# Buffer Overflow Vulnerability Lab

57117217戚吴祺

Turning Off Countermeasures

sudo sysctl -w kernel.randomize\_va\_space=0

gcc -fno-stack-protector example.c

sudo ln -sf /bin/zsh /bin/sh

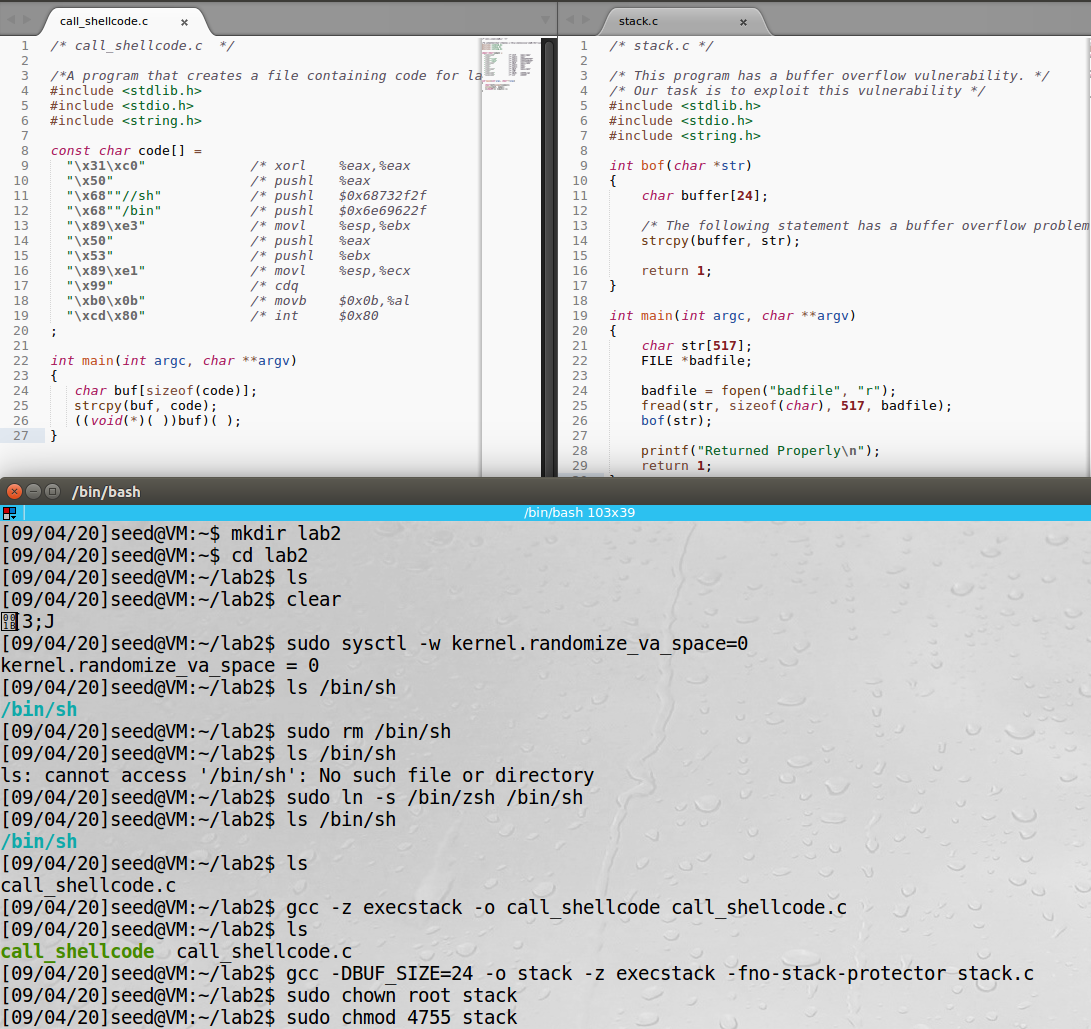
## Task 1: Running Shellcode

gcc -z execstack -o call\_shellcode call\_shellcode.c

