Seed Labs progress log

**Buffer Overflow Lab:**

9/24 – set up VM properly, downloaded “Buffer overflow” files. Began to watch video series on Buffer Overflow.

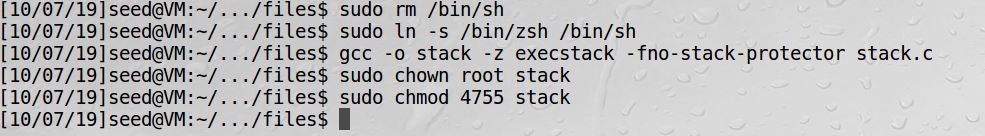
**Pre Task- Turning off counter measures.**

Before a buffer overflow attack can happen, certain counter measures have to be changes or shut off.

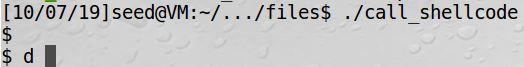
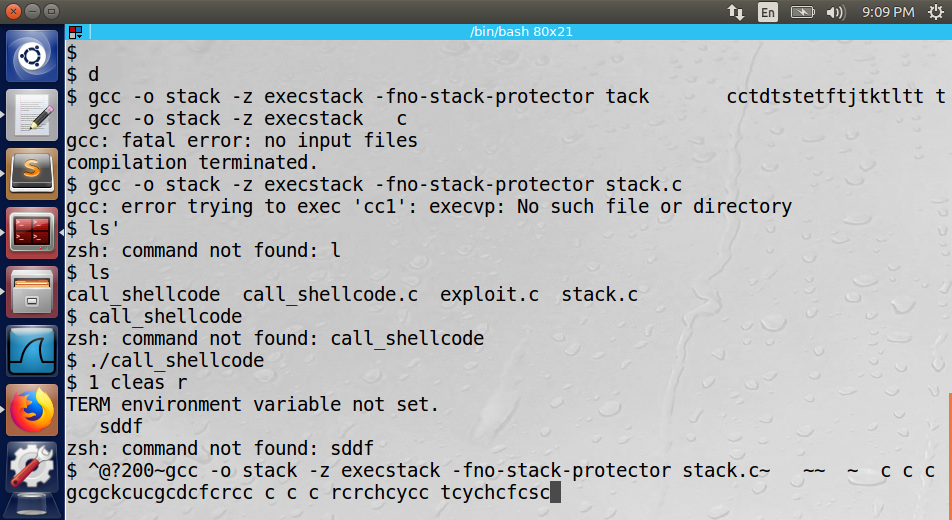
* Change space randomization to 0:



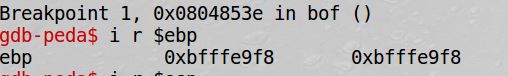
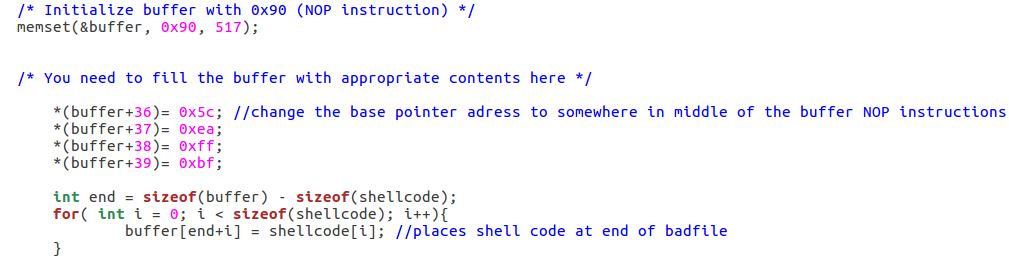
* Linking to different shell that doesn’t have counter measures. And then compiling the stack code with counter measures shut off:

