F-Secure Cyber Security Competition 2019

[Binary and Web-Application Exploitation]

1) **Secure Login:**

the reverse loop is actually building base64 from down up
the base64 is the flag: ZnNzdXBlcnNlY3VyZWxvZ2luY3liZXI=

Challenge: <DOWNLOAD>

2) **Bleep blop [WEBAPP] :**

Look for robot.txt then look what page is allowed.

go to that page

You find the flag in a comment section

Challenge: <DOWNLOAD>

3) **Overflowed**:

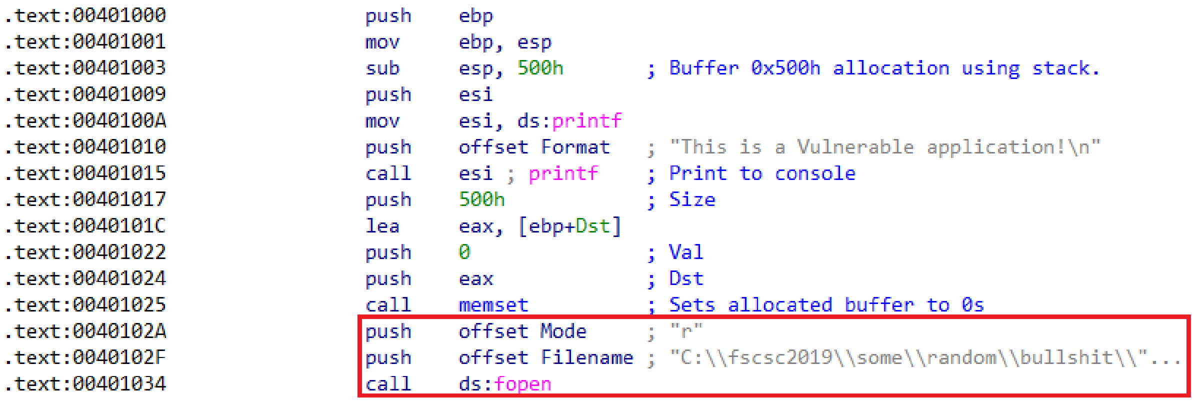
Challenge: <DOWNLOAD>

This challenge contains a vulnerable application (VulnApp.exe) and an exploit file (bad_food.bin). The vulnerable application is susceptible to stack based buffer overflow which could lead to remote code execution.

The vulnerable application reads 2048 bytes (0x800h) of data from a file and store it to a buffer that is only allocated for 1280 bytes (0x500h). This will write the exploit code in stack and eventually overwriting the return addresses in stack eventually corrupting the execution flow.

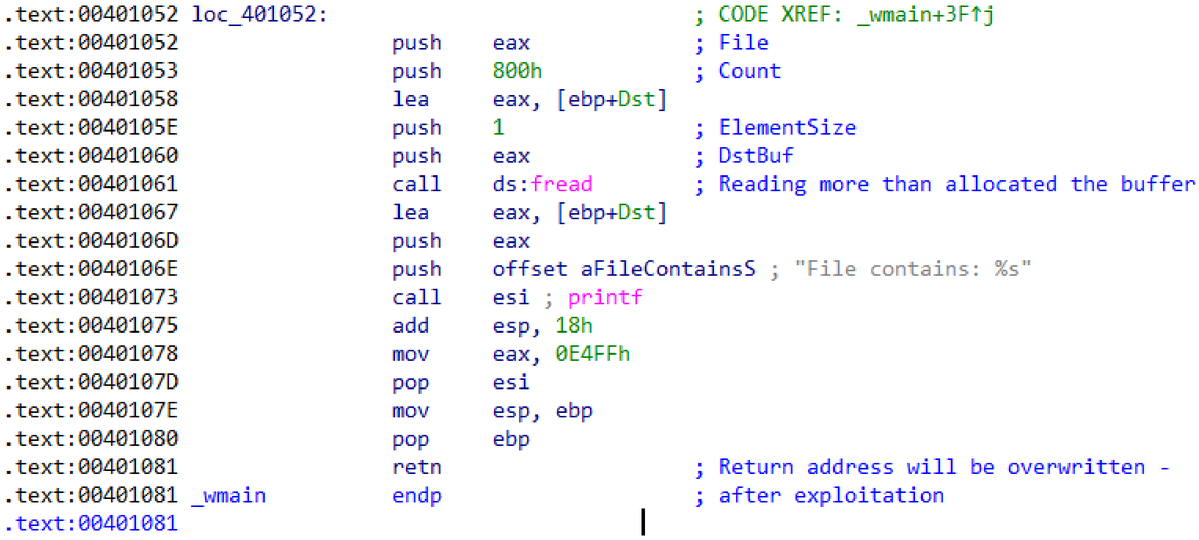
How to solve:

Upon checking the main function of the file, we can see that it opens a file from (C:\fscsc2019\some\random\bullshit\badfood.bin).



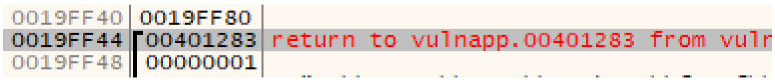
```
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 500h        ; Buffer 0x500h allocation using stack.
.text:00401009                 push    esi
.text:0040100A                 mov     esi, ds:printf
.text:00401010                 push    offset Format    ; "This is a Vulnerable application!\n"
.text:00401015                 call    esi ; printf     ; Print to console
.text:00401017                 push    500h             ; Size
.text:0040101C                 lea     eax, [ebp+Dst]
.text:00401022                 push    0                ; Val
.text:00401024                 push    eax              ; Dst
.text:00401025                 call    memset           ; Sets allocated buffer to 0s
.text:0040102A                 push    offset Mode      ; "r"
.text:0040102F                 push    offset Filename ; "C:\\fscsc2019\\some\\random\\bullshit\\"...
.text:00401034                 call    ds:fopen
```

It will procced reading the file and write it in a allocated buffer. Overflow is only archived if the input file is more than 1280 bytes (0x500h).



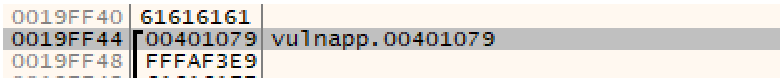
```
.text:00401052 loc_401052:                             ; CODE XREF: _wmain+3F↑j
.text:00401052                 push    eax             ; File
.text:00401053                 push    800h            ; Count
.text:00401058                 lea     eax, [ebp+Dst]
.text:0040105E                 push    1               ; ElementSize
.text:00401060                 push    eax             ; DstBuf
.text:00401061                 call    ds:fread        ; Reading more than allocated the buffer
.text:00401067                 lea     eax, [ebp+Dst]
.text:0040106D                 push    eax
.text:0040106E                 push    offset aFileContainsS ; "File contains: %s"
.text:00401073                 call    esi ; printf
.text:00401075                 add     esp, 18h
.text:00401078                 mov     eax, 0E4FFh
.text:0040107D                 pop     esi
.text:0040107E                 mov     esp, ebp
.text:00401080                 pop     ebp
.text:00401081                 retn                    ; Return address will be overwritten -
.text:00401081 _wmain          endp                    ; after exploitation
.text:00401081
```

Before corruption:



```
0019FF40 | 0019FF80 |
0019FF44 |00401283 | return to vulnapp.00401283 from vulr
0019FF48 |00000001 |
```

After corruption:



```
0019FF40 | 61616161 |
0019FF44 |00401079 | vulnapp.00401079
0019FF48 |FFFAF3E9 |
```

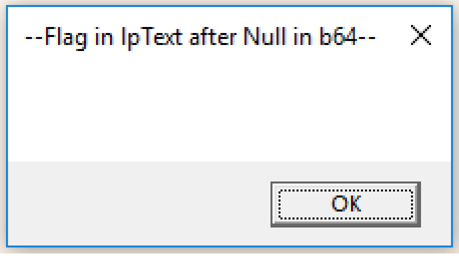
The return address was overflowed and was overwritten with an exploit defined address (0x0401079).

After executing 0x401081h (RETN), the return address is redirected to 0x401079 which a jump instruction to stack where the exploit code is located.

00401079 | FFE4 | jmp esp (JUMP to stack)

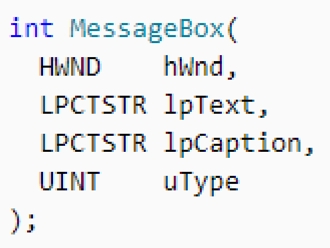
Exploit code:

The exploit code will prompt a message box containing a hint about the flag.

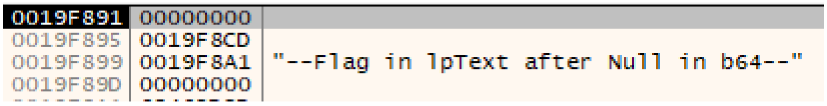


--Flag in lpText after Null in b64--

Looking at MessageBoxA parameter in MSDN, lpText push 3rd in stack (from bottom to top).

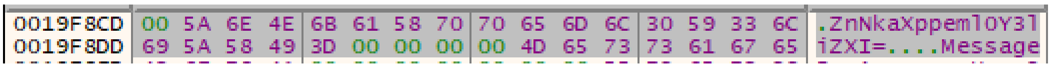


```
int MessageBox(
  HWND    hWnd,
  LPCTSTR lpText,
  LPCTSTR lpCaption,
  UINT    uType
);
```



```
0019F891 00000000
0019F895 0019F8CD
0019F899 0019F8A1  "--Flag in lpText after Null in b64--"
0019F89D 00000000
```

Inspecting 0019F8CD, we are able to locate a base 64 string : ZnNkaXppeml0Y3liZXI=



```
0019F8CD 00 5A 6E 4E 6B 61 58 70 70 65 6D 6C 30 59 33 6C .ZnNkaXppeml0Y3l
0019F8DD 69 5A 58 49 3D 00 00 00 00 4D 65 73 73 61 67 65 iZXI=....Message
```

Flag:

fsdizizitcyber

4) **Shellc**:

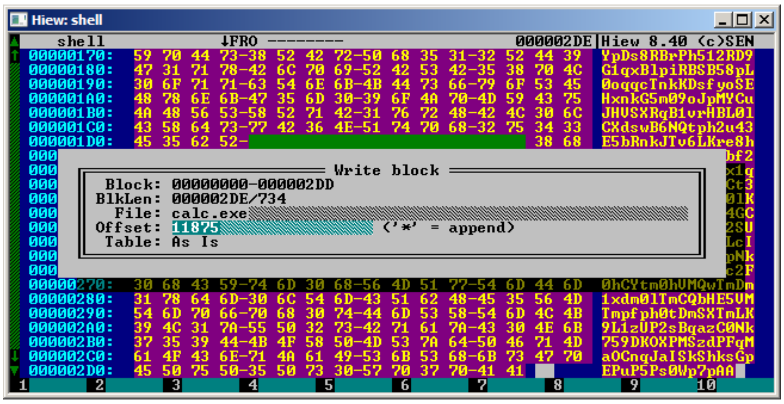
Challenge: <DOWNLOAD>

The participant need to load the alphanumeric shellcode to memory in order to properly debug. The shellcode looks like text but it is alphanumeric encoded shell.

In order for us to solve, we need to load the shellcode to memory in order for us to proceed with debugging. We will use a 32bit calc.exe and copy the shellcode in the **entry point**. This way when we load the calc.exe, we will be able to debug the shellcode on entry point.

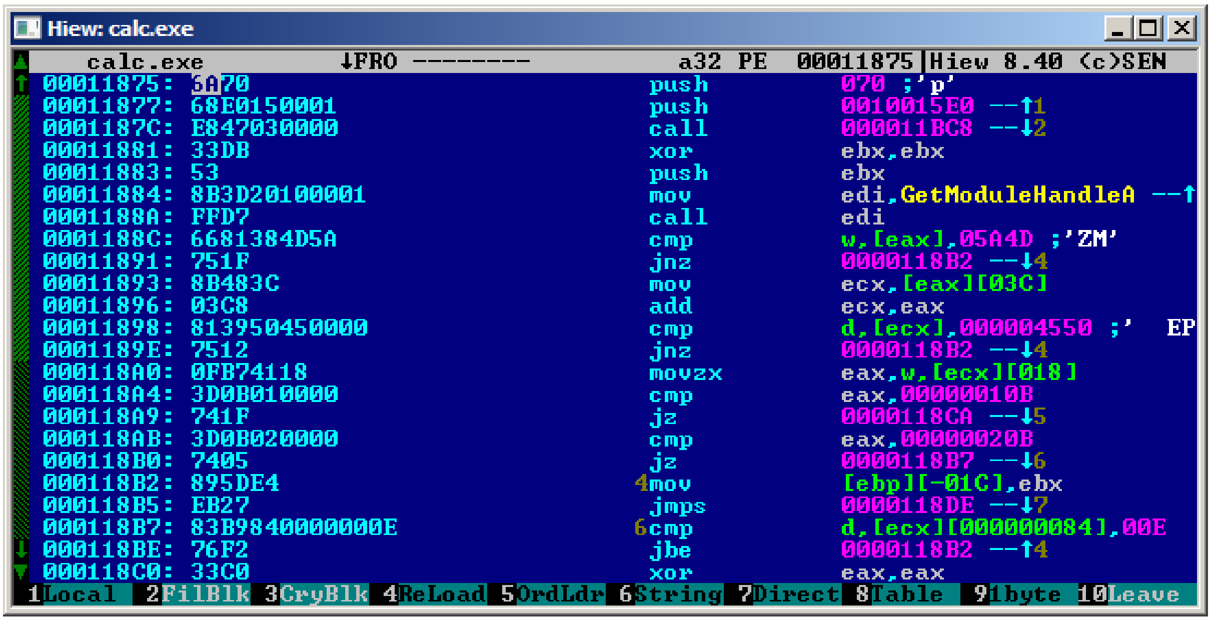
We need to copy the shellcode to a working executable file for us to proceed with debugging.

In the image below, we are selecting all data from shellcode which we will copy/overwrite to calc.exe entry point.



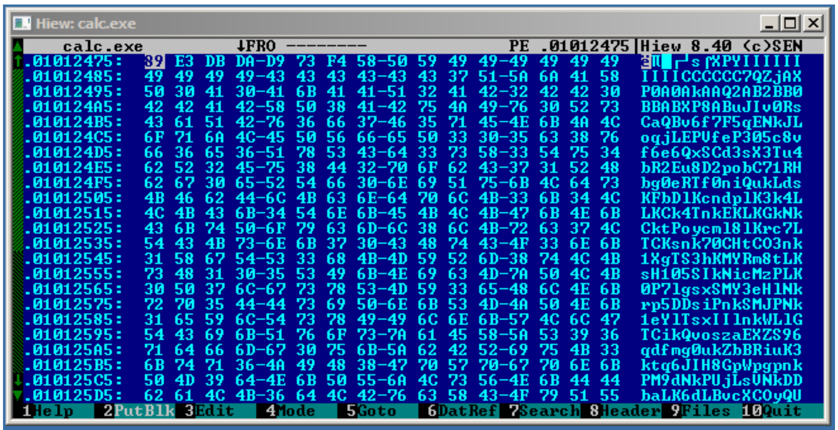
```
 shell             ↓FRO --------              000002DE|Hiew 8.40 (c)SEN
00000170:  59 70 44 73-38 52 42 72-50 68 35 31-32 52 44 39  YpDs8RBrPh512RD9
00000180:  47 31 71 78-42 6C 70 69-52 42 53 42-35 38 70 4C  G1qxBlpiRBSB58pL
00000190:  30 6F 71 71-63 54 6E 6B-4B 44 73 66-79 6F 53 45  0oqqcTnkKDsfyoSE
000001A0:  48 78 6E 6B-47 35 6D 30-39 6F 4A 70-4D 59 43 75  HxnkG5m09oJpMYCu
000001B0:  4A 48 56 53-58 52 71 42-31 76 72 48-42 4C 30 6C  JHVSXRqB1vrHBL0l
000001C0:  43 58 64 73-77 42 36 4E-51 74 70 68-32 75 34 33  CXdswB6NQtph2u43
000001D0:  45 35 62 52-                               38 68  E5bRnkJTv6LKre8h
000      ══════════════════ Write block ══════════════════            bf2
000         Block: 00000000-000002DD                                  x1q
000        BlkLen: 000002DE/734                                       Ct3
000          File: calc.exe                                           01K
000        Offset: 11875           ('*' = append)                     4GC
000         Table: As Is                                              2SU
000                                                                   LcI
000                                                                   pNk
000                                                                   c2F
00000270:  30 68 43 59-74 6D 30 68-56 4D 51 77-54 6D 44 6D  0hCYtm0hVMQwTmDm
00000280:  31 78 64 6D-30 6C 54 6D-43 51 62 48-45 35 56 4D  1xdm0lTmCQbHE5VM
00000290:  54 6D 70 66-70 68 30 74-44 6D 53 58-54 6D 4C 4B  TmpfphOtDmSXTmLK
000002A0:  39 4C 31 7A-55 50 32 73-42 71 61 7A-43 30 4E 6B  9L1zUP2sBqazC0Nk
000002B0:  37 35 39 44-4B 4F 58 50-4D 53 7A 64-50 46 71 4D  759DKOXPMSzdPFqM
000002C0:  61 4F 43 6E-71 4A 61 49-53 6B 53 68-6B 73 47 70  aOCnqJaISkShksGp
000002D0:  45 50 75 50-35 50 73 30-57 70 37 70-41 41        EPuP5Ps0Wp7pAA
1        2        3        4        5        6        7        8        9       10
```

Clean copy of calc.exe

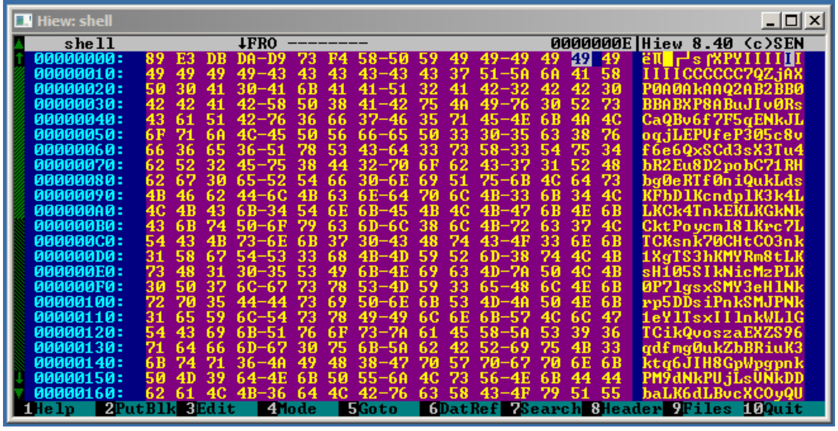


```
Hiew: calc.exe
   calc.exe            ↓FRO ---------          a32 PE   00011875 |Hiew 8.40 (c)SEN
00011875: 6A70                   push        070 ;'p'
00011877: 68E0150001             push        0010015E0 --↑1
0001187C: E847030000             call        000011BC8 --↓2
00011881: 33DB                   xor         ebx,ebx
00011883: 53                     push        ebx
00011884: 8B3D20100001           mov         edi,GetModuleHandleA --↑
0001188A: FFD7                   call        edi
0001188C: 6681384D5A             cmp         w,[eax],05A4D ;'ZM'
00011891: 751F                   jnz         0000118B2 --↓4
00011893: 8B483C                 mov         ecx,[eax][03C]
00011896: 03C8                   add         ecx,eax
00011898: 813950450000           cmp         d,[ecx],000004550 ;'  EP
0001189E: 7512                   jnz         0000118B2 --↓4
000118A0: 0FB74118               movzx       eax,w,[ecx][018]
000118A4: 3D0B010000             cmp         eax,00000010B
000118A9: 741F                   jz          0000118CA --↓5
000118AB: 3D0B020000             cmp         eax,00000020B
000118B0: 7405                   jz          0000118B7 --↓6
000118B2: 895DE4                4mov         [ebp][-01C],ebx
000118B5: EB27                   jmps        0000118DE --↓7
000118B7: 83B98400000000E       6cmp         d,[ecx][000000084],00E
000118BE: 76F2                   jbe         0000118B2 --↑4
000118C0: 33C0                   xor         eax,eax
 1Local  2FilBlk 3CryBlk 4ReLoad 5OrdLdr 6String 7Direct 8Table  91byte 10Leave
```

After copying, We are able to see that calc.exe now contains the shellcode from the other file. You can use any binary editor to the copy process.

