HW3:

Task 1:

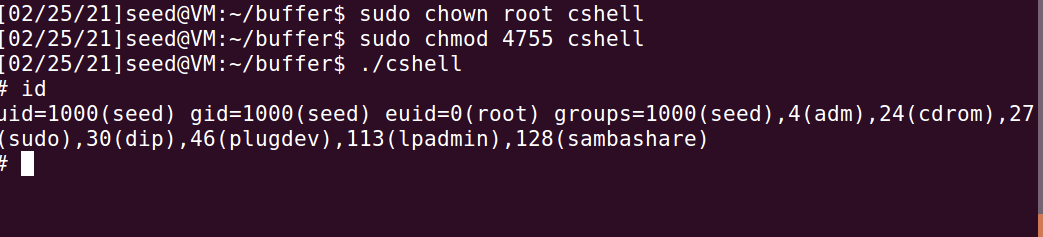
The shell has been saved as a .c file. After compiling and executing the code we get the shell

And compiled using the command gcc cshell.c -o cshell. It launches the shell

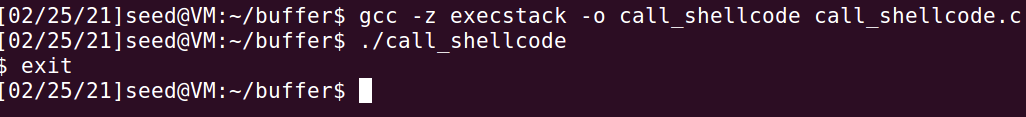


We make it a setuid root program. While executing it gives us the root shell where euid=0. Which means we have the root access.

.



Now when we compile the shell code of the same code above and execute it it gives us the same output. It launches the shell.



B) vulnerable program

Here we compile the stack.c program and when we execute it without root privileges we were able to exploit the vulnerability. Then we make it a setuid root program and the we have exploited the vulnerability and gain root privaleges.

While executing we get “Returned successfully” as the output.

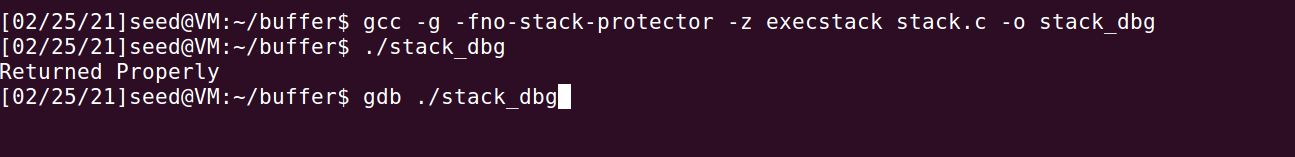
We create the file badfile which contains the malicious code



Task 2:

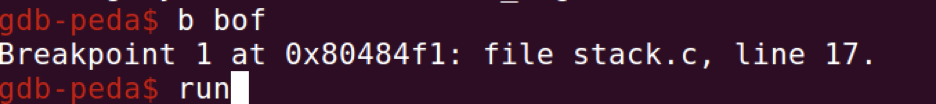
Inorder to find the offset value and return address we use the debug command

.

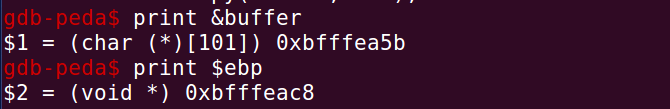


Since bof is the vulnerable function we break the bof using “ b bof” and then run it. This will give us the buffer and ebb value