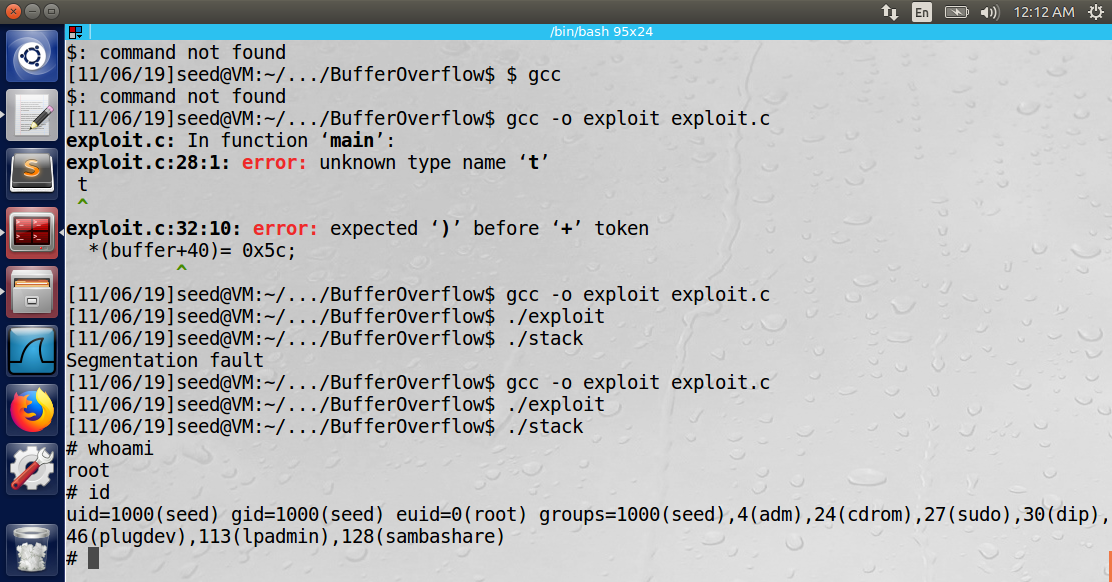


**Task 1 – Running Shell code.**

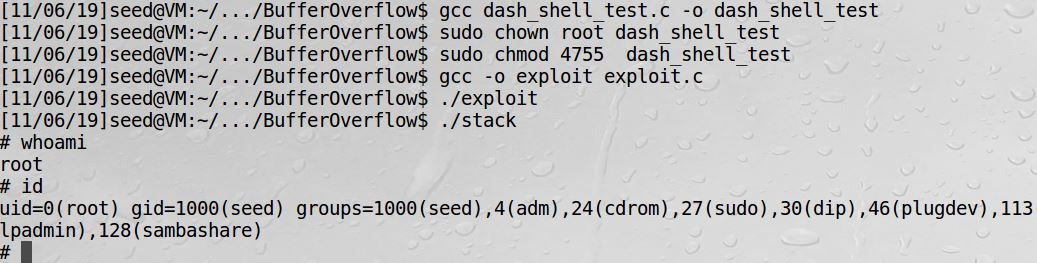
* Compile with counter measures shut off: 
* Running the shellcode program-
* After running the shell program the system was in shell mode. Without realizing what happened I kept trying to call different cmds in the terminal and kept getting a call back from the ZSH shell program that the cmds are not allowed. 

**Task 2 – Exploiting the vulnerability:**

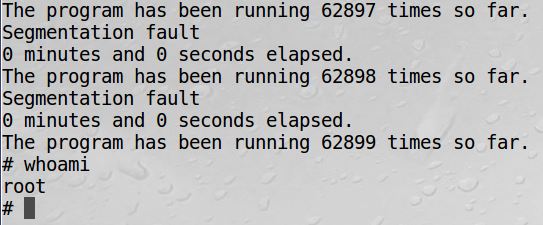
* I used gdb to find the memory address for the stack frame base pointer ($ebp) which was 0xbfffe9f8.
* In the exploit.c file I changed the 4 memory addresses after the buffer which is the return address location in the stack frame to a memory location in middle of the buffers NOP location. I added 100 to the ebp to get 0xbfffea5c memory location. All the NOP operations are placed in a file called badfile that gets reads into the buffer location.
* I then ran a loop to place the shell code at the end of the buffer.
* After compiling and running the exploit.c program and then running the stack program, I was able to get ROOT shell privilege.



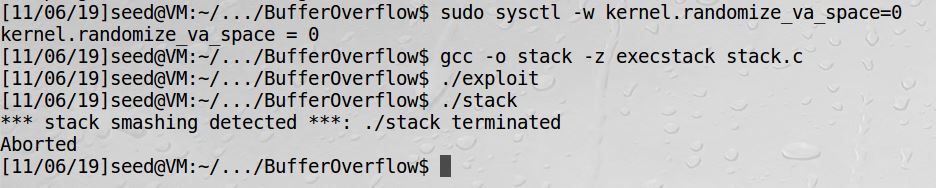
**Task 3 - Defeating dash’s Countermeasure:**

* Added command to the shell code that bypasses dash counter measures to prevent buffer overflow if the effected UID des not equal the real UID. After running the program I still was able to get ROOT shell. 

