F-Secure Cybersecurity Competition Malaysia 2019

Qualifying Round Questions & Answers

QUESTION 1

Deobfuscate the file and identify the flag from the PowerShell script.

File/Folder Name: Q1

```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
-----------------------------------------------------------------------------
```

ANSWER Q1

1) Running the script gives output : Pay me 10 Bitcoin for answer!!!
2) Basically for PowerShell we have to search for Invoke-Expression (IEX).

   At the end of the script, instruction are piped using |.

   Checking code after | shows `( $sHElLId[1]+$SHeLLid[13]+'X')`

   Running `( $sHElLId[1]+$SHeLLid[13]+'X')` in PowerShell gives output 'iex'.

3) Let's look at the code before the | instruction:

   ```
   . ( $eNv:COmSpEC[4,26,25]-jOiN'')("$(Set  'OfS'  '') " + [STRING] ($AXoj7p[ -
   1 ..-( $AXoj7p.LENGth ) ])+" $( seT-vaRiaBLE 'OFs'  ' ')" )
   ```

   `. ( $eNv:COmSpEC[4,26,25]-jOiN'')` is again 'iex'

   `[STRING] ($AXoj7p[ -1 ..-( $AXoj7p.LENGth ) ])` – this means that whatever will be the value in variable ($AXoj7p will be reversed and passed to IEX for execution

4) The first part of the script shows the value of the variable $AXOJ7p:

   ```
   SEt-VArIAble  AXOJ7p  (" ))63]rahc[,'zcR' ecAlpErc-
   43]rahc[,)56]rahc[+27]rahc[+711]rahc[(ecAlpErc- )'})AHu!!!rewsna rof
   '+'n'+'iocti'+'B 01 '+'em yaP'+'AHu('+'t'+'soh-
   etirW{esle})AHurebyc'+'m0s'+'n'+'aRer0M0Nsf'+'AHu(tsoh-et'+'irW{)0'+'1 e'+'l-
   xzcR( '+'fi;

   '+'02=xzcR'((()'x'+]31[DIlLEHS$+]1[dILlEhs$ (. "  ) ;
   ```

5) By step 3, we know that this value needs to be reversed, so let's reverse the value using cyberchef (https://gchq.github.io/CyberChef/)

   Reversed value :

```
)  " .( $shElLId[1]+$SHELlID[13]+'x')(('Rczx=20'+'
;if'+' (Rczx -l'+'e 1'+'0){Wri'+'te-
host(uHA'+'fsN0M0reRa'+'n'+'s0m'+'cyberuHA)}else{Write-hos'+'t'+'(uHA'+'Pay
me'+' 10 B'+'itcoi'+'n'+' for answer!!!uHA)}') -
crEplAce([char]117+[char]72+[char]65),[char]34  -crEplAce 'Rcz',[char]36)) "(
```

6) Now we can see that there are 2 replace function inside this value:

   `[char]117+[char]72+[char]65),[char]34` → uHA replaced with "

   `'Rcz',[char]36` → Rcz replaced with `$`

7) We can already see our required output after reversing the variable value, let's apply the replacement, and remove the concatenation string '+' as well.

   `Wri'+'te-host(uHA'+'fsN0M0reRa'+'n'+'s0m'+'cyberuHA)}` becomes `Write-host("fsN0M0reRans0mcyber")}` which outputs our answer

QUESTION 2

Investigate the Spreadsheet and identify the flag.
HINT: Flag is in a cell hidden within the sheet.

File/Folder Name: Q2

```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
-----------------------------------------------------------------------------
```

ANSWER Q2

The Excel file contains the following type of sheets:-

- Normal: (Sheet1 - Sheet3)
- Hidden: (Sheet4 - Sheet6)
- Very Hidden: (Sheet7 - Sheet20)

One of the Very Hidden sheets contains a visible string ("I'm here:") at B7, which also contains the flag, written in white colour with a long whitespaces in between the string "I'm here:" and the flag.

One of way to solve the challenge:

- Open the file using Microsoft Excel, and then press Alt+F11, which will open the Visual Basic Editor.
- In the Project Explorer Tab, select Sheet13 and change the Visible property to: -1 - xlSheetVisible
- Go back to the main Excel view and Sheet13 will now be visible.
- Open Sheet13 and find cell B7.
- Select all of the characters in B7 cell and change the font color to black to make the flag visible.

