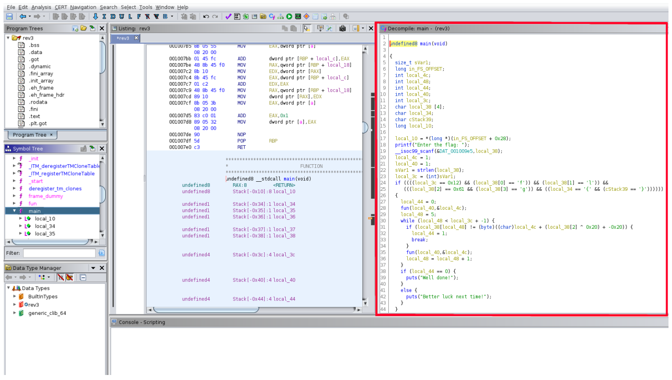
de.ci.phe.red LAB – CTF 2

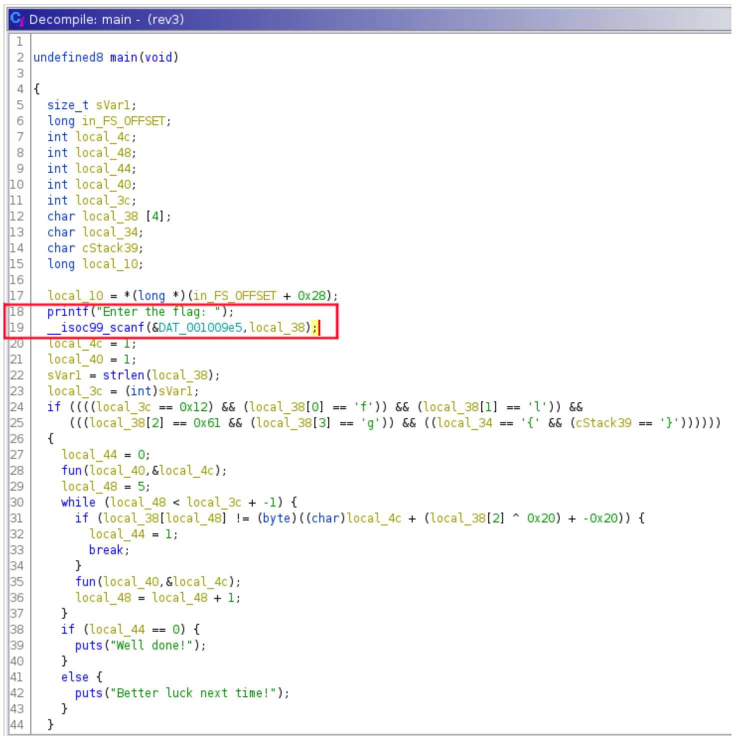
rev3 – writeup

Open the file rev3 in ghidra to decompile its main function.

The decompiled code of the main function looks as follows:



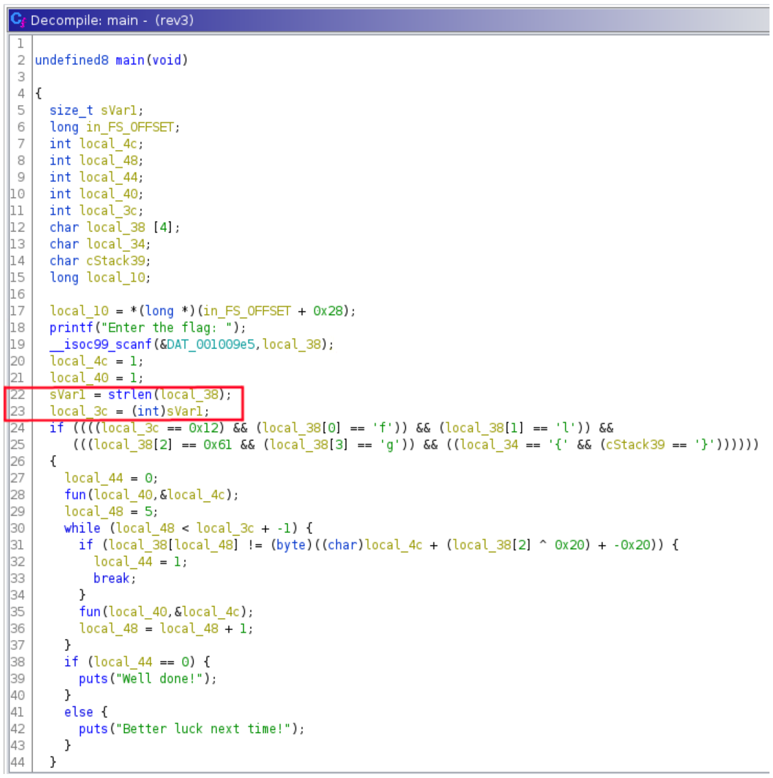
Now, line 18 prints the statement “Enter the flag: ” using printf() and line 19 takes the flag as input from the user using scanf() as shown below:



Decompile: main - (rev3)

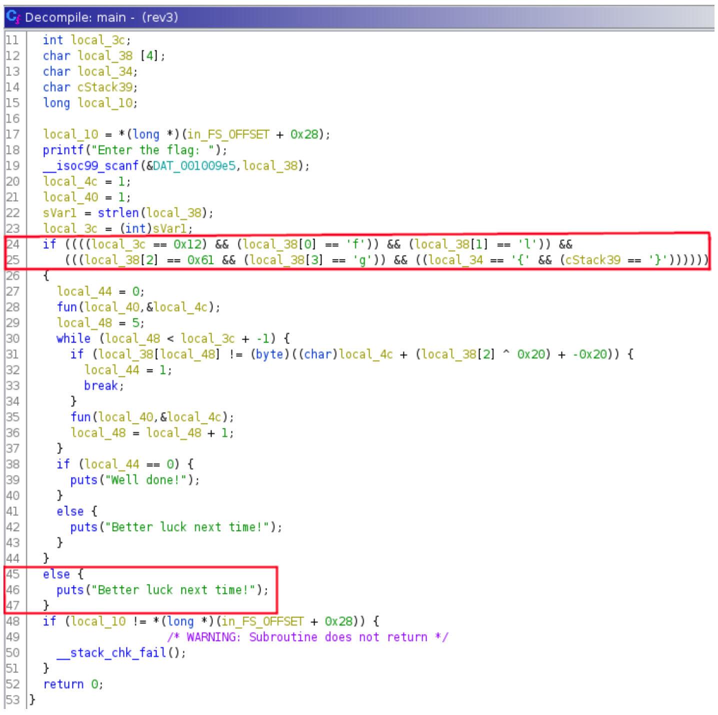
```
 1
 2  undefined8 main(void)
 3
 4  {
 5    size_t sVar1;
 6    long in_FS_OFFSET;
 7    int local_4c;
 8    int local_48;
 9    int local_44;
10    int local_40;
11    int local_3c;
12    char local_38 [4];
13    char local_34;
14    char cStack39;
15    long local_10;
16
17    local_10 = *(long *)(in_FS_OFFSET + 0x28);
18    printf("Enter the flag: ");
19    __isoc99_scanf(&DAT_001009e5,local_38);
20    local_4c = 1;
21    local_40 = 1;
22    sVar1 = strlen(local_38);
23    local_3c = (int)sVar1;
24    if ((((local_3c == 0x12) && (local_38[0] == 'f')) && (local_38[1] == 'l')) &&
25       (((local_38[2] == 0x61 && (local_38[3] == 'g')) && ((local_34 == '{' && (cStack39 == '}'))))))
26    {
27      local_44 = 0;
28      fun(local_40,&local_4c);
29      local_48 = 5;
30      while (local_48 < local_3c + -1) {
31        if (local_38[local_48] != (byte)((char)local_4c + (local_38[2] ^ 0x20) + -0x20)) {
32          local_44 = 1;
33          break;
34        }
35        fun(local_40,&local_4c);
36        local_48 = local_48 + 1;
37      }
38      if (local_44 == 0) {
39        puts("Well done!");
40      }
41      else {
42        puts("Better luck next time!");
43      }
44    }
```