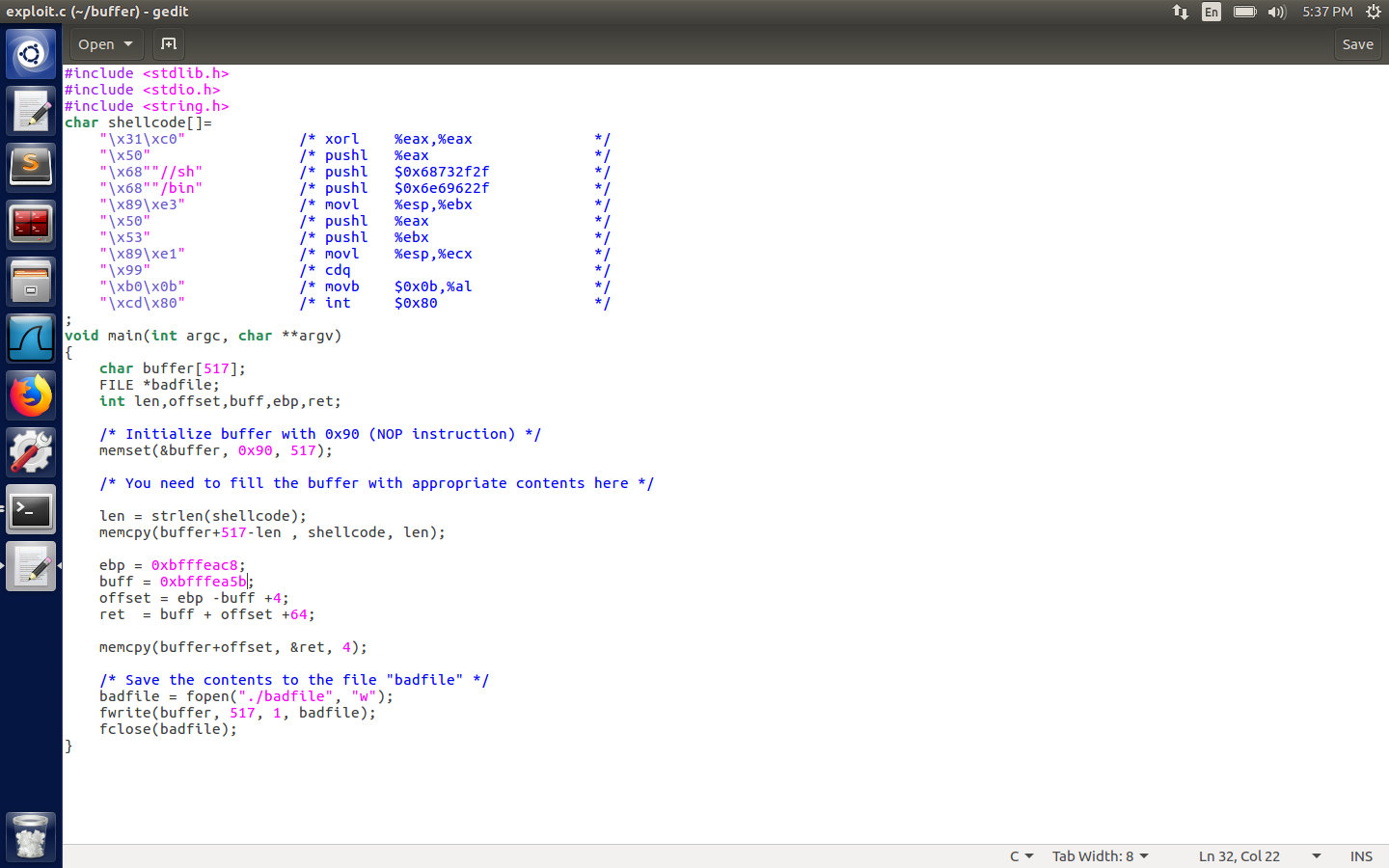
Since the vulnerable function is bof we break bof with command “b bof” and we run it



Since bof is the vulnerable program we make a breakpoint at this function and then we run it which will give us the buffer and ebp value

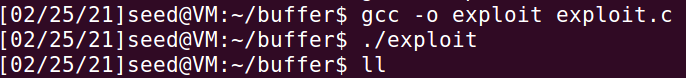


We create a badfile using touch badfile command which is where the malicious command are present.



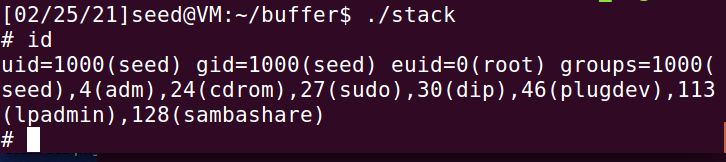
The exploit.c file is modified as as shown in the figure above.