QUESTION 3

Identify the file and then extract the flag.
HINT: flag is in radix 55?, 65?, 75?, .? or 95?

File/Folder Name: Q3

```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
-----------------------------------------------------------------------------
```

ANSWER Q3

This challenge contains an unknown file. When the user tries to identify the file, they will find it is a .MSG file that been exported.

The .MSG can be opened using any email client, such as Thunderbird or Outlook.

When opening the message, the following message is displayed:

"*Hello Fellows!*

Here is the flag as we discussed.

AohEiF(I<gASu!rA7]7r@V'Q

I hope you remember what base it is :p

Regards,

Fellow"

From the message we can see that there is a hint, “what base it is”. After multiple base’s attempts, such as base64 35 etc.., you will find that base85 is the only base that outputs a readable ASCII string with the flag.

QUESTION 4

Identify the flag from the image. Key can be found in the file to unlock the flag.

File/Folder Name: Q4

```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
```

```
-----------------------------------------------------------------------------
```

ANSWER Q4

The image has a hint under the image comments and description of the company.

This challenge contains a “flag.txt” embedded in the fs.jpeg, but a password is needed to extract it. The password is 'fsecure', based on the comment section from the image. Software to use to extract can be steghide or any other stenography tools.

QUESTION 5

Debug and identify the the flag from the shellcode.
HINT: Just continue debugging

File/Folder Name: Q5

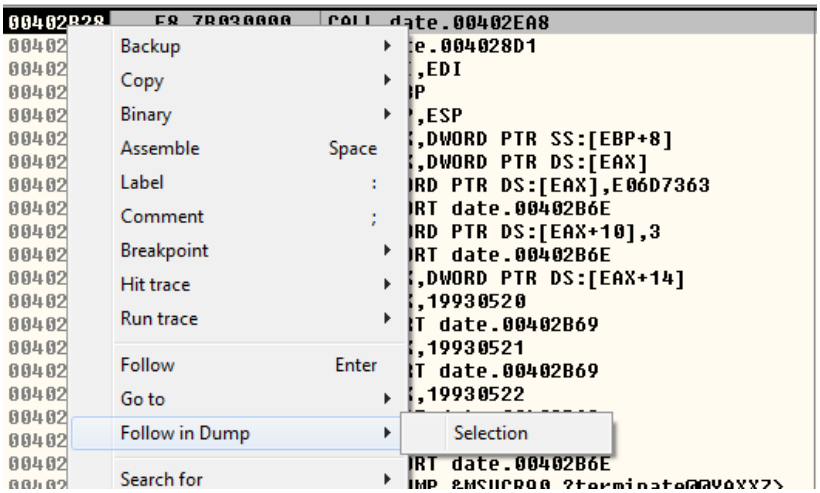
```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
-----------------------------------------------------------------------------
```

ANSWER Q5

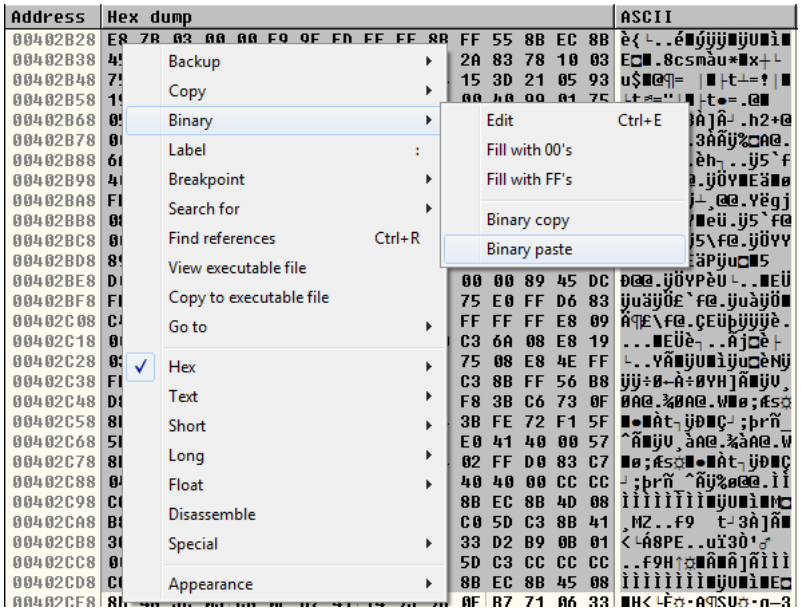
You can use any available shellcode debugger to debug this shellcode. Here, we describe an easier way to load the shellcode in any program.

