


writeup - bamboofox2021

BetterThanASM

The source file can be found at [here](#).

When I saw that source file with **.ll extension**, it makes me confused because I haven't work on it before. After searching Google in quite long time, I know it is assembly of LLVM compiler. Here, I have two ways to solve challenge: (1) read assembly, understand deeply it or (2) using **clang** compiler assembly- source code to executable file(binary file) and using normally IDA tools to solve.

 Clang compiler uses the LLVM compiler infrastructure as its back end and it is developed to act as a drop-in replacement for the GCC.

In this write-up, I will present the second way to solve challenge(it's the significant difference about time comparing to reading entirely assembly format of llvm, which solution I did while the contest was running).

Installing *clang* on Kali Linux

```
1 sudo apt-get install clang
2
3 # create executable file from assembly format.
4 clang task.ll -o task
```

After compiling the **task.ll** with clang, I tried to run executable file and type string as **0123456789**, but receive fail notification as below:

```
1 $ ./task
2 Only the chosen one will know what the flag is!
3 Are you the chosen one?
```

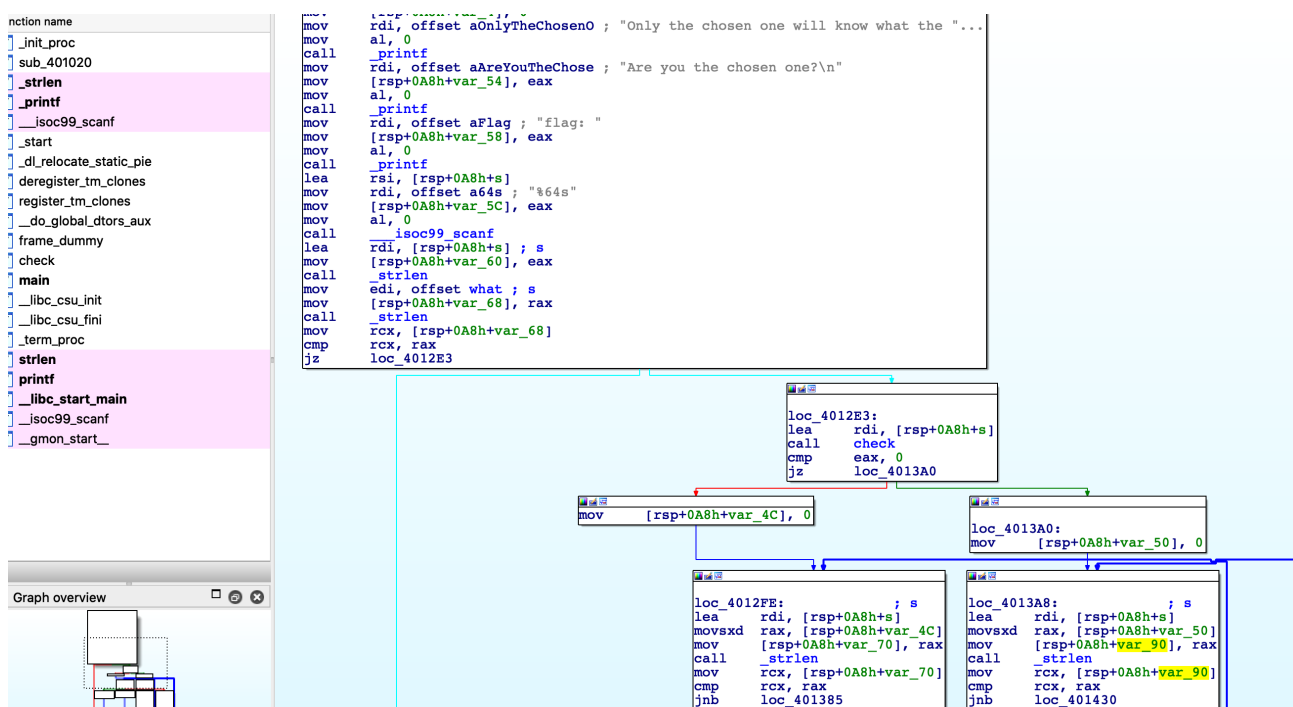
```

4  flag: 0123456789
5
6  [] [] [] [] [] You are not the chosen one! [] [] [] [] []
7