This c program is compiled and executed . here a badfile is created.

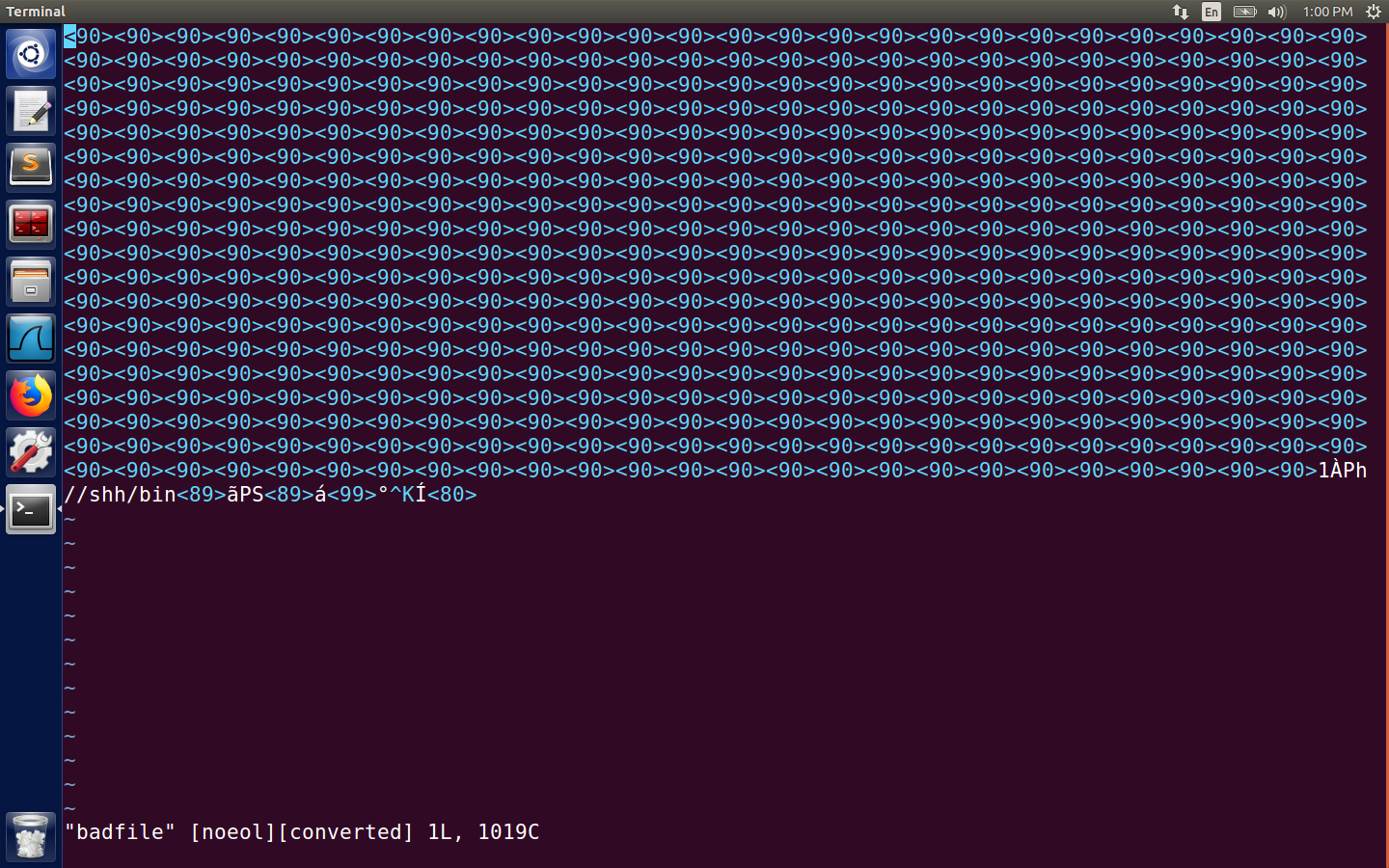


Now execute the stack program and we get root access which means the attack was successful.

