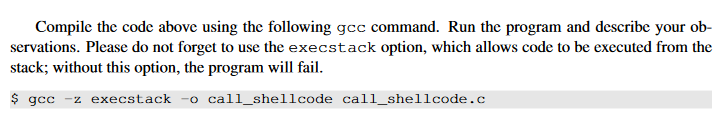
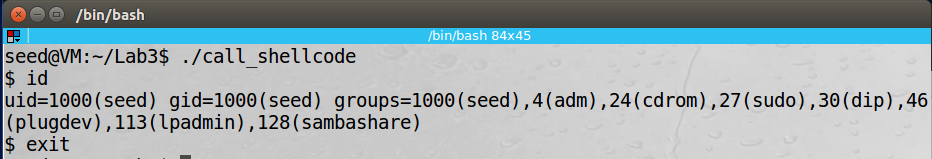
Task 1:

****

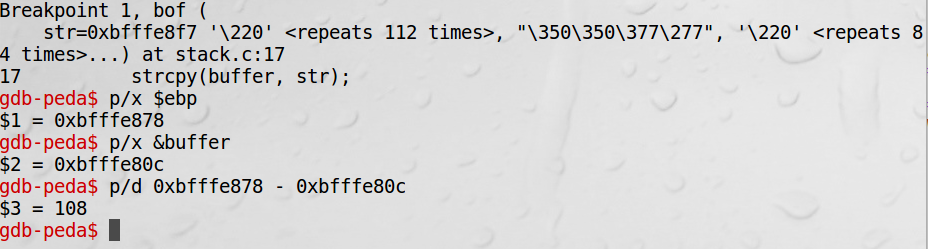
When we run it we get some warning in our terminal as well as an executable of call\_shellcode with the following privileges:



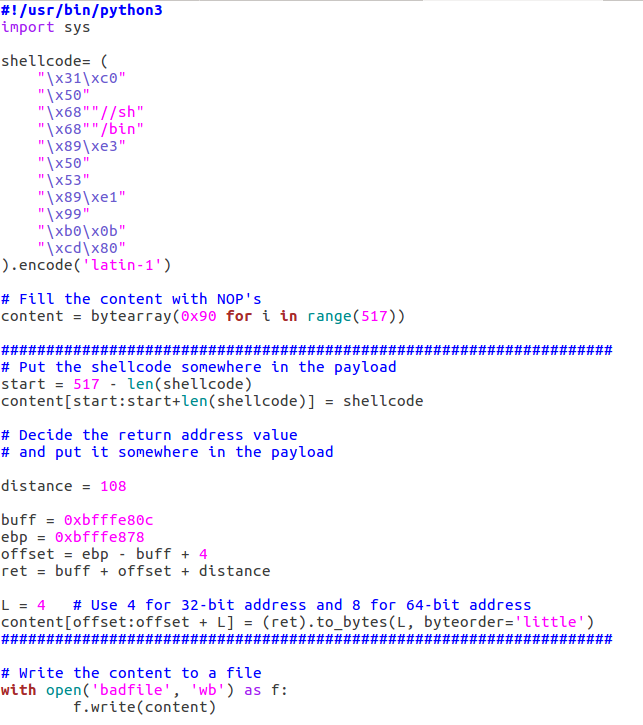
Task 2:

Implement changes to exploit.py, show them, show root shell

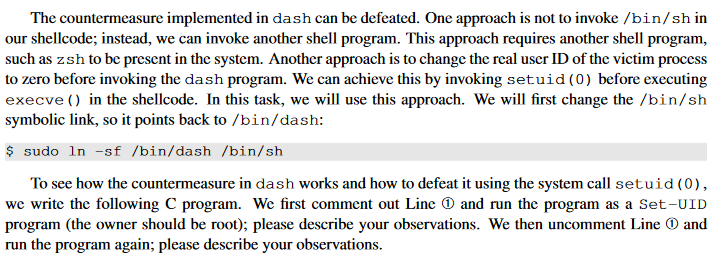
Using our debugger to get more info from the bof breakpoint we just get the addresses of $ebp and &buffer to find the difference, which is shown in the screenshot below:



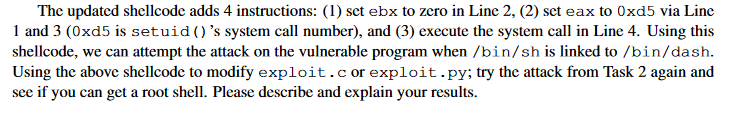
And the implemented code should look like:



Task3:

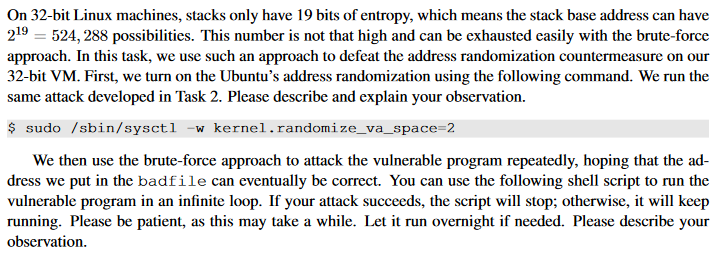


The shell will always still be a $ (underprivileged) shell because of the check in place in dash. However once the setuid command is uncommented the shell will again be a privileged shell.

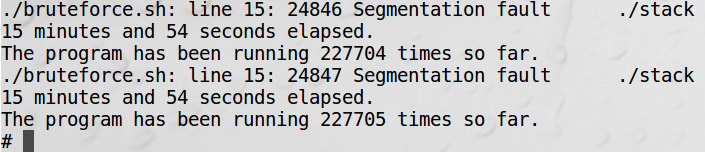


Once the setuid command was added to the shellcode, the exploit worked again.

Task 4:

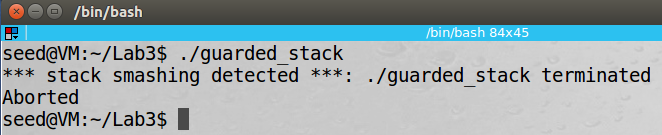


The code took about 16 minutes to brute force its way in.



Task 5:

* turn off the address randomization
* Repeat task , with -fno-stack-protector
* For this task, you will recompile the vulnerable program, stack.c, to use GCC StackGuard, execute task 1 again, and report your observations. You may report any error messages you observe



Task 6:

* turn off the address randomization first
* recompile our vulnerable program using the noexecstack option, and repeat the attack in Task 2
* Can you get a shell? If not, what is the problem? How does this protection scheme make your attacks difficult.You should describe your observation and explanation in your lab report. You can use the following instructions to turn on the non-executable stack protection
  + gcc -o stack -fno-stack-protector -z noexecstack stack.c