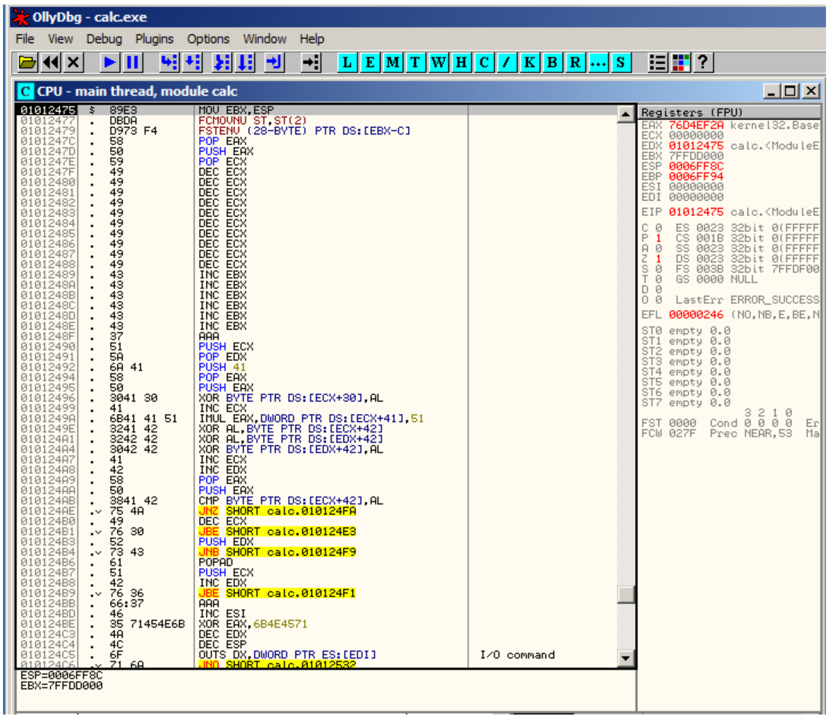


```
Hiew: calc.exe
   calc.exe          ↓FRO ---------          PE .01012475 |Hiew 8.40 (c)SEN
.01012475:  89 E3 DB DA-D9 73 F4 58-50 59 49 49-49 49 49 49  ëπ ┌┘s ƒXPYIIIIII
.01012485:  49 49 49 49-43 43 43 43-43 43 37 51-5A 6A 41 58  IIIICCCCCC7QZjAX
.01012495:  50 30 41 30-41 6B 41 41-51 32 41 42-32 42 42 30  P0A0AkAAQ2AB2BB0
.010124A5:  42 42 41 42-58 50 38 41-42 75 4A 49-76 30 52 73  BBABXP8ABuJIv0Rs
.010124B5:  43 61 51 42-76 36 66 37-46 35 71 45-4E 6B 4A 4C  CaQBv6f7F5qENkJL
.010124C5:  6F 71 6A 4C-45 50 56 66-65 50 33 30-35 63 38 76  oqjLEPVfeP305c8v
.010124D5:  66 36 65 36-51 78 53 43-64 33 73 58-33 54 75 34  f6e6QxSCd3sX3Tu4
.010124E5:  62 52 32 45-75 38 44 32-70 6F 62 43-37 31 52 48  bR2Eu8D2pobC71RH
.010124F5:  62 67 30 65-52 54 66 30-6E 69 51 75-6B 4C 64 73  bg0eRTf0niQukLds
.01012505:  4B 46 62 44-6C 4B 63 6E-64 70 6C 4B-33 6B 34 4C  KFbDlKcndplK3k4L
.01012515:  4C 4B 43 6B-34 54 6E 6B-45 4B 4C 4B-47 6B 4E 6B  LKCk4TnkEKLKGkNk
.01012525:  43 6B 74 50-6F 79 63 6D-6C 38 6C 4B-72 63 37 4C  CktPoycml8lKrc7L
.01012535:  54 43 4B 73-6E 6B 37 30-43 48 74 43-4F 33 6E 6B  TCKsnk70CHtCO3nk
.01012545:  31 58 67 54-53 33 68 4B-4D 59 52 6D-38 74 4C 4B  1XgTS3hKMYRm8tLK
.01012555:  73 48 31 30-35 53 49 6B-4E 69 63 4D-7A 50 4C 4B  sH105SIkNicMzPLK
.01012565:  30 50 37 6C-67 73 78 53-4D 59 33 65-48 6C 4E 6B  0P7lgsxSMY3eHlNk
.01012575:  72 70 35 44-44 73 69 50-6E 6B 53 4D-4A 50 4E 6B  rp5DDsiPnkSMJPNk
.01012585:  31 65 59 6C-54 73 78 49-49 6C 6E 6B-57 4C 6C 47  1eYlTsxIIlnkWLlG
.01012595:  54 43 69 6B-51 76 6F 73-7A 61 45 58-5A 53 39 36  TCikQvoszaEXZS96
.010125A5:  71 64 66 6D-67 30 75 6B-5A 62 42 52-69 75 4B 33  qdfmg0ukZbBRiuK3
.010125B5:  6B 74 71 36-4A 49 48 38-47 70 57 70-67 70 6E 6B  ktq6JIH8GpWpgpnk
.010125C5:  50 4D 39 64-4E 6B 50 55-6A 4C 73 56-4E 6B 44 44  PM9dNkPUjLsVNkDD
.010125D5:  62 61 4C 4B-36 64 4C 42-76 63 58 43-4F 79 51 55  baLK6dLBvcXCOyQU
 1Help  2PutBlk 3Edit   4Mode   5Goto   6DatRef 7Search 8Header 9Files 10Quit
```



```
Hiew: shell
   shell             ↓FRO ---------            0000000E |Hiew 8.40 (c)SEN
00000000:  89 E3 DB DA-D9 73 F4 58-50 59 49 49-49 49 49 49  ëπ ┌┘s ƒXPYIIIIII
00000010:  49 49 49 49-43 43 43 43-43 43 37 51-5A 6A 41 58  IIIICCCCCC7QZjAX
00000020:  50 30 41 30-41 6B 41 41-51 32 41 42-32 42 42 30  P0A0AkAAQ2AB2BB0
00000030:  42 42 41 42-58 50 38 41-42 75 4A 49-76 30 52 73  BBABXP8ABuJIv0Rs
00000040:  43 61 51 42-76 36 66 37-46 35 71 45-4E 6B 4A 4C  CaQBv6f7F5qENkJL
00000050:  6F 71 6A 4C-45 50 56 66-65 50 33 30-35 63 38 76  oqjLEPVfeP305c8v
00000060:  66 36 65 36-51 78 53 43-64 33 73 58-33 54 75 34  f6e6QxSCd3sX3Tu4
00000070:  62 52 32 45-75 38 44 32-70 6F 62 43-37 31 52 48  bR2Eu8D2pobC71RH
00000080:  62 67 30 65-52 54 66 30-6E 69 51 75-6B 4C 64 73  bg0eRTf0niQukLds
00000090:  4B 46 62 44-6C 4B 63 6E-64 70 6C 4B-33 6B 34 4C  KFbDlKcndplK3k4L
000000A0:  4C 4B 43 6B-34 54 6E 6B-45 4B 4C 4B-47 6B 4E 6B  LKCk4TnkEKLKGkNk
000000B0:  43 6B 74 50-6F 79 63 6D-6C 38 6C 4B-72 63 37 4C  CktPoycml8lKrc7L
000000C0:  54 43 4B 73-6E 6B 37 30-43 48 74 43-4F 33 6E 6B  TCKsnk70CHtCO3nk
000000D0:  31 58 67 54-53 33 68 4B-4D 59 52 6D-38 74 4C 4B  1XgTS3hKMYRm8tLK
000000E0:  73 48 31 30-35 53 49 6B-4E 69 63 4D-7A 50 4C 4B  sH105SIkNicMzPLK
000000F0:  30 50 37 6C-67 73 78 53-4D 59 33 65-48 6C 4E 6B  0P7lgsxSMY3eHlNk
00000100:  72 70 35 44-44 73 69 50-6E 6B 53 4D-4A 50 4E 6B  rp5DDsiPnkSMJPNk
00000110:  31 65 59 6C-54 73 78 49-49 6C 6E 6B-57 4C 6C 47  1eYlTsxIIlnkWLlG
00000120:  54 43 69 6B-51 76 6F 73-7A 61 45 58-5A 53 39 36  TCikQvoszaEXZS96
00000130:  71 64 66 6D-67 30 75 6B-5A 62 42 52-69 75 4B 33  qdfmg0ukZbBRiuK3
00000140:  6B 74 71 36-4A 49 48 38-47 70 57 70-67 70 6E 6B  ktq6JIH8GpWpgpnk
00000150:  50 4D 39 64-4E 6B 50 55-6A 4C 73 56-4E 6B 44 44  PM9dNkPUjLsVNkDD
00000160:  62 61 4C 4B-36 64 4C 42-76 63 58 43-4F 79 51 55  baLK6dLBvcXCOyQU
 1Help  2PutBlk 3Edit   4Mode   5Goto   6DatRef 7Search 8Header 9Files 10Quit
```

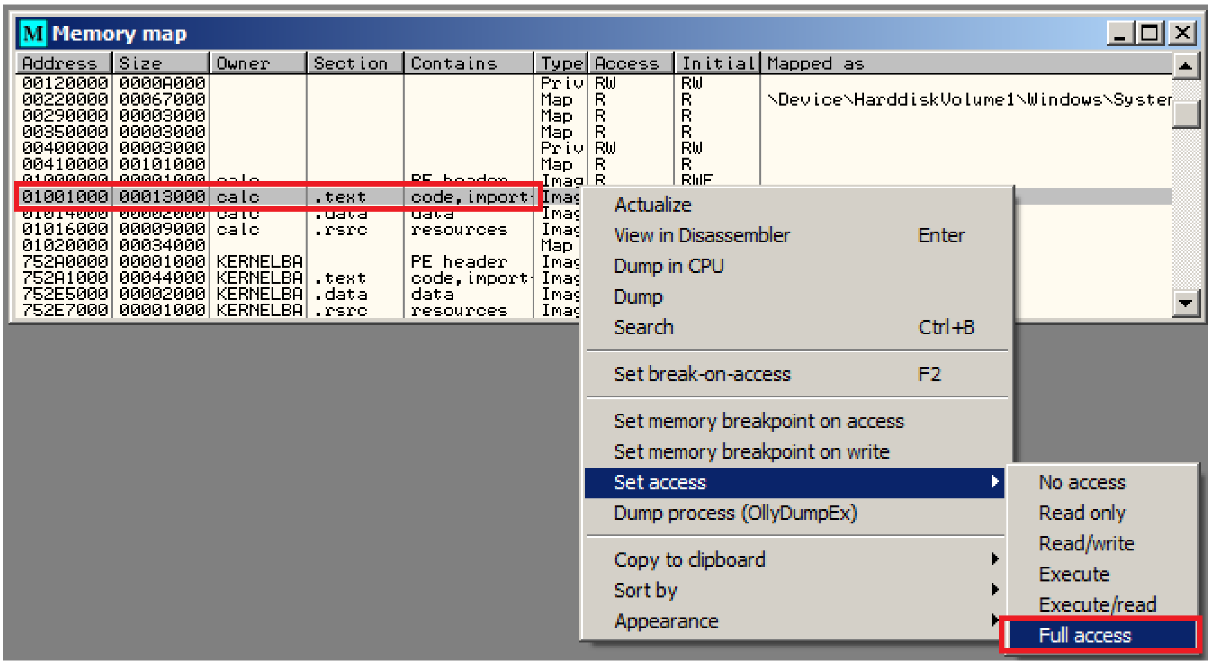
Opening calc.exe to debugger will show the shellcode on entry point.


OllyDbg - calc.exe
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
CPU - main thread, module calc

```
01012475  $  89E3          MOV EBX,ESP
01012477  .  DBDA          FCMOVNU ST,ST(2)
01012479  .  D973 F4       FSTENV (28-BYTE) PTR DS:[EBX-C]
0101247C  .  58            POP EAX
0101247D  .  50            PUSH EAX
0101247E  .  59            POP ECX
0101247F  .  49            DEC ECX
01012480  .  49            DEC ECX
01012481  .  49            DEC ECX
01012482  .  49            DEC ECX
01012483  .  49            DEC ECX
01012484  .  49            DEC ECX
01012485  .  49            DEC ECX
01012486  .  49            DEC ECX
01012487  .  49            DEC ECX
01012488  .  49            DEC ECX
01012489  .  43            INC EBX
0101248A  .  43            INC EBX
0101248B  .  43            INC EBX
0101248C  .  43            INC EBX
0101248D  .  43            INC EBX
0101248E  .  43            INC EBX
0101248F  .  37            AAA
01012490  .  51            PUSH ECX
01012491  .  5A            POP EDX
01012492  .  6A 41         PUSH 41
01012494  .  58            POP EAX
01012495  .  50            PUSH EAX
01012496  .  3041 30       XOR BYTE PTR DS:[ECX+30],AL
01012499  .  41            INC ECX
0101249A  .  6B41 41 51    IMUL EAX,DWORD PTR DS:[ECX+41],51
0101249E  .  3241 42       XOR AL,BYTE PTR DS:[ECX+42]
010124A1  .  3242 42       XOR AL,BYTE PTR DS:[EDX+42]
010124A4  .  3042 42       XOR BYTE PTR DS:[EDX+42],AL
010124A7  .  41            INC ECX
010124A8  .  42            INC EDX
010124A9  .  58            POP EAX
010124AA  .  50            PUSH EAX
010124AB  .  3841 42       CMP BYTE PTR DS:[ECX+42],AL
010124AE  .v 75 4A         JNZ SHORT calc.010124FA
010124B0  .  49            DEC ECX
010124B1  .v 76 30         JBE SHORT calc.010124E3
010124B3  .  52            PUSH EDX
010124B4  .v 73 43         JNB SHORT calc.010124F9
010124B6  .  61            POPAD
010124B7  .  51            PUSH ECX
010124B8  .  42            INC EDX
010124B9  .v 76 36         JBE SHORT calc.010124F1
010124BB  .  66:37         AAA
010124BD  .  46            INC ESI
010124BE  .  35 71454E6B   XOR EAX,6B4E4571
010124C3  .  4A            DEC EDX
010124C4  .  4C            DEC ESP
010124C5  .  6F            OUTS DX,DWORD PTR ES:[EDI]          I/O command
010124C6  .v 71 6A         JNO SHORT calc.01012532
ESP=0006FF8C
EBX=7FFDD000
```

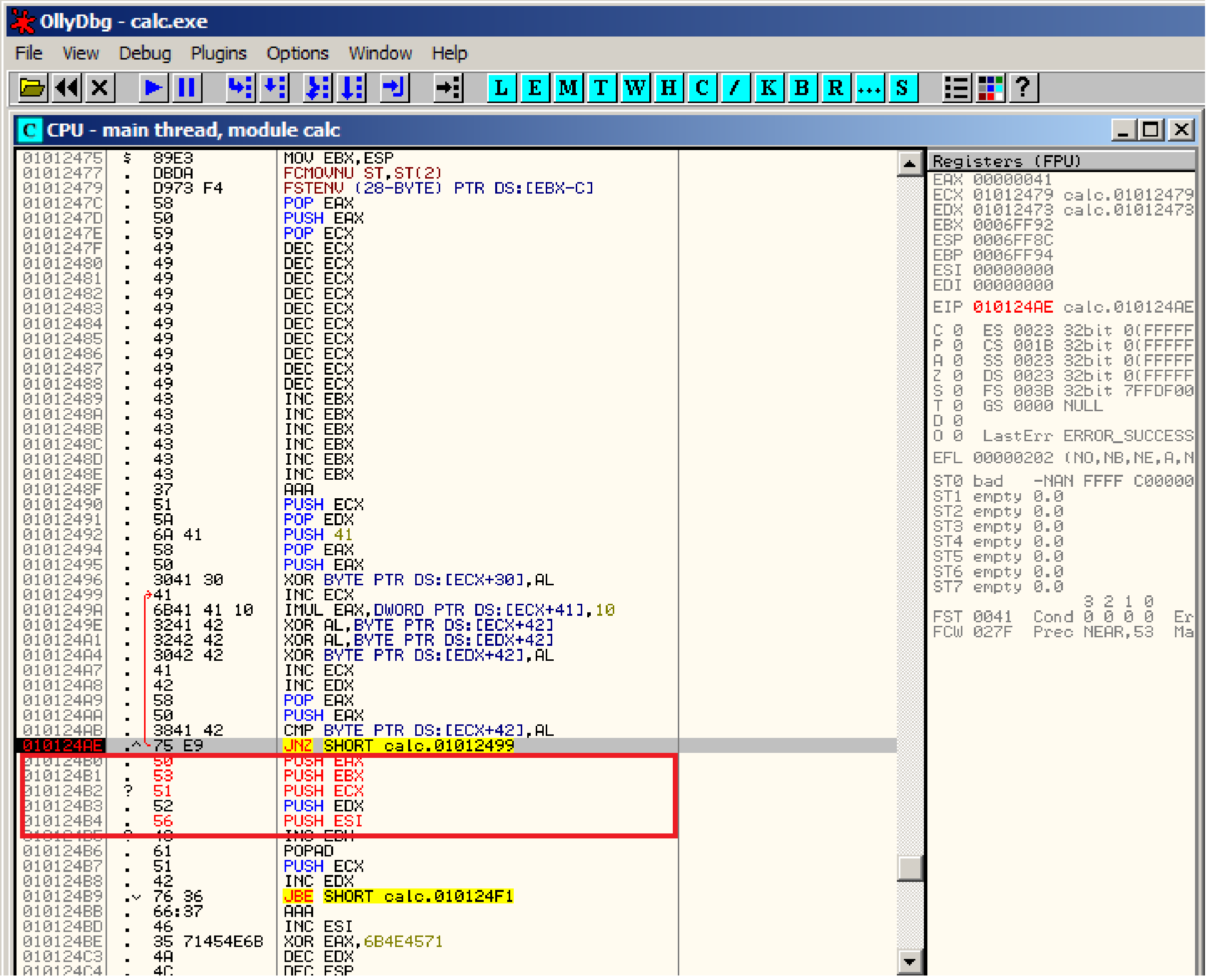
```
Registers (FPU)
EAX 76D4EF2A kernel32.Base
ECX 00000000
EDX 01012475 calc.<ModuleE
EBX 7FFDD000
ESP 0006FF8C
EBP 0006FF94
ESI 00000000
EDI 00000000

EIP 01012475 calc.<ModuleE

C 0  ES 0023 32bit 0(FFFFF
P 1  CS 001B 32bit 0(FFFFF
A 0  SS 0023 32bit 0(FFFFF
Z 1  DS 0023 32bit 0(FFFFF
S 0  FS 003B 32bit 7FFDF00
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS

EFL 00000246 (NO,NB,E,BE,N

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0
FST 0000  Cond 0 0 0 0   Er
FCW 027F  Prec NEAR,53   Ma
```

We need to set the section access flag as calc.exe ".text" section do not have write access. We need to set it will full access.



```
Address  Size     Owner    Section Contains     Type Access Initial Mapped as
00120000 0000A000                               Priv RW     RW
00220000 00067000                               Map  R      R       \Device\HarddiskVolume1\Windows\Syster
00290000 00003000                               Map  R      R
00350000 00003000                               Map  R      R
00400000 00003000                               Priv RW     RW
00410000 00101000                               Map  R      R
01000000 00001000 calc             PE header    Imag R      RWE
01001000 00013000 calc     .text   code,import  Imag
01014000 00002000 calc     .data   data         Imag
01016000 00009000 calc     .rsrc   resources    Imag
01020000 00034000                               Map
752A0000 00001000 KERNELBA         PE header    Imag
752A1000 00044000 KERNELBA .text   code,import  Imag
752E5000 00002000 KERNELBA .data   data         Imag
752E7000 00001000 KERNELBA .rsrc   resources    Imag
```

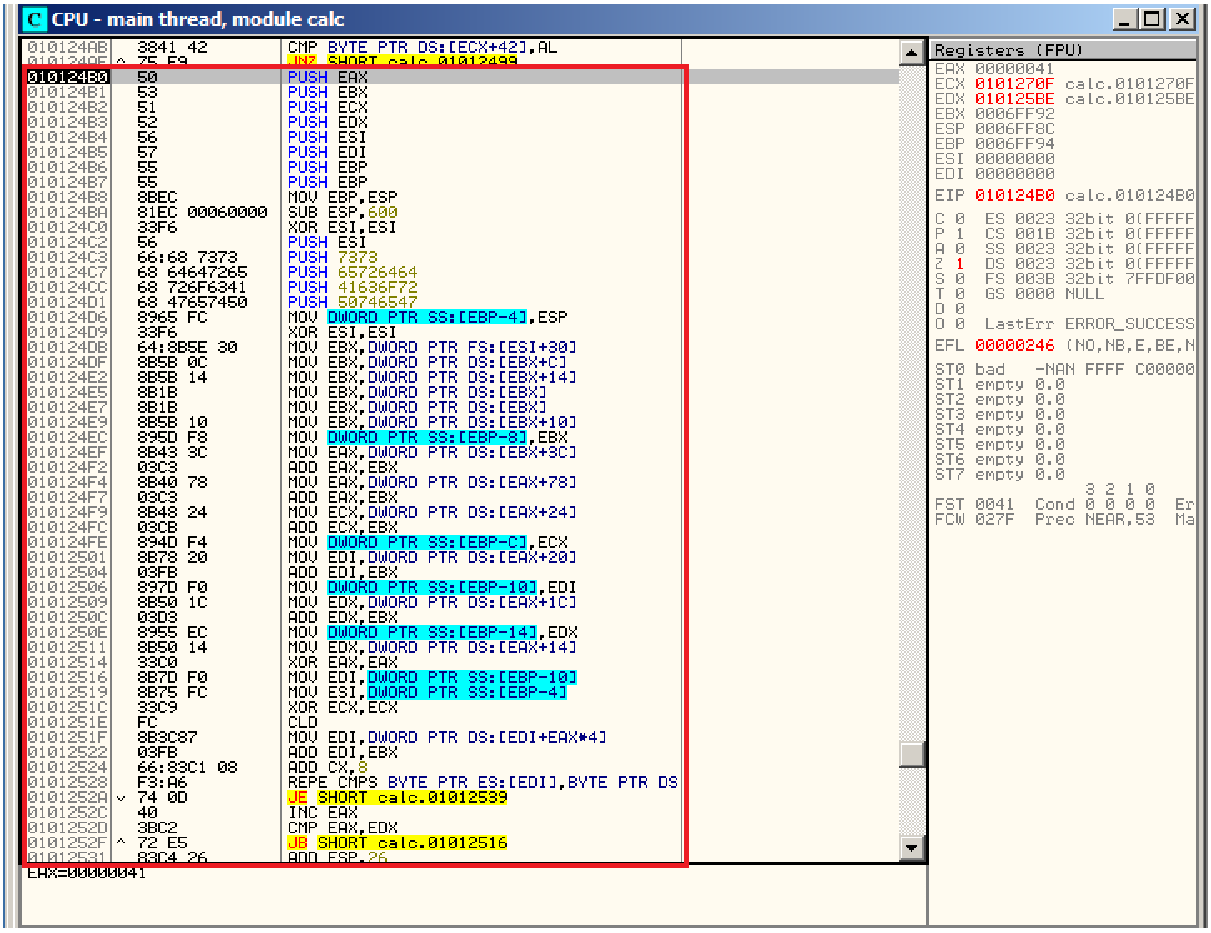
We are able to see a decryption loop that will decrypt the next layer of the code.



```
01012475  $  89E3          MOV EBX,ESP
01012477  .  DBDA          FCMOVNU ST,ST(2)
01012479  .  D973 F4       FSTENV (28-BYTE) PTR DS:[EBX-C]
0101247C  .  58            POP EAX
0101247D  .  50            PUSH EAX
0101247E  .  59            POP ECX
0101247F  .  49            DEC ECX
01012480  .  49            DEC ECX
01012481  .  49            DEC ECX
01012482  .  49            DEC ECX
01012483  .  49            DEC ECX
01012484  .  49            DEC ECX
01012485  .  49            DEC ECX
01012486  .  49            DEC ECX
01012487  .  49            DEC ECX
01012488  .  49            DEC ECX
01012489  .  43            INC EBX
0101248A  .  43            INC EBX
0101248B  .  43            INC EBX
0101248C  .  43            INC EBX
0101248D  .  43            INC EBX
0101248E  .  43            INC EBX
0101248F  .  37            AAA
01012490  .  51            PUSH ECX
01012491  .  5A            POP EDX
01012492  .  6A 41         PUSH 41
01012494  .  58            POP EAX
01012495  .  50            PUSH EAX
01012496  .  3041 30       XOR BYTE PTR DS:[ECX+30],AL
01012499  .  41            INC ECX
0101249A  .  6B41 41 10    IMUL EAX,DWORD PTR DS:[ECX+41],10
0101249E  .  3241 42       XOR AL,BYTE PTR DS:[ECX+42]
010124A1  .  3242 42       XOR AL,BYTE PTR DS:[EDX+42]
010124A4  .  3042 42       XOR BYTE PTR DS:[EDX+42],AL
010124A7  .  41            INC ECX
010124A8  .  42            INC EDX
010124A9  .  58            POP EAX
010124AA  .  50            PUSH EAX
010124AB  .  3841 42       CMP BYTE PTR DS:[ECX+42],AL
010124AE  .^ 75 E9         JNZ SHORT calc.01012499
010124B0  .  50            PUSH EAX
010124B1  .  53            PUSH EBX
010124B2  ?  51            PUSH ECX
010124B3  .  52            PUSH EDX
010124B4  .  56            PUSH ESI
010124B5  ?  40            INC EAX
010124B6  .  61            POPAD
010124B7  .  51            PUSH ECX
010124B8  .  42            INC EDX
010124B9  .v 76 36         JBE SHORT calc.010124F1
010124BB  .  66:37         AAA
010124BD  .  46            INC ESI
010124BE  .  35 71454E6B   XOR EAX,6B4E4571
010124C3  .  4A            DEC EDX
010124C4  .  4C            DEC ESP
```

```
Registers (FPU)
EAX 00000041
ECX 01012479 calc.01012479
EDX 01012473 calc.01012473
EBX 0006FF92
ESP 0006FF8C
EBP 0006FF94
ESI 00000000
EDI 00000000

EIP 010124AE calc.010124AE

C 0  ES 0023 32bit 0(FFFFF
P 0  CS 001B 32bit 0(FFFFF
A 0  SS 0023 32bit 0(FFFFF
Z 0  DS 0023 32bit 0(FFFFF
S 0  FS 003B 32bit 7FFDF00
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS

EFL 00000202 (NO,NB,NE,A,N

ST0 bad   -NAN FFFF C00000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
                3 2 1 0
FST 0041  Cond 0 0 0 0  Er
FCW 027F  Prec NEAR,53  Ma
```