Open Ollydbg -> Goto File-> Open -> Open any exe file you like and stop at entry point.

Right click on Entry point -> Follow in Dump -> Selection.



Now copy the hex bytes of Pickahu.bin and paste in dump.



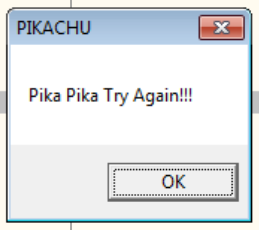
Now entry point of the file will be replaced by the shellcode and you can start debugging.

```
00402B28    31C9            XOR ECX,ECX
00402B2A    64:8B41 30      MOV EAX,DWORD PTR FS:[ECX+30]
00402B2E    8B40 0C         MOV EAX,DWORD PTR DS:[EAX+C]
00402B31    8B70 14         MOV ESI,DWORD PTR DS:[EAX+14]
00402B34    AD              LODS DWORD PTR DS:[ESI]
00402B35    96              XCHG EAX,ESI
00402B36    AD              LODS DWORD PTR DS:[ESI]
00402B37    8B58 10         MOV EBX,DWORD PTR DS:[EAX+10]
00402B3A    8B53 3C         MOV EDX,DWORD PTR DS:[EBX+3C]
00402B3D    01DA            ADD EDX,EBX
00402B3F    8B52 78         MOV EDX,DWORD PTR DS:[EDX+78]
00402B42    01DA            ADD EDX,EBX
00402B44    8B72 20         MOV ESI,DWORD PTR DS:[EDX+20]
00402B47    01DE            ADD ESI,EBX
00402B49    31C9            XOR ECX,ECX
00402B4B    41              INC ECX
00402B4C    AD              LODS DWORD PTR DS:[ESI]
00402B4D    01D8            ADD EAX,EBX
00402B4F    8138 47657450   CMP DWORD PTR DS:[EAX],50746547
00402B55   ^75 F4           JNZ SHORT date.00402B4B
00402B57    8178 04 726F6:  CMP DWORD PTR DS:[EAX+4],41636F72
ECX=00000000

Address   Hex dump                                          ASCII
00402B28  31 C9 64 8B 41 30 8B 40 0C 8B 70 14 AD 96 AD 8B  1Éd■A0■@.■p¶-■-■
00402B38  58 10 8B 53 3C 01 DA 8B 52 78 01 DA 8B 72 20 01  X+■S<  Ú■Rx  Ú■r
00402B48  DE 31 C9 41 AD 01 D8 81 38 47 65 74 50 75 F4 81  Þ1ÉA-  ø8GetPuô
00402B58  78 04 72 6F 63 41 75 EB 81 78 08 64 64 72 65 75  x-rocAuëx◘ddreu
00402B68  E2 8B 72 24 01 DE 66 8B 0C 4E 49 8B 72 1C 01 DE  â■r$  Þf■.NI■r
00402B78  8B 14 8E 01 DA 31 C9 53 52 51 68 61 72 79 41 68  ■¶■  Ú1ÉSRQharyA
00402B88  4C 69 62 72 68 4C 6F 61 64 54 53 FF D2 83 C4 0C  LibrhLoadTSÿÒ■Ä.
00402B98  59 50 31 C0 66 B8 6C 6C 50 68 33 32 2E 64 68 75  YP1Àf¸llPh32.dhu
00402BA8  73 65 72 54 FF 54 24 10 83 C4 0C 50 31 C0 B8 65  serTÿT$►■Ä.P1À¸e
00402BB8  73 73 23 50 83 6C 24 03 23 68 50 72 6F 63 68 45  ss#P■l$♥#hProchE
00402BC8  78 69 74 54 FF 74 24 1C FF 54 24 1C 83 C4 0C 50  xitTÿt$∟ÿT$∟Ä.P
00402BD8  31 C0 B8 6F 78 41 23 50 83 6C 24 03 23 68 61 67  1À¸oxA#P■l$♥#hag
00402BE8  65 42 68 4D 65 73 73 54 FF 74 24 14 FF 54 24 20  eBhMessTÿt$¶ÿT$ 
00402BF8  83 C4 0C 50 31 C0 66 B8 21 21 50 68 61 69 6E 21  ■Ä.P1Àf¸!!Phain!
00402C08  68 79 20 41 67 68 61 20 54 72 68 20 50 69 6B 68  hy Agha Trh Pikh
00402C18  50 69 6B 61 54 31 C0 B8 43 48 55 23 50 83 6C 24  PikaT1À¸CHU#P■l$
00402C28  03 23 68 50 49 4B 41 54 31 C0 50 FF 74 24 04 FF  ♥#hPIKAT1ÀPÿt$♦ÿ
00402C38  74 24 14 31 C0 50 FF 54 24 38 83 C4 28 31 C0 B8  t$¶1ÀPÿT$8■Ä(1À¸
00402C48  62 65 72 23 50 83 6C 24 03 23 68 68 75 63 79 68  ber#P■l$♥#hhucyh
00402C58  69 6B 34 63 68 6C 76 65 50 68 74 33 63 74 68 66  ik4chlvePht3cthf
00402C68  73 64 33 54 31 C0 B8 43 48 55 23 50 83 6C 24 03  sd3T1À¸CHU#P■l$♥
00402C78  23 68 50 49 4B 41 54 31 C0 50 FF 74 24 04 FF 74  #hPIKAT1ÀPÿt$♦ÿt
00402C88  24 14 31 C0 50 FF 54 24 38 83 C4 28 31 C0 50 FF  $¶1ÀPÿT$8■Ä(1ÀPÿ
00402C98  54 24 08 CC CC CC CC CC 8B FF 55 8B EC 8B 4D 08  T$◘ÌÌÌÌÌ■ÿU■ì■M◘
```

