000036C77C ↳ debug010:0036C7
RDI 000000000036D110 ↳ debug010:0036D1
```

```
C:\Windows\syswow64\usp10.dll
C:\Windows\syswow64\ole32.dll
C:\Windows\system32\rpcrt4.dll
C:\Windows\syswow64\oleaut32.dll
Line 19 of 26
```

```
0036C6C0  72391ADA  version.dll:version
0036C6C4  0036D080  debug010:0036D080
0036C6C8  10000000
0036C6CC  00000000
0036C6D0  00000000
0036C6D4  00000003
0036C6D8  00000080
0036C6DC  00000000
0036C6E0  00000001
0036C6E4  00000001
0036C6E8  00000000
0036C6EC  00000000
0036C6F0  00000000
0036C6F4  0048DEA8  debug016:0048DEA8
0036C6F8  0048D690  debug016:0048D690
0036C6FC  0048CE78  debug016:0048CE78
0036C700  0000000F
0036C704  0048E738  debug016:0048E738
```

If we edit the Data file with a hexadecimal editor we will see their values, which will help us to identify it in memory later (beginning with E3 84):

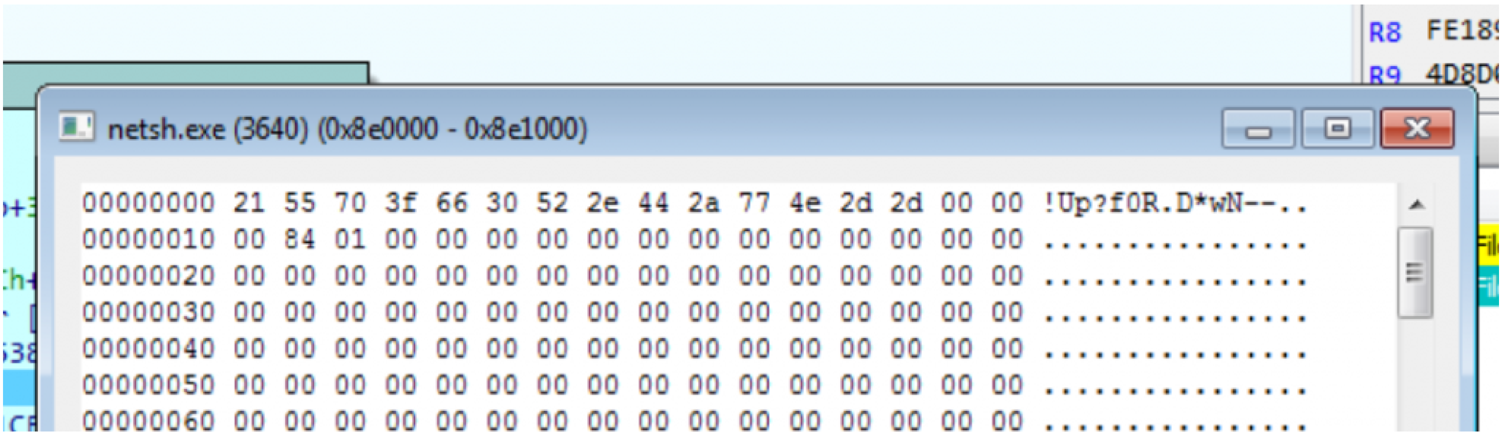

HxD - [C:\Users\Lucas\Desktop\Data]
Archivo Edición Buscar Ver Análisis Extras Ventanas Ayuda
16
Windows (ANSI)
hex
Data

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Texto decodificado

00000000  E3 84 CE D2 89 F1 AF F0 86 E8 13 A5 F8 60 B3 A7  ã„ÎÒ‰ñ¯ð†è.¥ø`³§
00000010  4F D0 6D CB F7 E2 46 4B F1 00 ED BD 9E CF E9 40  OÐmË÷âFKñ.í½žÏé@
00000020  38 2F A8 A5 D3 13 C4 28 29 9E 17 58 BF CF 50 4D  8/¨¥Ó.Ä()ž.X¿ÏPM
00000030  A0 BC 90 3F 81 67 F8 04 33 6E B0 34 14 F9 6A 79   ¼.?.gø.3n°4.ùjy
00000040  79 D5 DF 4E D4 FE A4 FF B6 4B F4 AA 4D 20 6D E4  yÕßNÔþ¤ÿ¶KôªM mä
00000050  10 15 BF 38 70 C2 B1 F8 BA 8A 4C A9 46 21 4C 2B  ..¿8pÂ±øºŠL©F!L+
00000060  AF D9 2B 5A 1E 67 2C 88 C1 85 C7 6E 06 E2 F0 D5  ¯Ù+Z.g,ˆÁ…Çn.âðÕ
00000070  B6 D9 C1 49 22 F4 6D 68 5B 3C 91 34 84 22 8E 00  ¶ÙÁI"ômh[<‘4„"Ž.
00000080  46 8C 19 9C 52 65 F9 44 EF E1 A9 12 FA 00 CE 3F  FŒ.œReùDïá©.ú.Î?
00000090  31 3D E7 72 5D CC 51 F5 E6 DE 13 C2 BE 02 0C 73  1=çr]ÌQõæÞ.Â¾..s
000000A0  50 5A F7 98 4C 25 AA 61 95 8A 29 64 8D BE 77 0A  PZ÷˜L%ªa•Š)d.¾w.
```

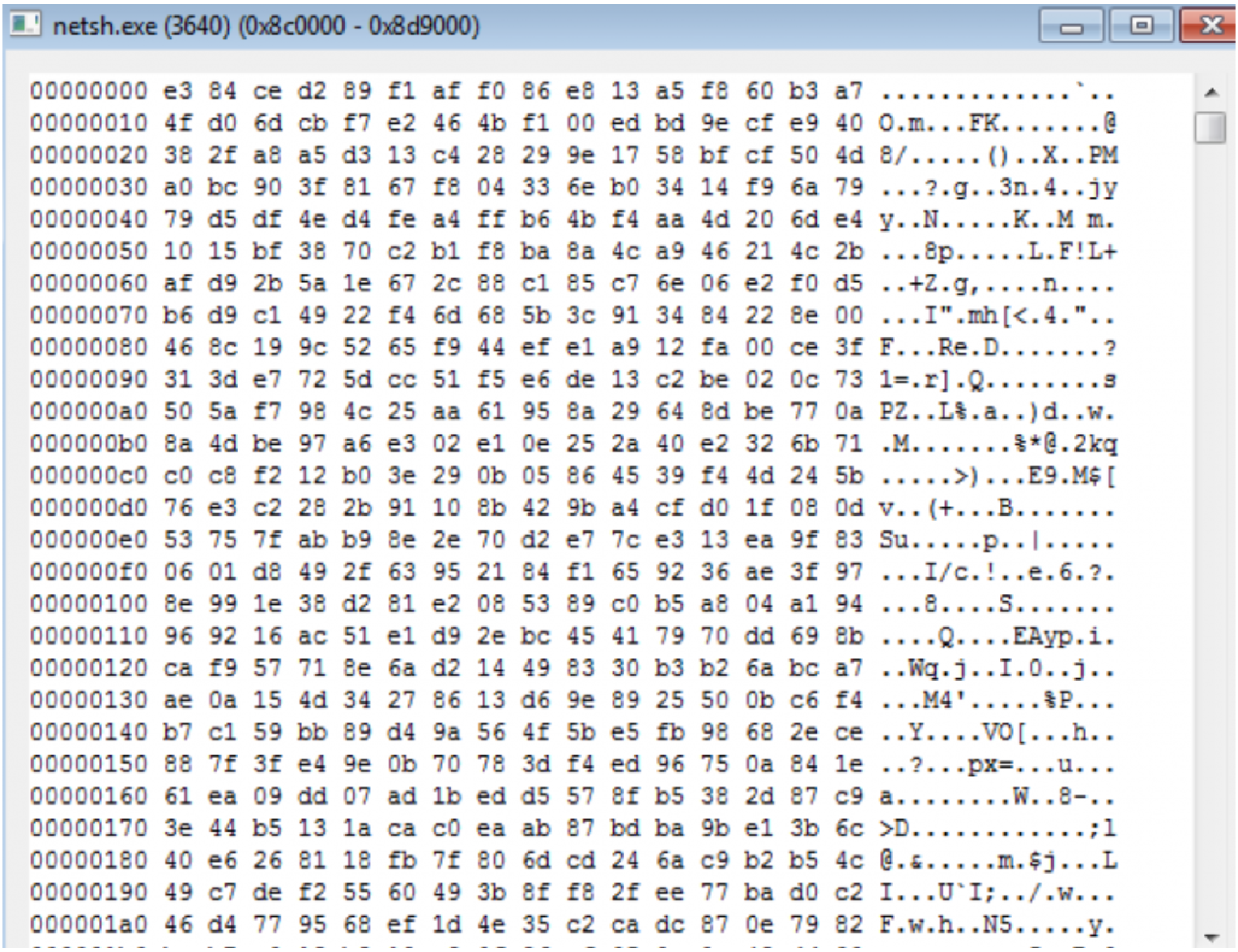
After reading the file from disk, a child process netsh.exe is created. This just-created child process is where several new memory segments will be located (a total of 5, including the final decrypted payload) to decrypt the final “TokyoX” payload. The APIs which were observed for the creation and writing of the remote process are the native APIs NtAllocateVirtualmemory and NtwriteVirtualmemory.

First, it creates two segments: 100Kb where the encrypted payload is located and which comes from the disc, and another one of 4Kb. In the 4Kb segment we observe how the following string is set (which will be the string used for the decrypting process):



```
00000000 21 55 70 3f 66 30 52 2e 44 2a 77 4e 2d 2d 00 00 !Up?f0R.D*wN--..
00000010 00 84 01 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

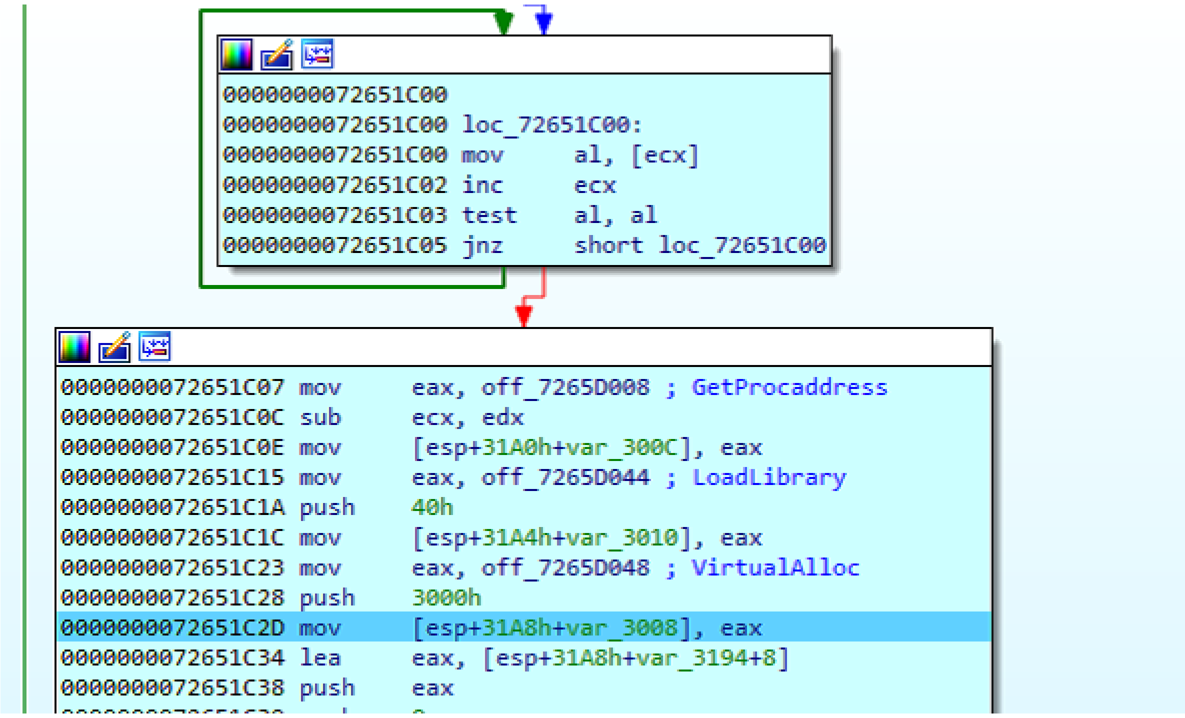
The other memory segment of 100Kb contains the following (encrypted content, as we see how it matches the content from Data file on Disk):



netsh.exe (3640) (0x8c0000 - 0x8d9000)

```
00000000 e3 84 ce d2 89 f1 af f0 86 e8 13 a5 f8 60 b3 a7 .............`..
00000010 4f d0 6d cb f7 e2 46 4b f1 00 ed bd 9e cf e9 40 O.m...FK.......@
00000020 38 2f a8 a5 d3 13 c4 28 29 9e 17 58 bf cf 50 4d 8/.....()..X..PM
00000030 a0 bc 90 3f 81 67 f8 04 33 6e b0 34 14 f9 6a 79 ...?.g..3n.4..jy
00000040 79 d5 df 4e d4 fe a4 ff b6 4b f4 aa 4d 20 6d e4 y..N.....K..M m.
00000050 10 15 bf 38 70 c2 b1 f8 ba 8a 4c a9 46 21 4c 2b ...8p.....L.F!L+
00000060 af d9 2b 5a 1e 67 2c 88 c1 85 c7 6e 06 e2 f0 d5 ..+Z.g,....n....
00000070 b6 d9 c1 49 22 f4 6d 68 5b 3c 91 34 84 22 8e 00 ...I".mh[<.4."..
00000080 46 8c 19 9c 52 65 f9 44 ef e1 a9 12 fa 00 ce 3f F...Re.D.......?
00000090 31 3d e7 72 5d cc 51 f5 e6 de 13 c2 be 02 0c 73 1=.r].Q........s
000000a0 50 5a f7 98 4c 25 aa 61 95 8a 29 64 8d be 77 0a PZ..L%.a..)d..w.
000000b0 8a 4d be 97 a6 e3 02 e1 0e 25 2a 40 e2 32 6b 71 .M.......%*@.2kq
000000c0 c0 c8 f2 12 b0 3e 29 0b 05 86 45 39 f4 4d 24 5b .....>)...E9.M$[
000000d0 76 e3 c2 28 2b 91 10 8b 42 9b a4 cf d0 1f 08 0d v..(+...B.......
000000e0 53 75 7f ab b9 8e 2e 70 d2 e7 7c e3 13 ea 9f 83 Su.....p..|.....
000000f0 06 01 d8 49 2f 63 95 21 84 f1 65 92 36 ae 3f 97 ...I/c.!..e.6.?.
00000100 8e 99 1e 38 d2 81 e2 08 53 89 c0 b5 a8 04 a1 94 ...8....S.......
00000110 96 92 16 ac 51 e1 d9 2e bc 45 41 79 70 dd 69 8b ....Q....EAyp.i.
00000120 ca f9 57 71 8e 6a d2 14 49 83 30 b3 b2 6a bc a7 ..Wq.j..I.0..j..
00000130 ae 0a 15 4d 34 27 86 13 d6 9e 89 25 50 0b c6 f4 ...M4'.....%P...
00000140 b7 c1 59 bb 89 d4 9a 56 4f 5b e5 fb 98 68 2e ce ..Y....VO[...h..
00000150 88 7f 3f e4 9e 0b 70 78 3d f4 ed 96 75 0a 84 1e ..?...px=...u...
00000160 61 ea 09 dd 07 ad 1b ed d5 57 8f b5 38 2d 87 c9 a........W..8-..
00000170 3e 44 b5 13 1a ca c0 ea ab 87 bd ba 9b e1 3b 6c >D............;l
00000180 40 e6 26 81 18 fb 7f 80 6d cd 24 6a c9 b2 b5 4c @.&.....m.$j...L
00000190 49 c7 de f2 55 60 49 3b 8f f8 2f ee 77 ba d0 c2 I...U`I;../.w...
000001a0 46 d4 77 95 68 ef 1d 4e 35 c2 ca dc 87 0e 79 82 F.w.h..N5.....y.
```

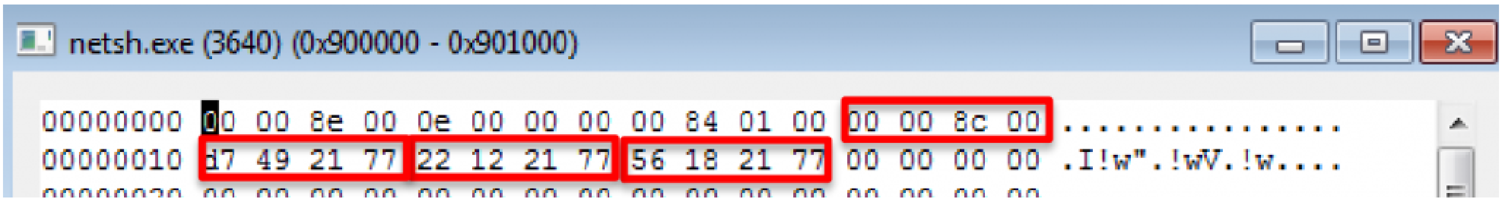
After the creation of these two segments, a third segment is allocated, where it is loaded the absolute memory addresses from several win32 APIs (VirtualAlloc, LoadLibrary, GetProcAddress, the home address of the coded payload, etc.) for its later use by the loader:



```
0000000072651C00
0000000072651C00 loc_72651C00:
0000000072651C00 mov     al, [ecx]
0000000072651C02 inc     ecx
0000000072651C03 test    al, al
0000000072651C05 jnz     short loc_72651C00
```

```
0000000072651C07 mov     eax, off_7265D008 ; GetProcaddress
0000000072651C0C sub     ecx, edx
0000000072651C0E mov     [esp+31A0h+var_300C], eax
0000000072651C15 mov     eax, off_7265D044 ; LoadLibrary
0000000072651C1A push    40h
0000000072651C1C mov     [esp+31A4h+var_3010], eax
0000000072651C23 mov     eax, off_7265D048 ; VirtualAlloc
0000000072651C28 push    3000h
0000000072651C2D mov     [esp+31A8h+var_3008], eax
0000000072651C34 lea     eax, [esp+31A8h+var_3194+8]
0000000072651C38 push    eax
```

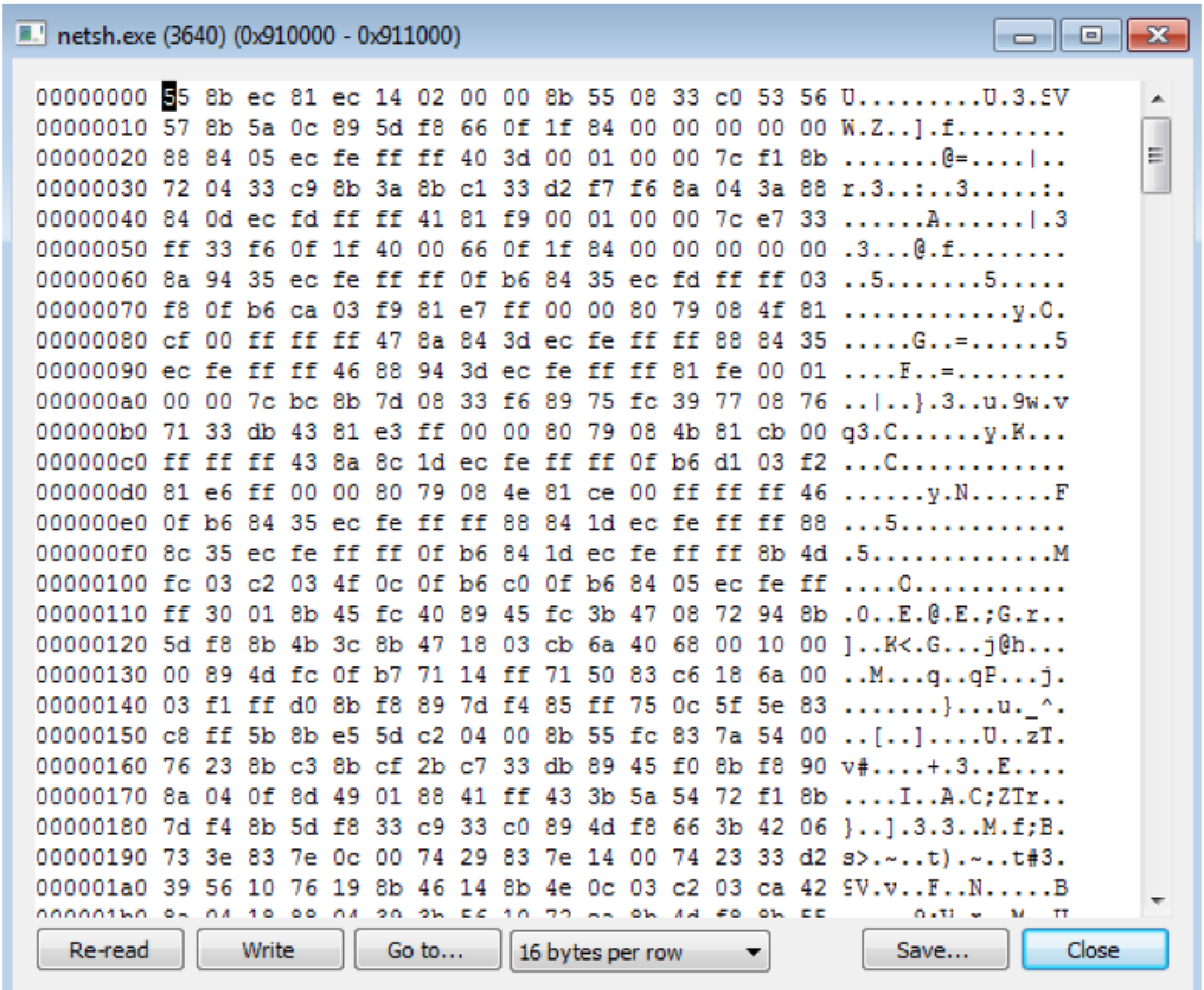
We can notice how the segment will have the memory addresses (starting from 123.exe they are located in netsh.exe segment through the version.dll code):



netsh.exe (3640) (0x900000 - 0x901000)

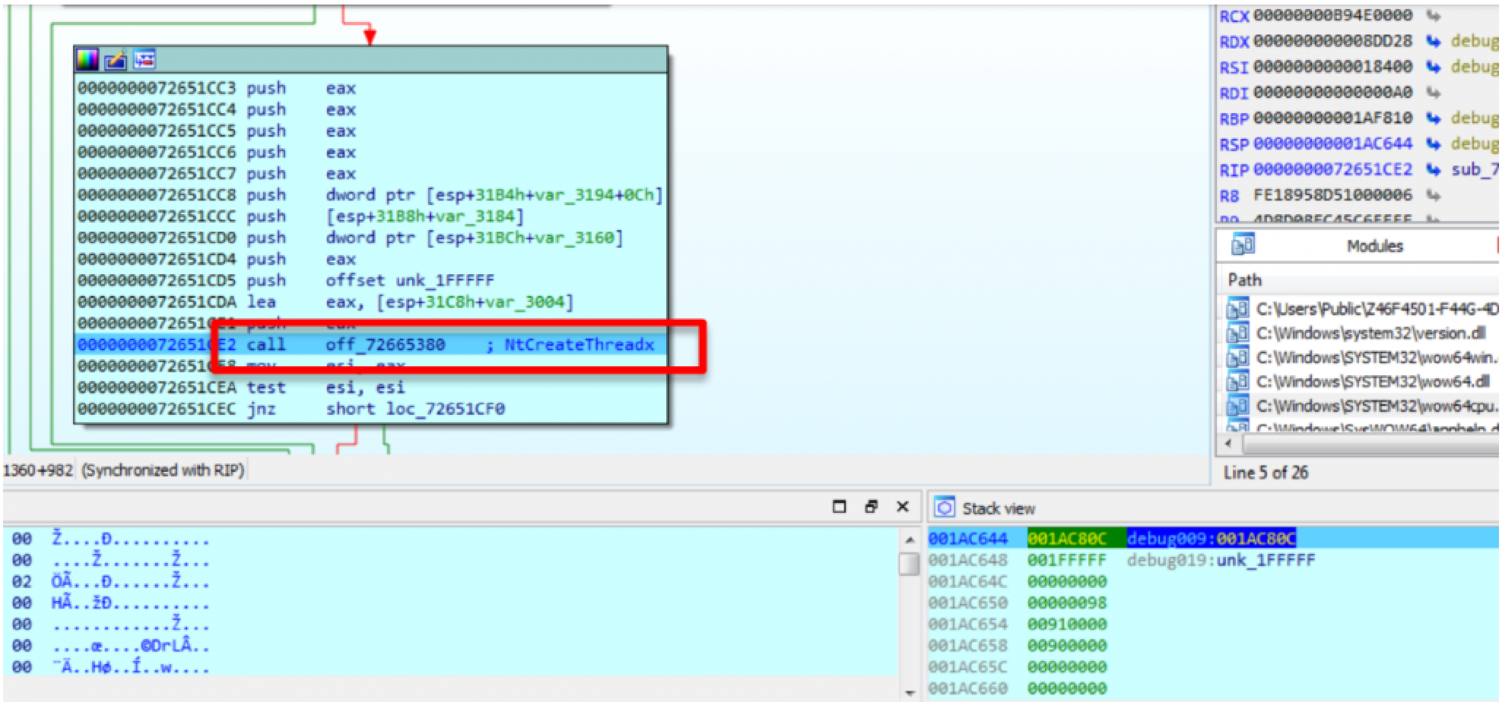
```
00000000 00 00 8e 00 0e 00 00 00 00 84 01 00 00 00 8c 00 ................
00000010 d7 49 21 77 22 12 21 77 56 18 21 77 00 00 00 00 .I!w".!wV.!w....
```

Then, another segment of 4Kb is created where it loads the code that will decrypt and load the final payload.



```
00000000 55 8b ec 81 ec 14 02 00 00 8b 55 08 33 c0 53 56 U.........U.3.SV
00000010 57 8b 5a 0c 89 5d f8 66 0f 1f 84 00 00 00 00 00 W.Z..].f........
00000020 88 84 05 ec fe ff ff 40 3d 00 01 00 00 7c f1 8b .......@=....|..
00000030 72 04 33 c9 8b 3a 8b c1 33 d2 f7 f6 8a 04 3a 88 r.3..:..3.....:.
00000040 84 0d ec fd ff ff 41 81 f9 00 01 00 00 7c e7 33 ......A......|.3
00000050 ff 33 f6 0f 1f 40 00 66 0f 1f 84 00 00 00 00 00 .3...@.f........
00000060 8a 94 35 ec fe ff ff 0f b6 84 35 ec fd ff ff 03 ..5.......5.....
00000070 f8 0f b6 ca 03 f9 81 e7 ff 00 00 80 79 08 4f 81 ............y.O.
00000080 cf 00 ff ff ff 47 8a 84 3d ec fe ff ff 88 84 35 .....G..=......5
00000090 ec fe ff ff 46 88 94 3d ec fe ff ff 81 fe 00 01 ....F..=........
000000a0 00 00 7c bc 8b 7d 08 33 f6 89 75 fc 39 77 08 76 ..|..}.3..u.9w.v
000000b0 71 33 db 43 81 e3 ff 00 00 80 79 08 4b 81 cb 00 q3.C......y.K...
000000c0 ff ff ff 43 8a 8c 1d ec fe ff ff 0f b6 d1 03 f2 ...C............
000000d0 81 e6 ff 00 00 80 79 08 4e 81 ce 00 ff ff ff 46 ......y.N......F
000000e0 0f b6 84 35 ec fe ff ff 88 84 1d ec fe ff ff 88 ...5............
000000f0 8c 35 ec fe ff ff 0f b6 84 1d ec fe ff ff 8b 4d .5.............M
00000100 fc 03 c2 03 4f 0c 0f b6 c0 0f b6 84 05 ec fe ff ....O...........
00000110 ff 30 01 8b 45 fc 40 89 45 fc 3b 47 08 72 94 8b .0..E.@.E.;G.r..
00000120 5d f8 8b 4b 3c 8b 47 18 03 cb 6a 40 68 00 10 00 ]..K<.G...j@h...
00000130 00 89 4d fc 0f b7 71 14 ff 71 50 83 c6 18 6a 00 ..M...q..qP...j.
00000140 03 f1 ff d0 8b f8 89 7d f4 85 ff 75 0c 5f 5e 83 .......}...u._^.
00000150 c8 ff 5b 8b e5 5d c2 04 00 8b 55 fc 83 7a 54 00 ..[..]....U..zT.
00000160 76 23 8b c3 8b cf 2b c7 33 db 89 45 f0 8b f8 90 v#....+.3..E....
00000170 8a 04 0f 8d 49 01 88 41 ff 43 3b 5a 54 72 f1 8b ....I..A.C;ZTr..
00000180 7d f4 8b 5d f8 33 c9 33 c0 89 4d f8 66 3b 42 06 }..].3.3..M.f;B.
00000190 73 3e 83 7e 0c 00 74 29 83 7e 14 00 74 23 33 d2 s>.~..t).~..t#3.
000001a0 39 56 10 76 19 8b 46 14 8b 4e 0c 03 c2 03 ca 42 9V.v..F..N.....B
```

Finally, the “TokyoX” loader runs from the DLL (version.dll) in netsh.exe through the API NtcreateThreadEx and we see the start of the last page created in the stack:



```
0000000072651CC3 push    eax
0000000072651CC4 push    eax
0000000072651CC5 push    eax
0000000072651CC6 push    eax
0000000072651CC7 push    eax
0000000072651CC8 push    dword ptr [esp+31B4h+var_3194+0Ch]
0000000072651CCC push    [esp+31B8h+var_3184]
0000000072651CD0 push    dword ptr [esp+31BCh+var_3160]
0000000072651CD4 push    eax
0000000072651CD5 push    offset unk_1FFFFF
0000000072651CDA lea     eax, [esp+31C8h+var_3004]
0000000072651CE1 push    eax
0000000072651CE2 call    off_72665380    ; NtCreateThreadx
0000000072651CE8 mov     esi, eax
0000000072651CEA test    esi, esi
0000000072651CEC jnz     short loc_72651CF0
```

1360+982 (Synchronized with RIP)