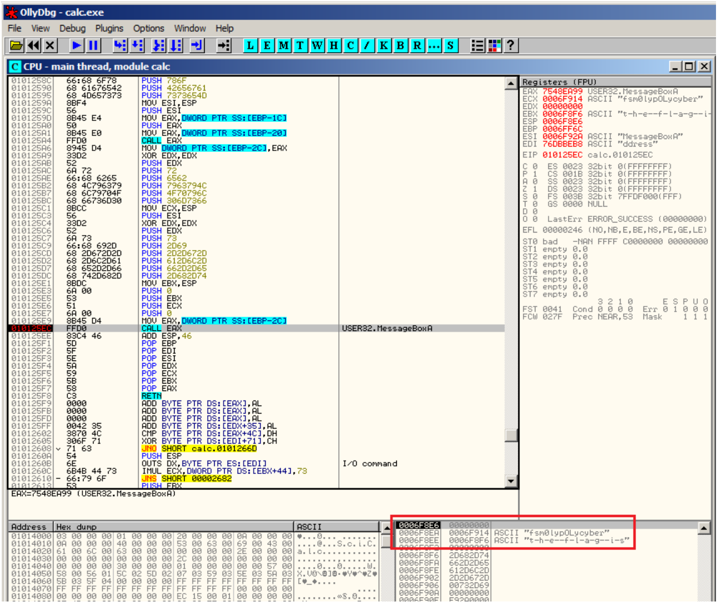
After decryption, the code will look like the image below.



```
010124AB    3841 42          CMP BYTE PTR DS:[ECX+42],AL
010124AE  ^ 75 E9            JNZ SHORT calc.01012499
010124B0    50               PUSH EAX
010124B1    53               PUSH EBX
010124B2    51               PUSH ECX
010124B3    52               PUSH EDX
010124B4    56               PUSH ESI
010124B5    57               PUSH EDI
010124B6    55               PUSH EBP
010124B7    55               PUSH EBP
010124B8    8BEC             MOV EBP,ESP
010124BA    81EC 00060000    SUB ESP,600
010124C0    33F6             XOR ESI,ESI
010124C2    56               PUSH ESI
010124C3    66:68 7373       PUSH 7373
010124C7    68 64647265      PUSH 65726464
010124CC    68 726F6341      PUSH 41636F72
010124D1    68 47657450      PUSH 50746547
010124D6    8965 FC          MOV DWORD PTR SS:[EBP-4],ESP
010124D9    33F6             XOR ESI,ESI
010124DB    64:8B5E 30       MOV EBX,DWORD PTR FS:[ESI+30]
010124DF    8B5B 0C          MOV EBX,DWORD PTR DS:[EBX+C]
010124E2    8B5B 14          MOV EBX,DWORD PTR DS:[EBX+14]
010124E5    8B1B             MOV EBX,DWORD PTR DS:[EBX]
010124E7    8B1B             MOV EBX,DWORD PTR DS:[EBX]
010124E9    8B5B 10          MOV EBX,DWORD PTR DS:[EBX+10]
010124EC    895D F8          MOV DWORD PTR SS:[EBP-8],EBX
010124EF    8B43 3C          MOV EAX,DWORD PTR DS:[EBX+3C]
010124F2    03C3             ADD EAX,EBX
010124F4    8B40 78          MOV EAX,DWORD PTR DS:[EAX+78]
010124F7    03C3             ADD EAX,EBX
010124F9    8B48 24          MOV ECX,DWORD PTR DS:[EAX+24]
010124FC    03CB             ADD ECX,EBX
010124FE    894D F4          MOV DWORD PTR SS:[EBP-C],ECX
01012501    8B78 20          MOV EDI,DWORD PTR DS:[EAX+20]
01012504    03FB             ADD EDI,EBX
01012506    897D F0          MOV DWORD PTR SS:[EBP-10],EDI
01012509    8B50 1C          MOV EDX,DWORD PTR DS:[EAX+1C]
0101250C    03D3             ADD EDX,EBX
0101250E    8955 EC          MOV DWORD PTR SS:[EBP-14],EDX
01012511    8B50 14          MOV EDX,DWORD PTR DS:[EAX+14]
01012514    33C0             XOR EAX,EAX
01012516    8B7D F0          MOV EDI,DWORD PTR SS:[EBP-10]
01012519    8B75 FC          MOV ESI,DWORD PTR SS:[EBP-4]
0101251C    33C9             XOR ECX,ECX
0101251E    FC               CLD
0101251F    8B3C87           MOV EDI,DWORD PTR DS:[EDI+EAX*4]
01012522    03FB             ADD EDI,EBX
01012524    66:83C1 08       ADD CX,8
01012528    F3:A6            REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS
0101252A  v 74 0D            JE SHORT calc.01012539
0101252C    40               INC EAX
0101252D    3BC2             CMP EAX,EDX
0101252F  ^ 72 E5            JB SHORT calc.01012516
01012531    83C4 26          ADD ESP,26
EAX=00000041
```

```
Registers (FPU)
EAX 00000041
ECX 0101270F calc.0101270F
EDX 010125BE calc.010125BE
EBX 0006FF92
ESP 0006FF8C
EBP 0006FF94
ESI 00000000
EDI 00000000

EIP 010124B0 calc.010124B0

C 0  ES 0023 32bit 0(FFFFF
P 1  CS 001B 32bit 0(FFFFF
A 0  SS 0023 32bit 0(FFFFF
Z 1  DS 0023 32bit 0(FFFFF
S 0  FS 003B 32bit 7FFDF00
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS

EFL 00000246 (NO,NB,E,BE,N

ST0 bad   -NAN FFFF C00000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0
FST 0041  Cond 0 0 0 0   Er
FCW 027F  Prec NEAR,53   Ma
```

Debugging firther, we are able to see a call to MessageBoxA and its parameter is pushed in stack. The flag is located in the stack as a parameter of MessageBoxA API call.



```
0101258C    66:68 6F78       PUSH 786F
01012590    68 61676542      PUSH 42656761
01012595    68 4D657373      PUSH 7373654D
0101259A    8BF4             MOV ESI,ESP
0101259C    56               PUSH ESI
0101259D    8B45 E4          MOV EAX,DWORD PTR SS:[EBP-1C]
010125A0    50               PUSH EAX
010125A1    8B45 E0          MOV EAX,DWORD PTR SS:[EBP-20]
010125A4    FFD0             CALL EAX
010125A6    8945 D4          MOV DWORD PTR SS:[EBP-2C],EAX
010125A9    33D2             XOR EDX,EDX
010125AB    52               PUSH EDX
010125AC    6A 72            PUSH 72
010125AE    66:68 6265       PUSH 6562
010125B2    68 4C796379      PUSH 7963794C
010125B7    68 6C79704F      PUSH 4F70796C
010125BC    68 66736D30      PUSH 306D7366
010125C1    8BCC             MOV ECX,ESP
010125C3    56               PUSH ESI
010125C4    33D2             XOR EDX,EDX
010125C6    52               PUSH EDX
010125C7    6A 73            PUSH 73
010125C9    66:68 692D       PUSH 2D69
010125CD    68 2D672D2D      PUSH 2D2D672D
010125D2    68 2D6C2D61      PUSH 612D6C2D
010125D7    68 652D2D66      PUSH 662D2D65
010125DC    68 742D682D      PUSH 2D682D74
010125E1    8BDC             MOV EBX,ESP
010125E3    6A 00            PUSH 0
010125E5    53               PUSH EBX
010125E6    51               PUSH ECX
010125E7    6A 00            PUSH 0
010125E9    8B45 D4          MOV EAX,DWORD PTR SS:[EBP-2C]
010125EC    FFD0             CALL EAX                              USER32.MessageBoxA
010125EE    83C4 46          ADD ESP,46
010125F1    5D               POP EBP
010125F2    5F               POP EDI
010125F3    5E               POP ESI
010125F4    5A               POP EDX
010125F5    59               POP ECX
010125F6    5B               POP EBX
010125F7    58               POP EAX
010125F8    C3               RETN
010125F9    0000             ADD BYTE PTR DS:[EAX],AL
010125FB    0000             ADD BYTE PTR DS:[EAX],AL
010125FD    0000             ADD BYTE PTR DS:[EAX],AL
010125FF    0042 35          ADD BYTE PTR DS:[EDX+35],AL
01012602    3870 4C          CMP BYTE PTR DS:[EAX+4C],DH
01012605    306F 71          XOR BYTE PTR DS:[EDI+71],CH
01012608  v 71 63            JNO SHORT calc.0101266D
0101260A    54               PUSH ESP
0101260B    6E               OUTS DX,BYTE PTR ES:[EDI]             I/O command
0101260C    6B4B 44 73       IMUL ECX,DWORD PTR DS:[EBX+44],73
01012610  - 66:79 6F         JNS SHORT 00002682
01012613    53               PUSH EBX
EAX=7548EA99 (USER32.MessageBoxA)
```

```
Registers (FPU)
EAX 7548EA99 USER32.MessageBoxA
ECX 0006F914 ASCII "fsm0lypOLycyber"
EDX 00000000
EBX 0006F8F6 ASCII "t-h-e--f-l-a-g--i-
ESP 0006F8E6
EBP 0006FF6C
ESI 0006F92A ASCII "MessageBoxA"
EDI 76DBBEB8 ASCII "ddress"

EIP 010125EC calc.010125EC

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 bad   -NAN FFFF C0000000 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O
FST 0041  Cond 0 0 0 0  Err 0 1 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1
```

```
Address   Hex dump                                          ASCII
01014000  03 00 00 00 01 00 00 00 20 00 00 00 0A 00 00 00   ♥...☺... .......
01014010  0A 00 00 00 40 00 00 00 53 00 63 00 69 00 43 00   ....@...S.c.i.C.
01014020  61 00 6C 00 63 00 00 00 00 00 00 00 2E 00 00 00   a.l.c...........
01014030  00 00 00 00 00 00 00 00 2C 00 00 00 00 00 00 00   ........,.......
01014040  00 00 00 00 30 00 00 00 01 00 00 00 00 00 57 00   ....0...☺.....W.
01014050  58 00 56 01 5C 02 5D 02 07 03 59 03 5E 03 5A 03   X.V☺\☻]☻•♥Y♥^♥Z♥
01014060  5B 03 5F 04 00 00 00 00 FF FF FF FF FF FF FF FF   [♥_♦....
01014070  FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00       ....
01014080  00 00 00 00 00 00 00 00 EC 15 00 01 00 00 00 00   ........∞§.☺....
```

```
0006F8E6   00000000
0006F8EA   0006F914  ASCII "fsm0lypOLycyber"
0006F8EE   0006F8F6  ASCII "t-h-e--f-l-a-g--i-s"
0006F8F2   00000000
0006F8F6   2D682D74
0006F8FA   662D2D65
0006F8FE   612D6C2D
0006F902   2D2D672D
0006F906   00732D69
0006F90A   00000000
0006F90E   E9200000
```

Flag:

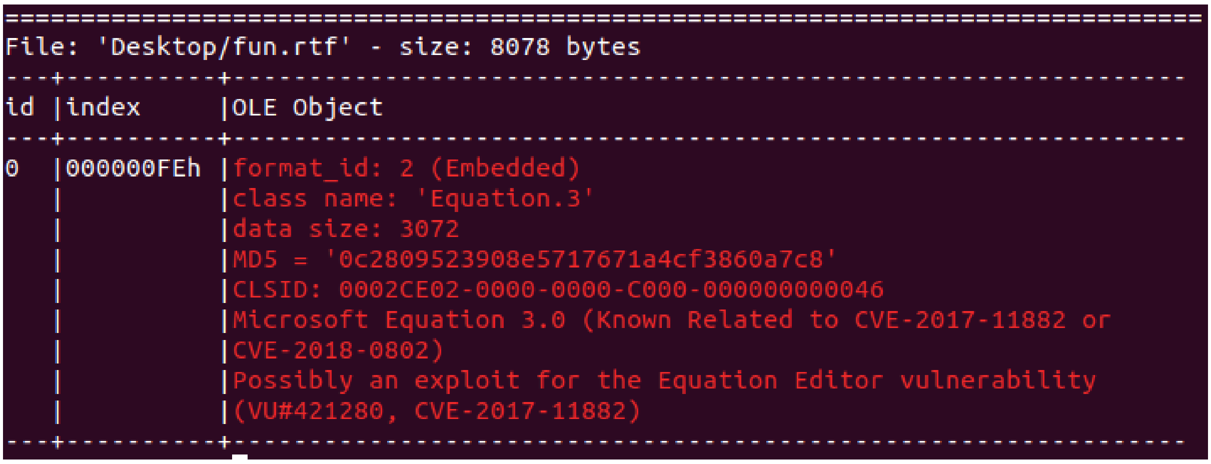
fsm0lypOLycyber

5) _**CVE-2017**_:

Challenge: <DOWNLOAD>

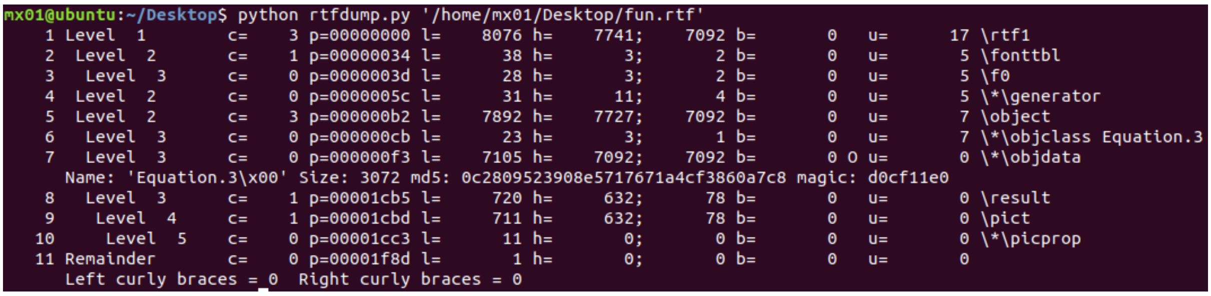
This sample exploits CVE-2017-11882 Equation vulnerability which leads to remote code execution. In this challenge, the exploit is built to automatically trigger a command using CMD upon opening the RTF document.

Using rtfobj we are able to identify the vulnerability used in this RTF file.



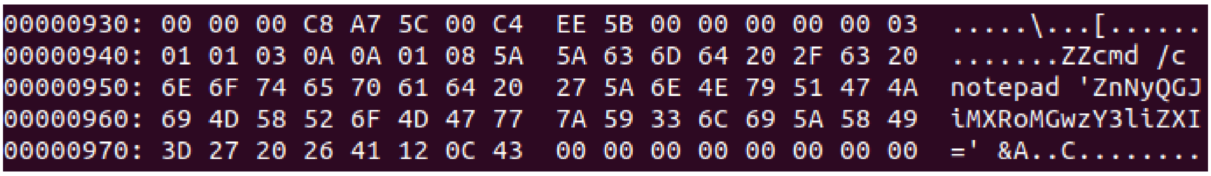
```
===============================================================================
File: 'Desktop/fun.rtf' - size: 8078 bytes
---+----------+---------------------------------------------------------------
id |index     |OLE Object
---+----------+---------------------------------------------------------------
0  |000000FEh |format_id: 2 (Embedded)
   |          |class name: 'Equation.3'
   |          |data size: 3072
   |          |MD5 = '0c2809523908e5717671a4cf3860a7c8'
   |          |CLSID: 0002CE02-0000-0000-C000-000000000046
   |          |Microsoft Equation 3.0 (Known Related to CVE-2017-11882 or
   |          |CVE-2018-0802)
   |          |Possibly an exploit for the Equation Editor vulnerability
   |          |(VU#421280, CVE-2017-11882)
---+----------+---------------------------------------------------------------
```

Using rtfdump.py we are able to identify object related to equation:



```
mx01@ubuntu:~/Desktop$ python rtfdump.py '/home/mx01/Desktop/fun.rtf'
    1 Level  1        c=    3 p=00000000 l=    8076 h=    7741;    7092 b=       0   u=      17 \rtf1
    2  Level  2       c=    1 p=00000034 l=      38 h=       3;       2 b=       0   u=       5 \fonttbl
    3   Level  3      c=    0 p=0000003d l=      28 h=       3;       2 b=       0   u=       5 \f0
    4  Level  2       c=    0 p=0000005c l=      31 h=      11;       4 b=       0   u=       5 \*\generator
    5  Level  2       c=    3 p=000000b2 l=    7892 h=    7727;    7092 b=       0   u=       7 \object
    6   Level  3      c=    0 p=000000cb l=      23 h=       3;       1 b=       0   u=       7 \*\objclass Equation.3
    7   Level  3      c=    0 p=000000f3 l=    7105 h=    7092;    7092 b=       0 O u=       0 \*\objdata
      Name: 'Equation.3\x00' Size: 3072 md5: 0c2809523908e5717671a4cf3860a7c8 magic: d0cf11e0
    8   Level  3      c=    1 p=00001cb5 l=     720 h=     632;      78 b=       0   u=       0 \result
    9    Level  4     c=    1 p=00001cbd l=     711 h=     632;      78 b=       0   u=       0 \pict
   10     Level  5    c=    0 p=00001cc3 l=      11 h=       0;       0 b=       0   u=       0 \*\picprop
   11 Remainder       c=    0 p=00001f8d l=       1 h=       0;       0 b=       0   u=       0
      Left curly braces = 0  Right curly braces = 0
```

We will then dump the 7th section where the Equation object is found and print it in hex.



```
00000930: 00 00 00 C8 A7 5C 00 C4  EE 5B 00 00 00 00 00 03  .....\...[......
00000940: 01 01 03 0A 0A 01 08 5A  5A 63 6D 64 20 2F 63 20  .......ZZcmd /c 
00000950: 6E 6F 74 65 70 61 64 20  27 5A 6E 4E 79 51 47 4A  notepad 'ZnNyQGJ
00000960: 69 4D 58 52 6F 4D 47 77  7A 59 33 6C 69 5A 58 49  iMXRoMGwzY3liZXI
00000970: 3D 27 20 26 41 12 0C 43  00 00 00 00 00 00 00 00  =' &A..C........
```

We are able to see a command (CMD) that opens a notepad with additional base64 encoded string (ZnNyQGJiMXRoMGwzY3liZXI=).

Flag:

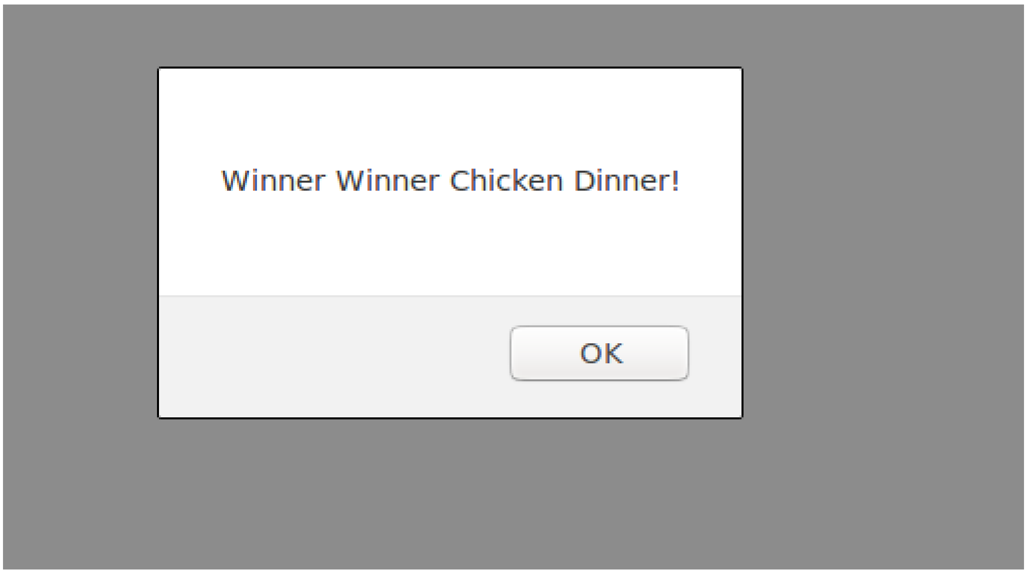
fsr@bb1th0l3cyber

[Reverse Engineering]

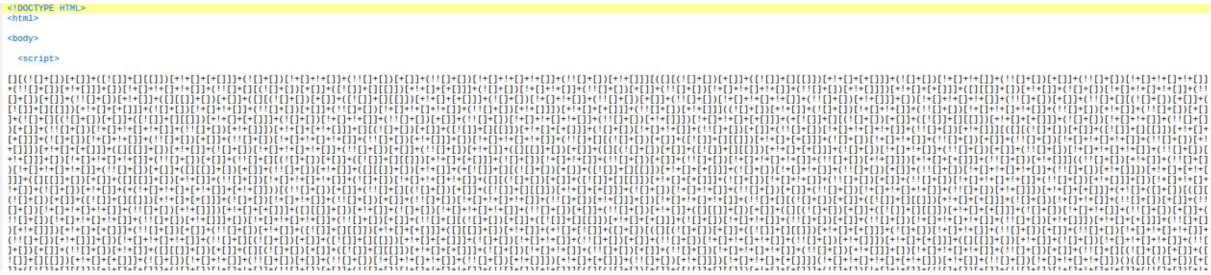
1) PUBG_Scr1pt:

Challenge: <DOWNLOAD>

When page is loaded in Web Browser, it shows following pop-up


Winner Winner Chicken Dinner!

To look at the code let's either "view page-source" or open the file in any text editor



```
<!DOCTYPE HTML>
<html>

<body>

  <script>
```

Clearly the JS code is obfuscated with characters. Online google search "JS obfuscated with characters" will show you multiple obfuscators such as Jsfuck, Jsnice, Obfuscator.IO, JJEncode etc.

But only Jsfuck fits the characters used in the script i.e. six characters: [,], (,), !, and +

Search any online deobfuscator for Jsfuck like this https://enkhee-osiris.github.io/Decoder-JSFuck/, Feed the script to de-obfuscate and you will get the result (Note that some of the de-obfuscator present online will ignore the comment and you will not be able to see the flag. It's always good to try 2-3 online decoders when you are deobfuscating codes)

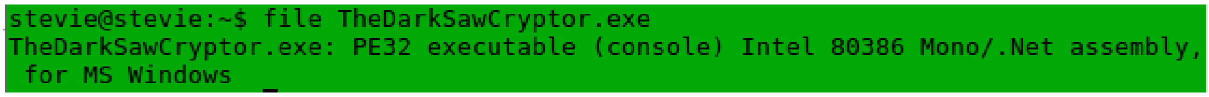
Result : alert('Winner Winner Chicken Dinner!'); // fsPubG4ddict3dcyber

flag : fsPubG4ddict3dcyber

2) TheDarkSawCryptor:

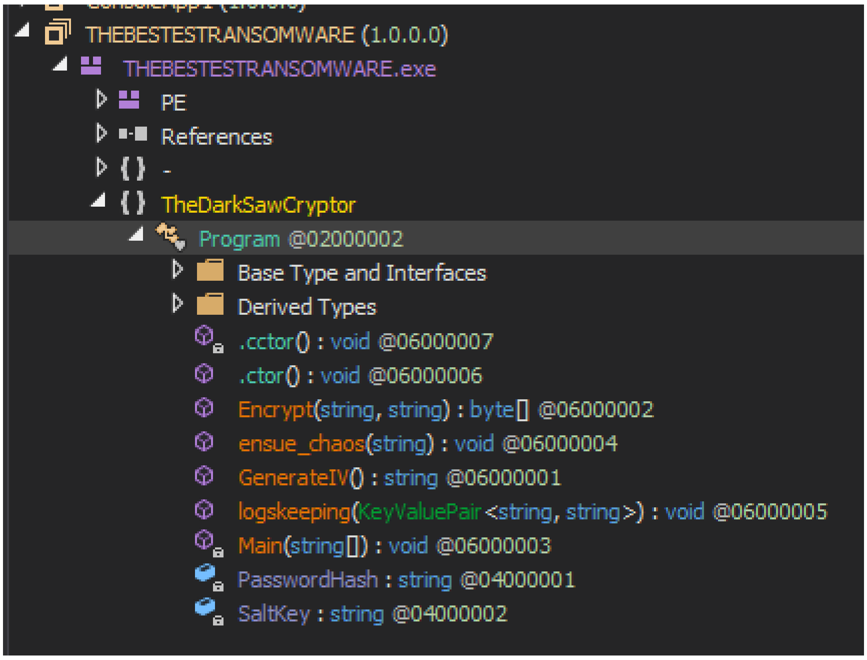
Challenge: <DOWNLOAD>

In this challenge, you are provided with three files, namely desktop.ini (hidden), TheDarkSawCryptor.exe, and secret.joker.saw. Quickly looking at desktop.ini and secret.joker.saw, they appear to be some sort of data file and most likely encrypted/encoded. Checking TheDarkSawCryptor through the file command reveals it to be a .NET binary (as shown below), which means that it could probably be decompiled by a .NET decompiler such as dnSpy.



