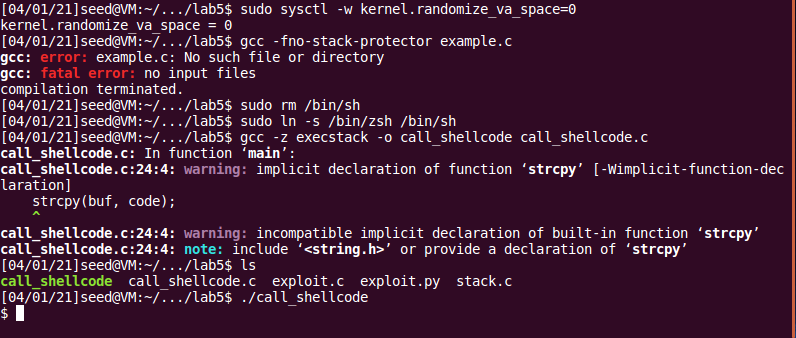
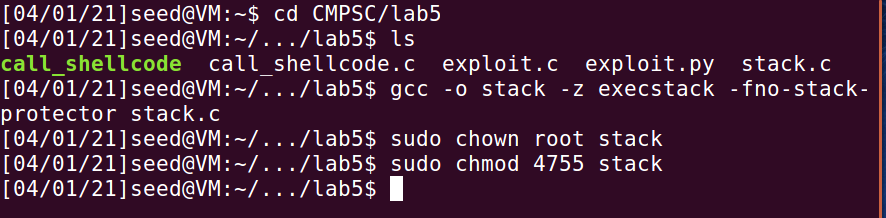
Shane Hagan

CMPSC 443

Lab 5

Task 1



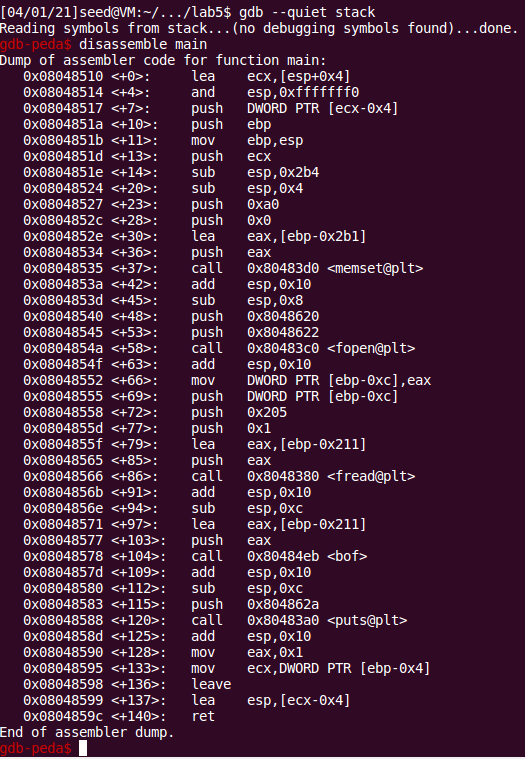


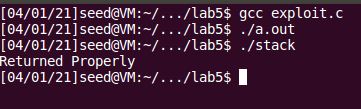
We begin the lab by exploiting the program. Here, we are disabling the feature of the program, while