Keep on debugging until you get the 1st Messagebox displaying “Pika Pika Try Again!!!”

```
00402C32    50              PUSH EAX
00402C33    FF7424 04       PUSH DWORD PTR SS:[ESP+4]
00402C37    FF7424 14       PUSH DWORD PTR SS:[ESP+14]
00402C3B    31C0            XOR EAX,EAX
00402C3D    50              PUSH EAX
00402C3E    FF5424 38       CALL DWORD PTR SS:[ESP+38]
00402C42    83C4 28         ADD ESP,28
00402C45    31C0            XOR EAX,EAX
00402C47    B8 62657223     MOV EAX,23726562
00402C4C    50              PUSH EAX
00402C4D    836C24 03 23    SUB DWORD PTR SS:[ESP+3],23
00402C52    68 68756379     PUSH 79637568
00402C57    68 696B3463     PUSH 63346B69
```



Press OK and proceed till you see the call to user32.MessageBoxA.

Check the pushed parameters and you have your flag, fsd3t3ctlvePik4chucyber or fsd3t3ct1vePik4chucyber based on the binary you have.

```
00402C8A   31C0          XOR EAX,EAX
00402C8C   50            PUSH EAX
00402C8D   FF5424 38     CALL DWORD PTR SS:[ESP+38]        user32.MessageBoxA
00402C91   83C4 28       ADD ESP,28
00402C94   31C0          XOR EAX,EAX
00402C96   50            PUSH EAX
00402C97   FF5424 08     CALL DWORD PTR SS:[ESP+8]
00402C9B   CC            INT3
00402C9C   CC            INT3
00402C9D   CC            INT3
Stack SS:[0012FF74]=7601EA11 (user32.MessageBoxA)
Jump from 00402C7C

Address   Hex dump                                          ASCII
00402B28  31 C9 64 8B 41 30 8B 40 0C 8B 70 14 AD 96 AD 8B   1Ed■A0■@.■p¶-■-■
00402B38  58 10 8B 53 3C 01 DA 8B 52 78 01 DA 8B 72 20 01   X←■S<  Ú■Rx  Ú■r
00402B48  DE 31 C9 41 AD 01 D8 81 38 47 65 74 50 75 F4 81   Þ1ÉA-  Ø8GetPuô
00402B58  78 04 72 6F 63 41 75 EB 81 78 08 64 64 72 65 75   x♦rocAuë x◘ddreu
00402B68  E2 8B 72 24 01 DE 66 8B 0C 4E 49 8B 72 1C 01 DE   â■r$ Þf■.NI■r
00402B78  8B 14 8E 01 DA 31 C9 53 52 51 68 61 72 79 41 68   ■¶■  Ú1ÉSRQharyA
00402B88  4C 69 62 72 68 4C 6F 61 64 54 53 FF D2 83 C4 0C   LibrhLoadTSÿÒ■Â.
00402B98  59 50 31 C0 66 B8 6C 6C 50 68 33 32 2E 64 68 75   YP1Àf llPh32.dhu

0012FF3C  00000000
0012FF40  0012FF5C  ASCII "fsd3t3ctlvePik4chucyber"
0012FF44  0012FF50  ASCII "PIKACHU"
0012FF48  00000000
0012FF4C  0012FF50  ASCII "PIKACHU"
0012FF50  414B4950
0012FF54  00554843
0012FF58  0012FF5C  ASCII "fsd3t3ctlvePik4chucyber"
0012FF5C  33647366
```

QUESTION 6

Debug the application to get the flag:
HINT: What day is it?

File/Folder Name: Q6

```
--------------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
--------------------------------------------------------------------------------
```

ANSWER Q6

