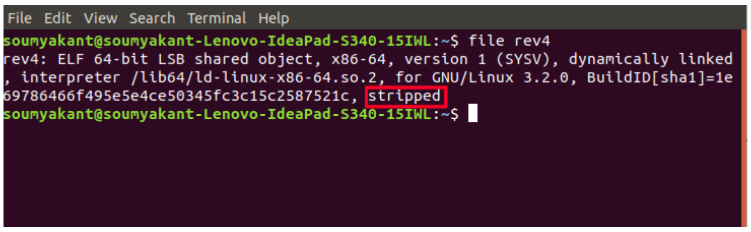
de.ci.phe.red LAB – CTF 2

rev4 – writeup

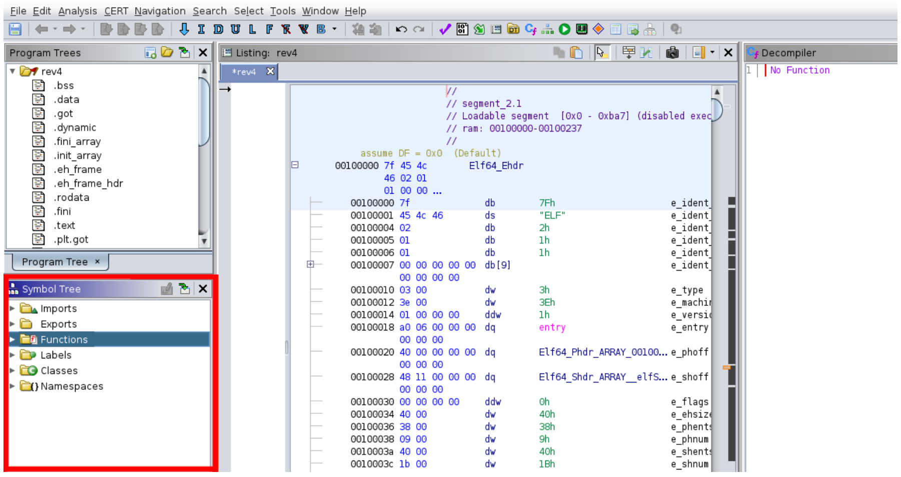
Open terminal and use file command to see the file type of rev4:



```
soumyakant@soumyakant-Lenovo-IdeaPad-S340-15IWL:~$ file rev4
rev4: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked
, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=1e
69786466f495e5e4ce50345fc3c15c2587521c, stripped
soumyakant@soumyakant-Lenovo-IdeaPad-S340-15IWL:~$
```

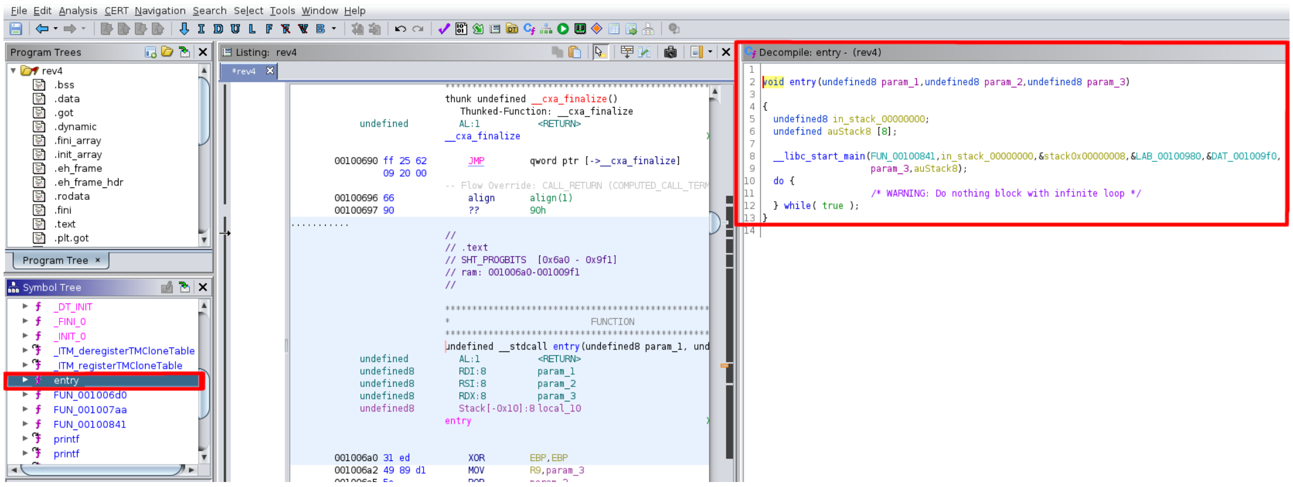
We can see that rev4 is a stripped binary, i.e. it does not contain symbols such as ‘main’.

