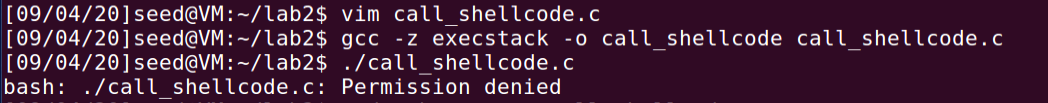
网安实训课程实验报告 lab2

57118115 陈烨

Buffer Overﬂow Vulnerability Lab

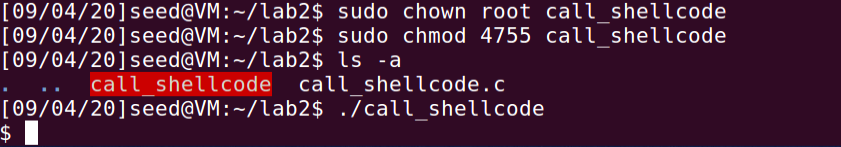
**Task1: RunningShellcode**

Run the program and describe your observations.



Call\_shellcode程序不能打开shell

提升call\_shellcode程序的权限后



Call\_shellcode成功调用了shell

**Task2: Exploiting the Vulnerability**

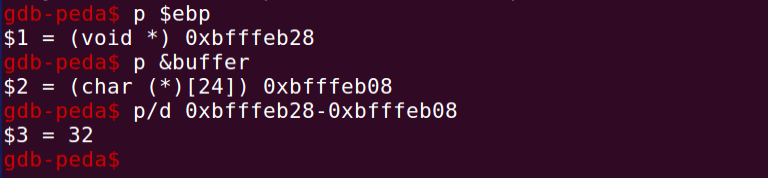
**实验之前特别注意要sudo sysctl -w kernel.randomize\_va\_space=0**

