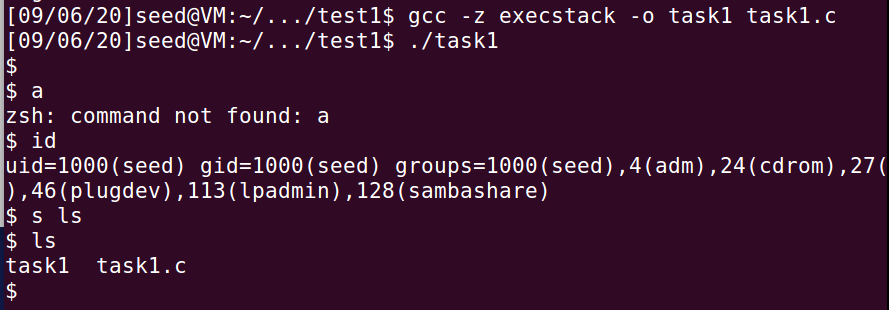
Lab2-report

57117208 傅靖邦

task1

运行文件代码，成功进入shell



task2

电脑屏幕的照片上有文字

描述已自动生成屏幕上有字

描述已自动生成

从上图可以看到，漏洞程序读取badfile 文件到缓冲区str，且str的地址为0xbffff437，计算上shellcode偏移量100（0x64）,则shellcode地址为0xbffff49b。偏移量为0x24

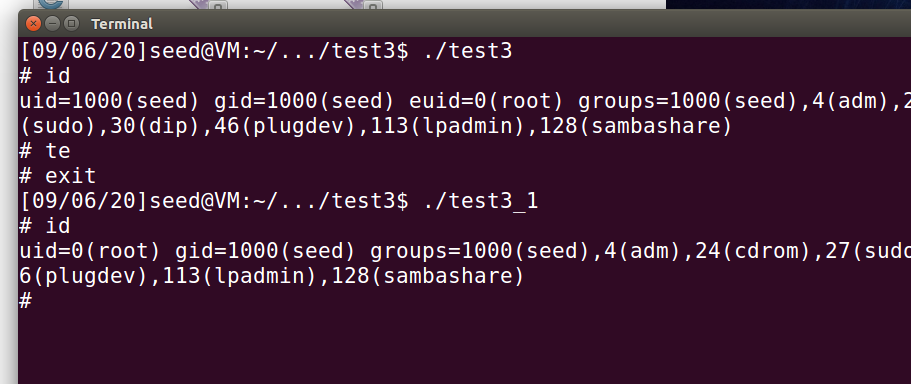
所以exploit.c代码中加入

﻿strcpy(buffer+100,shellcode);

strcpy(buffer+0x24,"\x9b\xf4\xff\xbf");

task3

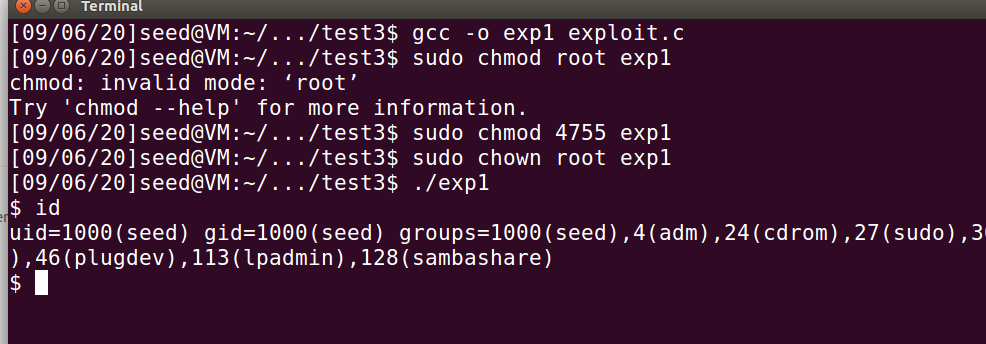
编译dash\_shell\_test.c文件生成test3（不设置uid的可执行文件）和test3\_1（设置uid的可执行文件），分别运行得到结果。



