## DESCRIPTION

*There are some hidden functions in this binary. Try calling them from gdb. One of them will give you the flag.*

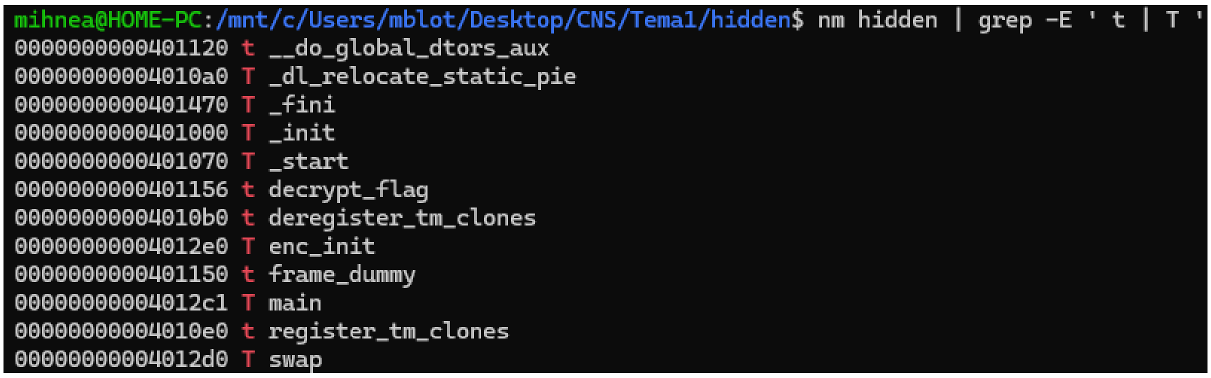
## RESOURCES

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

## APPROACHES

1. The first thing to do was to run the program and I saw that nothing happens. That meant that most likely the main function is a simple **return 0**;

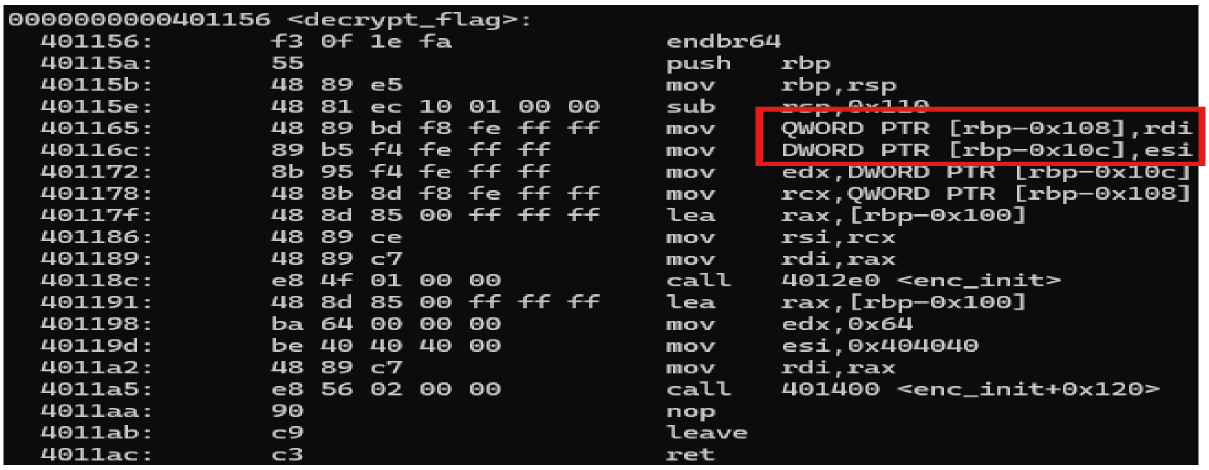
2. The description suggested that there are some hidden functions in the binary so my first approach was to use **nm utility** to find all the symbols in the **.text** section where the code resides with hope that one of the functions seen there is the one that I am searching for:



```
mihnea@HOME-PC:/mnt/c/Users/mblot/Desktop/CNS/Tema1/hidden$ nm hidden | grep -E ' t | T '
0000000000401120 t __do_global_dtors_aux
00000000004010a0 T _dl_relocate_static_pie
0000000000401470 T _fini
0000000000401000 T _init
0000000000401070 T _start
0000000000401156 t decrypt_flag
00000000004010b0 t deregister_tm_clones
00000000004012e0 T enc_init
0000000000401150 t frame_dummy
00000000004012c1 T main
00000000004010e0 t register_tm_clones
00000000004012d0 T swap
```

3. From all of the functions above, the only one that sound to me for this task was **decrypt_flag** as it felt like it was exactly what I was looking for by the name. I was also interested in the **enc_init** function but after dissasembling it, I found out it was actually an enormous more like a mathematical algorithm happening also being called from **decrypt_flag,** so I quickly forgot about it.