在实验过程中ebp寄存器的值不能改变

编译stack程序的时候加上gcc的参数-g使得可以进行调试，

进入gdb，在bof函数之前设置断点 ，run运行stack，程序停留在bof入口

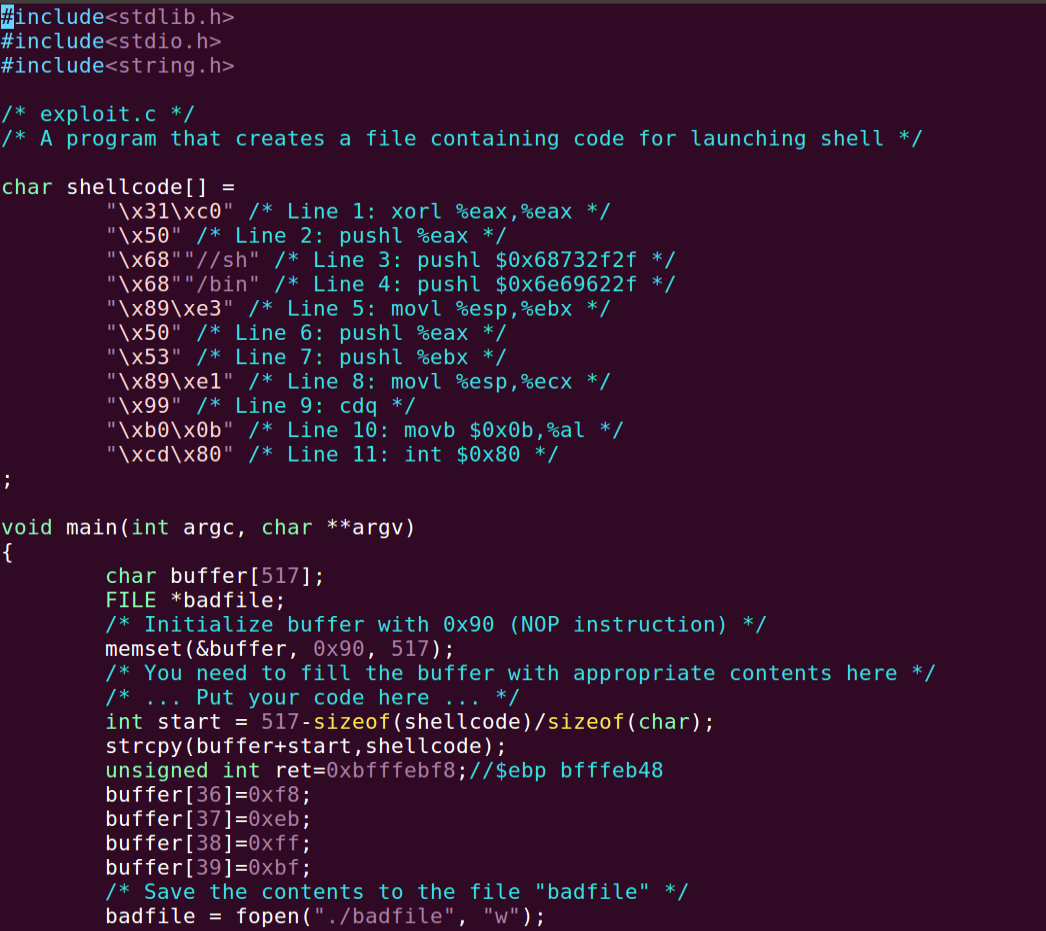
查看寄存器中存储的数据如下：



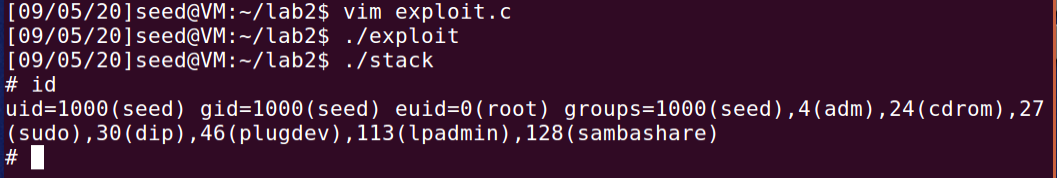
查看到了ebp寄存器的值，已经buffer数组的地址，二者相差32+4=36（ebp寄存器此时的值需要加上4才是bof函数返回地址所在的内存地址）

接下来修改exploit.c程序，源码如下：

其中从buffer[36]-buffer[39]是原先的return address，我们希望把他覆盖修改成导向恶意代码的地址，修改地址为ebp地址后与ebp相差为4的倍数，buffer是小端读取

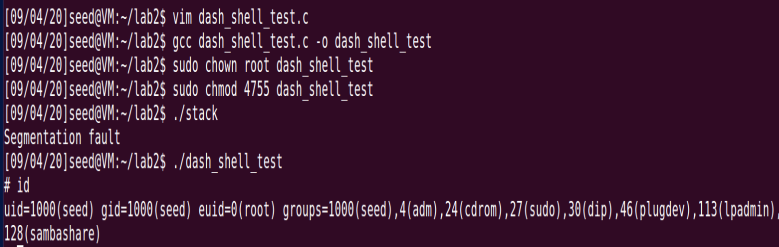


运行exploit和stack，拿到了root权限的shell

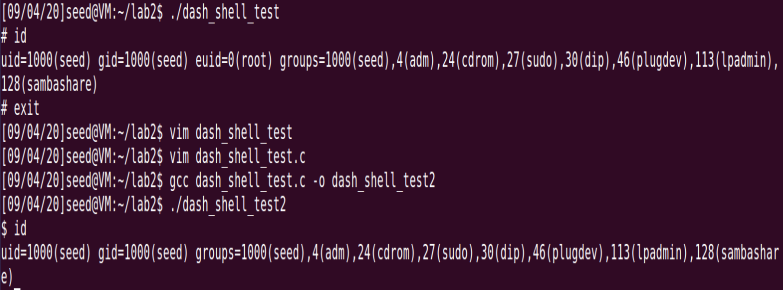


**Task 3: Defeating dash's Countermeasure**

编写编译dash\_shell\_test程序，在取消注释setuid(0)的情况下可以看到，shell为root的shell



然后将Setuid(0)注释，编译dash\_shell\_test2



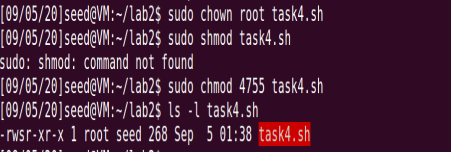
可以看到，获得的shell为普通用户的shell

如果RUID与EUID不同，或者用户组的RUID与EUID不同时，dash会降低权限从而进行保护。通过setuid(0)将RUID也设置成root，就与euid相通，那么dash就不会降低权限了。

