DESCRIPTION

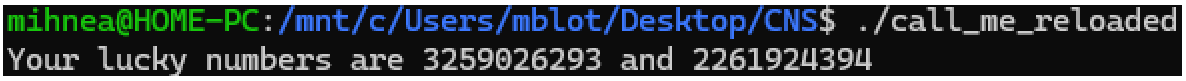
Someone has tampered with the executable file. Please fix it and call the call_me function! The fixed binary should give you the flag when you run it.

RESOURCES

As part of the challenge I received an executable file called **call_me** as an attachment for analysis.

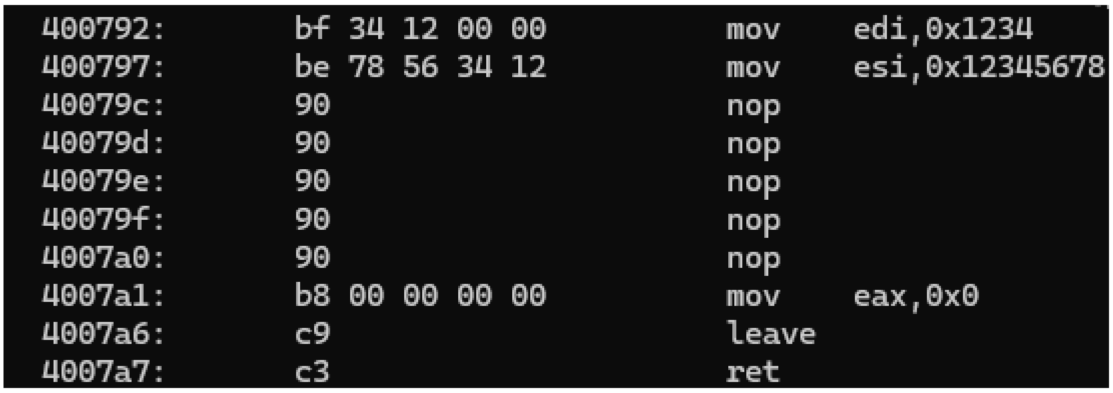
APPROACHES

1. First of all, I tried to just run the executable and I ended up with a message and two random numbers every time.



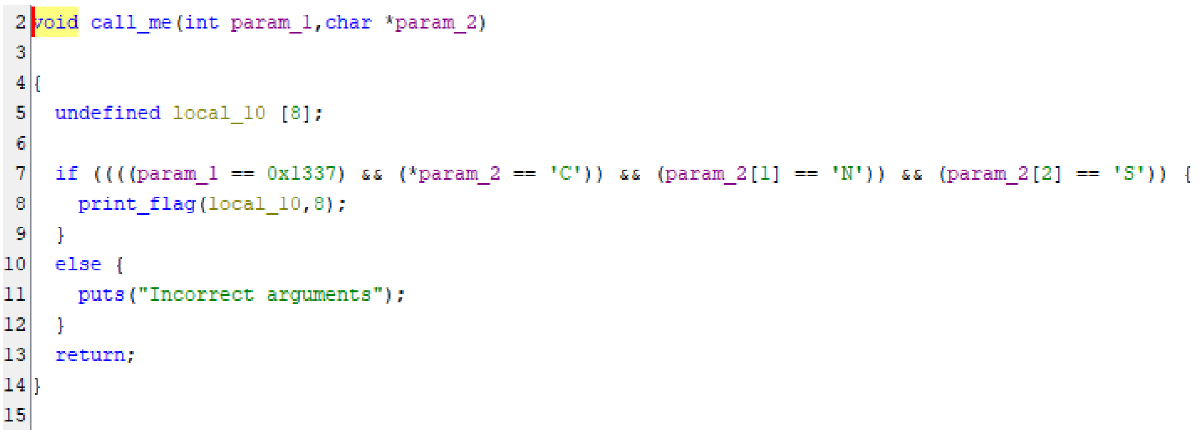
```
mihnea@HOME-PC:/mnt/c/Users/mblot/Desktop/CNS$ ./call_me_reloaded
Your lucky numbers are 3259026293 and 2261924394
```

