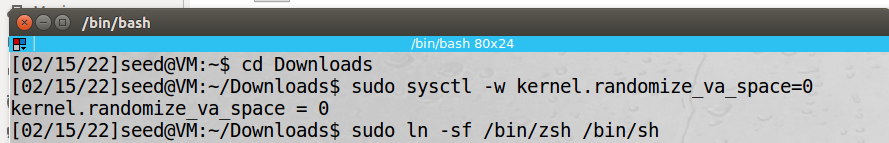
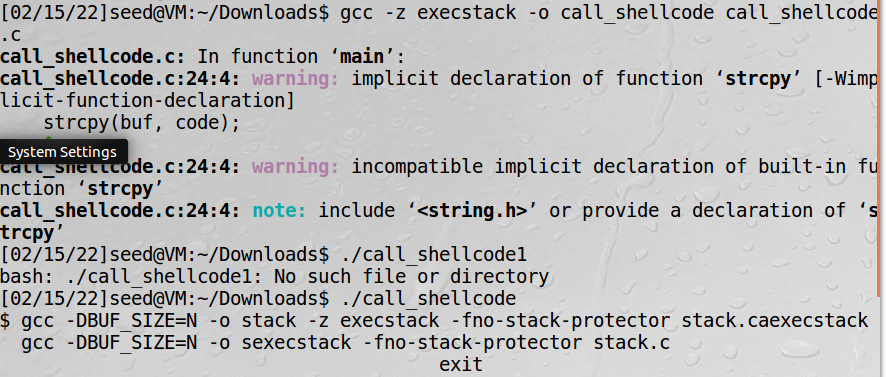
Homework 2 Report

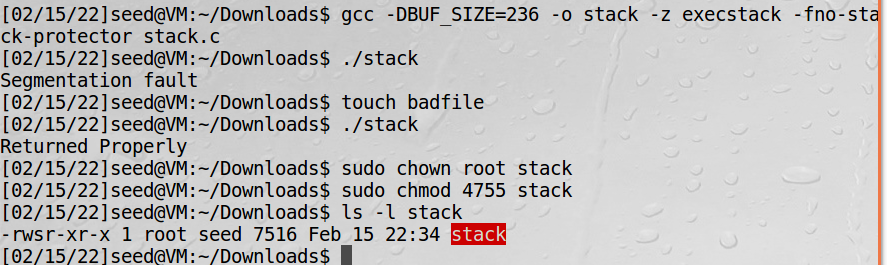
2.1 Turning off Countermeasures



2.2 Task1: Running Shellcode

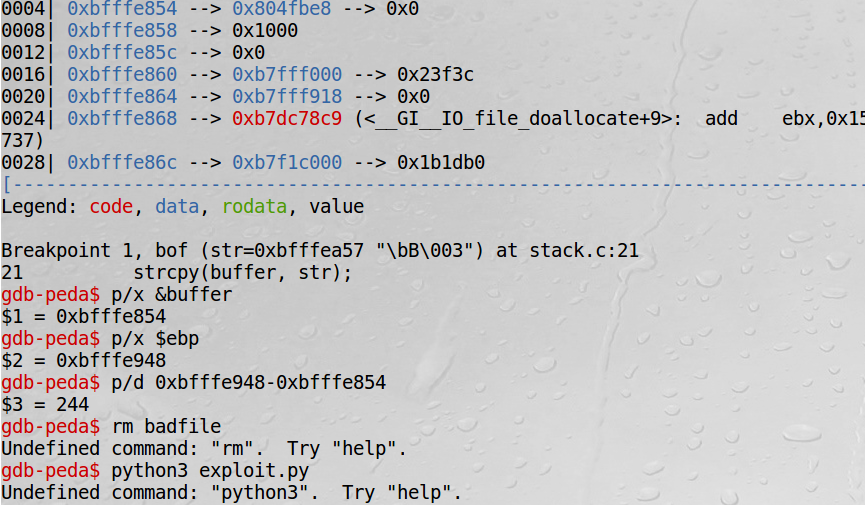
I observed that it opened up the regular shell that we the user can interact with like a normal shell.

2.3 The Vulnerable Program



Above I did some testing to understand the stack file.

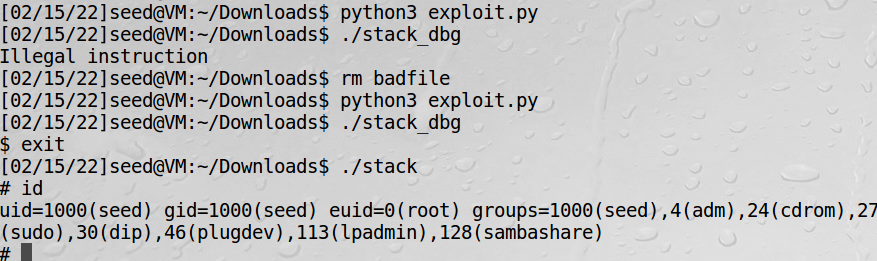
2.4 Task 2: Exploiting the Vulnerability



The above was done to find what code was needed to be added. I used the debug mode on the stack program to determine the buffer address and the base address pointer so that I could calculate how far after the buffer we need to go in order to cause the buffer overflow exploit. I did this in exploit.py. The picture below is the section I changed in exploit.py to get it to work as intended.