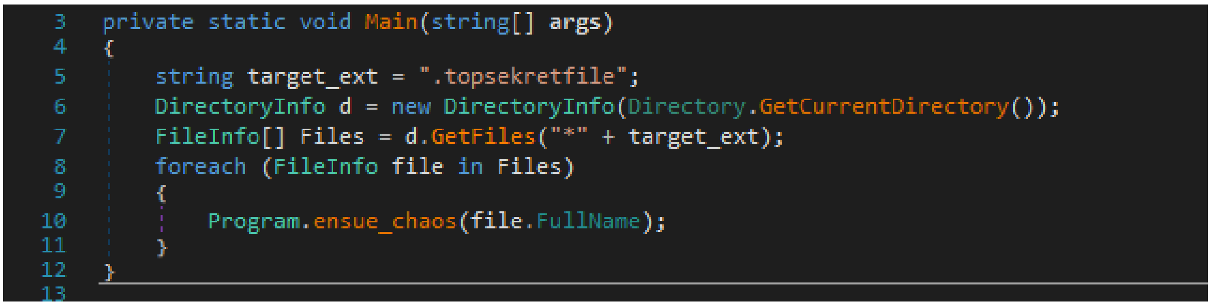
```
stevie@stevie:~$ file TheDarkSawCryptor.exe
TheDarkSawCryptor.exe: PE32 executable (console) Intel 80386 Mono/.Net assembly,
 for MS Windows
```

After decompiling the binary, we can see 5 methods in this binary, as shown below.



```
THEBESTESTRANSOMWARE (1.0.0.0)
  THEBESTESTRANSOMWARE.exe
    PE
    References
    { } -
    { } TheDarkSawCryptor
      Program @02000002
        Base Type and Interfaces
        Derived Types
        .cctor() : void @06000007
        .ctor() : void @06000006
        Encrypt(string, string) : byte[] @06000002
        ensue_chaos(string) : void @06000004
        GenerateIV() : string @06000001
        logskeeping(KeyValuePair<string, string>) : void @06000005
        Main(string[]) : void @06000003
        PasswordHash : string @04000001
        SaltKey : string @04000002
```

We'll start with analysing the main method. It appears to enumerate through files with the extension ".topsekretfile" in the same directory and call the method ensue_chaos, passing the filepath as an argument, as shown below.



```
private static void Main(string[] args)
{
    string target_ext = ".topsekretfile";
    DirectoryInfo d = new DirectoryInfo(Directory.GetCurrentDirectory());
    FileInfo[] Files = d.GetFiles("*" + target_ext);
    foreach (FileInfo file in Files)
    {
        Program.ensue_chaos(file.FullName);
    }
}
```

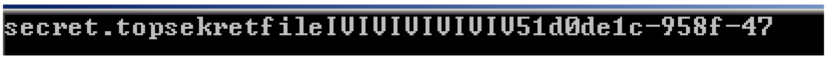
That means we're interested in ensue_chaos method next. Analyzing this method, it appears to read a file, encrypt the file with the IV being generated through GenerateIV method (which calls System.Guid.NewGuid method to generate a unique GUID), call a method called logskeeping (which takes in parameter t that is of type KeyValuePair, which is storing the filename and the IV), and lastly create a text file with the 'ransom note', if it doesn't exist.

If our objective is to decrypt the files, then we're next interested in the encryption method (method name: Encrypt). It appears to be using an AES encryption with a secret key generated from a pre-defined password hash ("ADMIN") & salt key ("SALTYKEY") to derive the secret key, and the generated IV that was passed in as a parameter.

Lastly, since the IV key seems to be passed into logskeeping method, that's what we're going to look at next. Looking at it, it appears to be writing the filename as well as the IV used to encrypt that file inside a file called desktop.ini. However, it appears to be 'encrypting/decrypting' the file with a simple XOR key of 293.

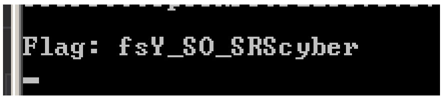
Having all this info, it means that the IV key that was used to encrypt the file "secret" is likely inside the file desktop.ini that was provided, and the method logskeeping appears to be decrypting the file each time to append, re-encrypt, and re-write the file. Therefore, we can use part of the code there (snippet shared below) to decrypt the desktop.ini file.

After decrypting the file 'desktop.ini', we can see that indeed a file named "secret.topsekretfile" was encrypted with the IV key "51d0de1c-958f-47" (as shown below).



```
secret.topsekretfileIVIVIVIVIVIV51d0de1c-958f-47
```

So given that we have all the variables that were used to generate the symmetric key for the encryption, we can write a decryptor (snippet shared below) to decrypt the file. Upon decrypting the file, we get the flag (as shown below).



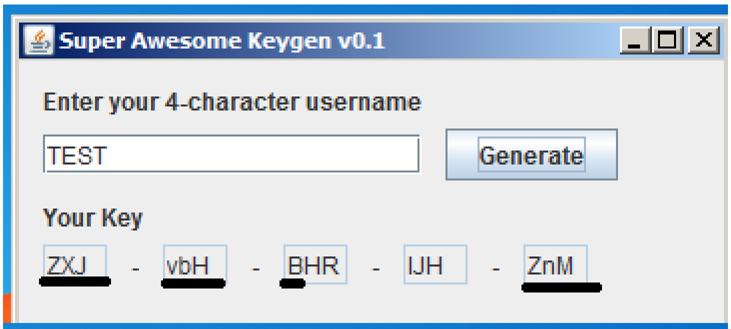
```
Flag: fsY_SO_SRScyber
_
```

```
//C# code snippet

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;namespace ConsoleApp4
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(readLog()); //print out the content of decrypted log file to get the IV key
            string filepath = "secret.joker.saw"; // encrypted file that is placed in same directory as
running code
            byte[] s = File.ReadAllBytes(filepath);
            Console.WriteLine(Decrypt(s,"51d0de1c-958f-47")); //IV key was extracted from the
decrypted log file "desktop.ini".
        }
        public static string readLog()
        {
            List<byte> joker = new List<byte>();
            using (StreamReader file = File.OpenText("desktop.ini"))
            {
                string s = file.ReadToEnd();
                byte[] anarchy = Encoding.UTF8.GetBytes(s);
                foreach (byte b in anarchy)
                {
                    joker.Add((byte)((int)b ^ 293));
                }
            }
            return Encoding.UTF8.GetString(joker.ToArray(), 0, joker.Count<byte>());
        }
        private static readonly string PasswordHash = "ADMIN";
        private static readonly string SaltKey = "SALTYKEY";
        public static string Decrypt(byte[] cipherTextBytes, string IV)
        {
            byte[] keyBytes = new Rfc2898DeriveBytes(Program.PasswordHash,
Encoding.ASCII.GetBytes(Program.SaltKey)).GetBytes(32);
            RijndaelManaged symmetricKey = new RijndaelManaged
            {
                Mode = CipherMode.CBC,
                Padding = PaddingMode.None
            };
            ICryptoTransform decryptor = symmetricKey.CreateDecryptor(keyBytes,
Encoding.ASCII.GetBytes(IV));
            MemoryStream memoryStream = new MemoryStream(cipherTextBytes);
            CryptoStream cryptoStream = new CryptoStream(memoryStream, decryptor,
CryptoStreamMode.Read);
            byte[] plainTextBytes = new byte[cipherTextBytes.Length];
            int decryptedByteCount = cryptoStream.Read(plainTextBytes, 0, plainTextBytes.Length);
            memoryStream.Close();
            cryptoStream.Close();
            return Encoding.UTF8.GetString(plainTextBytes, 0,
decryptedByteCount).TrimEnd("\0".ToCharArray());
        }
    }
}
```

3) TheSuperAwesomeKeyGen:

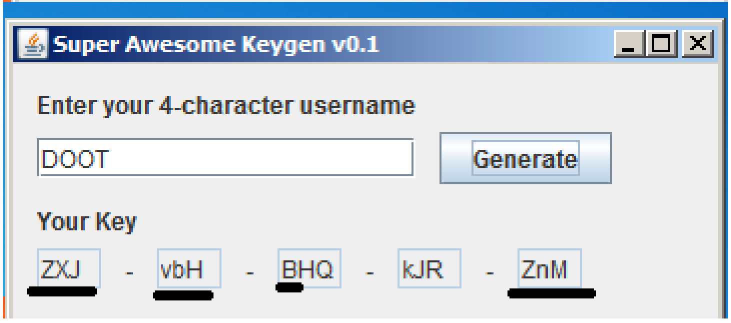
Challenge: <DOWNLOAD>

In this challenge, we're provided a java executable (JAR) file, which appears to take in a 4-character long string to generate some key. Playing around with different input, we can see that the first 6 and last 3 characters of the generated key are always constant (shown below).



Enter your 4-character username: TEST

Your Key: ZXJ - vbH - BHR - IJH - ZnM

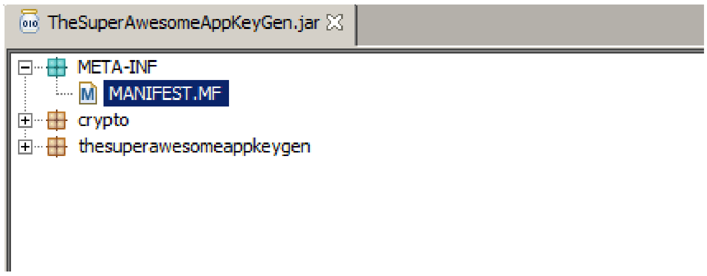


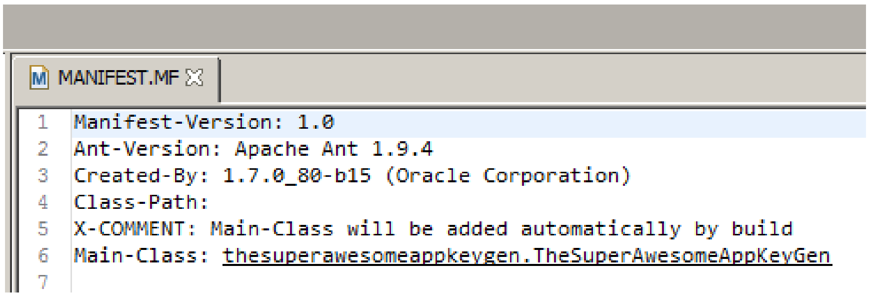
Enter your 4-character username: DOOT

Your Key: ZXJ - vbH - BHQ - kJR - ZnM

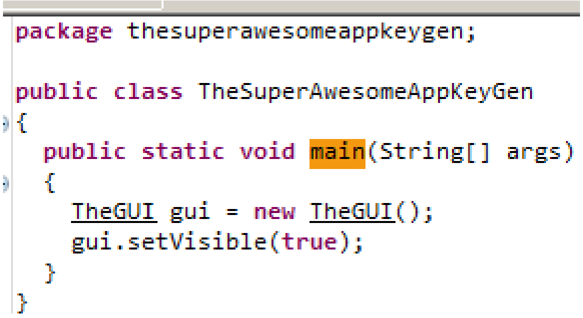
Now we decompile the JAR file to analyze the java code behind it.

After decompiling, we can see 2 packages, namely crypto and thesuperawesomeappkeygen. From the manifest file, we can identify that the main method is located in thesuperawesomeappkeygen.TheSuperAwesomeAppKeyGen class, which seems to just create an instance of TheGUI class and set its visible property to true, as shown below.



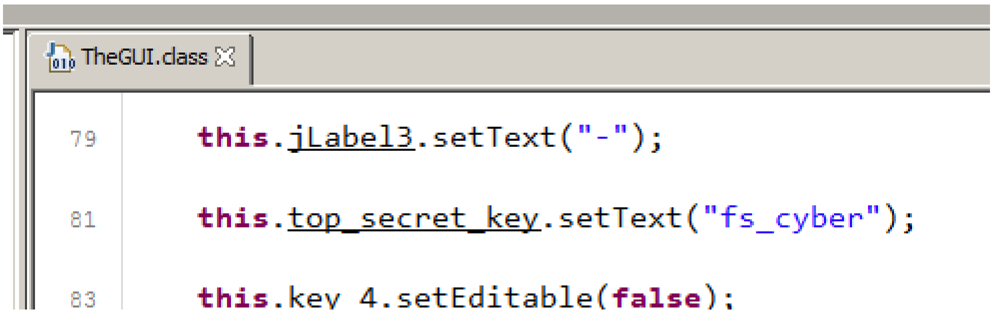


```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.9.4
Created-By: 1.7.0_80-b15 (Oracle Corporation)
Class-Path: 
X-COMMENT: Main-Class will be added automatically by build
Main-Class: thesuperawesomeappkeygen.TheSuperAwesomeAppKeyGen

```



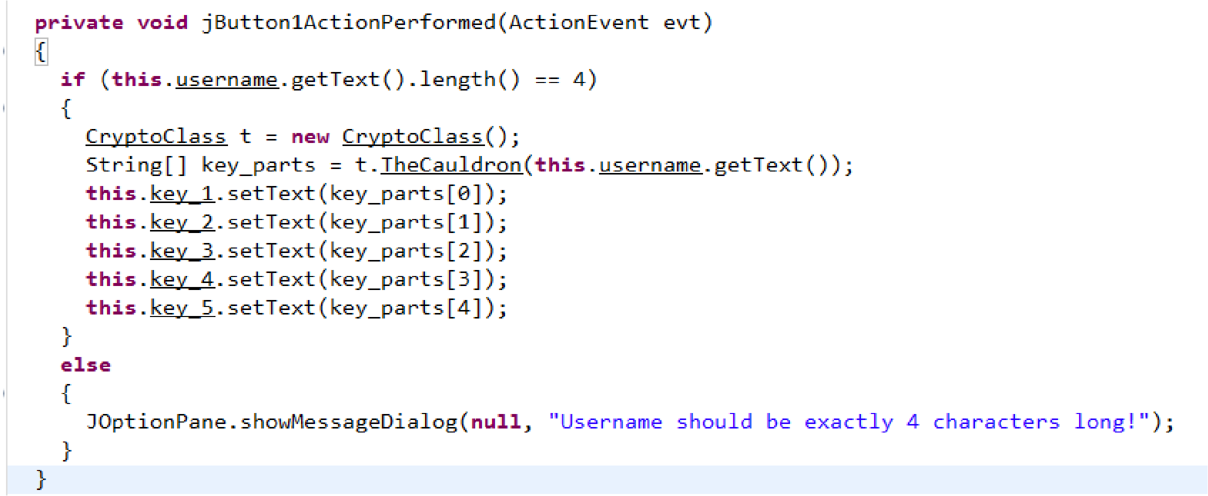
```
package thesuperawesomeappkeygen;

public class TheSuperAwesomeAppKeyGen
{
  public static void main(String[] args)
  {
    TheGUI gui = new TheGUI();
    gui.setVisible(true);
  }
}
```

Inside the constructor of TheGUI class, we can see that after calling the initComponents() function to initialize the GUI components, it sets a field called "top_secret_key" to false, and searching the class for top_secret_key, we can see that it stores the string "fs_cyber", as shown below.



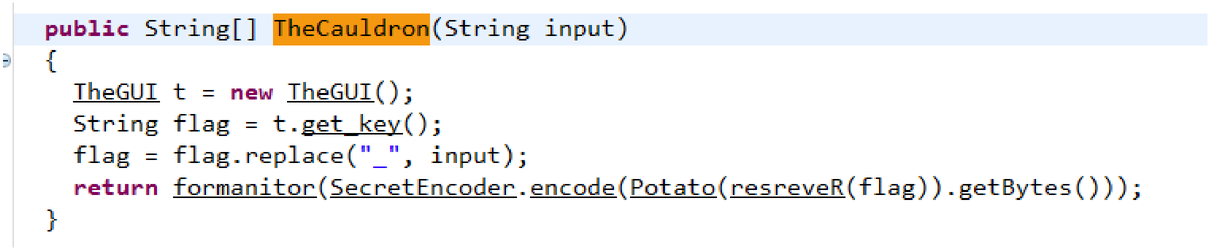
```
    this.jLabel3.setText("-");

    this.top_secret_key.setText("fs_cyber");

    this.key_4.setEditable(false);
```

Given that the button "Generate" in the application generates the key, we can see that there is a JButton object named jButton1 which refers to the button, with an action listener method called jButtonActionPerformed (as shown below).



```
private void jButton1ActionPerformed(ActionEvent evt)
{
  if (this.username.getText().length() == 4)
  {
    CryptoClass t = new CryptoClass();
    String[] key_parts = t.TheCauldron(this.username.getText());
    this.key_1.setText(key_parts[0]);
    this.key_2.setText(key_parts[1]);
    this.key_3.setText(key_parts[2]);
    this.key_4.setText(key_parts[3]);
    this.key_5.setText(key_parts[4]);
  }
  else
  {
    JOptionPane.showMessageDialog(null, "Username should be exactly 4 characters long!");
  }
}
```

Inside this method, it appears to pass the inputted username as a parameter into the method crypto.CryptoClass.TheCauldron. Following this method, we can see that the method calls get_key() function from TheGUI class, which returns value of top_secret_key ("fs_cyber"). It replaces the underscore with the inputted username (e.g. username: TEST, value will be fsTESTcyber) and passes the bytes representation of the string into a function called resreveR, the result of which is passed into a method called Potato, result of which is passed into SecretEncoder.encode, result of which is finally passed into formanitor, as shown below.



```
public String[] TheCauldron(String input)
{
  TheGUI t = new TheGUI();
  String flag = t.get_key();
  flag = flag.replace("_", input);
  return formanitor(SecretEncoder.encode(Potato(resreveR(flag)).getBytes()));
}
```

Looking at each method, these are their functionalities:

1. resreveR – Reverse string
2. Potato – ROT13 implementation
3. SecretEncoder.encode – base64 implementation with charset of A-Za-z0-9
4. forminator – Split the provided string into 5 parts, and return an array of length 5 with the splitted parts.

Given these information, and that we are provided with ZXJ-vbH-BFQ-kJH-ZnM, we can start reversing all of these encoding techniques, so first we concatenate the key parts to get ZXJvbHBFQkJHZnM, which is encoded b64, decoding it returns erolpEBBGfs, which is then inputted into a ROT13 function, which returns rebycROOTsf, which when reversed returns the flag.

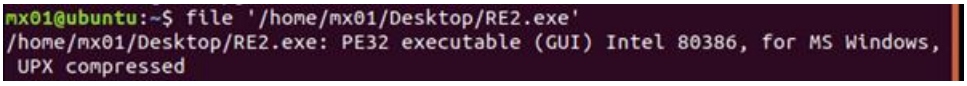
Flag:fsTOORcyber

4) Packed up:

Challenge: <DOWNLOAD>

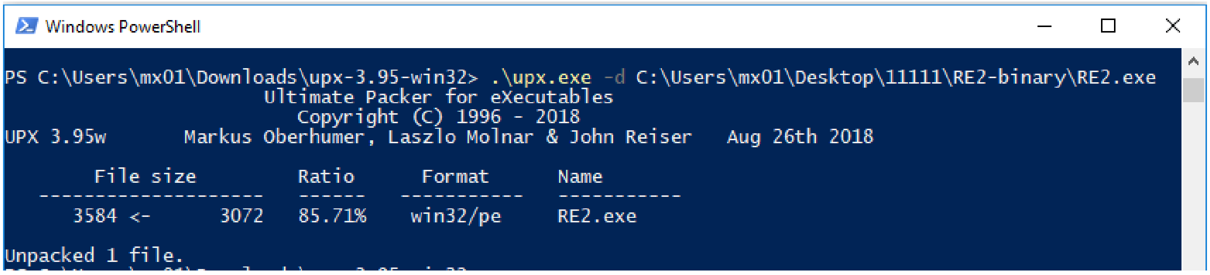
This challenge requires participant to identify the packer used and decompress the file.

Using PE identification in Linux tool we are able to identify that this file is packed with UPX.



```
mx01@ubuntu:~$ file '/home/mx01/Desktop/RE2.exe'
/home/mx01/Desktop/RE2.exe: PE32 executable (GUI) Intel 80386, for MS Windows,
 UPX compressed
```

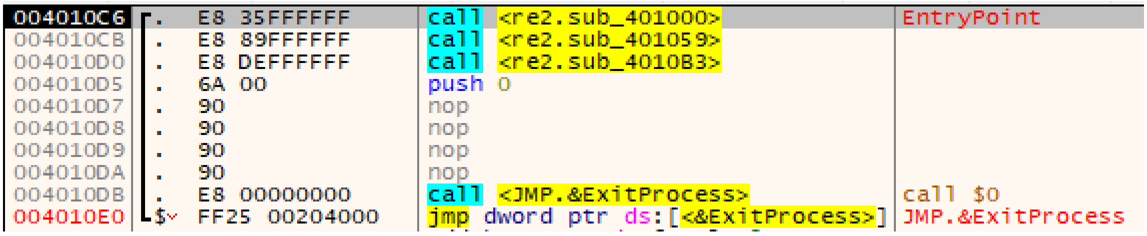
Using UPX tool we are able to unpack the PE file.



```
PS C:\Users\mx01\Downloads\upx-3.95-win32> .\upx.exe -d C:\Users\mx01\Desktop\11111\RE2-binary\RE2.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95w       Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
      3584 <-      3072   85.71%    win32/pe     RE2.exe

Unpacked 1 file.
```

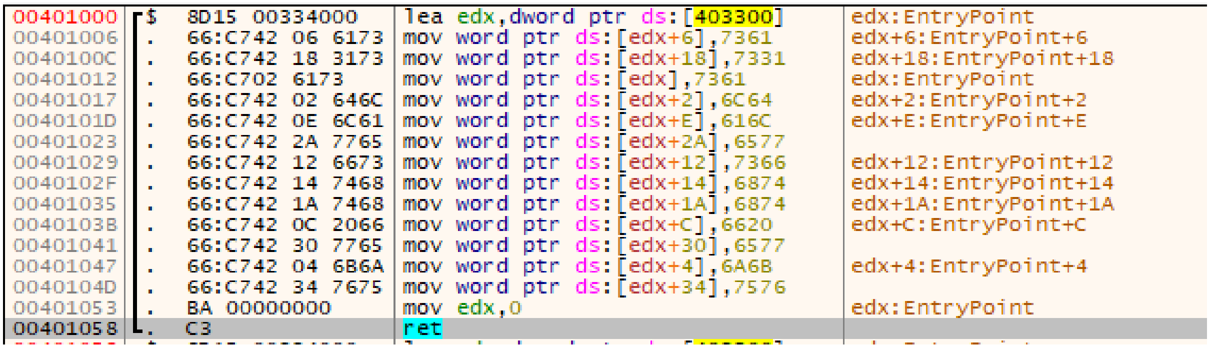
After decompressing the file, We can load the file to a disassembler and we are able to see three CALL function.

