DESCRIPTION

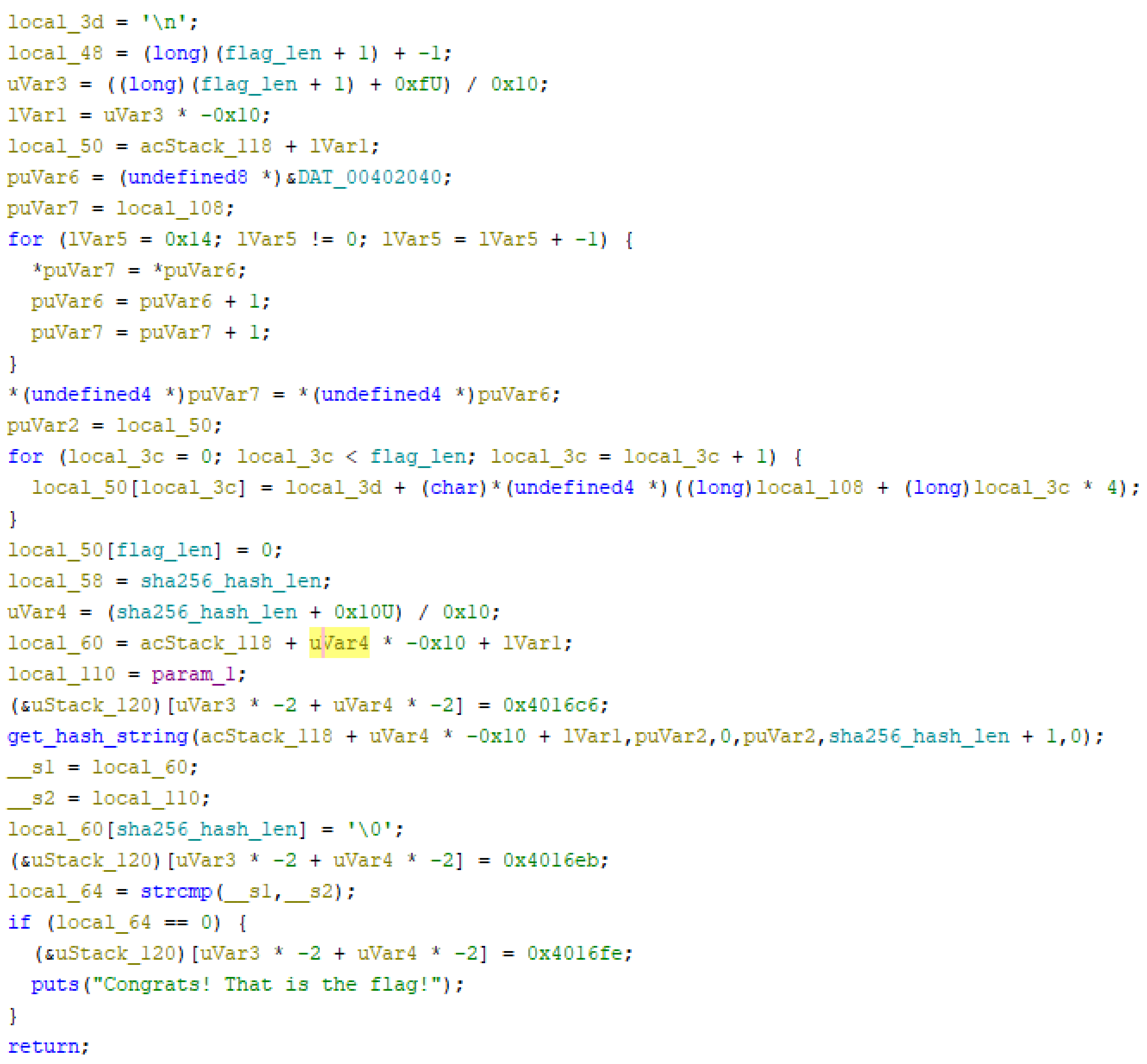
Find me, call me properly, and I'll give you the flag.