Here the root id is euid=0

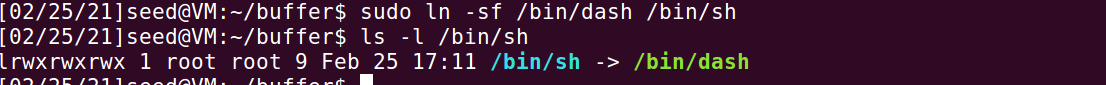


Badfile obtained.

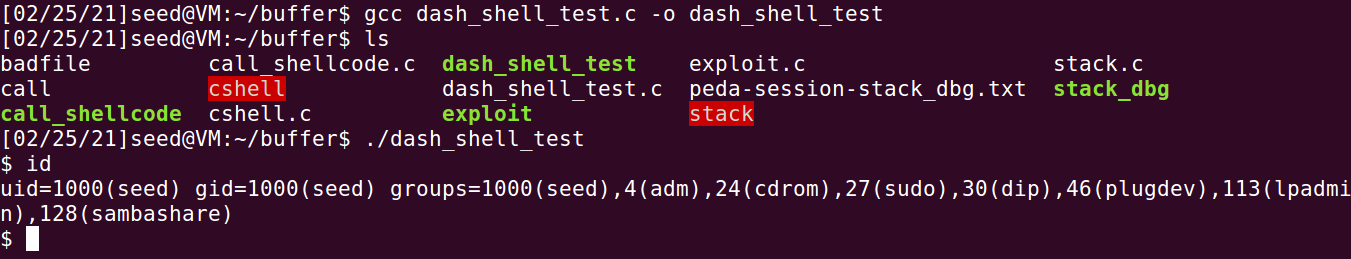


Task3 :

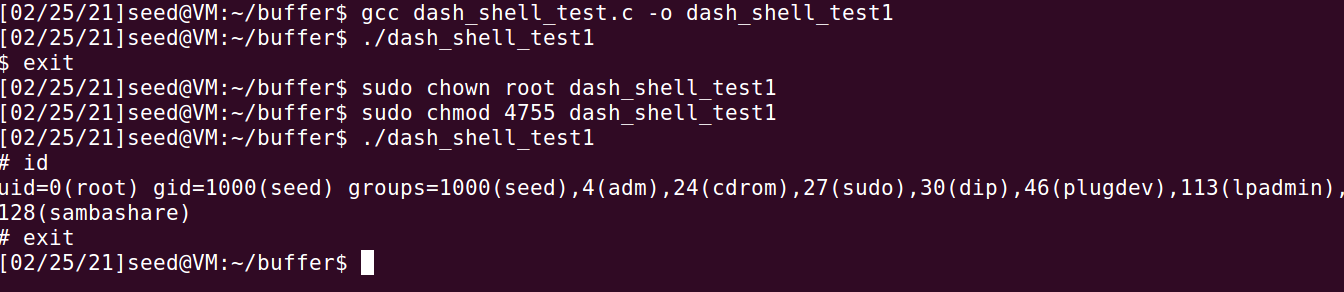
We should first map /bin/sh to /bin/dash