Line 22 finds the length of the input string using strlen function and line 23 stores the length in a variable named **local_3c**:


Decompile: main - (rev3)

```
 1
 2 undefined8 main(void)
 3
 4 {
 5   size_t sVar1;
 6   long in_FS_OFFSET;
 7   int local_4c;
 8   int local_48;
 9   int local_44;
10   int local_40;
11   int local_3c;
12   char local_38 [4];
13   char local_34;
14   char cStack39;
15   long local_10;
16
17   local_10 = *(long *)(in_FS_OFFSET + 0x28);
18   printf("Enter the flag: ");
19   __isoc99_scanf(&DAT_001009e5,local_38);
20   local_4c = 1;
21   local_40 = 1;
22   sVar1 = strlen(local_38);
23   local_3c = (int)sVar1;
24   if ((((local_3c == 0x12) && (local_38[0] == 'f')) && (local_38[1] == 'l')) &&
25      (((local_38[2] == 0x61 && (local_38[3] == 'g')) && ((local_34 == '{' && (cStack39 == '}'))))))
26   {
27     local_44 = 0;
28     fun(local_40,&local_4c);
29     local_48 = 5;
30     while (local_48 < local_3c + -1) {
31       if (local_38[local_48] != (byte)((char)local_4c + (local_38[2] ^ 0x20) + -0x20)) {
32         local_44 = 1;
33         break;
34       }
35       fun(local_40,&local_4c);
36       local_48 = local_48 + 1;
37     }
38     if (local_44 == 0) {
39       puts("Well done!");
40     }
41     else {
42       puts("Better luck next time!");
43     }
44   }
```

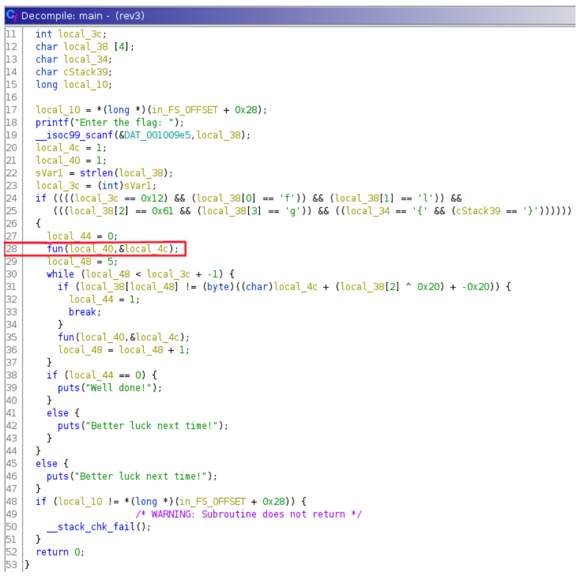
The ‘if condition’ in line 24 and 25 checks if the length of the input string is equal to 0x12 i.e. 18 and also the format of the input string i.e. whether the input string is of the form **flag{...}** or not. In case the if condition is not satisfied then the else statement (in line 45) is executed and line 46 prints “Better luck next time” :


Decompile: main - (rev3)

```
11   int local_3c;
12   char local_38 [4];
13   char local_34;
14   char cStack39;
15   long local_10;
16
17   local_10 = *(long *)(in_FS_OFFSET + 0x28);
18   printf("Enter the flag: ");
19   __isoc99_scanf(&DAT_001009e5,local_38);
20   local_4c = 1;
21   local_40 = 1;
22   sVar1 = strlen(local_38);
23   local_3c = (int)sVar1;
24   if ((((local_3c == 0x12) && (local_38[0] == 'f')) && (local_38[1] == 'l')) &&
25      (((local_38[2] == 0x61 && (local_38[3] == 'g')) && ((local_34 == '{' && (cStack39 == '}'))))))
26   {
27     local_44 = 0;
28     fun(local_40,&local_4c);
29     local_48 = 5;
30     while (local_48 < local_3c + -1) {
31       if (local_38[local_48] != (byte)((char)local_4c + (local_38[2] ^ 0x20) + -0x20)) {
32         local_44 = 1;
33         break;
34       }
35       fun(local_40,&local_4c);
36       local_48 = local_48 + 1;
37     }
38     if (local_44 == 0) {
39       puts("Well done!");
40     }
41     else {
42       puts("Better luck next time!");
43     }
44   }
45   else {
46     puts("Better luck next time!");
47   }
48   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
49                     /* WARNING: Subroutine does not return */
50     __stack_chk_fail();
51   }
52   return 0;
53 }
```