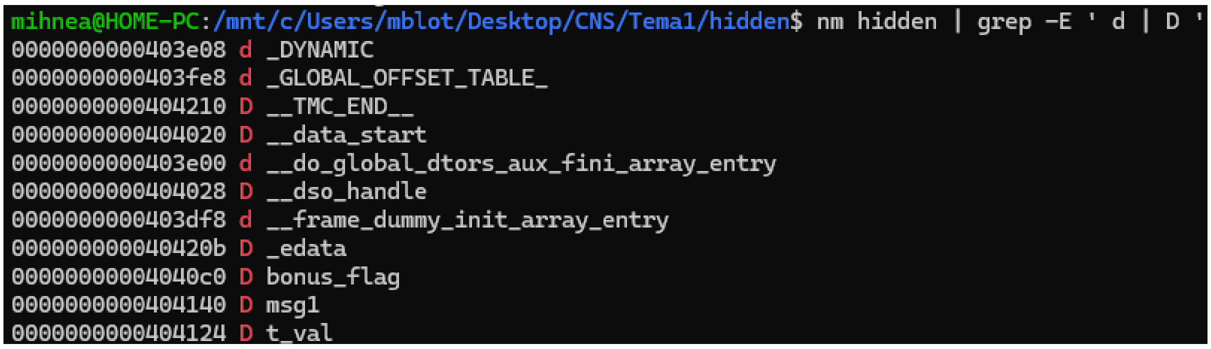
4. The dissassemble of **decrypt_flag** was pretty promising however:



```
0000000000401156 <decrypt_flag>:
  401156:       f3 0f 1e fa             endbr64
  40115a:       55                      push   rbp
  40115b:       48 89 e5                mov    rbp,rsp
  40115e:       48 81 ec 10 01 00 00    sub    rsp,0x110
  401165:       48 89 bd f8 fe ff ff    mov    QWORD PTR [rbp-0x108],rdi
  40116c:       89 b5 f4 fe ff ff       mov    DWORD PTR [rbp-0x10c],esi
  401172:       8b 95 f4 fe ff ff       mov    edx,DWORD PTR [rbp-0x10c]
  401178:       48 8b 8d f8 fe ff ff    mov    rcx,QWORD PTR [rbp-0x108]
  40117f:       48 8d 85 00 ff ff ff    lea    rax,[rbp-0x100]
  401186:       48 89 ce                mov    rsi,rcx
  401189:       48 89 c7                mov    rdi,rax
  40118c:       e8 4f 01 00 00          call   4012e0 <enc_init>
  401191:       48 8d 85 00 ff ff ff    lea    rax,[rbp-0x100]
  401198:       ba 64 00 00 00          mov    edx,0x64
  40119d:       be 40 40 40 00          mov    esi,0x404040
  4011a2:       48 89 c7                mov    rdi,rax
  4011a5:       e8 56 02 00 00          call   401400 <enc_init+0x120>
  4011aa:       90                      nop
  4011ab:       c9                      leave
  4011ac:       c3                      ret
```

5. I understood that **decrypt_flag** is a function that expects 2 parameters, most likely the first one on **8 bytes** and the second one on **4 bytes**. The function seems like it reservers space for a local variable and then uses the two given parameters to call **enc_init**. Then, it uses the local variable and two fixed values **0x404040** and **0x64** to call another function. So, it seems like **decrypt_flag** received p1 and p2 and calls enc_init (local_var, p1, p2) and then (enc_init+0x120)(local_var, 0x404040, 0x64).

