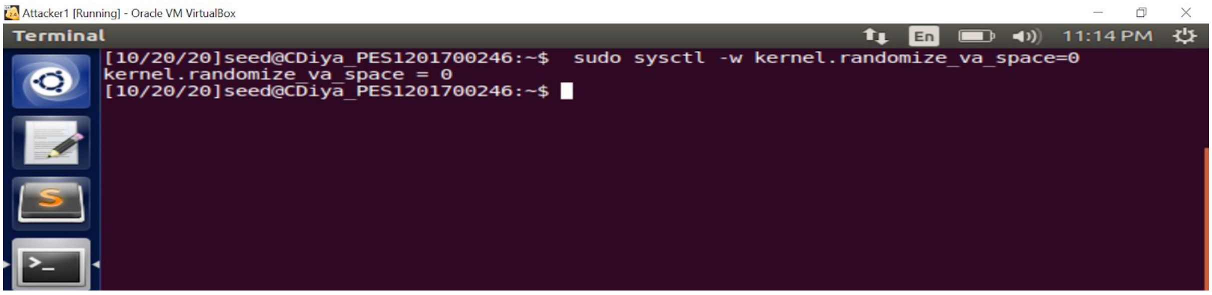
IS Laboratory 5

Format string Attack Lab

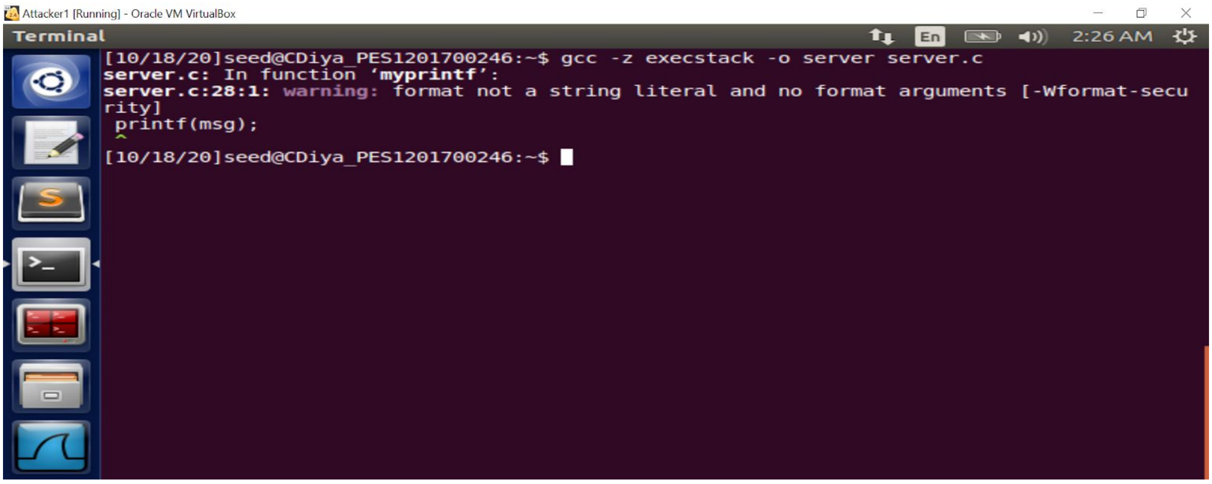
C Diya
PES1201700246

Client: 10.0.2.12
Server: 10.0.2.11

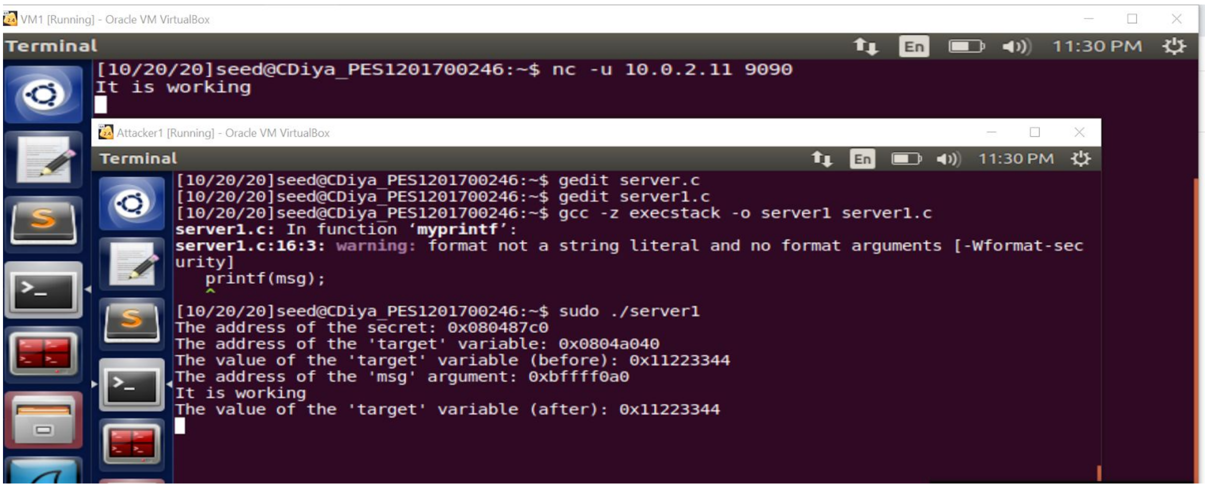
Task 1: Vulnerable Program



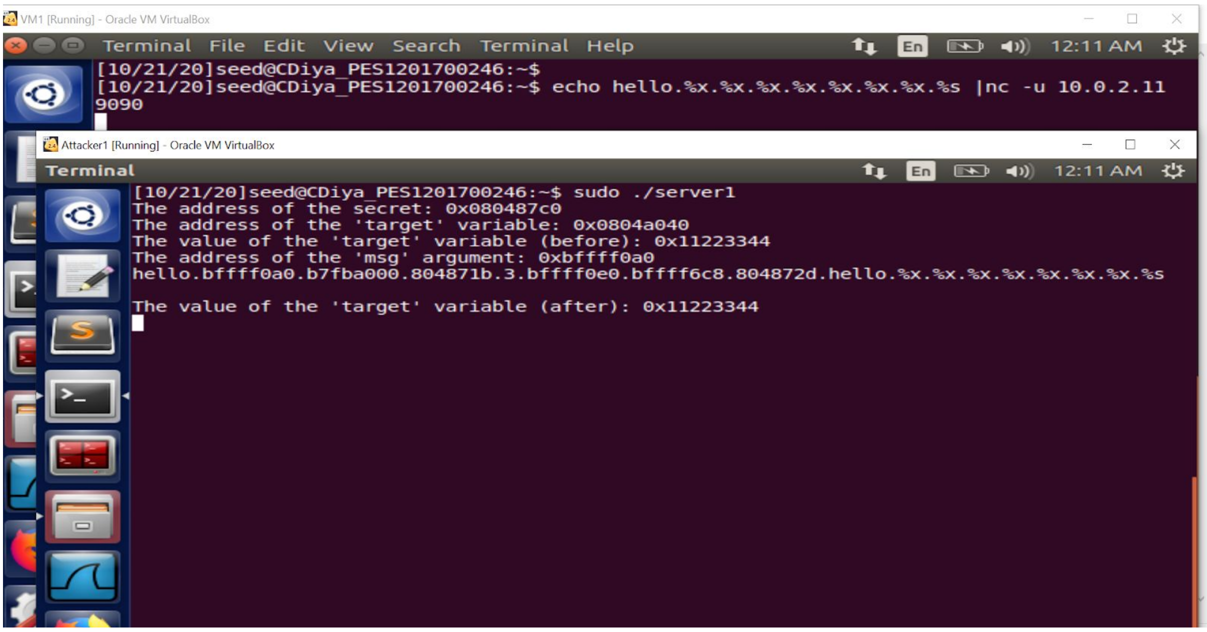
Observation : The screenshot above shows the address randomization is turned off to make the attack easier.



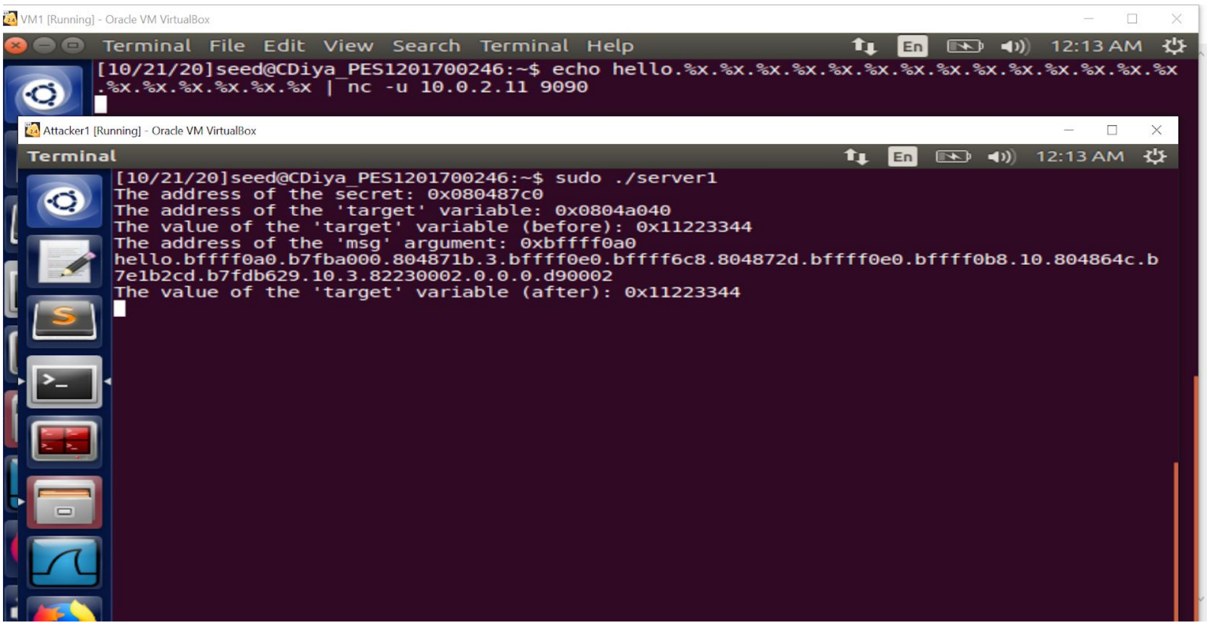
Observation : The screenshot above shows that when server.c is compiled, a warning message is received. It is because a format specifier was not used while using the printf. This warning message is a countermeasure implemented by the gcc compiler against format string vulnerabilities.



Observation : The screenshot above shows that data can be sent to the server(10.0.2.11). “It is working” is echoed from the client which is sent to the server. This message can be seen on the server VM’s terminal. The server program is supposed to print out whatever is sent by the client. The server listens to port 9090. On the client VM, data is sent to the server using the nc command, where the flag "-u" means UDP.



Observation : The screenshot above shows that data can be sent to the server(10.0.2.11). The message is echoed from the client which is sent to the server. This message argument address can be seen on the server VM’s terminal. The “hello” message can be seen as well due to the %s.



Observation : The screenshot above shows that data can be sent to the server(10.0.2.11). The message is echoed from the client which is sent to the server. This message argument address can be seen on the server VM's terminal.

