F-Secure Cyber Security Competition Malaysia 2019

Semi-Finals Round Questions & Answers

Question 1

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

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

In this challange you just have to identify and remove IEX (Invoke-Expression) from powershell Depending on the sample you receive you will find .($enV:cOMsPEc[4,15,25]-JoIN'') at the start of script or |&($enV:cOMSpEC[4,26,25]-JOIN'') at the end of the script.

$enV:cOMsPEc[4,15,25]-JoIN and $enV:cOMSpEC[4,26,25]-JOIN is 'IEX"

Just remove .($enV:cOMsPEc[4,15,25]-JoIN'') or |&($enV:cOMSpEC[4,26,25]-JOIN'') based on your script and run the powershell. Deobfuscated script will be shown which has the flag

```
PS C:\Users\NEO> ("$(set-itEM  'VarIabLe:Ofs' '') " +[STRiNG]( '24z78!3d-32-30h3b&69&66-20N28h24h78h20-2d!6cz65
 $x=20;if ($x -le 10){Write-host("fsThIngsUDO4cyber")}else{Write-host("No, This is not the answer!!!")}


PS C:\Users\NEO> -JOiN( '24-78I3dw32I30V3bj69%66p20V28j24V78-20,2d-6cV65-20,31-30w29j7bV57V72-69I74V65I2dp68-6f
$x=20;if ($x -le 10){Write-host("fsThIngsVDO4cyber")}else{Write-host("No, This is not the answer!!!")}
```

Question 2

Investigate the DOS command and identify the flag.

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

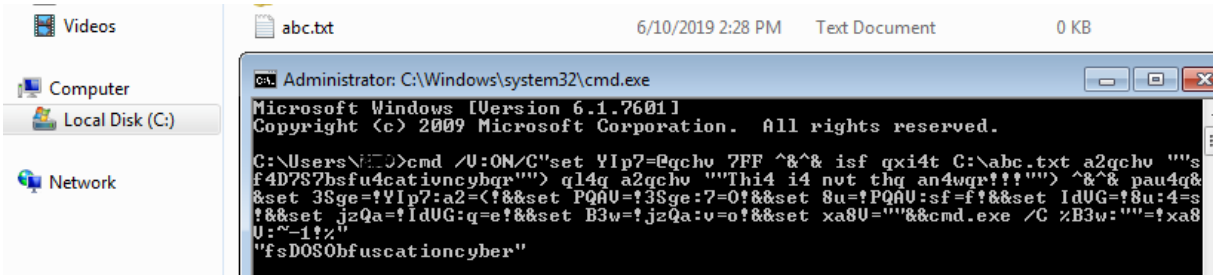
2 ways to solve :

1) Hard way : Replace all the set variables value back into the code.
   For example
   set DEO=!FEb:j=@! → replace j with @
   set lOTk=!DEO:m=s! → replace m with s
   set 3Sge=!YIp7:a2=(! → replace a2 with (
   set PQAV=!3Sge:7=O! → replace 7 with O

   After replacements you will see the echoed flag

2) Easy way :

   It's visible in obfuscated code that C:\abc.txt is checked for existence. Creating c:\abc.txt and running command gives result.



Question 3

Find and identify the flag hidden in the document.

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

One of the way to solve the challenge:
1. Open the file using Microsoft Word.
2. Press CTRL+A to select all of the elements on the page.
3. Notice there are several square shapes in a white color.
4. Inspect each the Alternate text section of each shape by: Right click on the shape > Select Format AutoShape/Picture > Select AltText tab.
5. The square shape at the top left corner on the page contains the actual flag in the AltText section.

Question 4

Find and identify the flag from the given audio file.
HINT: try to unlock the flag.

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

By guessing the key. We find that the key is “fsecure” similar to previous challenges. How to solve:
1) There is a hidden text file inside the audio file.
2) Tools that can be used to solve it:

- QuickStego
- AudioStegano
- DeepSound
- MP3Stego
- Steghide

Question 5

Debug and identify the the flag from the provided script artifact's.

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

Decompile the pyc to py, or use https://python-decompiler.com/en/ to decompile back to source code.