Now, open rev4 in Ghidra and go to **Functions** in the **Symbol Tree**:



```
                          //
                          // segment_2.1
                          // Loadable segment  [0x0 - 0xba7] (disabled exec
                          // ram: 00100000-00100237
                          //
                  assume DF = 0x0  (Default)
      00100000 7f 45 4c         Elf64_Ehdr
               46 02 01
               01 00 00 ...
         00100000 7f             db       7Fh                    e_ident_
         00100001 45 4c 46       ds       "ELF"                  e_ident_
         00100004 02             db       2h                     e_ident_
         00100005 01             db       1h                     e_ident_
         00100006 01             db       1h                     e_ident_
         00100007 00 00 00 00 00 db[9]                           e_ident_
                  00 00 00 00
         00100010 03 00          dw       3h                     e_type
         00100012 3e 00          dw       3Eh                    e_machir
         00100014 01 00 00 00    ddw      1h                     e_versic
         00100018 a0 06 00 00 00 dq       entry                  e_entry
                  00 00 00
         00100020 40 00 00 00 00 dq       Elf64_Phdr_ARRAY_00100... e_phoff
                  00 00 00
         00100028 48 11 00 00 00 dq       Elf64_Shdr_ARRAY__elfS... e_shoff
                  00 00 00
         00100030 00 00 00 00    ddw      0h                     e_flags
         00100034 40 00          dw       40h                    e_ehsize
         00100036 38 00          dw       38h                    e_phents
         00100038 09 00          dw       9h                     e_phnum
         0010003a 40 00          dw       40h                    e_shents
         0010003c 1b 00          dw       1Bh                    e_shnum
```

From **Functions**, select **entry**, this will give the decompiled code for **entry**:

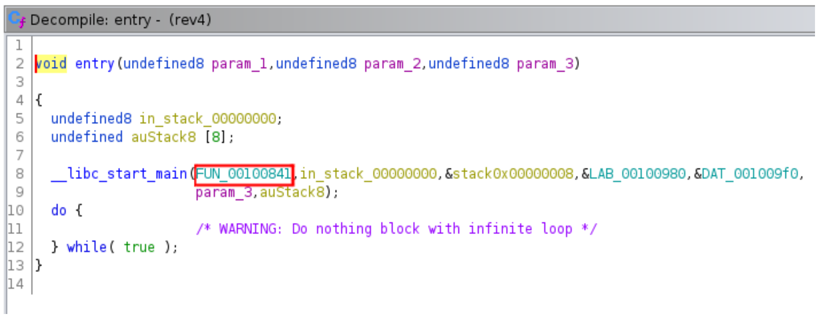


```
                          ****************************************************
                          thunk undefined __cxa_finalize()
                            Thunked-Function: __cxa_finalize
        undefined           AL:1          <RETURN>
                          __cxa_finalize
      00100690 ff 25 62       JMP       qword ptr [->__cxa_finalize]
               09 20 00
                          -- Flow Override: CALL_RETURN (COMPUTED_CALL_TERM
      00100696 66             align     align(1)
      00100697 90             ??        90h
...........
                          //
                          // .text
                          // SHT_PROGBITS  [0x6a0 - 0x9f1]
                          // ram: 001006a0-001009f1
                          //

                          ****************************************************
                          *                         FUNCTION
                          ****************************************************
                          undefined __stdcall entry(undefined8 param_1, und
        undefined           AL:1          <RETURN>
        undefined8          RDI:8         param_1
        undefined8          RSI:8         param_2
        undefined8          RDX:8         param_3
        undefined8          Stack[-0x10]:8 local_10
                          entry
      001006a0 31 ed          XOR       EBP,EBP
      001006a2 49 89 d1       MOV       R9,param_3
```