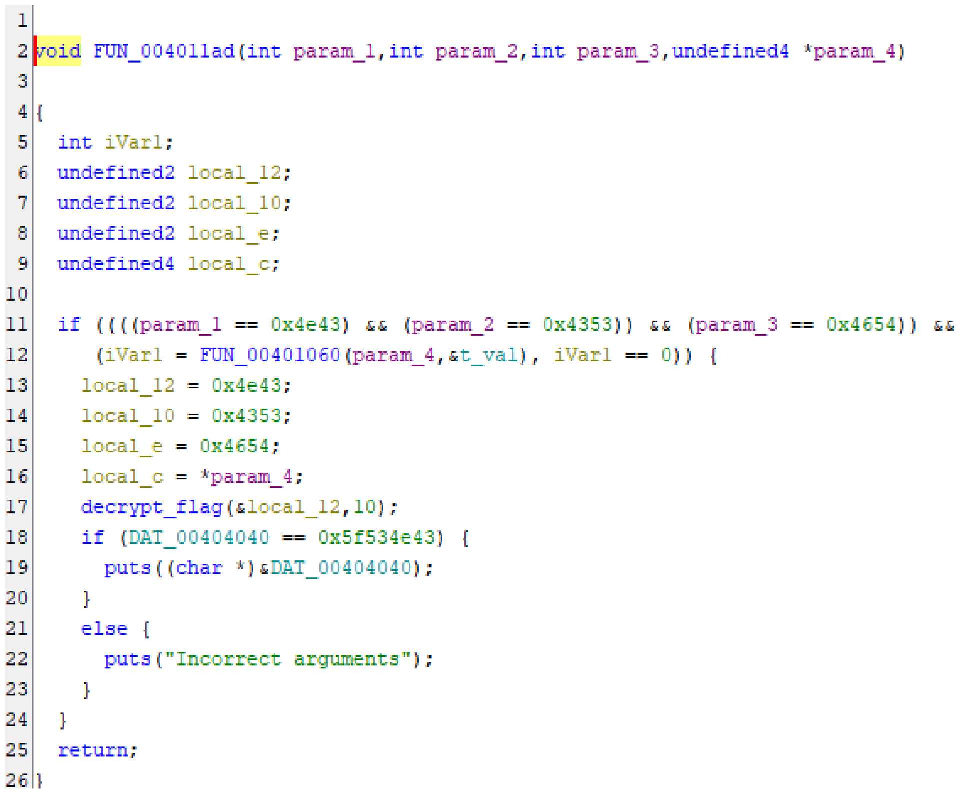
6. Knowing this, I tried to see what parameters I can use to call **decrypt_flag.** For this, I used again **nm utility** to find out what symbols are in the **.data** section this time:



```
mihnea@HOME-PC:/mnt/c/Users/mblot/Desktop/CNS/Tema1/hidden$ nm hidden | grep -E ' d | D '
0000000000403e08 d _DYNAMIC
0000000000403fe8 d _GLOBAL_OFFSET_TABLE_
0000000000404210 D __TMC_END__
0000000000404020 D __data_start
0000000000403e00 d __do_global_dtors_aux_fini_array_entry
0000000000404028 D __dso_handle
0000000000403df8 d __frame_dummy_init_array_entry
000000000040420b D _edata
00000000004040c0 D bonus_flag
0000000000404140 D msg1
0000000000404124 D t_val
```

7. From here, I found out that I can use **bonus_flag**, **msg1** and **t_val** to call **decrypt_flag.** I tried every combination possible of parameters, but none of them worked out for me. I also noticed that, **decrypt_flag** is not calling any **printf** or **puts** so I thought that maybe the flag is not going to be print at the console and knowing that the first parameters has **8 bytes** I tried to maybe send a **stack address** where the function could write and then print the stack to see if I would find the flag there but this did not work as well. I included **.gdb_history** to see all the comands used for trial and error.

8. Then, I had no more ideas and I used **Ghidra to decompile the executable file** and I looked through the existing functions and I found out that there were other functions without names inside the executable such as:



```

void FUN_004011ad(int param_1,int param_2,int param_3,undefined4 *param_4)

{
  int iVar1;
  undefined2 local_12;
  undefined2 local_10;
  undefined2 local_e;
  undefined4 local_c;

  if ((((param_1 == 0x4e43) && (param_2 == 0x4353)) && (param_3 == 0x4654)) &&
     (iVar1 = FUN_00401060(param_4,&t_val), iVar1 == 0)) {
    local_12 = 0x4e43;
    local_10 = 0x4353;
    local_e = 0x4654;
    local_c = *param_4;
    decrypt_flag(&local_12,10);
    if (DAT_00404040 == 0x5f534e43) {
      puts((char *)&DAT_00404040);
    }
    else {
      puts("Incorrect arguments");
    }
  }
  return;
}
```

9. That being said, I have seen that this function calls **decrypt_flag** which I was already trying to call. This function receives 4 parameters in the following order: 0x4e43, 0x4353, 0x4654. The last parameter is sent to the function at address **0x00401060** which is a **strcmp.** The other parameter is **t_val,** which means that the 4th parameter has to be **t_val** as well.

10. The discovery above brings us to the final call and solution of the task:

**call ((void (*)()) 0x4011ad)(0x4e43, 0x4353, 0x4654, (char *) &t_val)**

11. The call above retrieved the flag:

**CNS_CTF{The_zombies_were_having_fun_the_party_had_just_begun}**