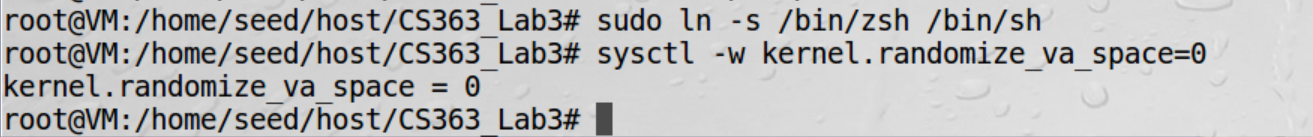
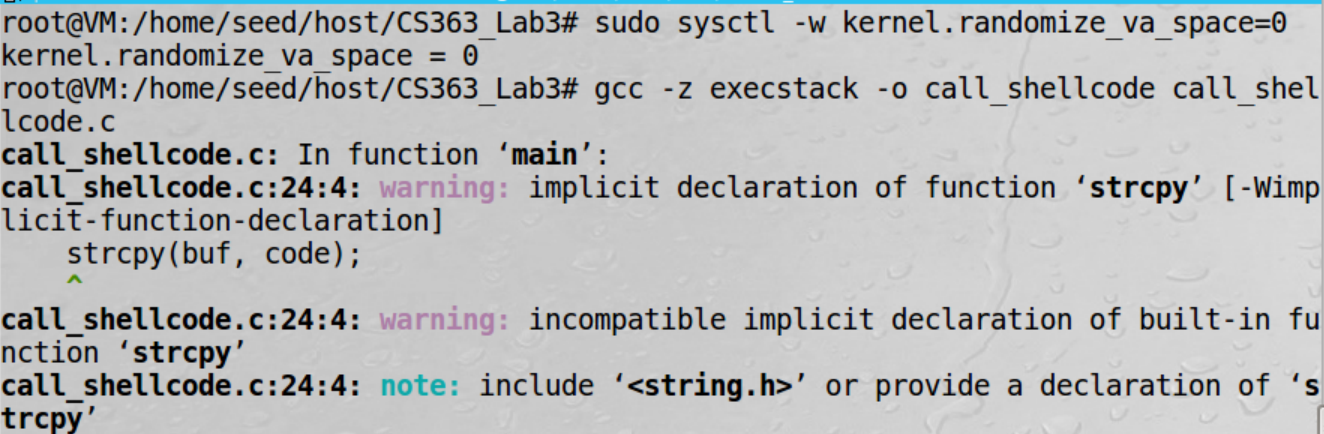
Tyler Bartlett  
1/24/2019  
CS363 Spring 2019  
Prof. Yanwei Wu  
Lab 3 – Buffer Overflow

Disabling Address Space Randomization:

The first thing I did was setup my environment per lab instructions to disable “Address Space Randomization” which randomizes the starting addresses of the heap and stack. We need to be able to guess these addresses for this lab, so we disable this feature to aid in this.  


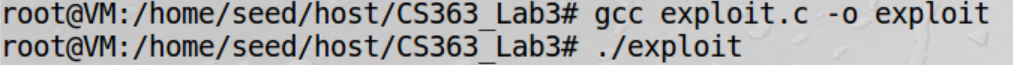
Shell Code:

Before we can do the attack, we must first have a shellcode. A shellcode is the code to launch a shell. It must be loaded into the memory so that we can force the vulnerable program to jump to it. The shellcode was given to us in our lab resources.



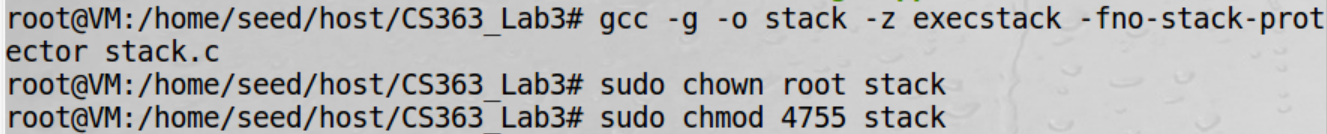
Exploit:

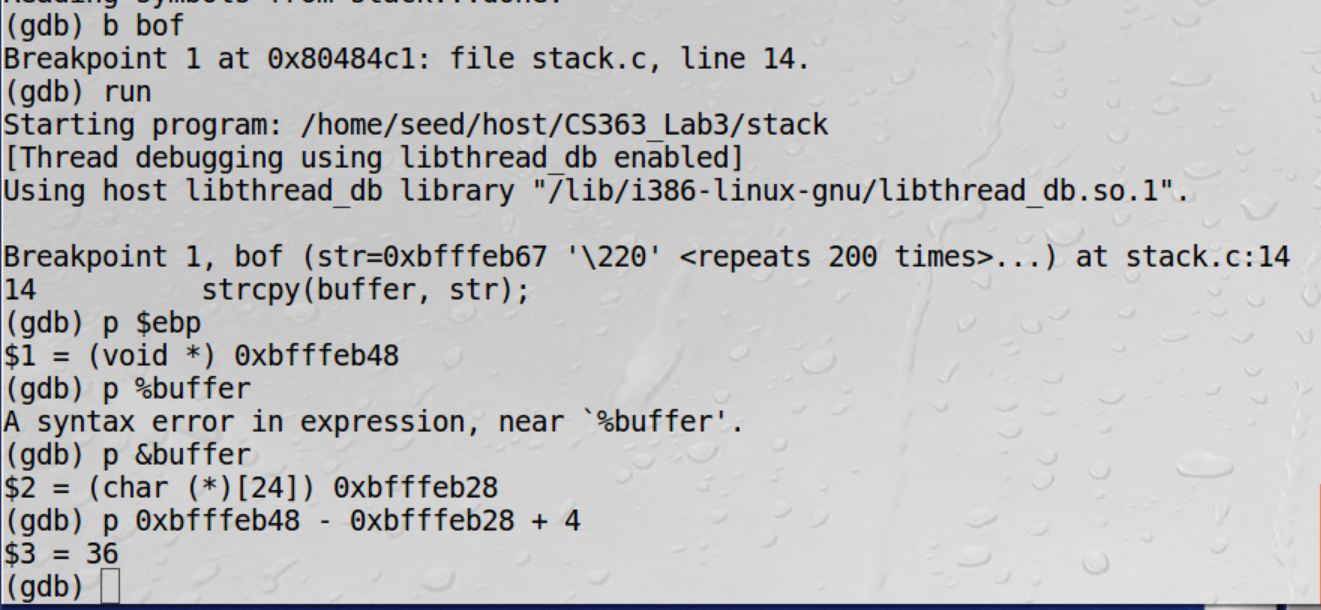
The next thing I did was create the badfile, which was done by running exploit.c

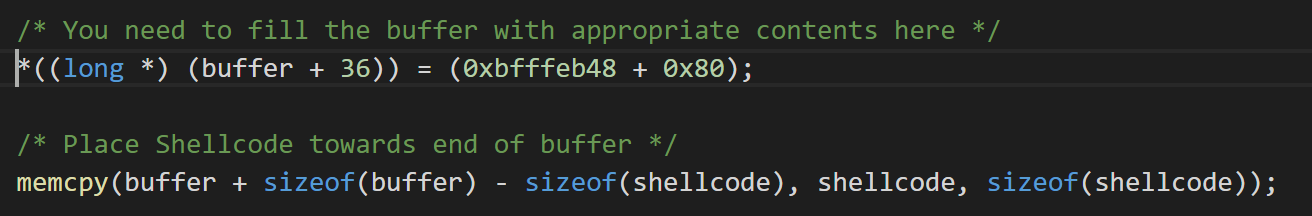


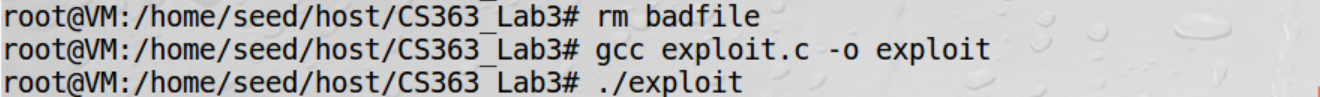
I then compiled a set-uid root version stack.c to use in our lab, as seen below.

Note that the stack.c file when compiled, must be marked to run as an executable stack which we achieve with the options *-z execstack.* We must also compile with option *-fno-stack-protector* to disable the “StackGuard Protection Scheme” which protects from Buffer Overflows. The *-g­* is included to reduce the distance of addresses between buffer and frame pointer.