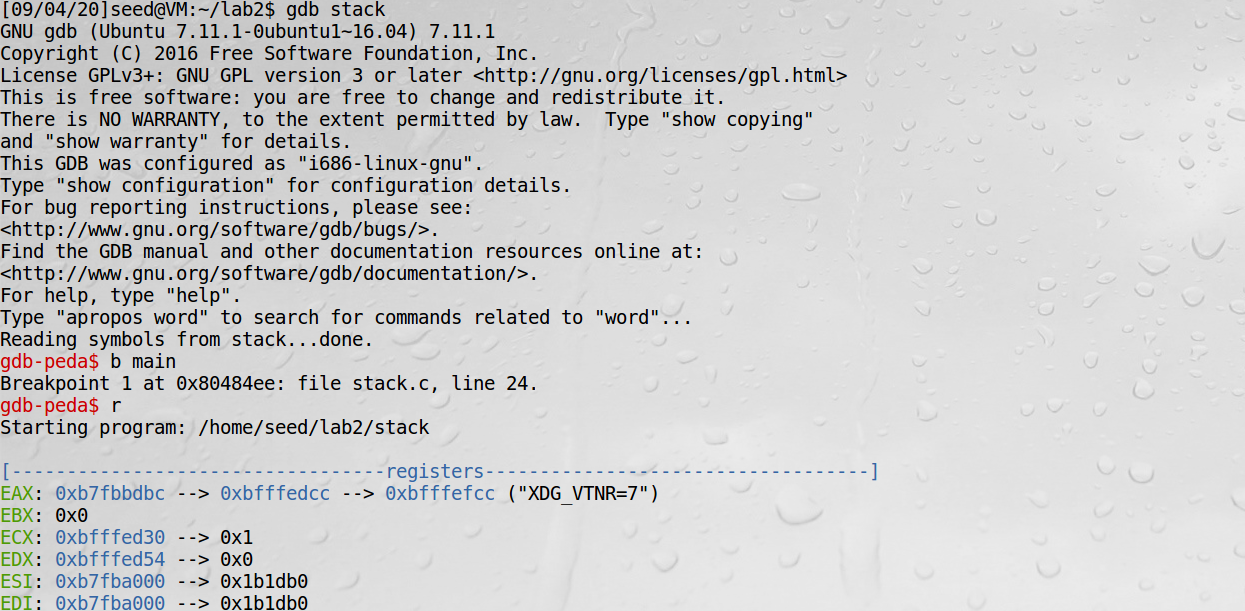
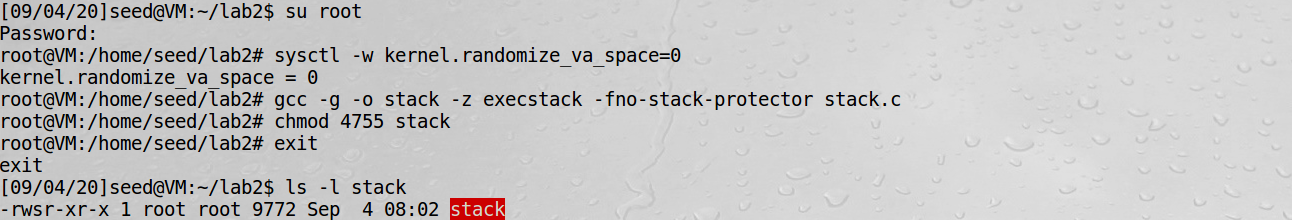
gcc -DBUF\_SIZE=24 -o stack -z execstack -fno-stack-protector stack.c

sudo chown root stack

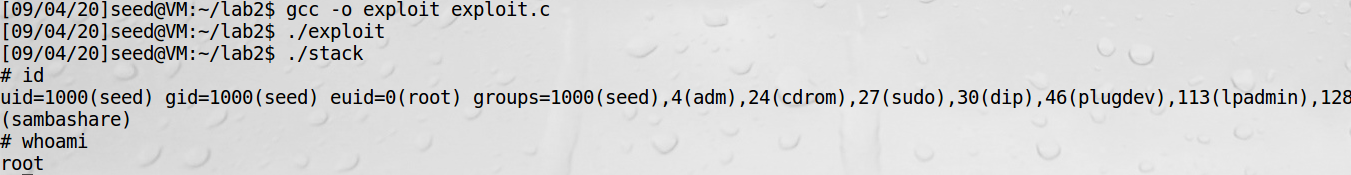
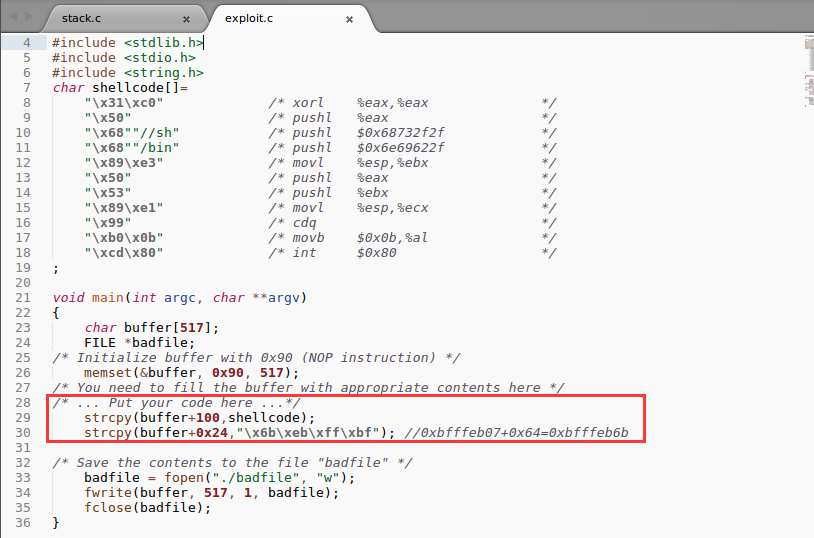
sudo chmod 4755 stack