Hence the if condition needs to be satisfied i.e. the input string should be of length 18 and it should be in the form **flag{...}**.

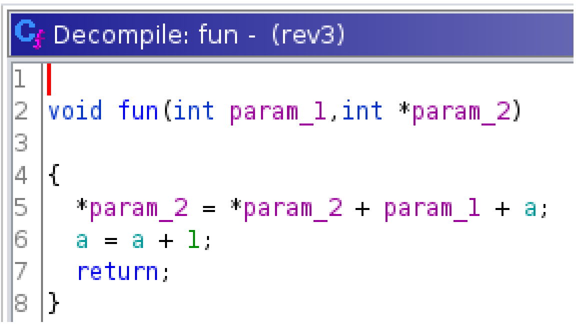
Now, inside the block of the if statement, there is a call to the function **fun()** in line 28. The function **fun()** takes two parameters: first one is the value of the variable **local_40** (which is equal to 1 as initialized earlier) and second is the address of the variable **local_4c** :



Decompile: main - (rev3)

```
11   int local_3c;
12   char local_38 [4];
13   char local_34;
14   char cStack39;
15   long local_10;
16 
17   local_10 = *(long *)(in_FS_OFFSET + 0x28);
18   printf("Enter the flag: ");
19   __isoc99_scanf(&DAT_001009e5,local_38);
20   local_4c = 1;
21   local_40 = 1;
22   sVar1 = strlen(local_38);
23   local_3c = (int)sVar1;
24   if ((((local_3c == 0x12) && (local_38[0] == 'f')) && (local_38[1] == 'l')) &&
25      (((local_38[2] == 0x61 && (local_38[3] == 'g')) && ((local_34 == '{' && (cStack39 == '}'))))))
26   {
27     local_44 = 0;
28     fun(local_40,&local_4c);
29     local_48 = 5;
30     while (local_48 < local_3c + -1) {
31       if (local_38[local_48] != (byte)((char)local_4c + (local_38[2] ^ 0x20) + -0x20)) {
32         local_44 = 1;
33         break;
34       }
35       fun(local_40,&local_4c);
36       local_48 = local_48 + 1;
37     }
38     if (local_44 == 0) {
39       puts("Well done!");
40     }
41     else {
42       puts("Better luck next time!");
43     }
44   }
45   else {
46     puts("Better luck next time!");
47   }
48   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
49                     /* WARNING: Subroutine does not return */
50     __stack_chk_fail();
51   }
52   return 0;
53 }
```

Double clicking on the function **fun()** will show the definition of the function which looks like:

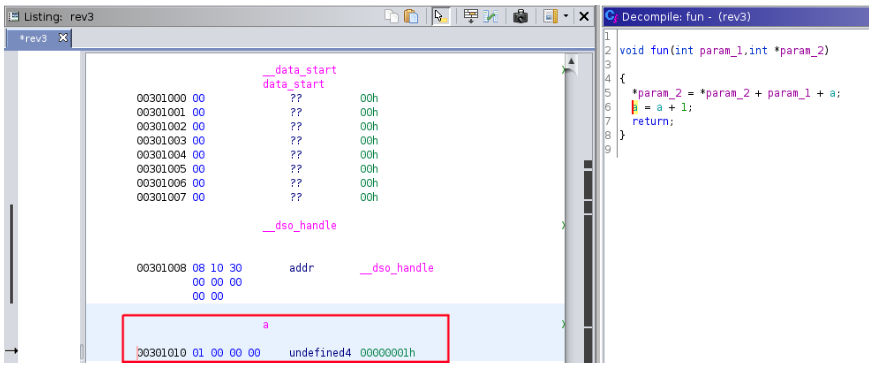


Decompile: fun - (rev3)

```
1 
2 void fun(int param_1,int *param_2)
3 
4 {
5   *param_2 = *param_2 + param_1 + a;
6   a = a + 1;
7   return;
8 }
```