```
RCX 00000000B94E0000
RDX 000000000008DD28  debug
RSI 0000000000018400  debug
RDI 00000000000000A0
RBP 00000000001AF810  debug
RSP 00000000001AC644  debug
RIP 0000000072651CE2  sub_7
R8  FE18958D51000006
R9  4D8D08EC45C6FFFF
```

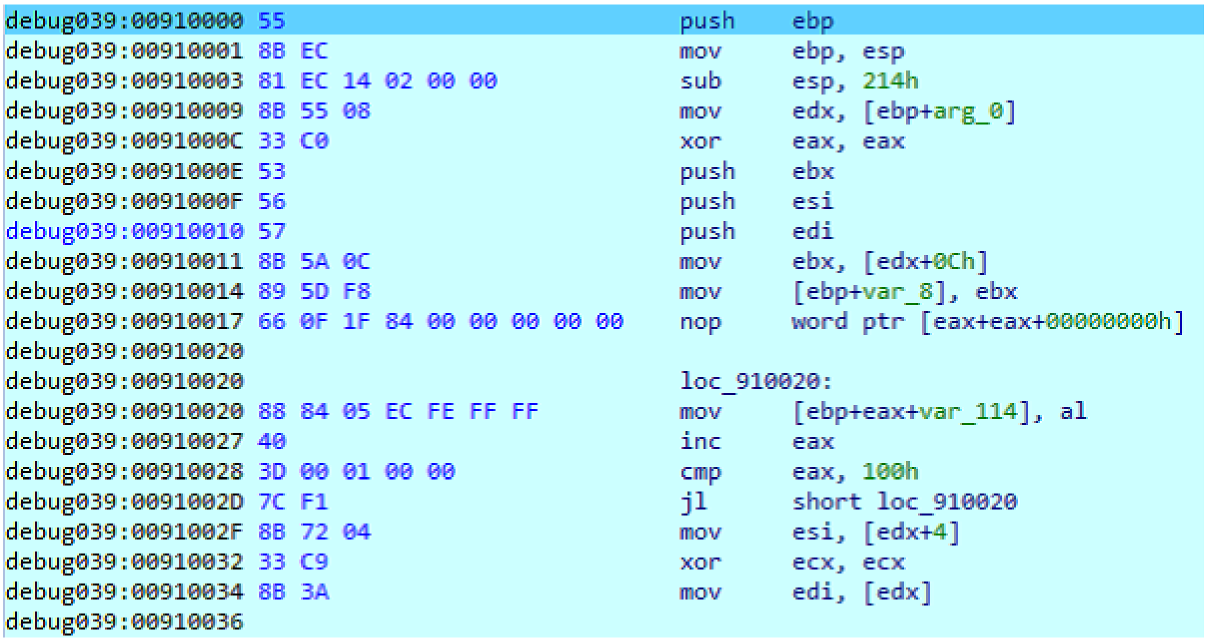
Path
C:\Users\Public\Z46F4501-F44G-4D
C:\Windows\system32\version.dll
C:\Windows\SYSTEM32\wow64win.
C:\Windows\SYSTEM32\wow64.dll
C:\Windows\SYSTEM32\wow64cpu.
C:\Windows\SysWOW64\apphelp.d

Line 5 of 26

```
00  Ž....Ð..........
00  ....Ž.......Ž...
02  ÕÃ...Ð......Ž...
00  HÃ..žÐ..........
00  ............Ž...
00  ....æ....©DrLÂ..
00  ¨Ã..Hø..Í..w....
```

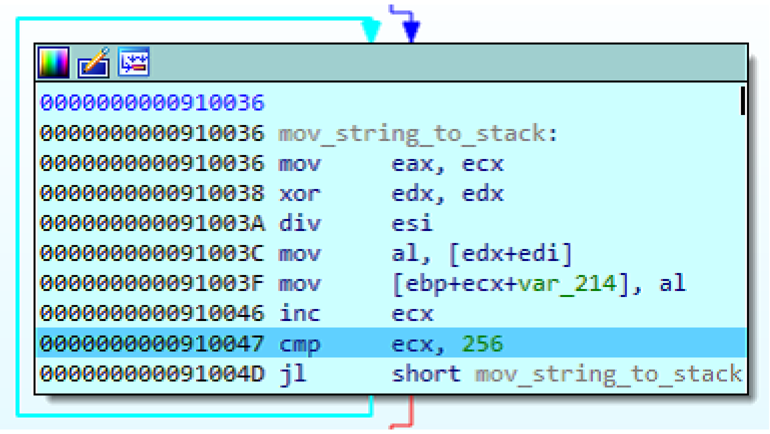
```
001AC644  001AC80C  debug009:001AC80C
001AC648  001FFFFF  debug019:unk_1FFFFF
001AC64C  00000000
001AC650  00000098
001AC654  00910000
001AC658  00900000
001AC65C  00000000
001AC660  00000000
```

After the execution of NtCreateThreadEx, as indicated, the loader is initiated in netsh.exe in the segment:

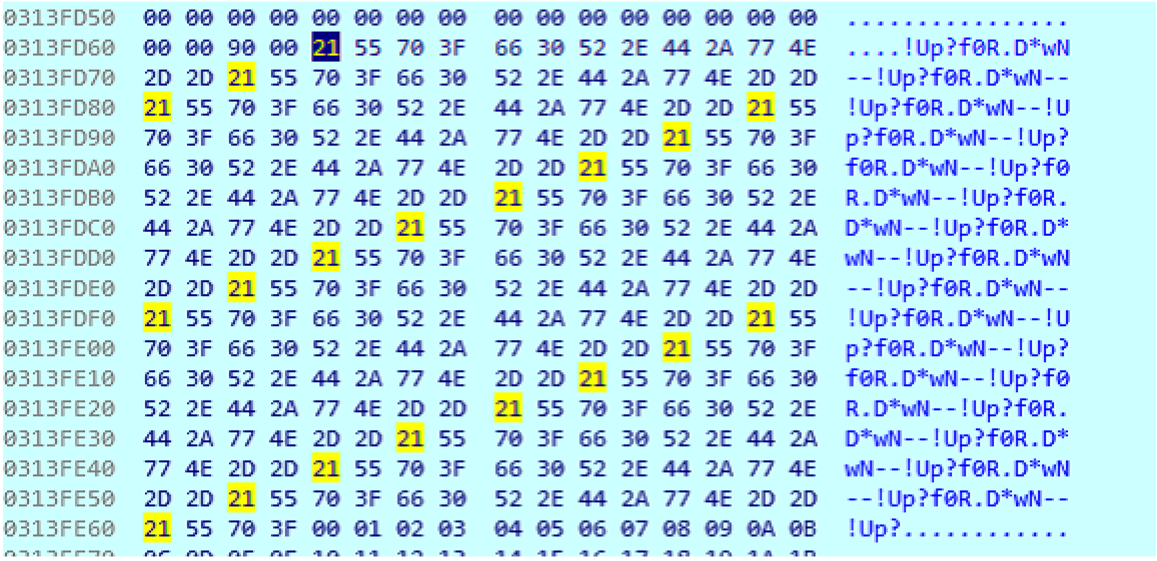


```
debug039:00910000 55                          push    ebp
debug039:00910001 8B EC                       mov     ebp, esp
debug039:00910003 81 EC 14 02 00 00           sub     esp, 214h
debug039:00910009 8B 55 08                    mov     edx, [ebp+arg_0]
debug039:0091000C 33 C0                       xor     eax, eax
debug039:0091000E 53                          push    ebx
debug039:0091000F 56                          push    esi
debug039:00910010 57                          push    edi
debug039:00910011 8B 5A 0C                    mov     ebx, [edx+0Ch]
debug039:00910014 89 5D F8                    mov     [ebp+var_8], ebx
debug039:00910017 66 0F 1F 84 00 00 00 00 00  nop     word ptr [eax+eax+00000000h]
debug039:00910020
debug039:00910020                             loc_910020:
debug039:00910020 88 84 05 EC FE FF FF        mov     [ebp+eax+var_114], al
debug039:00910027 40                          inc     eax
debug039:00910028 3D 00 01 00 00              cmp     eax, 100h
debug039:0091002D 7C F1                       jl      short loc_910020
debug039:0091002F 8B 72 04                    mov     esi, [edx+4]
debug039:00910032 33 C9                       xor     ecx, ecx
debug039:00910034 8B 3A                       mov     edi, [edx]
debug039:00910036
```

Once the execution is moved to the netsh.exe process, it takes the string located in the initial 4Kb segment, copies it into the stack and replicates it (0x100, 256 bytes) to match the specific block size of 256bytes. In the following screenshots we can observe how the block ends with the string “!Up?” when it reaches the value 0x100 in hexadecimal.

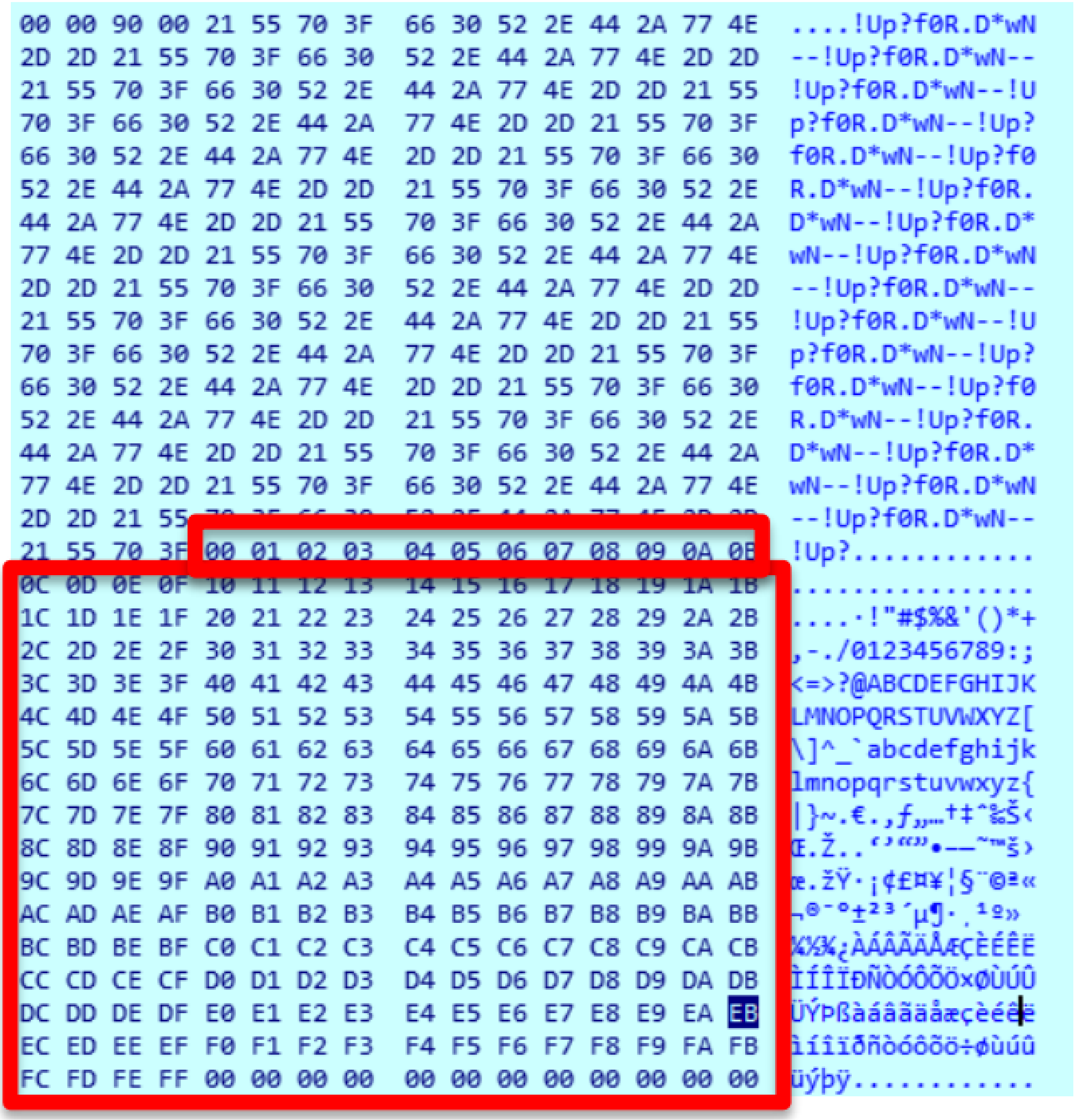


```
0000000000910036
0000000000910036 mov_string_to_stack:
0000000000910036 mov     eax, ecx
0000000000910038 xor     edx, edx
000000000091003A div     esi
000000000091003C mov     al, [edx+edi]
000000000091003F mov     [ebp+ecx+var_214], al
0000000000910046 inc     ecx
0000000000910047 cmp     ecx, 256
000000000091004D jl      short mov_string_to_stack
```



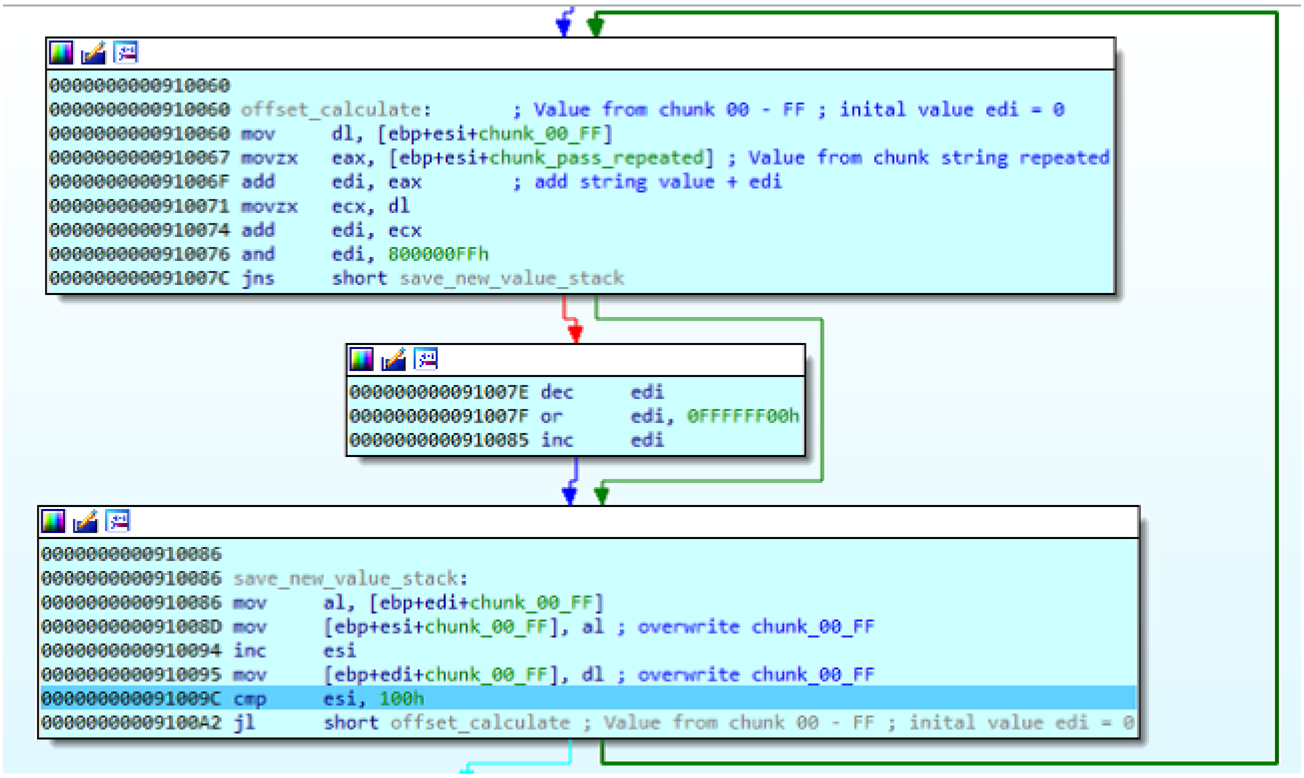
```
0313FD50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0313FD60  00 00 90 00 21 55 70 3F  66 30 52 2E 44 2A 77 4E  ....!Up?f0R.D*wN
0313FD70  2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
0313FD80  21 55 70 3F 66 30 52 2E  44 2A 77 4E 2D 2D 21 55  !Up?f0R.D*wN--!U
0313FD90  70 3F 66 30 52 2E 44 2A  77 4E 2D 2D 21 55 70 3F  p?f0R.D*wN--!Up?
0313FDA0  66 30 52 2E 44 2A 77 4E  2D 2D 21 55 70 3F 66 30  f0R.D*wN--!Up?f0
0313FDB0  52 2E 44 2A 77 4E 2D 2D  21 55 70 3F 66 30 52 2E  R.D*wN--!Up?f0R.
0313FDC0  44 2A 77 4E 2D 2D 21 55  70 3F 66 30 52 2E 44 2A  D*wN--!Up?f0R.D*
0313FDD0  77 4E 2D 2D 21 55 70 3F  66 30 52 2E 44 2A 77 4E  wN--!Up?f0R.D*wN
0313FDE0  2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
0313FDF0  21 55 70 3F 66 30 52 2E  44 2A 77 4E 2D 2D 21 55  !Up?f0R.D*wN--!U
0313FE00  70 3F 66 30 52 2E 44 2A  77 4E 2D 2D 21 55 70 3F  p?f0R.D*wN--!Up?
0313FE10  66 30 52 2E 44 2A 77 4E  2D 2D 21 55 70 3F 66 30  f0R.D*wN--!Up?f0
0313FE20  52 2E 44 2A 77 4E 2D 2D  21 55 70 3F 66 30 52 2E  R.D*wN--!Up?f0R.
0313FE30  44 2A 77 4E 2D 2D 21 55  70 3F 66 30 52 2E 44 2A  D*wN--!Up?f0R.D*
0313FE40  77 4E 2D 2D 21 55 70 3F  66 30 52 2E 44 2A 77 4E  wN--!Up?f0R.D*wN
0313FE50  2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
0313FE60  21 55 70 3F 00 01 02 03  04 05 06 07 08 09 0A 0B  !Up?............
0313FE70  0C 0D 0E 0F 10 11 12 13  14 15 16 17 18 19 1A 1B
```

After the block is created with the replicated string, the values from 00 to FF are found and used for the decrypting process.



