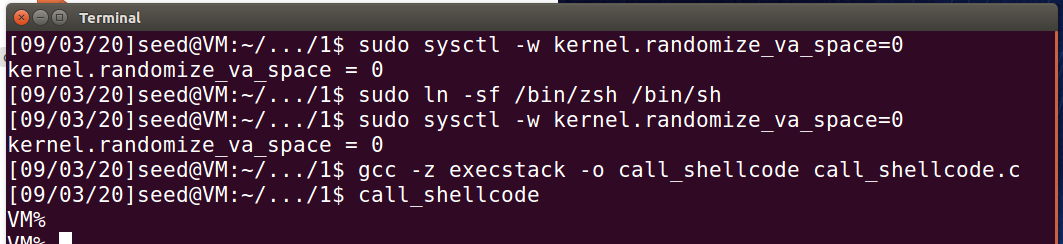
**lab2实验报告**

57117133 臧宇豪

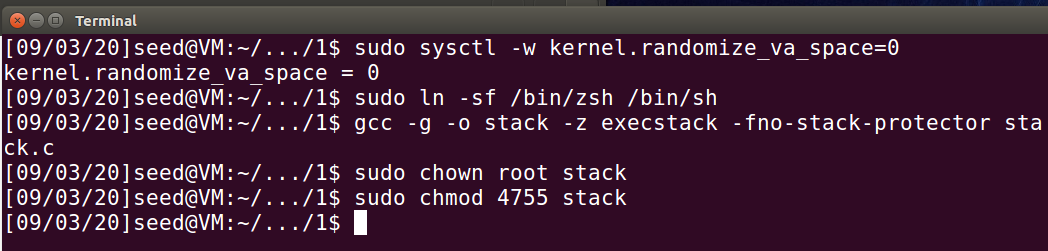
**一、Buffer Overflow Vulnerability Lab**

**Task 1: Running Shellcode**

运行call\_shellcode 程序，发现成功调出shell：

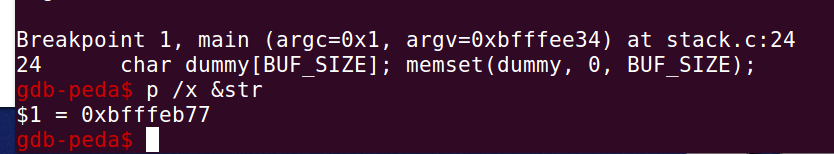


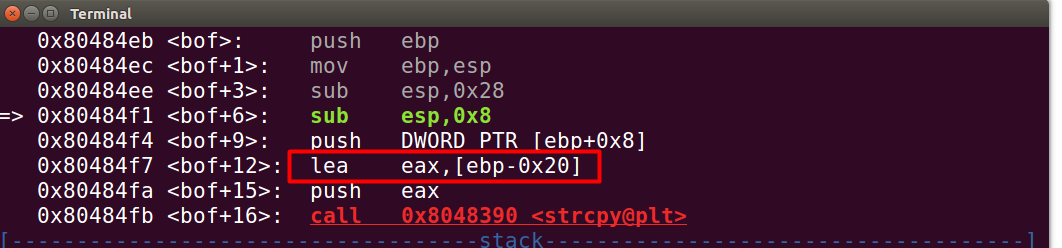
编译stack.c函数：



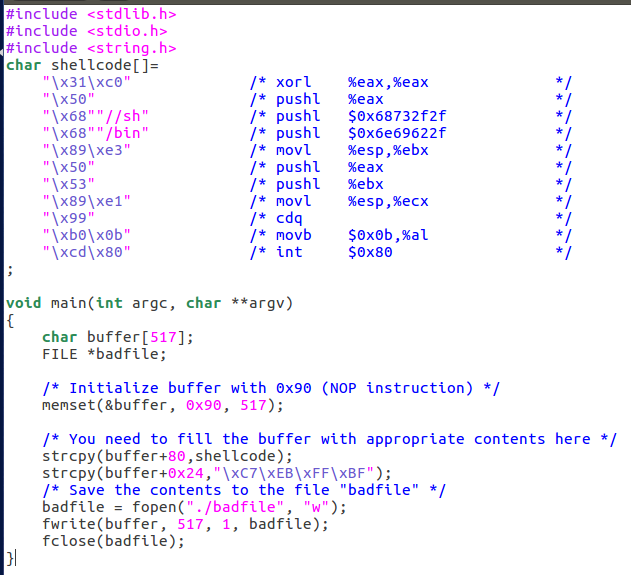