Decompile: entry - (rev4)

```
void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)

{
  undefined8 in_stack_00000000;
  undefined auStack8 [8];

  __libc_start_main(FUN_00100841,in_stack_00000000,&stack0x00000008,&LAB_00100980,&DAT_001009f0,
                    param_3,auStack8);
  do {
                    /* WARNING: Do nothing block with infinite loop */
  } while( true );
}
```

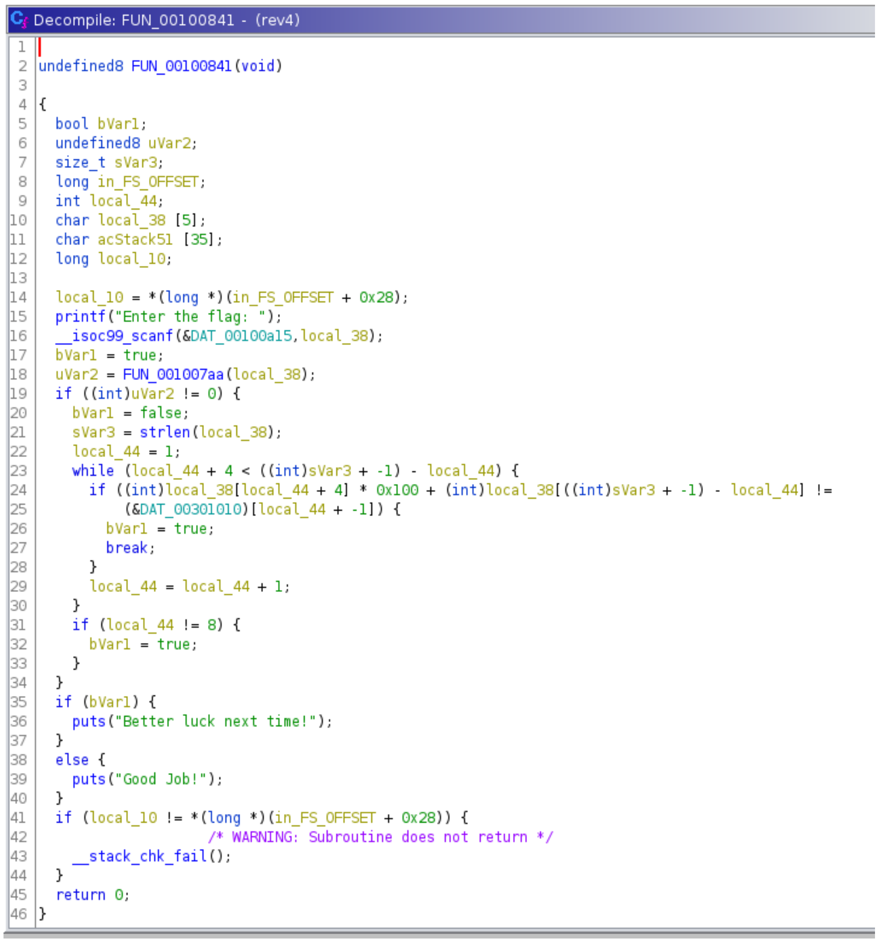
We now select the first parameter of **__libc_start_main()** :



Decompile: entry - (rev4)

```

void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)

{
  undefined8 in_stack_00000000;
  undefined auStack8 [8];

  __libc_start_main(FUN_00100841,in_stack_00000000,&stack0x00000008,&LAB_00100980,&DAT_001009f0,
                    param_3,auStack8);
  do {
                    /* WARNING: Do nothing block with infinite loop */
  } while( true );
}

```

This will lead us to the decompiled code for the **main** function:



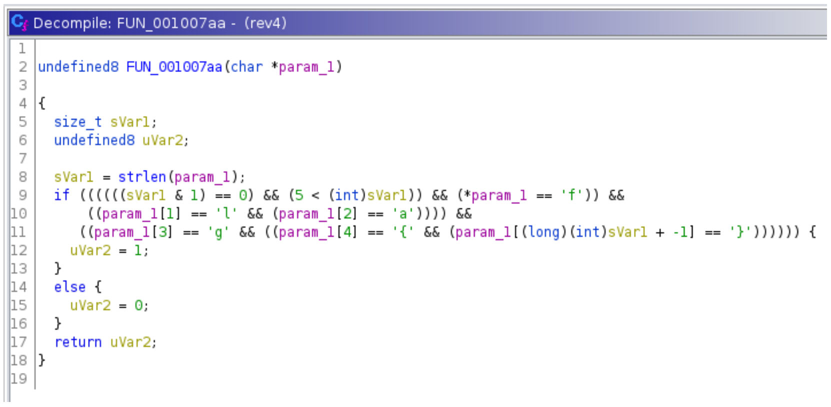
Decompile: FUN_00100841 - (rev4)

```
 1 
 2 undefined8 FUN_00100841(void)
 3 
 4 {
 5   bool bVar1;
 6   undefined8 uVar2;
 7   size_t sVar3;
 8   long in_FS_OFFSET;
 9   int local_44;
10   char local_38 [5];
11   char acStack51 [35];
12   long local_10;
13 
14   local_10 = *(long *)(in_FS_OFFSET + 0x28);
15   printf("Enter the flag: ");
16   __isoc99_scanf(&DAT_00100a15,local_38);
17   bVar1 = true;
18   uVar2 = FUN_001007aa(local_38);
19   if ((int)uVar2 != 0) {
20     bVar1 = false;
21     sVar3 = strlen(local_38);
22     local_44 = 1;
23     while (local_44 + 4 < ((int)sVar3 + -1) - local_44) {
24       if ((int)local_38[local_44 + 4] * 0x100 + (int)local_38[((int)sVar3 + -1) - local_44] !=
25           (&DAT_00301010)[local_44 + -1]) {
26         bVar1 = true;
27         break;
28       }
29       local_44 = local_44 + 1;
30     }
31     if (local_44 != 8) {
32       bVar1 = true;
33     }
34   }
35   if (bVar1) {
36     puts("Better luck next time!");
37   }
38   else {
39     puts("Good Job!");
40   }
41   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
42                     /* WARNING: Subroutine does not return */
43     __stack_chk_fail();
44   }
45   return 0;
46 }
```

Here line 15 prints "Enter the flag: " using printf() and line 16 takes the flag as input from the user using scanf().

Line 18 makes a call to the function **FUN_001007aa()** by passing the parameter **local_38** which contains the input string given by the user.

Double clicking on **FUN_001007aa()** shows the definition this function which is as follows:

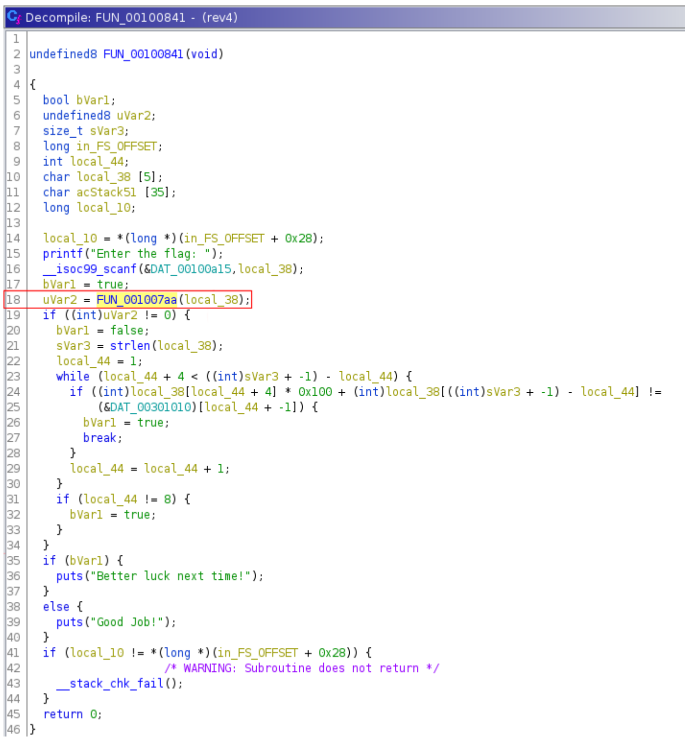


Decompile: FUN_001007aa - (rev4)

```c

undefined8 FUN_001007aa(char *param_1)

{
  size_t sVar1;
  undefined8 uVar2;

  sVar1 = strlen(param_1);
  if ((((((sVar1 & 1) == 0) && (5 < (int)sVar1)) && (*param_1 == 'f')) &&
      ((param_1[1] == 'l' && (param_1[2] == 'a')))) &&
     ((param_1[3] == 'g' && ((param_1[4] == '{' && (param_1[(long)(int)sVar1 + -1] == '}')))))) {
    uVar2 = 1;
  }
  else {
    uVar2 = 0;
  }
  return uVar2;
}

```

We can see that the function **FUN_001007aa()** checks if the length of the input string is even and greater than 5 and if it is of the form **flag{...}**. If all these conditions are satisfied, then the function returns 1. Otherwise it returns 0.

The value retutned by **FUN_001007aa()** is stored in a variable **uVar2** :



Decompile: FUN_00100841 - (rev4)

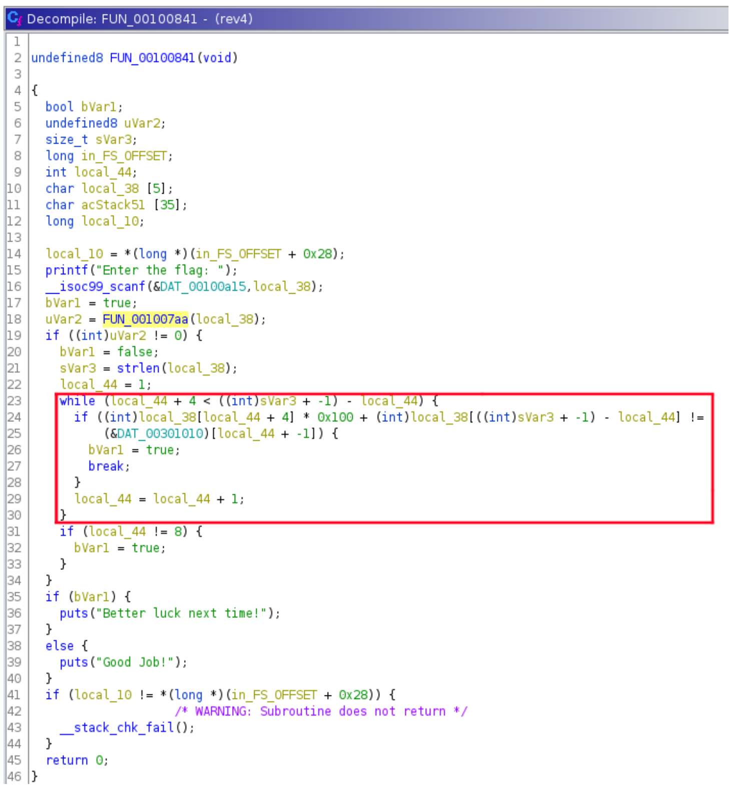
```c
1
2  undefined8 FUN_00100841(void)
3
4  {
5    bool bVar1;
6    undefined8 uVar2;
7    size_t sVar3;
8    long in_FS_OFFSET;
9    int local_44;
10   char local_38 [5];
11   char acStack51 [35];
12   long local_10;
13
14   local_10 = *(long *)(in_FS_OFFSET + 0x28);
15   printf("Enter the flag: ");
16   __isoc99_scanf(&DAT_00100a15,local_38);
17   bVar1 = true;
18   uVar2 = FUN_001007aa(local_38);
19   if ((int)uVar2 != 0) {
20     bVar1 = false;
21     sVar3 = strlen(local_38);
22     local_44 = 1;
23     while (local_44 + 4 < ((int)sVar3 + -1) - local_44) {
24       if ((int)local_38[local_44 + 4] * 0x100 + (int)local_38[((int)sVar3 + -1) - local_44] !=
25           (&DAT_00301010)[local_44 + -1]) {
26         bVar1 = true;
27         break;
28       }
29       local_44 = local_44 + 1;
30     }
31     if (local_44 != 8) {
32       bVar1 = true;
33     }
34   }
35   if (bVar1) {
36     puts("Better luck next time!");
37   }
38   else {
39     puts("Good Job!");
40   }
41   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
42                     /* WARNING: Subroutine does not return */
43     __stack_chk_fail();
44   }
45   return 0;
46 }
```

