

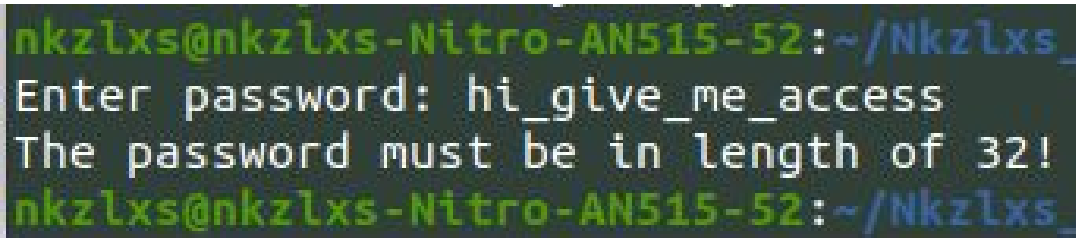
Wargames MY 2020 write-up

Category: Reverse-Engineering (a.k.a. RE)

Challenge Name: babyrev

Introduction

In this challenge we are given an executable file (not exe) which will accept 32 characters

A terminal window with a dark background and green text. The prompt is 'nkz1xs@nkz1xs-Nitro-AN515-52:~/Nkz1xs_'. The user has entered 'hi_give_me_access'. The program responds with 'The password must be in length of 32!'. The prompt is repeated at the bottom.

```
nkz1xs@nkz1xs-Nitro-AN515-52:~/Nkz1xs_  
Enter password: hi_give_me_access  
The password must be in length of 32!  
nkz1xs@nkz1xs-Nitro-AN515-52:~/Nkz1xs_
```

An example of how to program is

Of course, trying random input will not work since this is not the way to solve it.