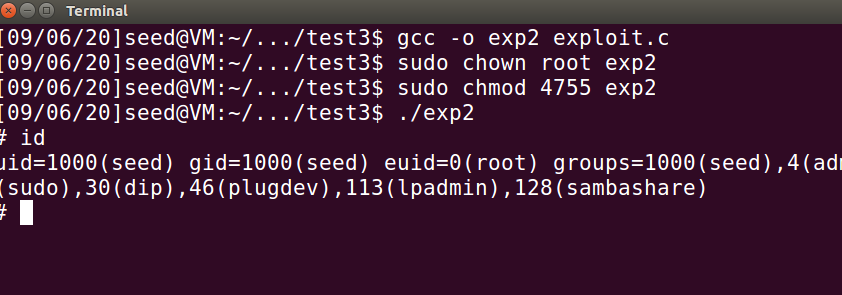
发现，设置uid后获取shell能得到root权限，否则获取的shell没有root权限。

编译exploit.c生成exp1（不设置uid的可执行文件），和exp2（设置uid的可执行文件），运行。

exp1



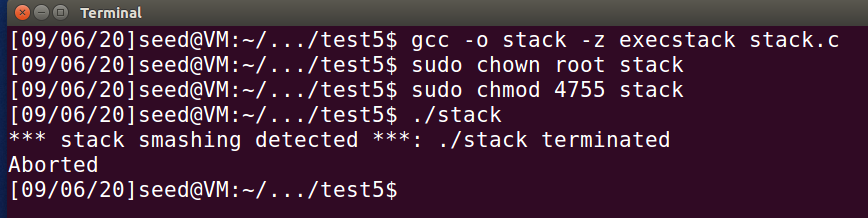
exp2



可以发现未设置uid，shellcode获取权限为user权限，设置uid后shellcode获取root权限。

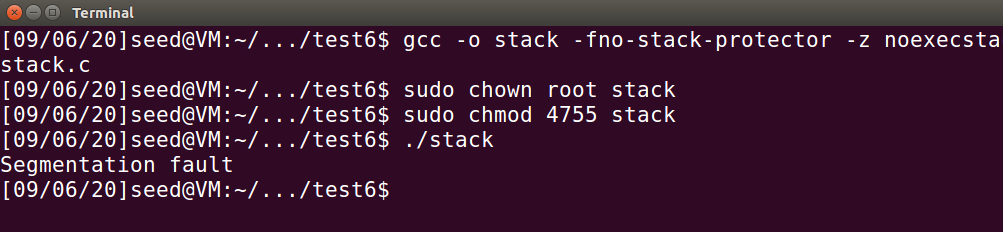
task4

task5



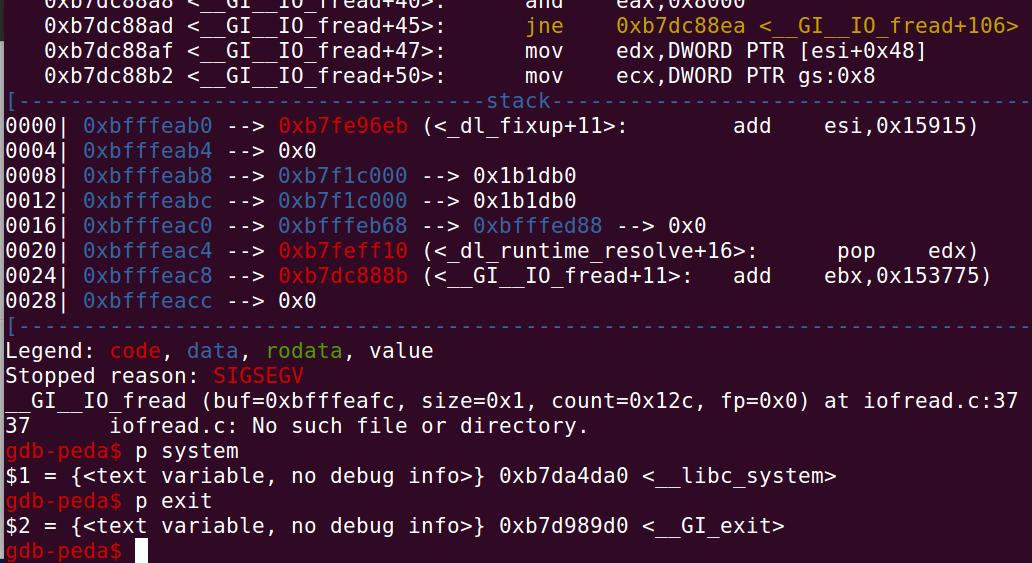
检测到了栈溢出，程序终止。

task6



段中的代码不能执行，报错Segmentation fault.

