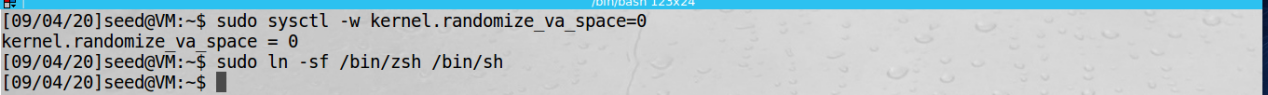
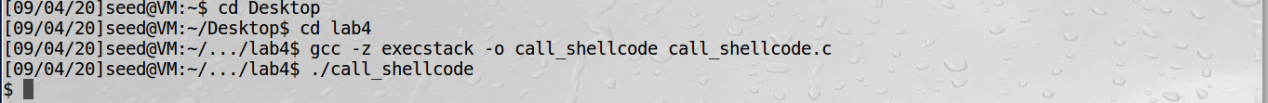
**Lab4**

**2.1+2.2 Task1**

关闭随机地址功能，并将/bin/sh链接到/zsh：

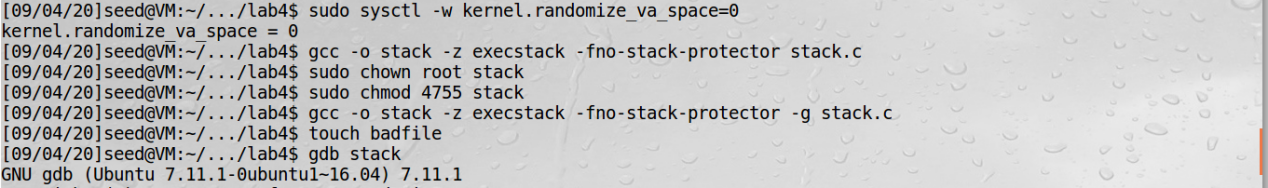


运行call\_shellcode.c文件，成功调用shell。



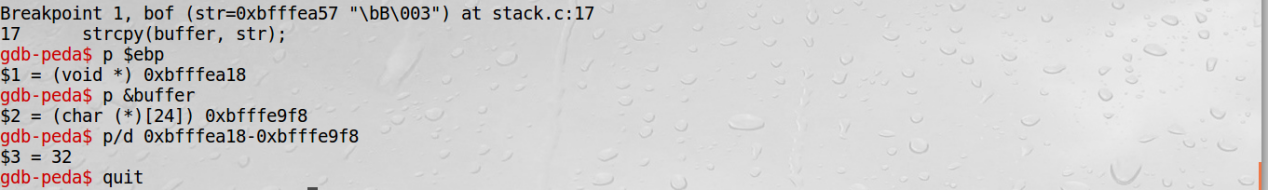
**2.3+2.4 Task2**

编译stack.c并使其成为set-uid程序：

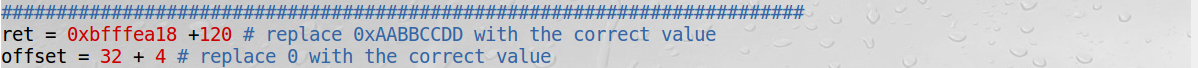


找出缓冲区的起始地址与返回地址之间的距离：

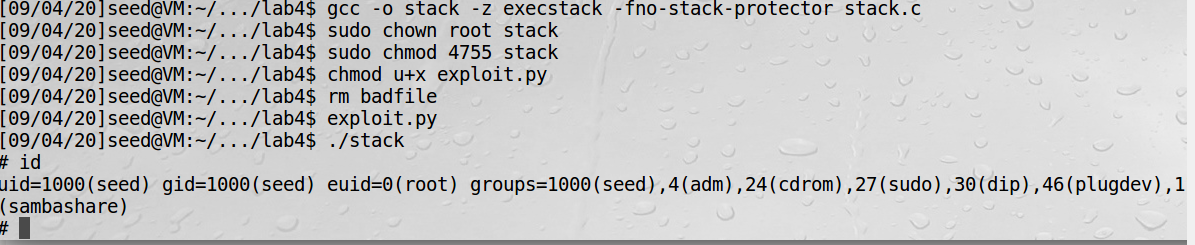




根据上述结果，完成exploit.py的编写：