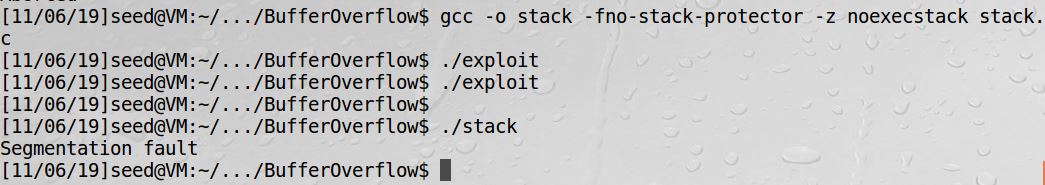
**Task 4 - Defeating Address Randomization**

* With address randomization turned back on, I ran the attack again using brute force trying to eventually get the right location for the shell and I was able to gain ROOT privilege. The
* Program looped for about 15 minutes and ran 62899 times until I was able to get Root Privilege.

**Task 5 -Turn on the StackGuard Protection:**

* In this task I turned address randomization back off, and then recompiled the stack.c program without the StackGuard protection. When I re-executed the attack, I received an error of “stack smashing detected”.

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

* In this lab I recompiled the stack.c program with the “noexecstack option that prevents running shell code on the stack. When I re attempted the attack, I received a Segmentation fault error. 

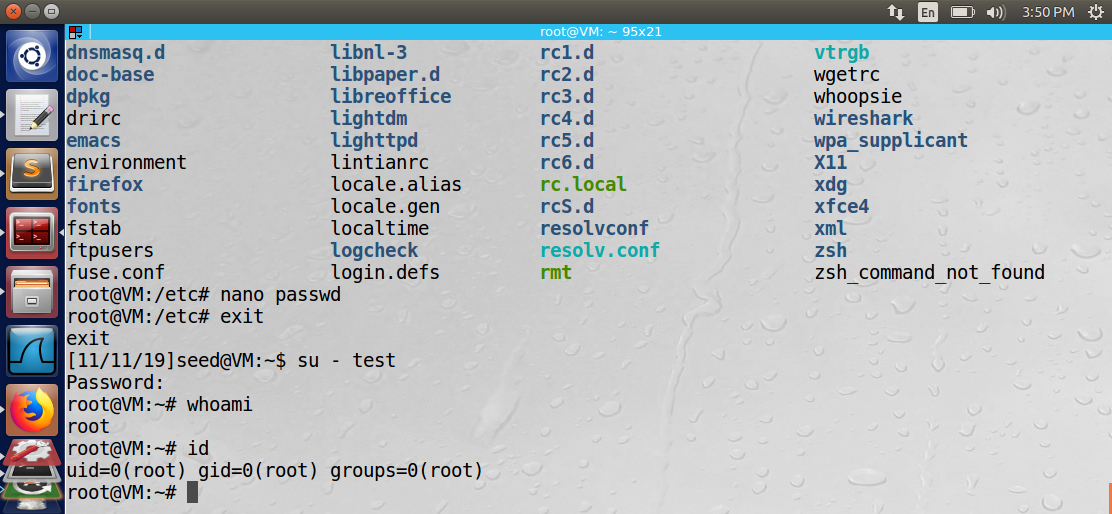
**Conclusion:**

* I was able to launch a root shell via a buffer overflow vulnerability.
  + I used the debugger to find the memory location of the return address.
  + I used the information to add contents to a bad file to be used in the stack program that has the buffer overflow vulnerability.
* The countermeasures to prevent a buffer overflow where that where effective in preventing the exploit from being carried out are,
  + StackGuard
  + Non-Executable stack.
* The countermeasures that failed to prevent a buffer overflow exploit from being carried out were,
  + Dash counter measures
  + Address Randomization.

**Race Condition Lab:**

**Task 1 Choosing Our Target:**

* This Lab is to verify that the target file which is the system password file, with the Ubuntu magic password works to create a root user that does not require a password;
* I added the following line to the password file that has the “magic value” that does not require a password to sign in to test user and the user has root privilege.
  + test:U6aMy0wojraho:0:0:test:/root:/bin/bash

****

**Task 2**

**Task 3**

**Task 4**