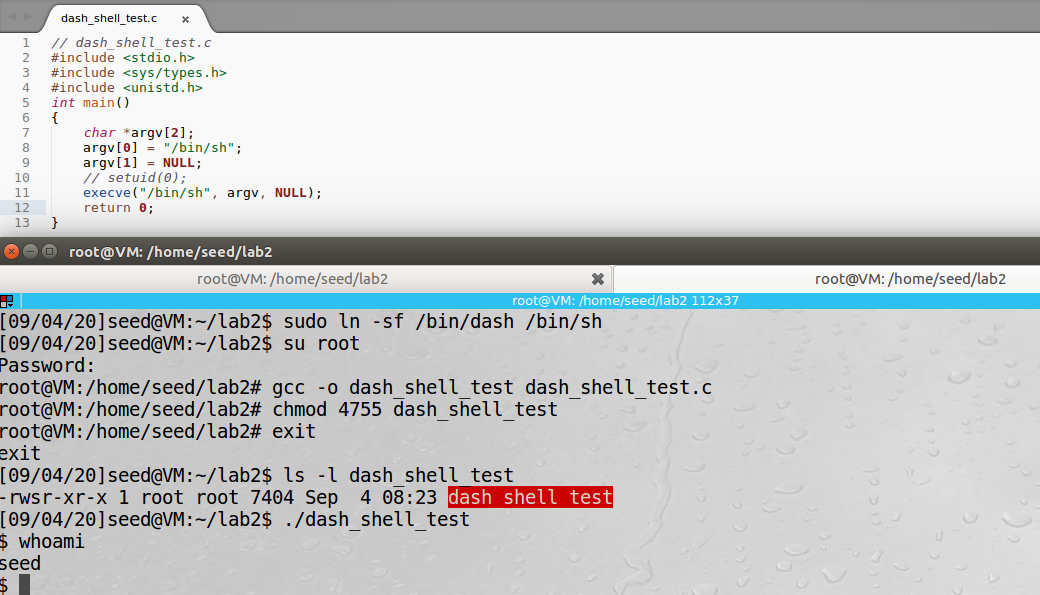
## Task 2: Exploiting the Vulnerability



0xbfffeb07+0x64=0xbfffeb6b

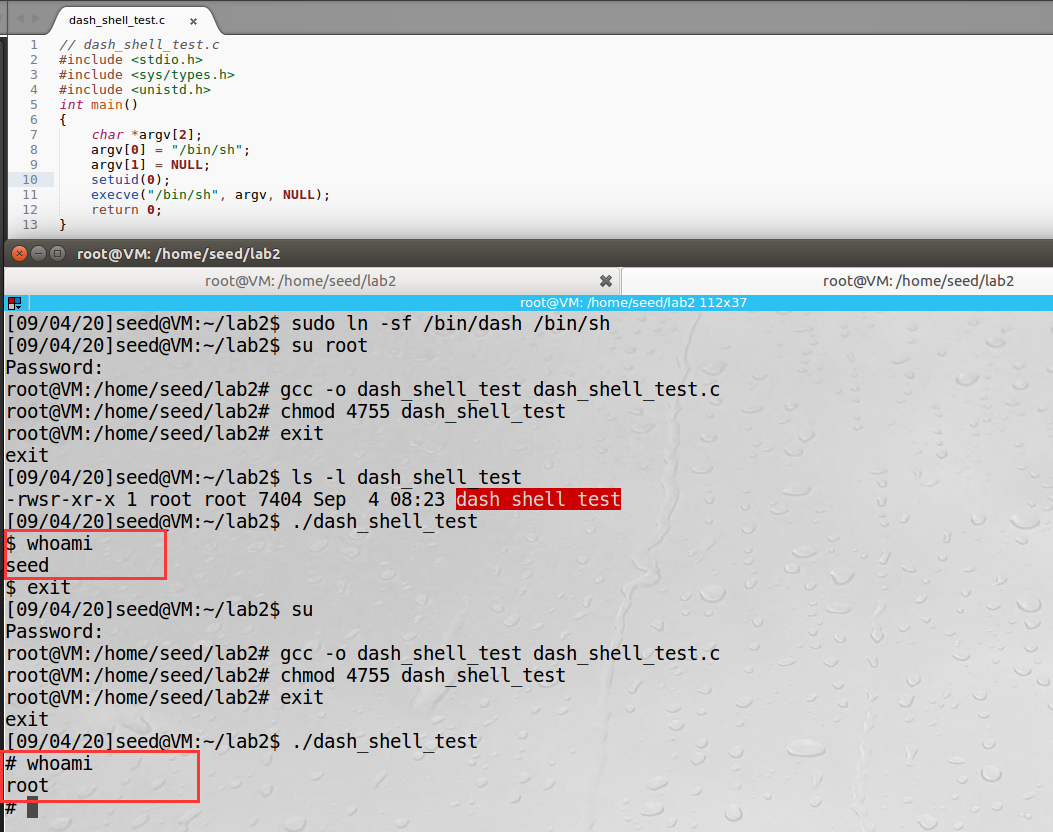


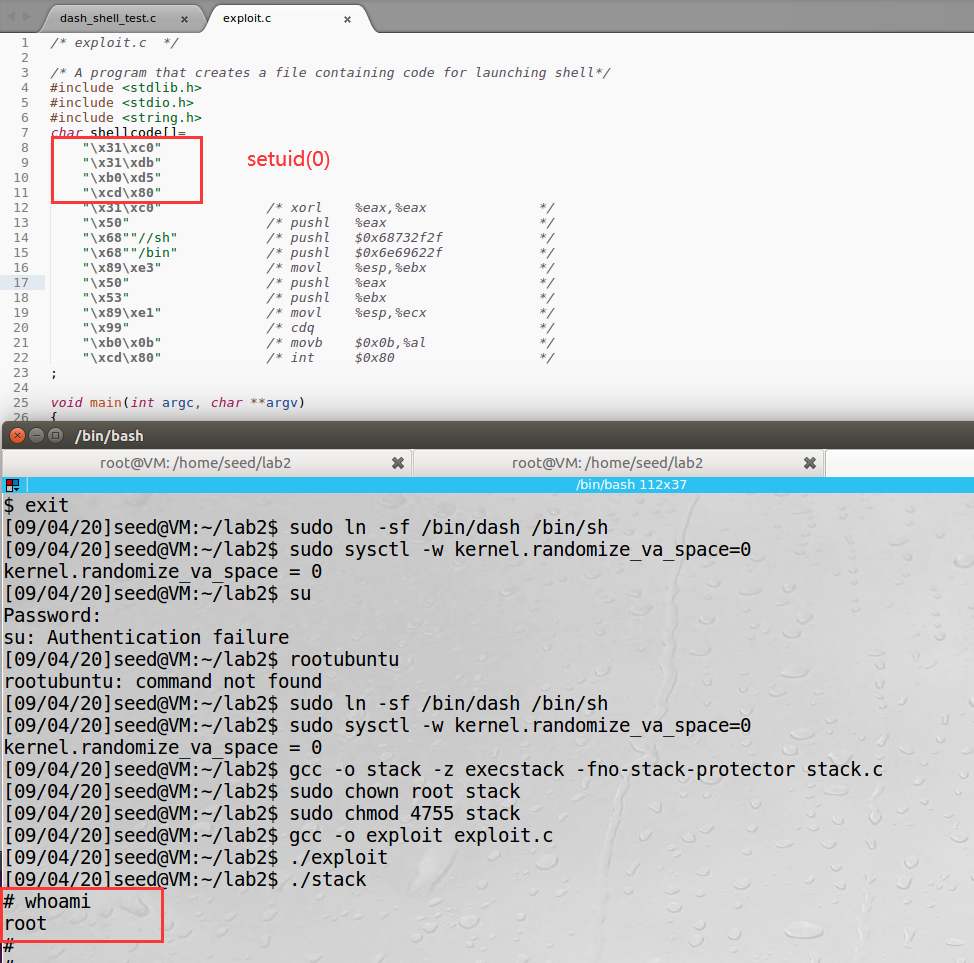
## Task 3: Defeating dash’s Countermeasure



setuid(0)

