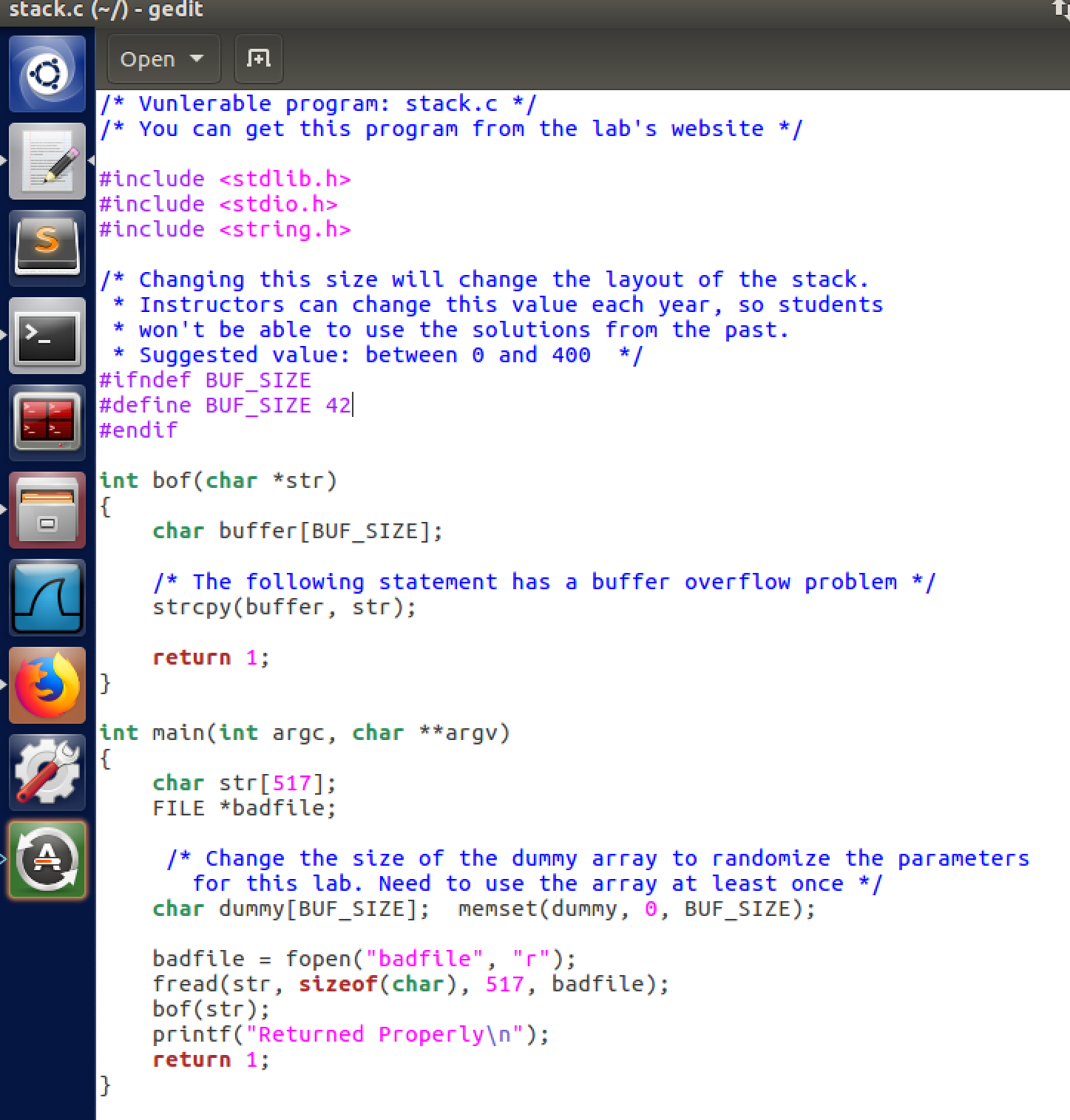
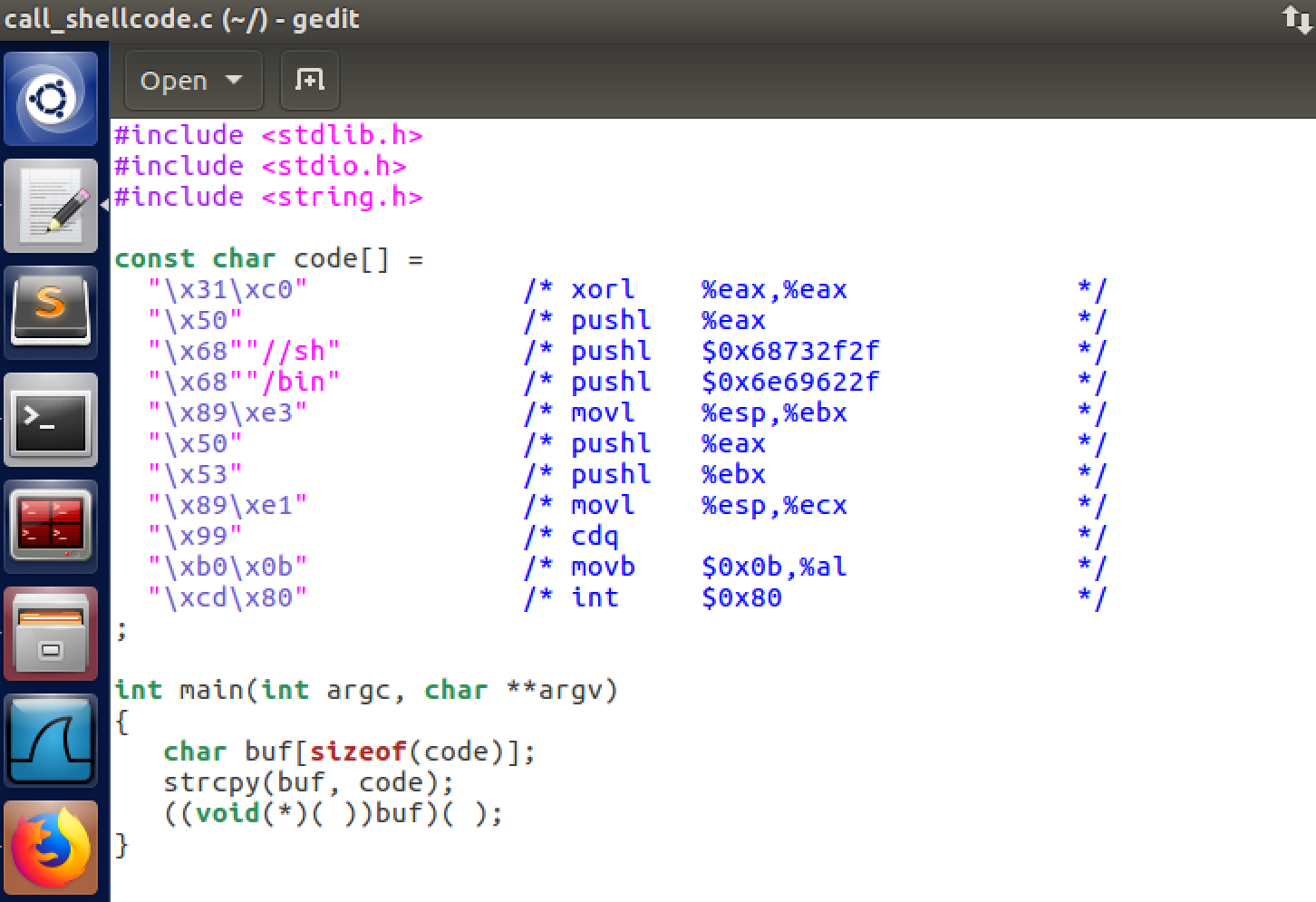
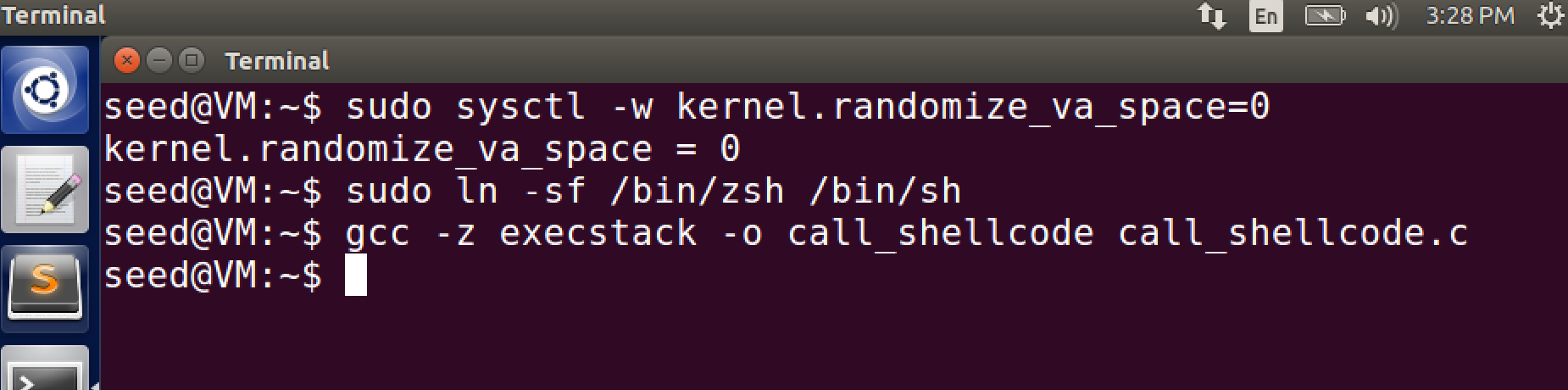
Lab 4