**Task 2: Exploiting the Vulnerability**

查看stack中str地址：

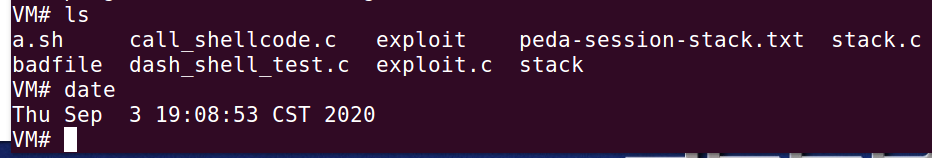




完善exploit.c 的代码：

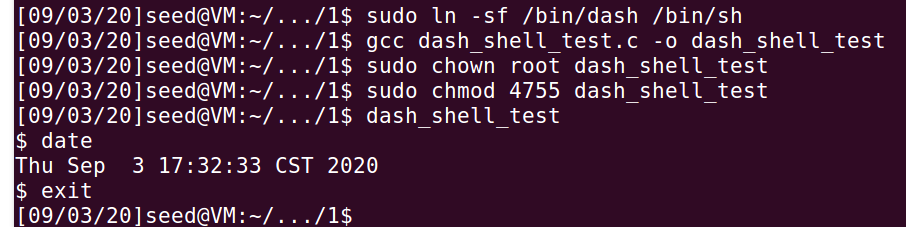


实验成功：

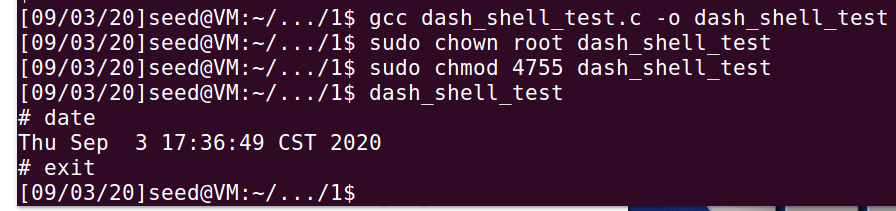


**Task 3: Defeating dash’s Countermeasure**

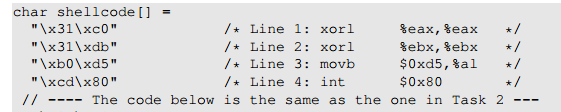
对于dash\_shell\_test.c，当注释setuid(0)时，获取的是用户权限的shell：