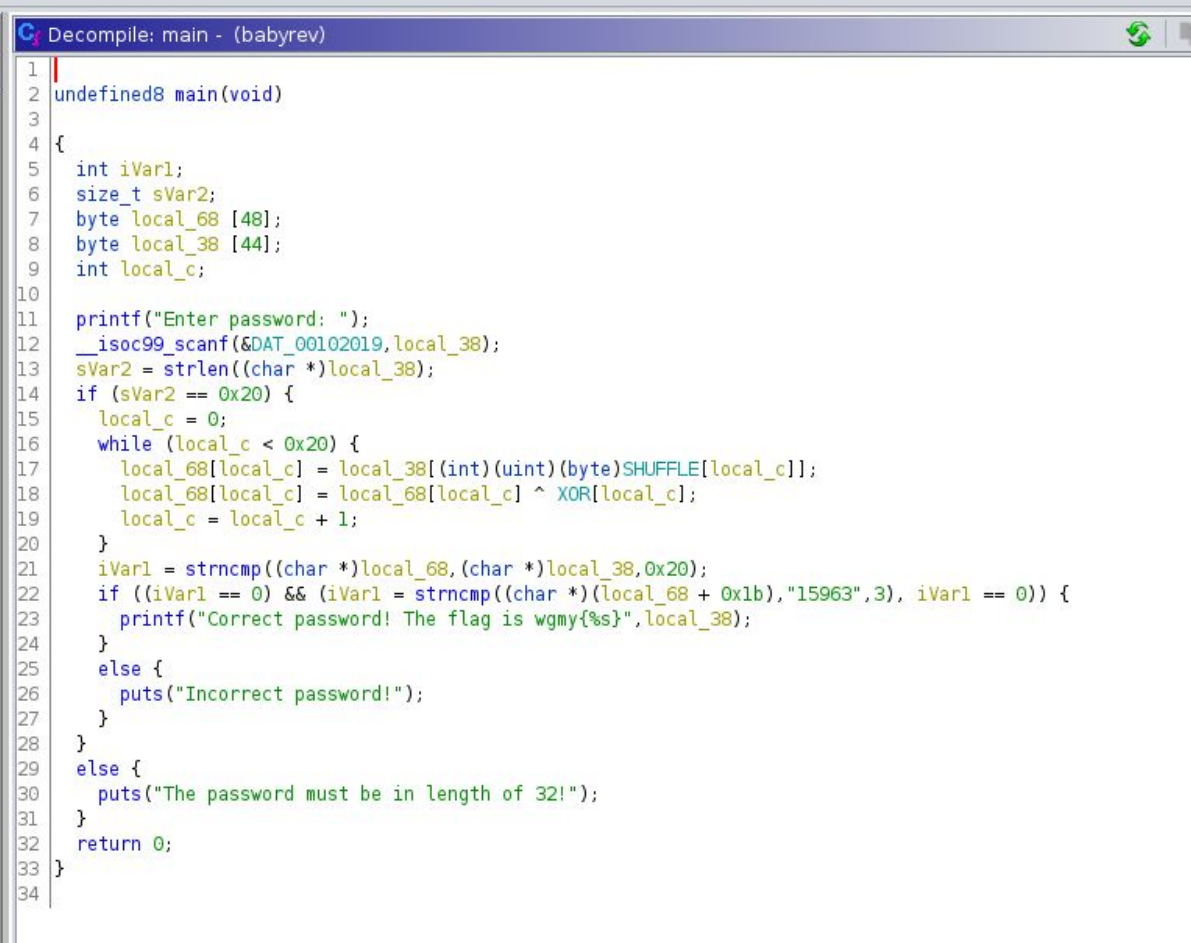
Tools used

1. [Ghidra](#) v9.1.2- For analysis of the executable
2. Python
 - a. [Z3 Library](#) - For solving the password

Investigating the Executable

First, open up Ghidra, select CodeBrowser, import the executable, accept to perform analysis on the executable as the program suggested.

At the decompiled main() function, we can clearly see how this program handles input and what is the condition for the flag to be printed.



