Lab4 :Buffer Overflow Vulnerability Lab

Yukui Ye

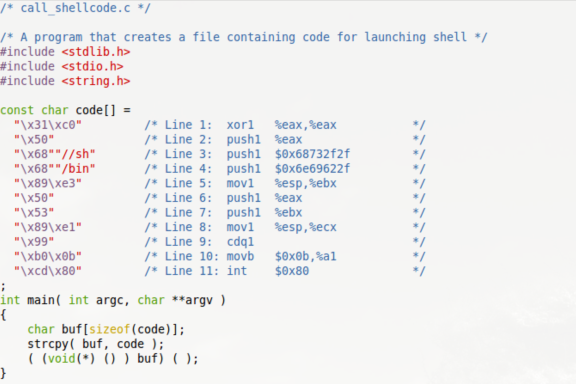
SUID : 439644268

Initial Setup: Turn off randomization

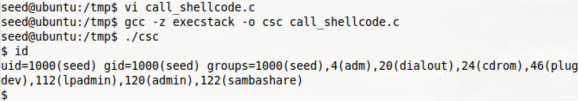


Task1: Exploiting The Vulnerability

Step1: malicious shellcode.



Compile it and run it. It runs successfully.

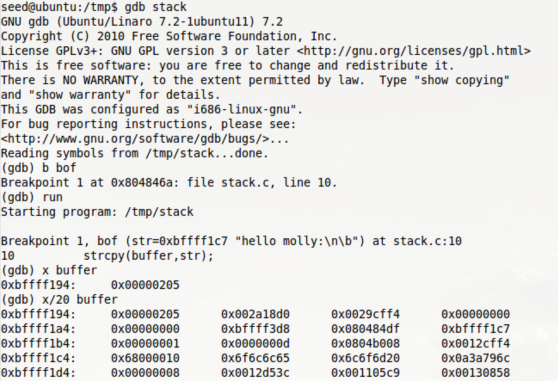


Step2:





Creat “badfile” which only include simple string, then login as normal user and debug stack program in order to find out the return address.

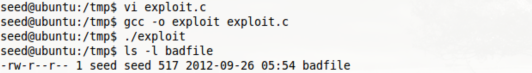


From the screenshot, we can tell that the address of buffer is “0xbffff194”, at breakpoint1, it shows the string address is “0xbffff1c7”, when we do commend to display array address in memory, it shows up the string address is the last number of the second line, therefore, the third address of the second line which is “0x080484df” is the return address , and the second address of the second line which is “0xbffff3d8” stands for the frame pointer. As for the test, we only need the address of the buffer.

Creat Badfile

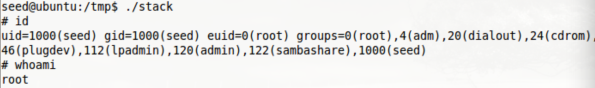
As we got the buffer address, we can creat the badfile now.



I creat a pointer to point to the position of the return address, and use the buffer address we got from gdb debug +400 , we will put our malicious code at this address, and assign this address to the return place where the pointer point to. 

We can tell that our exploit program can run successfully, then we login as root and compile the stack without debug tag.



Now, login as normal user and run the Set-UID program.

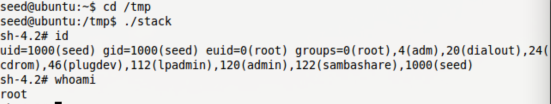
We can tell that, we are successfully change the effective Id to 0, therefore we got the permission of the root.

Task2: Protection in /bin/bash

Login as root to change /bin/sh ;

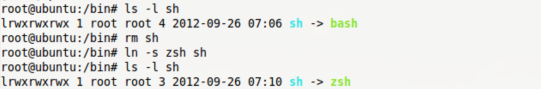


we are in bash shell, and run stack program;

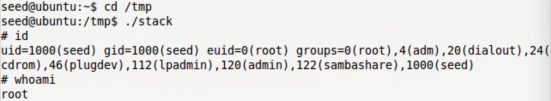


we can tell that we got the privilege of root.

Next we login as root change the shell to zsh;



run the stack program under zsh shell;



we also get the privilege of the root.

Reason ： we can tell from the screenshot , there is no big different between bash shell and zsh shell except the prompt for command, we all get the privilege of the root. I think that is because bash shell can still hold the permission of the effective user when running the set-uid program in Ubuntu 11.4 , if we use previous Ubuntu , I think we need to add “seteuid(0)” in the badfile in order to manually set effective id . then get the permission of the root.

Tast3 : Address Randomization

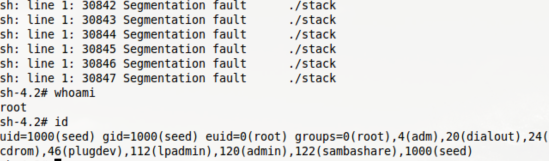
Turn on the memory randomization.



run the stack program in a shell loop script in order to get more change to hit the malicious code.



It run a while , then stopped. Which means it hit the malicious code.



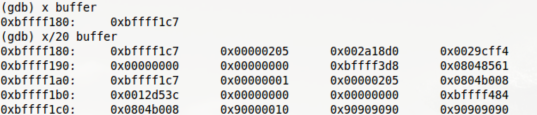
Although we open the randomization, In a short while, we still got the permission of the root.

Task4: Stack Guard

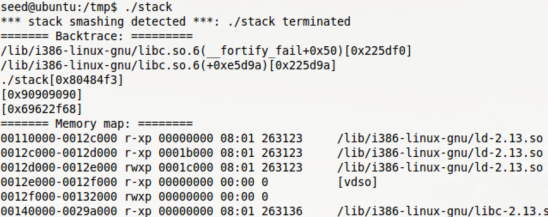
Turn off the memory randomization, and recompile the stack program by removing the “-fno-stack-protector in gcc command.



debugging stack program.



we can see that there is one element between buffer and return address, it changes each time we debug it. But we could still find where is return address located, and we need to modify badfile to add a new item. I assign value on that address arbitrarily since the value of the item is random.



we get “stack smashing detected” information instead of getting the privilege of the root. When we run the program, the new item we mentioned above would be assigned a value, and it will check whether it has been modified before the stack shrink. If the program find that the value is different, it will consider the program has buffer overflow problem, therefore it refuse to continue running.

Task5: Non-executable Stack

We remove “-z execstack” in gcc command to make stack nonexecutable. We will run “call\_shellcode.c” to see whether we will run code in stack.



from previously task, we can successfully run csc program, but now we got segmentation fault, which means the program did not run at all as well as the code in stack. Thus, the normal user cannot run the code in stack when stack program turn off “non-executable stack”, this is a good way to protect Set-UID program from malicious code attacking with buffer-overflow vulnerability.