The if statement in line 19 checks if the value returned by **FUN_001007aa()** is not equal to **0**. In case the value returned by **FUN_001007aa()** is **0**, then the if condition becomes false and when we go to line 35, the if condition present there becomes true since **bvar1** was initialized to **true**. Hence the program outputs "**Better luck next time!**".

Hence, **FUN_001007aa()** must output **1** for the correct flag and the **if** condition in line 19 must become **true**.

Line 21 calculates the length of the input string using strlen() function and stores it in a variable **sVar3**.

Now, we focus on the while loop in line 23:


Decompile: FUN_00100841 - (rev4)

```c
 1
 2 undefined8 FUN_00100841(void)
 3
 4 {
 5   bool bVar1;
 6   undefined8 uVar2;
 7   size_t sVar3;
 8   long in_FS_OFFSET;
 9   int local_44;
10   char local_38 [5];
11   char acStack51 [35];
12   long local_10;
13
14   local_10 = *(long *)(in_FS_OFFSET + 0x28);
15   printf("Enter the flag: ");
16   __isoc99_scanf(&DAT_00100a15,local_38);
17   bVar1 = true;
18   uVar2 = FUN_001007aa(local_38);
19   if ((int)uVar2 != 0) {
20     bVar1 = false;
21     sVar3 = strlen(local_38);
22     local_44 = 1;
23     while (local_44 + 4 < ((int)sVar3 + -1) - local_44) {
24       if ((int)local_38[local_44 + 4] * 0x100 + (int)local_38[((int)sVar3 + -1) - local_44] !=
25           (&DAT_00301010)[local_44 + -1]) {
26         bVar1 = true;
27         break;
28       }
29       local_44 = local_44 + 1;
30     }
31     if (local_44 != 8) {
32       bVar1 = true;
33     }
34   }
35   if (bVar1) {
36     puts("Better luck next time!");
37   }
38   else {
39     puts("Good Job!");
40   }
41   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
42                     /* WARNING: Subroutine does not return */
43     __stack_chk_fail();
44   }
45   return 0;
46 }
```

Number of iterations of the while loop:

In ideal scenario i.e. for the correct flag, the while loop will run as long as
local_44 + 4 < sVar3 -1 – local_44 i.e. **2*local_44 < sVar3 – 5**

Also from line 31, we can infer that the value of **local_44** must be equal to **8** after the end of the while loop. Otherwise the if condition in line 35 will be true and the code will print “Better luck next itme!”.

Hence the first time when the condition of the while loop will be **false**, will occur when **local_44 = 8**

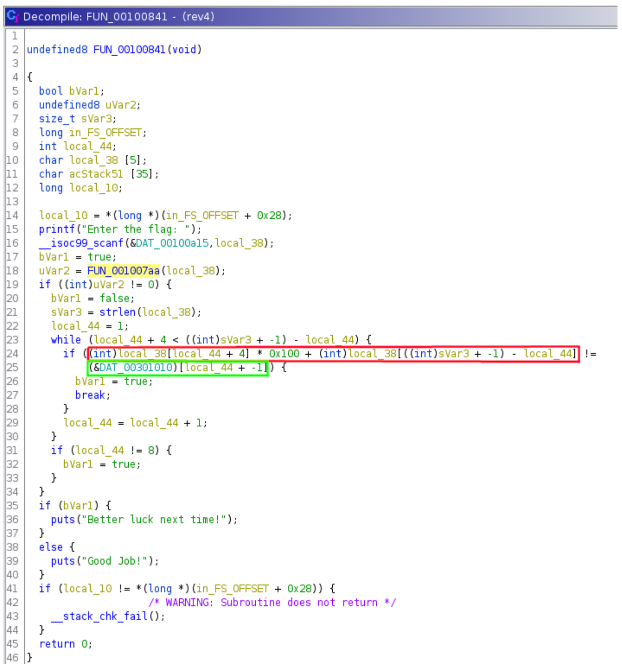
The condition **2 * local_44 < sVar3 – 5** will become **2 * 8 < sVar3 – 5** i.e. **21 < sVar3**