Recommended tool uncompyle6: https://pypi.org/project/uncompyle6/

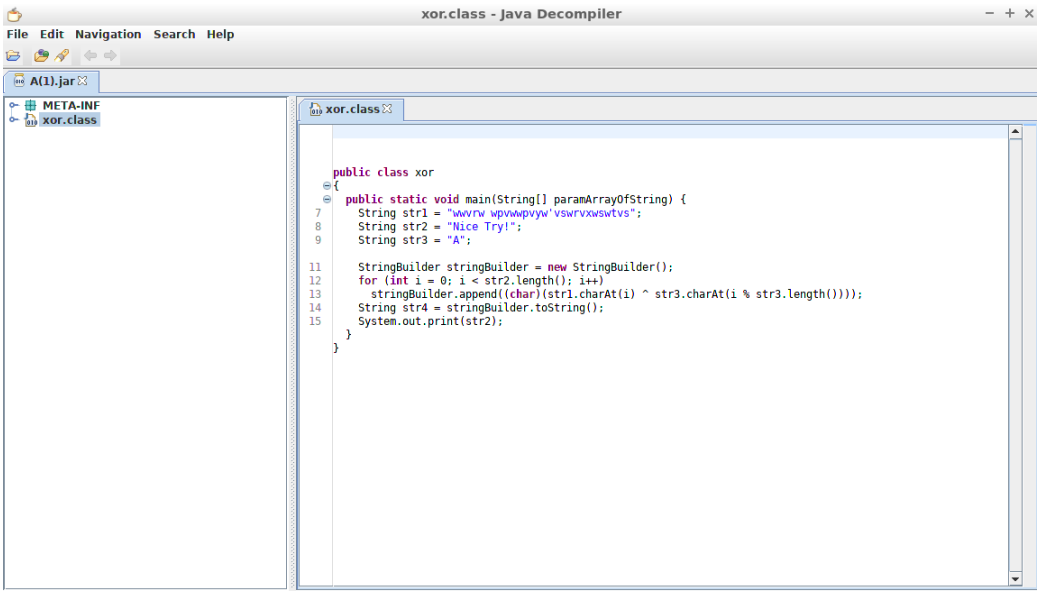
Question 6

Debug the jar file to identify the flag:
HINT: What is printed?

```
--------------------------------------------------------------------------------
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 we can view the Java code using Java decompiler which can be found in the below URL:
https://github.com/java-decompiler/jd-gui/releases
When opening the challenge using the jd-gui we see a class called xor.class



If we analyse the code, we will notice that it's a simple XOR against the key "A":

so "wwvrw wpvwwpvyw'vswrvxwswtvs" ^ "A" = flag

To achieve this, either XOR the two values our self's or modify the printed str2 from last function to "str4" which contain the XORed flag.

Question 7

Can you crack the password and get the flag.
Hint: Remember John? The ripper that rockyou.

```
--------------------------------------------------------------------------
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 challenge can be brute-forced using rockyou password list and it may take up long duration to find the correct password.

The correct way to solve the challenge is:
1. Use file command to find out the file type. Potential compression method: gzip, bzip2, tar, 7z

2. Use the correct decompression tools to decompress:

- For gzip: *gzip -d [filename]*
- For bzip2: *bzip2 -d [filename]*
- For tar: *tar -xvf [filename]*
- For 7z: *7z x [filename]*

3. Use John the Ripper with rockyou password list to brute-force the password.

1. zip2john [filename] >> zip.hash
2. john --wordlist=rockyou.txt zip.hash

Question 8

Debug the application (1171.exe) and identify the flag.
Hint: packed?fake?

Dependencies:
libgcc_s_dw2-1.dll

libstdc++-6.dll
libwinpthread-1.dll

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

File asks the user a set of TRUE/FALSE questions.

File is UPX packed and the UPX signature is spoofed with a fake visual studio signature.

