Amir Mansha

CYSE 211

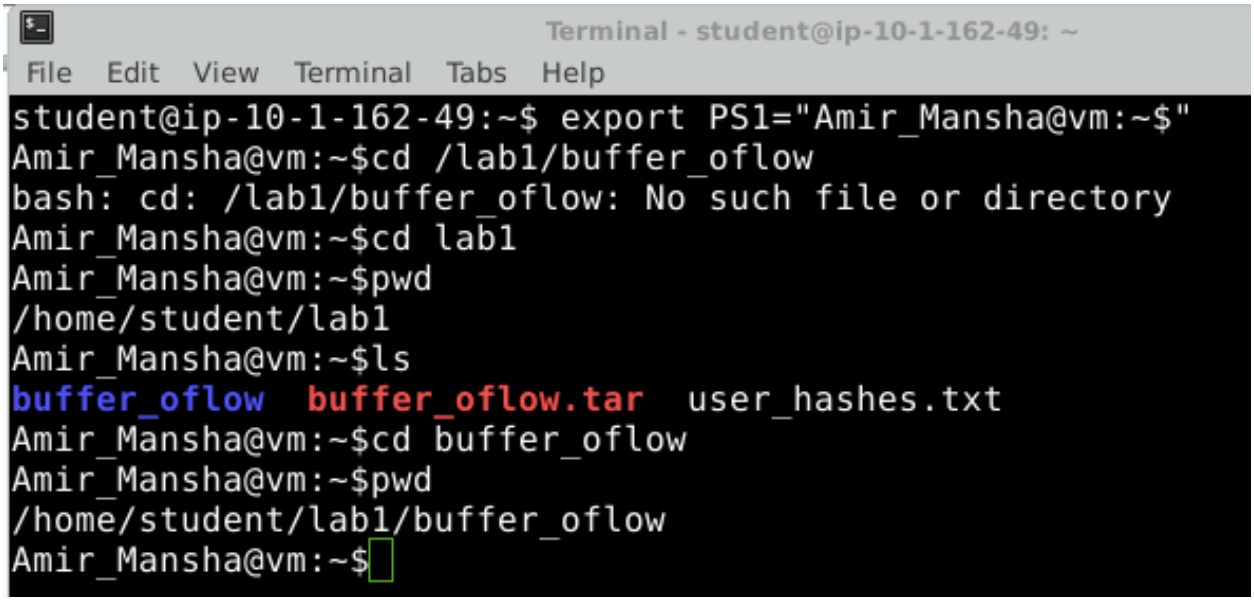
Buffer Overflow Lab

Professor Gebril

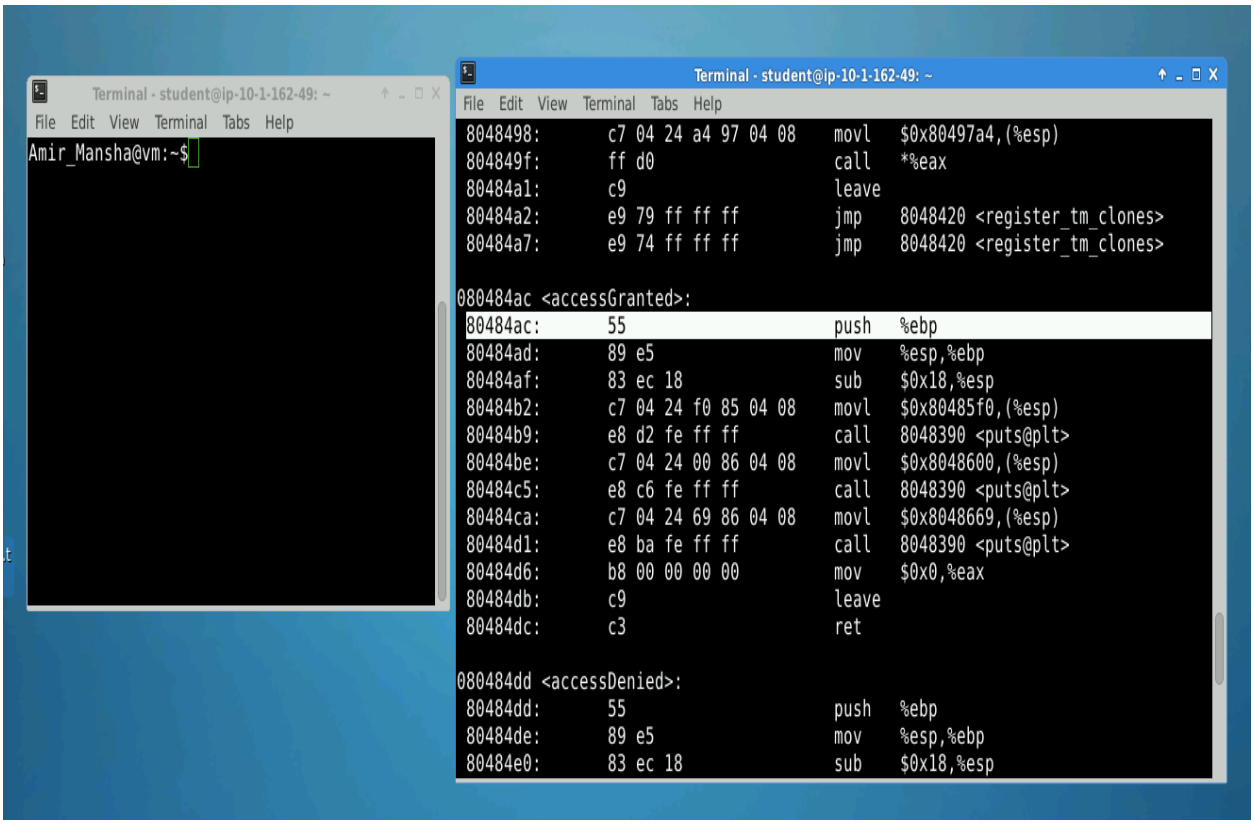
Task 1

In this task we want to get the base pointer address “ebp” so we can use that in the next tasks to overflow the buffer. To do find the address