The real user ID of the victim process was changed to zero before invoking the dash program.

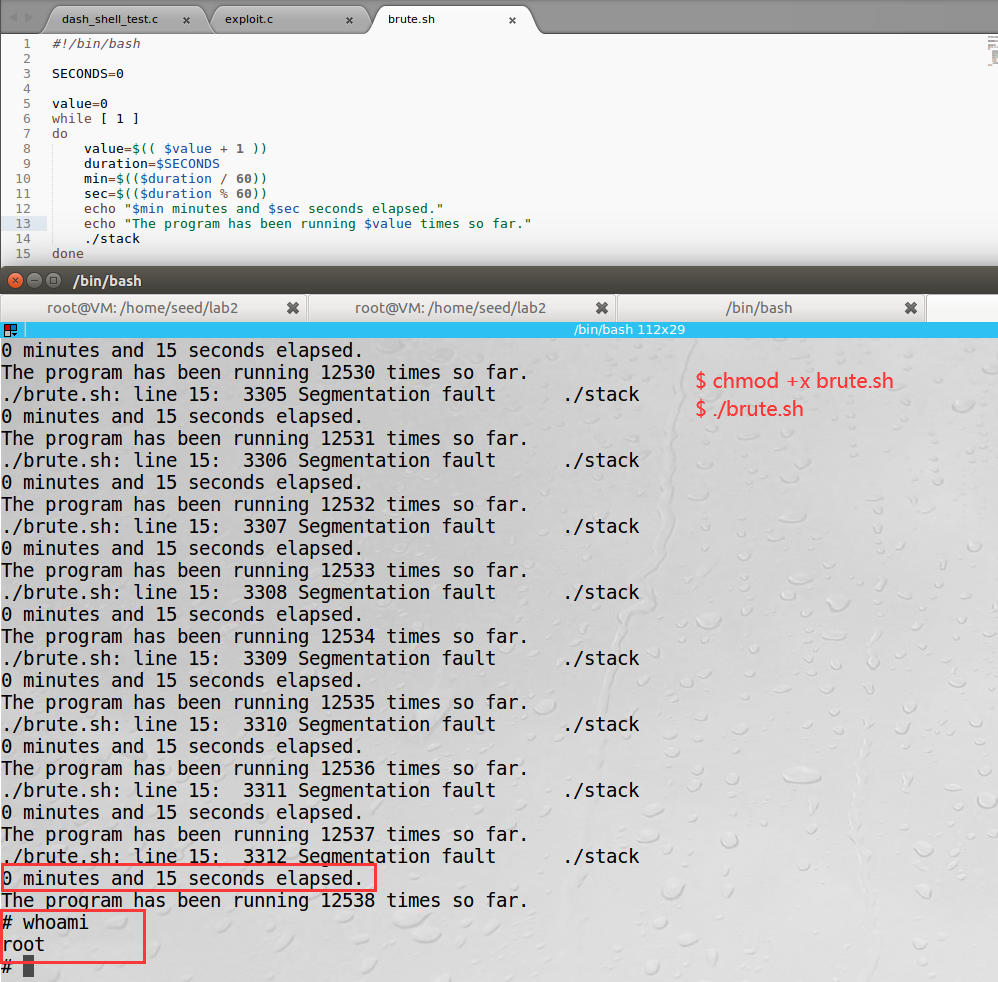




The real user ID of the victim process was changed to zero before invoking the dash program. Therefore, dash’s Countermeasure was defeated.