Once unpacked using UPX unpacker and loaded in Ollydbg, user can see what to answer through ---> Reference Strings.

```
[ESP],Set_A.004A6045    ASCII "true"
[ESP],Set_A.004A6045    ASCII "true"
[ESP],Set_A.004A604A    ASCII "false"
[ESP],Set_A.004A6045    ASCII "true"
[ESP],Set_A.004A604A    ASCII "false"
[ESP],Set_A.004A604A    ASCII "false"
[ESP],Set_A.004A6045    ASCII "true"
[ESP],Set_A.004A6045    ASCII "true"
[ESP],Set_A.004A6045    ASCII "true"
[ESP],Set_A.004A604A    ASCII "false"
```

As can seen from above first question when running the binary require the right answer “true”. Second one is also “true” then “false” and so on until the flag is printed for you 😊

Question 9

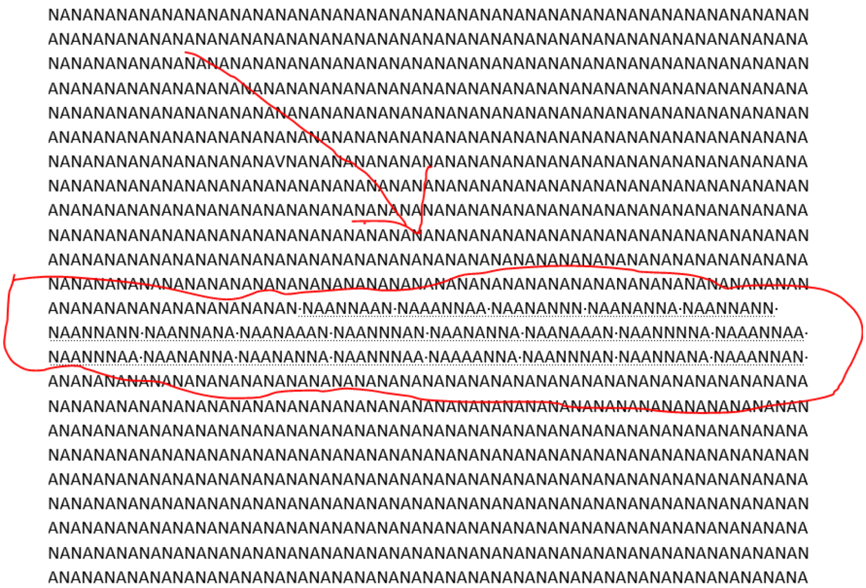
Investigate the provided doc as it hold important data.
Keep in mind that we got the file from a spy.
HINT: The string is encoded


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

N = 0
A = 1

Show hidden font --> see in the middle of the noise



1. There will be highlighted strings of N's and A's
2. Copy that then replace the N's with = 0
3. Replace the 1's with 1
4. Then copy the output to = Binary to Ascii

Question 10

Debug the application and identify the flag.

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

In this challenge, you are provided with a PE file. Upon executing it, it appears to output a Youtube link (shown below).

We're going to first disassemble the file in IDA to get an understanding of the overall flow and structure, as well as identify anti-debugging checks that can be bypassed. Once the file is loaded, we're first going to the initialized data section (press Ctrl+S to open jump to segment dialog box and select .rdata section) to look for interesting strings similar to the Youtube link.

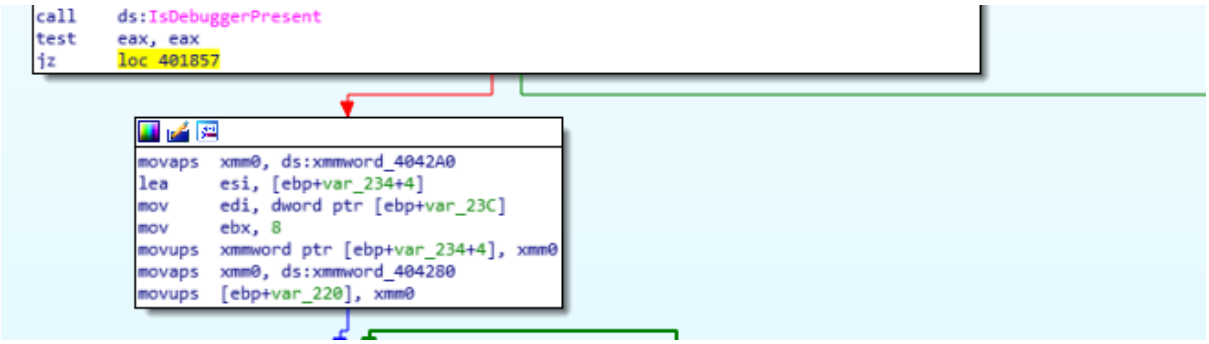
Scrolling through the section, we see three interesting variables at 0x4041c8, 0x4041f4, 0x40422. We'll flatten the variable at 0x4041f4 by selecting the address and pressing A to get the image below.

```
.rdata:004041C8 aHttpsWwwYoutub db 'https://www.youtube.com/watch?v=dQw4w9WgXcQ',0
.rdata:004041C8                                         ; DATA XREF: sub_401000+8C8↑o
.rdata:004041C8                                         ; sub_401000+8ED↑o
.rdata:004041F4 aSomePeopleJust db '¯_(?)_/¯ SOME PEOPLE JUST LIKE TO WATCH THE WOLRD BURN',0
.rdata:004041F4                                         ; DATA XREF: sub_401000+629↑o
.rdata:0040422B                 db    0
.rdata:0040422C aHttpsYoutuBe1x db 'https://youtu.be/1XZGHOxnCto',0
.rdata:0040422C                                         ; DATA XREF: sub_401000:loc_401809↑o
```