We can see that the first parameter of **fun()** is an integer and the second is an integer pointer. Value of **param_1** will be equal to value of **local_40** i.e. 1 and value of **param_2** will be equal to the address of **local_4c**. Hence changing anything at the location pointed by **param_2** will also change the value of **local_4c**.

The statement: ***param_2 = *param_2 + param_1 + a;** in line 5 changes the value of ***param_2** i.e. **local_4c** to **local_4c + 1 + a**. Here **a** is a global variable and can be examined by double clicking on it:



Listing: rev3

*rev3

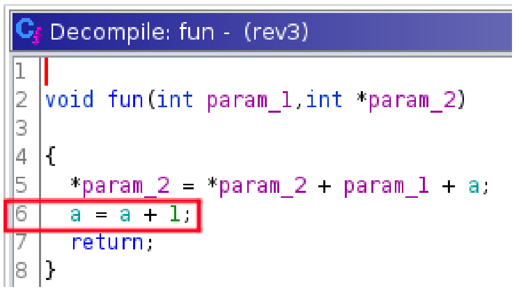
```
                                 __data_start
                                 data_start
        00301000 00              ??         00h
        00301001 00              ??         00h
        00301002 00              ??         00h
        00301003 00              ??         00h
        00301004 00              ??         00h
        00301005 00              ??         00h
        00301006 00              ??         00h
        00301007 00              ??         00h

                                 __dso_handle

        00301008 08 10 30        addr       __dso_handle
                 00 00 00
                 00 00

                                 a

        00301010 01 00 00 00     undefined4 00000001h
```

Decompile: fun - (rev3)

```
1 
2 void fun(int param_1,int *param_2)
3 
4 {
5   *param_2 = *param_2 + param_1 + a;
6   a = a + 1;
7   return;
8 }
9 
```

We can see that the value of **a** is **1** initially. Hence when **fun()** will be called for the first time, the value of **local_4c** will change to **local_4c + 1 + 1** i.e. **local_4c + 2**.

Line 6 in **fun()**, updates the value of **a** by adding 1 to it:



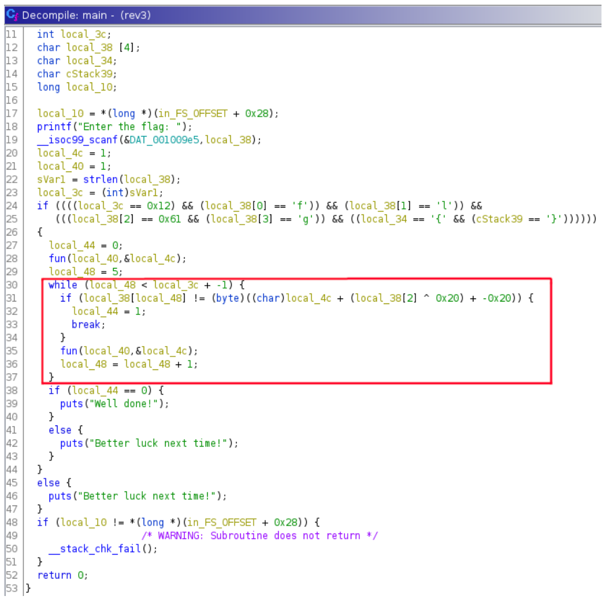
```
1  
2  void fun(int param_1,int *param_2)
3  
4  {
5    *param_2 = *param_2 + param_1 + a;
6    a = a + 1;
7    return;
8  }
```

Hence, to summarize, each time we call the function **fun()**:

- The value of **local_4c** is increased by **1 + a**
- The value of **a** increases by **1**.