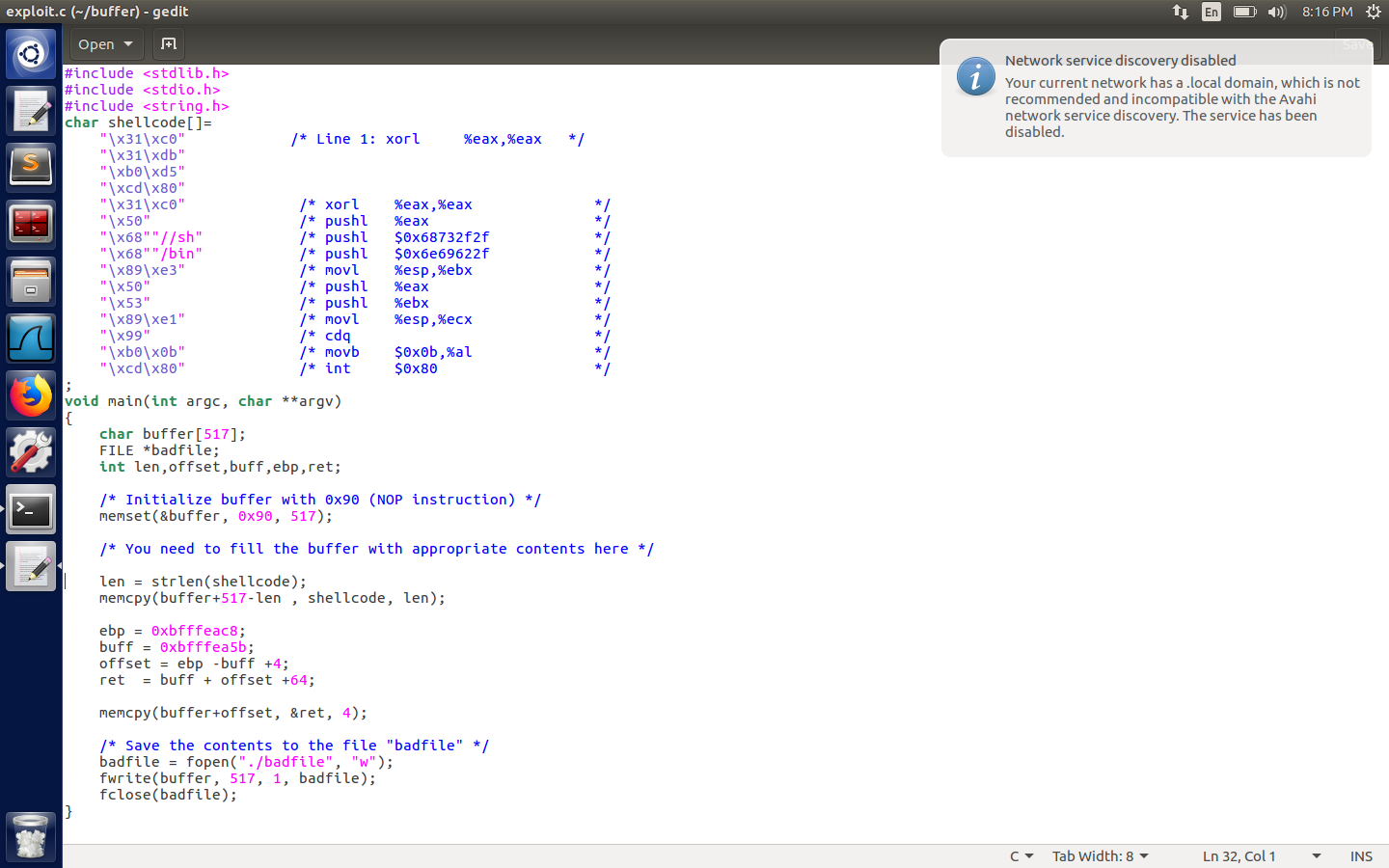
Now we compile the dash\_shell\_test code and execute it. We can see that we have got a shell.Even if it’s a setuid program it doesn’t give us root shell .This is coz the countermeasures work.



.Now we uncomment the setuid=0 command line. We execute it with a different execute file. Set it to setuid program. Now when we execute the program its will give us the root shell.

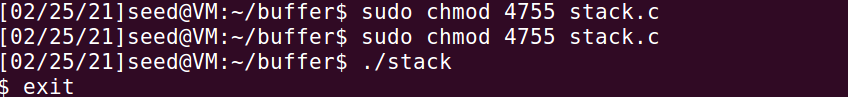


Now we modify the exploit.c as shown in the figure below.