Now we'll rename (by selecting the address and pressing N) the variable at 0x4041c8 (which was outputted earlier on) as youtube_link_1, 0x4041f4 as weird_text, and 0x40422 as youtube_link_2, as shown below.

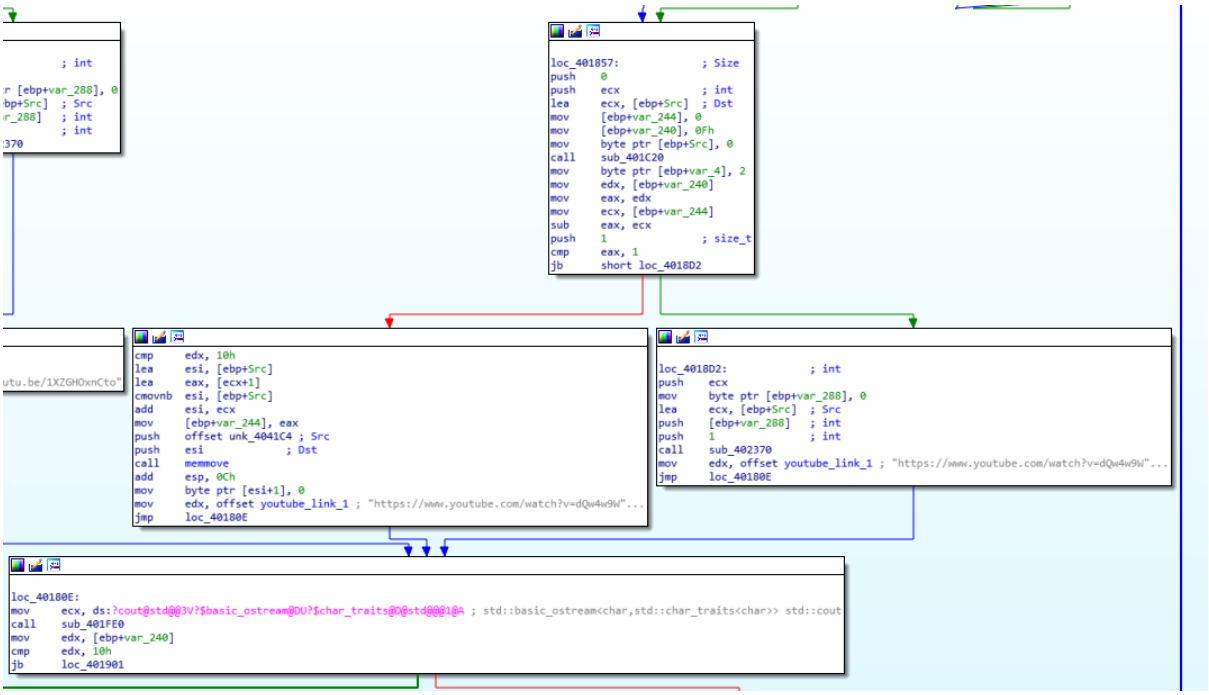
```
.rdata:004041C8 youtube_link_1  db 'https://www.youtube.com/watch?v=dQw4w9WgXcQ',0
.rdata:004041C8                                         ; DATA XREF: sub_401000+8C8↑o
.rdata:004041C8                                         ; sub_401000+8ED↑o
.rdata:004041F4 weird_text      db '¯_(?)_/¯ SOME PEOPLE JUST LIKE TO WATCH THE WOLRD BURN',0
.rdata:004041F4                                         ; DATA XREF: sub_401000+629↑o
.rdata:0040422B                 db    0
.rdata:0040422C youtube_link_2  db 'https://youtu.be/1XZGHOxnCto',0
.rdata:0040422C                                         ; DATA XREF: sub_401000:loc_401809↑o
```