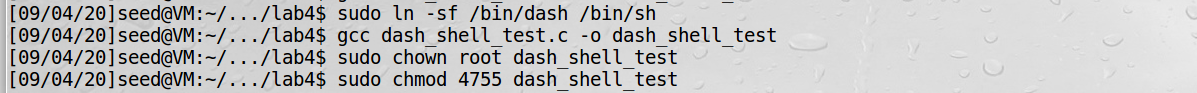


运行程序，攻击成功。



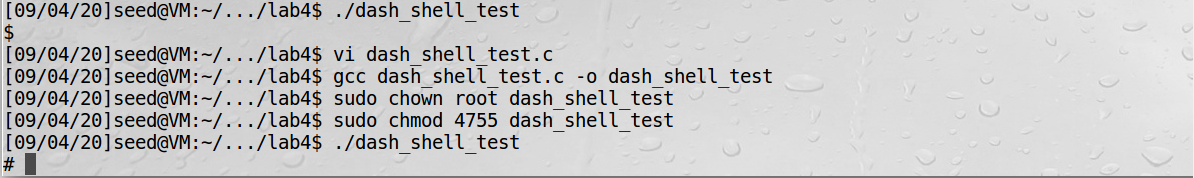
**2.5 Task3**

将链接改回/bin/dash并将dash\_shell\_test.c设置为set-uid程序：

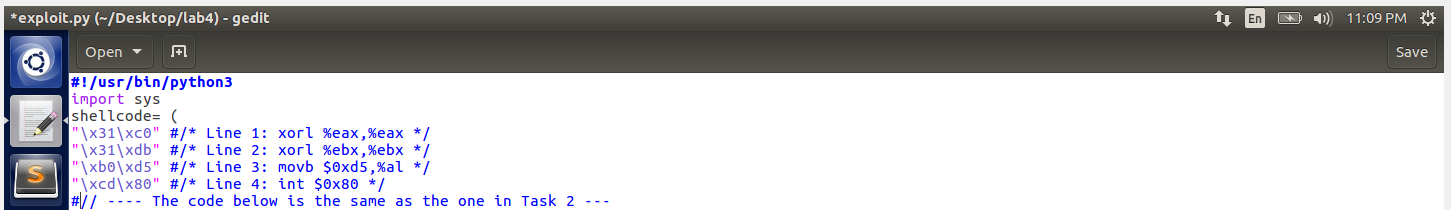


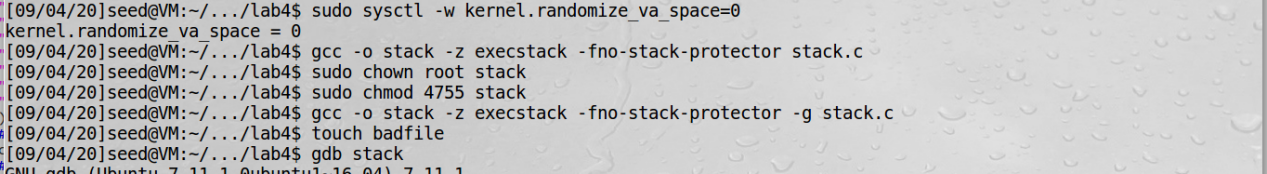
运行dash\_shell\_test.c，发现当系统并未链接到另一个shell程序（如zsh）时，dash shell的降权机制有效；加上setuid(0)后运行，成功绕过了保护机制，得到了root权限。

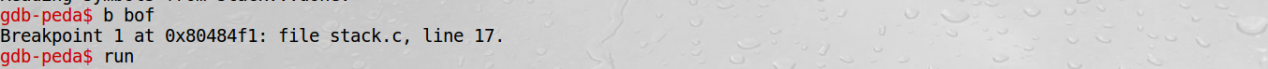
由于在调用dash之前，setuid(0)将进程的RUID改为了0，因此可以成功绕过对set-uid特权的检测，避免降权。