Main Function:



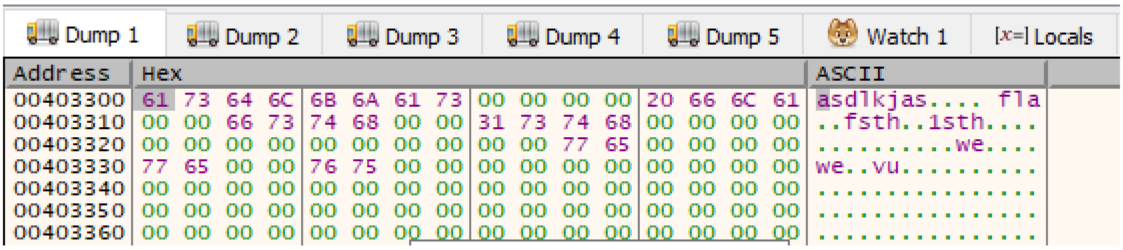
```
004010C6 ┌.  E8 35FFFFFF      call <re2.sub_401000>                    EntryPoint
004010CB │.  E8 89FFFFFF      call <re2.sub_401059>
004010D0 │.  E8 DEFFFFFF      call <re2.sub_4010B3>
004010D5 │.  6A 00            push 0
004010D7 │.  90               nop
004010D8 │.  90               nop
004010D9 │.  90               nop
004010DA │.  90               nop
004010DB │.  E8 00000000      call <JMP.&ExitProcess>                  call $0
004010E0 └$v FF25 00204000    jmp dword ptr ds:[<&ExitProcess>]        JMP.&ExitProcess
```

Function 1:



```
00401000 ┌$  8D15 00334000     lea edx,dword ptr ds:[403300]          edx:EntryPoint
00401006 │.  66:C742 06 6173   mov word ptr ds:[edx+6],7361           edx+6:EntryPoint+6
0040100C │.  66:C742 18 3173   mov word ptr ds:[edx+18],7331          edx+18:EntryPoint+18
00401012 │.  66:C702 6173      mov word ptr ds:[edx],7361             edx:EntryPoint
00401017 │.  66:C742 02 646C   mov word ptr ds:[edx+2],6C64           edx+2:EntryPoint+2
0040101D │.  66:C742 0E 6C61   mov word ptr ds:[edx+E],616C           edx+E:EntryPoint+E
00401023 │.  66:C742 2A 7765   mov word ptr ds:[edx+2A],6577
00401029 │.  66:C742 12 6673   mov word ptr ds:[edx+12],7366          edx+12:EntryPoint+12
0040102F │.  66:C742 14 7468   mov word ptr ds:[edx+14],6874          edx+14:EntryPoint+14
00401035 │.  66:C742 1A 7468   mov word ptr ds:[edx+1A],6874          edx+1A:EntryPoint+1A
0040103B │.  66:C742 0C 2066   mov word ptr ds:[edx+C],6620           edx+C:EntryPoint+C
00401041 │.  66:C742 30 7765   mov word ptr ds:[edx+30],6577
00401047 │.  66:C742 04 6B6A   mov word ptr ds:[edx+4],6A6B           edx+4:EntryPoint+4
0040104D │.  66:C742 34 7675   mov word ptr ds:[edx+34],7576
00401053 │.  BA 00000000       mov edx,0                              edx:EntryPoint
00401058 └.  C3                ret
```

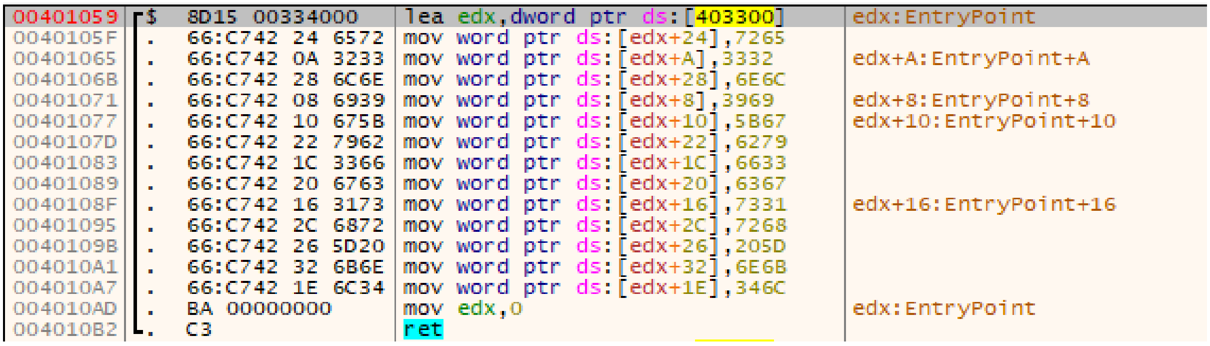
We can see that there are immediate constant being moved to EDX pointer. Let us inspect what is being written in EDX dump



```
Address   Hex                                               ASCII
00403300  61 73 64 6C 6B 6A 61 73 00 00 00 00 20 66 6C 61  asdlkjas.... fla
00403310  00 00 66 73 74 68 00 00 31 73 74 68 00 00 00 00  ..fsth..1sth....
00403320  00 00 00 00 00 00 00 00 00 00 77 65 00 00 00 00  ..........we....
00403330  77 65 00 00 76 75 00 00 00 00 00 00 00 00 00 00  we..vu..........
00403340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

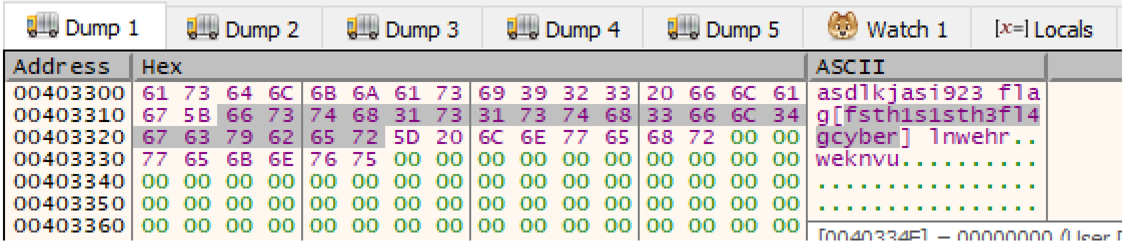
We are able to see that strings are being written after all move was executed.

Function 2:



```
00401059 ┌$  8D15 00334000     lea edx,dword ptr ds:[403300]          edx:EntryPoint
0040105F │.  66:C742 24 6572   mov word ptr ds:[edx+24],7265
00401065 │.  66:C742 0A 3233   mov word ptr ds:[edx+A],3332           edx+A:EntryPoint+A
0040106B │.  66:C742 28 6C6E   mov word ptr ds:[edx+28],6E6C
00401071 │.  66:C742 08 6939   mov word ptr ds:[edx+8],3969           edx+8:EntryPoint+8
00401077 │.  66:C742 10 675B   mov word ptr ds:[edx+10],5B67          edx+10:EntryPoint+10
0040107D │.  66:C742 22 7962   mov word ptr ds:[edx+22],6279
00401083 │.  66:C742 1C 3366   mov word ptr ds:[edx+1C],6633
00401089 │.  66:C742 20 6763   mov word ptr ds:[edx+20],6367
0040108F │.  66:C742 16 3173   mov word ptr ds:[edx+16],7331          edx+16:EntryPoint+16
00401095 │.  66:C742 2C 6872   mov word ptr ds:[edx+2C],7268
0040109B │.  66:C742 26 5D20   mov word ptr ds:[edx+26],205D
004010A1 │.  66:C742 32 6B6E   mov word ptr ds:[edx+32],6E6B
004010A7 │.  66:C742 1E 6C34   mov word ptr ds:[edx+1E],346C
004010AD │.  BA 00000000       mov edx,0                              edx:EntryPoint
004010B2 └.  C3                ret
```

Continuing to Function 2, we are still able to see movement of string to EDX pointer



```
Address   Hex                                              ASCII
00403300  61 73 64 6C 6B 6A 61 73 69 39 32 33 20 66 6C 61  asdlkjasi923 fla
00403310  67 5B 66 73 74 68 31 73 31 73 74 68 33 66 6C 34  g[fsth1s1sth3fl4
00403320  67 63 79 62 65 72 5D 20 6C 6E 77 65 68 72 00 00  gcyber] lnwehr..
00403330  77 65 6B 6E 76 75 00 00 00 00 00 00 00 00 00 00  weknvu..........
00403340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040334E] = 00000000 (User
```

Finally, after all MOV was executed, we are able to identify the flag from the dump.

Flag:

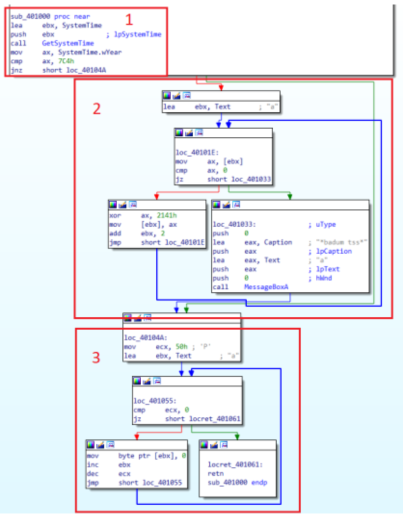
fsth1s1sth3fl4gcyber

5) What time is it?:

Challenge: <DOWNLOAD>

This challenge requires participant to bypass/satisfy system check by the application so it will procced to decryption of the flag.

Inspecting function 401000:



```
sub_401000 proc near
lea     ebx, SystemTime
push    ebx             ; lpSystemTime
call    GetSystemTime
mov     ax, SystemTime.wYear
cmp     ax, 7C4h
jnz     short loc_40104A

lea     ebx, Text       ; "a"

loc_40101E:
mov     ax, [ebx]
cmp     ax, 0
jz      short loc_401033

xor     ax, 2141h
mov     [ebx], ax
add     ebx, 2
jmp     short loc_40101E

loc_401033:             ; uType
push    0
lea     eax, Caption    ; "*badum tss*"
push    eax             ; lpCaption
lea     eax, Text       ; "a"
push    eax             ; lpText
push    0               ; hWnd
call    MessageBoxA

loc_40104A:
mov     ecx, 50h ; 'P'
lea     ebx, Text       ; "a"

loc_401055:
cmp     ecx, 0
jz      short locret_401061

mov     byte ptr [ebx], 0
inc     ebx
dec     ecx
jmp     short loc_401055

locret_401061:
retn
sub_401000 endp
```

We divided the analysis in 3 parts

1. System time check

    The code checks if the system time and compares if the year is 7C4h (1988 in decimal) . If the year is satisfied then the code will proceed will jump to part 2, else, it will jump to part 3.

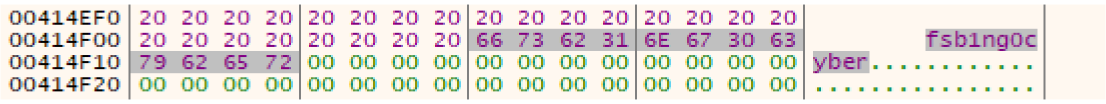
2. Flag decryption

    The flag is decrypted stored in EBX, the flag will be decrypted every WORD (2bytes) using XOR key (2141h). Flag can be identified at the bottom of 20h stream after decryption. The execution will then proceed to part 3.

3. Null out variables

    Values stored in EBX will be over written with an immediate constant of 0

Flag:



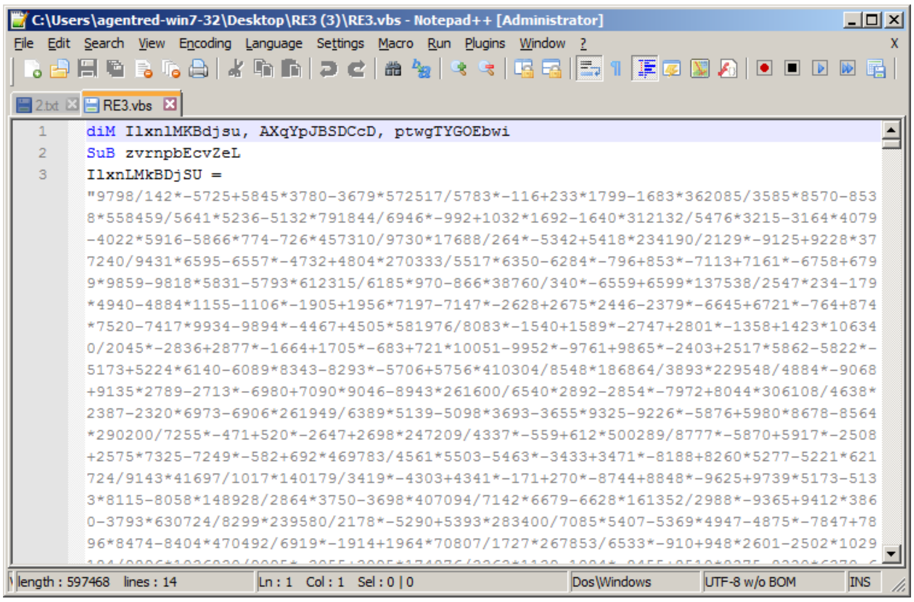
```
00414EF0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00414F00  20 20 20 20 20 20 20 20 66 73 62 31 6E 67 30 63          fsb1ng0c
00414F10  79 62 65 72 00 00 00 00 00 00 00 00 00 00 00 00  yber............
00414F20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

fsb1ng0cyber

6) Funny bunny:

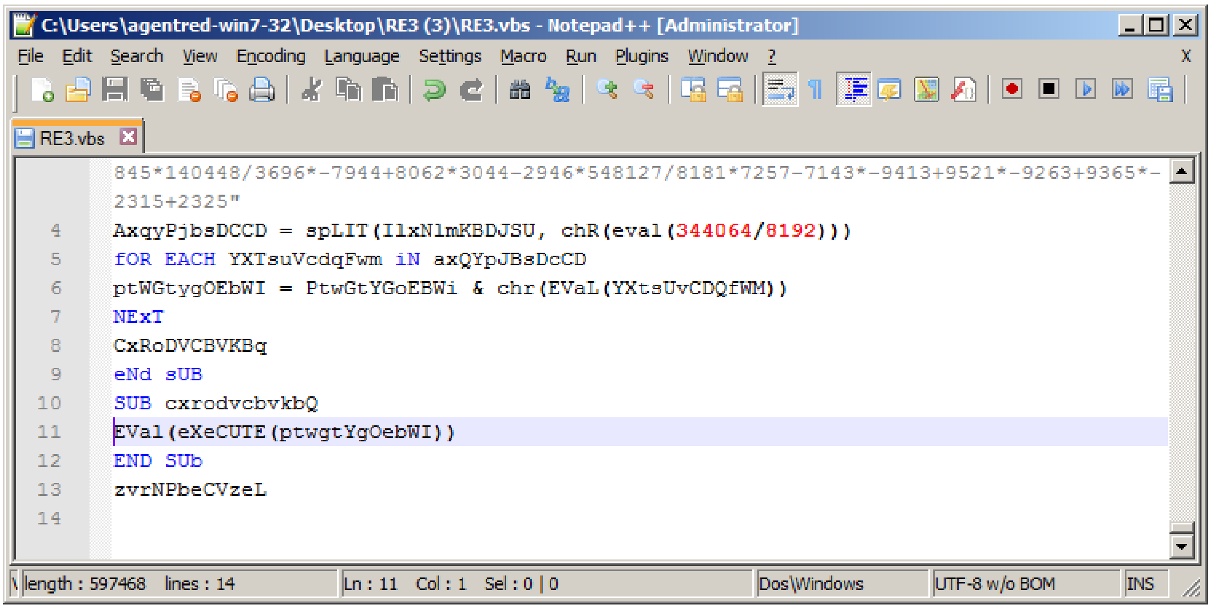
Challenge: <DOWNLOAD>

This challenge requires participants to analyse obfuscated VBS script and find the flag. This can be solved by piping out variables to a file and inspect the output of the script.


C:\Users\agentred-win7-32\Desktop\RE3 (3)\RE3.vbs - Notepad++ [Administrator]
File
Edit
Search
View
Encoding
Language
Settings
Macro
Run
Plugins
Window
?
2.txt
RE3.vbs
length : 597468 lines : 14
Ln : 1 Col : 1 Sel : 0 | 0
Dos\Windows
UTF-8 w/o BOM
INS

```
diM IlxnlMKBdjsu, AXqYpJBSDCcD, ptwgTYGOEbwi
SuB zvrnpbEcvZeL
IlxnLMkBDjSU =
"9798/142*-5725+5845*3780-3679*572517/5783*-116+233*1799-1683*362085/3585*8570-853
8*558459/5641*5236-5132*791844/6946*-992+1032*1692-1640*312132/5476*3215-3164*4079
-4022*5916-5866*774-726*457310/9730*17688/264*-5342+5418*234190/2129*-9125+9228*37
7240/9431*6595-6557*-4732+4804*270333/5517*6350-6284*-796+853*-7113+7161*-6758+679
9*9859-9818*5831-5793*612315/6185*970-866*38760/340*-6559+6599*137538/2547*234-179
*4940-4884*1155-1106*-1905+1956*7197-7147*-2628+2675*2446-2379*-6645+6721*-764+874
*7520-7417*9934-9894*-4467+4505*581976/8083*-1540+1589*-2747+2801*-1358+1423*10634
0/2045*-2836+2877*-1664+1705*-683+721*10051-9952*-9761+9865*-2403+2517*5862-5822*-
5173+5224*6140-6089*8343-8293*-5706+5756*410304/8548*186864/3893*229548/4884*-9068
+9135*2789-2713*-6980+7090*9046-8943*261600/6540*2892-2854*-7972+8044*306108/4638*
2387-2320*6973-6906*261949/6389*5139-5098*3693-3655*9325-9226*-5876+5980*8678-8564
*290200/7255*-471+520*-2647+2698*247209/4337*-559+612*500289/8777*-5870+5917*-2508
+2575*7325-7249*-582+692*469783/4561*5503-5463*-3433+3471*-8188+8260*5277-5221*621
724/9143*41697/1017*140179/3419*-4303+4341*-171+270*-8744+8848*-9625+9739*5173-513
3*8115-8058*148928/2864*3750-3698*407094/7142*6679-6628*161352/2988*-9365+9412*386
0-3793*630724/8299*239580/2178*-5290+5393*283400/7085*5407-5369*4947-4875*-7847+78
96*8474-8404*470492/6919*-1914+1964*70807/1727*267853/6533*-910+948*2601-2502*1029
```

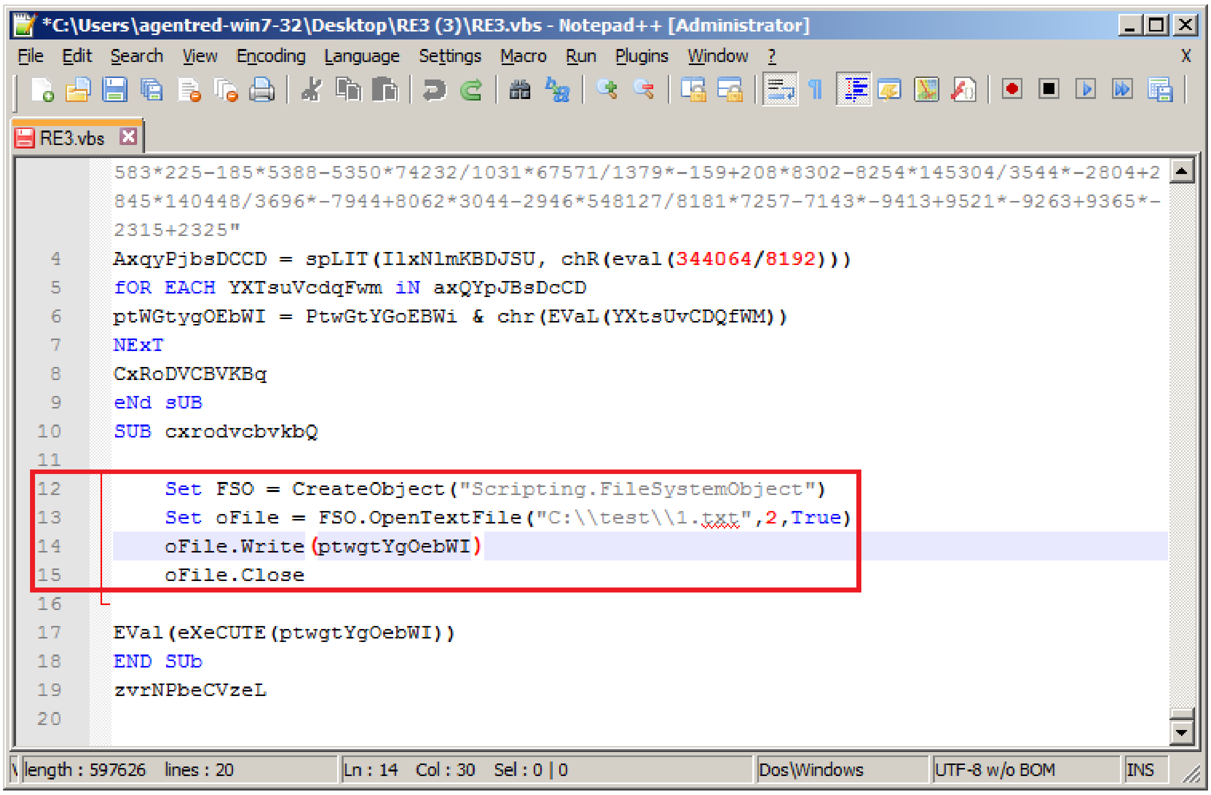
First layer of the VBS script


C:\Users\agentred-win7-32\Desktop\RE3 (3)\RE3.vbs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
RE3.vbs

```
845*140448/3696*-7944+8062*3044-2946*548127/8181*7257-7143*-9413+9521*-9263+9365*-
2315+2325"
AxqyPjbsDCCD = spLIT(IlxNlmKBDJSU, chR(eval(344064/8192)))
fOR EACH YXTsuVcdqFwm iN axQYpJBsDcCD
ptWGtygOEbWI = PtwGtYGoEBWi & chr(EVaL(YXtsUvCDQfWM))
NExT
CxRoDVCBVKBq
eNd sUB
SUB cxrodvcbvkbQ
EVal(eXeCUTE(ptwgtYgOebWI))
END SUb
zvrNPbeCVzeL
```

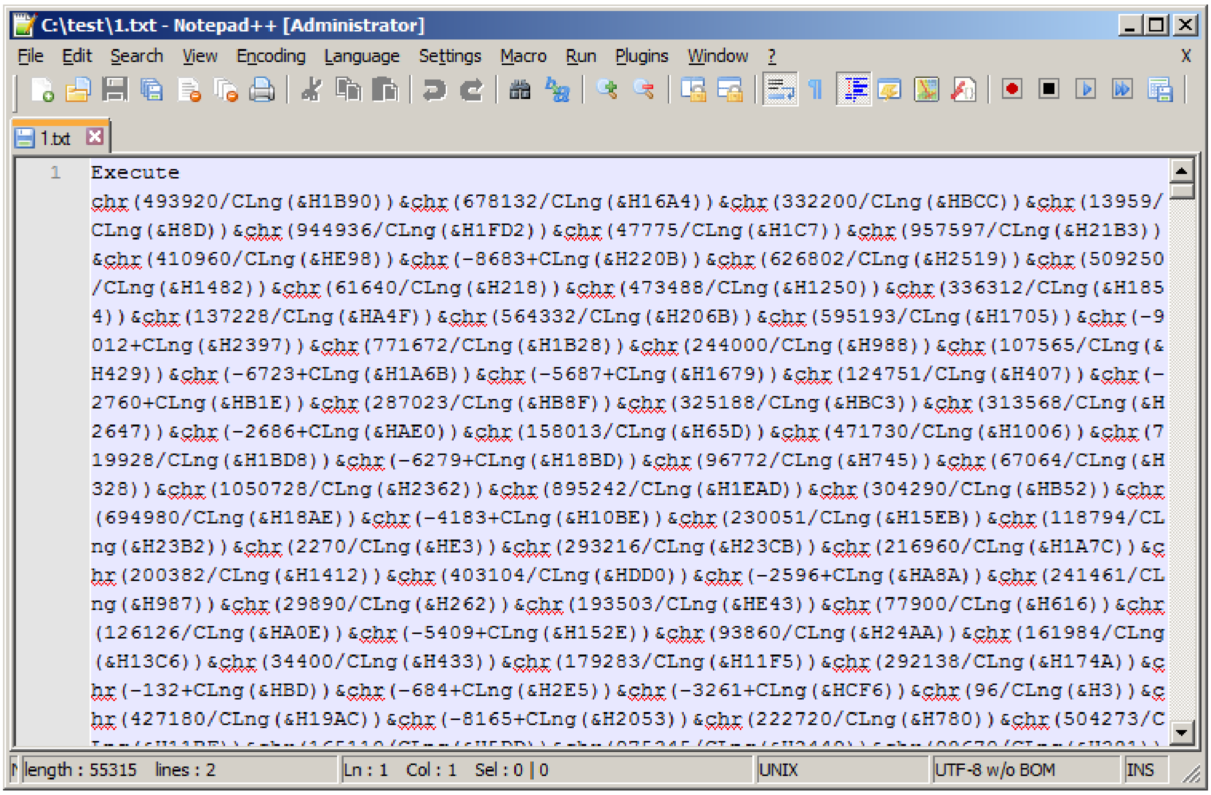
length : 597468 lines : 14
Ln : 11 Col : 1 Sel : 0 | 0
Dos\Windows
UTF-8 w/o BOM
INS

We can see at the bottom of the code that there is a loop to decode then after will invoke function “CxRoDVCBVKBq”. This function will Eval() variable “ptwgtYgOebWI”.