in both buffer_oflow1 and buffer_oflow2 I used the “odjdump -d” command. You can see in the screenshots below where I highlighted the address.

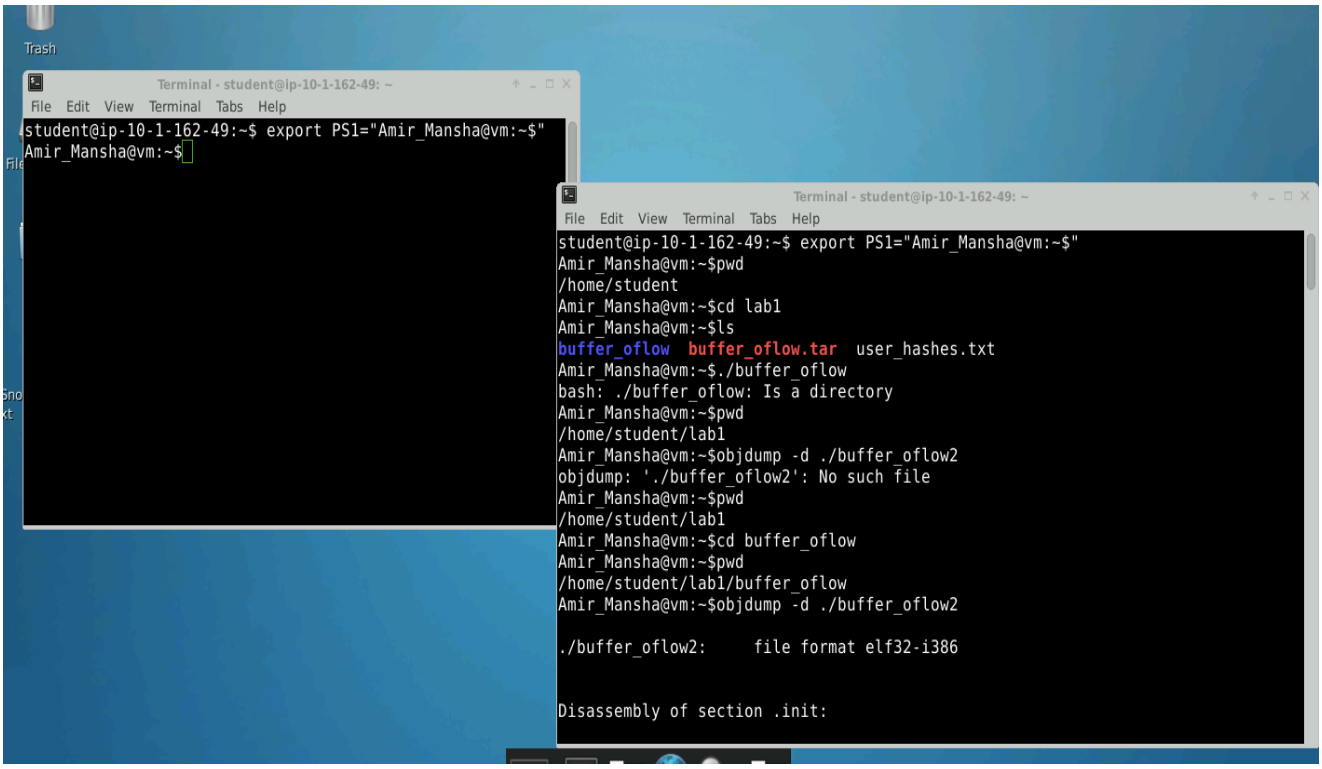
Firstly, I changed my directory to “lab1” and I extracted the buffer_oflow folder from the files application.