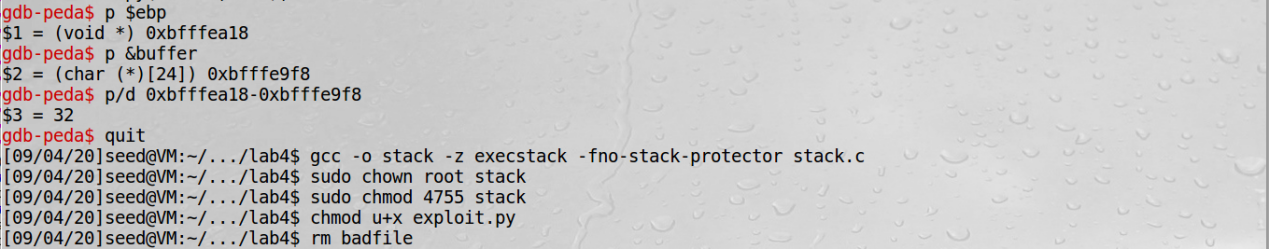


在攻击程序中增加代表setuid(0)的汇编语句，再次进行Task2中的攻击：







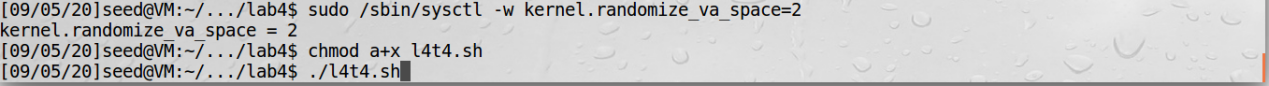


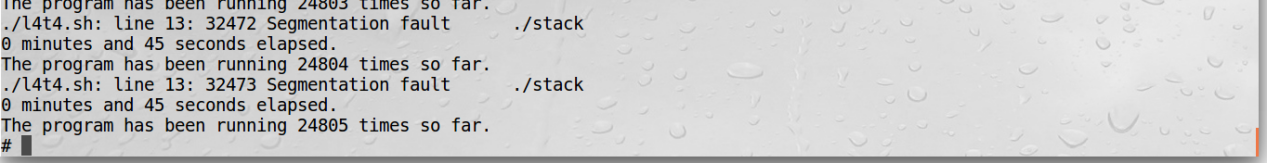
在没有链接到/zsh的情况下，攻击成功，证明了上述setuid(0)的有效性：



**2.6 Task4**

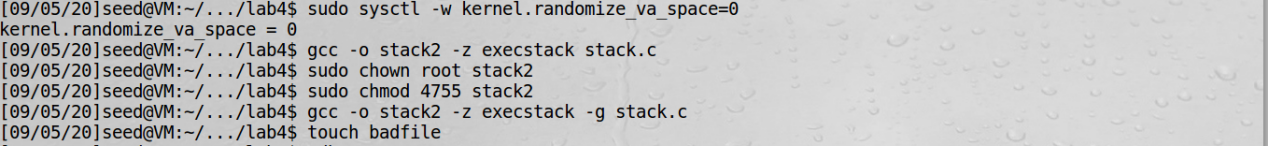
打开地址随机化，运行脚本，经过45s找到了地址并完成了攻击。

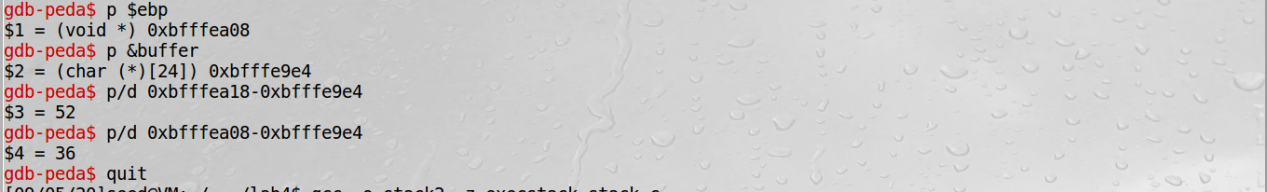




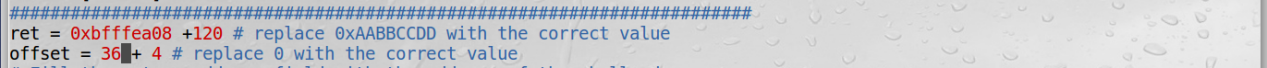
**2.7 Task5**

关闭地址随机化，并在使用StackGuard的情况下尝试攻击：





根据新的位置调整脚本：

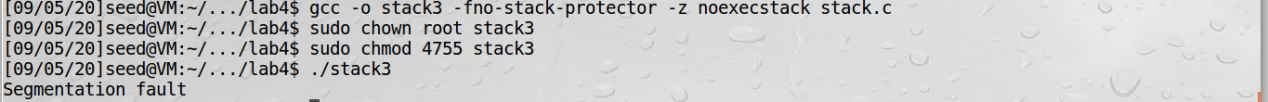