For this challenge, the file has an icon that gives away the fact that it's a Python script compiled into a binary exe file. Also by examining the strings, it can be seen that there are a lot of references to Python functions and modules, which also indicates the same thing.

The second step is using a popular de-compiler from py/exe → pyc . One that can be used for this case is: https://github.com/countercept/python-exe-unpacker.

Then run the command below after cloning the above repo:

```
python pyinstxtractor.py <filename>
```

A folder will be created with the filename_extracted as a dir. Inside the folder, you can find the Python modules used with the bytecode as well as <filename> as data file type.

On opening it in hex-editor, you can find the flag starting with <fs….cyber>. Also, the date it compares with can be seen, 1999-09-09.

QUESTION 7

Investigate the image and then decode the flag.

File/Folder Name: Q7

```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
-----------------------------------------------------------------------------
```

ANSWER Q7

The given image is encoded using Morse code. Decoding the picture will reveal the Morse code. Deciphering the Morse code will get the flag.

QUESTION 8

Debug the application and identify the flag.

File/Folder Name: Q8

Dependencies:

- Either:
  - Visual C++ Redistributable for Visual Studio 2015
    - https://www.microsoft.com/en-my/download/details.aspx?id=48145 or
    - https://www.microsoft.com/en-us/download/details.aspx?id=53587
- Or:
  - Visual C++ Redistributable Packages for Visual Studio 2013
    - https://www.microsoft.com/en-my/download/details.aspx?id=40784

```
-----------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
-----------------------------------------------------------------------------
```

ANSWER Q8

In this challenge, you are provided with a PE file. Firstly, we will execute the program, and by doing so, the program appears to display INPUT THE KEY: and takes in an input. Putting in some random string, it prints out a failure message, as shown below.

```
INPUT THE KEY: fwefwe
I'll give you an A for effort... but better luck next time!
Press any key to continue . . .
```

We can now open the file in a disassembler (e.g. IDA) to understand its overall flow and structure. By following the control flow, you can reach to the beginning of the user code section, as shown below:

```
loc_401AC7:
call    __p___argv
mov     edi, [eax]
call    __p___argc
mov     esi, eax
call    _get_initial_narrow_environment
push    eax
push    edi
push    dword ptr [esi]
call    sub_401090
add     esp, 0Ch
mov     esi, eax
call    sub_40225F
test    al, al
jz      short loc_401B5C
```