2. Then, this behavior and the name of the task made me think that is very similar to **call_me**. And I directly went to decompile the main function using Ghidra and objdump. I found out a similar main function with 5 NOPs at the end of the it that I should replace with a call to **call_me.**



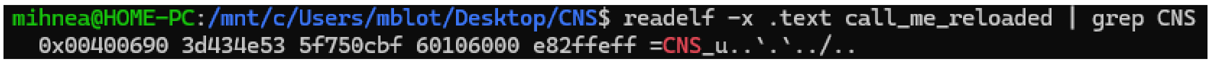
```
400792:       bf 34 12 00 00          mov    edi,0x1234
400797:       be 78 56 34 12          mov    esi,0x12345678
40079c:       90                      nop
40079d:       90                      nop
40079e:       90                      nop
40079f:       90                      nop
4007a0:       90                      nop
4007a1:       b8 00 00 00 00          mov    eax,0x0
4007a6:       c9                      leave
4007a7:       c3                      ret
```

3. However, this time I can see that before the NOPs, there isn't another call there are some values stored in ESI and EDI which are the first two parameters of a function so that made me check call_me function to see if it gets any parameters this time.



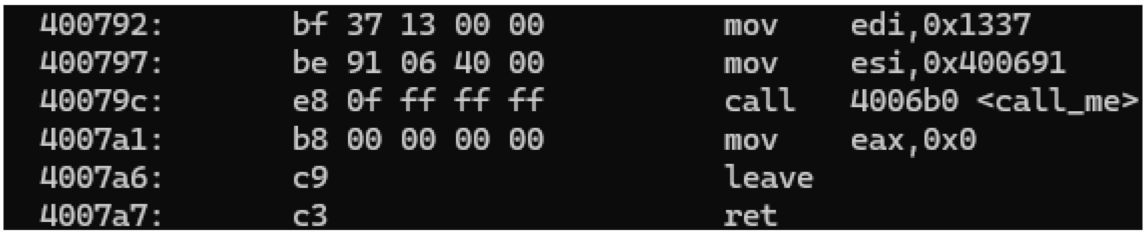
```
void call_me(int param_1,char *param_2)

{
  undefined local_10 [8];

  if ((((param_1 == 0x1337) && (*param_2 == 'C')) && (param_2[1] == 'N')) && (param_2[2] == 'S')) {
    print_flag(local_10,8);
  }
  else {
    puts("Incorrect arguments");
  }
  return;
}
```

4. From Ghidra we can understand that the first parameter is a fixed value: **0x1337** and the second parameter is **a string that has the first three characters: C, N and S**. Thus, I could replace the EDI value to 0x1337 (in little endian 37 13 00 00), but for ESI I should search inside the executable for maybe an address that contains a string starting with C, N and S.



```
mihnea@HOME-PC:/mnt/c/Users/mblot/Desktop/CNS$ readelf -x .text call_me_reloaded | grep CNS
  0x00400690 3d434e53 5f750cbf 60106000 e82ffeff =CNS_u..`.`../..
```

5. And with the picture above we found one such string starting at address 0x00400691 which will be the value of ESI. In little endian: 91 06 40 00;
6. Then we are also calculating the offset for the call which this time would be (~(0x4007a1 – 0x4006b0) = 0xFFFFFF0F which in little endian is 0F FF FF FF)
7. And, adding the E8 opcode we will have a final instruction of E8 0F FF FF FF;
8. That being said, I change the executable using **VIM** to the following aspect:



```
  400792:       bf 37 13 00 00          mov    edi,0x1337
  400797:       be 91 06 40 00          mov    esi,0x400691
  40079c:       e8 0f ff ff ff          call   4006b0 <call_me>
  4007a1:       b8 00 00 00 00          mov    eax,0x0
  4007a6:       c9                      leave
  4007a7:       c3                      ret
```

9. Now, after saving the executable and running it, I found the corresponding flag:

CNS_CTF{Asta-i_tuica_de_Tulcea}