攻击失败：



**2.8 Task6**

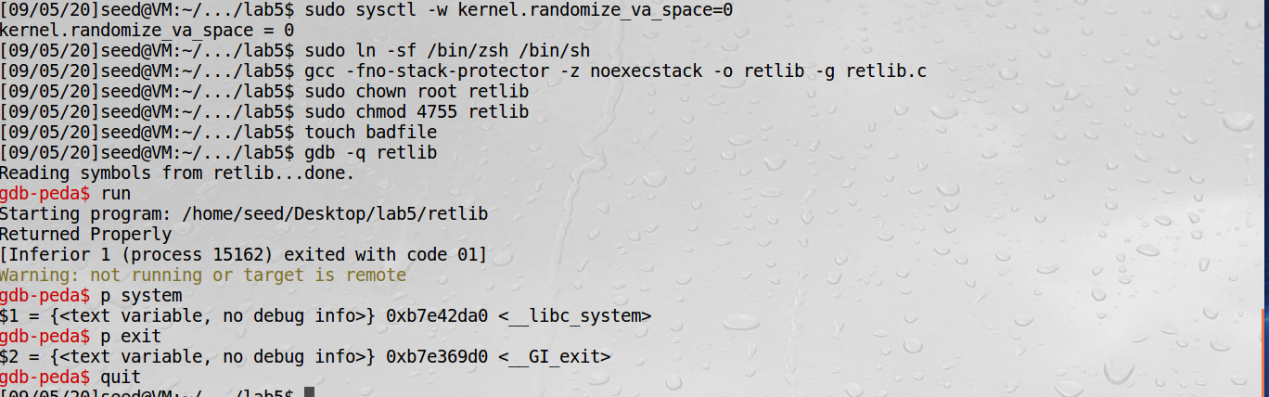
设置为堆栈不可执行，无法进入堆栈，故攻击无法实现。



**Lab5**

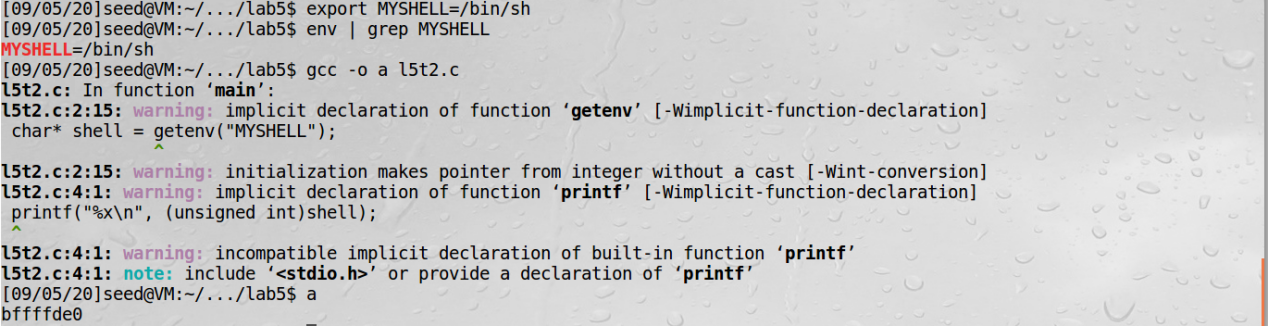
**2.1+2.2+2.3 Task1**

关闭随机地址功能，并将/bin/sh链接到/zsh，输出system()函数和exit()函数的地址：



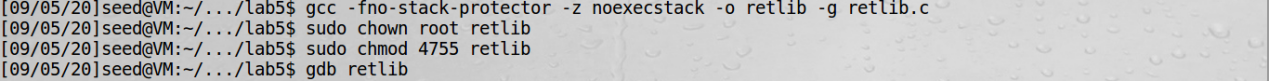
**2.4 Task2**

通过环境变量将“/bin/sh”置入内存，并找到该环境变量的地址：



**2.5 Task3**

找到ebp的位置（20）：





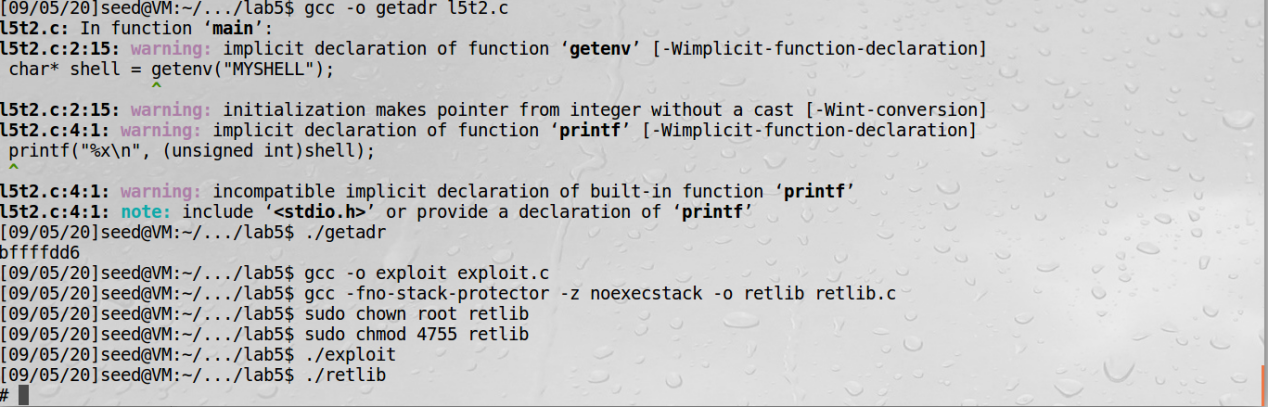


编写漏洞程序的代码：



其中，需要保持查找MYSHELL环境变量的程序名称与retlib长度一致，因为文件名的长度对环境变量的地址有影响。