*C:\Users\agentred-win7-32\Desktop\RE3 (3)\RE3.vbs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
RE3.vbs

```
583*225-185*5388-5350*74232/1031*67571/1379*-159+208*8302-8254*145304/3544*-2804+2
845*140448/3696*-7944+8062*3044-2946*548127/8181*7257-7143*-9413+9521*-9263+9365*-
2315+2325"
AxqyPjbsDCCD = spLIT(IlxNlmKBDJSU, chR(eval(344064/8192)))
fOR EACH YXTsuVcdqFwm iN axQYpJBsDcCD
ptWGtygOEbWI = PtwGtYGoEBWi & chr(EVaL(YXtsUvCDQfWM))
NExT
CxRoDVCBVKBq
eNd sUB
SUB cxrodvcbvkbQ

    Set FSO = CreateObject("Scripting.FileSystemObject")
    Set oFile = FSO.OpenTextFile("C:\\test\\1.txt",2,True)
    oFile.Write (ptwgtYgOebWI)
    oFile.Close

EVal(eXeCUTE(ptwgtYgOebWI))
END SUb
zvrNPbeCVzeL
```

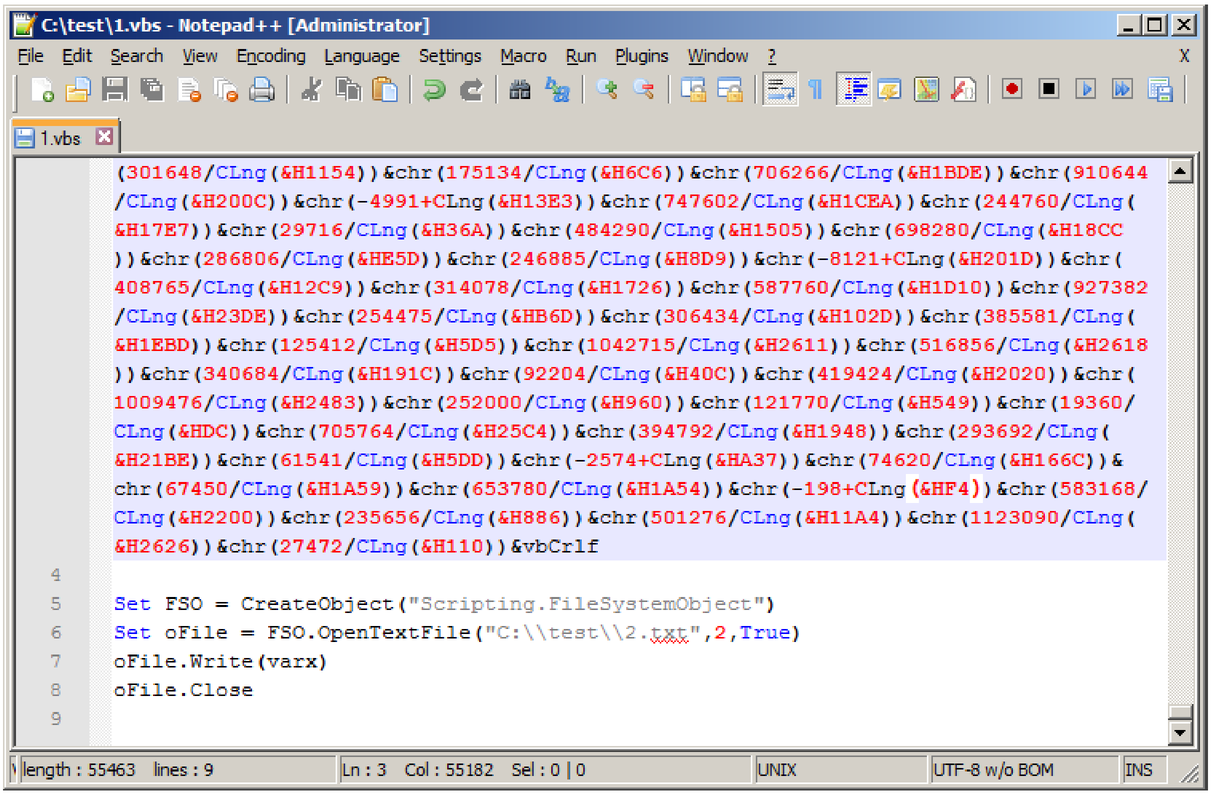
Using a file write function, we can dump the content of “ptwgtYgOebWI” to a file.


C:\test\1.txt - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
1.txt
length : 55315 lines : 2
Ln : 1 Col : 1 Sel : 0 | 0
UNIX
UTF-8 w/o BOM
INS

```
1   Execute
    chr(493920/CLng(&H1B90))&chr(678132/CLng(&H16A4))&chr(332200/CLng(&HBCC))&chr(13959/
    CLng(&H8D))&chr(944936/CLng(&H1FD2))&chr(47775/CLng(&H1C7))&chr(957597/CLng(&H21B3))
    &chr(410960/CLng(&HE98))&chr(-8683+CLng(&H220B))&chr(626802/CLng(&H2519))&chr(509250
    /CLng(&H1482))&chr(61640/CLng(&H218))&chr(473488/CLng(&H1250))&chr(336312/CLng(&H185
    4))&chr(137228/CLng(&HA4F))&chr(564332/CLng(&H206B))&chr(595193/CLng(&H1705))&chr(-9
    012+CLng(&H2397))&chr(771672/CLng(&H1B28))&chr(244000/CLng(&H988))&chr(107565/CLng(&
    H429))&chr(-6723+CLng(&H1A6B))&chr(-5687+CLng(&H1679))&chr(124751/CLng(&H407))&chr(-
    2760+CLng(&HB1E))&chr(287023/CLng(&HB8F))&chr(325188/CLng(&HBC3))&chr(313568/CLng(&H
    2647))&chr(-2686+CLng(&HAE0))&chr(158013/CLng(&H65D))&chr(471730/CLng(&H1006))&chr(7
    19928/CLng(&H1BD8))&chr(-6279+CLng(&H18BD))&chr(96772/CLng(&H745))&chr(67064/CLng(&H
    328))&chr(1050728/CLng(&H2362))&chr(895242/CLng(&H1EAD))&chr(304290/CLng(&HB52))&chr
    (694980/CLng(&H18AE))&chr(-4183+CLng(&H10BE))&chr(230051/CLng(&H15EB))&chr(118794/CL
    ng(&H23B2))&chr(2270/CLng(&HE3))&chr(293216/CLng(&H23CB))&chr(216960/CLng(&H1A7C))&c
    hr(200382/CLng(&H1412))&chr(403104/CLng(&HDD0))&chr(-2596+CLng(&HA8A))&chr(241461/CL
    ng(&H987))&chr(29890/CLng(&H262))&chr(193503/CLng(&HE43))&chr(77900/CLng(&H616))&chr
    (126126/CLng(&HA0E))&chr(-5409+CLng(&H152E))&chr(93860/CLng(&H24AA))&chr(161984/CLng
    (&H13C6))&chr(34400/CLng(&H433))&chr(179283/CLng(&H11F5))&chr(292138/CLng(&H174A))&c
    hr(-132+CLng(&HBD))&chr(-684+CLng(&H2E5))&chr(-3261+CLng(&HCF6))&chr(96/CLng(&H3))&c
    hr(427180/CLng(&H19AC))&chr(-8165+CLng(&H2053))&chr(222720/CLng(&H780))&chr(504273/C
```

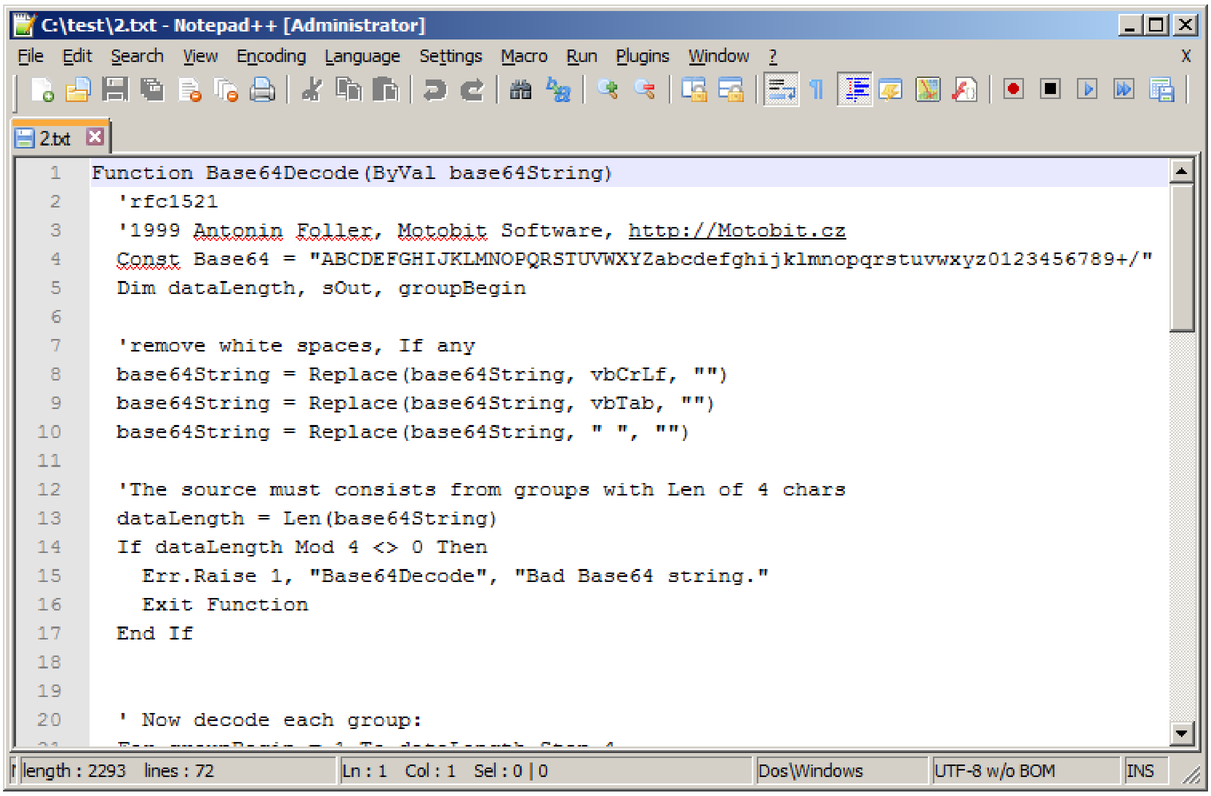
This is the content of variable “ptwgtYgOebWI” and the second layer of the code. This code Executes encoded parameter in line 1. In order for us to identify what is being executed, let us replace Execute (from line 1) with a variable.

Dim varx: varx = chr(493920/CLng(&H1B90)) &chr(678132/CLng(&H16A4)) … <encoded stream>


C:\test\1.vbs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
1.vbs
length : 55463 lines : 9
Ln : 3 Col : 55182 Sel : 0 | 0
UNIX
UTF-8 w/o BOM
INS

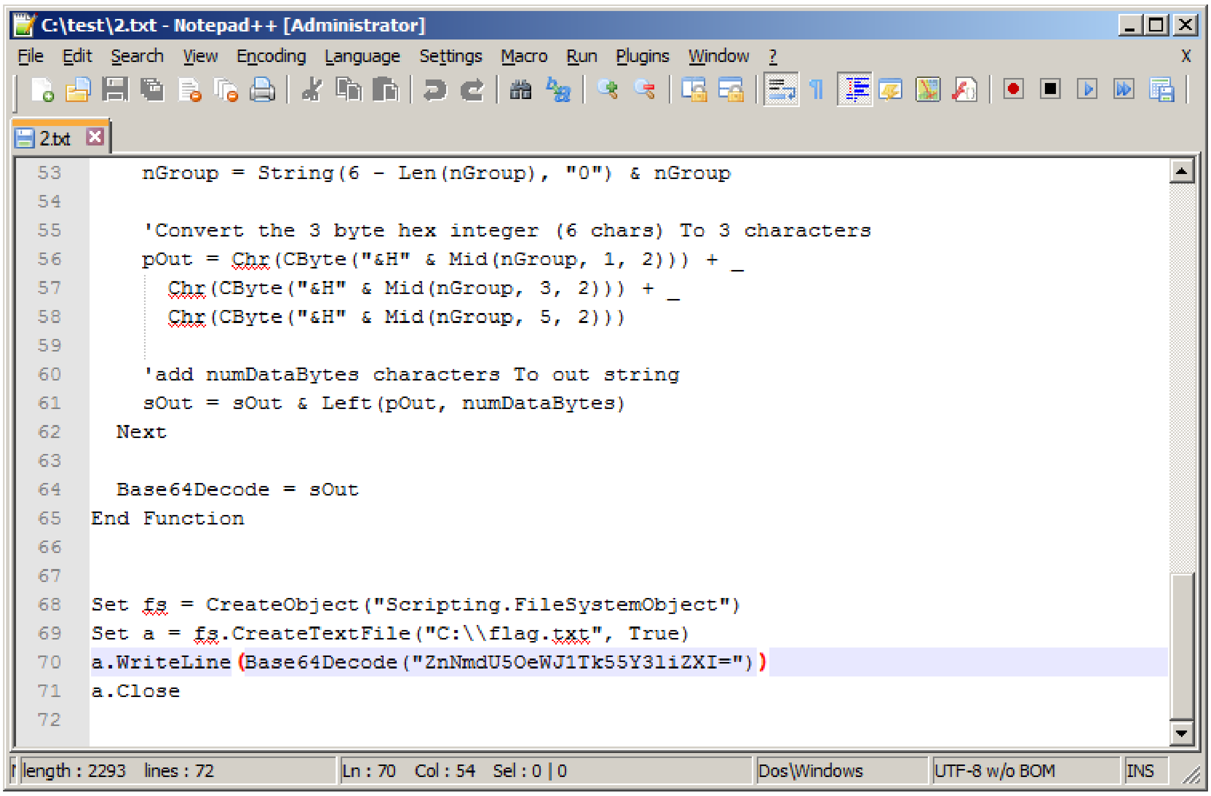
```
    (301648/CLng(&H1154))&chr(175134/CLng(&H6C6))&chr(706266/CLng(&H1BDE))&chr(910644
    /CLng(&H200C))&chr(-4991+CLng(&H13E3))&chr(747602/CLng(&H1CEA))&chr(244760/CLng(
    &H17E7))&chr(29716/CLng(&H36A))&chr(484290/CLng(&H1505))&chr(698280/CLng(&H18CC
    ))&chr(286806/CLng(&HE5D))&chr(246885/CLng(&H8D9))&chr(-8121+CLng(&H201D))&chr(
    408765/CLng(&H12C9))&chr(314078/CLng(&H1726))&chr(587760/CLng(&H1D10))&chr(927382
    /CLng(&H23DE))&chr(254475/CLng(&HB6D))&chr(306434/CLng(&H102D))&chr(385581/CLng(
    &H1EBD))&chr(125412/CLng(&H5D5))&chr(1042715/CLng(&H2611))&chr(516856/CLng(&H2618
    ))&chr(340684/CLng(&H191C))&chr(92204/CLng(&H40C))&chr(419424/CLng(&H2020))&chr(
    1009476/CLng(&H2483))&chr(252000/CLng(&H960))&chr(121770/CLng(&H549))&chr(19360/
    CLng(&HDC))&chr(705764/CLng(&H25C4))&chr(394792/CLng(&H1948))&chr(293692/CLng(
    &H21BE))&chr(61541/CLng(&H5DD))&chr(-2574+CLng(&HA37))&chr(74620/CLng(&H166C))&
    chr(67450/CLng(&H1A59))&chr(653780/CLng(&H1A54))&chr(-198+CLng(&HF4))&chr(583168/
    CLng(&H2200))&chr(235656/CLng(&H886))&chr(501276/CLng(&H11A4))&chr(1123090/CLng(
    &H2626))&chr(27472/CLng(&H110))&vbCrlf
4
5   Set FSO = CreateObject("Scripting.FileSystemObject")
6   Set oFile = FSO.OpenTextFile("C:\\test\\2.txt",2,True)
7   oFile.Write(varx)
8   oFile.Close
9
```

Variable “varx” will now contain the decoded stream which we dump to a file using file out.


C:\test\2.txt - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
2.txt
length : 2293 lines : 72
Ln : 1 Col : 1 Sel : 0 | 0
Dos\Windows
UTF-8 w/o BOM
INS

```
Function Base64Decode(ByVal base64String)
  'rfc1521
  '1999 Antonin Foller, Motobit Software, http://Motobit.cz
  Const Base64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
  Dim dataLength, sOut, groupBegin

  'remove white spaces, If any
  base64String = Replace(base64String, vbCrLf, "")
  base64String = Replace(base64String, vbTab, "")
  base64String = Replace(base64String, " ", "")

  'The source must consists from groups with Len of 4 chars
  dataLength = Len(base64String)
  If dataLength Mod 4 <> 0 Then
    Err.Raise 1, "Base64Decode", "Bad Base64 string."
    Exit Function
  End If


  ' Now decode each group:
```

The last layer of code is now revealed and contain base64 function.


C:\test\2.txt - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
2.txt
length : 2293 lines : 72
Ln : 70 Col : 54 Sel : 0 | 0
Dos\Windows
UTF-8 w/o BOM
INS

```
    nGroup = String(6 - Len(nGroup), "0") & nGroup

    'Convert the 3 byte hex integer (6 chars) To 3 characters
    pOut = Chr(CByte("&H" & Mid(nGroup, 1, 2))) + _
      Chr(CByte("&H" & Mid(nGroup, 3, 2))) + _
      Chr(CByte("&H" & Mid(nGroup, 5, 2)))

    'add numDataBytes characters To out string
    sOut = sOut & Left(pOut, numDataBytes)
  Next

  Base64Decode = sOut
End Function


Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("C:\\flag.txt", True)
a.WriteLine (Base64Decode("ZnNmdU5OeWJ1Tk55Y3liZXI="))
a.Close
```

It also contain the information of the flag which is encoded with base64.

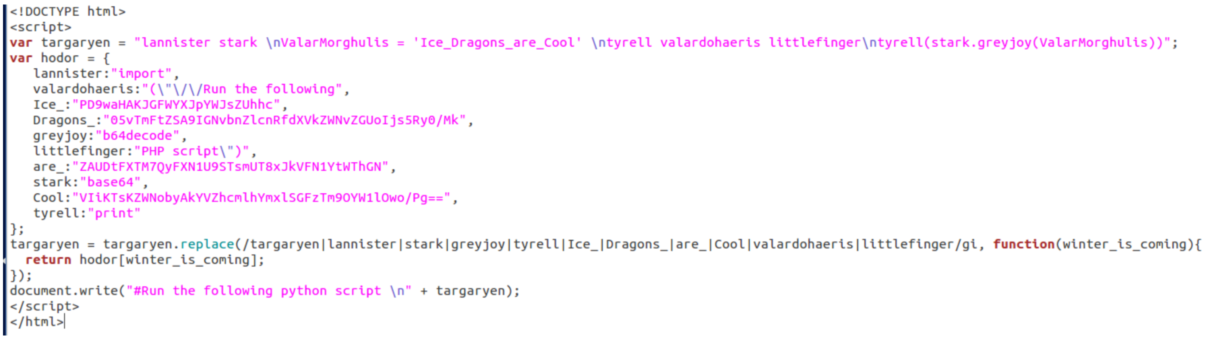
Flag:

fsfuNNybuNNycyber

7) Game Of Scripts:

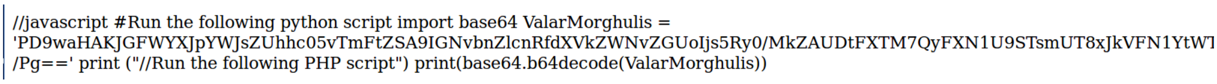
Challenge: <DOWNLOAD>

Provided script is a Javascript file and can be debugged using tools like Malzilla, Revelo, or the Browser itself. One other method is to add <!DOCTYPE html> at start and </html> at the end to script, change the extension to html an run in browser to get output



```
<!DOCTYPE html>
<script>
var targaryen = "lannister stark \nValarMorghulis = 'Ice_Dragons_are_Cool' \ntyrell valardohaeris littlefinger\ntyrell(stark.greyjoy(ValarMorghulis))";
var hodor = {
   lannister:"import",
   valardohaeris:"(\"\/\/Run the following",
   Ice_:"PD9waHAKJGFWYXJpYWJsZUhhc",
   Dragons_:"05vTmFtZSA9IGNvbnZlcnRfdXVkZWNvZGUoIjs5Ry0/Mk",
   greyjoy:"b64decode",
   littlefinger:"PHP script\")",
   are_:"ZAUDtFXTM7QyFXN1U9STsmUT8xJkVFN1YtWThGN",
   stark:"base64",
   Cool:"VIiKTsKZWNobyAkYVZhcmlhYmxlSGFzTm9OYW1lOwo/Pg==",
   tyrell:"print"
};
targaryen = targaryen.replace(/targaryen|lannister|stark|greyjoy|tyrell|Ice_|Dragons_|are_|Cool|valardohaeris|littlefinger/gi, function(winter_is_coming){
  return hodor[winter_is_coming];
});
document.write("#Run the following python script \n" + targaryen);
</script>
</html>
```

Output :



```
//javascript #Run the following python script import base64 ValarMorghulis =
'PD9waHAKJGFWYXJpYWJsZUhhc05vTmFtZSA9IGNvbnZlcnRfdXVkZWNvZGUoIjs5Ry0/MkZAUDtFXTM7QyFXN1U9STsmUT8xJkVFN1YtW
/Pg==' print ("//Run the following PHP script") print(base64.b64decode(ValarMorghulis))
```

Output is a python script, you can execute this script in python or just by looking at the script we can see that there is a base64 encoded data present in the script.

Decode the base64 data
PD9waHAKJGFWYXJpYWJsZUhhc05vTmFtZSA9IGNvbnZlcnRfdXVkZWNvZGUoIjs5Ry0/MkZAUDtFXTM7QyFXN1U9STsmUT8xJkVFN1YtWThGNVIiKTsKZWNobyAkYVZhcmlhYmxlSGFzTm9OYW1lOwo/Pg==

We get output as :

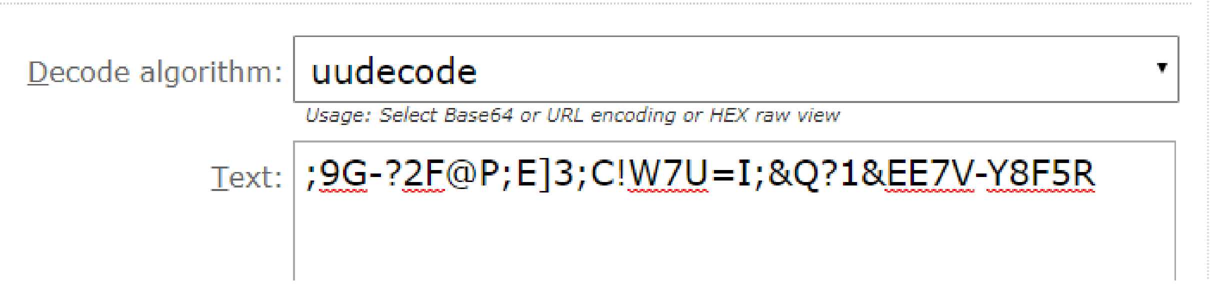
<?php

$aVariableHasNoName = convert_uudecode(";9G-?2F@P;E]3;C!W7U=I;&Q?1&EE7V-Y8F5R");

echo $aVariableHasNoName;

?>

Output is a Php file and a variable present inside it is uuencoded, lets decode it using https://decode.urih.com/ online decoder



Decode algorithm: uudecode

Text: ;9G-?2F@P;E]3;C!W7U=I;&Q?1&EE7V-Y8F5R

Final output : fs_Jh0n_Sn0w_Will_Die_cyber which is flag for this challenge.

8) _LONG_:

Challenge: <DOWNLOAD>

Full Write-up: https://medium.com/@cyjien/long-integer-overflow-ctf-writeup-804c7151f2b3

Answer for long value needed: "-4487045856232229816"

Then, need to print the variable "str" in getHexString function to get the flag.

Code Snippet

```
package main
import (
        "fmt"
        "math/big"
)
func bigInt(n uint64) *big.Int { return new(big.Int).SetUint64(n) }
func main() {
        big1 := bigInt(1)
        b := int64(-37)
        invB := new(big.Int)
        gcd := new(big.Int).GCD(invB, nil, bigInt(bb), N)
        if gcd.Cmp(big1) != 0 {
                panic("GCD != 1")
        }
        x := new(big.Int).Mul(invB, bigInt(a))
        x.And(x, Nmask)
        fmt.Printf("ans = %d signed %d", x, int64(x.Uint64()))
}
```

[Forensics]

1) ChunkyCap:

Challenge: <DOWNLOAD>

To solve the challenge:

Open the Pcap file with Wireshark application.