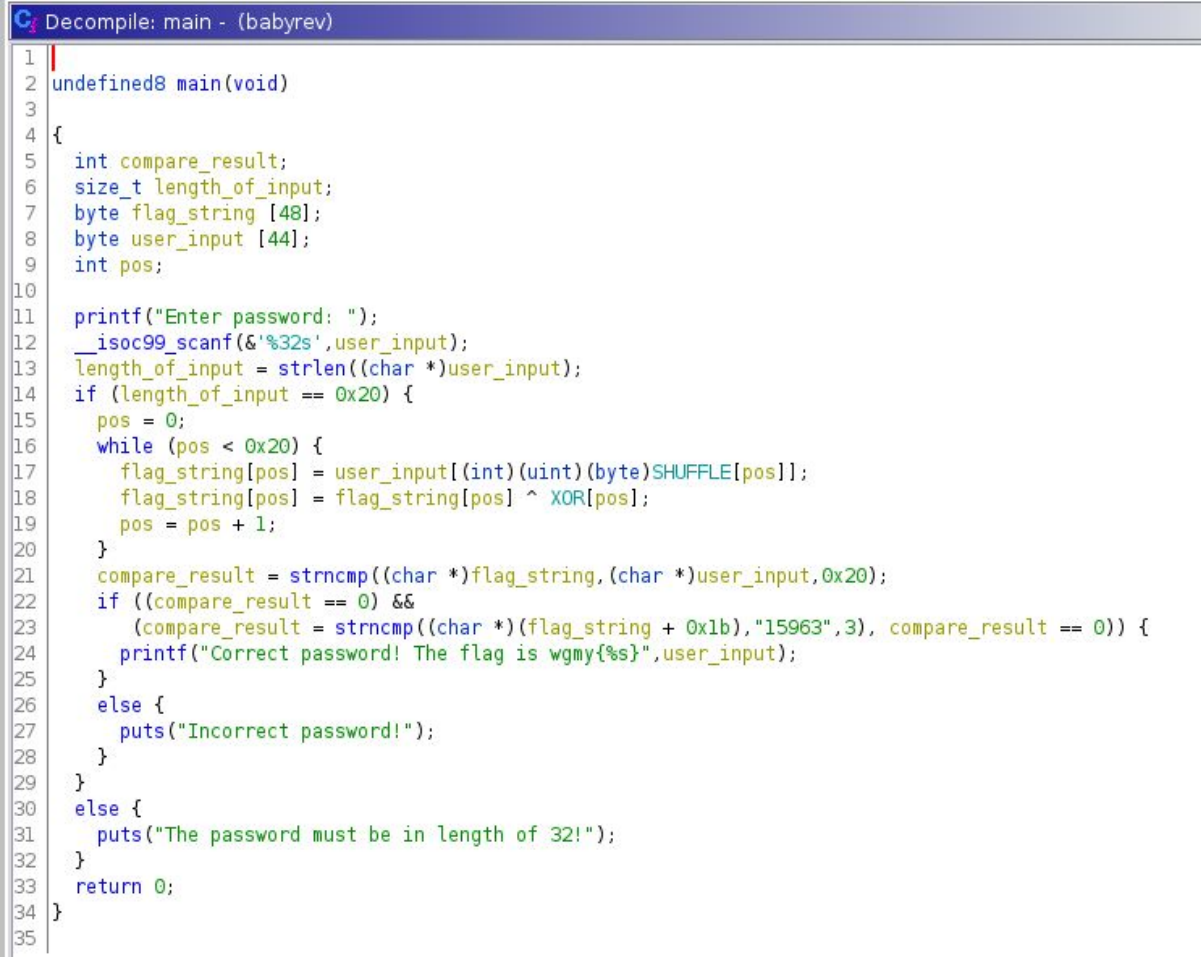
```

1  |
2  | undefined8 main(void)
3  |
4  | {
5  |     int iVar1;
6  |     size_t sVar2;
7  |     byte local_68 [48];
8  |     byte local_38 [44];
9  |     int local_c;
10 |
11 |     printf("Enter password: ");
12 |     __isoc99_scanf(&DAT_00102019,local_38);
13 |     sVar2 = strlen((char *)local_38);
14 |     if (sVar2 == 0x20) {
15 |         local_c = 0;
16 |         while (local_c < 0x20) {
17 |             local_68[local_c] = local_38[(int)(uint)(byte)SHUFFLE[local_c]];
18 |             local_68[local_c] = local_68[local_c] ^ XOR[local_c];
19 |             local_c = local_c + 1;
20 |         }
21 |         iVar1 = strncmp((char *)local_68,(char *)local_38,0x20);
22 |         if ((iVar1 == 0) && (iVar1 = strncmp((char *)local_68 + 0x1b,"15963",3), iVar1 == 0)) {
23 |             printf("Correct password! The flag is wgmy{%s}",local_38);
24 |         }
25 |         else {
26 |             puts("Incorrect password!");
27 |         }
28 |     }
29 |     else {
30 |         puts("The password must be in length of 32!");
31 |     }
32 |     return 0;
33 | }
34 |

```

The main() function of this program

Judging from the puts(s) and strcmp(s), we can certainly confirm that this is a program written in C. So let's rename some of it's variables for better investigation.



```

1  |
2  | undefined8 main(void)
3  |
4  | {
5  |     int compare_result;
6  |     size_t length_of_input;
7  |     byte flag_string [48];
8  |     byte user_input [44];
9  |     int pos;
10 |
11 |     printf("Enter password: ");
12 |     __isoc99_scanf(&"%32s",user_input);
13 |     length_of_input = strlen((char *)user_input);
14 |     if (length_of_input == 0x20) {
15 |         pos = 0;
16 |         while (pos < 0x20) {
17 |             flag_string[pos] = user_input[(int)(uint)(byte)SHUFFLE[pos]];
18 |             flag_string[pos] = flag_string[pos] ^ XOR[pos];
19 |             pos = pos + 1;
20 |         }
21 |         compare_result = strcmp((char *)flag_string, (char *)user_input,0x20);
22 |         if ((compare_result == 0) &&
23 |             (compare_result = strcmp((char *) (flag_string + 0x1b), "15963", 3), compare_result == 0)) {
24 |             printf("Correct password! The flag is wgmy{%s}",user_input);
25 |         }
26 |         else {
27 |             puts("Incorrect password!");
28 |         }
29 |     }
30 |     else {
31 |         puts("The password must be in length of 32!");
32 |     }
33 |     return 0;
34 | }
35 |

```

main() function with variable renamed

From the renamed function, we can see that the user_input is being processed 2 times, which is in line 17,18 and assigned to the variable flag_string.