Buffer_oflow1: the address is 8048ac



Buffer_oflow2: the address is 8048c5



Terminal - student@ip-10-1-162-49: ~

File Edit View Terminal Tabs Help

```
student@ip-10-1-162-49:~$ export PS1="Amir_Mansha@vm:~$"
Amir_Mansha@vm:~$
```

Terminal - student@ip-10-1-162-49: ~

File Edit View Terminal Tabs Help

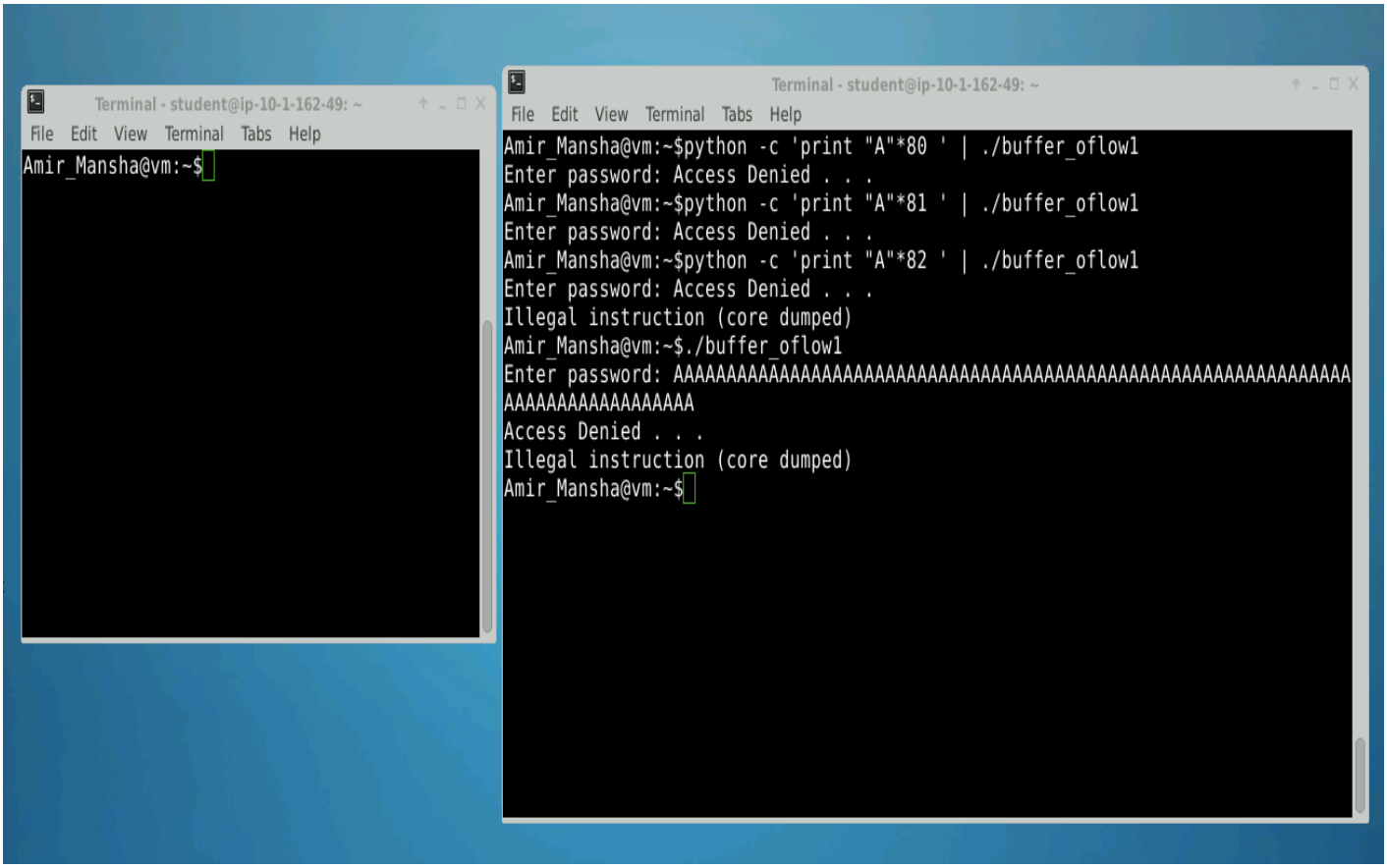
```
 80484b2:       c7 04 24 e0 86 04 08    movl   $0x80486e0,(%esp)
 80484b9:       e8 d2 fe ff ff          call   8048390 <puts@plt>
 80484be:       b8 00 00 00 00          mov    $0x0,%eax
 80484c3:       c9                      leave
 80484c4:       c3                      ret

080484c5 <accessGranted>:
 80484c5:       55                      push   %ebp
 80484c6:       89 e5                   mov    %esp,%ebp
 80484c8:       57                      push   %edi
 80484c9:       53                      push   %ebx
 80484ca:       83 ec 50                sub    $0x50,%esp
 80484cd:       8d 55 c6                lea    -0x3a(%ebp),%edx
 80484d0:       bb 32 00 00 00          mov    $0x32,%ebx
 80484d5:       b8 00 00 00 00          mov    $0x0,%eax
 80484da:       89 d1                   mov    %edx,%ecx
 80484dc:       83 e1 02                and    $0x2,%ecx
```

Task 2: In this task we need to figure out how many bytes or As we need to overflow the input buffer for both the programs. We find this out by using the python command “ python -c ‘print “A”*x ‘ | ./buffer_oflowx.

I started from 10 and then I kept adding 10’s for example, “A”*10 then “A”*20 then “A”*30 and so on until it gave me “segentation fault (core dumped” which means I have more As than needed.

