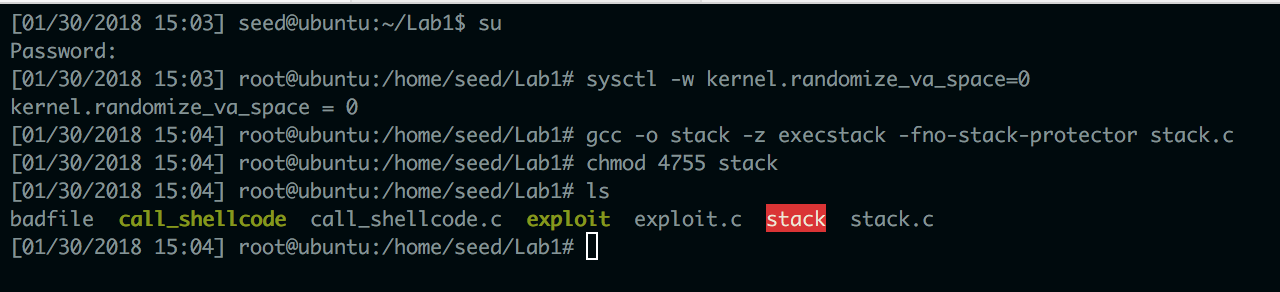
Homework #1

Buffer Overflow Lab, Task 1

For task 1 of the buffer overflow lab, we were given two files, *stack.c* and *exploit.c*, and we were tasked with modifying the *exploit.c* file to produce a text file, appropriately named *badfile*, that would compromise the vulnerable *stack.c* file when run and open up a root shell. Before performing the lab, it was necessary to compile the *stack.c* file without stackguard and with an executable stack. The *stack.c* file also had to be given root permissions. I performed this lab alongside my classmate, Chase Cloutier. The original *exploit.c* file merely wrote to *badfile* a string of length 517 that was composed completely of no-op characters, although the *exploit.c* file did contain the necessary shell code to open up a root shell from a c program given the necessary system permissions. Clearly, the *exploit.c* file as given would not be enough to smash the stack. Chase and I looked to the paper *Smashing the Stack for Fun and Profit* for inspiration when considering how to modify the *exploit.c* file. We agreed that since the buffer we were targeting was so small, we should place the shell code at the beginning of our *badfile* string. Once I inserted the shell code into the *badfile* string, all that was left to do was determine an appropriate return address that would get *stack.c* to execute the shell code, and then put that return address in a position where it would overwrite the original return address that *stack.c* intended to use. The paper *Smashing the Stack for Fun and Profit* suggested using an existing stack pointer to make a good guess at what the return address should be, and provided a method, *get\_sp* that I used in *exploit.c* to effectively guess where the buffer in *stack.c* would be in memory. Finally, it was an exercise in guesswork to determine exactly where to put this return address so that *stack.c* would return to the shell code that I put in its buffer. For my lab, it turned out that putting the return address 3 characters after the end of the buffer was enough to redirect *stack.c* and open up the root shell. The following screenshots document how I compiled stack.c in order for it to be successfully exploited, as well as the exploit in action.

Compile steps (a root user must compile and give root permissions to *stack.c*):  


Attack execution (badFile is displayed as well):  