Execute the modified exploit.c file. Compile the stack.c program

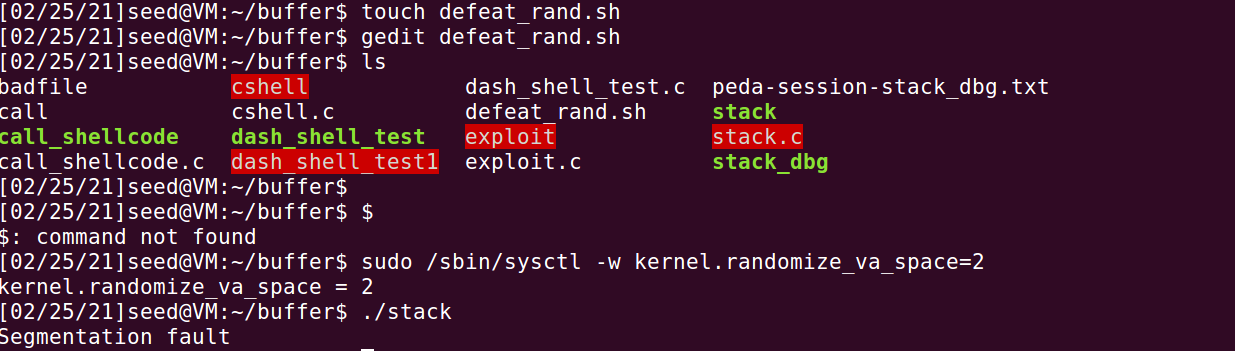
Now change it to a setuid process and execute the stack.c program and we will get the root access where “uid =0”



Task4:

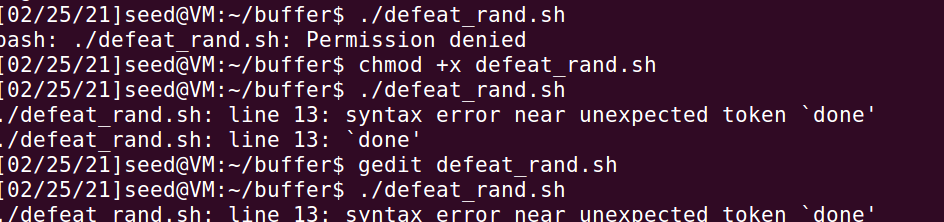
Create a file named defeat-rand.sh. Set space=2 when u set ASLR to defeat counter measure.

When we run stack.c w get segmentation fault.



Run the defeat\_rand.sh and we can see that permission is denied and this is because it doesn’t have executable pemission. Give the permission and execute again.

We get brute force attack



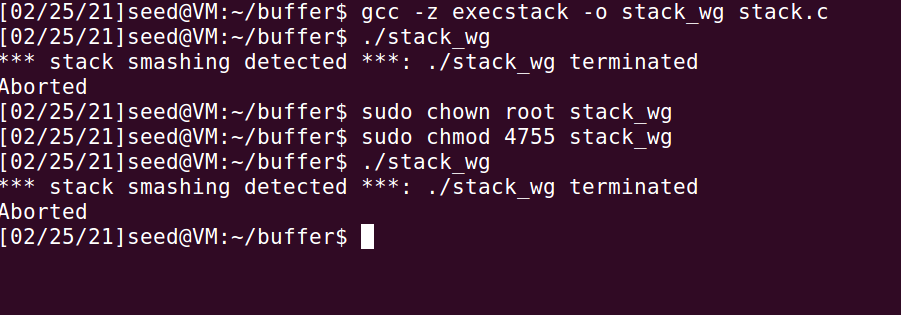
Task 5:

Turn off the memory randomization inorder to perform this task.



Now execute the stack.c program with a different executable . Here turn on the Stack gurad.Once executed they give an error called \*\*\*stack smashing detected\*\*

Now we can change it to setuid program still it would give us the same error. This means that the guard is on and is not able to attack.



Task 6:

Here w turn on the non executable stack protection.



Compile the stack using the above command .



Now lets execute as a normal executable program now. It gives us segmentation fault. Here eventhough there is no buffer overflow it is considered as non executable data. The malicious code is inside the stack.



Now lets change it to a setuid program and it will still give the same result. This means it is imposible to run the shellcode.