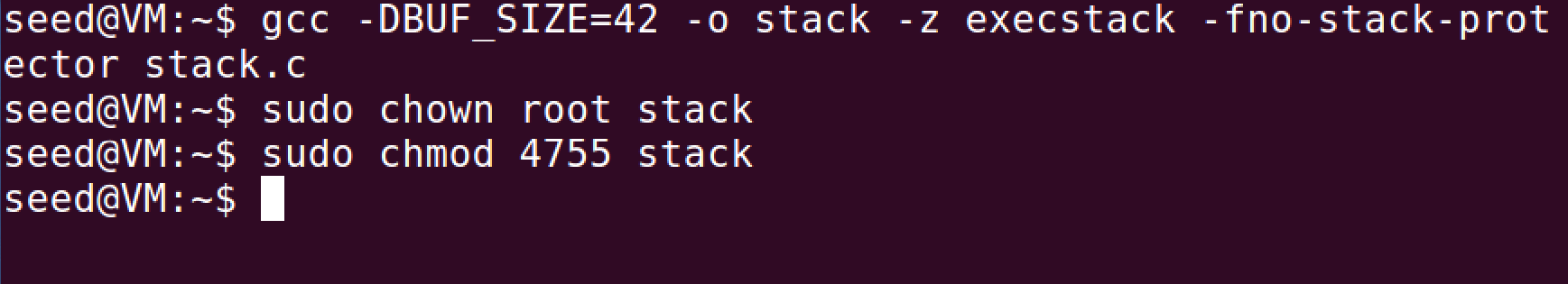
Task 1: Running Shellcode

Code Used: Here this code launches a shell by executing a shellcode stored in a buffer. The second code is what contains the buffer overflow vulnerability.



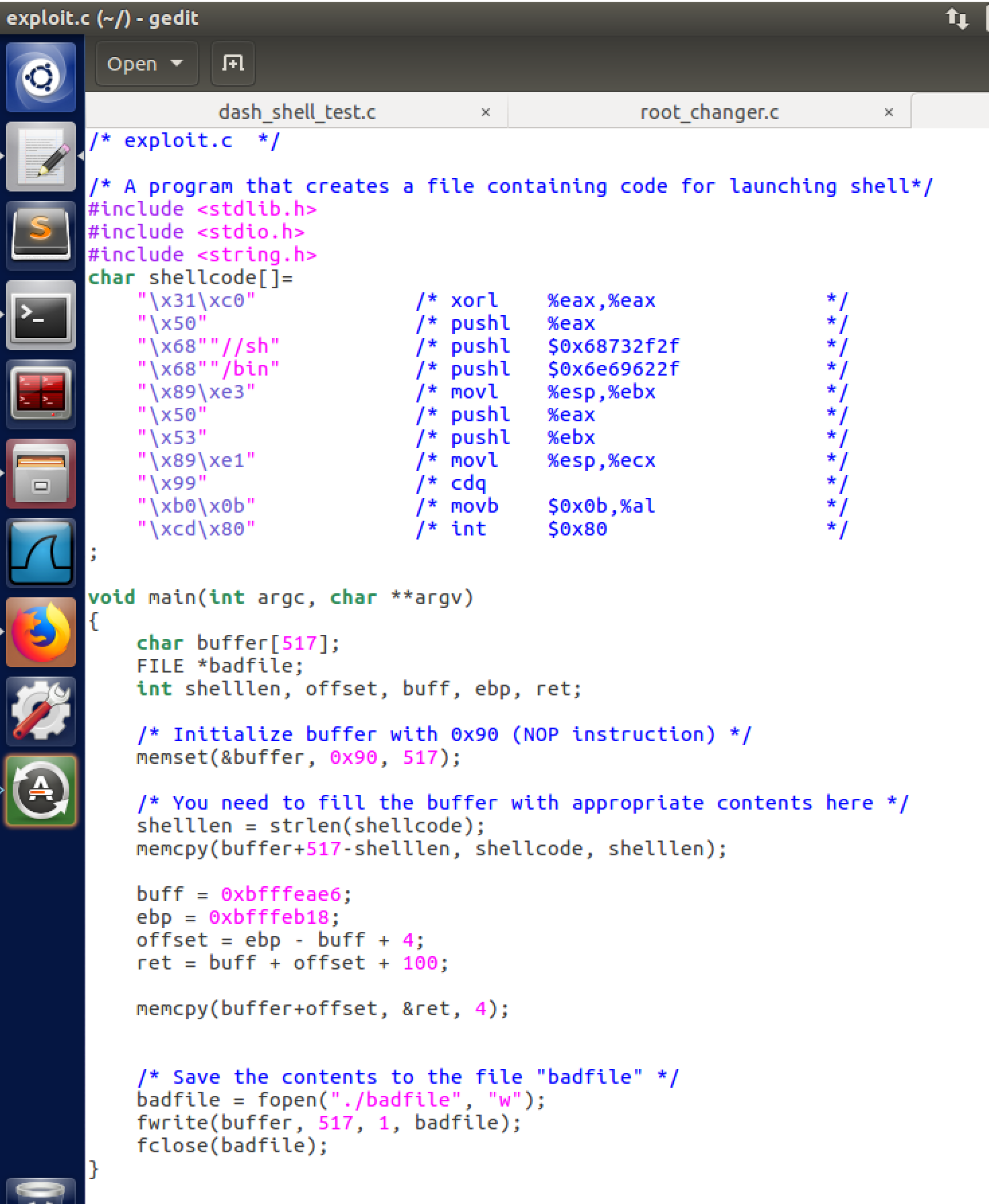
Commands Used: Before task 1 begins it has us disable some countermeasures and configure/bin/sh beforehand, so that is what the first two commands do. The last command is what compiles the code. The second picture shows the command for the second code used in task 1 which has a buffer overflow vulnerability which we will be exploiting later. It also sets