The condition for the flag is user_input must be exactly the same as the flag_string and the [27] to [31] characters of the flag_string is "15963".

```

pos = 0;
while (pos < 0x20) {
    flag_string[pos] = user_input[(int)(uint)(byte)SHUFFLE[pos]];
    flag_string[pos] = flag_string[pos] ^ XOR[pos];
    pos = pos + 1;
}
compare_result = strcmp((char *)flag_string, (char *)user_input, 0);
flag_string being calculated

```

```

compare_result = strcmp((char *)flag_string, (char *)user_input, 0x20);
if ((compare_result == 0) &&
    (compare_result = strcmp((char *)flag_string + 0x1b, "15963", 3), compare_result == 0)) {
    printf("Correct password! The flag is wgmy{%s}", user_input);
}
else {
    puts("Incorrect password!");
}

```

print flag condition

So, let's find out what's in the SHUFFLE and XOR.

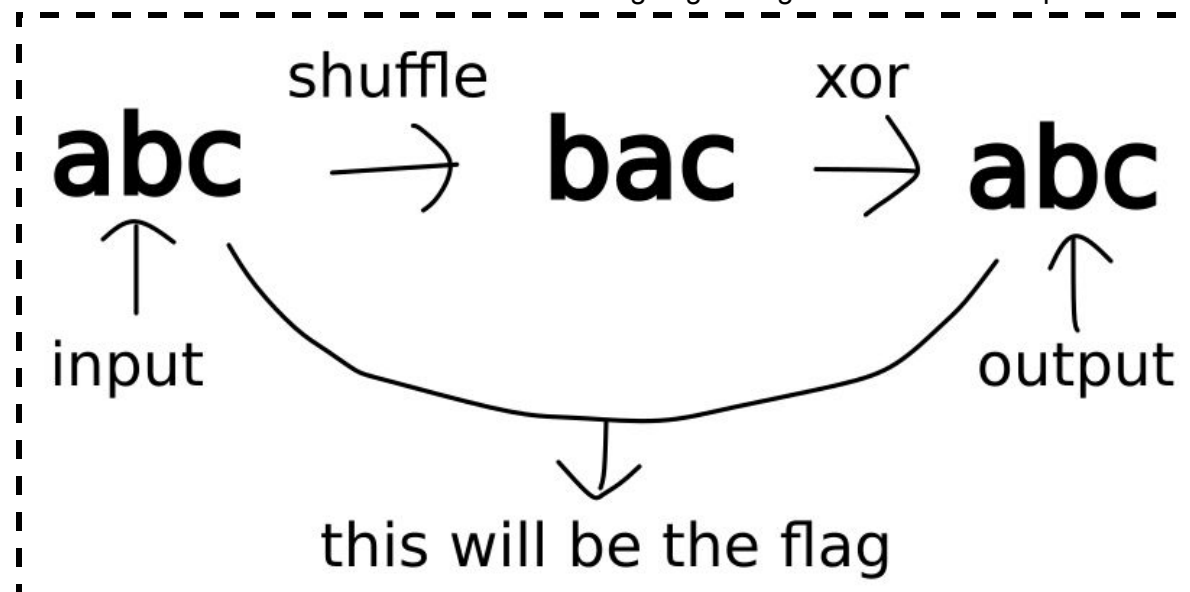
SHUFFLE(HEX) = 07 04 15 12 1d 13 1b
 08 1f 16 0f 06 0a 19 18 11 01 03 02 17 0d
 14 05 00 0c 1c 0b 1a 0e 1e 09 10

