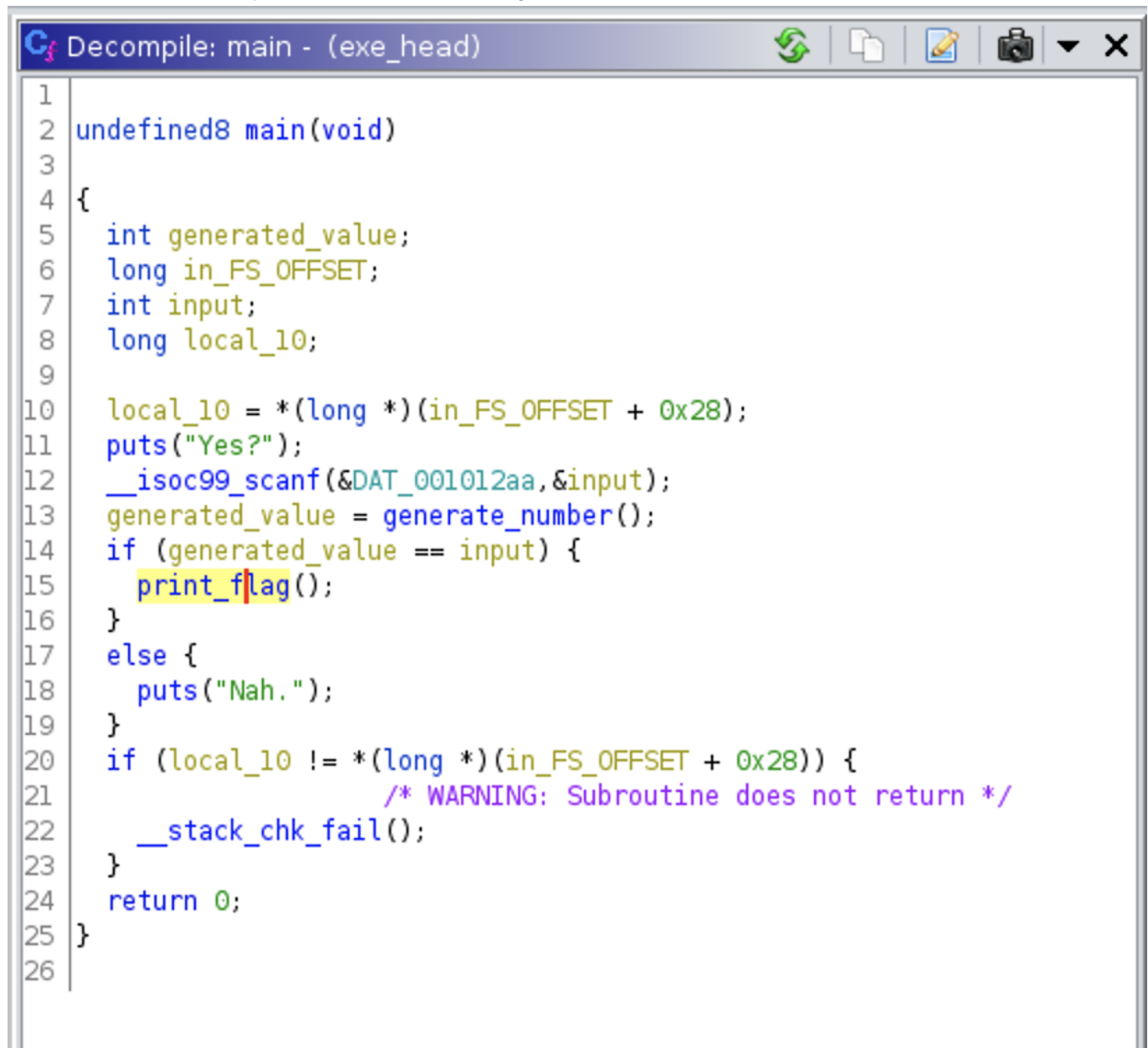


Writeup

Author: Mihhail Sokolov (msokolov) SCIPER: 338760

My head explodes challenge

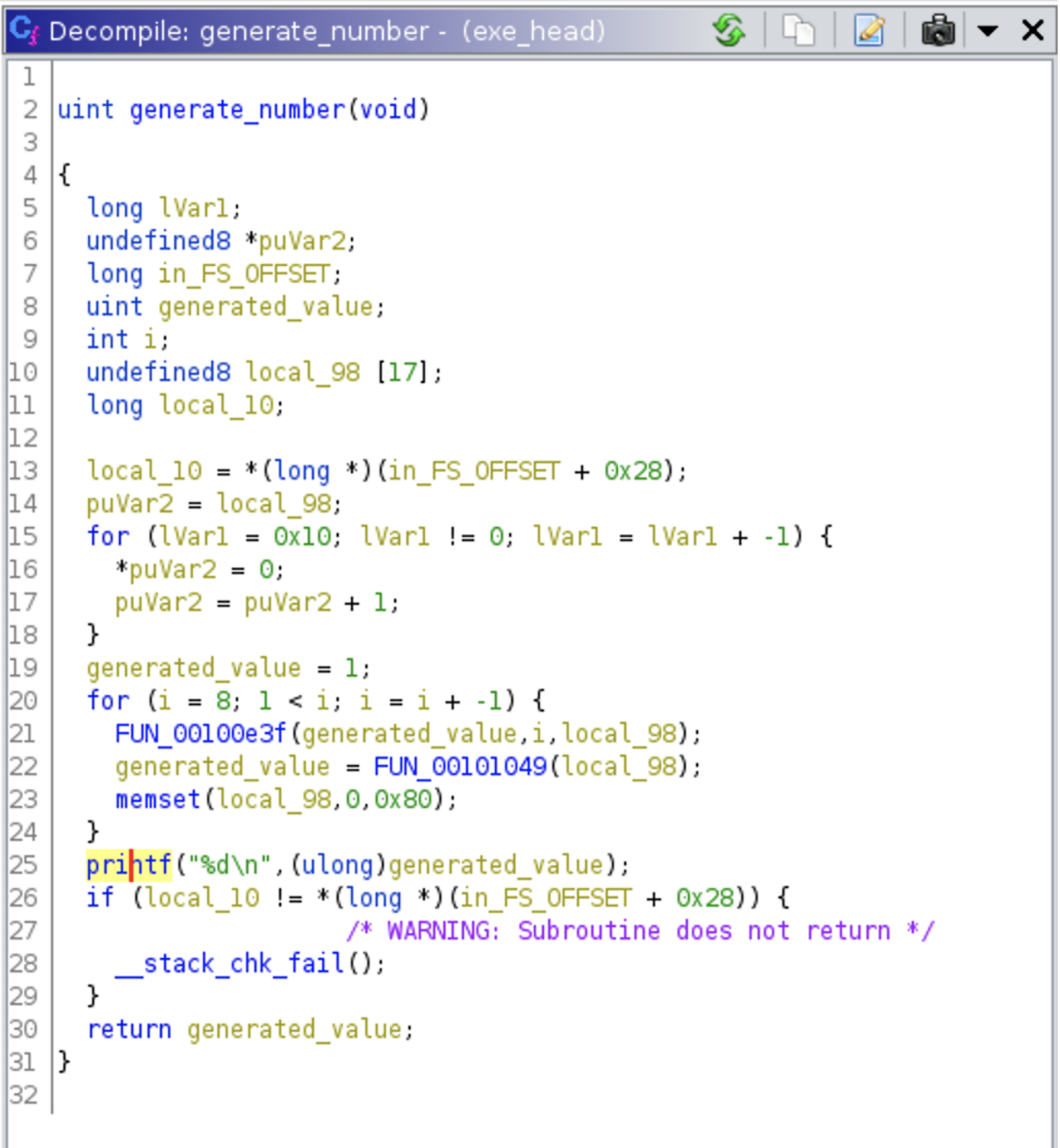
After opening the executable in Ghidra, looking at the decompiled functions, and some renaming of the functions and variables, we arrive at the following "main" code:



```
1
2 undefined8 main(void)
3
4 {
5     int generated_value;
6     long in_FS_OFFSET;
7     int input;
8     long local_10;
9
10    local_10 = *(long *)(in_FS_OFFSET + 0x28);
11    puts("Yes?");
12    __isoc99_scanf(&DAT_001012aa,&input);
13    generated_value = generate_number();
14    if (generated_value == input) {
15        print_flag();
16    }
17    else {
18        puts("Nah.");
19    }
20    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
21        /* WARNING: Subroutine does not return */
22        __stack_chk_fail();
23    }
24    return 0;
25 }
26
```

As can be seen from the code, in order to get the flag printed we need to enter the value and it should match with the value generated by another function.

So let's see how that value is being generated. After a little bit of renaming we arrive at the following code:

A screenshot of a decompiler window titled "Decompile: generate_number - (exe_head)". The window shows the decompiled C code for a function named "generate_number". The code is as follows:

```
1
2 uint generate_number(void)
3
4 {
5     long lVar1;
6     undefined8 *puVar2;
7     long in_FS_OFFSET;
8     uint generated_value;
9     int i;
10    undefined8 local_98 [17];
11    long local_10;
12
13    local_10 = *(long *)(in_FS_OFFSET + 0x28);