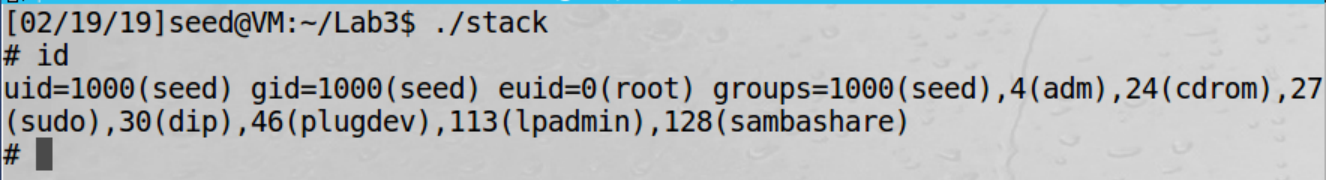


The next step was to launch our stack program in gdb to find the address’ of the buffer and face pointer and calculate the distance between them. 

I then go back to the exploit.c file and modify it to have the distance from the base of the buffer to the return address and with the address to the malicious code, which is the address of ebp plus some value. 

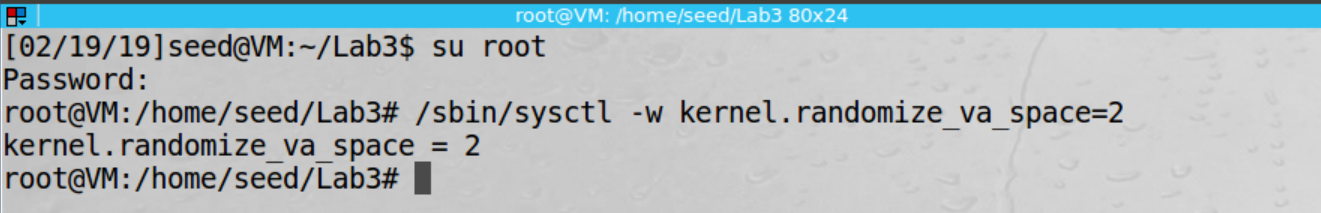
We then remove the old badfile and recompile the exploit.c file after our modification.

Now we run stack and see a root shell was achieved!

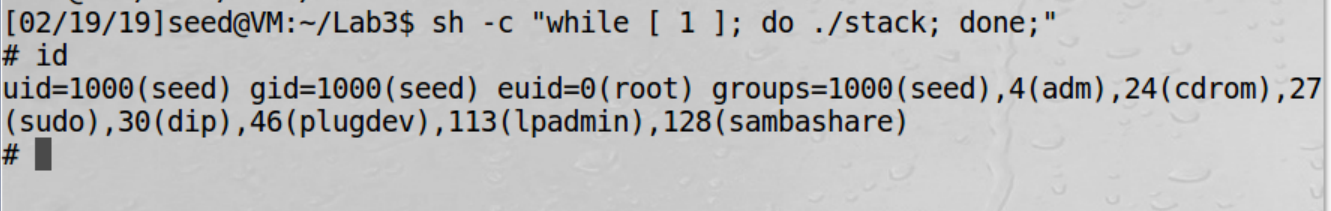


2.5 Task 2: Address Randomization:

Now we turn addess randomization back on and wil attempt to achieve a root shell but uppon launch, it gets a segmentation fault.

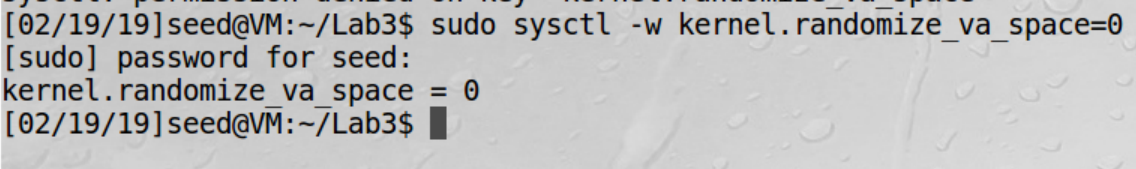


I ran the ./stack in a loop to run it many times until I achieved a root shell. It took approximately two minutes to complete.

