HTB Reverse Engineering Shuffleme

Tools: Ghidra, GDB, CyberChef

For this challenge we are mostly going to use the GDB debugger upgraded to PWNDBG.

Starting off in Ghidra we can go straight to the GO function and read what it does.

```
extract_blob((long)key_blob,0x20,(long)local_88);
extract_blob((long)data_blob,0x50,(long)local_68);
pEVar1 = EVP_aes_256_cbc();
```

This is the main info we need. We can see that the key_blob is 32 bytes (0x20) and that data_blob is 80 bytes (0x50). Also that it is being encrypted with AES 256 CBC.

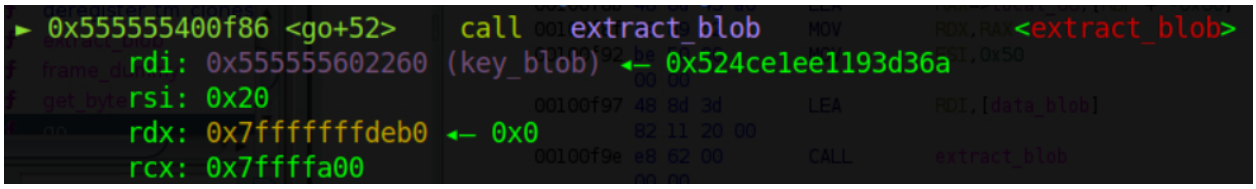
```
          00 00
00100f7f 48 8d 3d        LEA         RDI,[key_blob]
          da 12 20 00
00100f86 e8 7a 00        CALL        extract_blob
          00 00
00100f8b 48 8d 45 a0     LEA         RAX=>local_68,[RBP + -0x60]
00100f8f 48 89 c2        MOV         RDX,RAX
00100f92 be 50 00        MOV         ESI,0x50
          00 00
00100f97 48 8d 3d        LEA         RDI,[data_blob]
          82 11 20 00
00100f9e e8 62 00        CALL        extract_blob
          00 00
00100fa3 48 8b 85        MOV         RAX,qword ptr [RBP + local_
          78 ff ff ff
00100faa 48 89 c7        MOV         RDI,RAX
00100fad e8 de fc        CALL        <EXTERNAL>::EVP_aes_256_cbc
```

Now in order to get the data from the key_blob and data_blob we need to debug them and extract the bytes. So we are going to go into GDB and make our way to the GO function and set our break points at the call for extract_blob and instruction just after so we can get the correct bytes.(Note: extracting the bytes before navigating to the extract_blob will not yield correct data and therefore you cannot get the flag.)

```
pwndbg> starti
pwndbg> break *go+52
Breakpoint 1 at 0x555555400f86
pwndbg> break *go+57
Breakpoint 2 at 0x555555400f8b
```

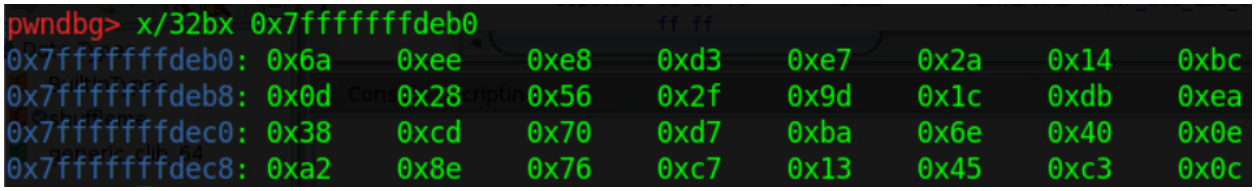
pwndbg> break *go+76
Breakpoint 3 at 0x555555400f9e
pwndbg> break *go+81
Breakpoint 4 at 0x555555400fa3
pwndbg> run 2

#the parameter after run should not matter. 2 was my choice since I needed a parameter so the program wouldn't crash/exit.



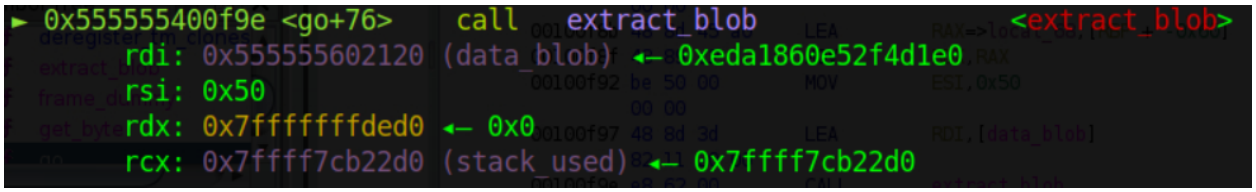
Now we can see the memory address at rdx that is holding our bytes. (Note:RDX is the address of the buffer holding the extracted data.) Since this is for the key_blob we will need 32 bytes by running the command x/32x 0x7fffffffdeb0.

pwndbg> c
pwndbg> x/32x 0x7fffffffdeb0



Great now we have our key! Save that for later and lets get the data for the flag.