Since this condition has to be false, therefore **sVar3** must be just less than or equal to 21. Since **sVar3** is even as checked earlier, therefore **sVar3 = 20** for correct flag. [It cannot be less than 20 because in that case **local_44** will not become **8** at the end of the loop]

Now, since **local_44** was initialized to **1** before the start of the loop, it is **8** at the end of the loop and it is incremented by **1** in each iteration.

Therefore the while loop will run for **8 - 1 = 7 iterations** for the correct flag.

Now, let’s focus on the if statement inside the while loop:



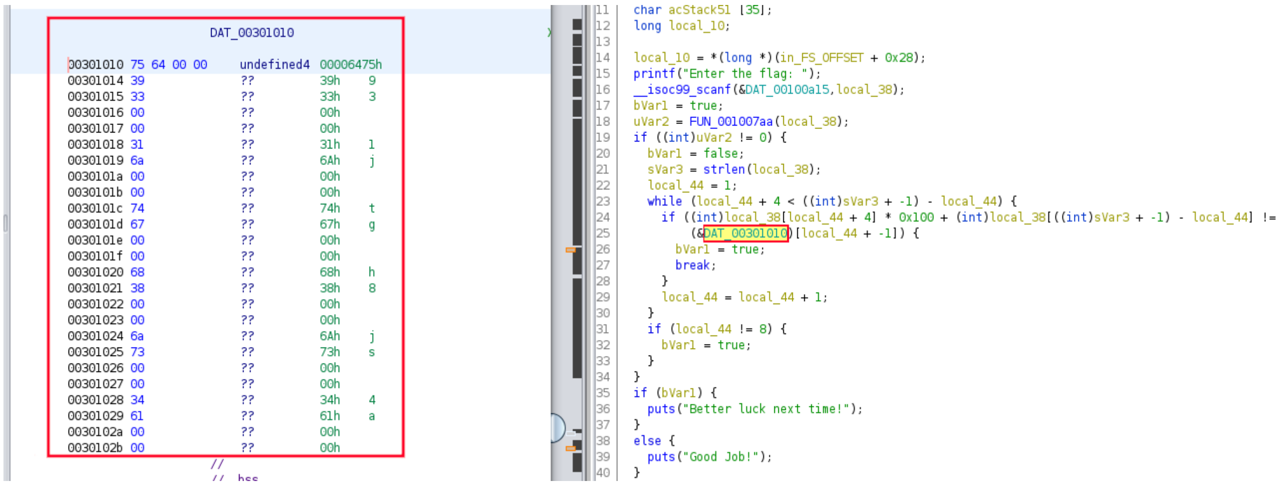
```
Decompile: FUN_00100841 - (rev4)

undefined8 FUN_00100841(void)

{
  bool bVar1;
  undefined8 uVar2;
  size_t sVar3;
  long in_FS_OFFSET;
  int local_44;
  char local_38 [5];
  char acStack51 [35];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  printf("Enter the flag: ");
  __isoc99_scanf(&DAT_00100a15,local_38);
  bVar1 = true;
  uVar2 = FUN_001007aa(local_38);
  if ((int)uVar2 != 0) {
    bVar1 = false;
    sVar3 = strlen(local_38);
    local_44 = 1;
    while (local_44 + 4 < ((int)sVar3 + -1) - local_44) {
      if ((int)local_38[local_44 + 4] * 0x100 + (int)local_38[((int)sVar3 + -1) - local_44] !=
          (&DAT_00301010)[local_44 + -1]) {
        bVar1 = true;
        break;
      }
      local_44 = local_44 + 1;
    }
    if (local_44 != 8) {
      bVar1 = true;
    }
  }
  if (bVar1) {
    puts("Better luck next time!");
  }
  else {
    puts("Good Job!");
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

It forms an integer (say **x** marked with **red** colour in the above figure) and checks if it is equal to some integer (say **y** marked with **green** colour in the above figure) stored in the memory.

Let **s** denote the **input_string** given by the user, then interger **x** is formed as:

x = s [local_44 + 4] * 0x100 + s [length_of_input_string – 1 – local_44]

To find the values taken by integer **y**, double click on **DAT_00301010** :