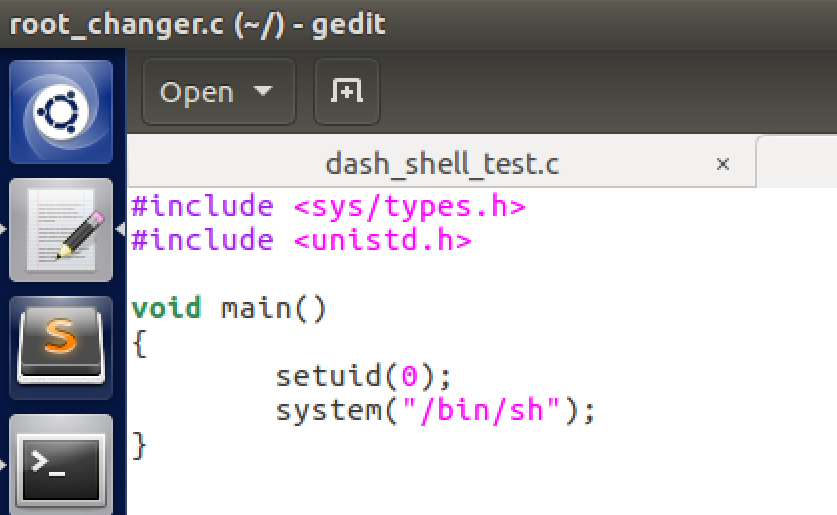




Task 2: Exploiting the vulnerability

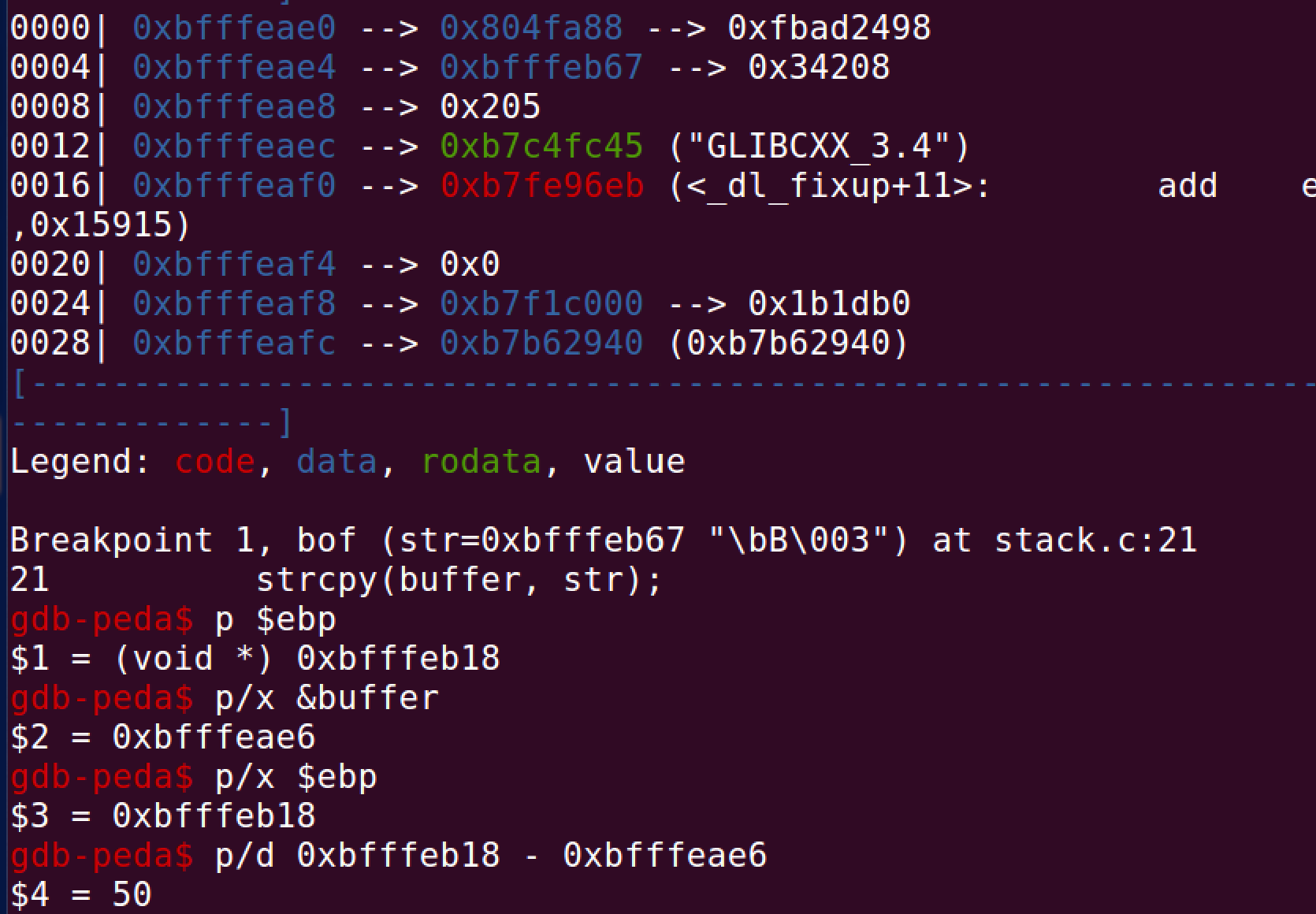
Code Used:

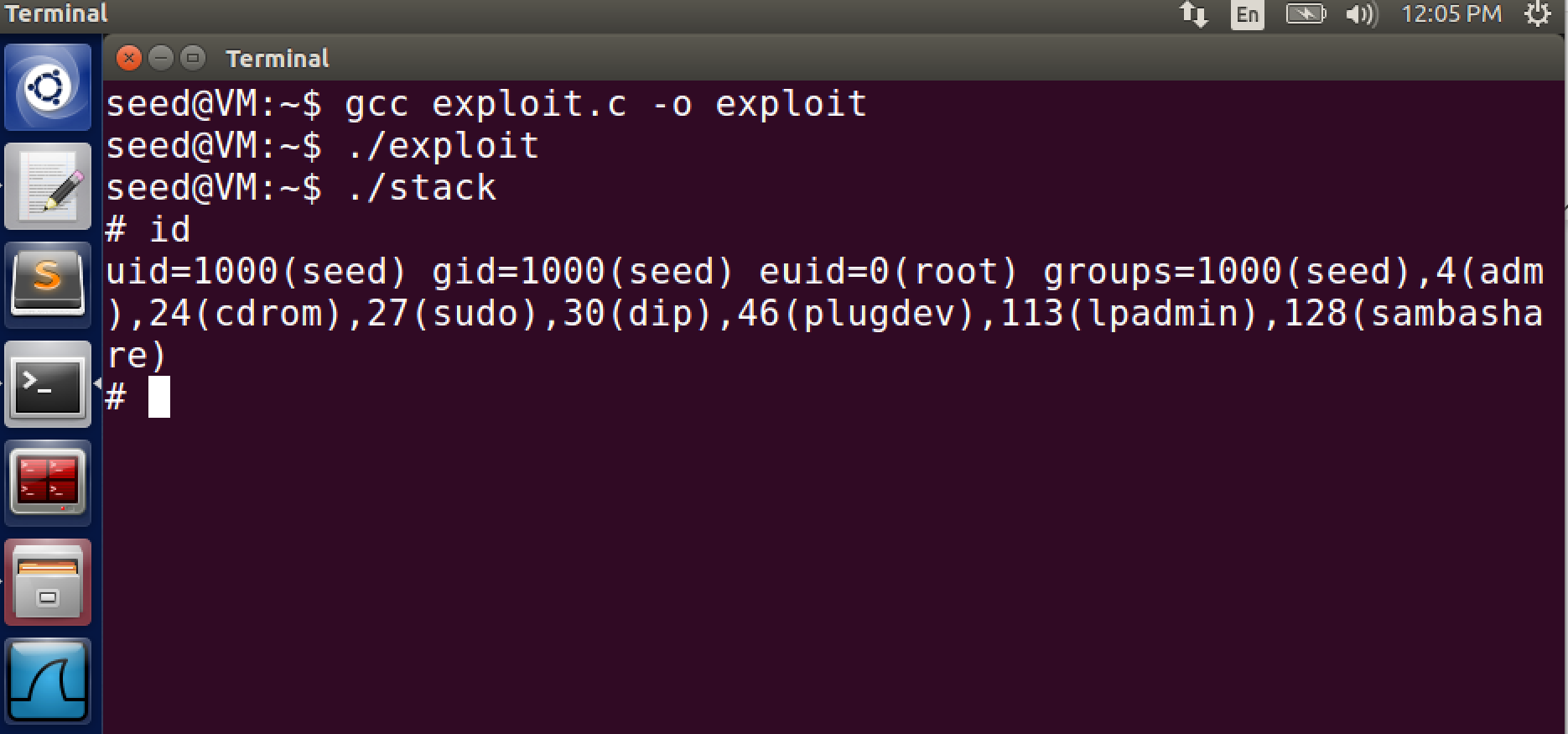


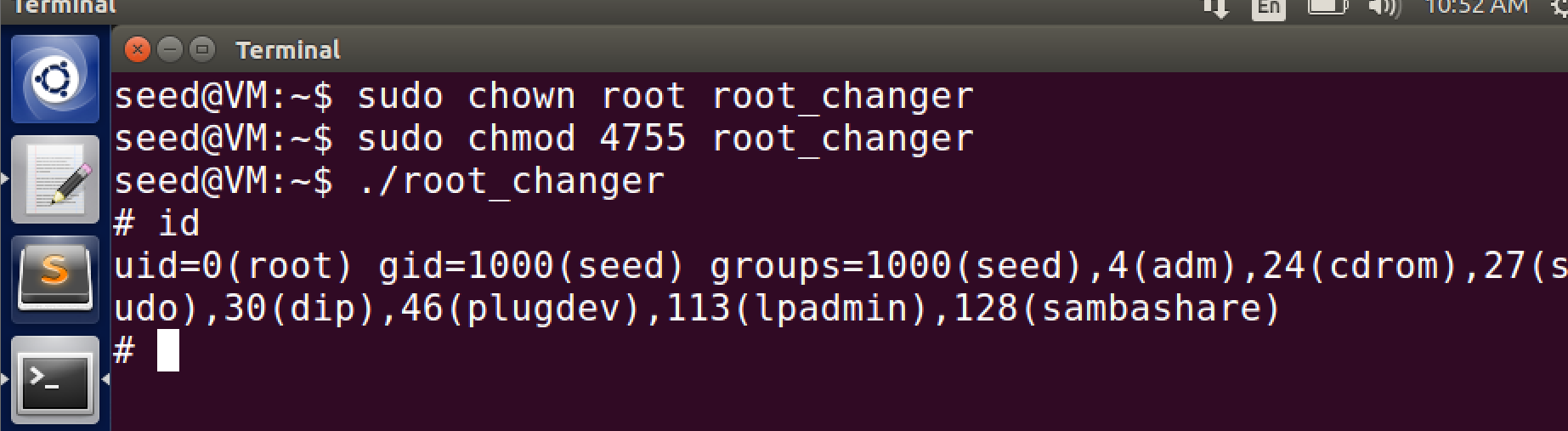


The code for exploit.c had to be modified in order for the program to do the attack. I had to find the addresses for both the buffer and the ebp as well as the distance between them. This allows for the buffer to be overflowed resulting in gaining root privilege over the program.

Commands Used:



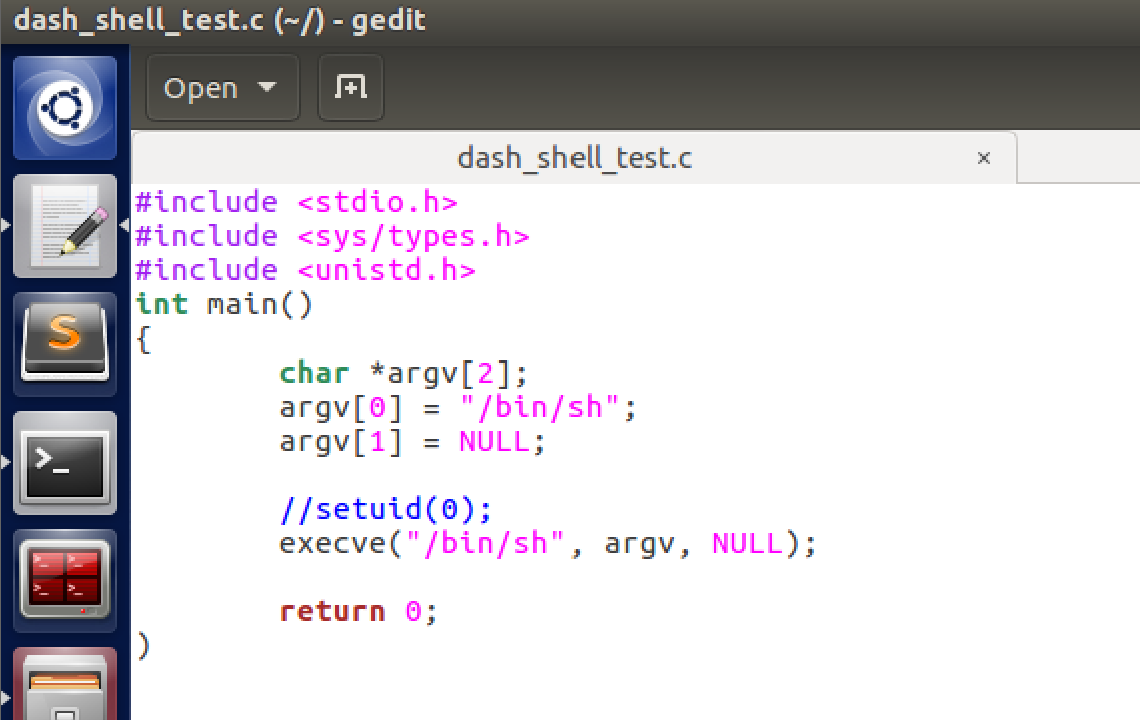


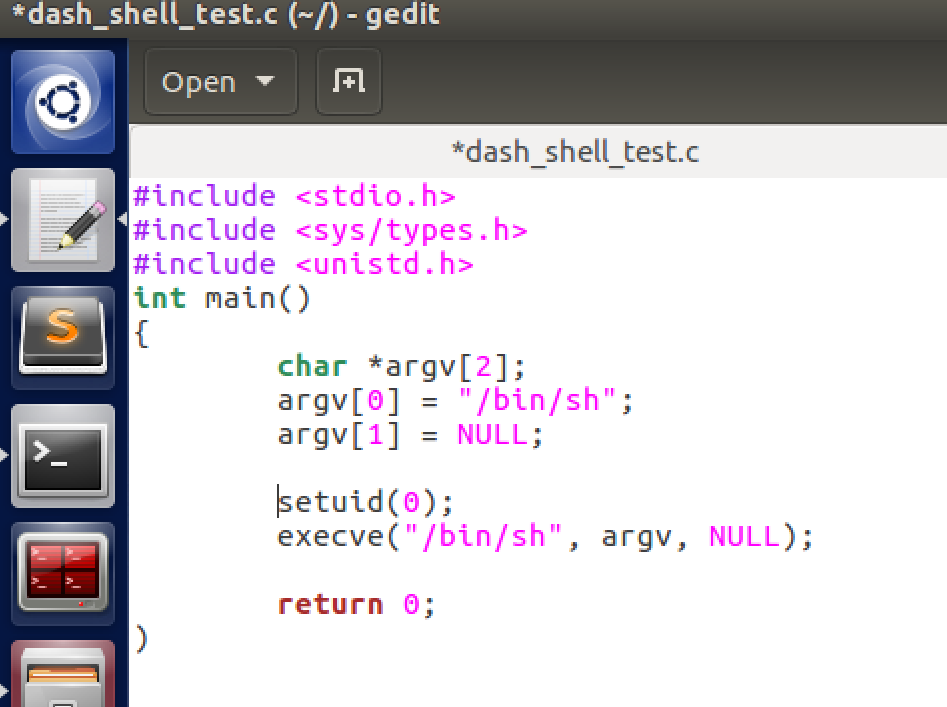


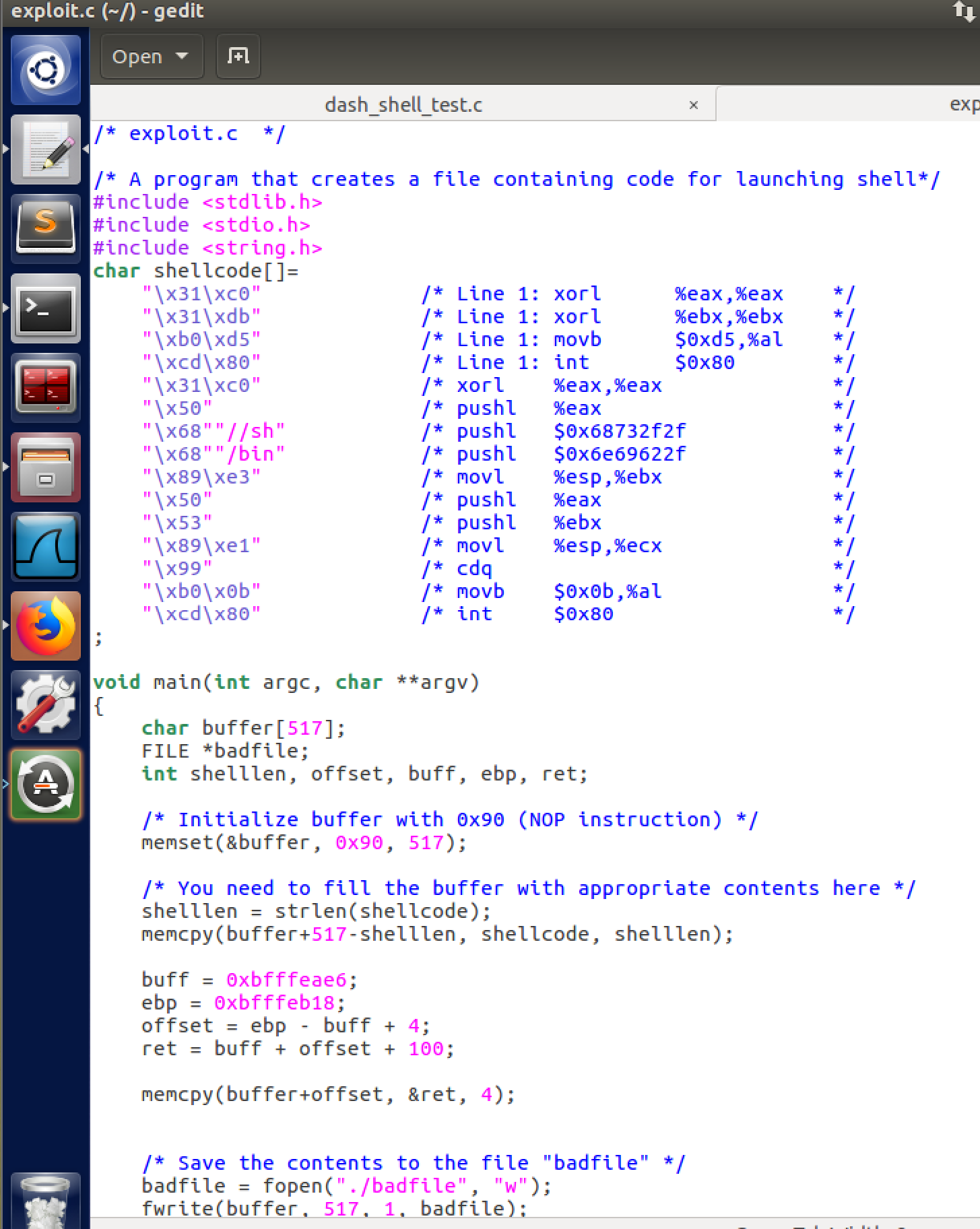
Here I had to find the addresses as well as the difference between them. Tested to make sure the attack worked, and was told to set the uid to root, and when I ran the program exactly as it said in the lab it did change it to root.

Task 3: Defeating dash’s Countermeasure

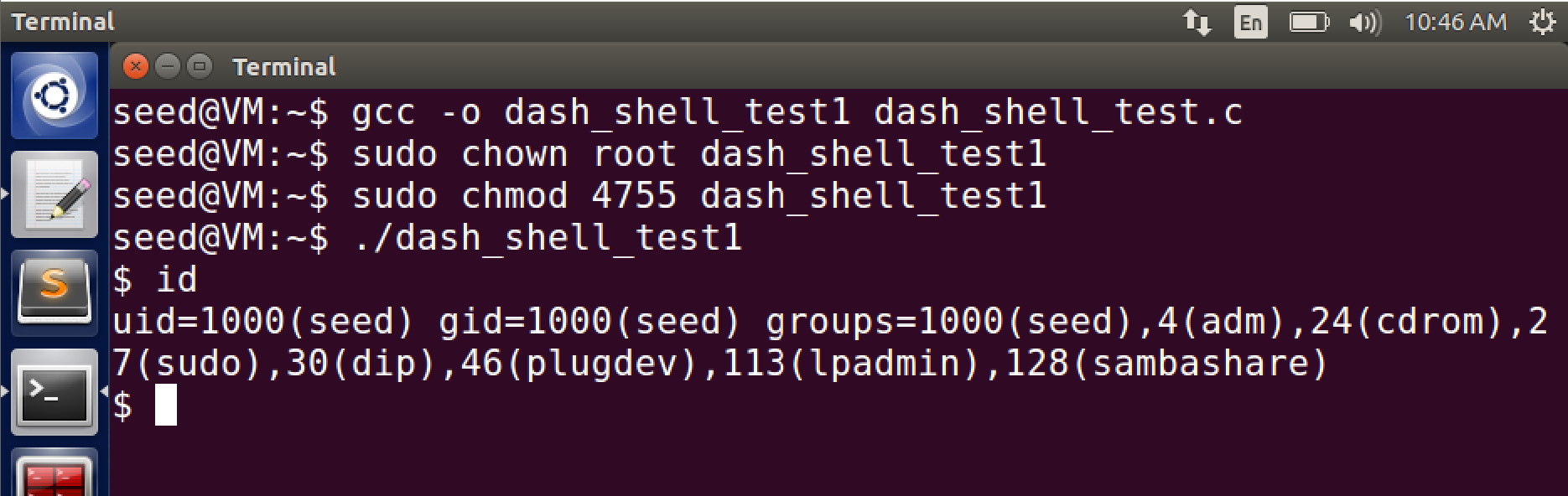
Code Used:



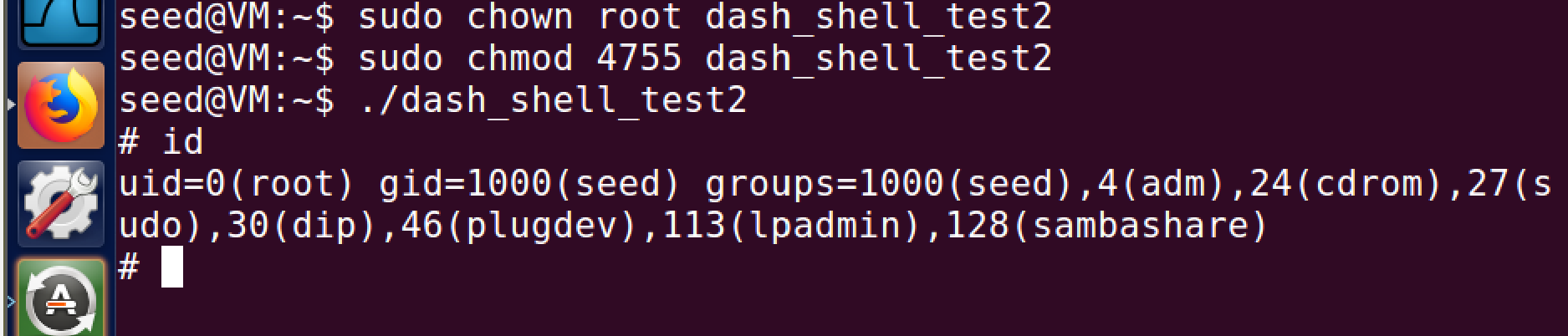


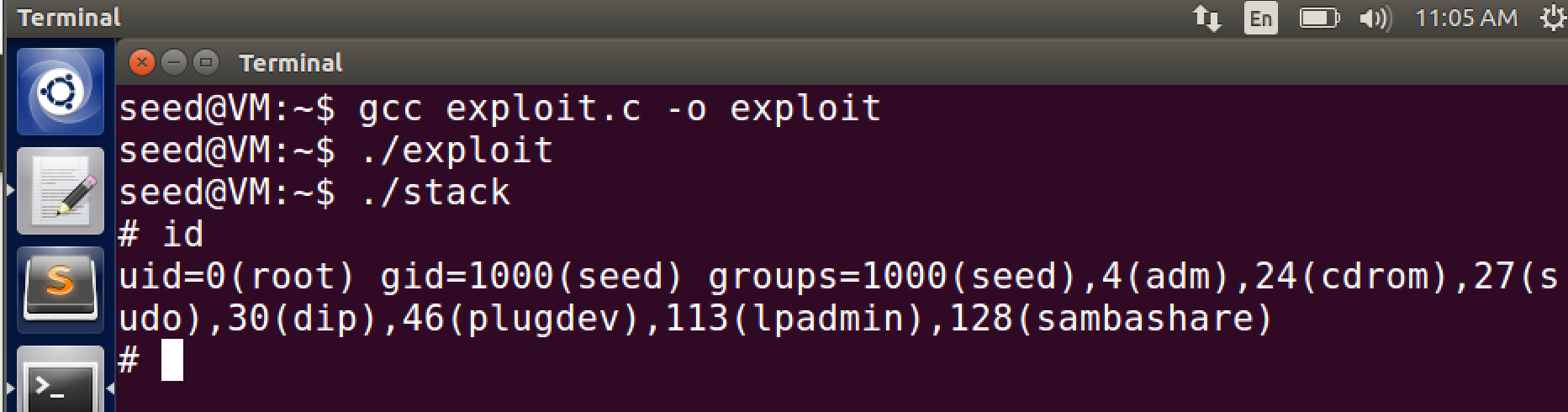


Commands Used:



Without the setuid(0) command it does not set the uid to root

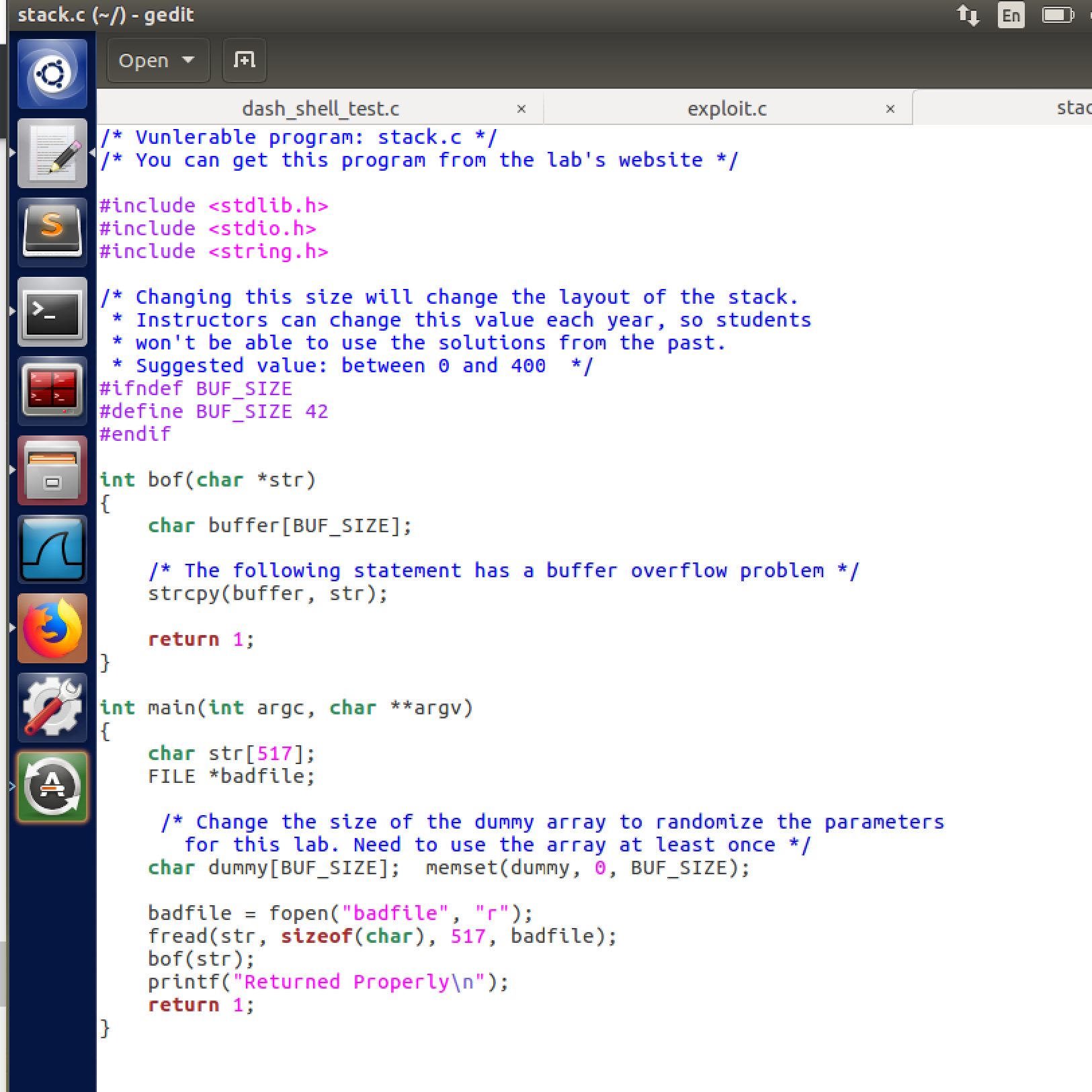


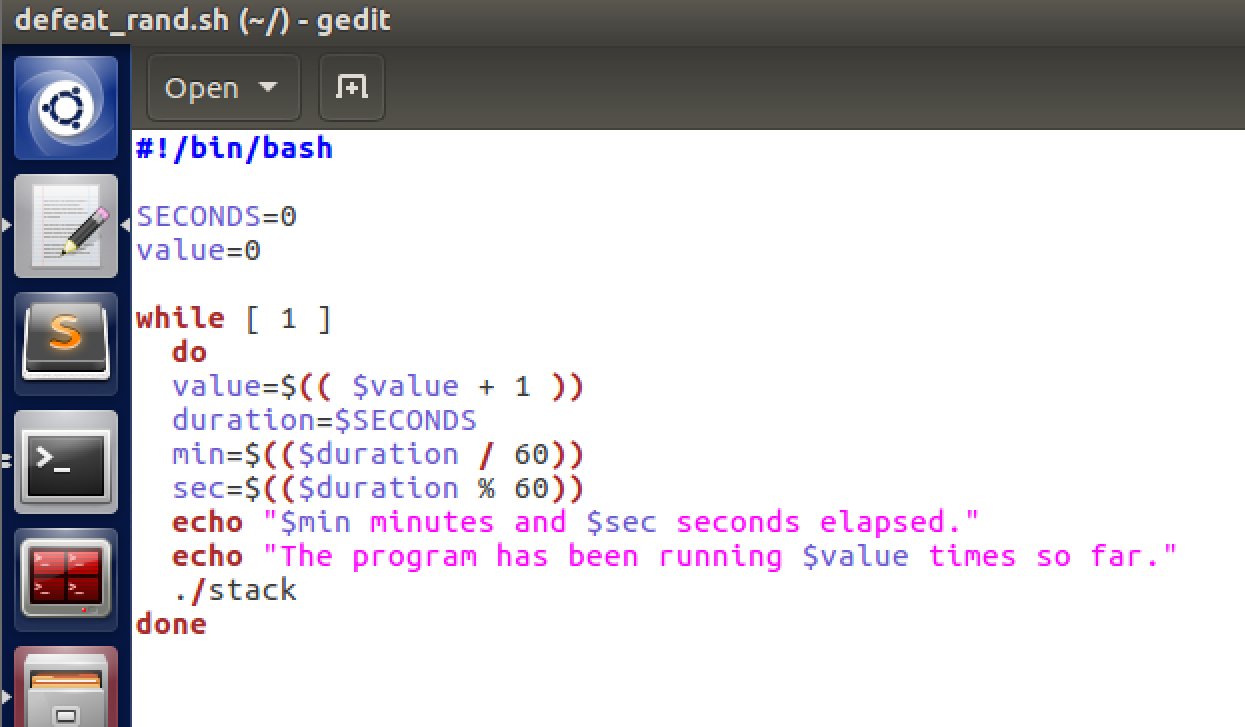
When we run the setuid(0) command it does change the uid to root 

Running the attack once more, we do get the root uid when we didn’t get it last time we ran the attack because the new code we added to stack.c called setuid(0) which allowed it to retrieve the root id.

Task 4: Defeating Address Randomization

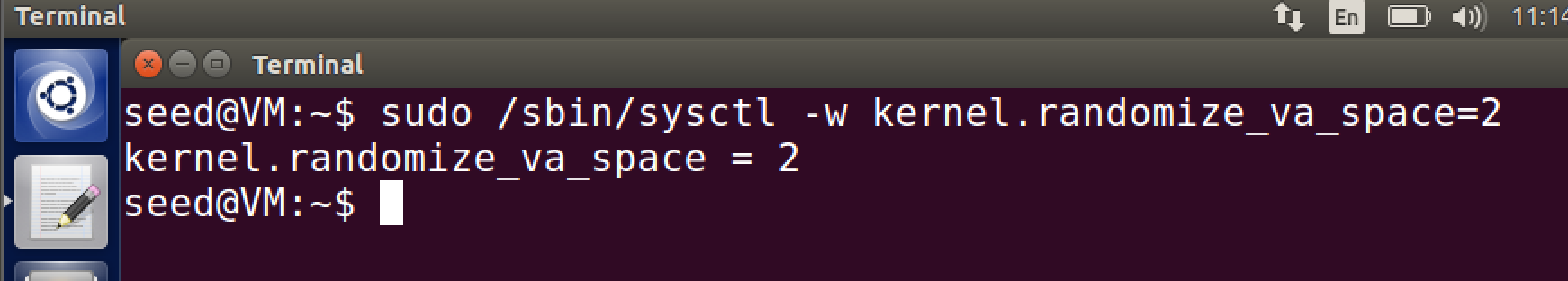
Code Used:

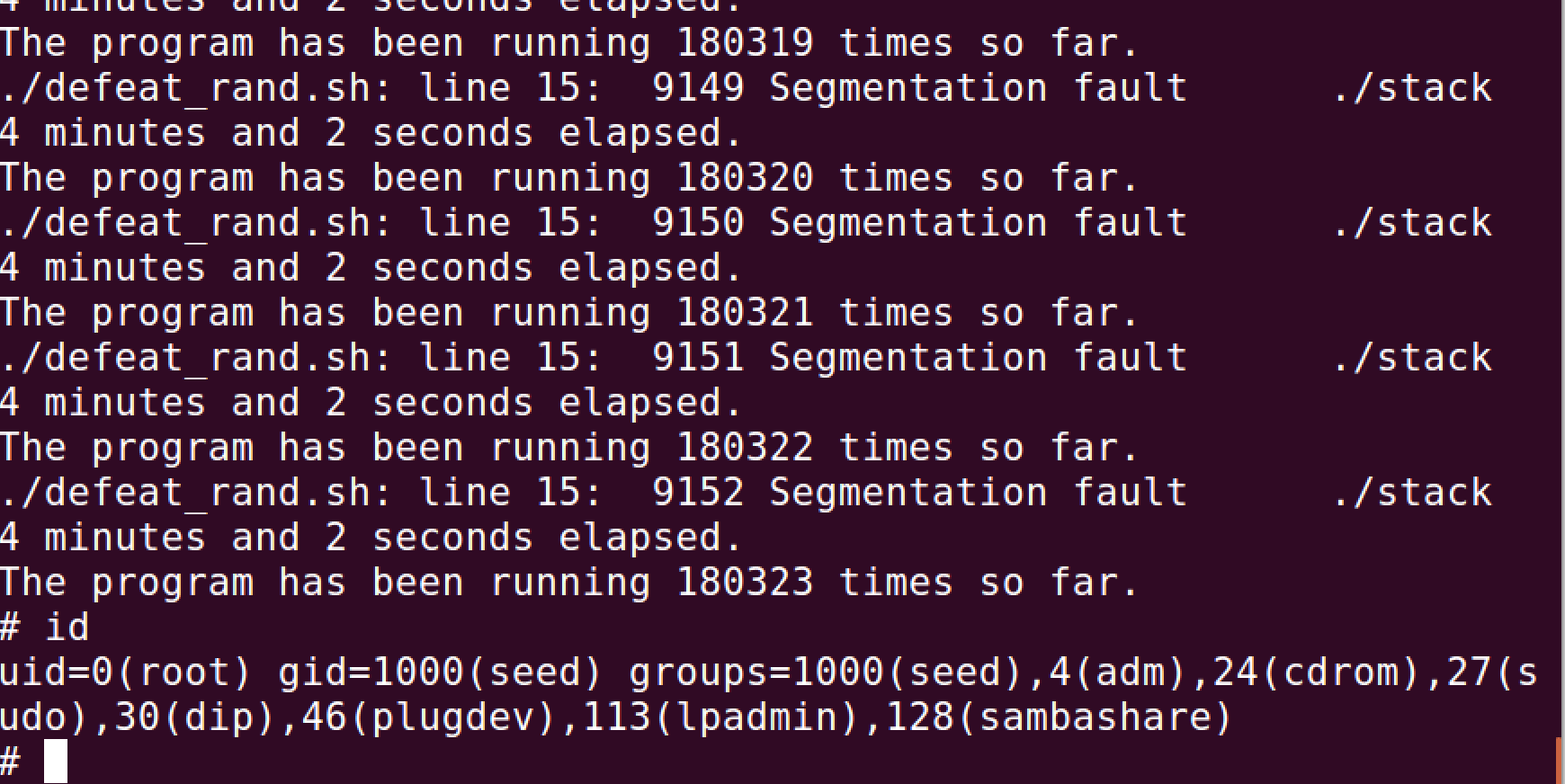




This shellscript will use brute force method to get pass the segmentation faults that ./stack will throw and it won’t stop until it is successful or I stop it manually.

Commands Used w/ output:

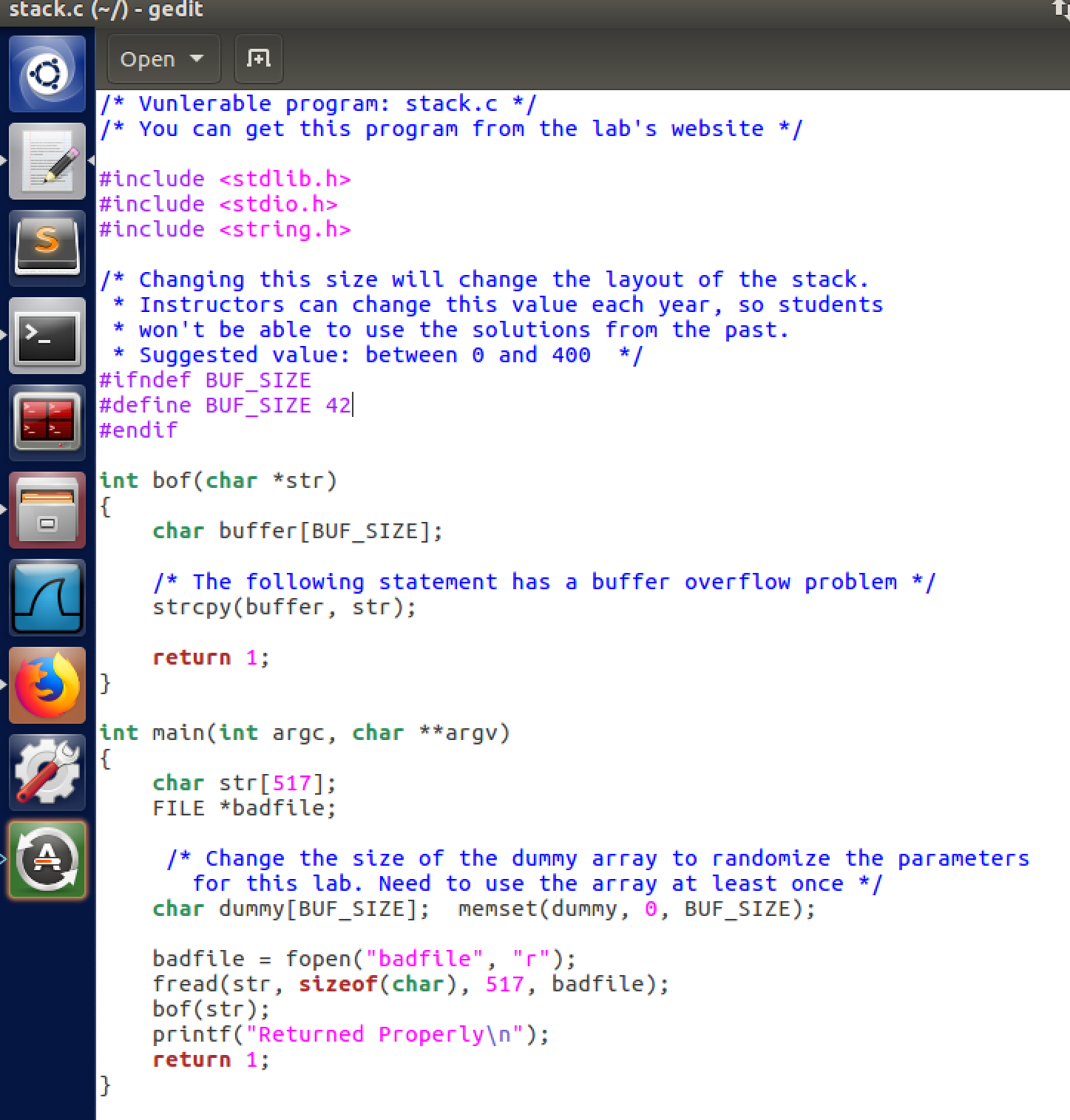




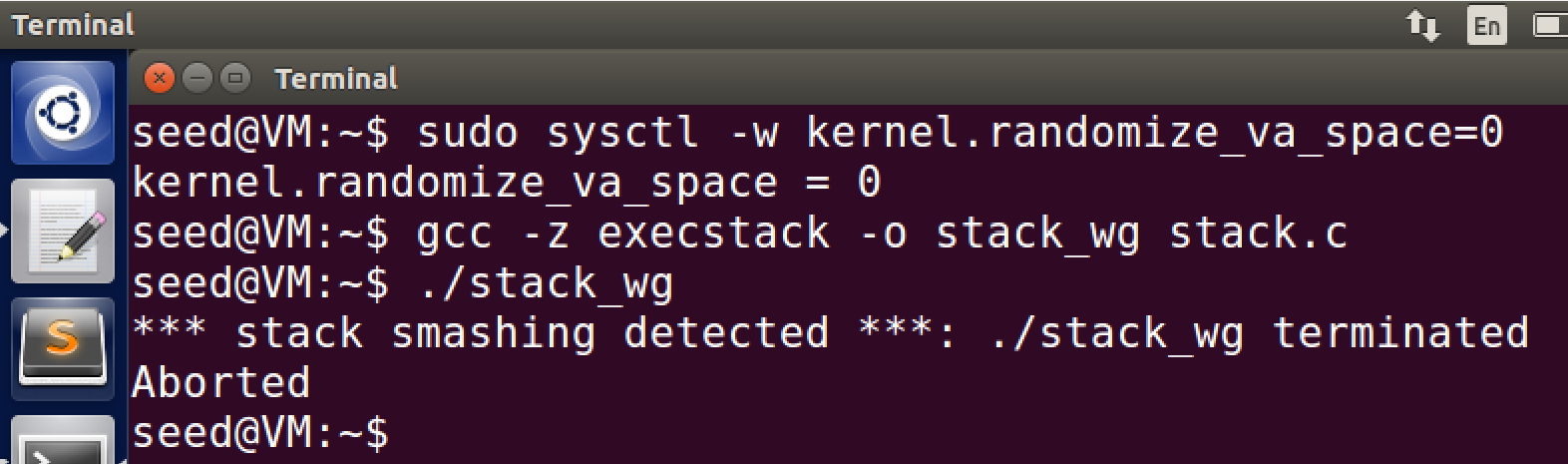
The shell program ran for a little over 4 minutes continuously running ./stack until it brute forced itself passed the address randomization protection and was able to run ./stack. It ran over 180,000 times.

Task 5: Turn on the StackGuard Protection

Code Used:



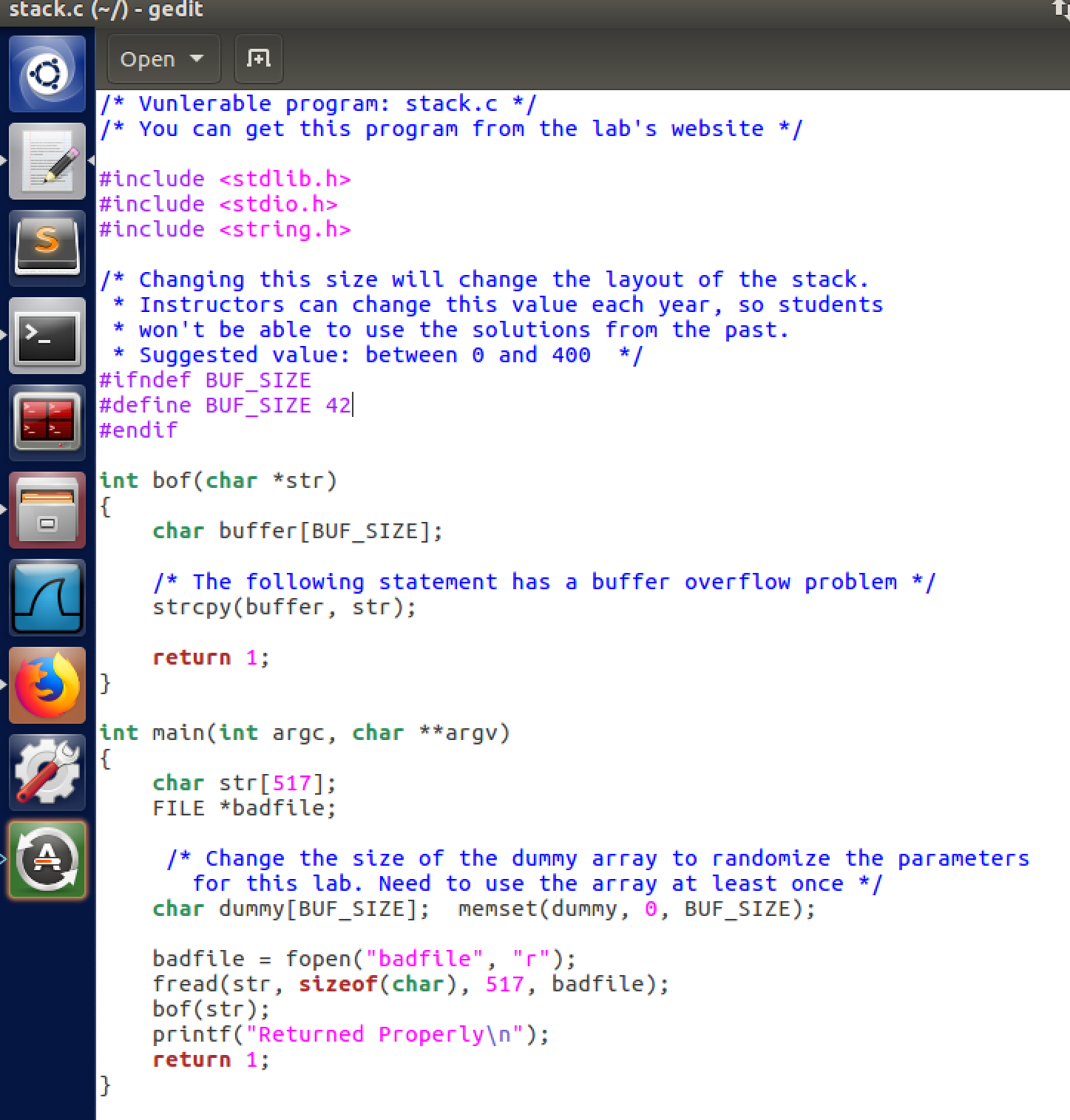
Commands Used:



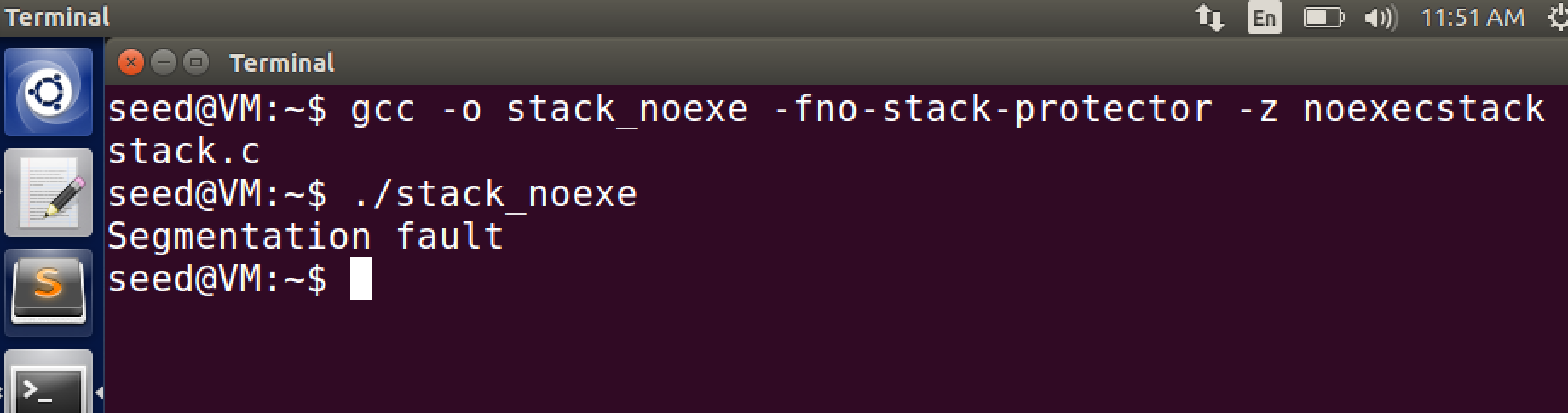
After turning off address randomization, trying to run stack.c it throws a stack smashing detected message indicating that stack guard is enabled and stopped the attack.

Task 6: Turn on the Non-executable Stack Protection

Code Used:



Commands Used:



Without the stack being executable you cannot run the attack because the malicious code is in the stack but the stack is no longer executable.