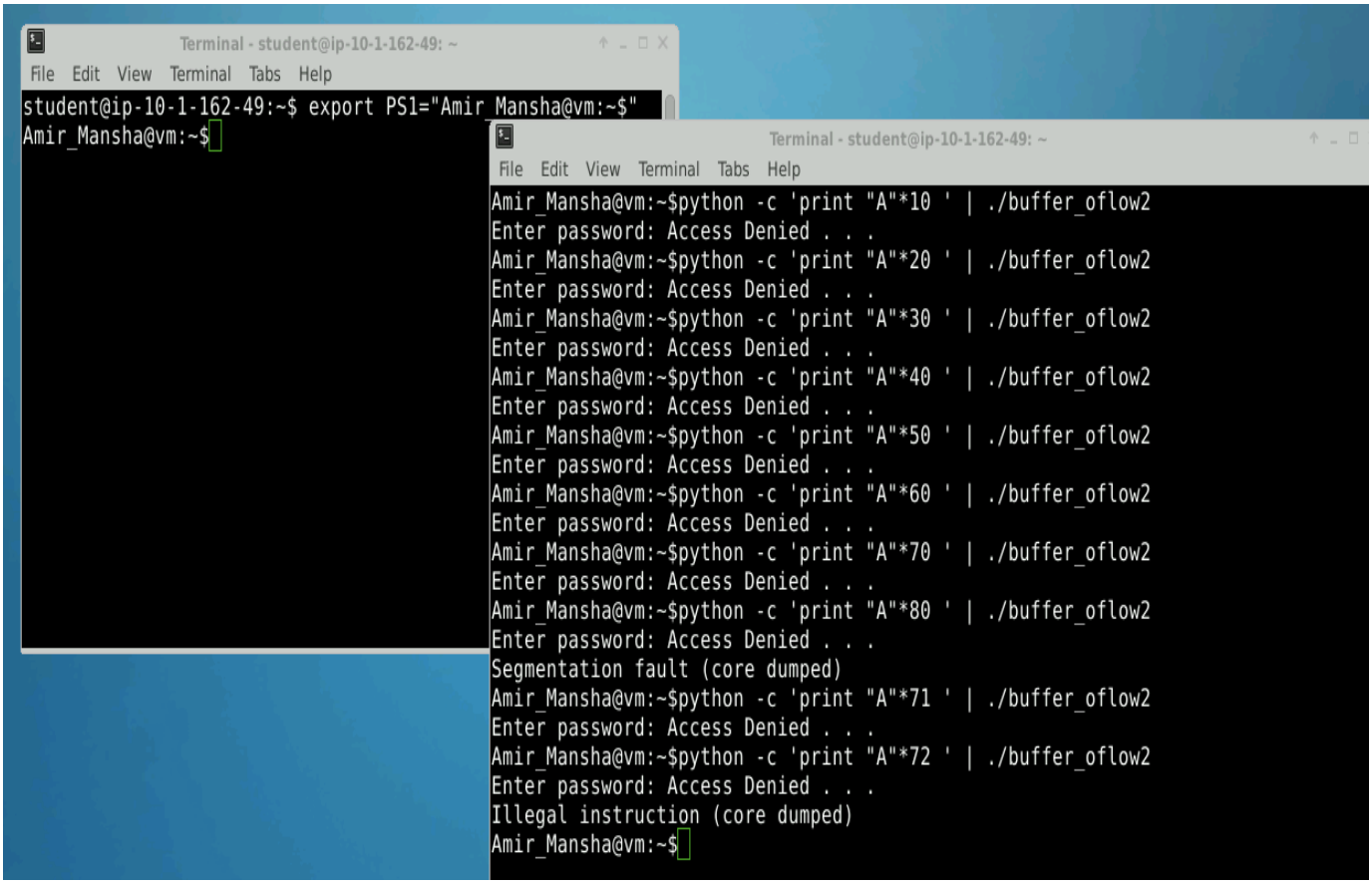
So, for buffer_oflow1 I guessed between 80 – 88 because 90 A’s were more than needed. I guessed 80 then 81 and then 82 where it finally gave me the “illegal instruction” (core dumped) which means that I am trying to overflow the buffer.



```
Amir_Mansha@vm:~$
```

```
Amir_Mansha@vm:~$python -c 'print "A"*80 ' | ./buffer_oflow1
Enter password: Access Denied . . .
Amir_Mansha@vm:~$python -c 'print "A"*81 ' | ./buffer_oflow1
Enter password: Access Denied . . .
Amir_Mansha@vm:~$python -c 'print "A"*82 ' | ./buffer_oflow1
Enter password: Access Denied . . .
Illegal instruction (core dumped)
Amir_Mansha@vm:~$./buffer_oflow1
Enter password: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
Access Denied . . .
Illegal instruction (core dumped)
Amir_Mansha@vm:~$
```

For buffer_oflow2, I repeated the same steps as buffer_oflow1, however, at 80 A's I got the segmentation fault (core dumped) message so I knew it could be between 70-79. I guessed 70, 71, and finally when I tried 72 A's I got the illegal instruction message which means I am on the right path to overflow the buffer.

