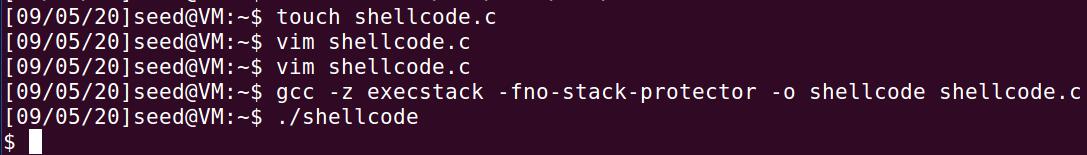
**Lab2实验报告**

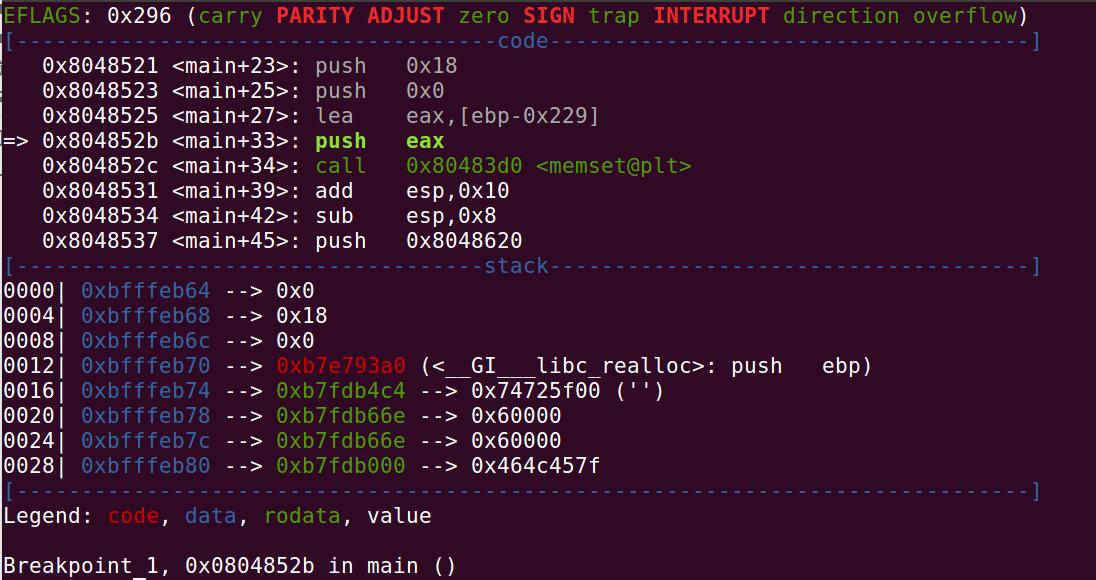
57117124 陈浩

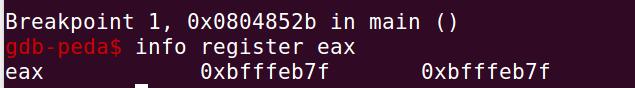
**（1）缓冲区溢出**

任务一：将所给程序使用gcc的-z execstack选项进行编译，运行后即成功调用了shell



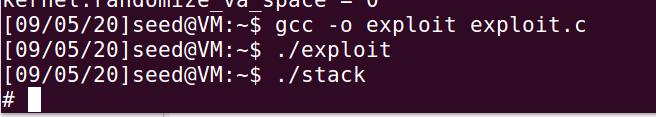
任务二：对stack.c进行编译，BUF\_SIZE值为默认的24，然后调试，找到buffer地址以及$ebp寄存器的值。在bof入口设置断点，命令run运行。$ebp的值指向前一个栈帧的地址，计算出buffer的起始位置与ebp之间的差值为32：

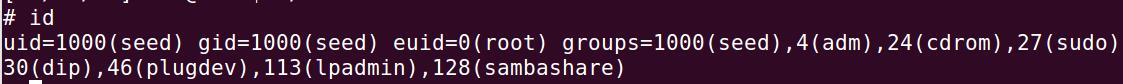




根据语句 strcpy(buffer+300,shellcode); 我们计算shellcode的地址为 0xbfffeb7f(十六进制)+300(十进制)=0xbfffecab(十六进制), 现在修改exploit.c文件！将 \x??\x??\x??\x?? 修改为 \xab\xec\xff\xbf。