Apply the following Wireshark filter:
(ip.src == 172.16.250.192) && (ip.dst == 172.16.250.186) && (tcp.flags == 0x018) && (frame.coloring_rule.name == "TCP")

Inspect the following packets and the contents:

Packet 1379: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\6 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\6)
Packet 1384: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\6" >> "85#z"

Packet 1485: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\10 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\10)
Packet 1490: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\10" >> "-p42"

Packet 1587: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\3 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\3)
Packet 1592: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\3" >> "8ej_"

Packet 1745: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\4 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\4)
Packet 1750: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\4" >> "r\a0"

Packet 1850: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\8 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\8)
Packet 1855: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\8" >> "2-05"

Packet 1983: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\1 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\1)
Packet 1988: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\1" >> "1r 0"

Packet 2032: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\5 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\5)
Packet 2038: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\5" >> ".4_p"

Packet 2116: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\2 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\2)
Packet 2121: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\2" >> "#80_"

Packet 2251: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\9 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\9)
Packet 2256: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\9" >> "\@e#"

Packet 2316: The uploaded file name and path (From C:\Users\ry4n\Documents\batch\final\7 >> to >> C:\Users\MALAYSIA\AppData\Local\Temp\7)
Packet 2321: The content of "C:\Users\MALAYSIA\AppData\Local\Temp\7" >> "_4e#"

Packet 2655: A batch file is being executed from the %temp% folder.

Steps to properly get the flag:

1. Simulate the file creations and the script executions.

Create the following file in the %temp% folder and insert the respective value (without the quotes ""):
Filepath: %temp%\6
value:"85#z"

Filepath: %temp%\10
value:"-p42"

Filepath: %temp%\3
value:"r\a0"

Filepath: %temp%\4
value:"85#z"

Filepath: %temp%\8
value:"2-05"

Filepath: %temp%\1
value:"1r 0"

Filepath: %temp%\5
value:".4_p"

Filepath: %temp%\2
value:"#80_"

Filepath: %temp%\9
value:\@e#"

Filepath: %temp%\7
value:"_4e#"

Filepath: %temp%\script.bat:
value:

```
===============================================================
@ECHO OFF
SET abet=abcdefghijklmnopqrstuvwxyz!@#-/\ .0123456789
SET cipher1=8p#j419z\6w.ae@0u2r5o!xk-cf b3g7hmqil/sntdvy
SET list=1 2 3 4 5 6 7 8 9 10
SET flag=
(
FOR %%b IN (%list%) DO (
(
 FOR /f "delims=" %%a IN (%%b) DO (
  SET line=%%a
  CALL :decipher
 )
)
))
ECHO %flag%
pause >nul
GOTO :EOF
:decipher
SET morf=%abet%
SET from=%cipher1%
GOTO trans
:encipher
SET from=%abet%
SET morf=%cipher1%
:trans
SET "enil="
:transl
SET $1=%from%
SET $2=%morf%
:transc
IF /i "%line:~0,1%"=="%$1:~0,1%" SET enil=%enil%%$2:~0,1%&GOTO transnc
SET $1=%$1:~1%
SET $2=%$2:~1%
IF DEFINED $2 GOTO transc
:: No translation - keep
SET enil=%enil%%line:~0,1%
:transnc
SET line=%line:~1%
IF DEFINED line GOTO transl
SET flag=%flag%%enil%
GOTO :eof
================================================================
```

2. Open the command prompt. Change directory to %temp% folder (cd %temp%)

3. Run the script.bat (a simple substitution cypher)

4. The flag will be displayed as: fs@pcap_and_simple_batch_encryptioncyber

2) H NOES! WHATS HAPPENING?:

Challenge: <DOWNLOAD>

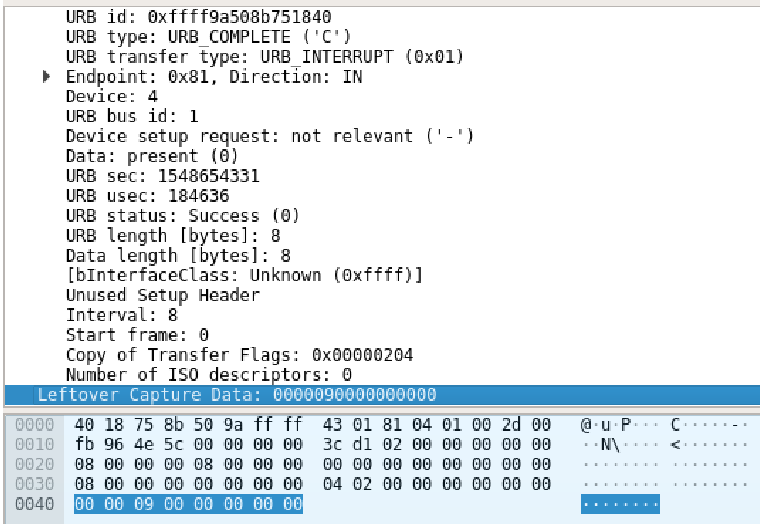
You are provided with an autoruns log file captured from a system and asked to investigate it. The first thing to note is that the autoruns file is just a log file, therefore the flag should most likely be hidden somewhere within one of the entries. With any autoruns file, you'll usually start by analyzing the startup programs and scheduled tasks for any suspicious entries. However, giving it a quick look, nothing looks out of place and you'd be quick to dismiss it, but there's more inside the scheduled task "TcpLogonProvider" than meets the eye. First hint would be questioning the logic behind using cmd.exe to start a program, when in fact the program path can be used directly as the task run entry, and if you drag across the task run command you'll find that it's been padded with lots of whitespaces to hide a second command which echos a base64 string, and by decoding the b64 string you'll get the flag.

Flag: fs0H_W0W_A_SCHT45Kcyber

3) Weird PCAP:

Challenge: <DOWNLOAD>

In this challenge, you are provided with a packet capture file. Looking at the packet protocol it appears to be a keyboard USB packet capture. We are provided with interrupt packets which hold the ‘keystroke’ data in the leftover capture data, e.g. shown below.



```
    URB id: 0xffff9a508b751840
    URB type: URB_COMPLETE ('C')
    URB transfer type: URB_INTERRUPT (0x01)
  ▶ Endpoint: 0x81, Direction: IN
    Device: 4
    URB bus id: 1
    Device setup request: not relevant ('-')
    Data: present (0)
    URB sec: 1548654331
    URB usec: 184636
    URB status: Success (0)
    URB length [bytes]: 8
    Data length [bytes]: 8
    [bInterfaceClass: Unknown (0xffff)]
    Unused Setup Header
    Interval: 8
    Start frame: 0
    Copy of Transfer Flags: 0x00000204
    Number of ISO descriptors: 0
  Leftover Capture Data: 0000090000000000

0000  40 18 75 8b 50 9a ff ff  43 01 81 04 01 00 2d 00   @·u·P···  C······-·
0010  fb 96 4e 5c 00 00 00 00  3c d1 02 00 00 00 00 00   ··N\····  <·······
0020  08 00 00 00 08 00 00 00  00 00 00 00 00 00 00 00   ········  ········
0030  08 00 00 00 00 00 00 00  04 02 00 00 00 00 00 00   ········  ········
0040  00 00 09 00 00 00 00 00                            ········
```

We can extract only the leftover capture data from each packet by using tshark tool

~ tshark -r <FILE_PATH>/weird_pcap.pcapng -T fields -e usb.capdata > captured_data

Looking at the extracted data, the only changing bytes are 1st and 3rd. Looking at the official specification for USB HID keyboards (https://www.usb.org/sites/default/files/documents/hid1_11.pdf, pg.60), we’ll realize that the first byte holds the modifier key, and the third byte holds the 1st keycode (the only keycode in this case). The modifier key byte is a bitfield, where each bit corresponds to a specific modifier key. When a bit is set to 1, the corresponding modifier key is being pressed, and the only modifier key value in our case is 0x02, which corresponds to LEFT_SHIFT. Therefore, if the capture data is 02:00:0c:00:00:00:00:00, 1st byte 0x02 indicates LEFT SHIFT key, and the 3rd byte is 0x0c (based on the Usage Tables provided for Keyboard at https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf), so the user has typed I.

Therefore, we can write a script to decode each packet’s capture data into the corresponding character that was typed (snippet shared below) and we will get the flag.

#Python code snippet

import string

```
scan_codes = {'04':'a', '05':'b', '06':'c', '07':'d', '08':'e', '09':'f', '0a':'g', '0b':'h', '0c':'i', '0d':'j', '0e':'k', '0f':'l', '10':'m', '11':'n', '12':'o', '13':'p', '14':'q', '15':'r', '16':'s', '17':'t', '18':'u', '19':'v', '1a':'w', '1b':'x', '1c':'y', '1d':'z', '1e':'1', '1f':'2', '20':'3', '21':'4', '22':'5', '23':'6', '24':'7', '25':'8', '26':'9', '27':'0','2d':'-'}

with open('captured_data') as file:

    lines = file.readlines()

text=""

for line in lines:

    vals = line.split(":")

    scan_code=vals[2]

    if scan_code=='00':

        continue

    char=scan_codes[scan_code]

    if vals[0] == "02":

        char=string.upper(char)

    text+=char

print(text)
```

Flag: fsl-SEE-WHAT-YOU-TYPED-THEREcyber

4) Image_file_:

Challenge: <DOWNLOAD>

System memory dump has been provided to you. We will be using Volatility to analyze the dump here. Let us first check which browser has been used to download the file

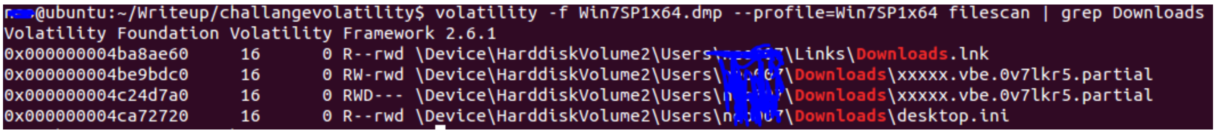
1) To get the process which is downloading the file us the pstree command as follows :

   ```
   volatility -f Win7SP1x64.dmp --profile=Win7SP1x64 pstree
   ```

   After running this command you can see that the Internet Explorer is running in the system

   ```
   .. 0xfffffa8002947060:iexplore.exe                  948   1856     28    964 2019-01-30 02:55:25 UTC+0000
   ```

2) As the default download folder of IE is ‘Downloads’, we are going to do filescan in this location first
   filescan -> volatility -f Win7SP1x64.dmp --profile=Win7SP1x64 filescan | grep Downloads



```
[redacted]@ubuntu:~/Writeup/challangevolatility$ volatility -f Win7SP1x64.dmp --profile=Win7SP1x64 filescan | grep Downloads
Volatility Foundation Volatility Framework 2.6.1
0x000000004ba8ae60      16      0 R--rwd \Device\HarddiskVolume2\Users\[redacted]\Links\Downloads.lnk
0x000000004be9bdc0      16      0 RW-rwd \Device\HarddiskVolume2\Users\[redacted]\Downloads\xxxxx.vbe.0v7lkr5.partial
0x000000004c24d7a0      16      0 RWD--- \Device\HarddiskVolume2\Users\[redacted]\Downloads\xxxxx.vbe.0v7lkr5.partial
0x000000004ca72720      16      0 R--rwd \Device\HarddiskVolume2\Users\[redacted]\Downloads\desktop.ini
```

Shows path of xxxxx.vbe along with offset 0x4c24d7a0

3) File xxxx.vbe looks suspicious, let's check the content of this file using dumpfiles command volatility -f Win7SP1x64.dmp --profile=Win7SP1x64 dumpfiles -Q 0x4c24d7a0 --name -D <path to dump file>

   Content : #@~^OAAAAA==   Um.bwDR21tKcJor]Ps+~AbYt,T6lP),^-cbLk=dLr[$VrV$k0-o|Ar#aBMAAA==^#~@

4) Dumped file is a vbe file so use any decoder(https://gallery.technet.microsoft.com/Encode-and-Decode-a-VB-a480d74c) to decode the vbe data.
   decoded file data is : WScript.Echo("XOR me with 0x5 : cv4ijs`sjidqlilq|f|g`w")

   XOR with 0x5 gives result - fs1lovevolatilitycyber, which is the flag

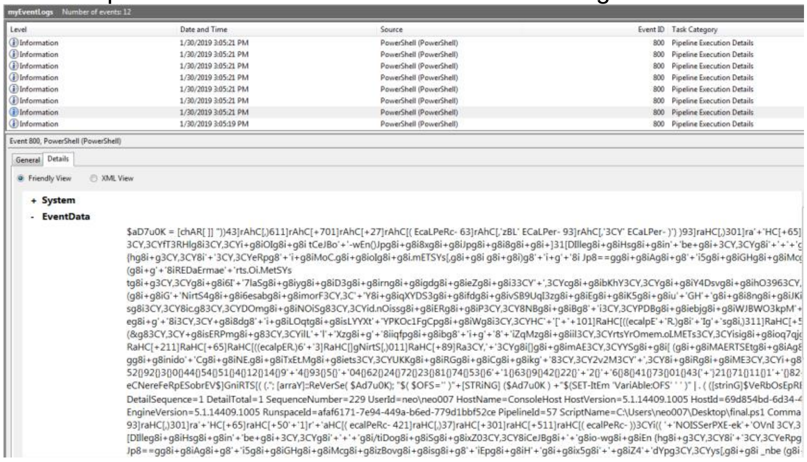
Read more volatility command here : https://github.com/volatilityfoundation/volatility/wiki/Command-Reference

5) **_P0wer_3vents_:**

   Challenge: <DOWNLOAD>

   In this case Windows Event Log has been provided and we have to find flag in it.

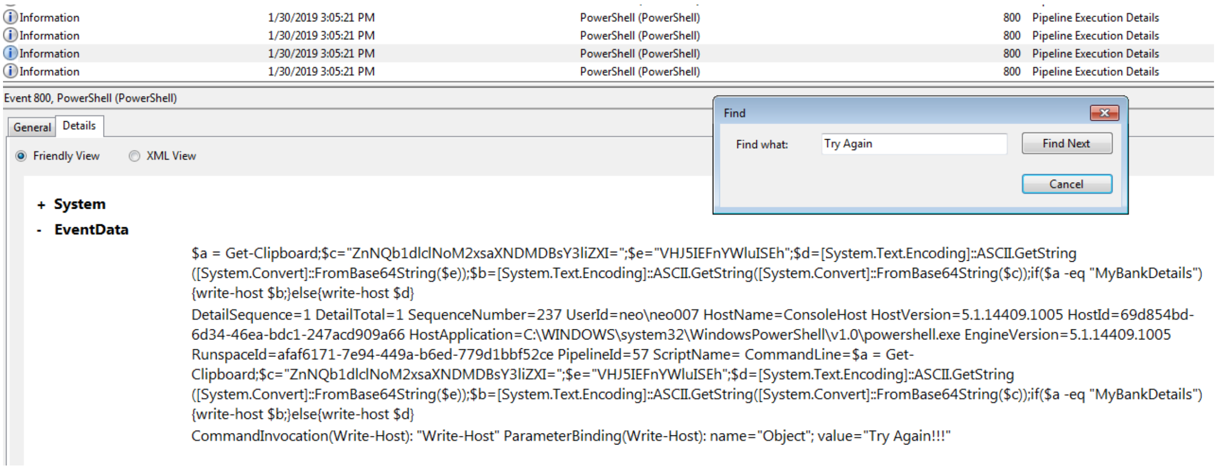
The first powershell event that executes is following :



```
myEventLogs   Number of events: 12

Level          Date and Time          Source                   Event ID  Task Category
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:21 PM   PowerShell (PowerShell)  800       Pipeline Execution Details
Information    1/30/2019 3:05:19 PM   PowerShell (PowerShell)  800       Pipeline Execution Details

Event 800, PowerShell (PowerShell)
General  Details
● Friendly View   ○ XML View

+ System
- EventData
        $aD7u0K = [chAR[ ]] "))43]rAhC[,)611]rAhC[+701]rAhC[+27]rAhC[( EcaLPeRc- 63]rAhC[,'zBL' ECaLPer- 93]rAhC[,'3CY' ECaLPer- )') )93]raHC[,)301]ra'+'HC[+65]
        3CY,3CYfT3RHlg8i3CY,3CYi+g8iOIg8i+g8i tCeJBo'+'-wEn()Jpg8i+g8i8xg8i+g8iJpg8i+g8i8g8i+g8i+]31[DIlleg8i+g8iHsg8i+g8in'+'be+g8i+3CY,3CYg8i'+'+'+'g
        (hg8i+g3CY,3CY8i'+'3CY,3CYeRpg8'+'i+g8iMoC.g8i+g8iolg8i+g8i.mETSYs[,g8i+g8i g8i+g8i)g8'+'i+g'+'8i Jp8==gg8i+g8iAg8i+g8'+'i5g8i+g8iGHg8i+g8iMcg
        (g8i+g'+'8iREDaErmae'+'rts.Oi.MetSYs
        tg8i+g3CY,3CYg8i+g8i6I'+'7laSg8i+g8iyg8i+g8iD3g8i+g8irng8i+g8igdg8i+g8ieZg8i+g8i33CY'+',3CYcg8i+g8ibKhY3CY,3CYg8i+g8iY4Dsvg8i+g8ihO3963CY,
        (g8i+g8iG'+'NirtS4g8i+g8i6esabg8i+g8imorF3CY,3C'+'Y8i+g8iqXYDS3g8i+g8ifdg8i+g8ivSB9UqI3zg8i+g8iEg8i+g8iK5g8i+g8iu'+'GH'+'g8i+g8i8ng8i+g8iJKi
        sg8i3CY,3CY8ic.g83CY,3CYDOmg8i+g8iNOiSg83CY,3CYid.nOissg8i+g8iERg8i+g8iP3CY,3CY8NBg8i+g8iBg8'+'i3CY,3CYPDBg8i+g8iebjg8i+g8iWJBWO3kpM'+
        eg8i+g'+'8i3CY,3CY+g8i8dg8'+'i+g8iLOqtg8i+g8isLYYXt'+'YPKOc1FgCpg8i+g8iWg8i3CY,3CYHC'+'['+'+101]RaHC[((ecalpE'+'R.)g8i'+'Ig'+'sg8i,)311]RaHC[+5
        (&g83CY,3CY+g8isERPmg8i+g83CY,3CYilL'+'l'+'Xzg8i+g'+'8iiqfpg8i+g8ibg8'+'i+g'+'8'+'iZqMzg8i+g8iil3CY,3CYrtsYrOmem.oLMETs3CY,3CYisig8i+g8ioq7qjq
        RaHC[+211]RaHC[+65]RaHC[((ecalpER.)6'+'3]RaHC[]gNirtS[,)011]RaHC[+89]Ra3CY,'+'3CYg8i[]g8i+g8imAE3CY,3CYYSg8i+g8i[ (g8i+g8iMAERTSEtg8i+g8iAg8
        gg8i+g8inido'+'Cg8i+g8iNE.g8i+g8iTxEt.Mg8i+g8iets3CY,3CYUKKg8i+g8iRGg8i+g8iCg8i+g8ikg'+'83CY,3CY2v2M3CY'+',3CY8i+g8iRg8i+g8iME3CY,3CYi+g8
        52{}92{}3{}0{}44{}54{}51{}4{}12{}14{}9'+'4{}93{}5{}'+'04{}62{}24{}72{}23{}81{}74{}53{}6'+'1{}63{}9{}42{}22{}'+'2{}'+'6{}8{}41{}73{}01{}43{'+'}21{}71{}11{}1'+'{}82
        eCNereFeRpESobrEV$]GniRTS[( (."; [arraY]::ReVerSe( $Ad7u0K); "$( $OFS='' )"+[STRiNG] ($Ad7u0K ) +"$(SET-ItEm 'VariAble:OFS' ' ' )" | . ( ([strinG]$VeRbOsEpRE
        DetailSequence=1 DetailTotal=1 SequenceNumber=229 UserId=neo\neo007 HostName=ConsoleHost HostVersion=5.1.14409.1005 HostId=69d854bd-6d34-4
        EngineVersion=5.1.14409.1005 RunspaceId=afaf6171-7e94-449a-b6ed-779d1bbf52ce PipelineId=57 ScriptName=C:\Users\neo007\Desktop\final.ps1 Comma
        93]raHC[,)301]ra'+'HC[+65]raHC[+50'+'1]r'+'aHC[( ecalPeRc- 421]raHC[,)37]raHC[+301]raHC[+511]raHC[( ecalPeRc- ))3CYi(( '+'NOISSerPXE-ek'+'OVnI 3CY,3
        [DIlleg8i+g8iHsg8i+g8in'+'be+g8i+3CY,3CYg8i'+'+'+'g8i/tiDog8i+g8iSg8i+g8ixZ03CY,3CY8iCeJBg8i+'+'g8io-wg8i+g8iEn (hg8i+g3CY,3CY8i'+'3CY,3CYeRpg
        Jp8==gg8i+g8iAg8i+g8'+'i5g8i+g8iGHg8i+g8iMcg8i+g8izBovg8i+g8isg8i+g8'+'iEpg8i+g8iH'+'g8i+g8ix5g8i'+'+g8iZ4'+'dYpg3CY,3CYys[,g8i+g8i _nbe (g8i
```

It is highly obfuscated and quite difficult to deobfuscate. Good thing about Powershell event logging is that it logs all the level of execution of powershell script. This means that we might be able to get the deobfuscated powershell script in logs.

To identify the script we have to take hint from question "I am getting "Try Again!!!". This means that the final script will have the script "Try Again!!!" . Let's find this string in Log



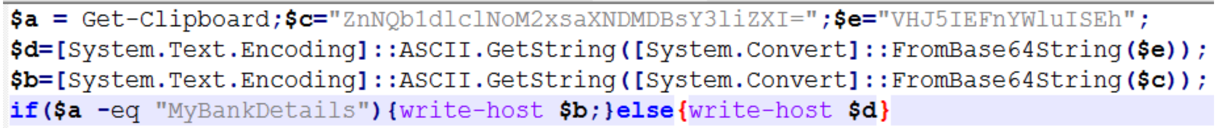
Information	1/30/2019 3:05:21 PM	PowerShell (PowerShell)	800	Pipeline Execution Details
Information	1/30/2019 3:05:21 PM	PowerShell (PowerShell)	800	Pipeline Execution Details
Information	1/30/2019 3:05:21 PM	PowerShell (PowerShell)	800	Pipeline Execution Details
Information	1/30/2019 3:05:21 PM	PowerShell (PowerShell)	800	Pipeline Execution Details

Event 800, PowerShell (PowerShell)

Find what: Try Again

```
$a = Get-Clipboard;$c="ZnNQb1dlclNoM2xsaXNDMDBsY3liZXI=";$e="VHJ5IEFnYWluISEh";$d=[System.Text.Encoding]::ASCII.GetString
([System.Convert]::FromBase64String($e));$b=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($c));if($a -eq "MyBankDetails")
{write-host $b;}else{write-host $d}
DetailSequence=1 DetailTotal=1 SequenceNumber=237 UserId=neo\neo007 HostName=ConsoleHost HostVersion=5.1.14409.1005 HostId=69d854bd-
6d34-46ea-bdc1-247acd909a66 HostApplication=C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe EngineVersion=5.1.14409.1005
RunspaceId=afaf6171-7e94-449a-b6ed-779d1bbf52ce PipelineId=57 ScriptName= CommandLine=$a = Get-
Clipboard;$c="ZnNQb1dlclNoM2xsaXNDMDBsY3liZXI=";$e="VHJ5IEFnYWluISEh";$d=[System.Text.Encoding]::ASCII.GetString
([System.Convert]::FromBase64String($e));$b=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($c));if($a -eq "MyBankDetails")
{write-host $b;}else{write-host $d}
CommandInvocation(Write-Host): "Write-Host" ParameterBinding(Write-Host): name="Object"; value="Try Again!!!"
```

Lets copy the script and analyze



```
$a = Get-Clipboard;$c="ZnNQb1dlclNoM2xsaXNDMDBsY3liZXI=";$e="VHJ5IEFnYWluISEh";
$d=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($e));
$b=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($c));
if($a -eq "MyBankDetails"){write-host $b;}else{write-host $d}
```

It’s clearly visible that if clipboard contains “MyBankDetails” variable $c is base64 decoded otherwise variable $e gets decoded. Let’s base64 decode both these variable to see result

$e="VHJ5IEFnYWluISEh"; → Try Again!!!