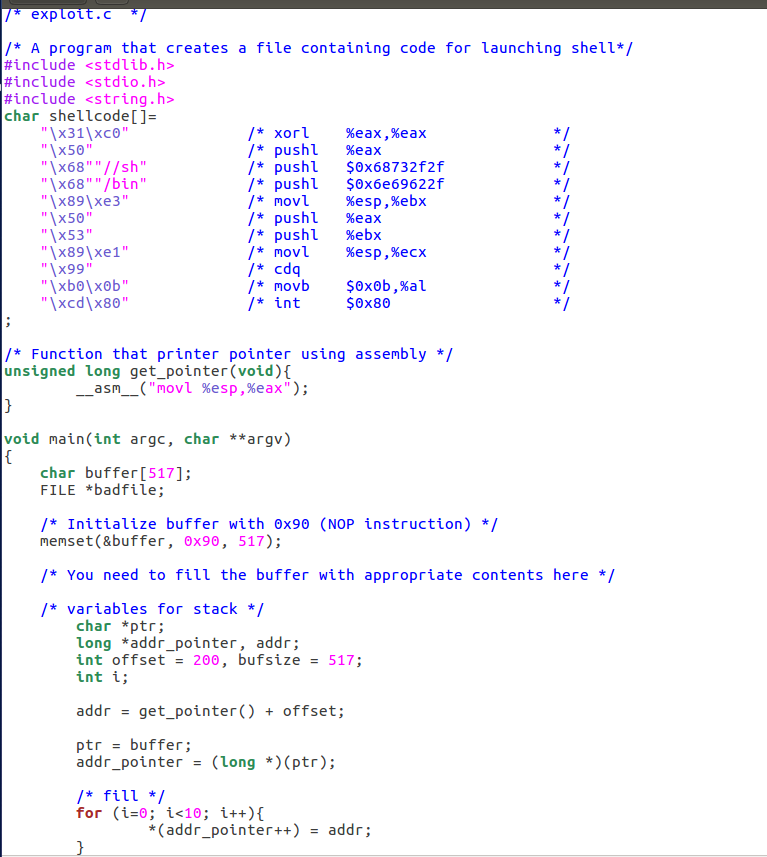
also giving ourselves permission. We also configure the program, or manipulate it, to do exactly what we

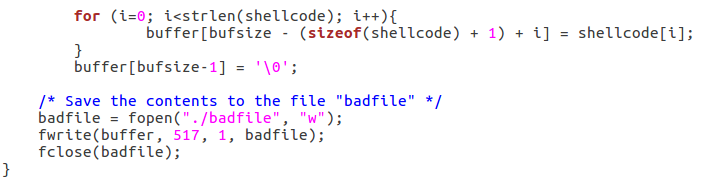
want. We then compile the program.

Task 2









Here is the entire program I decided to write in C. It has appropriate comments to describe exactly what

each step is doing. Overall, we compiled and then I used the gcc debugger to find out exactly where the

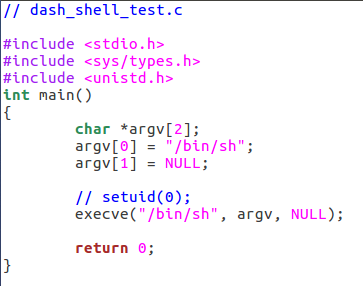
address of the buffer begins, goes through, and eventually ends. After finding the exact spots for the

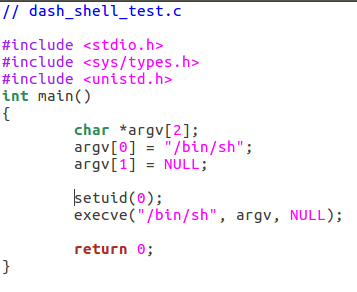
buffer, it was trivial to write the program to use the buffer to its advantage and run the program of

stack. As can be seen from the above screenshots, after we use this code and run the stack executable,

we get the line we are looking for (returned properly) This means that this step worked.

Task 3



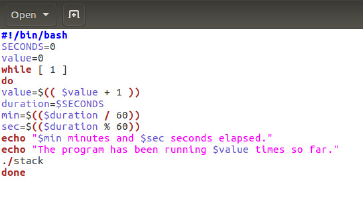


Seen from these two screenshots, we can comment out the line, and leave it in, for both

situations. After running the program, my observation is that it was successful and printed out the same

id’s both times (below screenshot). It makes a difference, but the id’s stayed the same it appears from this exercise.

Task 4



Here we create the program, run and then we see it will run for some amount of time, notifying us of

the number of times it has ran. Shortly after it segfaulted many times, it was able to inform us where the

address was able to do a complete return. We also observe that the id’s remained the same as well.

Task 5

Here, we observe the stack program will “abort” the program when we try to run it. This more than

likely means that if the stack safeguard is on, we cannot overflow the buffer, as it will notice the

buffer attempting to be edited and stops it before it can even occur.

Task 6

Here, we notice that the program will seg fault instead of aborting or running. We can observe

that this step doesn’t allow the program to just insert the code into the buffer, it differs from our

different strategies to overflow the buffer.