## Task 4: Defeating Address Randomization

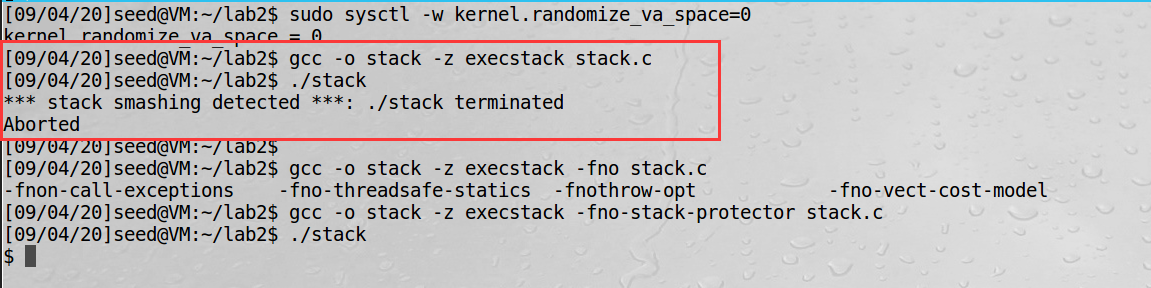
sudo /sbin/sysctl -w kernel.randomize\_va\_space=2



I’m a lucky dog. Just 15 seconds!

Stack base address is not too large and can be exhausted easily with the brute-force approach.

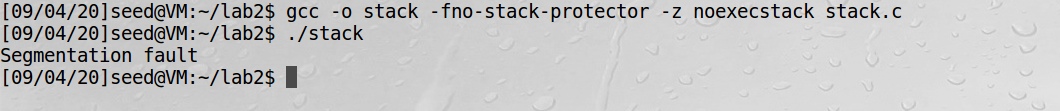
## Task 5: Turn on the StackGuard Protection



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

I fail to get a shell when my vulnerable program using the noexecstack option.

gcc -o stack -fno-stack-protector -z noexecstack stack.c

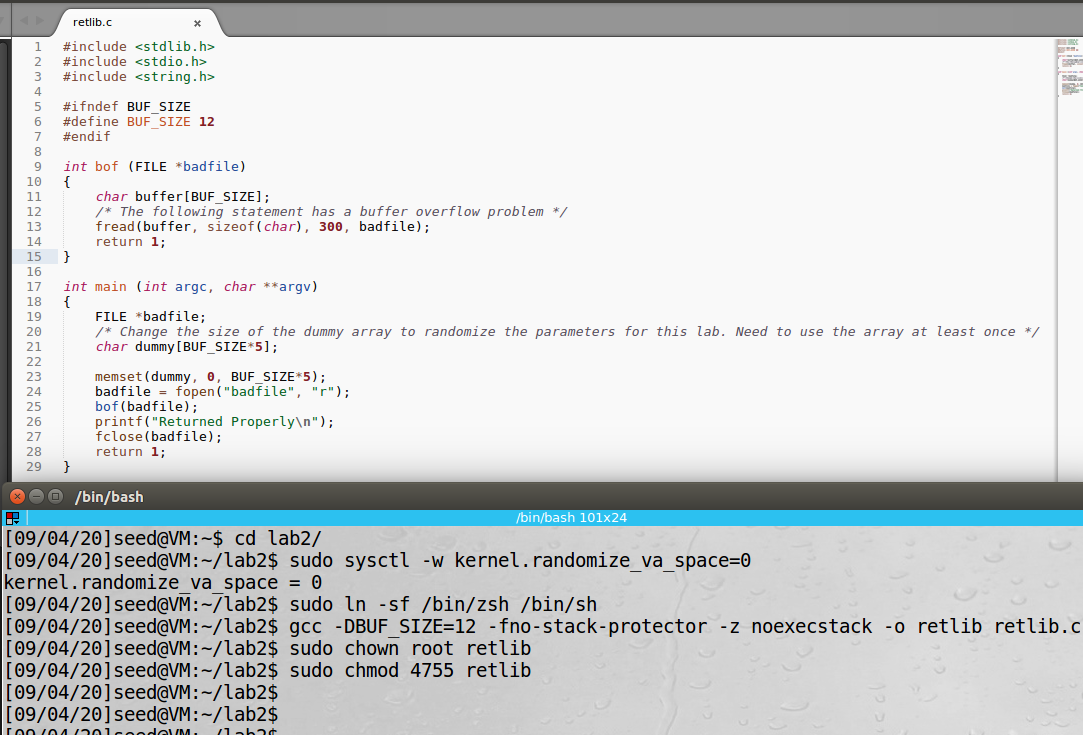


Non-executable stack makes it impossible to run shellcode on the stack.