```
00 00 90 00 21 55 70 3F  66 30 52 2E 44 2A 77 4E  ....!Up?f0R.D*wN
2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
21 55 70 3F 66 30 52 2E  44 2A 77 4E 2D 2D 21 55  !Up?f0R.D*wN--!U
70 3F 66 30 52 2E 44 2A  77 4E 2D 2D 21 55 70 3F  p?f0R.D*wN--!Up?
66 30 52 2E 44 2A 77 4E  2D 2D 21 55 70 3F 66 30  f0R.D*wN--!Up?f0
52 2E 44 2A 77 4E 2D 2D  21 55 70 3F 66 30 52 2E  R.D*wN--!Up?f0R.
44 2A 77 4E 2D 2D 21 55  70 3F 66 30 52 2E 44 2A  D*wN--!Up?f0R.D*
77 4E 2D 2D 21 55 70 3F  66 30 52 2E 44 2A 77 4E  wN--!Up?f0R.D*wN
2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
21 55 70 3F 66 30 52 2E  44 2A 77 4E 2D 2D 21 55  !Up?f0R.D*wN--!U
70 3F 66 30 52 2E 44 2A  77 4E 2D 2D 21 55 70 3F  p?f0R.D*wN--!Up?
66 30 52 2E 44 2A 77 4E  2D 2D 21 55 70 3F 66 30  f0R.D*wN--!Up?f0
52 2E 44 2A 77 4E 2D 2D  21 55 70 3F 66 30 52 2E  R.D*wN--!Up?f0R.
44 2A 77 4E 2D 2D 21 55  70 3F 66 30 52 2E 44 2A  D*wN--!Up?f0R.D*
77 4E 2D 2D 21 55 70 3F  66 30 52 2E 44 2A 77 4E  wN--!Up?f0R.D*wN
2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
21 55 70 3F 00 01 02 03  04 05 06 07 08 09 0A 0B  !Up?............
0C 0D 0E 0F 10 11 12 13  14 15 16 17 18 19 1A 1B  ................
1C 1D 1E 1F 20 21 22 23  24 25 26 27 28 29 2A 2B  ....·!"#$%&'()*+
2C 2D 2E 2F 30 31 32 33  34 35 36 37 38 39 3A 3B  ,-./0123456789:;
3C 3D 3E 3F 40 41 42 43  44 45 46 47 48 49 4A 4B  <=>?@ABCDEFGHIJK
4C 4D 4E 4F 50 51 52 53  54 55 56 57 58 59 5A 5B  LMNOPQRSTUVWXYZ[
5C 5D 5E 5F 60 61 62 63  64 65 66 67 68 69 6A 6B  \]^_`abcdefghijk
6C 6D 6E 6F 70 71 72 73  74 75 76 77 78 79 7A 7B  lmnopqrstuvwxyz{
7C 7D 7E 7F 80 81 82 83  84 85 86 87 88 89 8A 8B  |}~.€.‚ƒ„…†‡ˆ‰Š‹
8C 8D 8E 8F 90 91 92 93  94 95 96 97 98 99 9A 9B  Œ.Ž..‘’“”•–—˜™š›
9C 9D 9E 9F A0 A1 A2 A3  A4 A5 A6 A7 A8 A9 AA AB  œ.žŸ·¡¢£¤¥¦§¨©ª«
AC AD AE AF B0 B1 B2 B3  B4 B5 B6 B7 B8 B9 BA BB  ¬-®¯°±²³´µ¶·¸¹º»
BC BD BE BF C0 C1 C2 C3  C4 C5 C6 C7 C8 C9 CA CB  ¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊË
CC CD CE CF D0 D1 D2 D3  D4 D5 D6 D7 D8 D9 DA DB  ÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛ
DC DD DE DF E0 E1 E2 E3  E4 E5 E6 E7 E8 E9 EA EB  ÜÝÞßàáâãäåæçèéêë
EC ED EE EF F0 F1 F2 F3  F4 F5 F6 F7 F8 F9 FA FB  ìíîïðñòóôõö÷øùúû
FC FD FE FF 00 00 00 00  00 00 00 00 00 00 00 00  üýþÿ............
```

At this point, the loader transforms the 00-FF block with a series of additions combining the replicated string's block with the 00-FF block, as we can see:

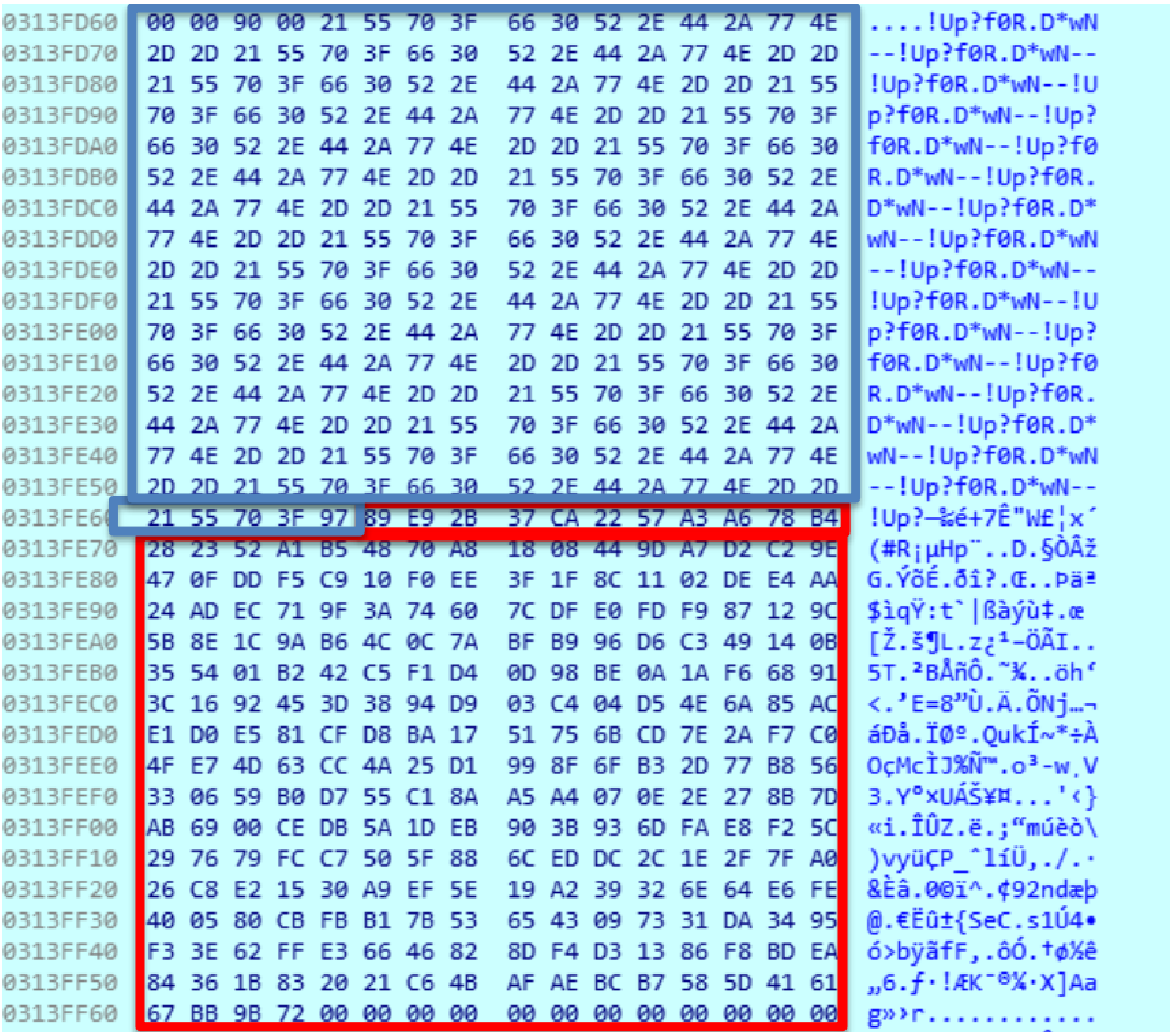


```
0000000000910060
0000000000910060 offset_calculate:        ; Value from chunk 00 - FF ; inital value edi = 0
0000000000910060 mov     dl, [ebp+esi+chunk_00_FF]
0000000000910067 movzx   eax, [ebp+esi+chunk_pass_repeated] ; Value from chunk string repeated
000000000091006F add     edi, eax         ; add string value + edi
0000000000910071 movzx   ecx, dl
0000000000910074 add     edi, ecx
0000000000910076 and     edi, 800000FFh
000000000091007C jns     short save_new_value_stack
```

```
000000000091007E dec     edi
000000000091007F or      edi, 0FFFFFF00h
0000000000910085 inc     edi
```

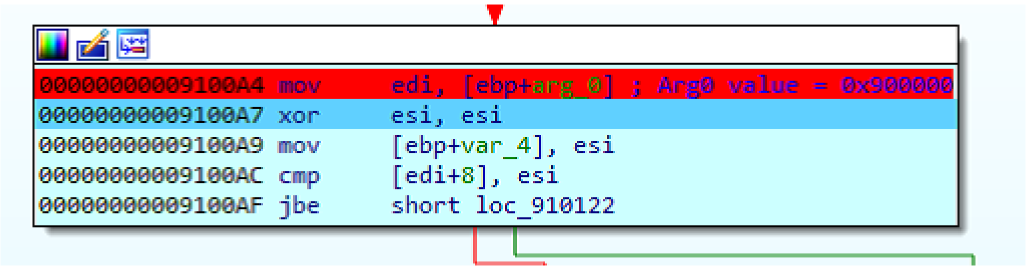
```
0000000000910086
0000000000910086 save_new_value_stack:
0000000000910086 mov     al, [ebp+edi+chunk_00_FF]
000000000091008D mov     [ebp+esi+chunk_00_FF], al ; overwrite chunk_00_FF
0000000000910094 inc     esi
0000000000910095 mov     [ebp+edi+chunk_00_FF], dl ; overwrite chunk_00_FF
000000000091009C cmp     esi, 100h
00000000009100A2 jl      short offset_calculate ; Value from chunk 00 - FF ; inital value edi = 0
```

The combination of the blue block (in following image) and the 00-FF block (pointed in red in previous image) results in the following block in memory, marked in red in the image:



```
0313FD60  00 00 90 00 21 55 70 3F  66 30 52 2E 44 2A 77 4E  ....!Up?f0R.D*wN
0313FD70  2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
0313FD80  21 55 70 3F 66 30 52 2E  44 2A 77 4E 2D 2D 21 55  !Up?f0R.D*wN--!U
0313FD90  70 3F 66 30 52 2E 44 2A  77 4E 2D 2D 21 55 70 3F  p?f0R.D*wN--!Up?
0313FDA0  66 30 52 2E 44 2A 77 4E  2D 2D 21 55 70 3F 66 30  f0R.D*wN--!Up?f0
0313FDB0  52 2E 44 2A 77 4E 2D 2D  21 55 70 3F 66 30 52 2E  R.D*wN--!Up?f0R.
0313FDC0  44 2A 77 4E 2D 2D 21 55  70 3F 66 30 52 2E 44 2A  D*wN--!Up?f0R.D*
0313FDD0  77 4E 2D 2D 21 55 70 3F  66 30 52 2E 44 2A 77 4E  wN--!Up?f0R.D*wN
0313FDE0  2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
0313FDF0  21 55 70 3F 66 30 52 2E  44 2A 77 4E 2D 2D 21 55  !Up?f0R.D*wN--!U
0313FE00  70 3F 66 30 52 2E 44 2A  77 4E 2D 2D 21 55 70 3F  p?f0R.D*wN--!Up?
0313FE10  66 30 52 2E 44 2A 77 4E  2D 2D 21 55 70 3F 66 30  f0R.D*wN--!Up?f0
0313FE20  52 2E 44 2A 77 4E 2D 2D  21 55 70 3F 66 30 52 2E  R.D*wN--!Up?f0R.
0313FE30  44 2A 77 4E 2D 2D 21 55  70 3F 66 30 52 2E 44 2A  D*wN--!Up?f0R.D*
0313FE40  77 4E 2D 2D 21 55 70 3F  66 30 52 2E 44 2A 77 4E  wN--!Up?f0R.D*wN
0313FE50  2D 2D 21 55 70 3F 66 30  52 2E 44 2A 77 4E 2D 2D  --!Up?f0R.D*wN--
0313FE60  21 55 70 3F 97 89 E9 2B  37 CA 22 57 A3 A6 78 B4  !Up?—‰é+7Ê"W£¦x´
0313FE70  28 23 52 A1 B5 48 70 A8  18 08 44 9D A7 D2 C2 9E  (#R¡µHp¨..D.§ÒÂž
0313FE80  47 0F DD F5 C9 10 F0 EE  3F 1F 8C 11 02 DE E4 AA  G.ÝõÉ.ðî?.Œ..Þäª
0313FE90  24 AD EC 71 9F 3A 74 60  7C DF E0 FD F9 87 12 9C  $ìqŸ:t`|ßàýù‡.œ
0313FEA0  5B 8E 1C 9A B6 4C 0C 7A  BF B9 96 D6 C3 49 14 0B  [Ž.š¶L.z¿¹–ÖÃI..
0313FEB0  35 54 01 B2 42 C5 F1 D4  0D 98 BE 0A 1A F6 68 91  5T.²BÅñÔ.˜¾..öh‘
0313FEC0  3C 16 92 45 3D 38 94 D9  03 C4 04 D5 4E 6A 85 AC  <.’E=8”Ù.Ä.ÕNj…¬
0313FED0  E1 D0 E5 81 CF D8 BA 17  51 75 6B CD 7E 2A F7 C0  áÐå.ÏØº.QukÍ~*÷À
0313FEE0  4F E7 4D 63 CC 4A 25 D1  99 8F 6F B3 2D 77 B8 56  OçMcÌJ%Ñ™.o³-w¸V
0313FEF0  33 06 59 B0 D7 55 C1 8A  A5 A4 07 0E 2E 27 8B 7D  3.Y°×UÁŠ¥¤...'‹}
0313FF00  AB 69 00 CE DB 5A 1D EB  90 3B 93 6D FA E8 F2 5C  «i.ÎÛZ.ë.;“múèò\
0313FF10  29 76 79 FC C7 50 5F 88  6C ED DC 2C 1E 2F 7F A0  )vyüÇP_ˆlíÜ,./.·
0313FF20  26 C8 E2 15 30 A9 EF 5E  19 A2 39 32 6E 64 E6 FE  &Èâ.0©ï^.¢92ndæþ
0313FF30  40 05 80 CB FB B1 7B 53  65 43 09 73 31 DA 34 95  @.€Ëû±{SeC.s1Ú4•
0313FF40  F3 3E 62 FF E3 66 46 82  8D F4 D3 13 86 F8 BD EA  ó>bÿãfF‚.ôÓ.†ø½ê
0313FF50  84 36 1B 83 20 21 C6 4B  AF AE BC B7 58 5D 41 61  „6.ƒ·!ÆK¯®¼·X]Aa
0313FF60  67 BB 9B 72 00 00 00 00  00 00 00 00 00 00 00 00  g»›r............
```

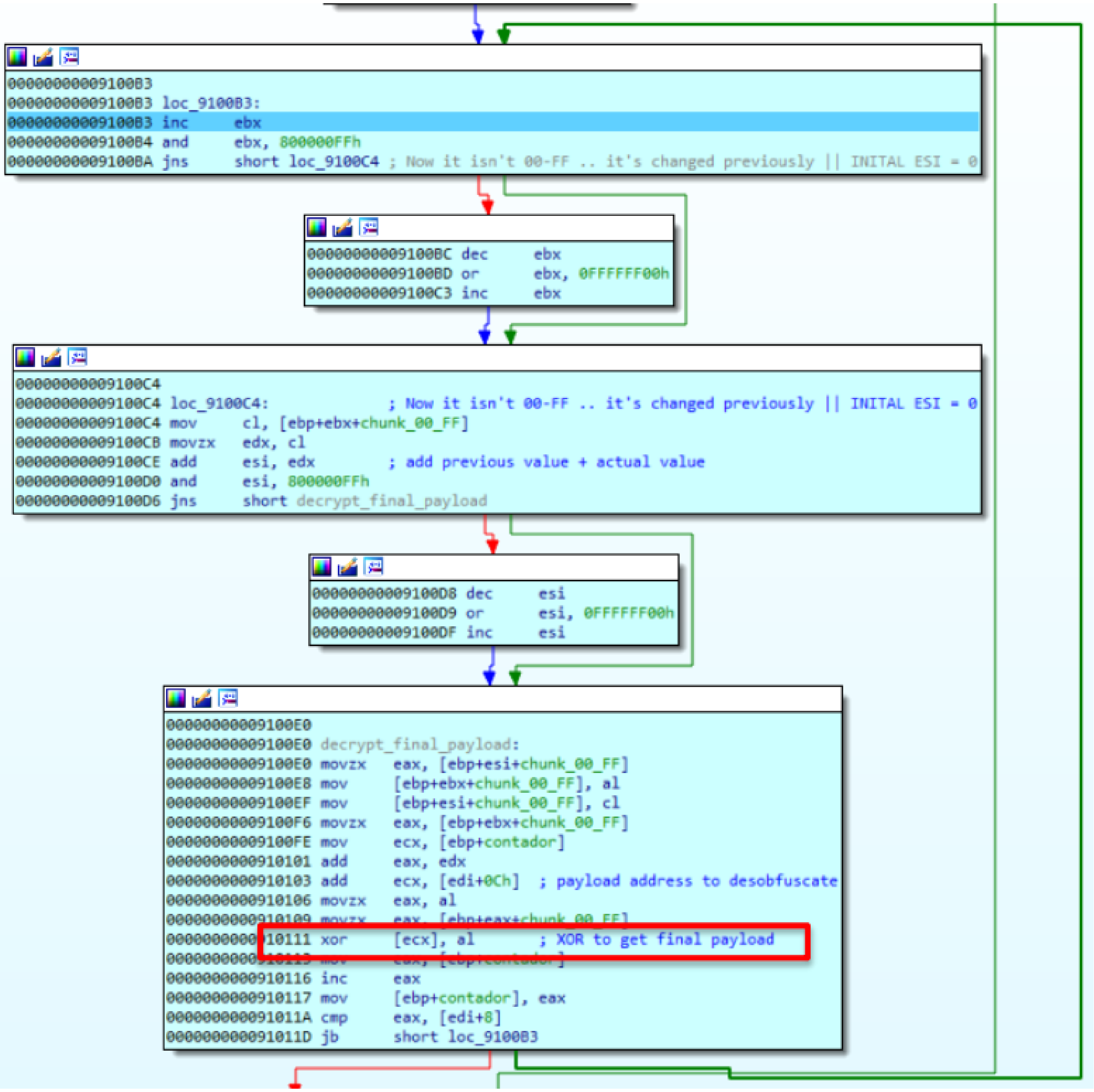
On the next step, the loader reads the initial argument, arg0, whose value is 0x900000 and points at the 4Kb block, which stores the absolute addresses to different API from Win32:



```
00000000009100A4 mov     edi, [ebp+arg_0] ; Arg0 value = 0x900000
00000000009100A7 xor     esi, esi
00000000009100A9 mov     [ebp+var_4], esi
00000000009100AC cmp     [edi+8], esi
00000000009100AF jbe     short loc_910122
```

After this, the decrypting process for the final payload begins. The decrypting process gets two values from the second block, exchanges and adds them, and the result serves as a final index to recover the element from the second block with which the xor will be achieved through the coded block.

This description of the decryption algorythm has been identified as the **RC4 algorythm**.



```
00000000009100B3
00000000009100B3 loc_9100B3:
00000000009100B3 inc     ebx
00000000009100B4 and     ebx, 800000FFh
00000000009100BA jns     short loc_9100C4 ; Now it isn't 00-FF .. it's changed previously || INITAL ESI = 0
```

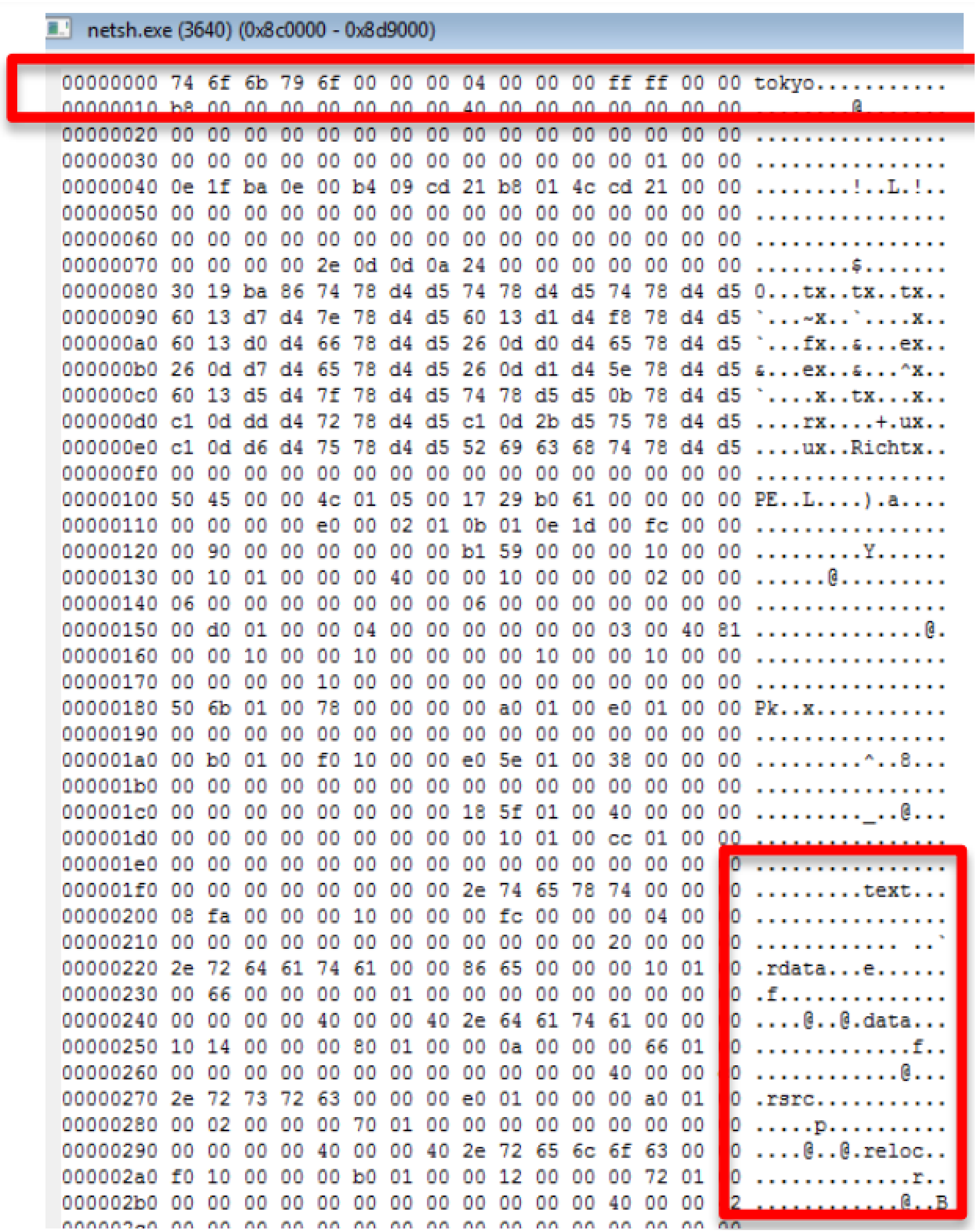
```
00000000009100BC dec     ebx
00000000009100BD or      ebx, 0FFFFFF00h
00000000009100C3 inc     ebx
```

```
00000000009100C4
00000000009100C4 loc_9100C4:             ; Now it isn't 00-FF .. it's changed previously || INITAL ESI = 0
00000000009100C4 mov     cl, [ebp+ebx+chunk_00_FF]
00000000009100CB movzx   edx, cl
00000000009100CE add     esi, edx        ; add previous value + actual value
00000000009100D0 and     esi, 800000FFh
00000000009100D6 jns     short decrypt_final_payload
```

```
00000000009100D8 dec     esi
00000000009100D9 or      esi, 0FFFFFF00h
00000000009100DF inc     esi
```

```
00000000009100E0
00000000009100E0 decrypt_final_payload:
00000000009100E0 movzx   eax, [ebp+esi+chunk_00_FF]
00000000009100E8 mov     [ebp+ebx+chunk_00_FF], al
00000000009100EF mov     [ebp+esi+chunk_00_FF], cl
00000000009100F6 movzx   eax, [ebp+ebx+chunk_00_FF]
00000000009100FE mov     ecx, [ebp+contador]
0000000000910101 add     eax, edx
0000000000910103 add     ecx, [edi+0Ch]  ; payload address to desobfuscate
0000000000910106 movzx   eax, al
0000000000910109 movzx   eax, [ebp+eax+chunk_00_FF]
0000000000910111 xor     [ecx], al       ; XOR to get final payload
0000000000910113 mov     eax, [ebp+contador]
0000000000910116 inc     eax
0000000000910117 mov     [ebp+contador], eax
000000000091011A cmp     eax, [edi+8]
000000000091011D jb      short loc_9100B3
```

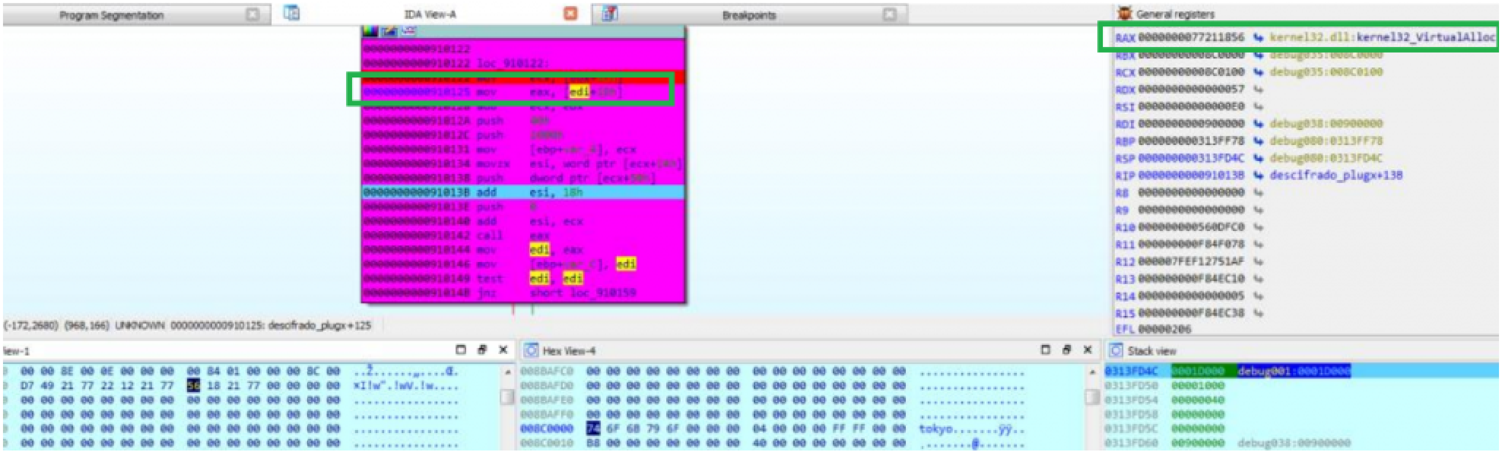
After the decryption process, we find a PE binary, as seen in the following image. In this case, the payload does not start with the traditional MZ header but the string “tokyo”:



netsh.exe (3640) (0x8c0000 - 0x8d9000)

```
00000000 74 6f 6b 79 6f 00 00 00 04 00 00 00 ff ff 00 00 tokyo...........
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 00 00 ........!..L.!..
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 2e 0d 0d 0a 24 00 00 00 00 00 00 00 ........$.......
00000080 30 19 ba 86 74 78 d4 d5 74 78 d4 d5 74 78 d4 d5 0...tx..tx..tx..
00000090 60 13 d7 d4 7e 78 d4 d5 60 13 d1 d4 f8 78 d4 d5 `...~x..`....x..
000000a0 60 13 d0 d4 66 78 d4 d5 26 0d d0 d4 65 78 d4 d5 `...fx..&...ex..
000000b0 26 0d d7 d4 65 78 d4 d5 26 0d d1 d4 5e 78 d4 d5 &...ex..&...^x..
000000c0 60 13 d5 d4 7f 78 d4 d5 74 78 d5 d5 0b 78 d4 d5 `....x..tx...x..
000000d0 c1 0d dd d4 72 78 d4 d5 c1 0d 2b d5 75 78 d4 d5 ....rx....+.ux..
000000e0 c1 0d d6 d4 75 78 d4 d5 52 69 63 68 74 78 d4 d5 ....ux..Richtx..
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000100 50 45 00 00 4c 01 05 00 17 29 b0 61 00 00 00 00 PE..L....).a....
00000110 00 00 00 00 e0 00 02 01 0b 01 0e 1d 00 fc 00 00 ................
00000120 00 90 00 00 00 00 00 00 b1 59 00 00 00 10 00 00 .........Y......
00000130 00 10 01 00 00 00 40 00 00 10 00 00 00 02 00 00 ......@.........
00000140 06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 ................
00000150 00 d0 01 00 00 04 00 00 00 00 00 00 03 00 40 81 ..............@.
00000160 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00 ................
00000170 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ................
00000180 50 6b 01 00 78 00 00 00 00 a0 01 00 e0 01 00 00 Pk..x...........
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 b0 01 00 f0 10 00 00 e0 5e 01 00 38 00 00 00 .........^..8...
000001b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001c0 00 00 00 00 00 00 00 00 18 5f 01 00 40 00 00 00 ........._..@...
000001d0 00 00 00 00 00 00 00 00 00 10 01 00 cc 01 00 00 ................
000001e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0  ................
000001f0 00 00 00 00 00 00 00 00 2e 74 65 78 74 00 00 0  .........text...
00000200 08 fa 00 00 00 10 00 00 00 fc 00 00 00 04 00 0  ................
00000210 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 0  ............ ..`
00000220 2e 72 64 61 74 61 00 00 86 65 00 00 00 10 01 0  .rdata...e......
00000230 00 66 00 00 00 00 01 00 00 00 00 00 00 00 00 0  .f..............
00000240 00 00 00 00 40 00 00 40 2e 64 61 74 61 00 00 0  ....@..@.data...
00000250 10 14 00 00 00 80 01 00 00 0a 00 00 00 66 01 0  .............f..
00000260 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 0  ............@...
00000270 2e 72 73 72 63 00 00 00 e0 01 00 00 00 a0 01 0  .rsrc...........
00000280 00 02 00 00 00 70 01 00 00 00 00 00 00 00 00 0  .....p..........
00000290 00 00 00 00 40 00 00 40 2e 72 65 6c 6f 63 00 0  ....@..@.reloc..
000002a0 f0 10 00 00 00 b0 01 00 00 12 00 00 00 72 01 0  .............r..
000002b0 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 2  ............@..B
000002c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Then, we see how it loads the VirtualAlloc absolute address (0x77211856) from the segment previously created:



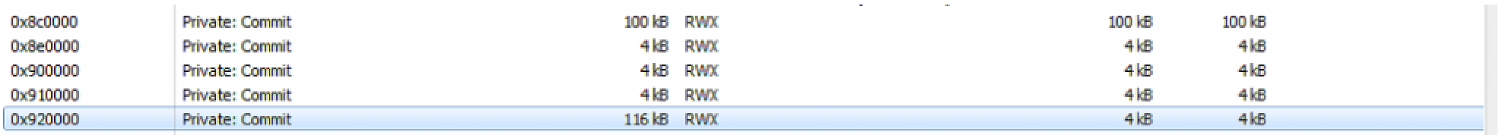
```
0000000000910122
0000000000910122 loc_910122:
0000000000910125 mov     eax, [edi+18h]
000000000091012A push    40h
000000000091012C push    1000h
0000000000910131 mov     [ebp+var_4], ecx
0000000000910134 movzx   esi, word ptr [ecx+14h]
0000000000910138 push    dword ptr [ecx+50h]
000000000091013B add     esi, 18h
000000000091013E push    0
0000000000910140 add     esi, ecx
0000000000910142 call    eax
0000000000910144 mov     edi, eax
0000000000910146 mov     [ebp+var_C], edi
0000000000910149 test    edi, edi
000000000091014B jnz     short loc_910159
```

```
RAX 0000000077211856 ↳ kernel32.dll:kernel32_VirtualAlloc
RCX 00000000000C0100 ↳ debug035:000C0100
RDX 0000000000000057 ↳
RSI 00000000000000E0 ↳
RDI 0000000000900000 ↳ debug038:00900000
RBP 000000000313FF78 ↳ debug080:0313FF78
RSP 000000000313FD4C ↳ debug080:0313FD4C
RIP 000000000091013B ↳ descifrado_plugx+13B
R8  0000000000000000 ↳
R9  0000000000000000 ↳
R10 00000000560DFC0 ↳
R11 000000000F84F078 ↳
R12 000007FEF12751AF ↳
R13 000000000F84EC10 ↳
R14 0000000000000005 ↳
R15 000000000F84EC38 ↳
EFL 00000206
```

```
0313FD4C  00010000 debug001:00010000
0313FD50  00001000
0313FD54  00000040
0313FD58  00000000
0313FD5C  00000000
0313FD60  00900000  debug038:00900000
```

(-172,2680) (968,166) UNKNOWN 0000000000910125: descifrado_plugx+125

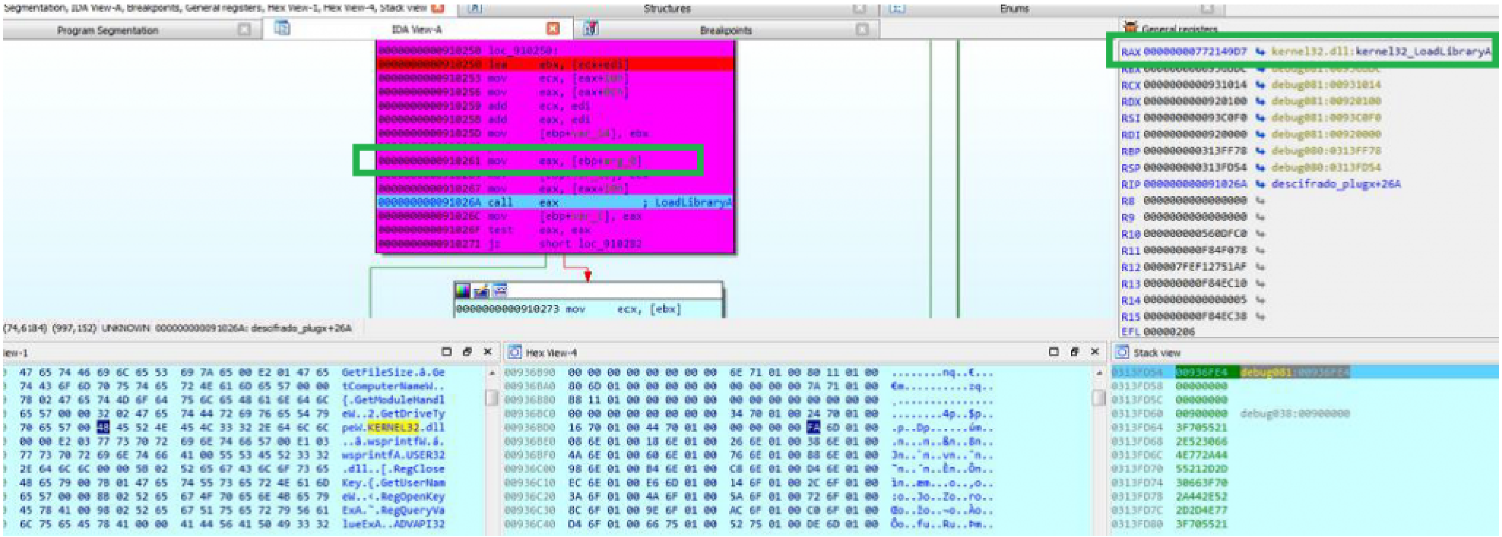
This creates another memory segment in the process netsh.exe with RWX licenses (that of 116Kb) which will be used to load the PE:



0x8c0000	Private: Commit	100 kB	RWX	100 kB	100 kB
0x8e0000	Private: Commit	4 kB	RWX	4 kB	4 kB
0x900000	Private: Commit	4 kB	RWX	4 kB	4 kB
0x910000	Private: Commit	4 kB	RWX	4 kB	4 kB
0x920000	Private: Commit	116 kB	RWX	4 kB	4 kB

In this new segment, it maps the binary using the virtual addresses as the regular Windows PE loader would do.

Then, it calls the API LoadLibraryA (it has the address since the DLL saved it in the memory segment) of the strings located in the mapped block:



```
0000000000910250 loc_910250:
0000000000910250 lea     ebx, [ecx+edi]
0000000000910253 mov     ecx, [eax+10h]
0000000000910256 mov     eax, [eax+0Ch]
0000000000910259 add     ecx, edi
000000000091025B add     eax, edi
000000000091025D mov     [ebp+var_14], ebx
0000000000910261 mov     eax, [ebp+arg_0]
0000000000910267 mov     eax, [eax+100h]
000000000091026A call    eax             ; LoadLibraryA
000000000091026C mov     [ebp+var_8], eax
000000000091026F test    eax, eax
0000000000910271 jz      short loc_910282

0000000000910273 mov     ecx, [ebx]
```

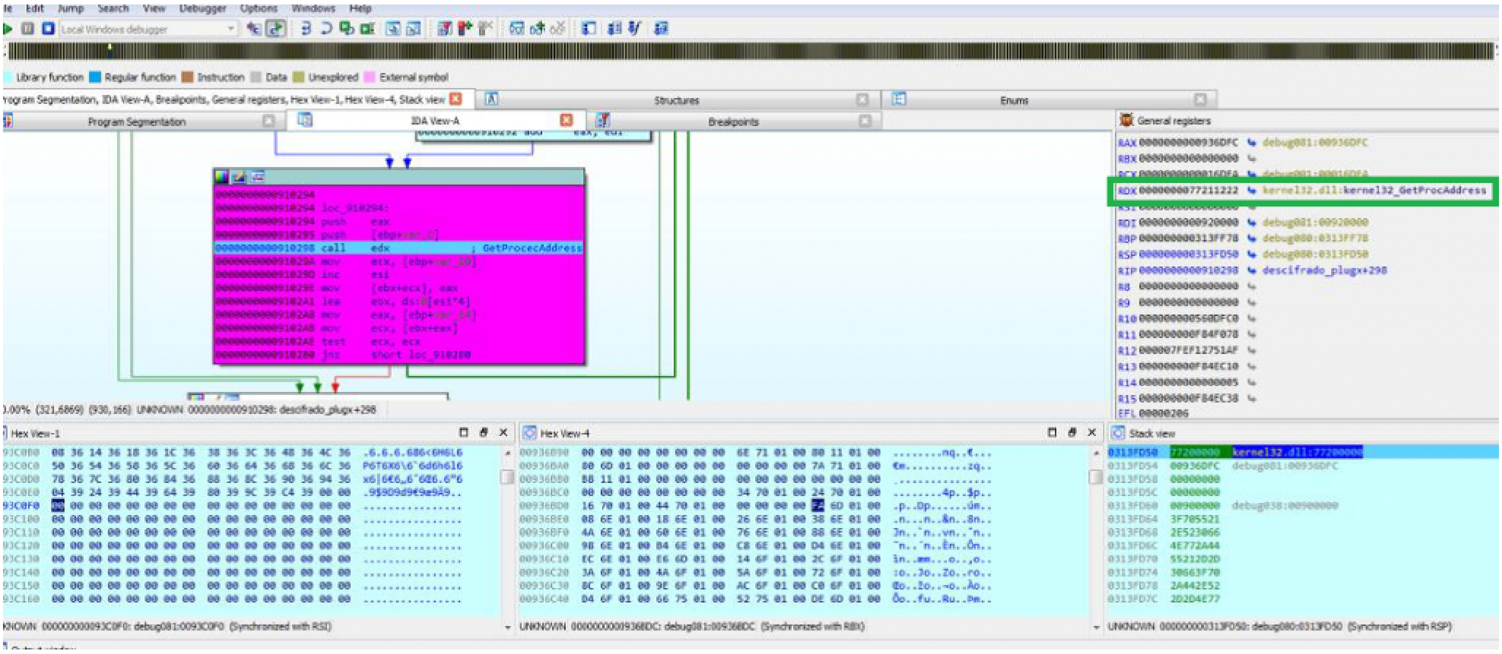
```
RAX 00000000772149D7 ↳ kernel32.dll:kernel32_LoadLibraryA
RCX 0000000000931014 ↳ debug081:00931014
RDX 0000000000920100 ↳ debug081:00920100
RSI 000000000093C0F0 ↳ debug081:0093C0F0
RDI 0000000000920000 ↳ debug081:00920000
RBP 000000000313FF78 ↳ debug080:0313FF78
RSP 000000000313FD54 ↳ debug080:0313FD54
RIP 000000000091026A ↳ descifrado_plugx+26A
R8  0000000000000000 ↳
R9  0000000000000000 ↳
R10 00000000560DFC0 ↳
R11 000000000F84F078 ↳
R12 000007FEF12751AF ↳
R13 000000000F84EC10 ↳
R14 0000000000000005 ↳
R15 000000000F84EC38 ↳
EFL 00000206
```

```
GetFileSize.ā.Ge
tComputerNameW..
{.GetModuleHandl
eW..2.GetDriveTy
peW.KERNEL32.dll
..ā.wsprintfW.ā.
wsprintfA.USER32
.dll..[.RegClose
Key.{.GetUserNam
eW..‹.RegOpenKey
ExA.˜.RegQueryVa
lueExA..ADVAPI32
```

```
0313FD54  00936FE4 debug081:00936FE4
0313FD58  00000000
0313FD5C  00000000
0313FD60  00900000  debug038:00900000
0313FD64  3F705521
0313FD68  2E523066
0313FD6C  4E772A44
0313FD70  55212D2D
0313FD74  30663F70
0313FD78  2A442E52
0313FD7C  2D2D4E77
0313FD80  3F705521
```

(74,6184) (997,152) UNKNOWN 000000000091026A: descifrado_plugx+26A

Then it calls GetProcAddress() to get the addresses of certain functions:



```
0000000000910294
0000000000910294 loc_910294:
0000000000910294 push    eax
0000000000910295 push    [ebp+var_C]
0000000000910298 call    edx              ; GetProcecAddress
000000000091029A mov     ecx, [ebp+var_10]
000000000091029D inc     esi
000000000091029E mov     [ebx+ecx], eax
00000000009102A1 lea     ebx, ds:0[esi*4]
00000000009102A8 mov     eax, [ebp+var_14]
00000000009102AB mov     ecx, [ebx+eax]
00000000009102AE test    ecx, ecx
00000000009102B0 jnz     short loc_9102B0
```

```
.00% (321,6869) (930,166) UNKNOWN 0000000000910298: descifrado_plugx+298
```

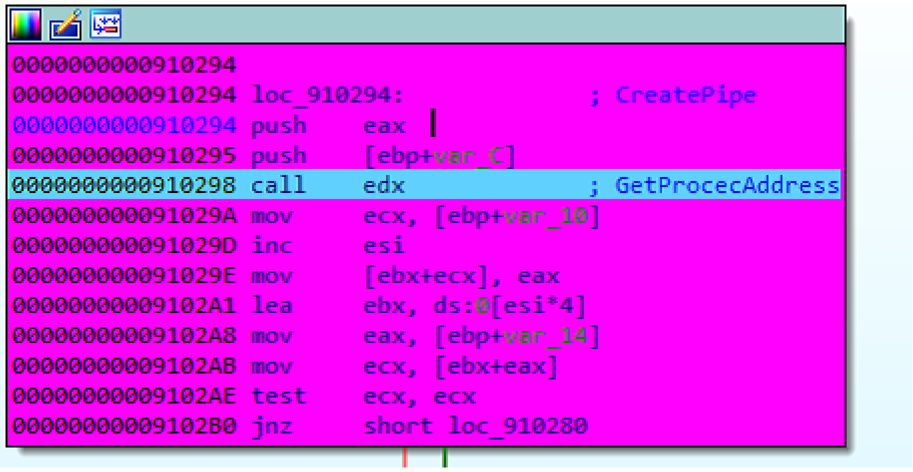
```
RAX 0000000000936DFC  debug001:00936DFC
RBX 0000000000000000
RDX 0000000077211222  kernel32.dll:kernel32_GetProcAddress
RDI 0000000000920000  debug001:00920000
RBP 000000000313FF78  debug080:0313FF78
RSP 000000000313FD50  debug080:0313FD50
RIP 0000000000910298  descifrado_plugx+298
R8  0000000000000000
R9  0000000000000000
R10 0000000000560DFC0
R11 000000000F84F078
R12 000007FEF12751AF
R13 000000000F84EC10
R14 0000000000000005
R15 000000000F84EC38
EFL 00000206
```

```
93C0B0  08 36 14 36 18 36 1C 36  38 36 3C 36 48 36 4C 36  .6.6.6.686<6H6L6
93C0C0  50 36 54 36 58 36 5C 36  60 36 64 36 68 36 6C 36  P6T6X6\6`6d6h6l6
93C0D0  78 36 7C 36 80 36 84 36  88 36 8C 36 90 36 94 36  x6|6€6„6ˆ6Œ6.6”6
93C0E0  04 39 24 39 44 39 64 39  80 39 9C 39 C4 39 00 00  .9$9D9d9€9œ9Ä9..
93C0F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
93C160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
KNOWN 000000000093C0F0: debug081:0093C0F0 (Synchronized with RSI)
```

```
00936B90  00 00 00 00 00 00 00 00  6E 71 01 00 80 11 01 00  ........nq..€...
00936BA0  80 6D 01 00 00 00 00 00  00 00 00 00 7A 71 01 00  €m..........zq..
00936BB0  88 11 01 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00936BC0  00 00 00 00 00 00 00 00  34 70 01 00 24 70 01 00  ........4p..$p..
00936BD0  16 70 01 00 44 70 01 00  00 00 00 00 FA 6D 01 00  .p..Dp......úm..
00936BE0  08 6E 01 00 18 6E 01 00  26 6E 01 00 38 6E 01 00  .n...n..&n..8n..
00936BF0  4A 6E 01 00 60 6E 01 00  76 6E 01 00 88 6E 01 00  Jn..`n..vn..ˆn..
00936C00  98 6E 01 00 B4 6E 01 00  C8 6E 01 00 D4 6E 01 00  ˜n..´n..Èn..Ôn..
00936C10  EC 6E 01 00 E6 6D 01 00  14 6F 01 00 2C 6F 01 00  ìn..æm...o..,o..
00936C20  3A 6F 01 00 4A 6F 01 00  5A 6F 01 00 72 6F 01 00  :o..Jo..Zo..ro..
00936C30  8C 6F 01 00 9E 6F 01 00  AC 6F 01 00 C0 6F 01 00  Œo..žo..¬o..Ào..
00936C40  D4 6F 01 00 66 75 01 00  52 75 01 00 DE 6D 01 00  Ôo..fu..Ru..Þm..
UNKNOWN 0000000000936BDC: debug081:00936BDC (Synchronized with RBX)
```