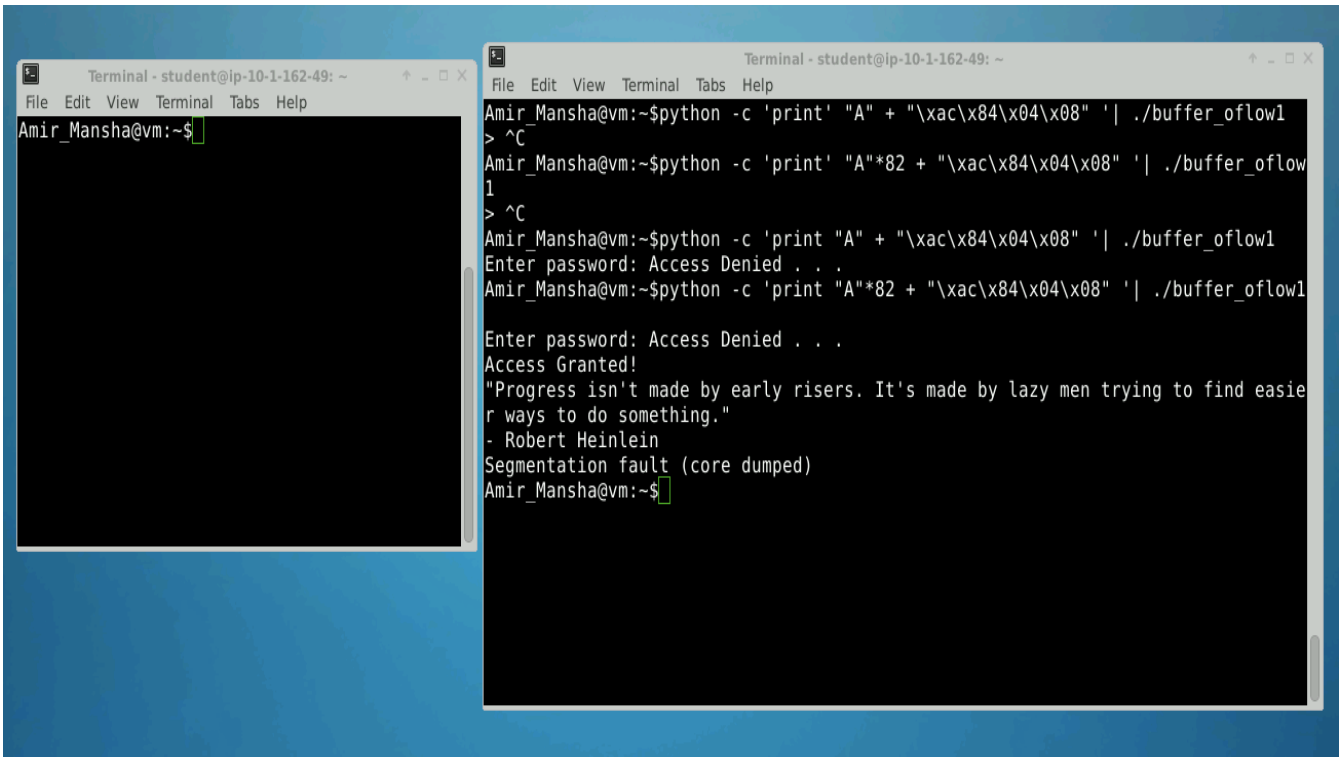


Task 3: In this task, we will use the base pointer address to try to overflow the buffer and get the “Access Granted!” message without entering the password.