```
                     DAT_00301010

00301010 75 64 00 00     undefined4 00006475h
00301014 39              ??         39h     9
00301015 33              ??         33h     3
00301016 00              ??         00h
00301017 00              ??         00h
00301018 31              ??         31h     1
00301019 6a              ??         6Ah     j
0030101a 00              ??         00h
0030101b 00              ??         00h
0030101c 74              ??         74h     t
0030101d 67              ??         67h     g
0030101e 00              ??         00h
0030101f 00              ??         00h
00301020 68              ??         68h     h
00301021 38              ??         38h     8
00301022 00              ??         00h
00301023 00              ??         00h
00301024 6a              ??         6Ah     j
00301025 73              ??         73h     s
00301026 00              ??         00h
00301027 00              ??         00h
00301028 34              ??         34h     4
00301029 61              ??         61h     a
0030102a 00              ??         00h
0030102b 00              ??         00h
                     //
                     //  bss
```

```
  char acStack51 [35];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  printf("Enter the flag: ");
  __isoc99_scanf(&DAT_00100a15,local_38);
  bVar1 = true;
  uVar2 = FUN_001007aa(local_38);
  if ((int)uVar2 != 0) {
    bVar1 = false;
    sVar3 = strlen(local_38);
    local_44 = 1;
    while (local_44 + 4 < ((int)sVar3 + -1) - local_44) {
      if ((int)local_38[local_44 + 4] * 0x100 + (int)local_38[((int)sVar3 + -1) - local_44] !=
          (&DAT_00301010)[local_44 + -1]) {
        bVar1 = true;
        break;
      }
      local_44 = local_44 + 1;
    }
    if (local_44 != 8) {
      bVar1 = true;
    }
  }
  if (bVar1) {
    puts("Better luck next time!");
  }
  else {
    puts("Good Job!");
  }
```

(&DAT_00301010)[local_44 – 1] denotes the element at index **local_44 – 1** of the integer array at **&DAT_00301010**.

Here an integer takes 4 bytes in memory.

Hence looking at the memory representation (which is in little endian) in the left hand side of the above picture, we can construct the entire array as:

0x6475	0x3339	0x6A31	0x6774	0x3868	0x736A	0x6134
0	1	2	3	4	5	6

Hence the values of **x** and **y** over all the iterations can be written as:

(Here we are denoting the input string as s)

Iteration	1	2	3	4	5	6	7
x	s[5]*0x100 + s[18]	s[6]*0x100 + s[17]	s[7]*0x100 + s[16]	s[8]*0x100 + s[15]	s[9]*0x100 + s[14]	s[10]*0x100 + s[13]	s[11]*0x100 + s[12]
y	0x6475	0x3339	0x6A31	0x6774	0x3868	0x736A	0x6134

Hence the ASCII values corressponding to different indices of Input string are:

	ASCII value	Character
s[5]	0x64	d
s[18]	0x75	u
s[6]	0x33	3
s[17]	0x39	9
s[7]	0x6A	j
s[16]	0X31	1
s[8]	0X67	g
s[15]	0X74	t
s[9]	0X38	8
s[14]	0X68	h
s[10]	0X73	s
s[13]	0X6A	j
s[11]	0X61	a
s[12]	0X34	4

Therefore the correct value of s is **flag{d3jg8sa4jht19u}** which is our flag.