Task 2: Understanding the Layout of the Stack

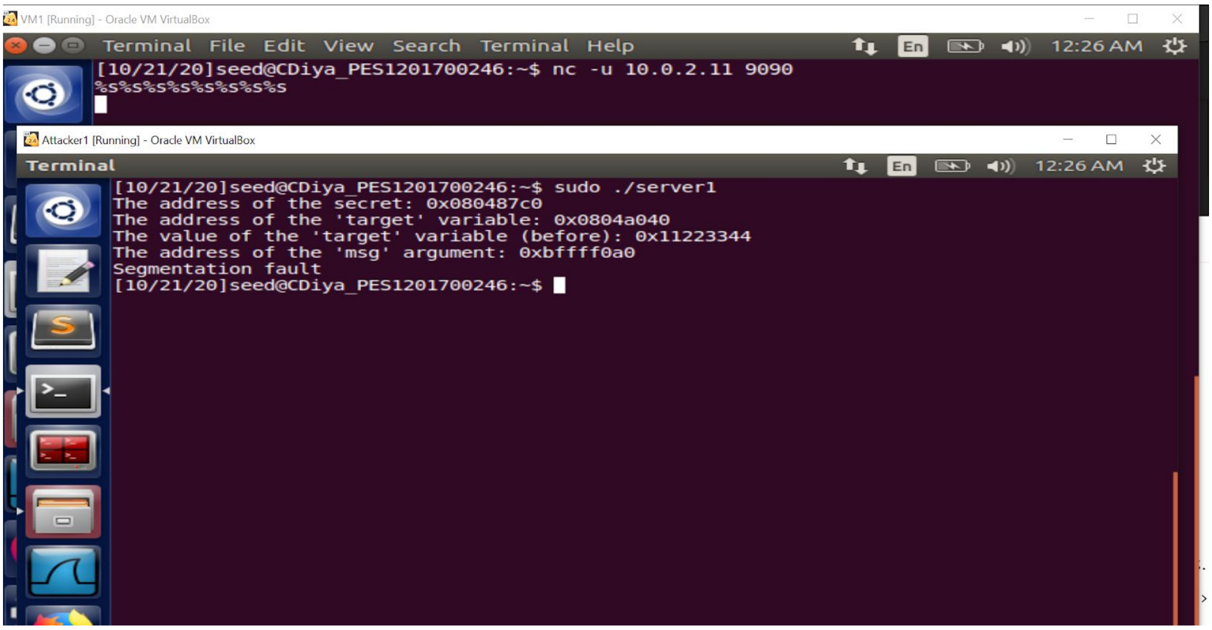
Answer 1:

The memory addresses at the following locations are the corresponding values:
Format String: 0xBFFFF080 (Msg Address – 4 * 8 | Buffer Start – 24 * 4)
Return Address: 0xBFFFF09C
Buffer Start: 0xBFFFF0E0

Answer 2:

Distance between the locations marked by 1 and 3 – 23 * 4 bytes = 92 bytes

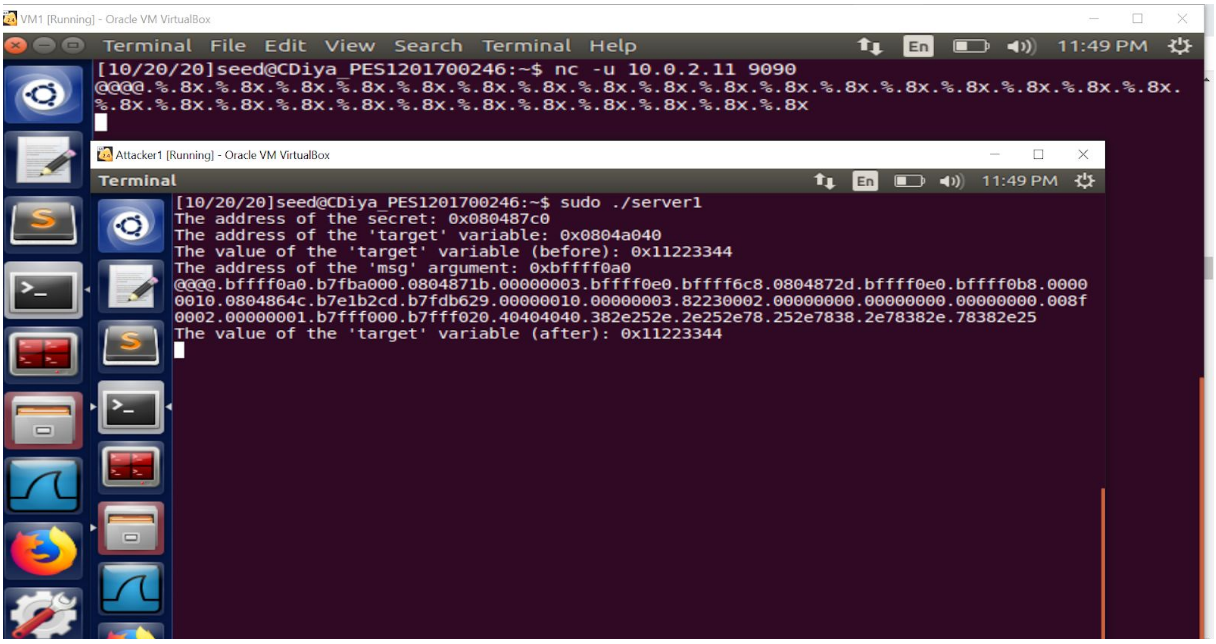
Task 3 : Crash the Program



Observation : The screenshot above shows the result when string of %s is used as an input to the program. Here, the program crashes because %s treats the obtained value from a location as an address and prints out the data stored at that address. The memory stored was not for the printf function and hence it might not contain addresses in all of the referenced locations, the program crashes. The value might contain references to protected memory or might not contain memory at all, leading to a crash. Thus, the program results in a segmentation fault as seen above.

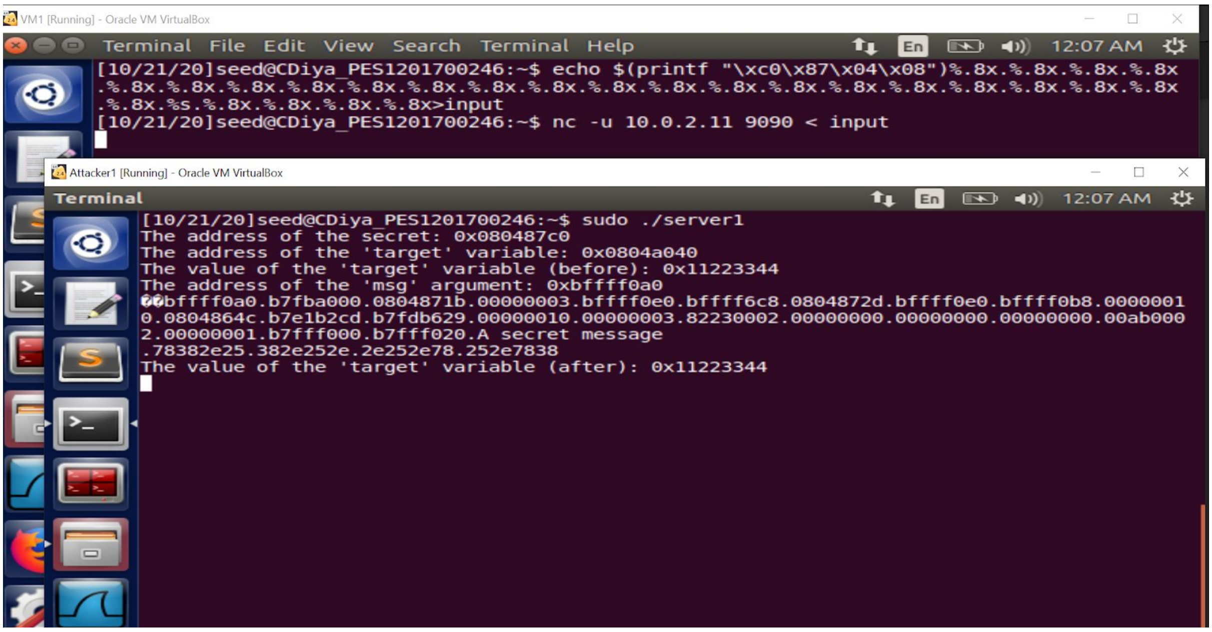
Task 4: Print Out the Server Program’s Memory

