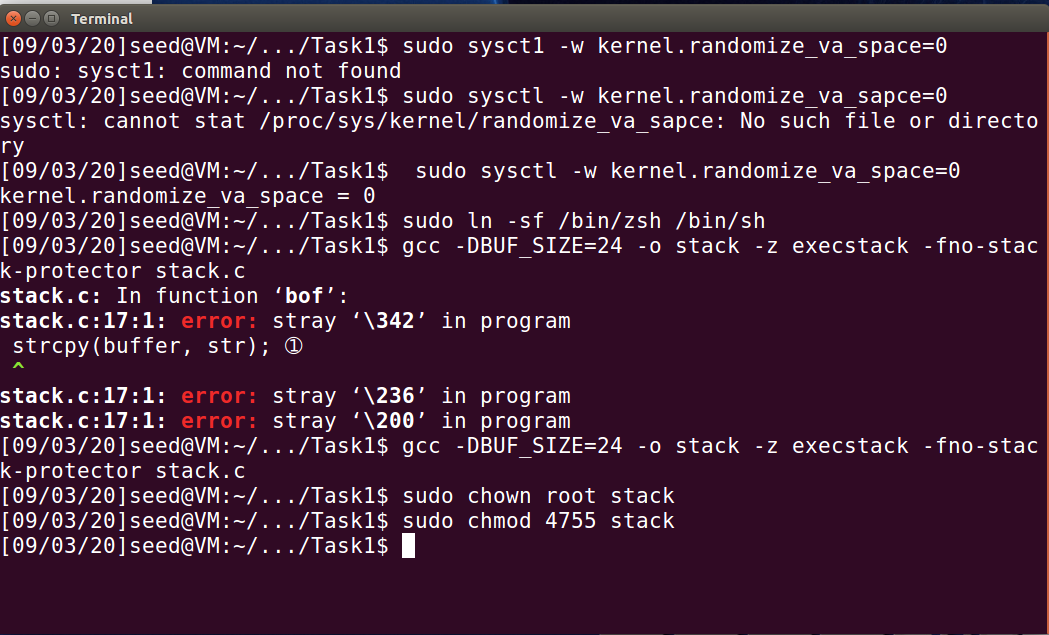
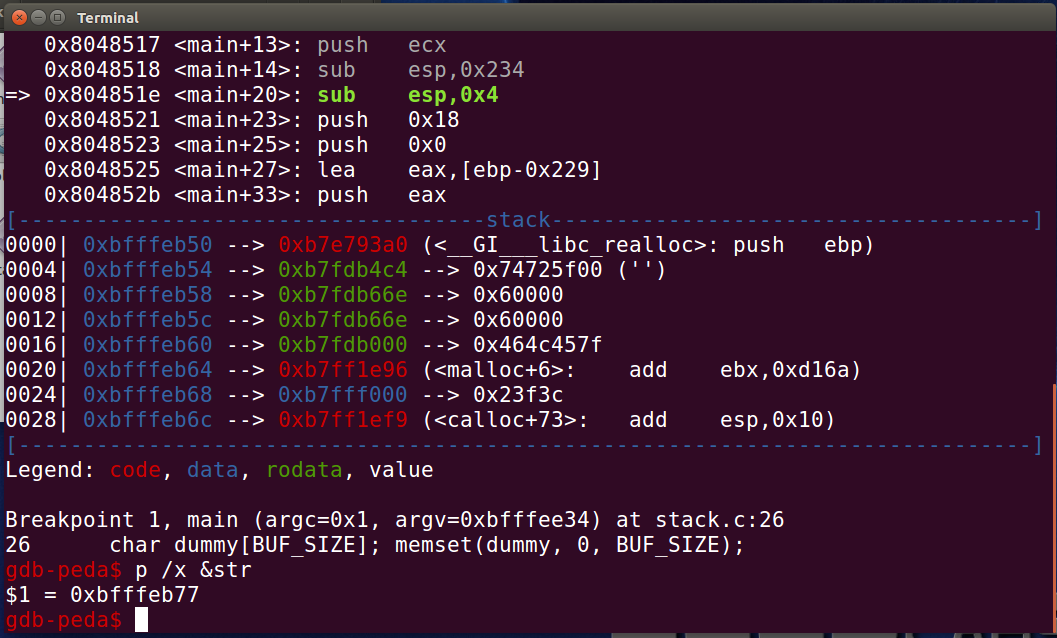
TASK1