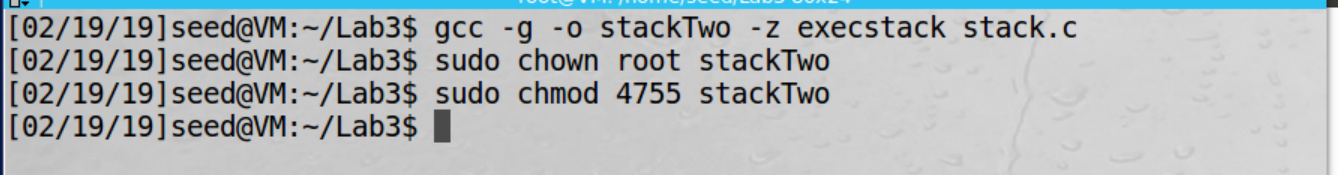


2.6 Task 3 Stack Guard:

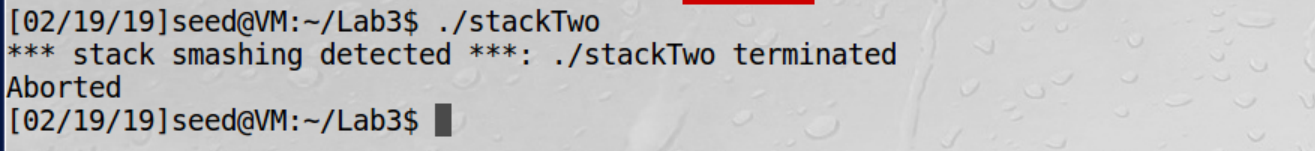
First step was to disable again the address randomization.



I then recompile the stack.c file with the fno-stack-protector off.

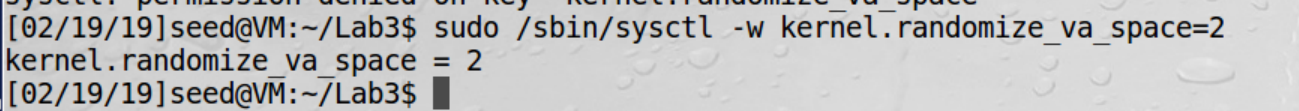


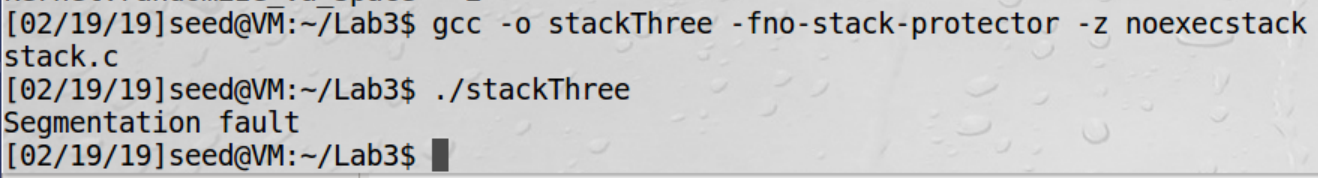
Now running the stack we get the following error:

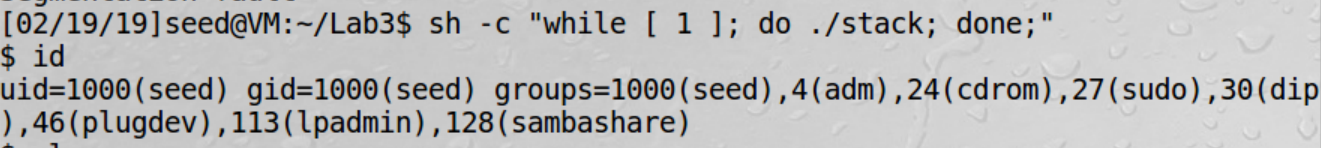


2.7 Task 4 Non-executable Stack:

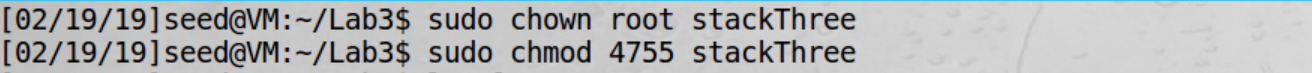
Again the first thing to do is disable address randomization.



I then recompile the stack.c file with the noexecstack option and tried running again, but got another segmantation fault. 

So I then attempted to run it in a loop again. 

I ended up getting a shell, but not root. I realized I forgot to change the owner and mode so I did this and tried again, as well as running the incorrect stack.



I let it run for approximately 5 minutes and after no luck getting a shell, I called it quits.