Task 4.A: Stack Data



Observation : The screenshot above shows that when the data -@@@@ and a series of %.8x is entered, it can be seen that at the 24th %x, the input 40404040 is observed and hence data that is stored on the stack was successfully read. The rest of the %x is also displaying the content of the stack. 24 format specifiers are required to print out the first 4 bytes of the input.

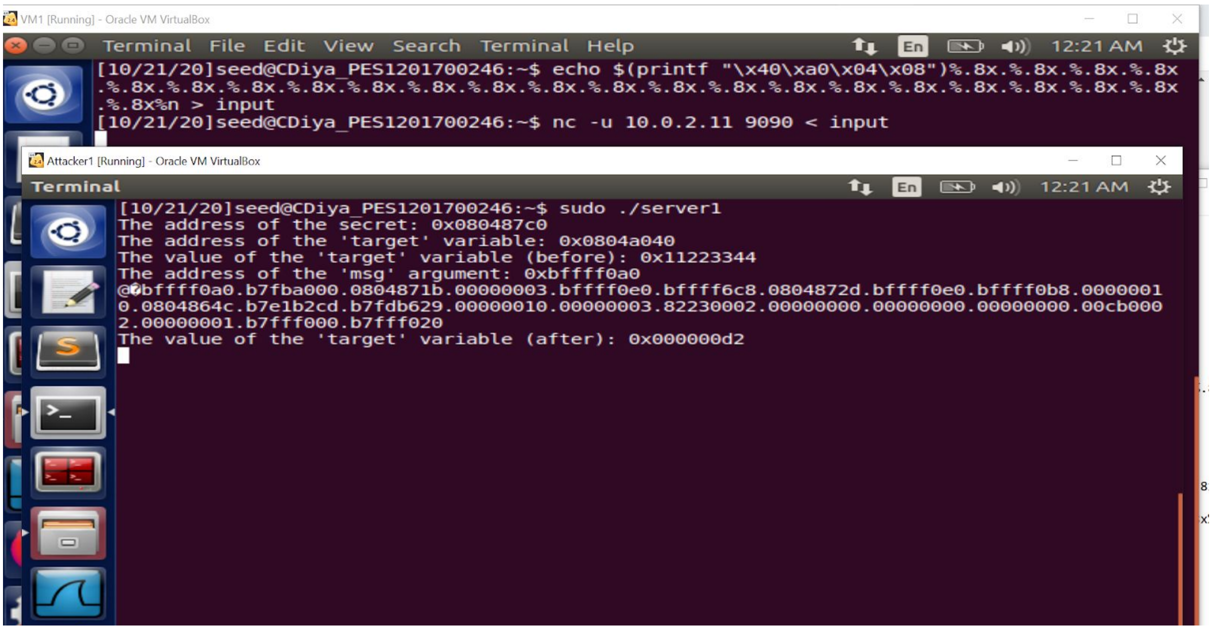
Task 4B: Heap Data



Observation : The screenshot above shows the message “A secret message” printed on the server machine’s terminal. Hence , the heap data was read successfully by storing the address of the heap data in the stack and then using the %s formatspecifier at the right location so that it reads the stored memory address and then gets the value from that address.

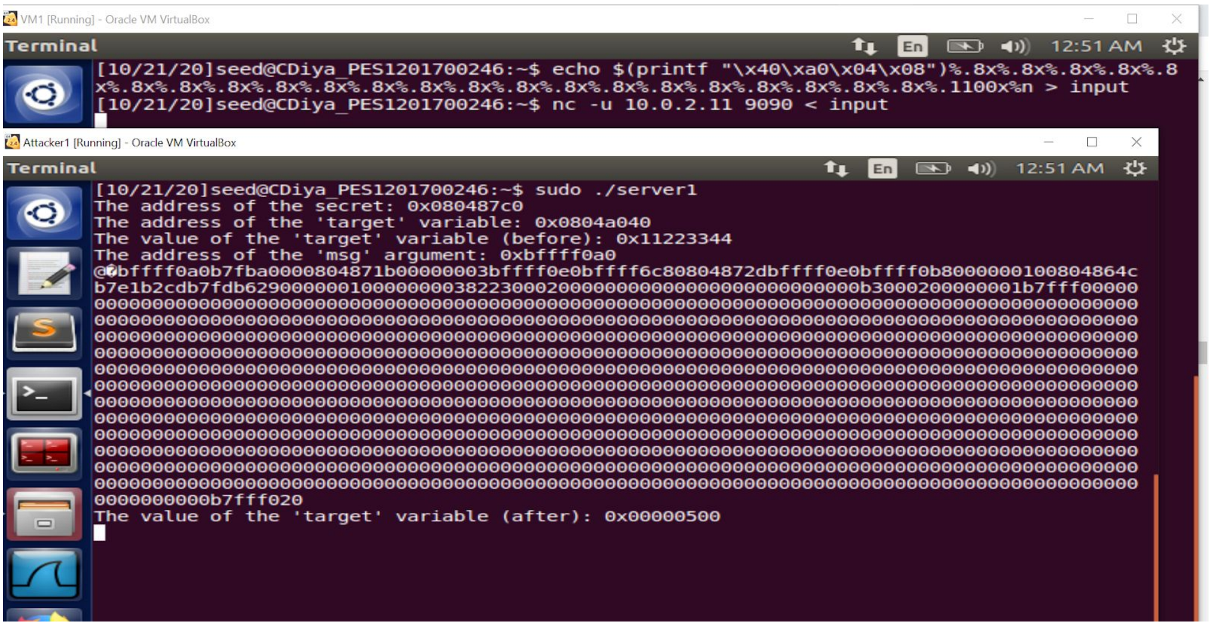
Task 5: Change the Server Program's Memory Task

