Lab5 Return-To-Libc Attack Lab

Yukui Ye

SUID: 439644268

*Initialization: turn off the randomization*



*Task1: Exploiting the Vulnerability*

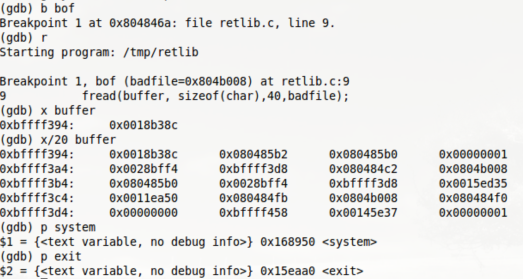
Creat the vulnerability program “retlib.c”



Compile the vulnerable program for debugging and turn off the stack protector.



login as normal user, debugging this program and find the address of library funtions by using gdb:

we can tell that the return address should be 0x080484c2 which is 24 bytes offset comparing to buffer address. Besides, the address of system is 0x168950, while the exit’s address is 0x15eaa0.

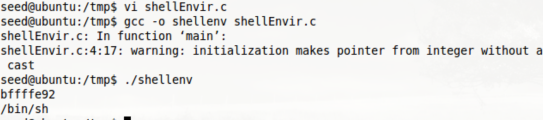
Set parameter for system() by using environment variable.



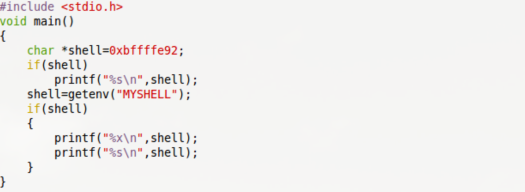
find address by using getenv() function:



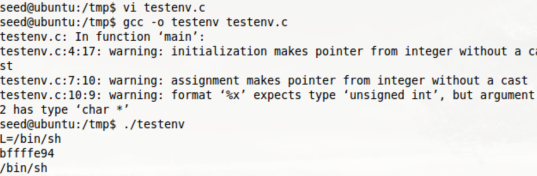
compile and run this program:



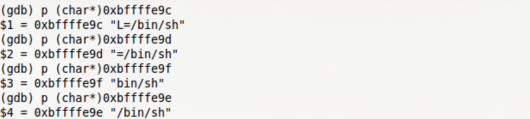
from the screenshot, we got the address which is 0xbfffe9c. we try to get “/bin/sh” by referencing this address:



compile and run this program



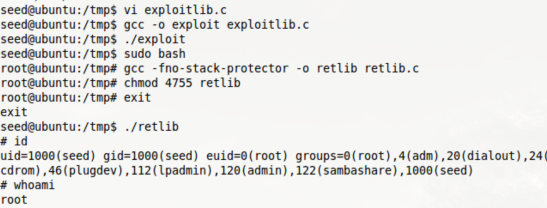
From the test result, we found that the address we got from previous step is not the real address but part of the real address(SHELL > L),thus the previous guess is not right, but I think the reall address is very close to this address(L=/bin/sh); Let us debug again to guess what is the real address of “/bin/sh”.



Here goes the big challenge, actually I tried several times by using the “/bin/sh”address which is acquired through gdb debugging(0xbffffe9e), unfortunately, I donot know why the address is not correct, it has slightly changed , thus I tried several times the addresses around the debugging address to find the exactly right adderss for “/bin/sh”which is “0xbffffe96”, the following is the exploitlib.c code which will generate badfile.



Compile and run the exploitlib.c program , and we need to compile retlib.c again to remove the debug tag. And run retlib in normal user account.



From the result, we can see that we successfully get the permission of the root.

*Task2: Protection in /bin/bash*

Link /bin/sh to /bin/bash and run the vulnerable program.



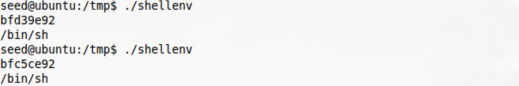
As screenshot shows that we still successfully get the root permission. This situation has perfectly matched the theory we got from previous buffer-overflow lab, that /bin/bash shell has no protection for preventing Set-UID program from reaching bash with effective id. In sum, it is similar as zsh.

*Task3: Address Randomization and Stack Smash Protection*

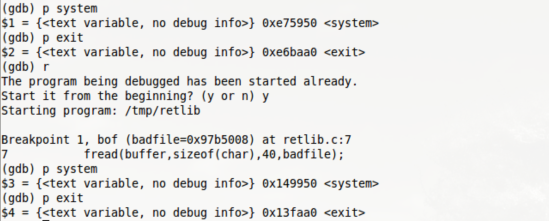
Turn on the randomization and change the shell back to original shell.



let us check the address of MYSHELL environment variable.



we run it two times, and we got two totally different address. This means the randomization worked very well. Then add the debug tag by recompile retlib.c program and do debug in normal user account.



we can tell that the address for system and exit are also changed frequently. Beside I found another interesting things is the difference between the first address of system and the second of system are the same as that of exit, which is “d2b000”, maybe we can guess next random address from this pattern. The difficult thing between buffer-overflow test and this one is that previous lab we only need to guess one address, but this lab we need to guess three address. But we can still do it based on the pattern we got.