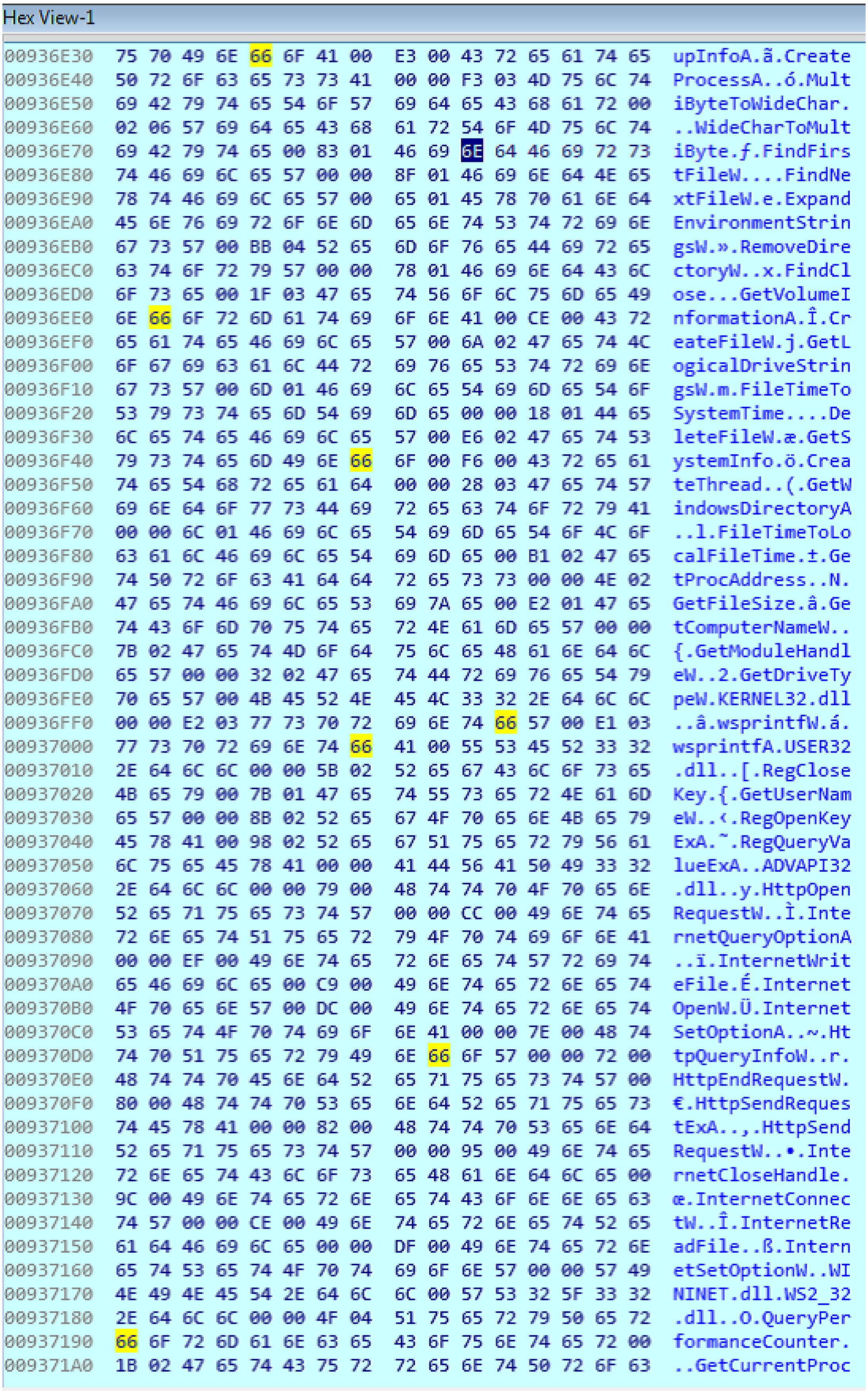
```
0313FD50  77200000  kernel32.dll:77200000
0313FD54  00936DFC  debug001:00936DFC
0313FD58  00000000
0313FD5C  00000000
0313FD60  00900000  debug038:00900000
0313FD64  3F705521
0313FD68  2E523066
0313FD6C  4E772A44
0313FD70  55212D2D
0313FD74  30663F70
0313FD78  2A442E52
0313FD7C  2D2D4E77
UNKNOWN 000000000313FD50: debug080:0313FD50 (Synchronized with RSP)
```

File Edit Jump Search View Debugger Options Windows Help
Local Windows debugger
Library function Regular function Instruction Data Unexplored External symbol
Program Segmentation, IDA View-A, Breakpoints, General registers, Hex View-1, Hex View-4, Stack view
Structures
Enums
Program Segmentation
IDA View-A
Breakpoints
General registers
Hex View-1
Hex View-4
Stack view



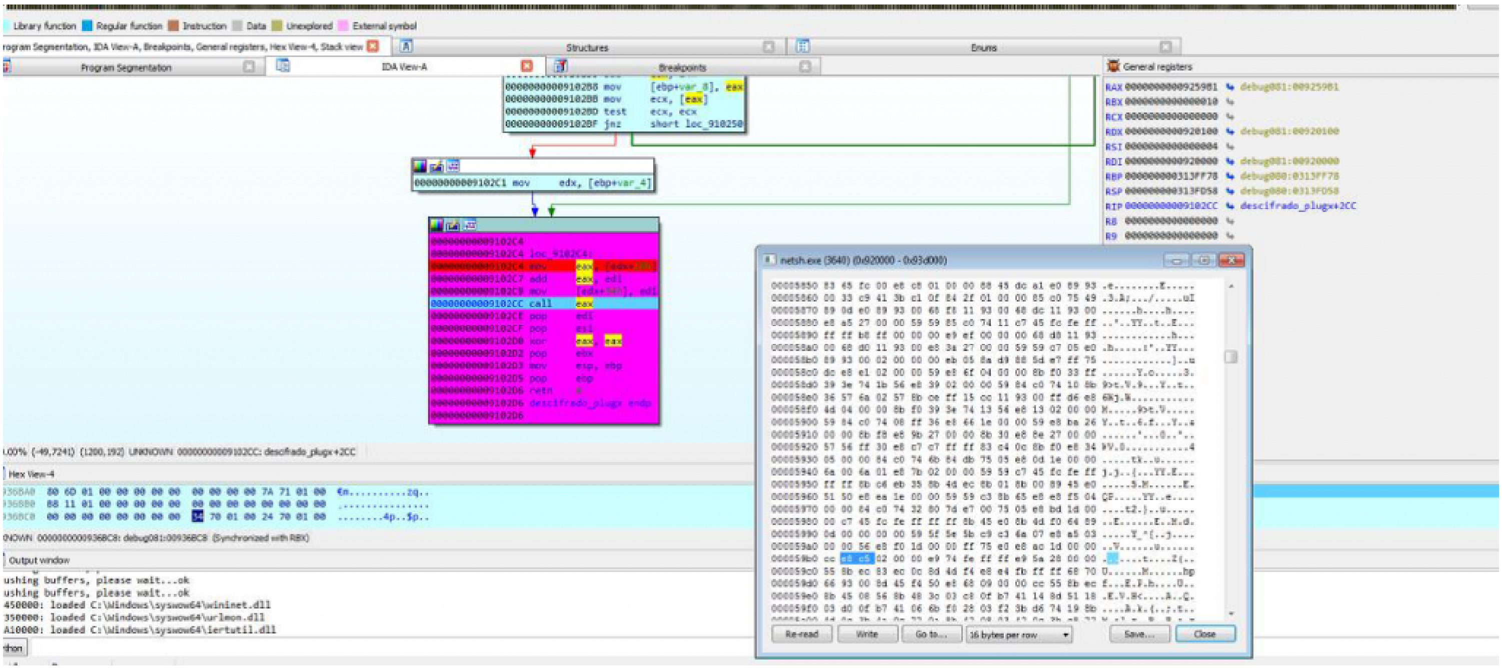
```
0000000000910294
0000000000910294 loc_910294:                 ; CreatePipe
0000000000910294 push    eax
0000000000910295 push    [ebp+var_C]
0000000000910298 call    edx                 ; GetProcecAddress
000000000091029A mov     ecx, [ebp+var_10]
000000000091029D inc     esi
000000000091029E mov     [ebx+ecx], eax
00000000009102A1 lea     ebx, ds:0[esi*4]
00000000009102A8 mov     eax, [ebp+var_14]
00000000009102AB mov     ecx, [ebx+eax]
00000000009102AE test    ecx, ecx
00000000009102B0 jnz     short loc_9102B0
```

Next, the libraries and functions block may be appreciated:



```
00936E30  75 70 49 6E 66 6F 41 00  E3 00 43 72 65 61 74 65  upInfoA.ã.Create
00936E40  50 72 6F 63 65 73 73 41  00 00 F3 03 4D 75 6C 74  ProcessA..ó.Mult
00936E50  69 42 79 74 65 54 6F 57  69 64 65 43 68 61 72 00  iByteToWideChar.
00936E60  02 06 57 69 64 65 43 68  61 72 54 6F 4D 75 6C 74  ..WideCharToMult
00936E70  69 42 79 74 65 00 83 01  46 69 6E 64 46 69 72 73  iByte.ƒ.FindFirs
00936E80  74 46 69 6C 65 57 00 00  8F 01 46 69 6E 64 4E 65  tFileW....FindNe
00936E90  78 74 46 69 6C 65 57 00  65 01 45 78 70 61 6E 64  xtFileW.e.Expand
00936EA0  45 6E 76 69 72 6F 6E 6D  65 6E 74 53 74 72 69 6E  EnvironmentStrin
00936EB0  67 73 57 00 BB 04 52 65  6D 6F 76 65 44 69 72 65  gsW.».RemoveDire
00936EC0  63 74 6F 72 79 57 00 00  78 01 46 69 6E 64 43 6C  ctoryW..x.FindCl
00936ED0  6F 73 65 00 1F 03 47 65  74 56 6F 6C 75 6D 65 49  ose...GetVolumeI
00936EE0  6E 66 6F 72 6D 61 74 69  6F 6E 41 00 CE 00 43 72  nformationA.Î.Cr
00936EF0  65 61 74 65 46 69 6C 65  57 00 6A 02 47 65 74 4C  eateFileW.j.GetL
00936F00  6F 67 69 63 61 6C 44 72  69 76 65 53 74 72 69 6E  ogicalDriveStrin
00936F10  67 73 57 00 6D 01 46 69  6C 65 54 69 6D 65 54 6F  gsW.m.FileTimeTo
00936F20  53 79 73 74 65 6D 54 69  6D 65 00 00 18 01 44 65  SystemTime....De
00936F30  6C 65 74 65 46 69 6C 65  57 00 E6 02 47 65 74 53  leteFileW.æ.GetS
00936F40  79 73 74 65 6D 49 6E 66  6F 00 F6 00 43 72 65 61  ystemInfo.ö.Crea
00936F50  74 65 54 68 72 65 61 64  00 00 28 03 47 65 74 57  teThread..(.GetW
00936F60  69 6E 64 6F 77 73 44 69  72 65 63 74 6F 72 79 41  indowsDirectoryA
00936F70  00 00 6C 01 46 69 6C 65  54 69 6D 65 54 6F 4C 6F  ..l.FileTimeToLo
00936F80  63 61 6C 46 69 6C 65 54  69 6D 65 00 B1 02 47 65  calFileTime.±.Ge
00936F90  74 50 72 6F 63 41 64 64  72 65 73 73 00 00 4E 02  tProcAddress..N.
00936FA0  47 65 74 46 69 6C 65 53  69 7A 65 00 E2 01 47 65  GetFileSize.â.Ge
00936FB0  74 43 6F 6D 70 75 74 65  72 4E 61 6D 65 57 00 00  tComputerNameW..
00936FC0  7B 02 47 65 74 4D 6F 64  75 6C 65 48 61 6E 64 6C  {.GetModuleHandl
00936FD0  65 57 00 00 32 02 47 65  74 44 72 69 76 65 54 79  eW..2.GetDriveTy
00936FE0  70 65 57 00 4B 45 52 4E  45 4C 33 32 2E 64 6C 6C  peW.KERNEL32.dll
00936FF0  00 00 E2 03 77 73 70 72  69 6E 74 66 57 00 E1 03  ..â.wsprintfW.á.
00937000  77 73 70 72 69 6E 74 66  41 00 55 53 45 52 33 32  wsprintfA.USER32
00937010  2E 64 6C 6C 00 00 5B 02  52 65 67 43 6C 6F 73 65  .dll..[.RegClose
00937020  4B 65 79 00 7B 01 47 65  74 55 73 65 72 4E 61 6D  Key.{.GetUserNam
00937030  65 57 00 00 8B 02 52 65  67 4F 70 65 6E 4B 65 79  eW..‹.RegOpenKey
00937040  45 78 41 00 98 02 52 65  67 51 75 65 72 79 56 61  ExA.˜.RegQueryVa
00937050  6C 75 65 45 78 41 00 00  41 44 56 41 50 49 33 32  lueExA..ADVAPI32
00937060  2E 64 6C 6C 00 00 79 00  48 74 74 70 4F 70 65 6E  .dll..y.HttpOpen
00937070  52 65 71 75 65 73 74 57  00 00 CC 00 49 6E 74 65  RequestW..Ì.Inte
00937080  72 6E 65 74 51 75 65 72  79 4F 70 74 69 6F 6E 41  rnetQueryOptionA
00937090  00 00 EF 00 49 6E 74 65  72 6E 65 74 57 72 69 74  ..ï.InternetWrit
009370A0  65 46 69 6C 65 00 C9 00  49 6E 74 65 72 6E 65 74  eFile.É.Internet
009370B0  4F 70 65 6E 57 00 DC 00  49 6E 74 65 72 6E 65 74  OpenW.Ü.Internet
009370C0  53 65 74 4F 70 74 69 6F  6E 41 00 00 7E 00 48 74  SetOptionA..~.Ht
009370D0  74 70 51 75 65 72 79 49  6E 66 6F 57 00 00 72 00  tpQueryInfoW..r.
009370E0  48 74 74 70 45 6E 64 52  65 71 75 65 73 74 57 00  HttpEndRequestW.
009370F0  80 00 48 74 74 70 53 65  6E 64 52 65 71 75 65 73  €.HttpSendReques
00937100  74 45 78 41 00 00 82 00  48 74 74 70 53 65 6E 64  tExA..‚.HttpSend
00937110  52 65 71 75 65 73 74 57  00 00 95 00 49 6E 74 65  RequestW..•.Inte
00937120  72 6E 65 74 43 6C 6F 73  65 48 61 6E 64 6C 65 00  rnetCloseHandle.
00937130  9C 00 49 6E 74 65 72 6E  65 74 43 6F 6E 6E 65 63  œ.InternetConnec
00937140  74 57 00 00 CE 00 49 6E  74 65 72 6E 65 74 52 65  tW..Î.InternetRe
00937150  61 64 46 69 6C 65 00 00  DF 00 49 6E 74 65 72 6E  adFile..ß.Intern
00937160  65 74 53 65 74 4F 70 74  69 6F 6E 57 00 00 57 49  etSetOptionW..WI
00937170  4E 49 4E 45 54 2E 64 6C  6C 00 57 53 32 5F 33 32  NINET.dll.WS2_32
00937180  2E 64 6C 6C 00 00 4F 04  51 75 65 72 79 50 65 72  .dll..O.QueryPer
00937190  66 6F 72 6D 61 6E 63 65  43 6F 75 6E 74 65 72 00  formanceCounter.
009371A0  1B 02 47 65 74 43 75 72  72 65 6E 74 50 72 6F 63  ..GetCurrentProc
```