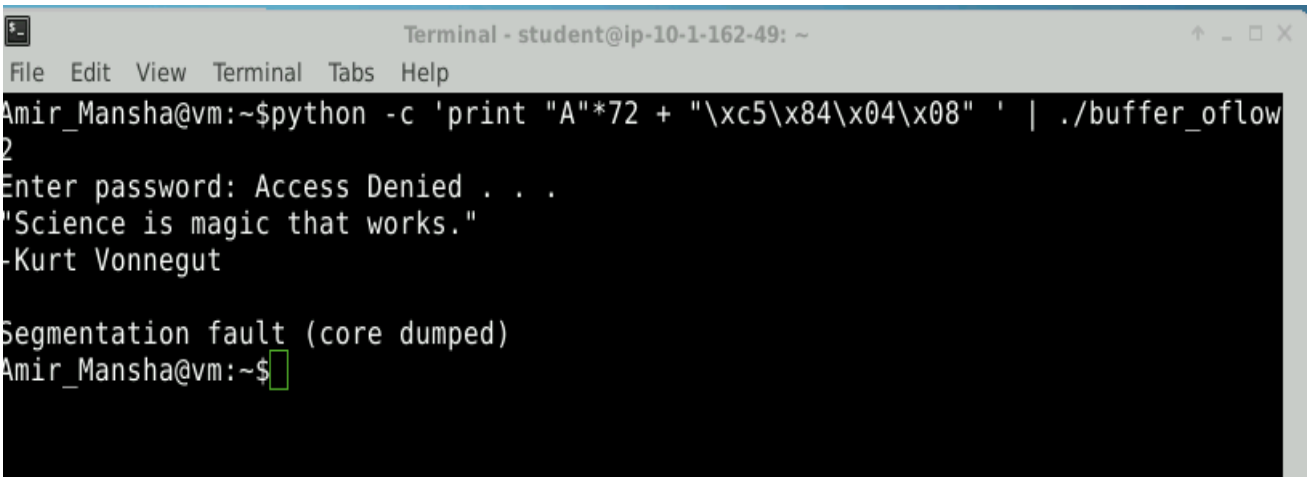
In order to do that, first, we have to convert the base pointer address to little endian format.

We have to use the similar command but just add the “ebp” in the little endian format as show in the screenshots.

For buffer_oflow1, it would be \xac\x84\x04\x08 because the ebp was 8048ac.

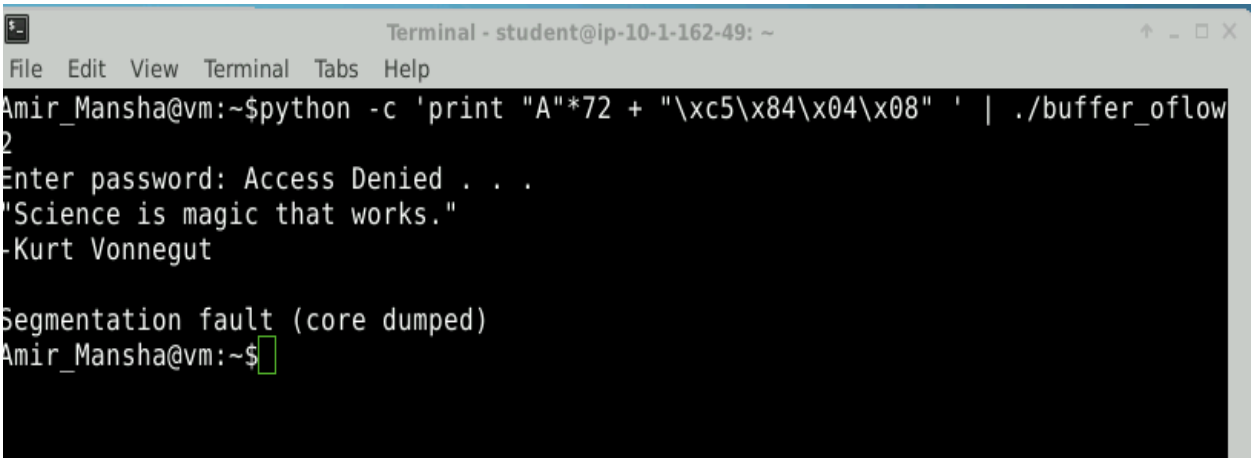


For buffer_oflow2, it would be \xc5\x84\x04\x08 because the ebp was 8048c5.



Task 4: In this task, once we have successfully accessed the program without entering the password, we will see a quote for the buffer_oflow2 program which is: “ Science is magic that works.” – Kurt Vonnegut

The last name of the author quoted in the buffer_oflow2 program is Vonnegut.



Task 5:

We get the segmentation fault (core dumped) after we successfully executed the buffer_oflow2 program without entering the password because we overflow the buffer, so we had more bytes or A's to overflow the password and when we do that the program tries to read/write outside the memory which is the illegal memory location. Essentially it is when the program tried to access somewhere in the memory that is not allowed.