攻击成功：



**Attack variation 1:**

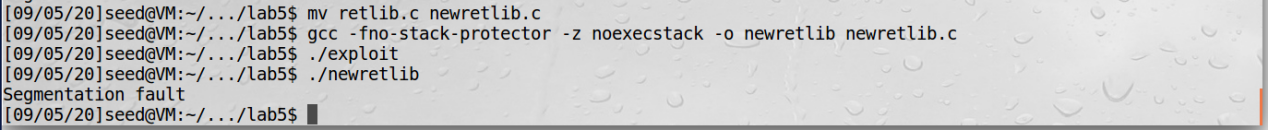
删除exit()函数的地址，攻击失败：





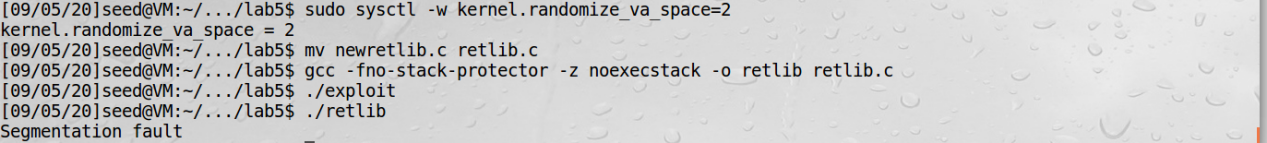
**Attack variation 2:**

更改retlib文件名，攻击失败：

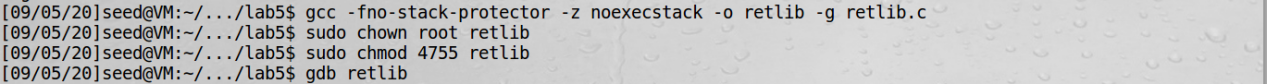


**2.6 Task4**

恢复retlib文件名，打开地址保护，发现攻击失败：

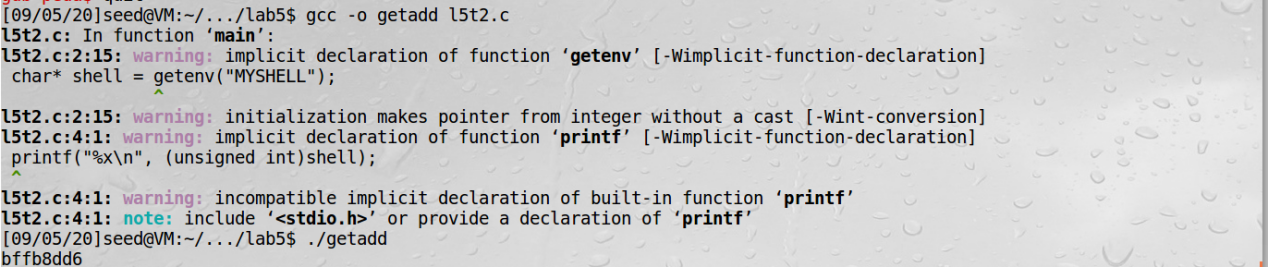


查找三个地址与X、Y、Z这六个值的变化：









由于ebp的位置未变，X、Y、Z都不改变，而一一对比可知/bin/sh、exit()、system()的地址都发生了改变。

**2.7 Task5**

将setuid(0)写入到system之前，可以在挂载到/bin/dash时完成攻击：



