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. In my opinion, this was by far the hardest task in this assignment. First of all, I tried to just run the executable and I ended up with a message and a random number every time.

```
mihnea@HOME-PC:/mnt/c/Users/mblot/Desktop/CNS/Tema1_Personal/call_me$ ./call_me
Your lucky number is 1590394490
```

2. Then, I opened up the executable with Ghidra and I saw nothing wrong with the **main** function.

```
2  undefined8 main(void)
3
4  {
5    uint local_10;
6    int local_c;
7
8    local_10 = 0;
9    local_c = open("/dev/urandom",0);
10   read(local_c,&local_10,4);
11   printf("Your lucky number is %u\n",(ulong)local_10);
12   close(local_c);
13   return 0;
14 }
15
```

3. **Main** function was exactly the one called, there was no **dummy** function as in **call_main** that had to be replaced with another function and also we can notice that the main function takes no input from the user so we cannot exploit any buffer overflow vulnerability as to override the return address for example.

4. That being said, the only choice left is to somehow explicitly call the **call_me function.** Firstly, I tried it dynamically, directly from GDB, but each time I did that, no matter from which execution point, I always got the **"You managed to call me in a wrong way. Try again"** message. That made me think that there would be some specific place from where I have to call this function because, **call_me** is a function that take no parameters as it can be seen in the following picture.

```
2 void call_me(void)
3
4 {
5   undefined local_10 [8];
6
7   decrypt_flag(local_10,8);
8   if (DAT_00601060 == 0x5f534e43) {
9     puts((char *)&DAT_00601060);
10  }
11  else {
12    puts("You managed to call me in a wrong way. Try again");
13  }
14  return;
15 }
16
```

5. So, that being said, I could not be wrong with the way I am calling it because there were no parameters to offer so there is nothing to be wrong. There was no input from the user so nothing to do here as well and also I did not see any way of getting out of executing just the **main** function which made me think that I should add a call to **call_me** inside the already executing **main** function. Thus, searching inside the main function with **objdump** I found a space of exactly 5 bytes where **NOP** instruction resided which was just enough space to change them with a call to **call_me.**

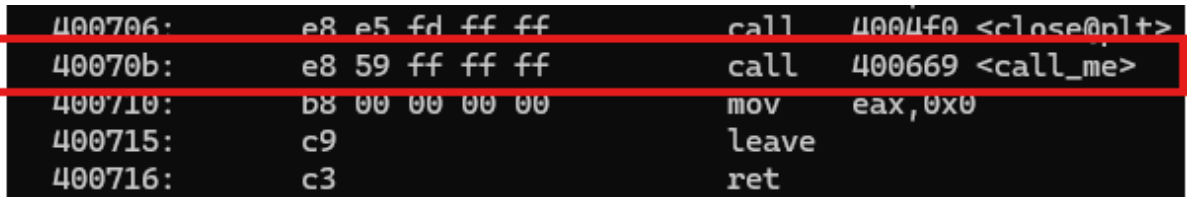
```
400706:       e8 e5 fd ff ff          call   4004f0 <close@plt>
40070b:       90                      nop
40070c:       90                      nop
40070d:       90                      nop
40070e:       90                      nop
40070f:       90                      nop
400710:       b8 00 00 00 00          mov    eax,0x0
400715:       c9                      leave
400716:       c3                      ret
```

6. Inspired by other calls to functions inside **main** I found out that the opcode for “call” instruction is **e8.** The documentation says that e8 is a call with a relative 32-bit offset in a little-endian byte order and that the offset is measured from the following instruction.

7. That being said, so when I am going the replace all these 5 NOPs with my call, the next instruction, or the address that the Instruction Pointer is going to have is **0x400710.** The address of the **call_me** function is **0x400669.**

8. So because the call is relative to the instruction pointer it means that the instruction pointer minus the address where we want to go in the two’s complement is the actual offset of the instruction (~(0x400710 – 0x400669) = **0xFFFFFF59** which would end up in little endian as: 59 FF FF FF, and adding the opcode, then the 5 NOPs should be replace with **e8 59 FF FF FF.**

9. Then I used **VIM** to changed the bytes as mentioned above and ended up with an executable as in the following picture



10. Now, after the new executable was saved an ran, I got the expected flag:

CNS_CTF{You_talk_the_talk_do_you_walk_the_walk}

11. I included in the ZIP the already modified executable that has just to be ran in order to obtain the flag and also a **.gdb_history** with all the failed tries of dynamically calling the **call_me** function.