攻击结果：





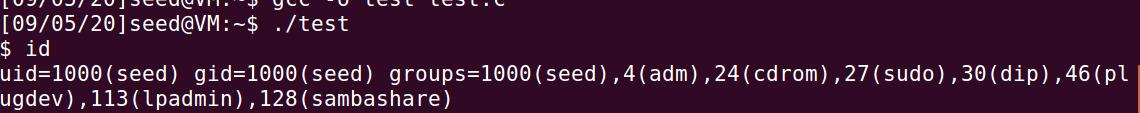
通过攻击，获得了root权限。

任务三：Defeating dash's Countermeasure

将所给程序dash\_shell\_test.c中的setuid(0)注释，调用stack，查看id命令，再取消注释，查看id，两者进行对比：



注释后，获取的shell为用户seed的shell

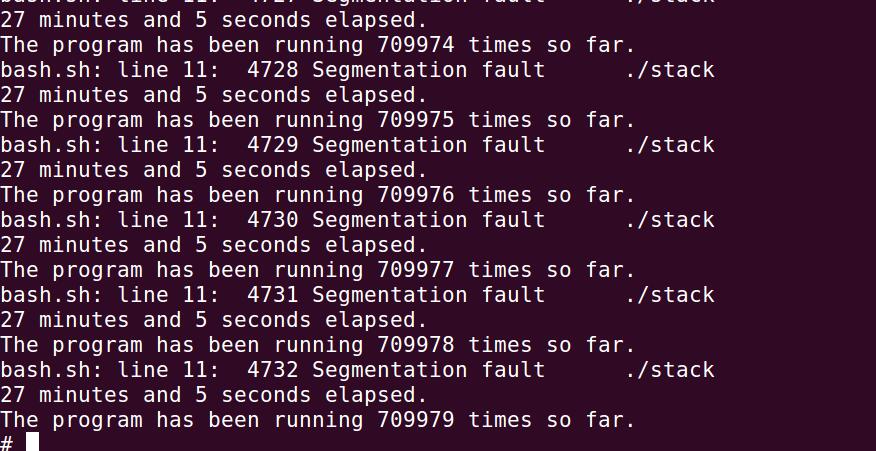


取消注释后，获取的shell为root用户的shell

把setuid(0)也编码到shellcode中，在dash下运行stack，就能发现同样得到了有root权限的shell。

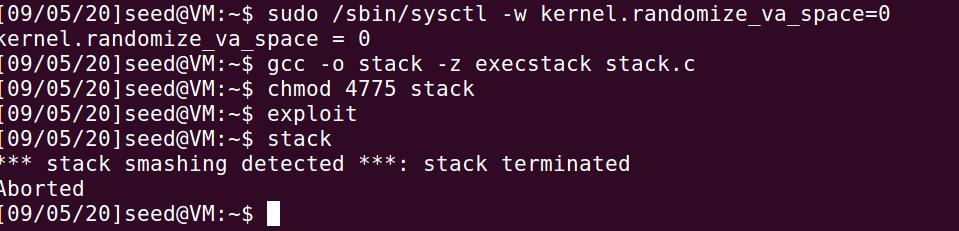
任务四：Defeating Address Randomization

使用命令sudo /sbin/sysctl -w kernel.randomize\_va\_space=2，打开Ubuntu的地址随机化，利用提供的程序进行暴力攻击，运行时间略长。



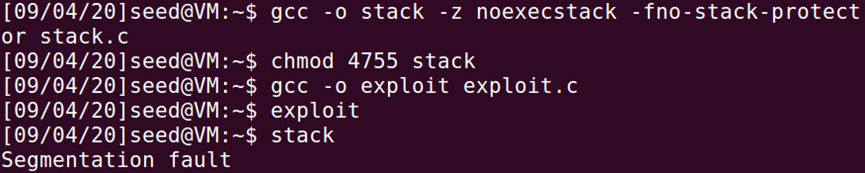
任务五：Turn on the StackGuard Protection

首先要先关闭地址随机化，然后编译不带-fno stack protector选项的程序，重新编译易受攻击的程序stack.c，可以看出由于栈保护机制，攻击失败。



任务六：Turn on the Non-executable Stack Protection

这次编译stack.c文件时，关闭栈可执行，然后执行Task 1相同操作。



攻击失败

**（2）Return-to-libc攻击实验**

任务七

编译执行所给的retlib程序，BUF\_SIZE设置为36。设置为setuid程序，Return-to-lib攻击本质还是缓冲区溢出攻击的一种，所以我们需要一段带有缓冲区溢出漏洞的程序。我们的目标是获取带有root权限的shell。

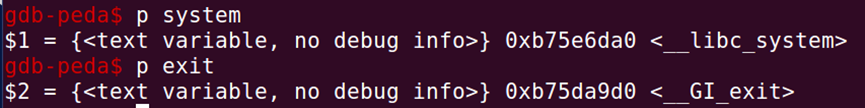
攻击程序用于给带有漏洞的程序传入参数，这个参数是一些指令，通过指令运行我们期望的sh。



其中addr1、addr2、addr3分别为BIN\_SH、system、exit 的地址。我们下一步要做的就是获取这些地址：

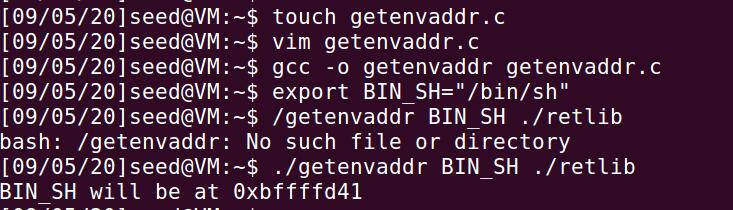
获取system和exit的地址

这两个程序是驻留在内核态的，所有程序共享内核态的函数，因此我们可以使用gdb来获取这两个程序的地址。



任务八

获取/bin/sh地址



任务九 实现攻击

填写exploit文件Task 2的地址和ebp、buffer的地址填上即可



发现获得了root的shell

任务十一 Turning on address randomization

用sudo sysctl -w kernel.randomize\_va\_space=2打开地址随机化保护，重新实施上次攻击，出现Segmentation fault. 因为打开保护之后/bin/sh的地址不再固定，通过之前的步骤可以查看，会发现地址一直变化。