5.A: Change the value to a different value



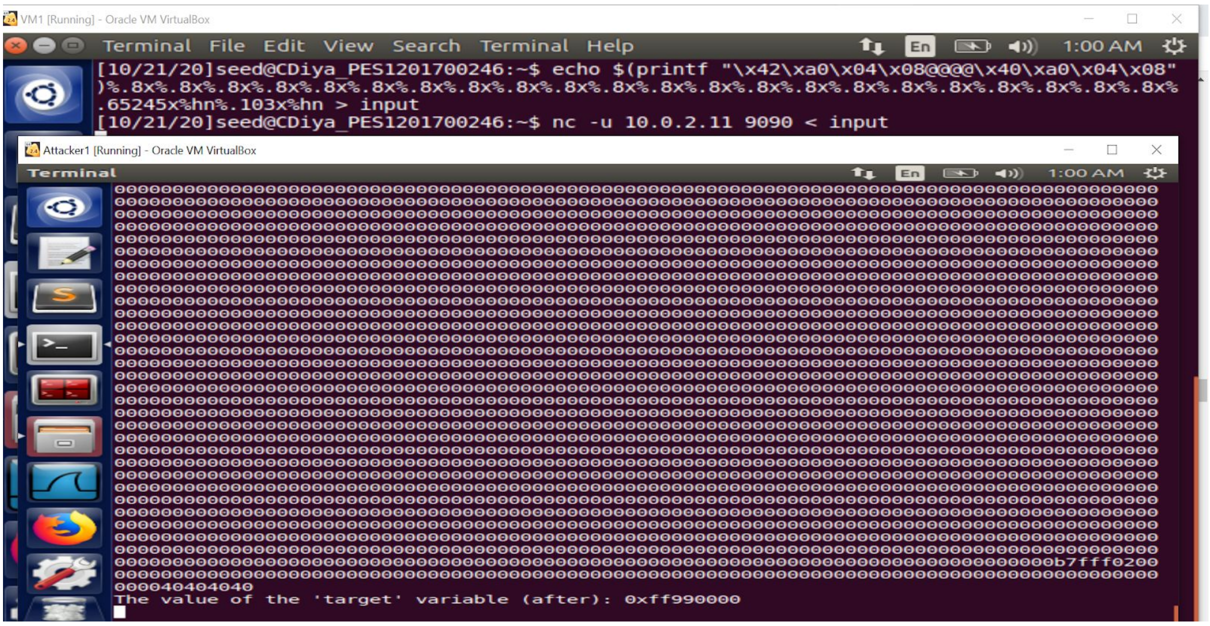
Observation : The screenshot above shows the target variable value has been changed. The above input is provided to the server and it can be seen that the target variable's value has changed from 0x11223344 to 0x000000d2. This is expected because 188 characters (23 * 8 + 4) are printed, and on entering %n at the address location stored in the stack, the value is changed {Hex value for 188}. Thus, memory's value was changed.

Task 5.B: Change the value to 0x500



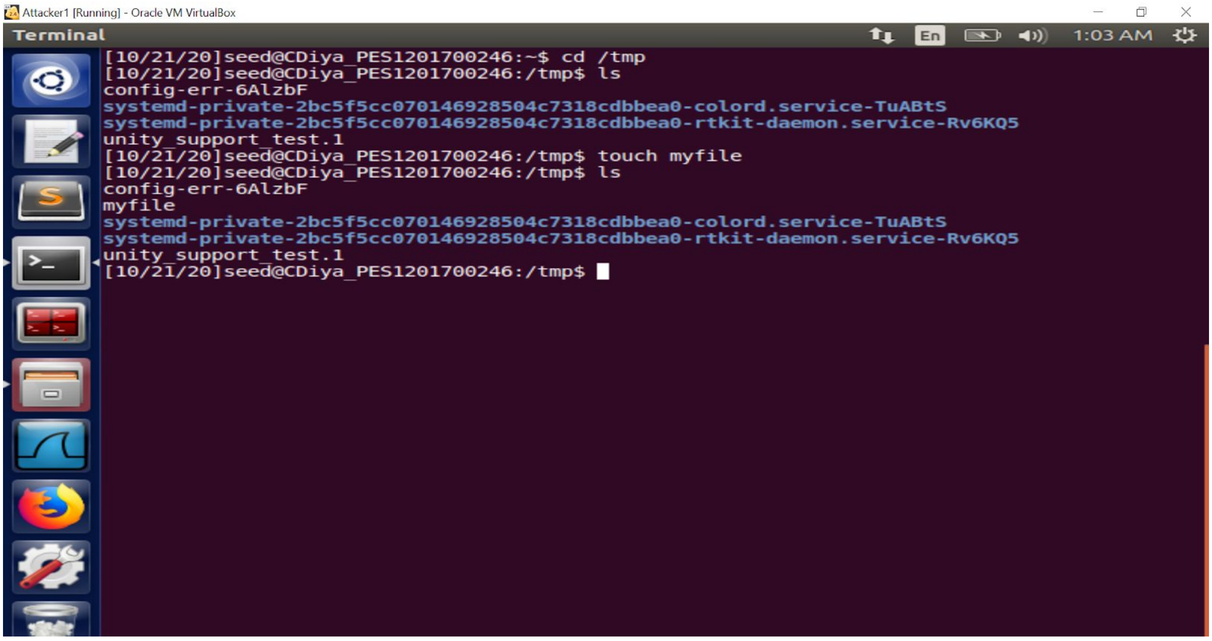
Observation : The screenshot above shows that the value is successfully changed from 0x11223344 to 0x0000500. To get a value of 500, the following is done 1280 – 188 =1100 in decimal, where 1280 stands for 500 in hex and 188 are the number of characters printed out before the 23rd %x. The 1100 characters are obtained using the precision modifier, and then using a %n to store the value.