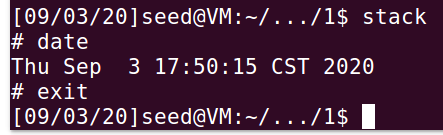
当对setuid(0)取消注释时，获取的是root权限的shell：



对shellcode进行添加，使得seuid(0)成为汇编语言：

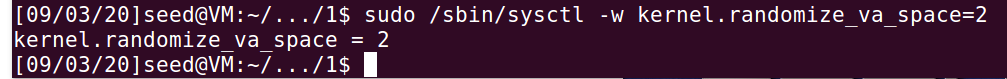


重新生成badfile后再次运行stack文件：

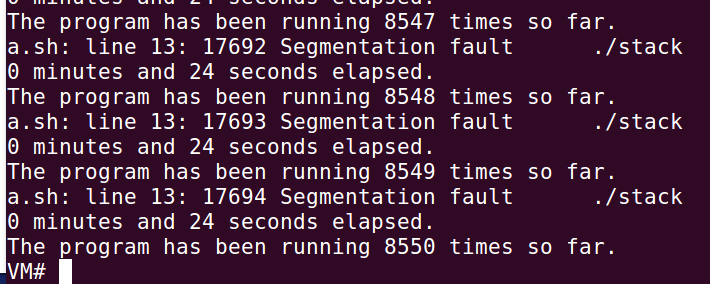


**Task 4: Defeating Address Randomization**

首先打开地址随机：



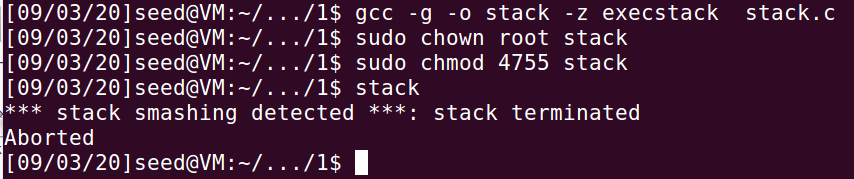
然后运行脚本，循环匹配：



经过24秒，8550个循环后，成功通过了地址随机的安全策略。

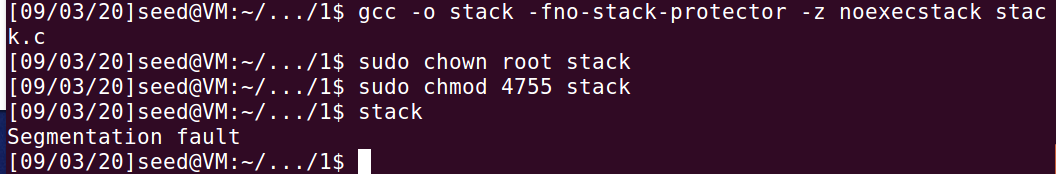
**Task 5: Turn on the StackGuard Protection**

按照要求操作后，程序无法运行：



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

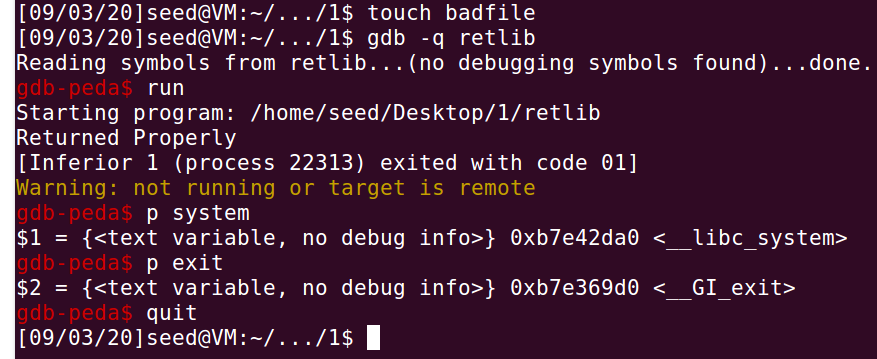
按照要求操作后，程序运行失败：



**二、Return-to-libc Attack Lab**

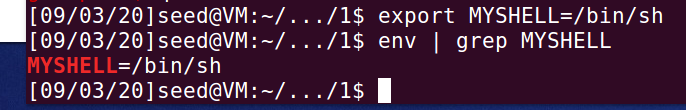
**Task 1: Finding out the addresses of libc functions**