$c="ZnNQb1dlclNoM2xsaXNDMDBsY3liZXI=" → fsPoWerSh3llisC00lcyber

fsPoWerSh3llisC00lcyber is our final flag

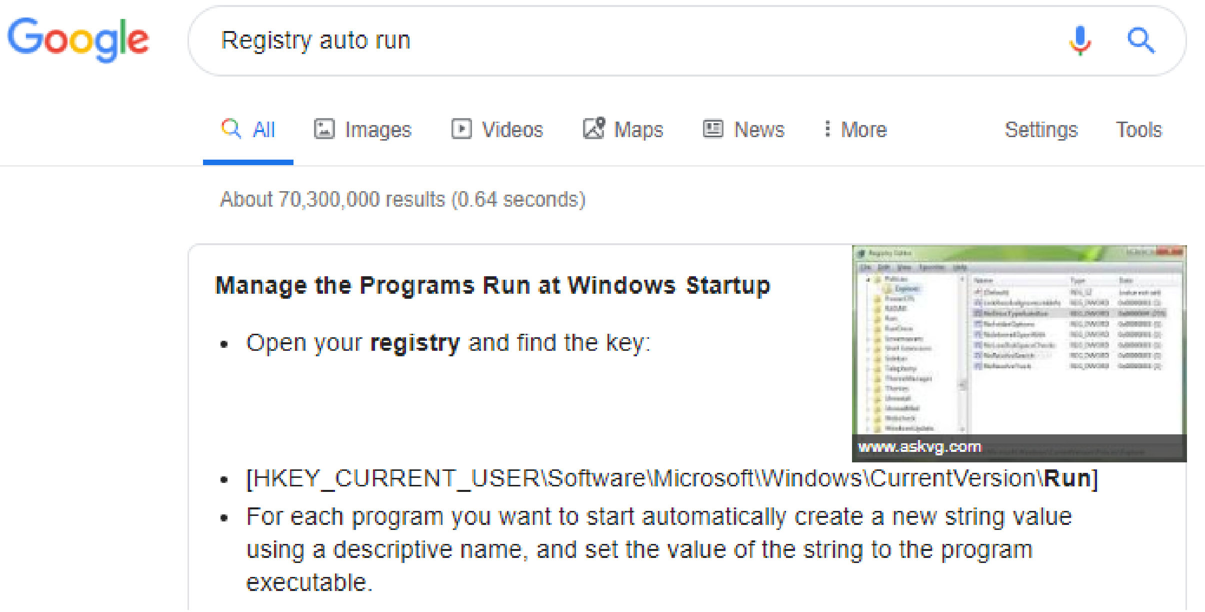
Learn more about
Powershell logging :https://www.secjuice.com/enterprise-powershell-protection-logging/

Powershell Obfuscation : https://www.endgame.com/blog/technical-blog/deobfuscating-powershell-putting-toothpaste-back-tube

6) Initial_rUn_:

Challenge: <DOWNLOAD>

The participants are required to identify registry which can be used to launch an application on startup.

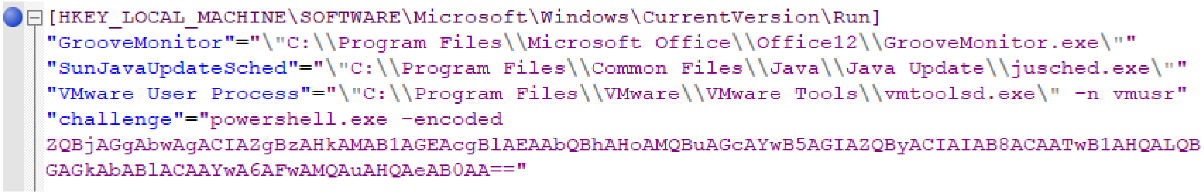


About 70,300,000 results (0.64 seconds)

Manage the Programs Run at Windows Startup

- Open your **registry** and find the key:
- [HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]
- For each program you want to start automatically create a new string value using a descriptive name, and set the value of the string to the program executable.

Let’s check the target registry from the registry file and investigate its key values.



```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"GrooveMonitor"="\"C:\\Program Files\\Microsoft Office\\Office12\\GrooveMonitor.exe\""
"SunJavaUpdateSched"="\"C:\\Program Files\\Common Files\\Java\\Java Update\\jusched.exe\""
"VMware User Process"="\"C:\\Program Files\\VMware\\VMware Tools\\vmtoolsd.exe\" -n vmusr"
"challenge"="powershell.exe -encoded
ZQBjAGgAbwAgACIAZgBzAHkAMAB1AGEAcgBlAEAAbQBhAHoAMQBuAGcAYwB5AGIAZQByACIAIAB8ACAATwB1AHQALQB
GAGkAbABlACAAYwA6AFwAMQAuAHQAeAB0AA=="
```

We are able to see application being executed on startup (GrooveMonitor, SunJavaUpdateSched, VMware User Process and challenge).

We can also see a suspicious base64 encoded string being invoked by a PowerShell and when decoded it turns out to be the flag.

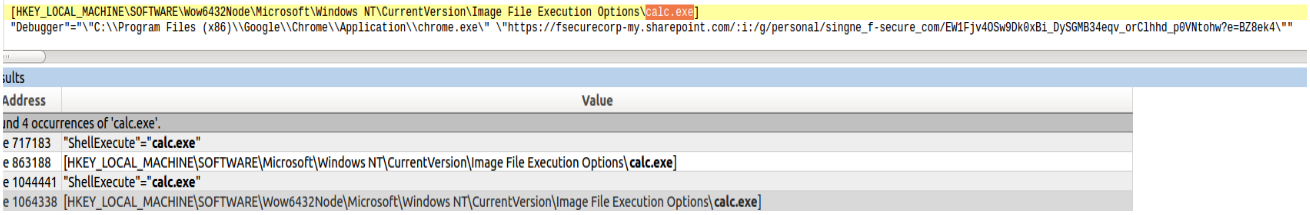
echo "fsy0uare@maz1ngcyber" | Out-File c:\1txt

Flag: fsy0uare@maz1ngcyber

7) IFEO:

Challenge: <DOWNLOAD>

We have been given a registry file to analyze. Question clearly states that the issue is with calculator. So let's search the calc.exe in registry by loading registry in 010 editor. There are 4 occurrence of calc.exe in registry and checking them all there is 1 entry which looks suspicious and fits the description provided in question

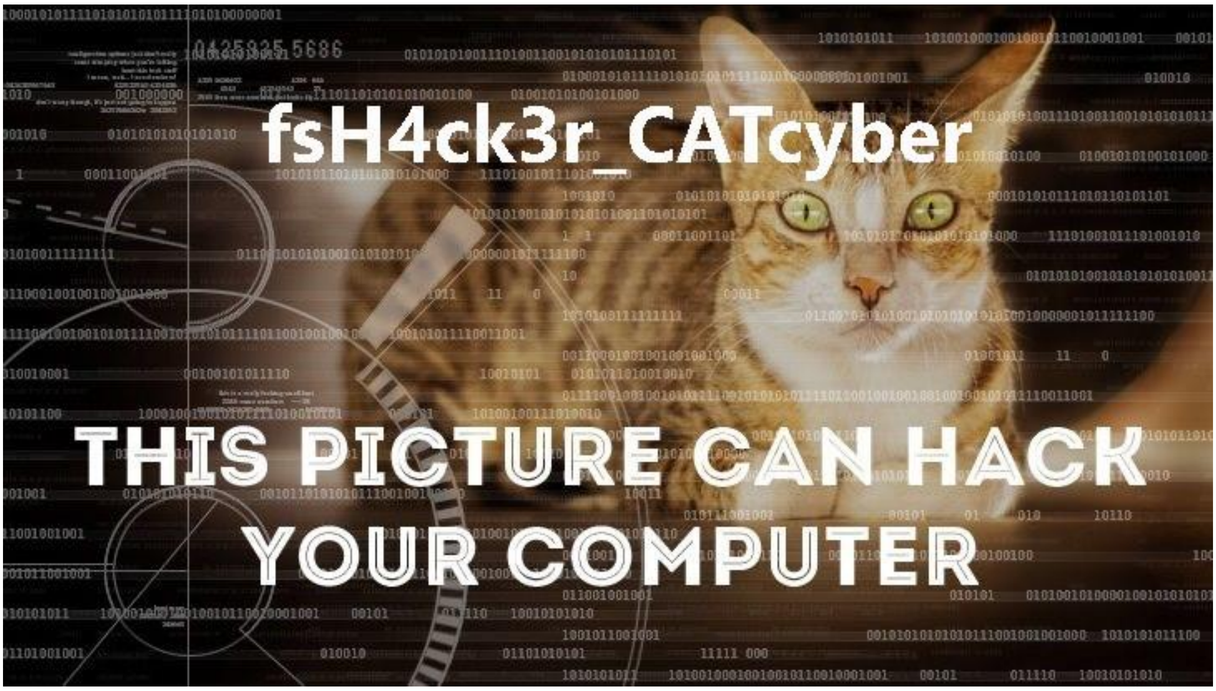


```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe]
"Debugger"="\"C:\\Program Files (x86)\\Google\\Chrome\\Application\\chrome.exe\" \"https://fsecurecorp-my.sharepoint.com/:i:/g/personal/singne_f-secure_com/EW1Fjv4OSw9Dk0xBi_DySGMB34eqv_orClhhd_p0VNtohw?e=BZ8ek4\""
```

```
sults
Address    Value
und 4 occurrences of 'calc.exe'.
e 717183   "ShellExecute"="calc.exe"
e 863188   [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe]
e 1044441  "ShellExecute"="calc.exe"
e 1064338  [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe]
```

In the above registry Image File execution Option's debugger of calc.exe is assigned to chrome.exe with a URL. In short, when calculator will be opened then chrome will start instead with the URL shown

Let's visit the URL and see what's there



That's it. Visiting the URL shows us a picture with flag : fsH4ck3r_CATcyber

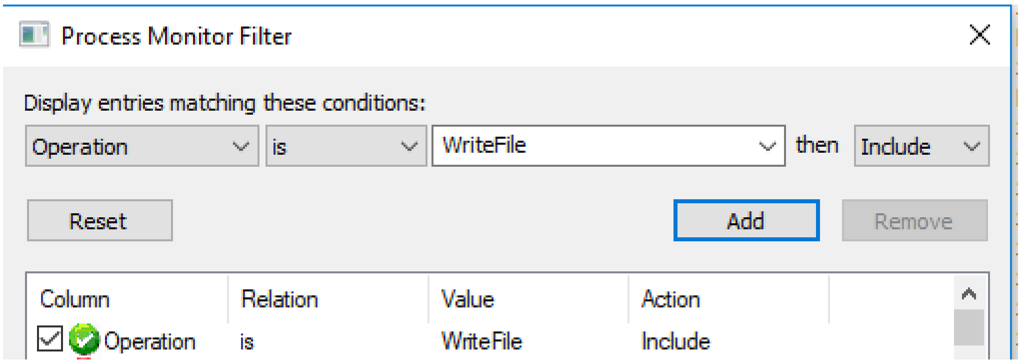
Read more about IFEO (Image file execution option) : https://attack.mitre.org/techniques/T1183/

8) Pr0cl0g:

Challenge: <DOWNLOAD>

Open the PML log in Procmon. Since we know that files are being written onto the disk, lets apply the writefile filter in procmon

Press Ctrl+L to open filter and proceed to apply filter as follows :



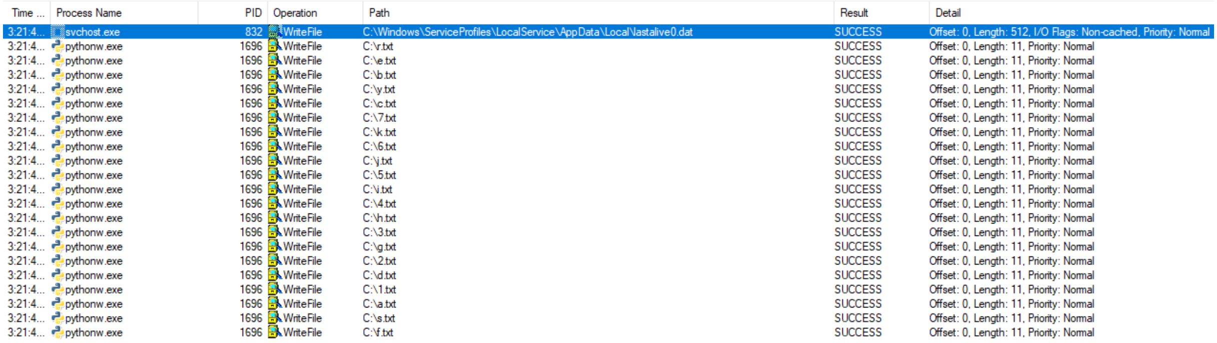
Process Monitor Filter

Display entries matching these conditions:

Operation | is | WriteFile | then | Include

Column	Relation	Value	Action
☑ Operation	is	WriteFile	Include

After applying the filter log looks like this :



Time ...	Process Name	PID	Operation	Path	Result	Detail
3:21:4...	svchost.exe	832	WriteFile	C:\Windows\ServiceProfiles\LocalService\AppData\Local\lastalive0.dat	SUCCESS	Offset: 0, Length: 512, I/O Flags: Non-cached, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\r.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\e.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\b.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\y.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\c.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\7.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\k.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\6.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\j.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\5.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\i.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\4.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\h.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\3.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\g.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\2.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\d.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\1.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\a.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\s.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal
3:21:4...	pythonw.exe	1696	WriteFile	C:\f.txt	SUCCESS	Offset: 0, Length: 11, Priority: Normal

Now, carefully observe the name of files written, it is flag format but in reverse (Ignore the file written by svchost.exe).
So, if we reverse the filenames being dropped we will get the flag as :

fsa1d2g3h4i5j6k7cyber

Read more about Procmon here : https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

[Miscellaneous]

1) **Unkn0wn_p1c:**

   Challenge: <DOWNLOAD>

   Using online exif tool to find the 'LensModel' then decrypt using base64

2) **WHATLIESWITHINME?:**

   Challenge: <DOWNLOAD>

   In this challenge, you are provided a JPEG image and asked to find the flag. There are a few ways to solve this.

Approach #1

Run the UNIX command strings and scroll through the printable strings to find ZnMwSF9IM0xMMF9USDNSM2N5YmVy at the very end, which looks like base64, decoding this base64 will yield the flag.

Approach #2

Parse the JPEG image to find out that there's an overlay (extra data) appended at the end of the image, which is ZnMwSF9IM0xMMF9USDNSM2N5YmVy, decoding this base64 string will yield the flag.

Flag: fs0H_H3LL0_TH3R3cyber

3) **ZIPception:**

   Challenge: <DOWNLOAD>

   In this challenge, you are provided a ZIP file which is password-protected. You could either try and crack the password, or through the power of trial-and-error realize that the archive name is the password of each archive. After extracting the first one, you'll realize that the child archive is also password-protected in the same fashion and you'll be left with a dilemma, to manually extract (without knowing how many more archive files lie ahead) or to write a script to automate things. The flag can be found within flag.txt inside the last nested archive. Here's a code snippet to automate things in python:

```
import zipfile
# dependencies: zipfile
# iteratively decompress nested password-protected archives with the password being the archive filename
# the initial archive should be within the same dir
# e.g. decompress("52525.zip")
def decompress(first_filename):
    file=first_filename
    child_file = str(file)
    while file is not None:
        passwd = file.split('.')[0]
        with zipfile.ZipFile(file,'r') as myzip:
            myzip.setpassword(passwd)
            file= myzip.namelist()[0]
            myzip.extract(file)
            if 'zip' not in file:
                file=None
        print file
```

4) Docommand:

Challenge: <DOWNLOAD>

The provided file is a macro-enabled document file which contains macro that executes a base64 encoded command which will gives out the hint about the actual location of the encoded flag and the encoder used.

The document file contains multiple instances of Shapes objects which each contains different values in their AlternativeText section (with one of the Shape contains the encoded command that provides the hints).

These are all the list of the Shapes objects and their AlternativeText values:

```
ActiveDocument.Shapes("1XLK96Fck").AlternativeText = "kcyM_GEL_ntnva"
ActiveDocument.Shapes("kPyb3ugvY").AlternativeText = "Gvyy_aRkG_Gvzr"
ActiveDocument.Shapes("kQHKzfk2Q").AlternativeText = "nyZbfg_guRer"
ActiveDocument.Shapes("EbYVWWyhK").AlternativeText = "orGgre_YhPx_gbzbeebj"
ActiveDocument.Shapes("WvWgcmcrP").AlternativeText = "PH_ntnva_AKGlrne"
ActiveDocument.Shapes("jqpLgPzjb").AlternativeText = "gel_UneQre"
ActiveDocument.Shapes("KfdNwDVvj").AlternativeText = "enaQbz_fghss"
ActiveDocument.Shapes("ErkCPiuDy").AlternativeText = "sfV_NZ_URERplore"
ActiveDocument.Shapes("ViXsr3ecx").AlternativeText = "uryyB_jbeyQQ"
ActiveDocument.Shapes("yYFjZAYYb").AlternativeText = "vnzonpx_gb_Hav"
ActiveDocument.Shapes("96Fck1XLK").AlternativeText = "uryyb_sebz_gurBgureFVqr"
ActiveDocument.Shapes("WWy2KEbYV").AlternativeText = "hcfvqr_qbja"
ActiveDocument.Shapes("zfk2QkQHK").AlternativeText = "pnag_lbh_frr"
ActiveDocument.Shapes("WWyhKEbYV").AlternativeText = "v_yvxr_gurjnl_h_gel"
ActiveDocument.Shapes("cmcrPWvWg").AlternativeText = "cyrnfr_gel_zber"
ActiveDocument.Shapes("gPzjbjqpL").AlternativeText = "tbbq_yhpx"
ActiveDocument.Shapes("wDVvjKfdN").AlternativeText = "unir_sha"
ActiveDocument.Shapes("PiuDyErkC").AlternativeText = "v_nz_univatSha"
ActiveDocument.Shapes("r3ecxViXs").AlternativeText = "vpna_frr_jung_hgelvat_gbqb"
ActiveDocument.Shapes("AYYbyYFjZ").AlternativeText = "cMD /c ""set e11=wersh&& set 9a1=ell.e&&
set 4q1=xe&& set g01=po&& set cm=.&& set tq=ight&& set 9s=ally&& set ls=cl&& set 8w=os&& set
tr=st&& set uy=ot&& set u8=r&& set oh=h&& set sp=_&& set ee=E&& set gt=th&& set ov=13&&""
c^md^.e^xe /c %g01%%e11%%9a1%%4q1% -ec
VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAFkAbwB1ACAAYQByAGUAIABuAGUAYQByAC4
AIABMAG8AYwBhAHQAZQAgAHQAaABpAHMAIABjAG8AbQBtAG%ee%AbgBkACAAaQBuACAAY
QAgAGgAZQB4ACAAdgBpAGUAdwBlAHIAIABhAG4AZAAgAHkAbwB1ACAAdwBpAGwAbAAgAGYA
aQBuAGQAIABtAGUALgAgA%ee%0AeQAgAFMASABBAFAARQAgAG4AYQBtAGUAIABpAHMAIAB
FAHIAawBDAFAAaQB1AEQAeQAgAGEAbgBkACAASQAgAGEAbQAgAGUAbgBjAG8AZABlAGQAI
AB3AGkAdABoAC4ALgAuACIA && echo %u8%%uy%%ov%"
```

Actual decoded command in Shapes("AYYbyYFjZ")= Write-Output "You are near. Locate this command in a hex viewer and you will find me. My SHAPE name is ErkCPiuDy and I am encoded with...rot13"

To solve the challenge:

1. Open Procmon (Sysinternal) application and filter the process based on Process Name: "cmd.exe"
2. Execute the file with Microsoft Word (Make sure macro is enabled).
3. Examine the Procmon for any cmd.exe process created.
4. Right click on one of the created cmd.exe process > select properties > Process tab > Command Line.

5. Notice that the command line contains the execution command of an encoded strings.
6. Copy out the command and execute it in a separate cmd.exe.
7. Notice the output contains the following hints: "You are near. Locate this command in a hex viewer and you will find me. My SHAPE name is ErkCPiuDy and I am encoded with...rot13"
8. Using your prefered hex editor, search for the string "ErkCPiuDy" in the document file.
9. You will find the AlternativeText section near the "ErkCPiuDy" strings which contains another string "sfV_NZ_URERplore".
10. Decode the string using ROT13 decoder and you will get flag: "fsI_AM_HEREcyber".

5) **D0cal3:**

Challenge: <DOWNLOAD>

The provided file is a macro-enabled excel file which contains macro which in general will only be properly executed if the system locale is set to Japanase. Specifically, it checks for the currency format of the system and will decrypt the payload (a simple cmd script) based on the parsed value. The intended script will only be properly decrypted when the system locale is set to Japanese.

To solve the challenge:

1. Change your system locale and format to use Japanase (Japan) in the Region settings.
2. Open Procmon (Sysinternal) application and filter the process based on Process Name: "cmd.exe"
3. Execute the file with Microsoft Excel (Make sure Macro is enabled)
4. Examine the Procmon for any cmd.exe process created.
5. Right click on one of the created cmd.exe process > select properties > Process tab > Command Line.
6. Notice that the content of the command line is: cmd /c "echo hello_world & echo hi_there ....& echo fsDomo_Arigatocyber"
7. Flag: fsDomo_Arigatocyber

[Cryptography]

1) **Listen to me!:**

Challenge: <DOWNLOAD>

The given audio are encoded with morse code.
Listen to the audio will reveal the morse code.
Decipher the morse code will get the encoded base32 ciphertext.
Decode the base32 ciphertext will get the flag.

2) Gaius the Great:

Challenge: <DOWNLOAD>

Decrypt base 32 string yield base 64 string
Decrypt base 64 string yields a new string
Reverse the text
The string is a ceaser cipher with a rotation of 13

3) Cooper's Message:
Challenge: Encoded: 000 0010 1100 0000 0 1010 001 0110 0100 111 010 0

ROT13-> Morse -> replace . with 0 -> replace - with 1

4) ITS NOT WHAT IT LOOKS LIKE!:

Challenge: .. - . - . . -. — — . .   —-- .— — -.   - ... . -. .— -... -. .. .. —
.... —-— . -.. . -.-.--

It looks like morse code, but it's actually using unicode homoglyphs to hide the message using the following tool: http://holloway.co.nz/steg/

5) I WANT TO KNOW:

Challenge: ngkjtphwbyvbddfgal

Vigenere cipher (key: friend) > Playfair cipher (key:friendabcghklmopqstuvwxyz)

6) Secret Message:

Challenge: (.-------........-.---.....--...---.-...-.----.---...-.-..-..-..---.-.-----.-.-.---.--.-..--.-.---...-.----.--......-.-.--
-..-)

morse code (. as 1, - as 0) > binary to decimal > polybius cipher (A to Z) letter j with i
decimal - 21433334443234424315133414151354121542

flag: fsnotmorsecodecyber

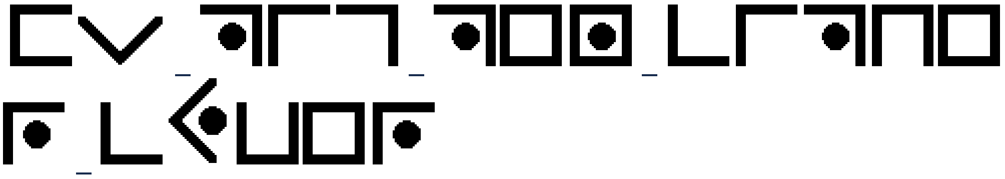
7) A ^ B = C | C ^ B = A | B = C ^ A:

Challenge: h}Qko}wQva|Qi{k}}Qmwlk|

XOR key = guess
Can use xor bruteforce with known word “cyber”

8) Penpher!:

Challenge:



Kindly use Pigpen Cipher to encode the flag