Task 5.C: Change the value to 0xFF990000

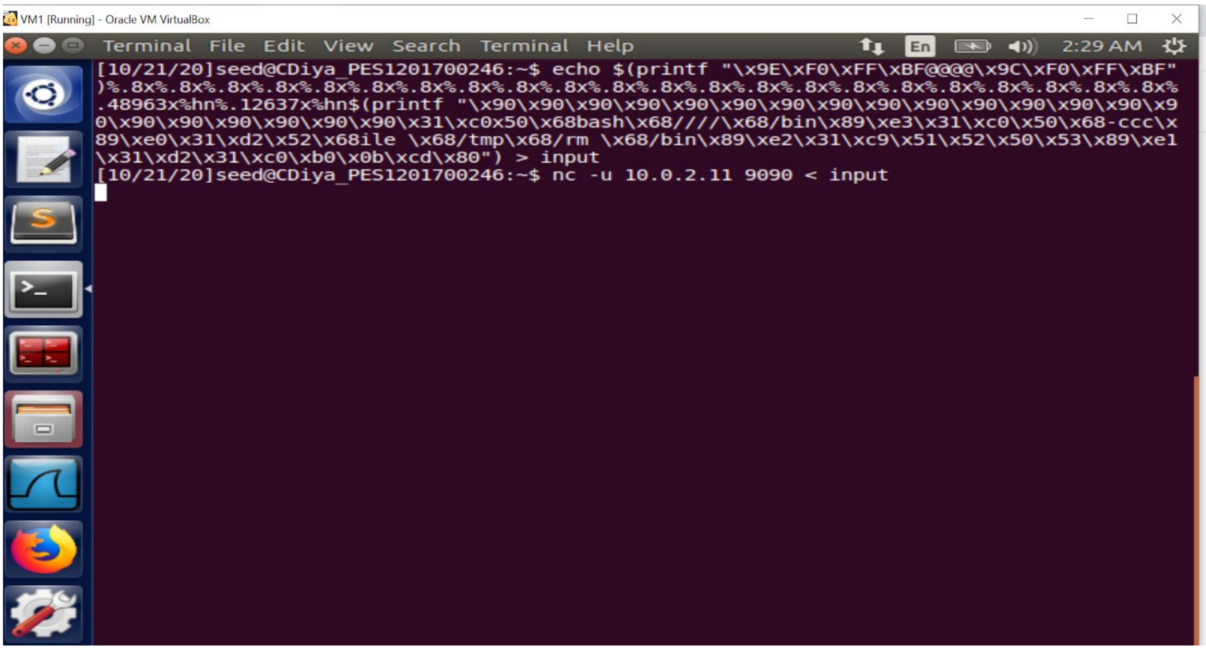


Observation: The screenshot above shows that the value of the target variable has successfully been changed to 0xff990000.
So, the memory addresses in 2 2-byte addresses are divided with the first address being the one containing a smaller value. This is because, %n is accumulative and hence storing the smaller value first and then adding characters to it and storing a larger value is optimal. In order to get a value of 0000, the value is overflowed, that leads for the memory to store only the lower 2 bytes of the value. Hence, 103 (decimal) is added to ff99 to get a value of 0000, that is stored in the lower byte of the destination address

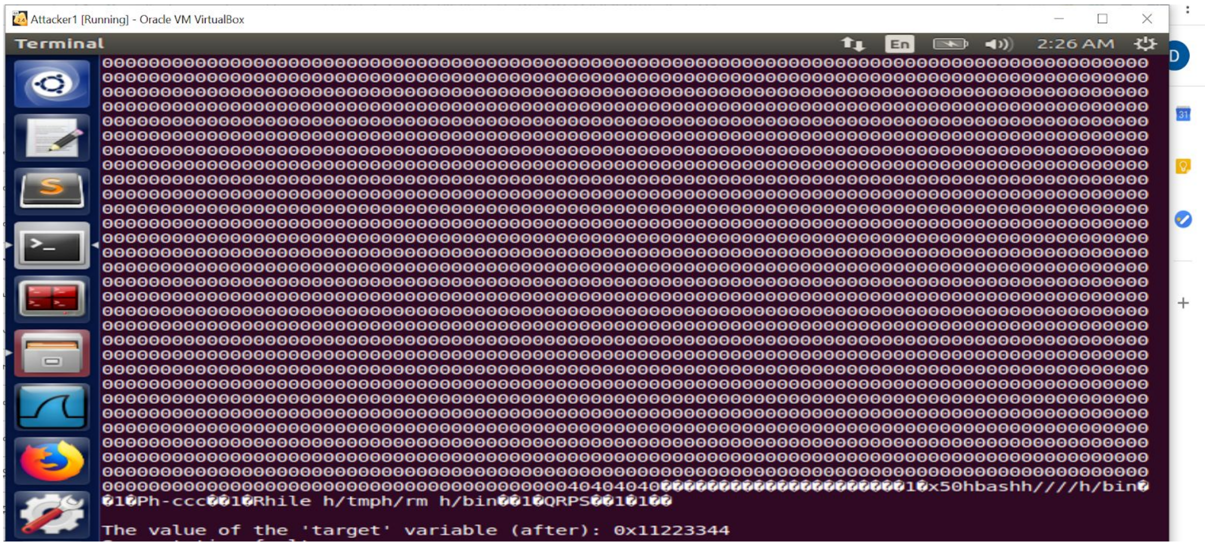
Task 6: Inject Malicious Code into the Server Program