Text

Description automatically generated

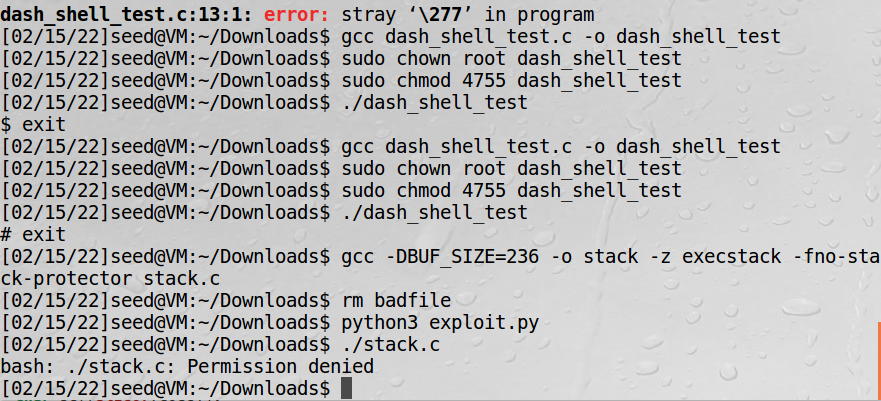
I confirmed it was working due to the picture below as I accessed the root shell:

2.5 Task 3: Defeating dash’s Countermeasure

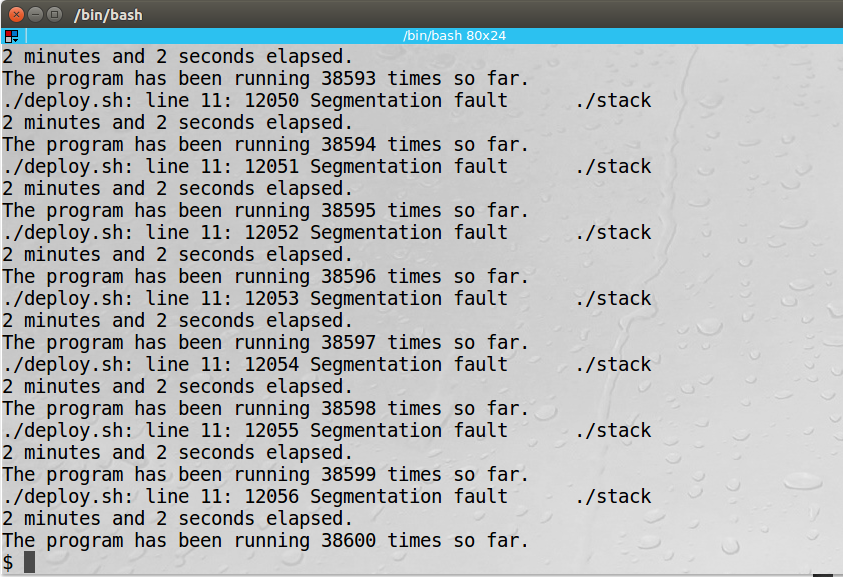
A screenshot of a computer

Description automatically generated with medium confidence

With the line 1 commented we are presented with a normal shell but with line 1 uncommented we are presented with the root shell. This is because by setting the UID to zero we become the root user and have access to the root shell.

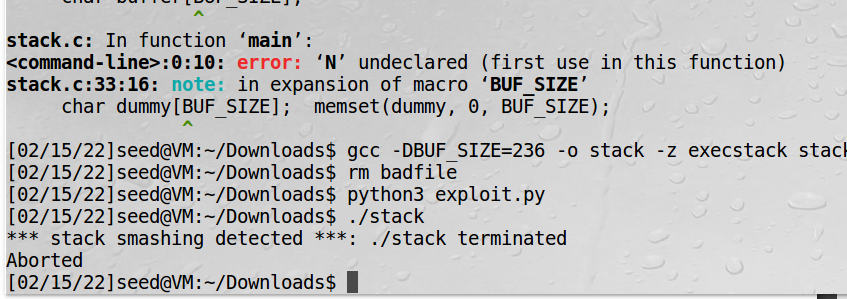


We could not access the shell root. This is because the symbolic link of /bin/sh returned to /bin/dash which prevents Ubuntu from being executed in a Set-UID process, which would be the stack executable.

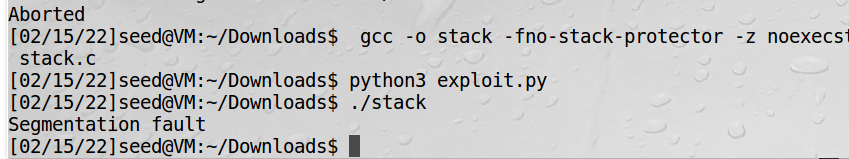
2.6 Task 4: Defeating Address Randomization

It took 38600 attempts to crack where the stack address is in order to allow us to attempt a buffer overflow exploit when address space randomization was on. Brute forcing can work but it will take a while to achieve results. in a x64 system it would take even longer to achieve as the would be more possibilities of address spaces.

2.7 Task 5: Turn on the StackGuard Protection



With StackGuard enabled we could not use the stack to perform a buffer overflow attack. The system detects it and prevents it from happening. The StackGuard is an effective counter measure to prevent stack manipulation.

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

Even with StackGaurd turned off we still get an error. This time it is a Segmentation error. This is probably due to the Address Space Randomization being on and causing us to access areas outside our intended target.

Summary:

I was intrigued by this lab a lot and enjoyed the hands-on experience with buffer overflow attacks. I enjoyed learning about the safety mechanisms that help prevent a buffer overflow attack and what could possibly happen if they were not there.