Here you can see that the program outputs "INPUT THE KEY: " and takes an input (std::basic_istream), and stores it inside a variable (shown below as input_val).

```
mov     edx, offset aInputTheKey ; "INPUT THE KEY: "
mov     ecx, ds:?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ; std::basic_ostream<char,std::char_traits<char>> std::cout
call    sub_401480
lea     eax, [ebp+input_val]
push    eax
mov     ecx, ds:?cin@std@@3V?$basic_istream@DU?$char_traits@D@std@@@1@A ; std::basic_istream<char,std::char_traits<char>> std::cin
call    ds:??5?$basic_istream@DU?$char_traits@D@std@@@std@@@QAEAAV01@AAH@Z ; std::basic_istream<char,std::char_traits<char>>::operator>>(int &)
call    func_isDebbuger
movzx   ecx, al
test    ecx, ecx
jnz     func_fail_out
```

However, it immediately calls a function (renamed as func_isDebugger in the image above), which returns the value of IsDebuggerPresent().

Based on the conditional jump, it will jump to func_fail_out (as shown above) if IsDebuggerPresent() returns true, and won't jump if it returns false.

Following the conditional jump, the program prints out variable fail_message (shown below):

```
mov     edx, offset aInputTheKey ; "INPUT THE KEY: "
mov     ecx, ds:?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ; std::basic_ostream<char,std::char_traits<char>> std::cout
call    sub_401480
lea     eax, [ebp+input_val]
push    eax
mov     ecx, ds:?cin@std@@3V?$basic_istream@DU?$char_traits@D@std@@@1@A ; std::basic_istream<char,std::char_traits<char>> std::cin
call    ds:??5?$basic_istream@DU?$char_traits@D@std@@@std@@QAEAAV01@AAH@Z ; std::basic_istream<char,std::char_traits<char>>::operator>>(int &)
call    func_isDebbuger
movzx   ecx, al
test    ecx, ecx
jnz     func_fail_out
```

```
cmp     [ebp+input_val], 539h
jnz     func_fail_out
```

```
mov     edx, 1
test    edx, edx
jz      func_fail_out
```

```
mov     eax, 1
test    eax, eax
jz      func_fail_out
```

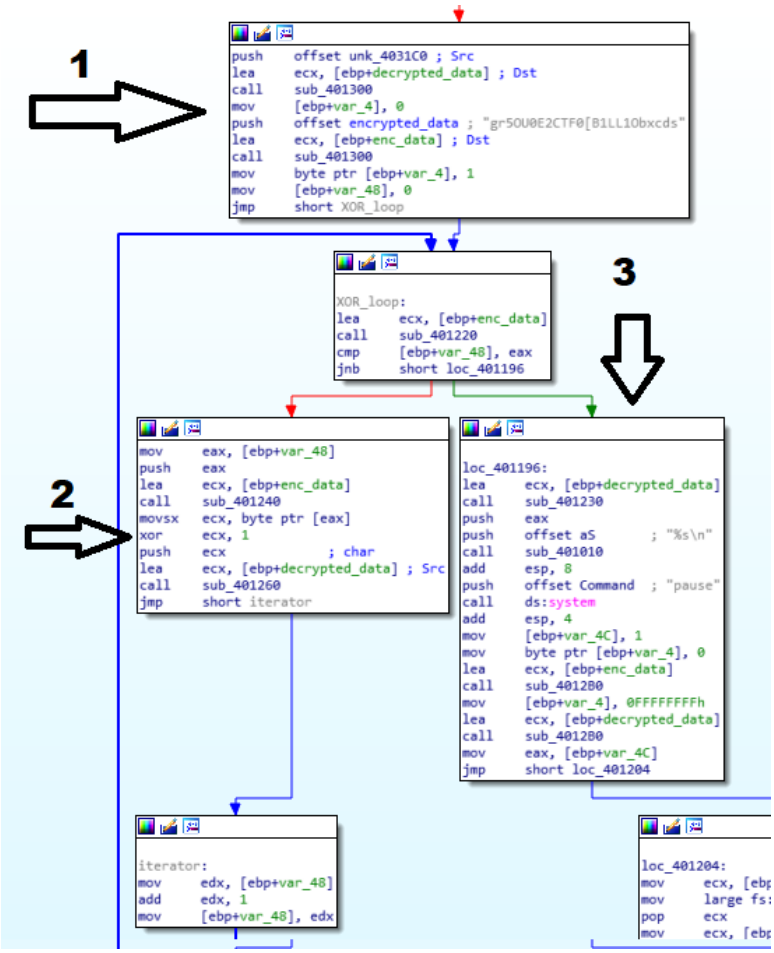
```
call    sub_401050
movzx   ecx, al
test    ecx, ecx
jnz     func_fail_out
```

```
func_fail_out:
mov     edx, offset fail_message ; "I'll give you an A for effort... but be"...
mov     ecx, ds:?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ; std::basic_ostream<char,std::char_traits<char>> std::cout
call    sub_401480
push    offset Command  ; "pause"
call    ds:system
add     esp, 4
mov     eax, 1
```