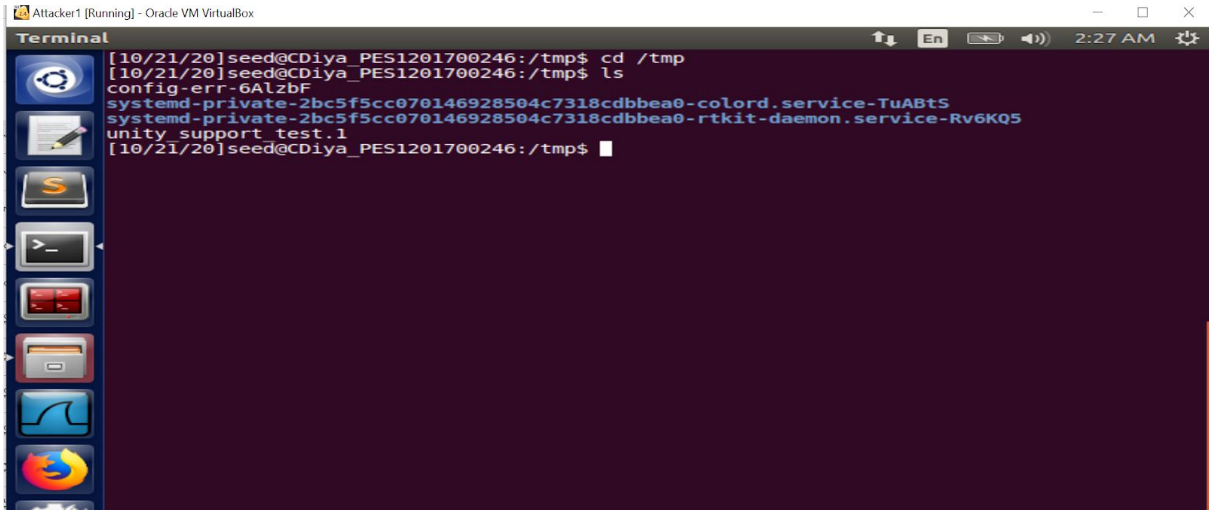
Observation : The screenshot above shows the creation of myfile in the tmp directory. The ls command shows that myfile was created.



Observation : The screenshot above shows the input string to remove the myfile created from the client machine

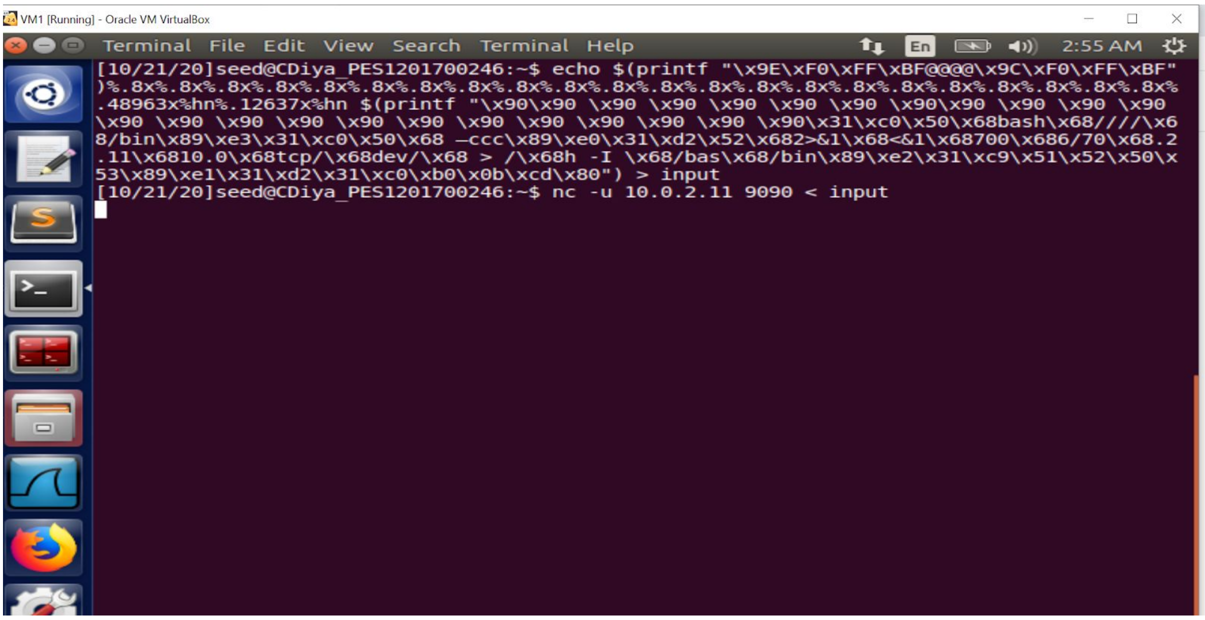


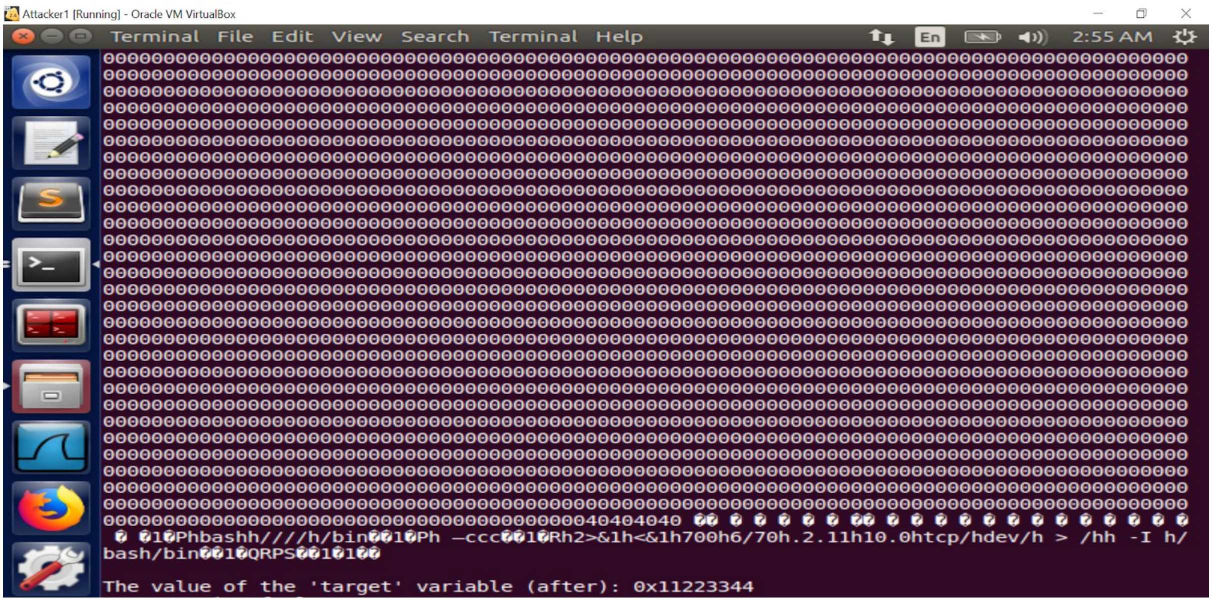
Observation : The screenshot above shows the output of the server.c execution on running the input command.