pwndbg> c

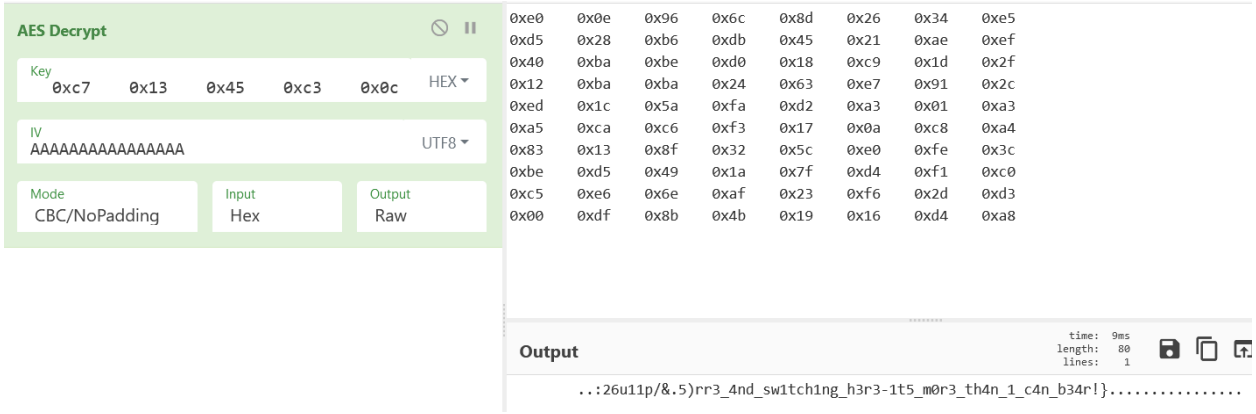


Again we can see our memory address for RDX holding the data we need. And this time we need 80 bytes.

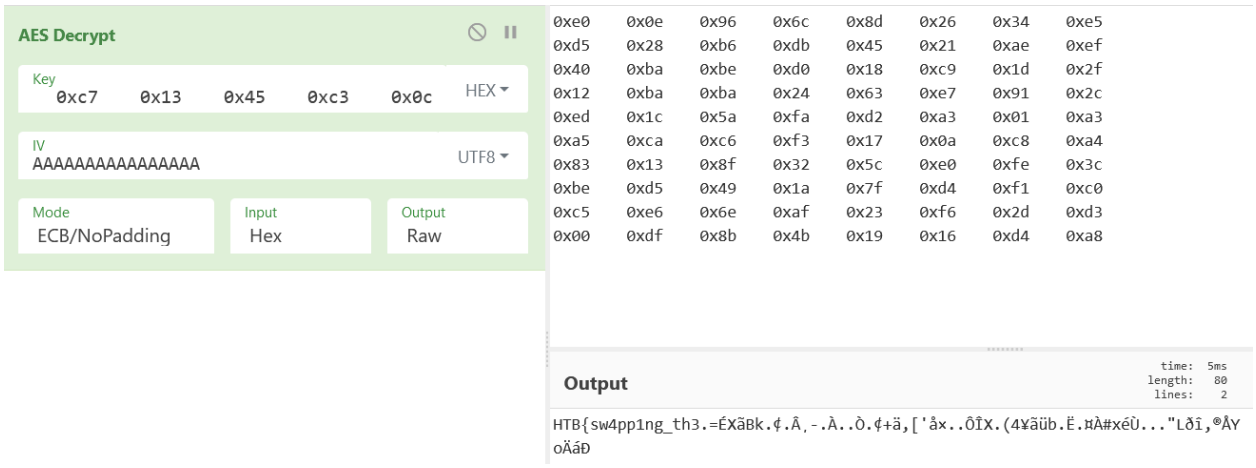
pwndbg> c
pwndbg> x/80bx 0x7fffffffded0

```
pwndbg> x/80bx 0x7fffffffded0
0x7fffffffded0: 0xe0    0x0e    0x96    0x6c    0x8d    0x26    0x34    0xe5
0x7fffffffded8: 0xd5    0x28    0xb6    0xdb    0x45    0x21    0xae    0xef
0x7fffffffdee0: 0x40    0xba    0xbe    0xd0    0x18    0xc9    0x1d    0x2f
0x7fffffffdee8: 0x12    0xba    0xba    0x24    0x63    0xe7    0x91    0x2c
0x7fffffffdef0: 0xed    0x1c    0x5a    0xfa    0xd2    0xa3    0x01    0xa3
0x7fffffffdef8: 0xa5    0xca    0xc6    0xf3    0x17    0x0a    0xc8    0xa4
0x7fffffffdf00: 0x83    0x13    0x8f    0x32    0x5c    0xe0    0xfe    0x3c
0x7fffffffdf08: 0xbe    0xd5    0x49    0x1a    0x7f    0xd4    0xf1    0xc0
0x7fffffffdf10: 0xc5    0xe6    0x6e    0xaf    0x23    0xf6    0x2d    0xd3
0x7fffffffdf18: 0x00    0xdf    0x8b    0x4b    0x19    0x16    0xd4    0xa8
```

Now that we have both the key_blob and data_blob bytes we can take these values into cyberchef to finally decipher the code. So head to https://gchq.github.io/CyberChef/



Even though we did everything we only got half the flag but after trying out ECB/Nopadding we can get the rest.



BOOM! There is the whole flag put together as HTB{sw4pp1ng_th3r3_4nd_sw1tch1ng_h3r3-1t5_m0r3_th4n_1_c4n_b34r!}

Summary: I had some help with this one and learned a lot more about debugging. Specifically about how RDX works and why the data was stored there for our flag. This may not be the actual way to achieve this flag since using cyberchef needed CBC and ECB to fully decipher but it is still a path to the goal. Until someone solves it another way I will stick with this method.