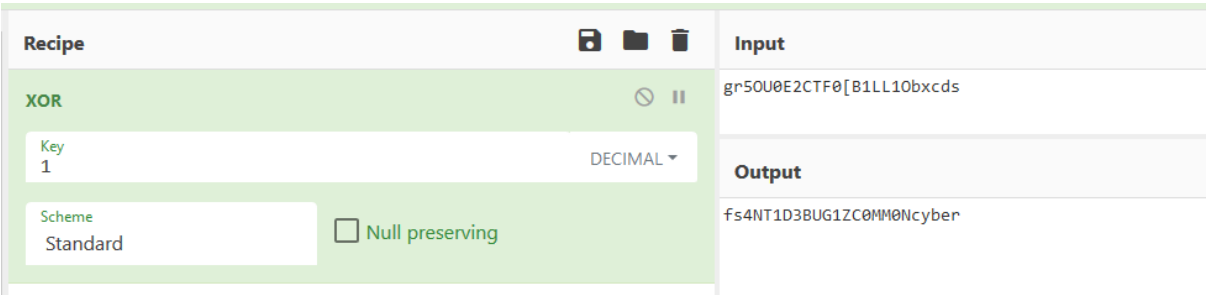
Looking at the graph, it appears that there are a set of consecutive conditional jumps that lead to func_fail_out, which indicate the conditional jumps that we would need to avoid.

At this point, we can immediately pass over these (it compares input_val to 539h (1337 in decimal)), which is followed by two tautological conditions which will always pass and also calls a subroutine that checks for DBG, dbg or ID in the window name of the top-level window).

Passing over these, we can see that the next section of the code appears to be XORing each character of the string "gr5OU0E2CTF0[B1LL1Obxcds" with 0x01 inside a loop, and later on printing it out, which we can suspect it to be the flag (protip: it is).



With this information, we can XOR each char of gr5OU0E2CTF0[B1LL1Obxcds with 0x01 (without the need for debugging), and get the flag, as shown below.



Flag: fs4NT1D3BUG1ZC0MM0Ncyber

QUESTION 9

Decode the file and identify the flag.
HINT: This file is encoded

File/Folder Name: Q9

```
---------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
---------------------------------------------------------------------------
```

ANSWER Q9

This challenge is an encoded VB Script without a file extension. The participant should be able to identify the filetype and encoding using the file header to decode the flag.

```
00000000:  23 40 7E 5E-4E 67 41 41-41 41 3D 3D-5C 6B 6F 24  #@~^NgAAAA==\ko$
00000010:  4B 36 2C 4A-30 6B 2B 09-5E 21 39 6B-55 6F 7C 78  K6,J0k+o^!9kUo|x
00000020:  47 32 4D 21-34 5E 6E 3A-31 58 34 7F-2E 45 7E 2C  G2M!4^n:1X4⌂.E~,
00000030:  76 6C 7E 7E-4A 50 34 7F-50 57 56 6D-4C 50 62 2F  vl~~JP4⌂PWVmLPb/
00000040:  6C 45 38 42-45 41 41 41-3D 3D 5E 23-7E 40 00     lE8BEAAA==^#~@
```

Searching “#@~^” will show references about VBE (Encoded VB). And using Microsoft Script Decoder, the flag will be revealed.