Now we'll go to the beginning of the code section (0x401000) and analyse the program flow. Initially, we can look for possible anti-debugging techniques and conditional jumps that change the control flow. The first one appears to be at 0x40147f (shown below)

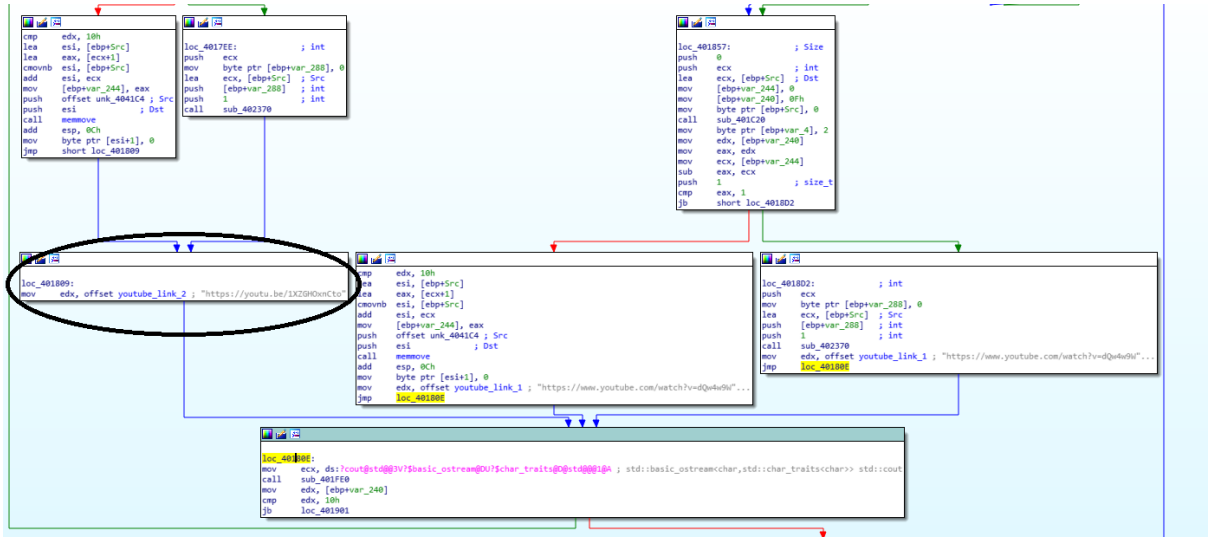


Before this conditional jump, the program calls IsDebuggerPresent then test eax, eax (which performs bitwise AND on the return value of IsDebuggerPresent which is stored in eax and stores the value in zero flag, ZF). Therefore, the program will jump if IsDebuggerPresent returns false, and it won't jump if it true.

