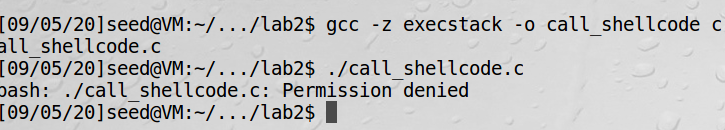
网安实训课程实验报告 lab2

57118135 许皓钦

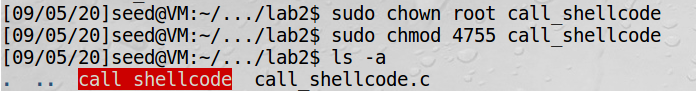
Buffer Overﬂow Vulnerability Lab

**Task 1: Running Shellcode**



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

**Task 2: Exploiting the Vulnerability**

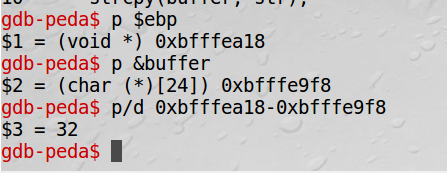
**实验之前特别注意要sudo sysctl -w kernel.randomize\_va\_space=0**

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：

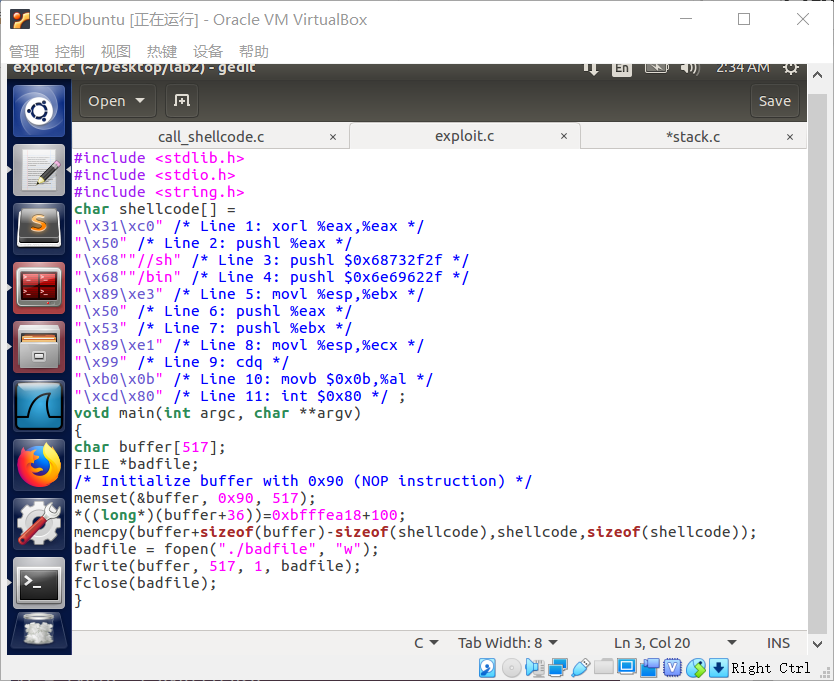


使⽤ p $ebp 查看 ebp 寄存器此时的值，使⽤ p &buffer 查看 buffer 数组的地址。 ebp 寄存器此时的

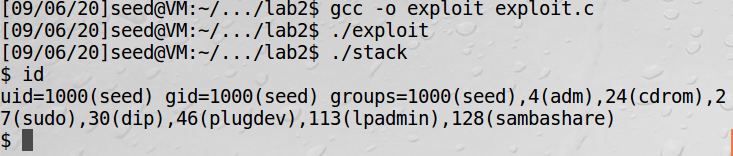
值再加4就是 bof 函数返回地址所在的内存地址，那么通过 p/d 就可以计算出 buffer 与 bof 返回地址

之间的距离为32+4=36。

因此，exploit.c`的程序为：

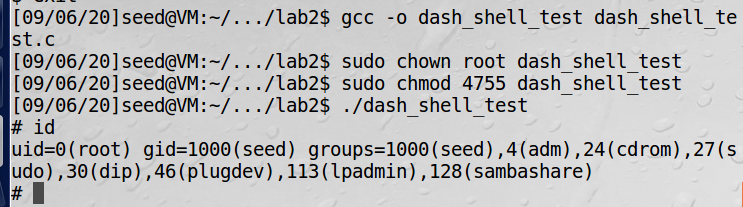


运行exploit和stack，拿到了root权限的shell

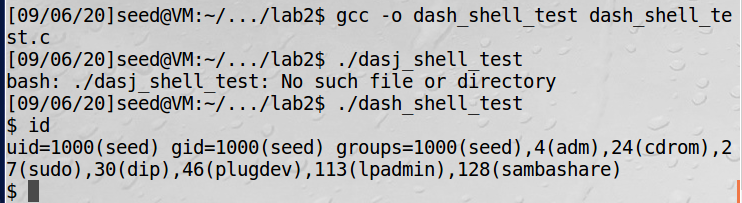


**Task 3: Defeating dash’s Countermeasure**

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



获得到的Shell为root的Shell。

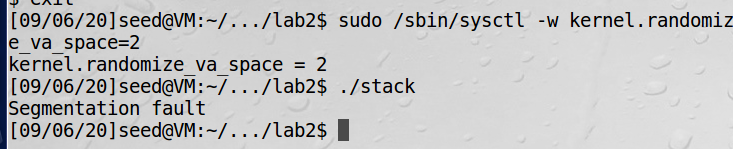
将 setuid(0) 的汇编代码加⼊shellcode后，在 dash 下运⾏ stack 也能获得root的Shell了。