14    puVar2 = local_98;
15    for (lVar1 = 0x10; lVar1 != 0; lVar1 = lVar1 + -1) {
16        *puVar2 = 0;
17        puVar2 = puVar2 + 1;
18    }
19    generated_value = 1;
20    for (i = 8; 1 < i; i = i + -1) {
21        FUN_00100e3f(generated_value,i,local_98);
22        generated_value = FUN_00101049(local_98);
23        memset(local_98,0,0x80);
24    }
25    printf("%d\n", (ulong)generated_value);
26    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
27        /* WARNING: Subroutine does not return */
28        __stack_chk_fail();
29    }
30    return generated_value;
31 }
32
```

Luckily for us the generated value is being printed. We can also see that other functions are also being called inside the loop operating on the `generated_value`, however if we go down the function call chain, we see that there is no random variables. This means that `generated_value` is deterministic and always has the same value. In the end, all we need to do to get the flag is to run the program twice: first time to enter wrong value and find out the correct generated value and second time to actually enter the correct

generated value and get the flag. This procedure can be seen in the screenshot below.

```
mihhail@Mihhails-MacBook-Pro ~ % nc hexhive005.iccluster.epfl.ch 9012
[You got logged in to a container which runs the challenge. You have 5 minutes.]
Yes?
1
40320
Nah.
mihhail@Mihhails-MacBook-Pro ~ % nc hexhive005.iccluster.epfl.ch 9012
[You got logged in to a container which runs the challenge. You have 5 minutes.]
Yes?
40320
40320
This is a good value...
SoftSec{cbbbed4a5c77b0057f3674b257090694c78018543e0c451a1251572df8dafdc35}
```