**Buffer Overflow Vulnerability Lab**

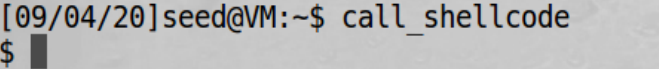
**实验报告**

57118112-王怡乐

注：本实验将BUF\_SIZE设置为12

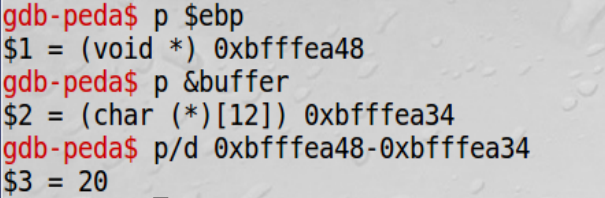
**Task1:Running Shellcode**

编译并执行call\_shellcode.c文件后获得shell：

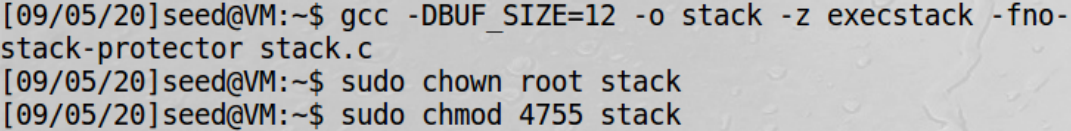


**Task2:Exploiting the Vulnerability**

用gdb调试所给程序stack.c，在函数bof（）处设置断点，获得ebp和buffer的地址：



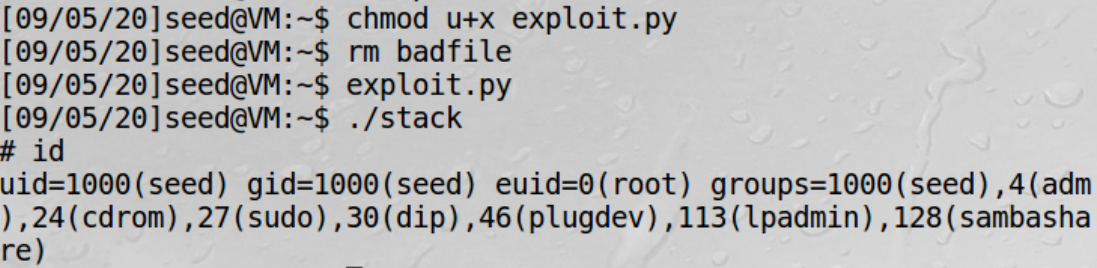
将stack设置为owner为root的特权程序：



编写exploit.py程序，此程序将生成文件badfile：

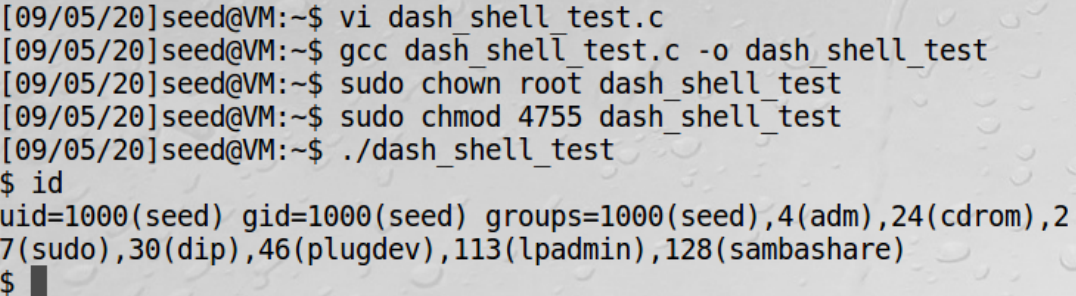


运行后成功获得有root权限的shell:

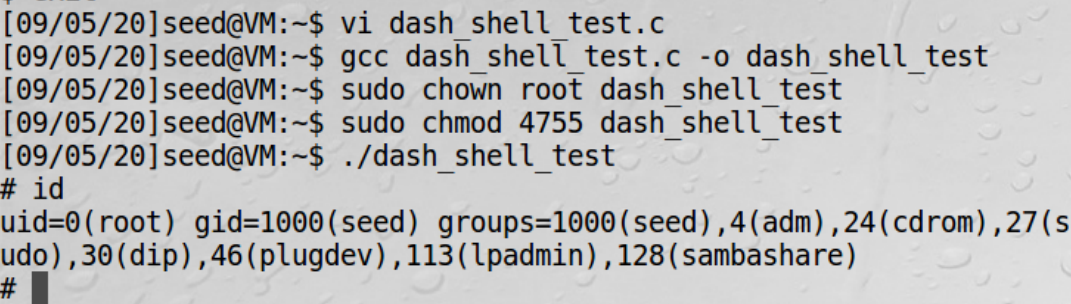


**Task3:Defeating dash’s Countermeasure**

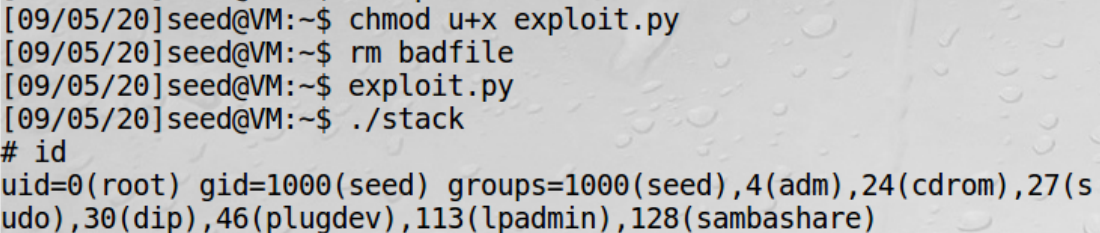
在setuid(0)被注释的情况下，由于/bin/sh被链接到/bin/dash， Ubuntu16.04在检测到程序的EUID和RUID不同时，会自动给程序降权，故只能进入普通用户：