按照操作步骤，完成如下，system()地址为0xb7e42da0；exit()地址为0xb7e369d0。

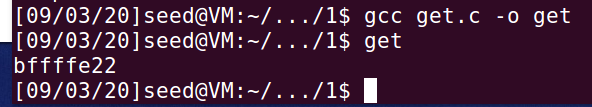


**Task 2: Putting the shell string in the memory**

设置环境变量：

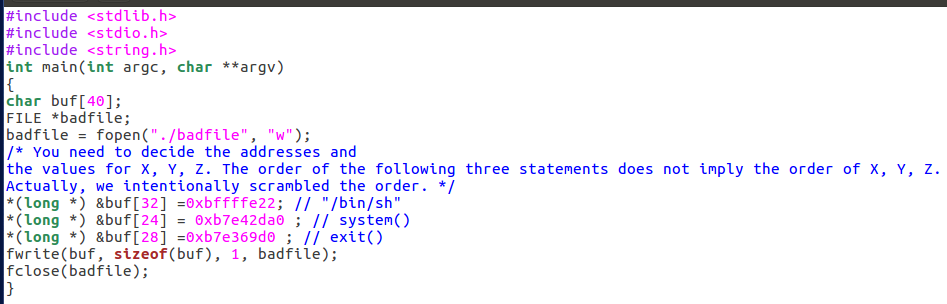


获取环境变量地址：bffffe22



**Task 3: Exploiting the buffer-overflow vulnerability**

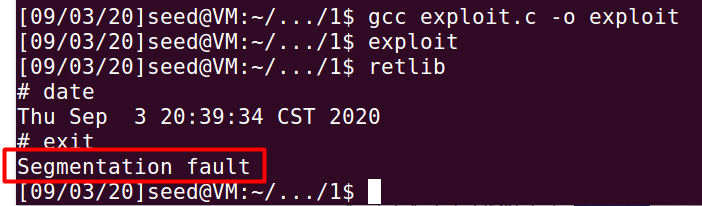
构造exploit.c代码：



对环境变量地址稍作修改，变成0x bffffe1c后，retlib程序执行成功：

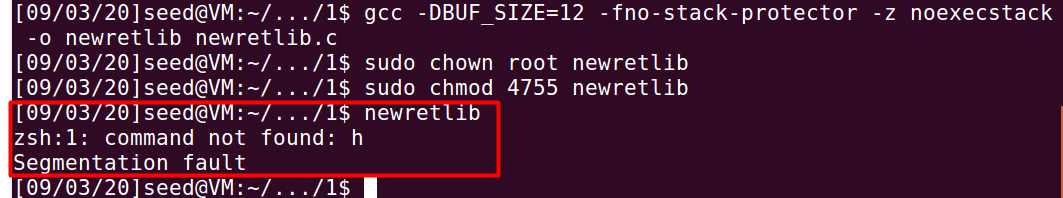


**变种问题一：**当注释掉exploit.c中exit()函数的地址时，能够成功提权，但是退出特权模式时出现问题：

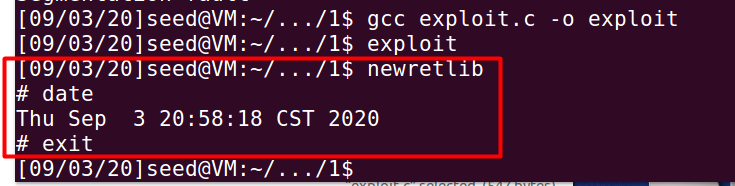


原因大致是当没有exit()，原来填充exit()地址的数据可能是任何值。而在栈中是先执行system()，之后调用原exit地址时就会崩溃。

**变种问题二：**程序改名newretlib.c之后，尝试运行失败：

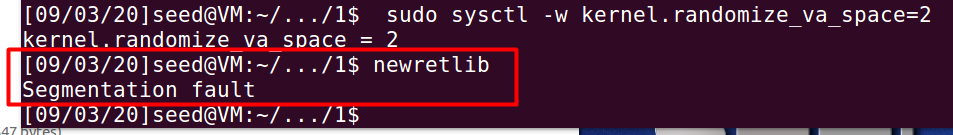


再将环境变量地址改为0xBFFF FE16，成功：



**Task 4: Turning on address randomization**

按照要求操作后，提权失败：



通过研究可以发现，三个参数的地址都发生了变化：