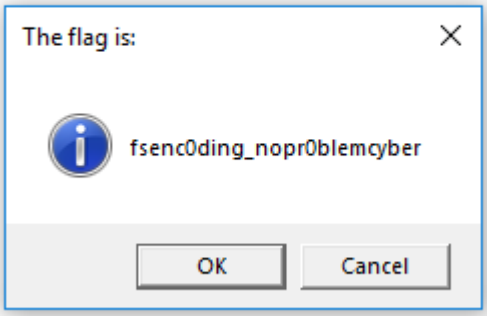
Input length: 79 lines: 1

```
#@~^NgAAAA==\ko$K6,J0k+
^!9kUo|xG2M!4^n:1X4 .E~,vl~~JP4 PWVmLPb/lE8BEAAA==^#~@
```

Output time: 1ms length: 54 lines: 1

```
MsgBox "fsenc0ding_nopr0blemcyber", 65, "The flag is:"
```

Adding “.vbe” extension and executing the file will also resolve the flag.



QUESTION 10

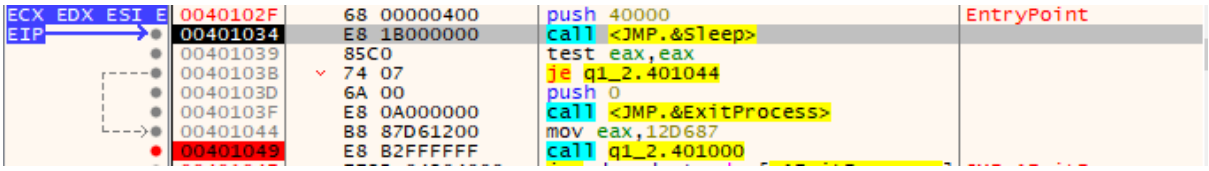
Debug the application and identify the flag.

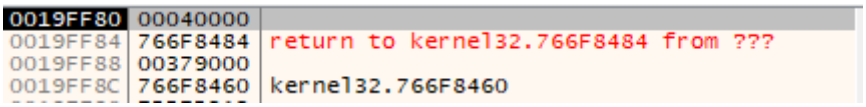
File/Folder Name: Q10

```
---------------------------------------------------------------------------
The answer for this challenge follows standard F-Secure Cyber Security 2019
competition flag:
fs<flag>cyber
Examples:
fsrandom_lksmlfkmdfijsfdsfcyber
fs_s0meth1ng_s0meth1ng_cyber
fsiamhandsomecyber
---------------------------------------------------------------------------
```

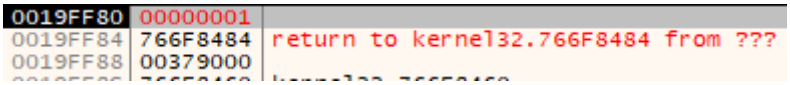
ANSWER Q10

This challenge requires the application to gracefully sleep before proceeding to the decryption of the flag. Sleep requires a DWORD value to be pushed in stack. In our case, 262.144 Seconds (0x040000h) was pushed to stack as parameter to the Sleep function.





We can manipulate the stack and replace it with a lower value (0x00000001h) to speed up Sleep to 1ms.



After the successful sleep, the application is now able to proceed to the decryption function (call sub_401000).

```
sub_401000 proc near
push    40000h          ; dwMilliseconds
call    Sleep
inc     eax
add     eax, 6
mov     ecx, 22h ; '"'
lea     esi, aUbeDSnugk7wxc4 ; "ube~d~snuGk7wXc4#u4qbUta" ; Encrypted Flag
lea     ebx, unk_403019
```

```
loc_40101F:             ; decryption loop using BYTE xor key
mov     dl, [esi]
xor     dl, al
mov     [ecx+ebx-1], dl ; Write decryted byte to Memory adress
inc     esi
dec     ecx
cmp     ecx, 0
jnz     short loc_40101F ; decryption loop using BYTE xor key
```

```
retn
sub_401000 endp
```

There is another Sleep API function and can be easily bypassed by modifying the pushed value in the stack. After the successful sleep, it will proceed with the decryption using a BYTE xor before it exits (RETN).

Address	Hex	ASCII
00403023	66 73 52 65 76 33 72 24 33 64 5F 70 30 6C 40 72	fsRev3r$3d_p0l@r
00403033	69 74 79 63 79 62 65 72 00 00 00 00 00 00 00 00	itycyber.....
00403043	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....