如果RUID与EUID不同，或⽤户组的RUID与EUID不同时， dash 会降权限。⽽我们通过 setuid(0) 将

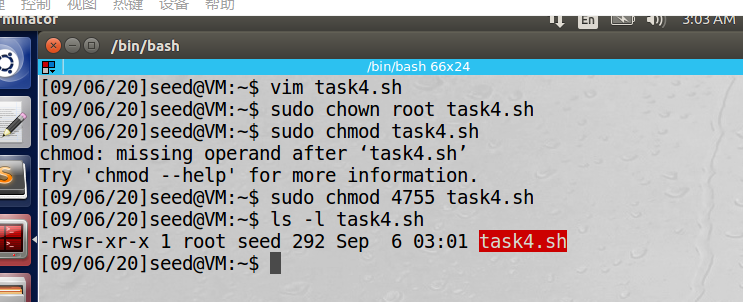
RUID也设置为root，就与euid相通了，那么 dash 就不会降权限了。

**Task 4: Defeating Address Randomization**

关闭地址随机化，首先设置，然后用task2的攻击方法，发现无法攻击成功，出现因地址不对而产生的段错误



使用随机化后，使用while循环语句，对其攻击

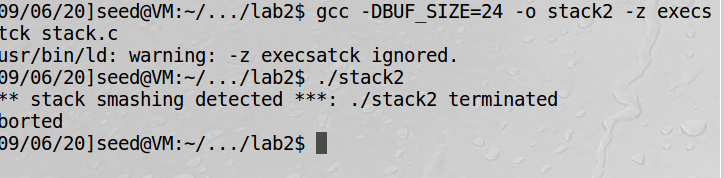


执行一段时间，始终无法获得root权限，可能是随机性太强的缘故

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

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

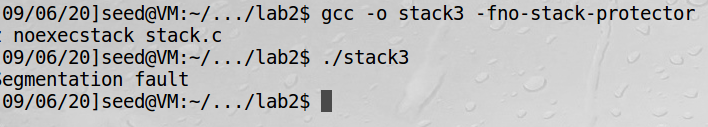
运行stack2报错，并且报告出错误原因，被栈溢出攻击了

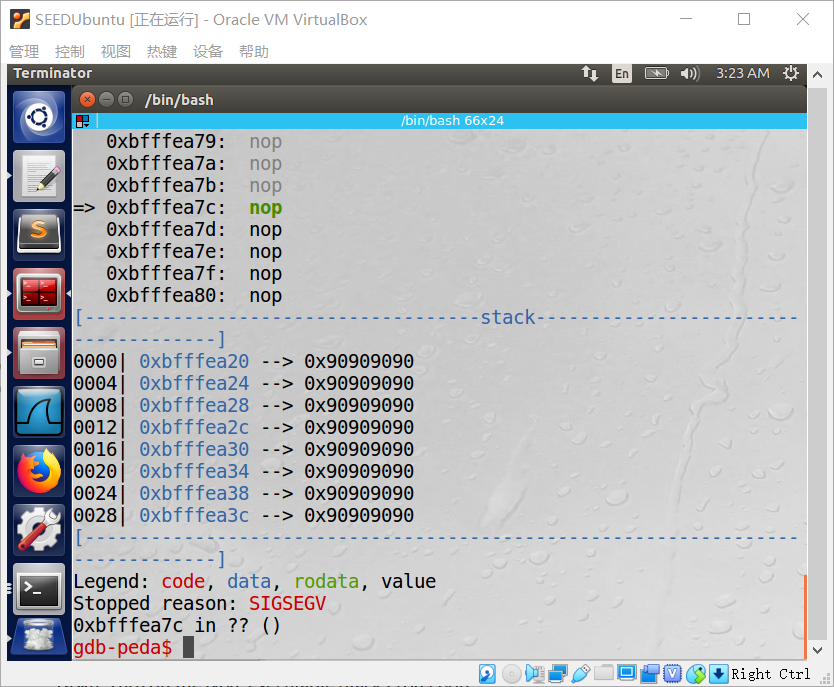


**Task 6: Turn on the Non-executable Stack Protection**

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0xbfffea7c），所以只要执行的指令的地址位于栈上，程序就会报错