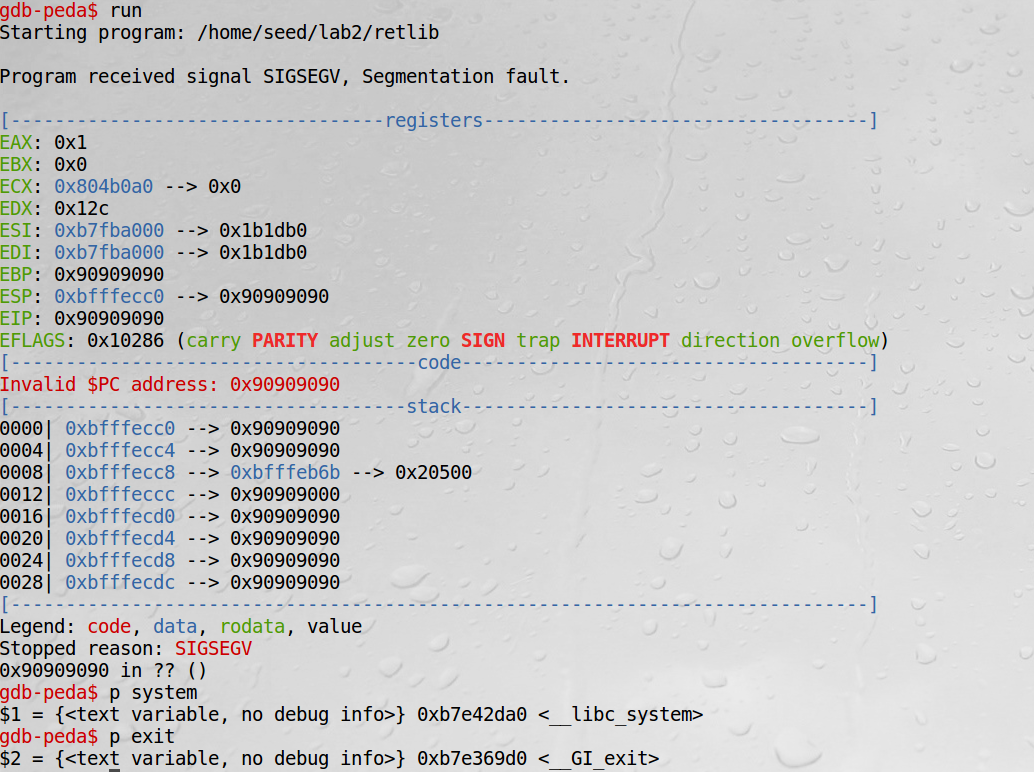
# Return-to-libc Attack Lab

The objective of this lab is to show that the non-executable stack protection does not work.

Compile the program using the "-z noexecstack" option in this lab.



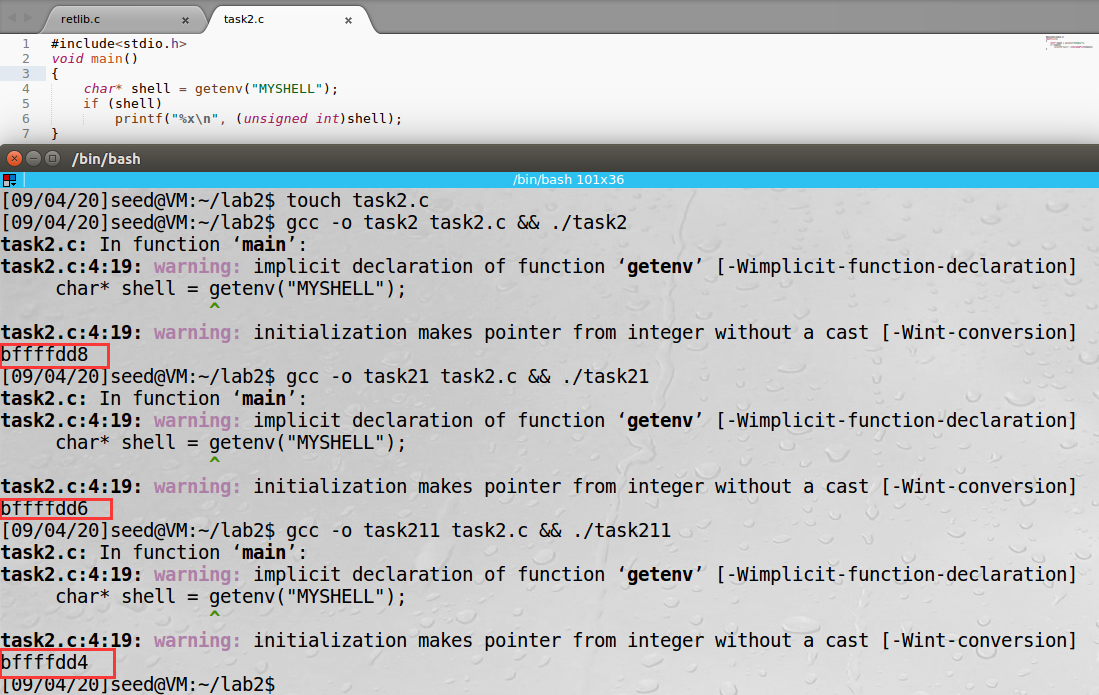
## Task 1: Finding out the addresses of libc functions