Observation : The screenshot above shows that when the ls command is run in the tmp directory on the server machine, the myfile is not present because it has been removed by the shell script run on the client. Thus, the attack was successful.

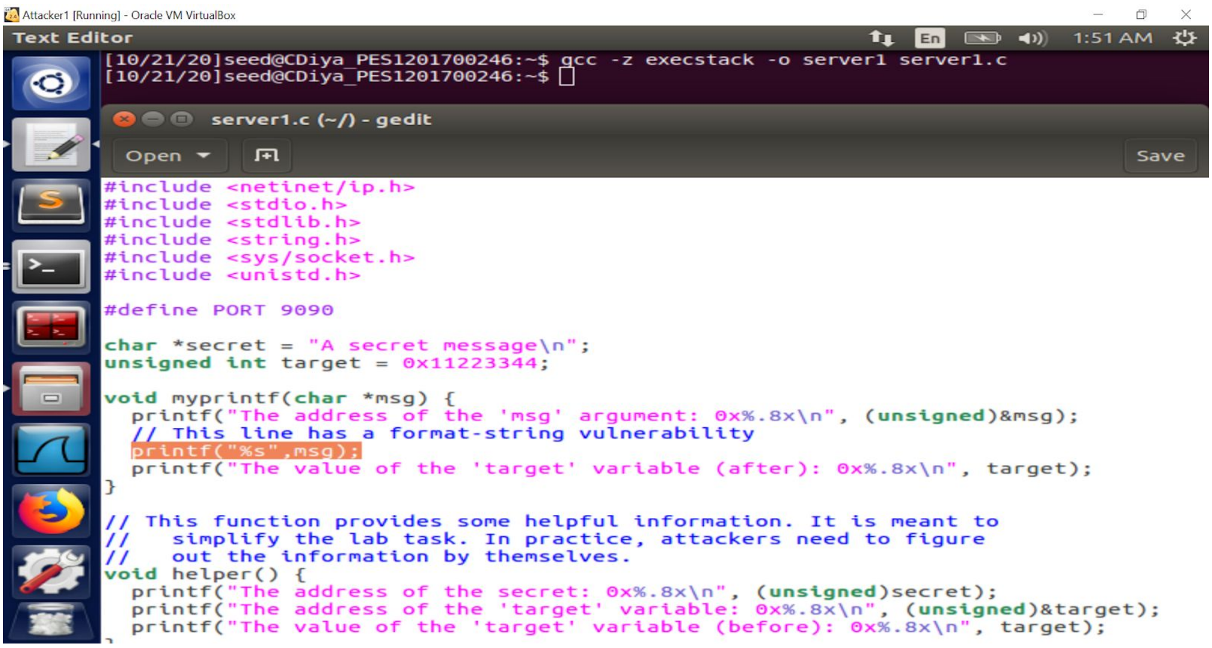
Task 7: Getting a Reverse Shell



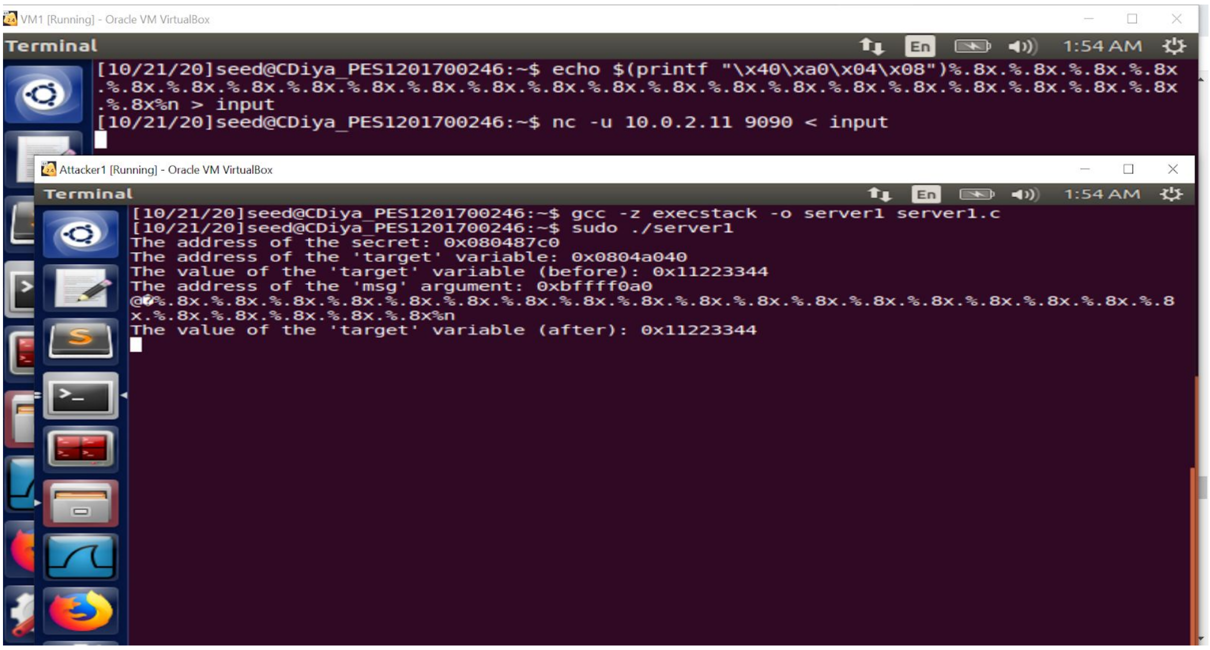


Observation : The screenshot above shows that running a TCP server that is listening to port 7070 on the attacker's machine and then entering this format string successfully achieved the reverse shell because the listening TCP server now is showing what was previously visible on the server. The connection to the VM is shown in the screenshot above. Thus, the #(root VM) has been obtained and the reverse shell attack has been successful.The reverse shell allows the victim machine to get the root shell of the server as indicated by # as well as root@VM.

Task 8: Fixing the Problem



Observation : The screenshot above shows the modification of the printf in server.c and adding a format specifier %s while printing msg. On compiling the modified code, no warning is generated and the code has compiled successfully since the format specifier was added. Thus, the problem has been fixed.



Observation : The input string is entered.
The screenshot above that the attack is not successful and the input is considered entirely as a string and not a format specifier anymore. The printf() in the server.c program printed the program input as a string.
Thus, adding the format specifier has removed the vulnerability.