After the correct mapping and having loaded the necessary libraries for its proper functioning, it calls EAX to run the decrypted and mapped payload:



```
0000000000910288 mov     [ebp+var_8], eax
000000000091028B mov     ecx, [eax]
000000000091028D test    ecx, ecx
000000000091028F jnz     short loc_910250

00000000009102C1 mov     edx, [ebp+var_4]

00000000009102C4
00000000009102C4 loc_9102C4:
00000000009102C4 mov     eax, [edx+28h]
00000000009102C7 add     eax, edi
00000000009102C9 mov     [edx+34h], edi
00000000009102CC call    eax
00000000009102CE pop     edi
00000000009102CF pop     esi
00000000009102D0 xor     eax, eax
00000000009102D2 pop     ebx
00000000009102D3 mov     esp, ebp
00000000009102D5 pop     ebp
00000000009102D6 retn    0
00000000009102D6 descifrado_plugx endp
00000000009102D6
```

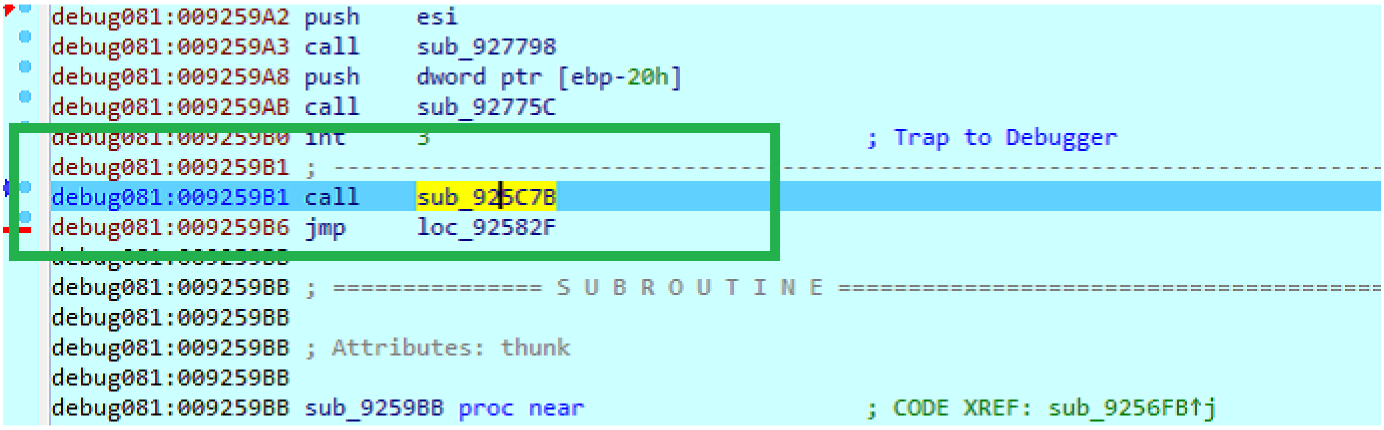
```
RAX 0000000000925981 ↳ debug081:00925981
RBX 0000000000000010 ↳
RCX 0000000000000000 ↳
RDX 0000000000920100 ↳ debug081:00920100
RSI 0000000000000004 ↳
RDI 0000000000920000 ↳ debug081:00920000
RBP 000000000313FF78 ↳ debug080:0313FF78
RSP 000000000313FD58 ↳ debug080:0313FD58
RIP 00000000009102CC ↳ descifrado_plugx+2CC
R8  0000000000000000 ↳
R9  0000000000000000 ↳
```

```
.00% (-49,7241) (1200,192) UNKNOWN 00000000009102CC: descifrado_plugx+2CC
```

```
36BA0  00 6D 01 00 00 00 00 00  00 00 00 00 7A 71 01 00  €n..........zq..
36BB0  88 11 01 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36BC8  00 00 00 00 00 00 00 00  34 70 01 00 24 70 01 00  ........4p..$p..
```

```
NOWN 00000000009368C8: debug081:009368C8 (Synchronized with RBX)
```

```
ushing buffers, please wait...ok
ushing buffers, please wait...ok
450000: loaded C:\Windows\syswow64\wininet.dll
350000: loaded C:\Windows\syswow64\urlmon.dll
A10000: loaded C:\Windows\syswow64\iertutil.dll
```

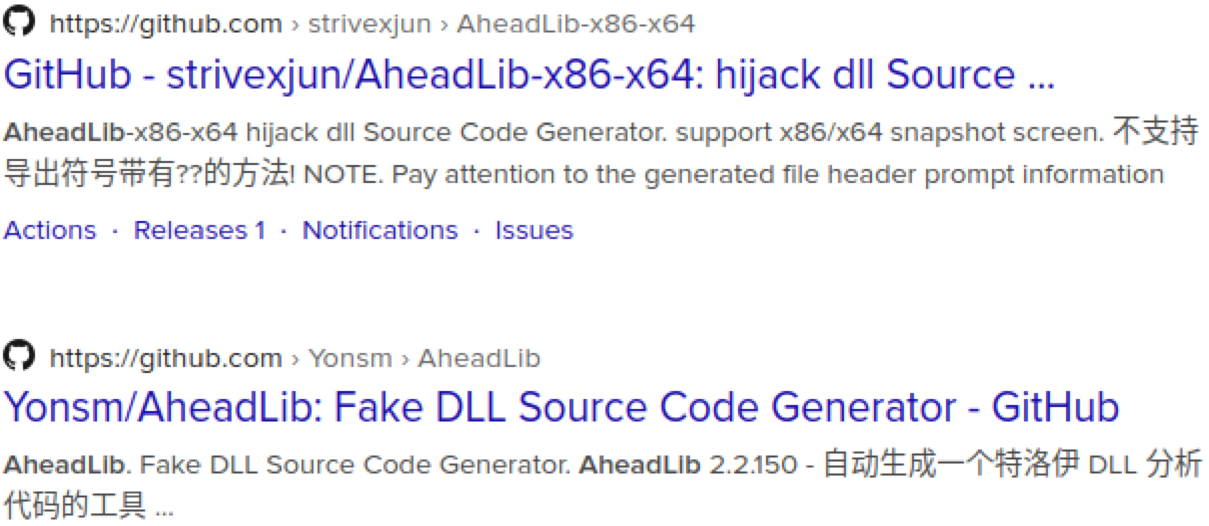


```
debug081:009259A2 push    esi
debug081:009259A3 call    sub_927798
debug081:009259A8 push    dword ptr [ebp-20h]
debug081:009259AB call    sub_92775C
debug081:009259B0 int     3                          ; Trap to Debugger
debug081:009259B1 ; ---------------------------------------------------------------
debug081:009259B1 call    sub_925C7B
debug081:009259B6 jmp     loc_92582F
debug081:009259BB
debug081:009259BB ; =============== S U B R O U T I N E ===========================
debug081:009259BB
debug081:009259BB ; Attributes: thunk
debug081:009259BB
debug081:009259BB sub_9259BB proc near               ; CODE XREF: sub_9256FB↑j
```

To summarize, this article goes through the process followed in memory after executing the Creative Cloud application until deploying TokyoX in memory. This DLL sideloading style is often linked to APT groups whose attribution is also linked to China, however being a known technique as it is, we are not able to consider any feasible attribution at the moment.

As reviewed at the beginning of the article, what we have named as “TokyoX” has not been identified as a known malware so far (at least, with the sources that we have).

Additionally, at some point of the analysis we identified a tool used by this group for the creation of version.dll, which pretends to be a Windows DLL located in SysWOW/System32. The string “AheadLib” found among the code of the malicious version.dll drew our attention, and we quickly found two chinese (casually or not) GitHub repositories with the source code of some tool called AheadLib.



https://github.com › strivexjun › AheadLib-x86-x64

GitHub - strivexjun/AheadLib-x86-x64: hijack dll Source ...

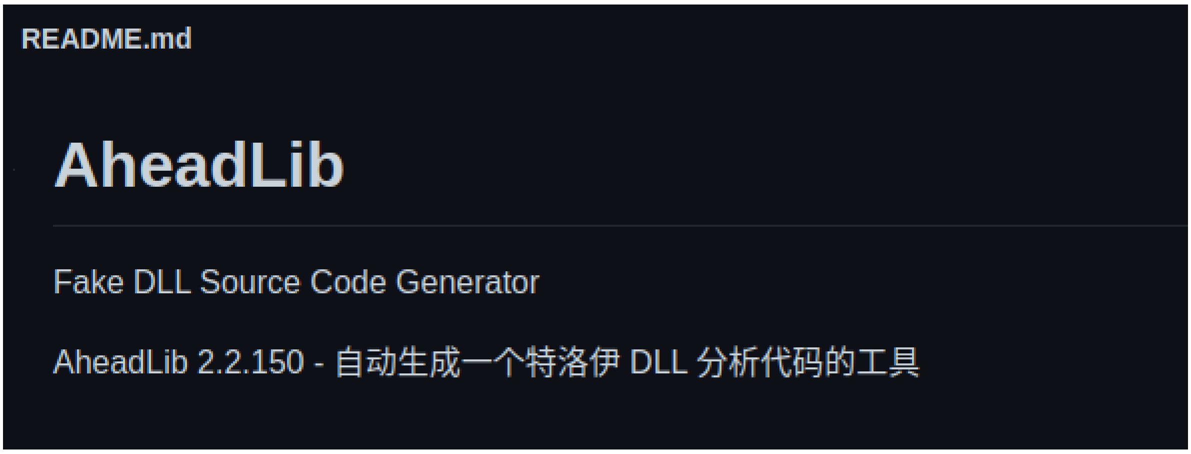
AheadLib-x86-x64 hijack dll Source Code Generator. support x86/x64 snapshot screen. 不支持导出符号带有??的方法! NOTE. Pay attention to the generated file header prompt information

Actions · Releases 1 · Notifications · Issues

https://github.com › Yonsm › AheadLib

Yonsm/AheadLib: Fake DLL Source Code Generator - GitHub

AheadLib. Fake DLL Source Code Generator. **AheadLib** 2.2.150 - 自动生成一个特洛伊 DLL 分析代码的工具 ...



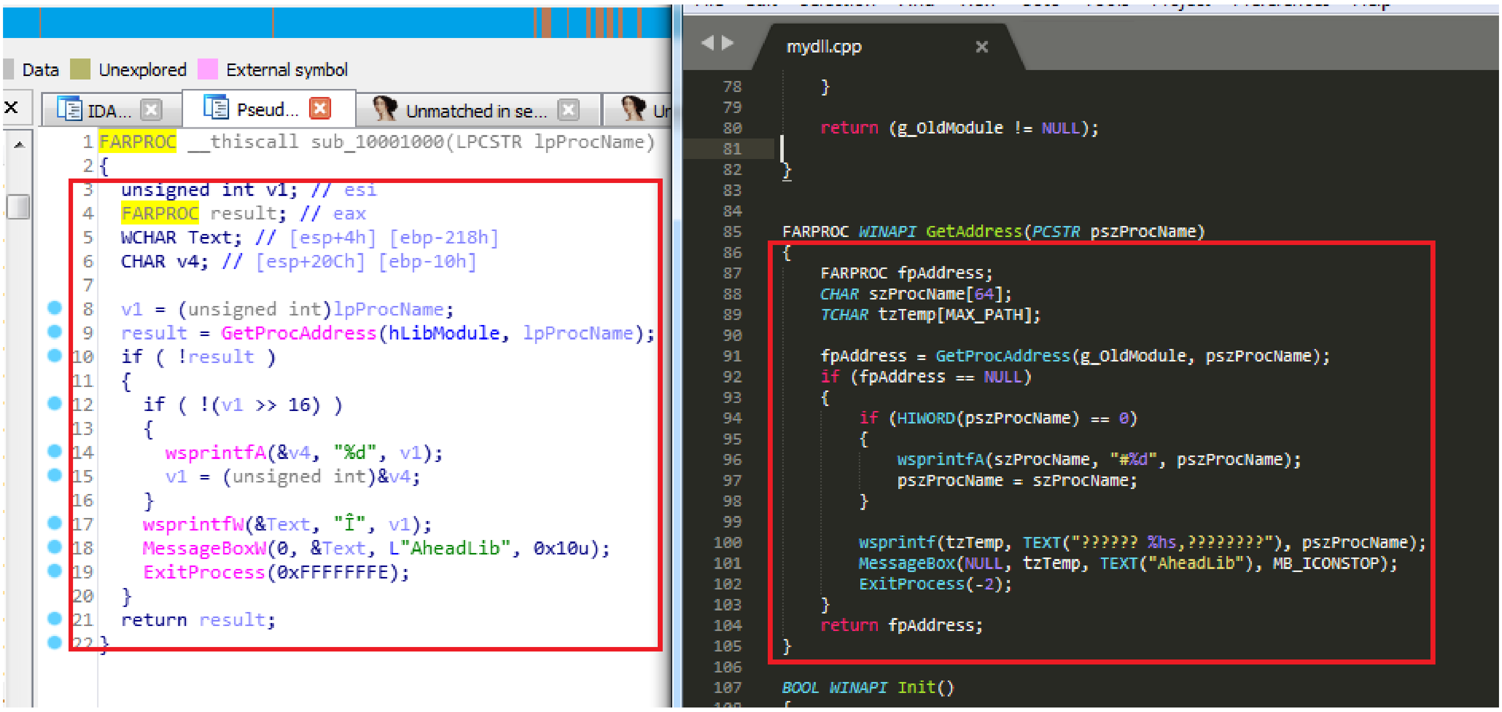
README.md

AheadLib

Fake DLL Source Code Generator

AheadLib 2.2.150 - 自动生成一个特洛伊 DLL 分析代码的工具

Basically, this tool will allow you to create a C++ source code file, implementing a DLL with the same exported functions as a given DLL. For the purpose of the current analysis we generated a source code file using this tool and giving the legitimate version.dll as input.



```
FARPROC __thiscall sub_10001000(LPCSTR lpProcName)
{
  unsigned int v1; // esi
  FARPROC result; // eax
  WCHAR Text; // [esp+4h] [ebp-218h]
  CHAR v4; // [esp+20Ch] [ebp-10h]

  v1 = (unsigned int)lpProcName;
  result = GetProcAddress(hLibModule, lpProcName);
  if ( !result )
  {
    if ( !(v1 >> 16) )
    {
      wsprintfA(&v4, "%d", v1);
      v1 = (unsigned int)&v4;
    }
    wsprintfW(&Text, "Î", v1);
    MessageBoxW(0, &Text, L"AheadLib", 0x10u);
    ExitProcess(0xFFFFFFFE);
  }
  return result;
}
```

```
    }

    return (g_OldModule != NULL);

}


FARPROC WINAPI GetAddress(PCSTR pszProcName)
{
    FARPROC fpAddress;
    CHAR szProcName[64];
    TCHAR tzTemp[MAX_PATH];

    fpAddress = GetProcAddress(g_OldModule, pszProcName);
    if (fpAddress == NULL)
    {
        if (HIWORD(pszProcName) == 0)
        {
            wsprintfA(szProcName, "#%d", pszProcName);
            pszProcName = szProcName;
        }

        wsprintf(tzTemp, TEXT("?????? %hs,????????"), pszProcName);
        MessageBox(NULL, tzTemp, TEXT("AheadLib"), MB_ICONSTOP);
        ExitProcess(-2);
    }
    return fpAddress;
}

BOOL WINAPI Init()
```

In the shown screenshot we can see on the left side the pseudocode generated by IDA Pro while analyzing the malicious version.dll sample. On the right side, we can observe the source code automatically generated by AheadLib using the legitimate version.dll as input. Even though the exported functions are not shown in the previous image, we can appreciate how there is a perfect match between both snippets.

We will post soon an analysis of the final “TokyoX” RAT and its capacities.

IOCs

- 382b3d3bb1be4f14dbc1e82a34946a52795288867ed86c6c43e4f981729be4fc
- 31.192.107[.]187:443