**system() 0xb7e42da0**

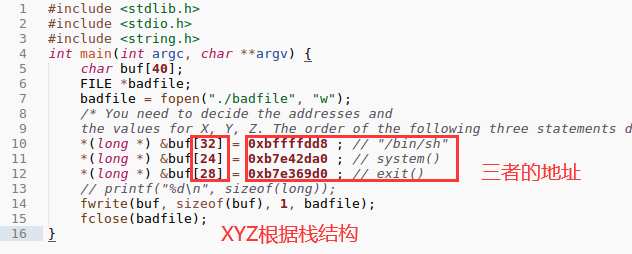
**exit() 0xb7e369d0**

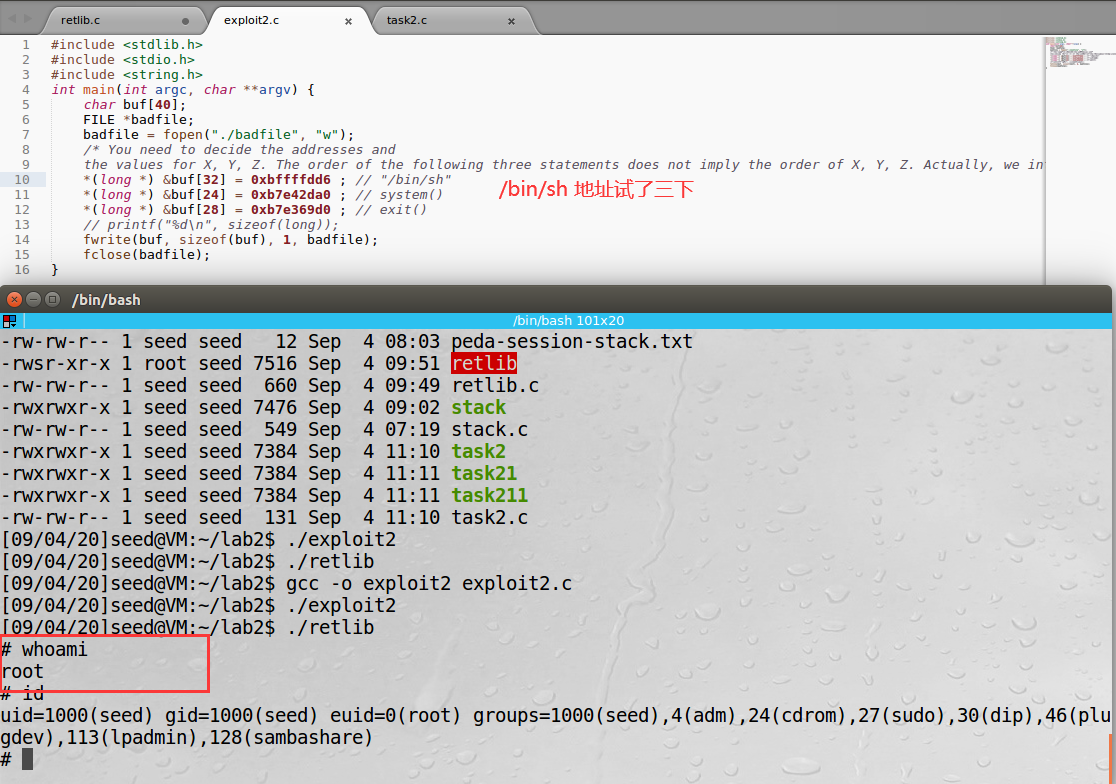
## Task 2: Putting the shell string in the memory



**/bin/sh 0xbffffdd8**

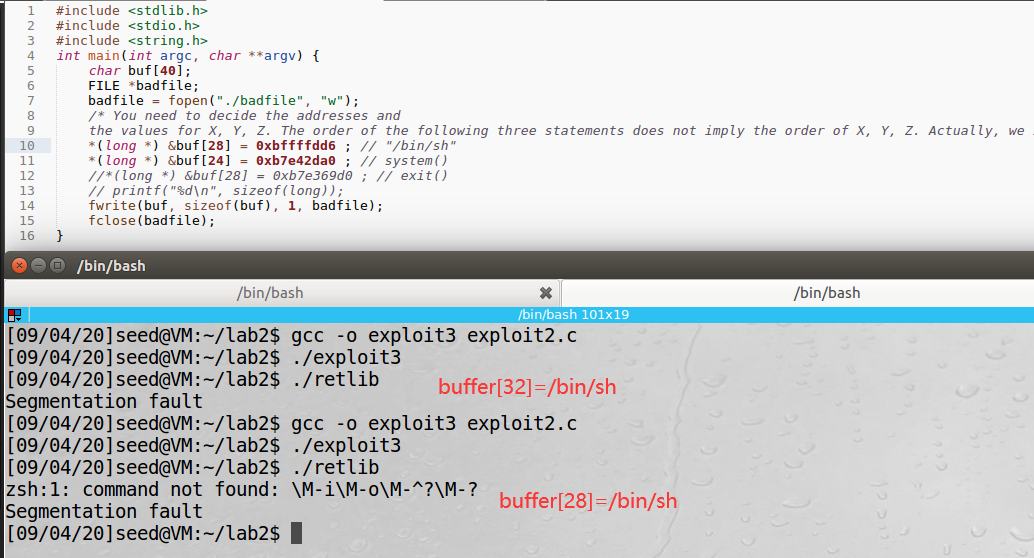
## Task 3: Exploiting the buffer-overflow vulnerability