SHUFFLE			
00104060	07	db	7h
00104061	04	??	04h
00104062	15	??	15h
00104063	12	??	12h
00104064	1d	??	1Dh
00104065	13	??	13h
00104066	1b	??	1Bh
00104067	08	??	08h
00104068	1f	??	1Fh
00104069	16	??	16h
0010406a	0f	??	0Fh
0010406b	06	??	06h
0010406c	0a	??	0Ah
0010406d	19	??	19h
0010406e	18	??	18h
0010406f	11	??	11h
00104070	01	??	01h
00104071	03	??	03h
00104072	02	??	02h
00104073	17	??	17h
00104074	0d	??	0Dh
00104075	14	??	14h
00104076	05	??	05h
00104077	00	??	00h
00104078	0c	??	0Ch
00104079	1c	??	1Ch
0010407a	0b	??	0Bh
0010407b	1a	??	1Ah
0010407c	0e	??	0Eh
0010407d	1e	??	1Eh
0010407e	09	??	09h
0010407f	10	??	10h

XOR(HEX) = 56 06 06 01 09 52 06
 03 51 04 57 07 52 07 50 06 06 06
 07 54 57 56 02 55 06 01 52 53 54
 0f 54 03

XOR			
00104080	56	char	'V'
00104081	06	??	06h
00104082	06	??	06h
00104083	01	??	01h
00104084	09	??	09h
00104085	52	??	52h
00104086	06	??	06h
00104087	03	??	03h
00104088	51	??	51h
00104089	04	??	04h
0010408a	57	??	57h
0010408b	07	??	07h
0010408c	52	??	52h
0010408d	07	??	07h
0010408e	50	??	50h
0010408f	06	??	06h
00104090	06	??	06h
00104091	06	??	06h
00104092	07	??	07h
00104093	54	??	54h
00104094	57	??	57h
00104095	56	??	56h
00104096	02	??	02h
00104097	55	??	55h
00104098	06	??	06h
00104099	01	??	01h
0010409a	52	??	52h
0010409b	53	??	53h
0010409c	54	??	54h
0010409d	0f	??	0Fh
0010409e	54	??	54h
0010409f	03	??	03h

Now, we got the value for the flag_string to be generated. What's left now is to find the combination of words that is still the same after going through a shuffle and xor process.



Cracking the password

Here we will be using z3solver library from python to do the job, the solution is as below

```
from z3 import *

shuffle_bytes=
0x070415121d131b081f160f060a191811010302170d1405000c1c0b1a0e1e0910
xor_bytes =
0x56060601095206035104570752075006060607545756025506015253540f5403

# The input a.k.a the password
input_32_bytes_word = [
    z3.BitVec(name=f"x{i}",bv=8) for i in range(0,32)
]

# The input after being shuffle'd
shuffle_result = []
for a in range(0,32):
    # Grabbing 1 bytes every iteration from the shufflebytes
    x = (shuffle_bytes >> a*8) & 0xff
    shuffle_result.append(input_32_bytes_word[x])

# The LSB of shuffle_bytes is put in the beginning of the array
# so the shuffle_result should be reversed
shuffle_result = shuffle_result[::-1]

# shuffle'd result get xor'd
yeah = Concat(shuffle_result) ^ xor_bytes

# the password
hoo = Concat(input_32_bytes_word)

# the extra condition found when investigating the executable
# the word from [27] to [31] should be "159"
hooaha =
Concat(shuffle_result[27],shuffle_result[28],shuffle_result[29]) ^
0x53540f

# create a z3 colver object
a_solver = Solver()

# add constraints into the solver
a_solver.add(yeah==hoo)
```

```

a_solver.add(hooha==0x313539) # 0x313539 is "159" btw

# printing out the result(password)
status = a_solver.check()
if str(status) == "sat":
    a_model = a_solver.model()
    for x in range(0,32):
        try:
            print(chr(a_model[input_32_bytes_word[x]].as_long()),end="")
        except:
            pass
    print()

```

After running the program, a value is printed

```

nkz1xs@nkz1xs-Nitro-AN515-52:~/Nkz
76420d7abbe073a20436d2fb14b15963
nkz1xs@nkz1xs-Nitro-AN515-52:~/Nkz

```

Lastly, let's test the result in the program.

```

nkz1xs@nkz1xs-Nitro-AN515-52:~/Nkz1xs_projects/ctf/wargames2020/rever
Enter password: 76420d7abbe073a20436d2fb14b15963
Correct password! The flag is wgmy{76420d7abbe073a20436d2fb14b15963}r
sa_engineer/haby_gou$

```

We have successfully found the flag!