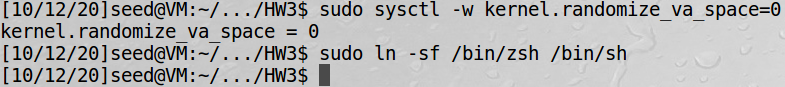
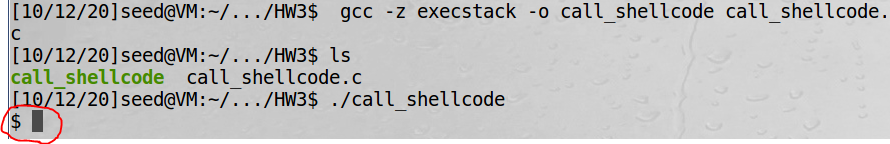
**2.1 –** Turning Off Countermeasures

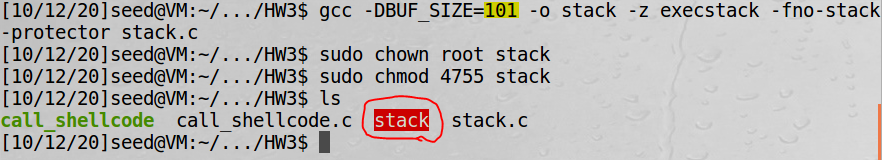


**2.2 – Task 1:** Running Shellcode



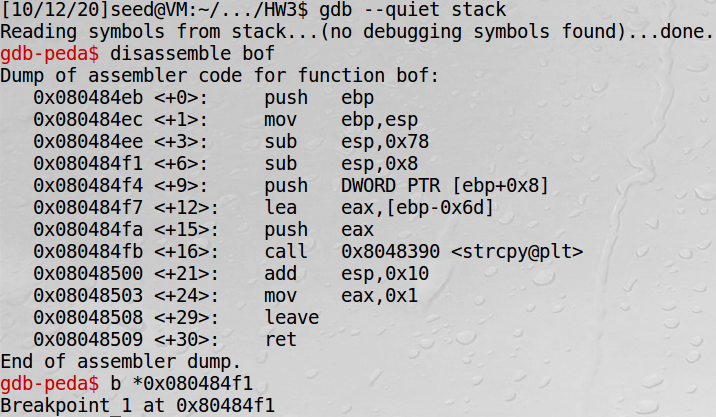
-Success! We are granted a shell.

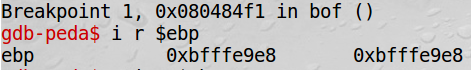
**2.3 –** The Vulnerable Program



-Success! The stack program compiled as expected and we were able to make it a root-owned Set-UID program.

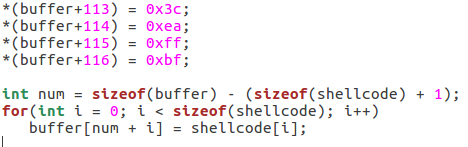
**2.4 – Task 2:** Exploiting the Vulnerability



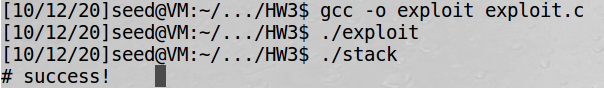


-find base pointer address and add 0x6d+4 (109+4=113) to get the return address we want to overwrite

-change exploit.c to overwrite return address with something that will land in our NOP instructions



-run exploit and then run stack, and we get a root shell



**2.5 – Task 3:** Defeating dash’s Countermeasure







-run without line ➀



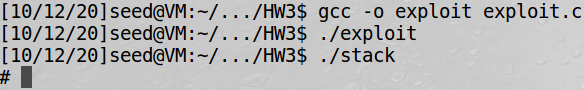
-Here we see that we are granted a shell, but the user ID belongs to the normal user.

-run with line ➀



-Here we can see (with line ➀), we are granted a real root shell (where the user ID is root)

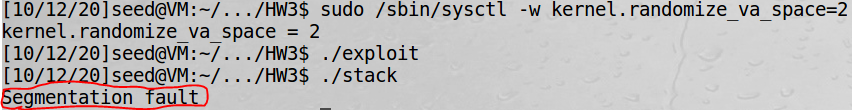
-rerun exploit with 4 new lines added



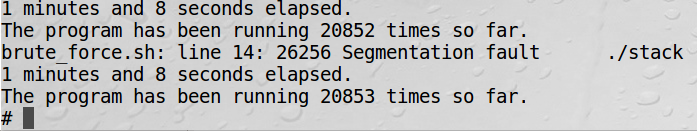
-Here we can see that these 4 lines have also allowed us to open a real root shell (where the user ID is root)

**2.6 – Task 4:** Defeating Address Randomization

-I’m assuming here that we keep the changes to the exploit program that were implemented in 2.5 (task 3)



-run brute force shell script



-eventually, the brute force technique breaks through the address randomization defense and we are granted a shell

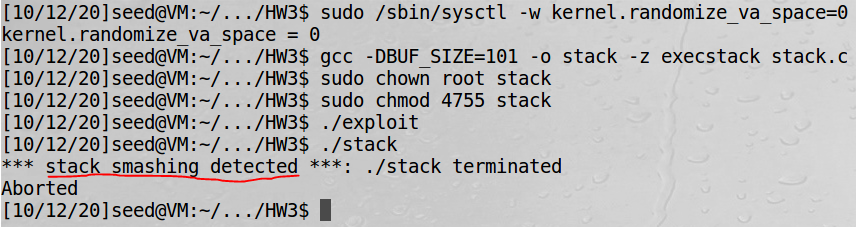
**2.7 – Task 5:** Turn on the StackGuard Protection

-turn off address randomization (first command below)

-still assuming that exploit program retains changes last made in 2.5 (task 3)

-recompile stack.c without -fno-stack-protector and make it a root-owned Set\_UID program again

-reattempt exploit attack from Task 2



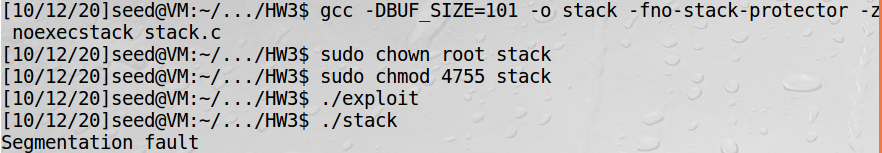
-As you can see, the attack is unsuccessful as the system says “stack smashing” was detected and it aborts the program.

**2.8 – Task 6:** Turn on the Non-executable Stack Protection

-address randomization is still turned off from Task 5 above

-recompile stack.c with StackGuard but with *no*execstack and make it a root-owned Set-UID program

-reattempt exploit attack from Task 2



-As you can see, the attack is unsuccessful. We get a segmentation fault because noexecstack makes it impossible to run shellcode on the stack. This means that all the shellcode we wrote in the exploit.c program is no longer able to be executed. So when we try to run ./stack, and stack tries to (unknowingly) run that “malicious” code, the system gives a seg fault to stop the shellcode from running which might be an attack. In this case, it saves the stack.c program from what is, in fact, an attack.