RESOURCES

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

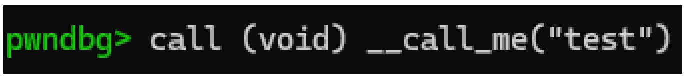
APPROACHES

1. Here the first approach was to again directly run the executable give it an input but I got a SHA Sum and the message **You're digging the wrong hole**;
2. Thus, I had no idea and decompiled the executable using Ghidra. Here I searched for functions that I could call and I found **__call_me(char *param1)**. It's implementation can be seen below:



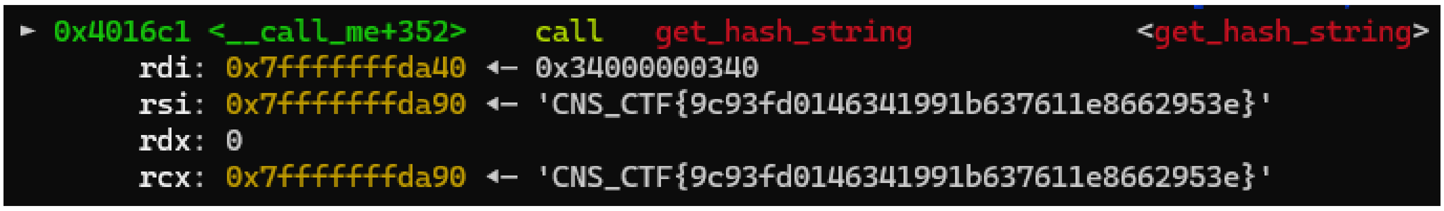
```
local_3d = '\n';
local_48 = (long)(flag_len + 1) + -1;
uVar3 = ((long)(flag_len + 1) + 0xfU) / 0x10;
lVar1 = uVar3 * -0x10;
local_50 = acStack_118 + lVar1;
puVar6 = (undefined8 *)&DAT_00402040;
puVar7 = local_108;
for (lVar5 = 0x14; lVar5 != 0; lVar5 = lVar5 + -1) {
  *puVar7 = *puVar6;
  puVar6 = puVar6 + 1;
  puVar7 = puVar7 + 1;
}
*(undefined4 *)puVar7 = *(undefined4 *)puVar6;
puVar2 = local_50;
for (local_3c = 0; local_3c < flag_len; local_3c = local_3c + 1) {
  local_50[local_3c] = local_3d + (char)*(undefined4 *)((long)local_108 + (long)local_3c * 4);
}
local_50[flag_len] = 0;
local_58 = sha256_hash_len;
uVar4 = (sha256_hash_len + 0x10U) / 0x10;
local_60 = acStack_118 + uVar4 * -0x10 + lVar1;
local_110 = param_1;
(&uStack_120)[uVar3 * -2 + uVar4 * -2] = 0x4016c6;
get_hash_string(acStack_118 + uVar4 * -0x10 + lVar1,puVar2,0,puVar2,sha256_hash_len + 1,0);
__s1 = local_60;
__s2 = local_110;
local_60[sha256_hash_len] = '\0';
(&uStack_120)[uVar3 * -2 + uVar4 * -2] = 0x4016eb;
local_64 = strcmp(__s1,__s2);
if (local_64 == 0) {
  (&uStack_120)[uVar3 * -2 + uVar4 * -2] = 0x4016fe;
  puts("Congrats! That is the flag!");
}
return;
```

3. Here, having the implementation, I started at the end and I saw a message **„Congrats! That is the flag!"** which would have been printed if a local variable would have been 0. This variable would have been 0 if two strings would have been the same.
4. The strings that should be the same are: **local_110** which we can see above that is exactly the string that we are sending to this function and **local_60** which we can see that it is exactly one of the parameters sent to **get_hash_string** function.
5. Thus, I direcly started the program with gdb, put a breakpoint at **get_hash_string** and called the **__call_me()** function with a random argument and then I stopped at the call to **get_hash_string** to see the input parameters.
6. You can start the gdb with the following command: **gdb --args ./call_me test**



```
pwndbg> call (void) __call_me("test")
```

7. And this would be a valid call to the function **__call_me()**
8. Then, stopping at the correct address, before the call of the **get_hash_string** function, gdb returns the following output:



```
► 0x4016c1 <__call_me+352>    call   get_hash_string             <get_hash_string>
       rdi: 0x7fffffffda40 ◂— 0x340000000340
       rsi: 0x7fffffffda90 ◂— 'CNS_CTF{9c93fd0146341991b637611e8662953e}'
       rdx: 0
       rcx: 0x7fffffffda90 ◂— 'CNS_CTF{9c93fd0146341991b637611e8662953e}'
```

9. From here, we can extract the flag as being:

CNS_CTF{9c93fd0146341991b637611e8662953e}