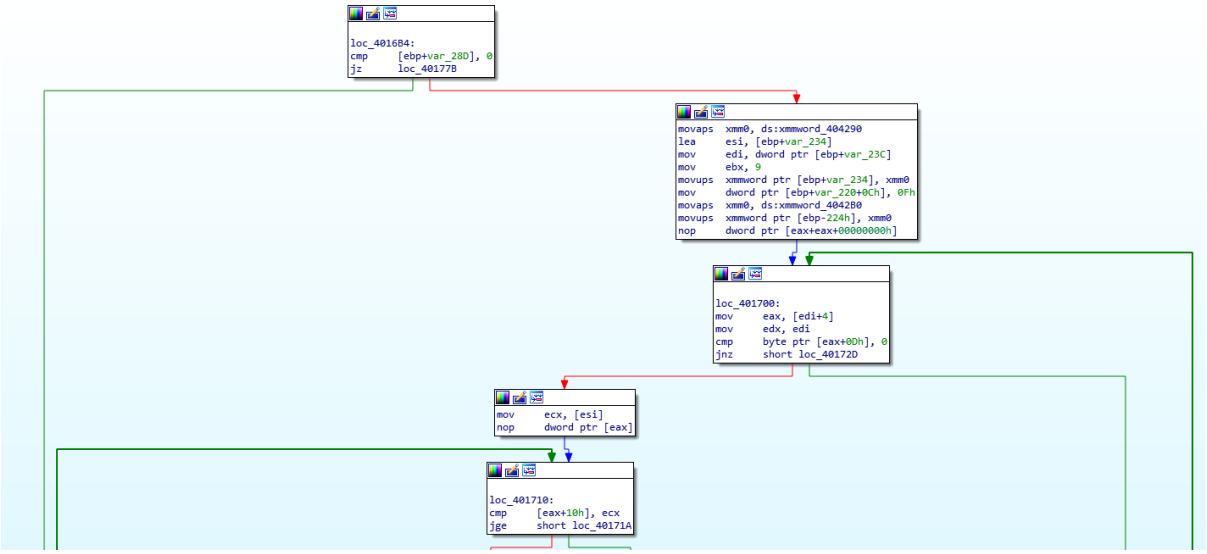
Looking at the program flow if the jump is taken, it looks like the program will print out youtube_link_1 (which is what was printed out when the program was executed). So we can immediately eliminate this branch, and then deduce that the program WANTS to be debugged.



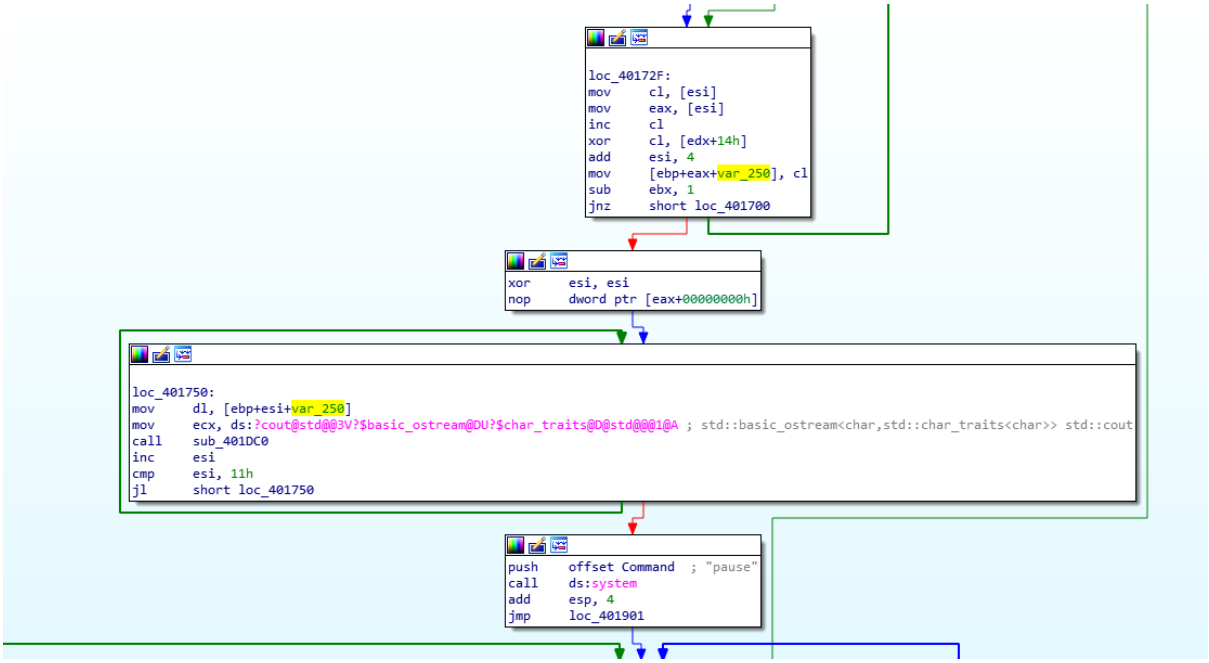
However, while analysing this branch, we can see that there are three code blocks that actually lead to the same function (0x40180e) that calls std::basic_ostream. Ignoring the two with youtube_link_1, we can see that the last code block appears to print youtube_link_2 (shown below)