```

Find the right input

- Opening IDA and import **task** file:



- Firstly, user-input must be equal with the length of given **what** string(len = 56) and then, if it's satisfied, it's passed as only parameter into **check** function.

check() function

- Re-open with **ghidra** tool to read pseudo- code of this function:

```

byte check(long input)
{
    byte bVar1;
    size_t len_what;
    byte ret_value;
    int i;

    ret_value = 1;
    i = 0;
    while( true ) {
        len_what = strlen(what);
        if (len_what <= (ulong)(long)i) break;
        bVar1 = *(byte *)(input + i);
        len_what = strlen(what);
        ret_value = (bVar1 ^ *(byte *)(input + (ulong)(long)(i + 1) % len_what)) == what[i] & ret_value;
        i = i + 1;
    }
    return ret_value;
}

```

- We easy conclude that the return value(initially, its value is 1) of this function depends **input** string and **what** string:
 - if it's 1 then all elements of input must satisfy above equation(choose this case)
 - if it's 0, we will have a lot's of potential input which can happen(ignore this case)

main() function

```

iVar2 = check(input);
if (iVar2 == 0) {
    local_50 = 0;
    while( true ) {
        len_input = strlen((char *)input);
        if (len_input <= (ulong)(long)local_50) break;
        bVar1 = flag[local_50];
        len_input = strlen(secret);
        input[local_50] = bVar1 ^ secret[(ulong)(long)local_50 % len_input];
        local_50 = local_50 + 1;
    }
    printf(format,input);
}
else {
    i = 0;
    while( true ) {
        len_input = strlen((char *)input);
        if (len_input <= (ulong)(long)i) break;
        bVar1 = input[i];
        len_input = strlen(secret);
        input[i] = bVar1 ^ secret[(ulong)(long)i % len_input];
        i = i + 1;
    }
    printf(format,input);
}

```

- As mentioning above, I guess that the return value of check() function is 1, so I will pay my attention to the second branch of program: each element of input will be XOR-ed **secret** string before printing to screen.
- Because there are no additional hints about original flag, so I brute-forced all possible characters of flag(in range 32-127, in ASCII). We also noticed that, when I have the first character of input, I easily can get its original form(**xor with secret[i]**) and second character's original form(**xor with what[i]**)

solve_BetterThanASM.py

```

1 secret = 'B\n|_"\x06\x1bg7#\x0f\n)\t0Q8_{Y\x13\x18\rP'
2 what = "\x17/'\x17\x1dJy\x03,\x11\x1e&\nexjONacA-&\x01LANH'."&\x12>#'Z\x0f0
3 alphabet = []
4
5 for i in range(30, 127):
6     alphabet.append(i)
7
8 flag = [0]*56
9 for i in alphabet:

```

```

10  print( 'i = ', chr(i))
11  flag[0] = i
12  for j in range(0, len(flag) - 1):
13      tmp = flag[j] ^ ord(secret[j % len(secret)]) #flag[i] ^ secret
14      ori_flag_i_1 = tmp ^ ord(what[j]) #ori_flag[i] ^ what --> ord_fl
15      flag[j + 1] = ori_flag_i_1 ^ ord(secret[(j + 1) % len(secret)])
16  print('flag{', end='')
17  for k in flag:
18      print(chr(k), end='')
19  print('}')
20  print('\n*****\n')
21  print()
22  #*****
23  '''
24  i = 7
25  flag{7h15_f14g_15_v3ry_v3ry_l0ng_4nd_1_h0p3_th3r3_4r3_n0_7yp0}
26
27  *****
28
29  '''

```