在setuid(0)为非注释的情况下，可进入特权用户模式：

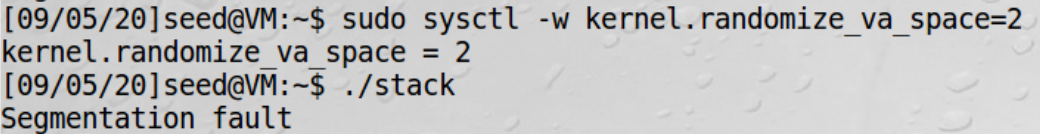


添加语句后再次执行Task2的语句，发现可进入特权用户模式。这是因为在执行原来的shellcode语句（调用shell）之前就已将程序的用户ID改成0，获得了root权限，执行后即可进入了root模式：

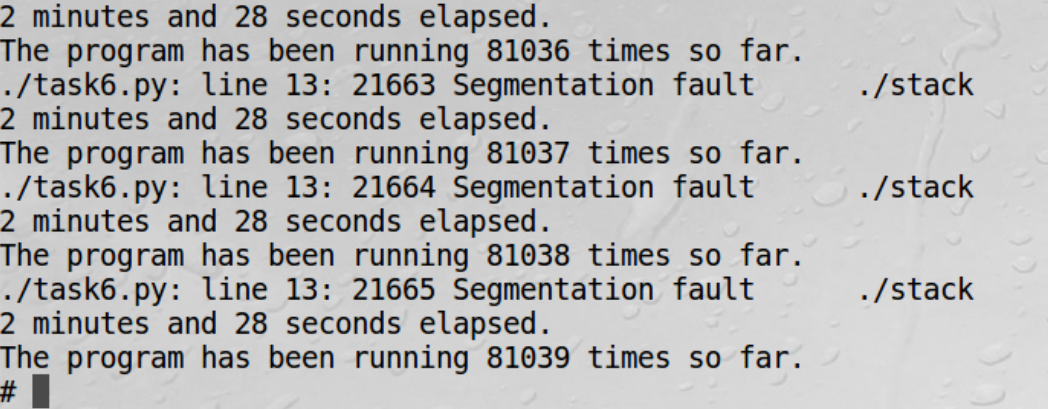


**Task4:Defeating Address Randomization**

由于开启了地址随机化，程序运行栈的基地址是随机的，所以执行一次难以执行成功。出现段错误，无法获得shell：

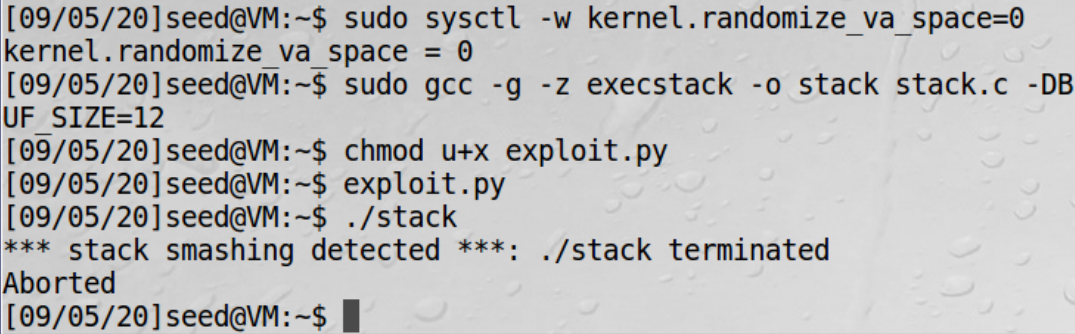


使用暴力搜索，程序在运行了2分28秒，执行81039次时成功匹配badfile中所给的地址，进入了特权模式：



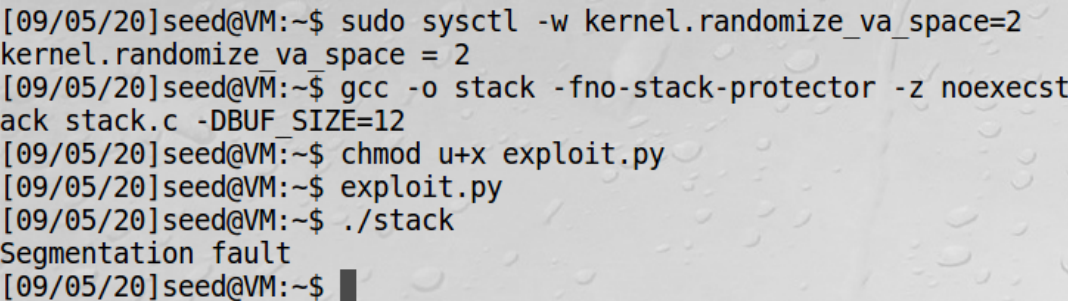
**Task5:Turn on the** **StackGuard Protection**

首先将地址随机化关闭，确保实验出现的结果仅仅受打开StackGuard Protection的影响。然后在有StackGuard Protection的情况下重新编译stack.c并运行，此时Task2的攻击不成功：



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

首先关闭地址随机化，使用noexecstack重新编译Task2中的stack.c并执行。此时不能得到shell：



**实验感想：**

此次实验主要学习了栈溢出的攻击和防御。在实验中，我进一步了解了内存中栈的结构，初步学习了对于栈溢出的攻击方法，其中重难点在于确定函数的返回地址，并将恶意代码放入栈中的正确位置。同时学习了对于栈溢出攻击的四种防御方法。这些知识让我觉得受益很深。