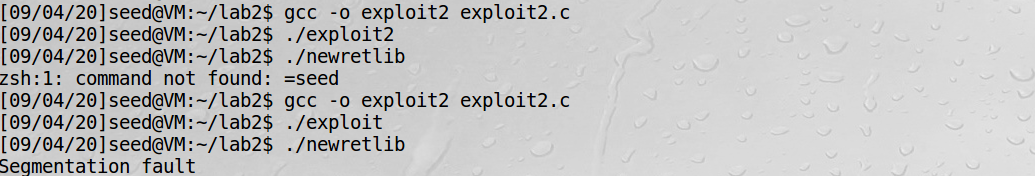


exit() function is really necessary.

If there is a random value and the program is likely to crash when the system function returns, a better idea is to store the address of the exit() function so that when the system function returns, it jumps to the exit() function and terminates the program perfectly.



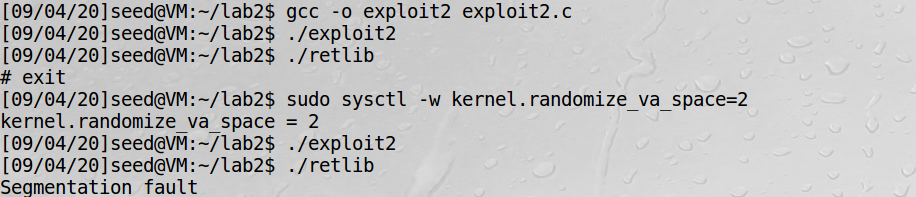
There will be a segmentation fault without exit() because of function call flow.



My attack failed because the address of /bin/sh has changed with the change of filename.

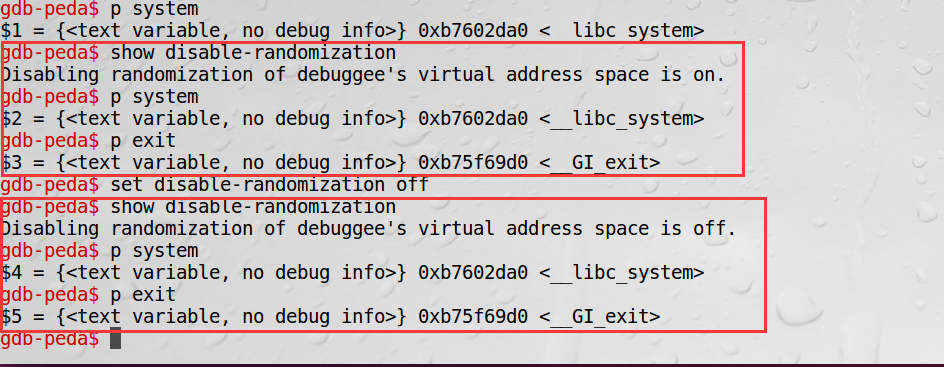
The address of the MYSHELL environment variable is related to the length of the program name.

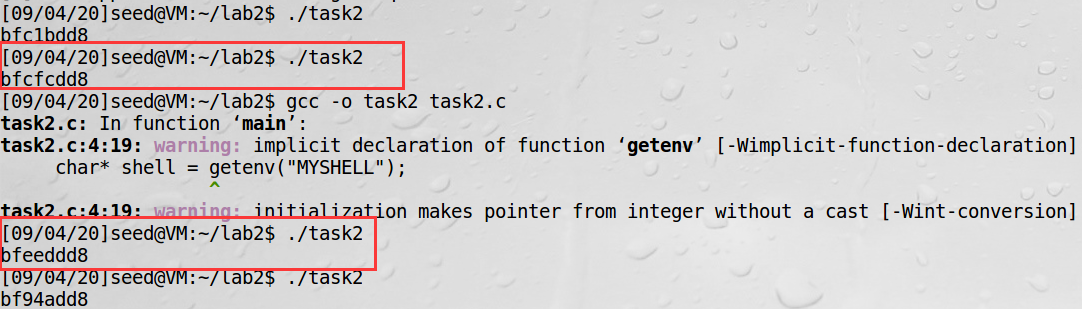
## Task 4: Turning on address randomization



After turning on the address randomization, the address below is incorrect. Therefore, the program will occur a segmentation fault.







The address of /bin/sh changed while that of system() and exit() didn’t change after turning on the address randomization

## Task 5: Defeat Shell’s countermeasure

setuid(0) like the lab above

invoke setuid(0) before invoking system()

In the vulnerable program, we intentionally used fread(), which, unlike strcpy(), is not affected by zeros.

## Task 6: Defeat Shell’s countermeasure without putting zeros in input

optional