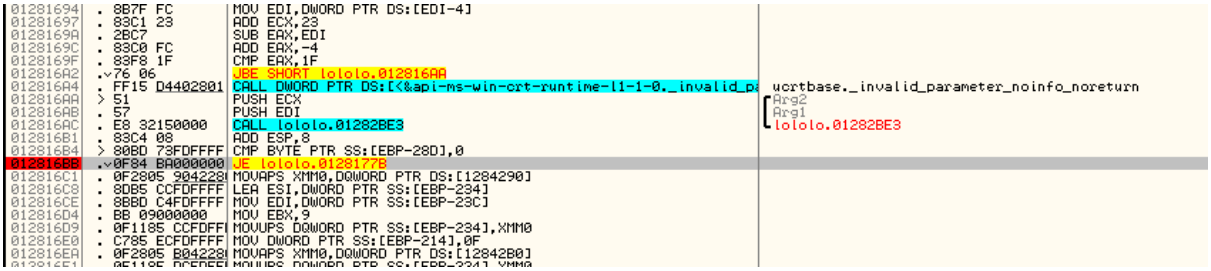
That means that there is another conditional jump that leads to this change in control flow and hence printing out the second youtube link (which is the second dead-end). Using this information, one technique we could use, would be to trace back from 0x401809 to the conditional jump that leads to this. By doing so, we arrive at address 0x4016bb (shown below).



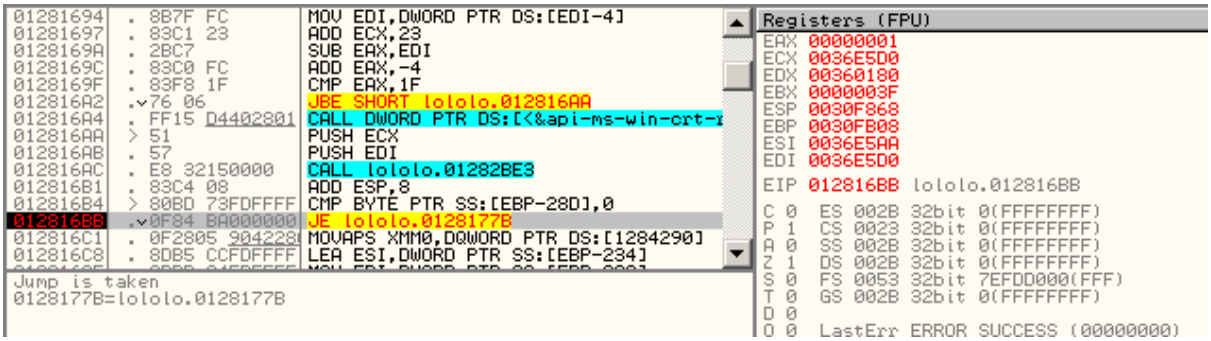
Following the other branch of this conditional jump, it appears to be decrypting/decoding some variable and printing it out then calling system pause command (shown below).



Having this information, it means that we can try and load the file in a debugger, set a breakpoint at the conditional jump that changed the control flow (0x4016bb) inside the program, which would be @ beginning_of_code_section + 0x6bb (0x4016bb – 0x401000), as shown below.



Executing the binary now, we will reach to the breakpoint, indicating that no conditional jumps have changed the control flow thus far. However, we can see that the jump will be taken, which indicates that this conditional jump is related to another check inside the program which is preceding the conditional jump (it checks if the window name of the top-level window matches the variable weird_text, which we set inside the initialized data section earlier), however, as we are interested on getting to the flag, we can simply change ZF to 0 and continue the execution.



After continuing with the execution, we can see the outputted flag inside the console window, as shown below.

Flag: fsR1CKR0LL3Dcyber