task1

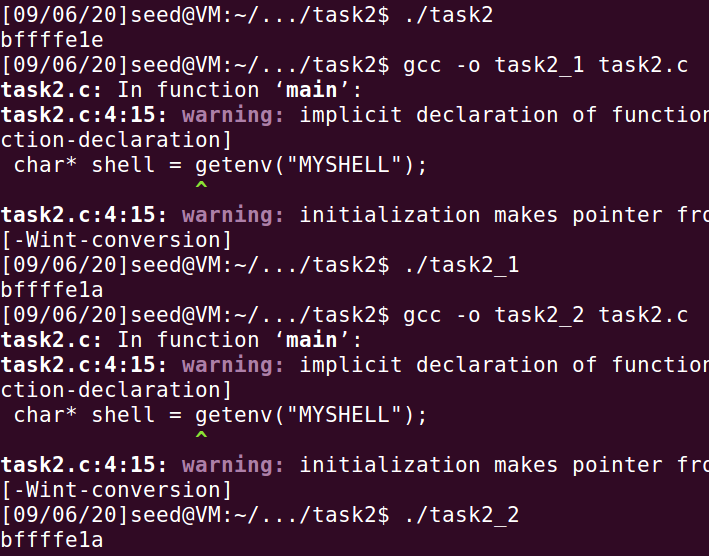


通过gdb运行程序可看出，system函数的地址是0xb7da4da0

exit函数的地址是0xb7d989d0

task2

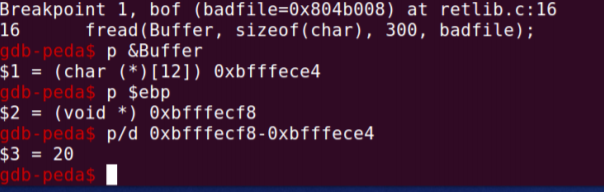
编译并多次运行task2.c



环境变量的地址在0xbffffela

task3

使用gdb查看retlib.c中bof函数执行时，Buffer到ebp的距离。

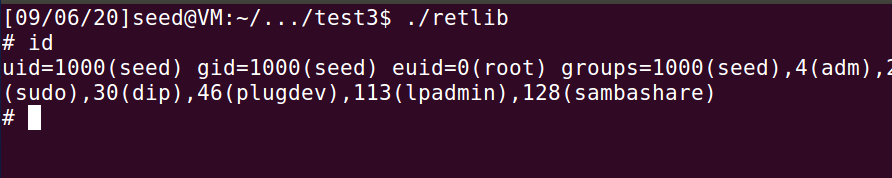


距离为 20。

system()偏移为 20+4=24，即 Y=24：

计算得到 exit()偏移为 Z=20+8=28，shell 变量偏移为 X=20+12=32，并将地址填入空缺代码

编译并运行，

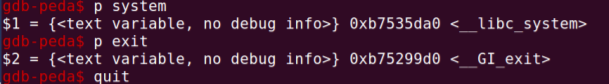


获得了shell

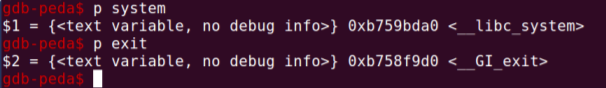
task4

打开虚拟地址随机化，gdb 调试 retlibc 程序：

第一次：



第二次：



system 和 exit 函数的地址都发生了变化

查看环境变量：





综上，地址随机化不会改变偏移量 XYZ 的值，但变量地址、system 函数地址和 exit 函数地 址都会发生改变，因此攻击失败。