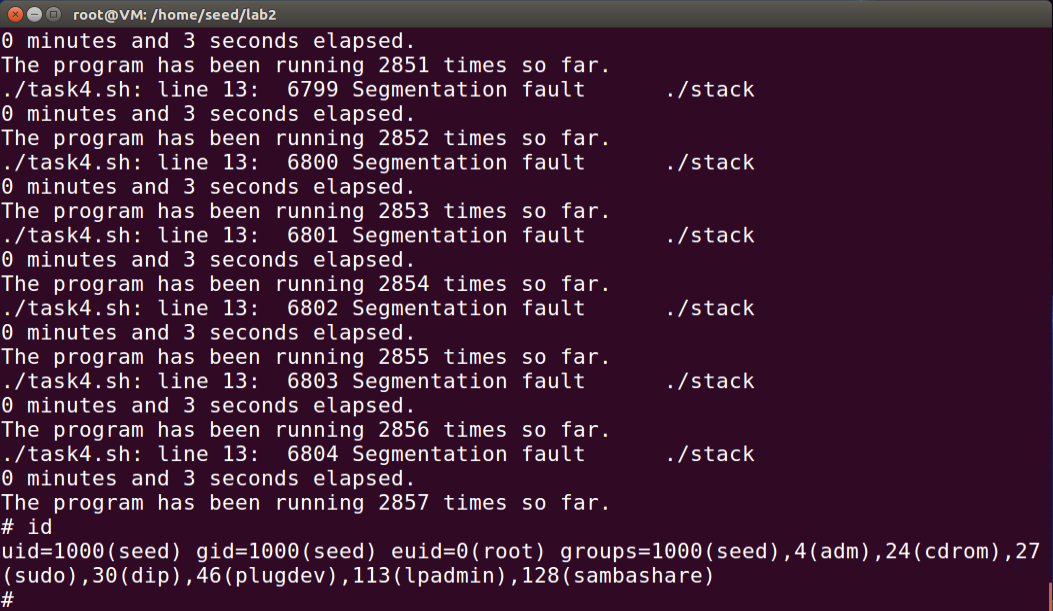
**Task 4: Defeating Address Randomization**

编写task4.sh 脚本程序

提升为root用户的setuid程序



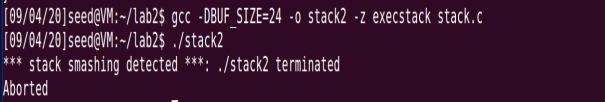
开始运行task4.sh脚本，大约4s后得到root权限的shell



**Task 5: Turn on the StackGuard Protection**

重新编译stack.c成stack2，取消-fno-stack-protector

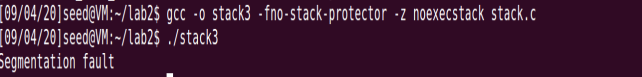
运行stack2报错，并且报告出错误原因，被栈溢出攻击了



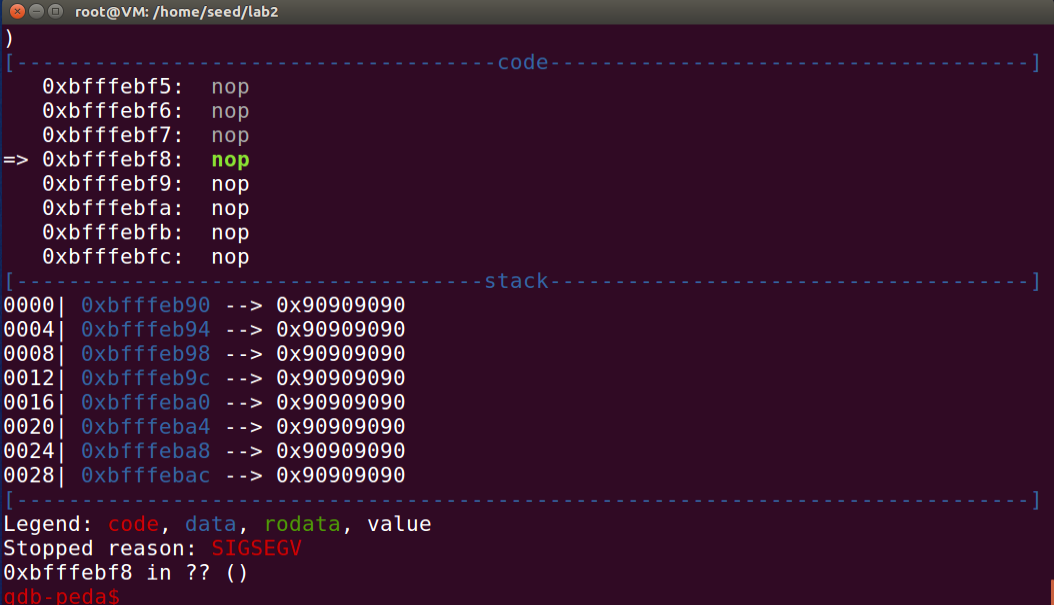
**Task6: Turn on the Non-executable Stack Protection**

实验开始前，关闭地址随机化

重新编译stack.c成stack3程序，在gcc编译时选择-z nostackexec选项，再次运行stack3时报错



进入gdb调试，运行stack3后



从报错结果可以看到，在执行第一个shellcode指令的时候就发生了错误（shellcode中设置的返回地址就是0xbfffebf8），所以只要执行的指令的地址位于栈上，程序就会报错