Therefore after the first call to **fun()**, the value of **local_4c** becomes **3** and the value of **a** becomes **2**.

Now, we focus on the while loop given below:



```
11   int local_3c;
12   char local_38 [4];
13   char local_34;
14   char cStack39;
15   long local_10;
16   
17   local_10 = *(long *)(in_FS_OFFSET + 0x28);
18   printf("Enter the flag: ");
19   __isoc99_scanf(&DAT_001009e5,local_38);
20   local_4c = 1;
21   local_40 = 1;
22   sVar1 = strlen(local_38);
23   local_3c = (int)sVar1;
24   if ((((local_3c == 0x12) && (local_38[0] == 'f')) && (local_38[1] == 'l')) &&
25      (((local_38[2] == 0x61 && (local_38[3] == 'g')) && ((local_34 == '{' && (cStack39 == '}'))))))
26   {
27     local_44 = 0;
28     fun(local_40,&local_4c);
29     local_48 = 5;
30     while (local_48 < local_3c + -1) {
31       if (local_38[local_48] != (byte)((char)local_4c + (local_38[2] ^ 0x20) + -0x20)) {
32         local_44 = 1;
33         break;
34       }
35       fun(local_40,&local_4c);
36       local_48 = local_48 + 1;
37     }
38     if (local_44 == 0) {
39       puts("Well done!");
40     }
41     else {
42       puts("Better luck next time!");
43     }
44   }
45   else {
46     puts("Better luck next time!");
47   }
48   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
49                       /* WARNING: Subroutine does not return */
50     __stack_chk_fail();
51   }
52   return 0;
53 }
```

The 'if statement' inside the while loop checks if the character at index **local_48** of the input string is equal to the character corresponding to the ASCII code **local_4c + (local_38[2] ^ 0x20) – 0x20**

Now, **local_38[2]** is the third character of our input string which is of the form: fl**a**g{...},

Hence value of **local_38[2]** is '**a**'.

Therefore, **local_4c + (local_38[2] ^ 0x20) – 0x20 = local_4c + (0x61 ^ 0x20) – 0x20**, where 0x61 is the ASCII code for 'a'.

Simplifying the above expression results in: **local_4c + 0x21**

Hence, in the 'if statement' at line 31, the character at the index **local_48** of the input string is compared with the character having the ASCII code **local_4c + 0x21** i.e. **local_4c + 33**.

In case they are not equal, the variable **local_44** is set to **1** and the break statement is executed which in turn executes the instruction at line 42 and prints “Better luck next time”.

So, our target is to satisfy the ‘if condition’ to avoid setting **local_44** to **1**.

When the ‘if condition’ is successfully satisfied, the function **fun()** is called with the same parameters as earlier and after that **local_48** is incremented by **1**.

Before entering the while loop, **fun()** was called once.

Hence the value of **local_4c** had become **3** and the value of **a** had become **2** before entering the while loop as explained above.

With subsequent calls to **fun()** in the while loop, the value of **local_48** and **a** will be updated similarly.

Note: *The while loop runs till **local_48** is less than **local_3c – 1**. Since **local_3c** is the length of the input string whose correct length is **18**, therefore for the correct input string, the loop will run as long as **local_48** is less than **local_3c - 1 = 17.** Also since in each iteration, **local_48** is incremented by **1** and it is initialized to **5**, therefore the loop will run for **17 – 5 = 12** times.*

Thus, the value of **local_48** and **a** at the start of each iteration of the while loop are as follows:

Iteration	1	2	3	4	5	6	7	8	9	10	11	12
local_4c	3	6	10	15	21	28	36	45	55	66	78	91
a	2	3	4	5	6	7	8	9	10	11	12	13

Now, the character to be checked with in each execution of the ‘if statement’ in the while loop can be given as:

Iteration	1	2	3	4	5	6	7	8	9	10	11	12
local_4c + 33	36	39	43	48	54	61	69	78	88	99	111	124
Character	$	‘	+	0	6	=	E	N	X	c	o	|

Hence the flag is **flag{$‘+06=ENXco|}**