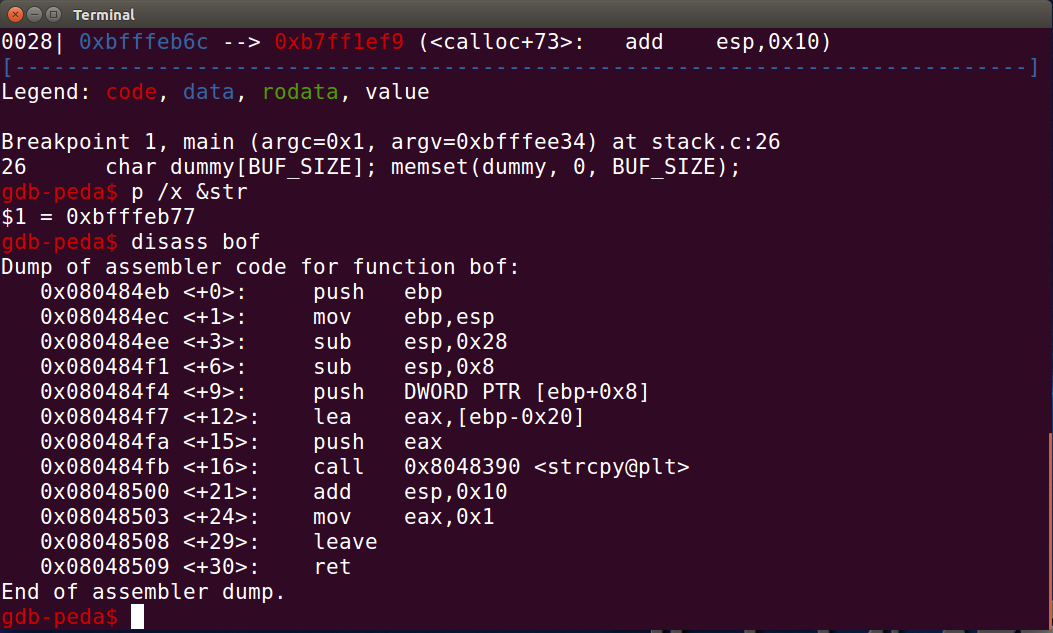
按照实验的要求关闭了系统的随机化地址空间功能，并且编译了攻击与被攻击文件

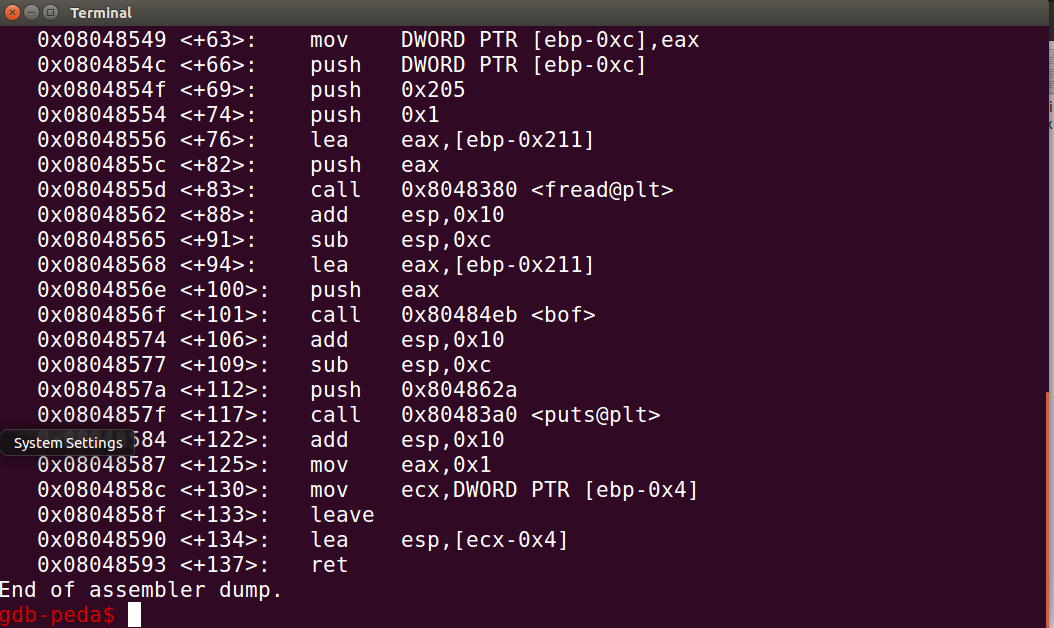


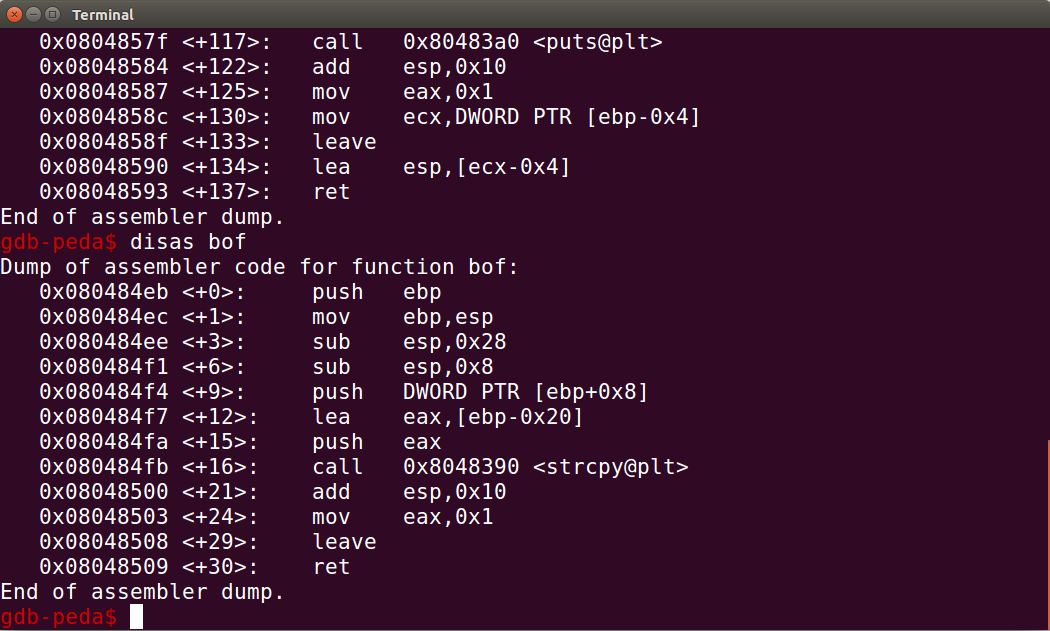


使用gdb查看str地址，计算出shellcode的地址0xBFFFEBDB



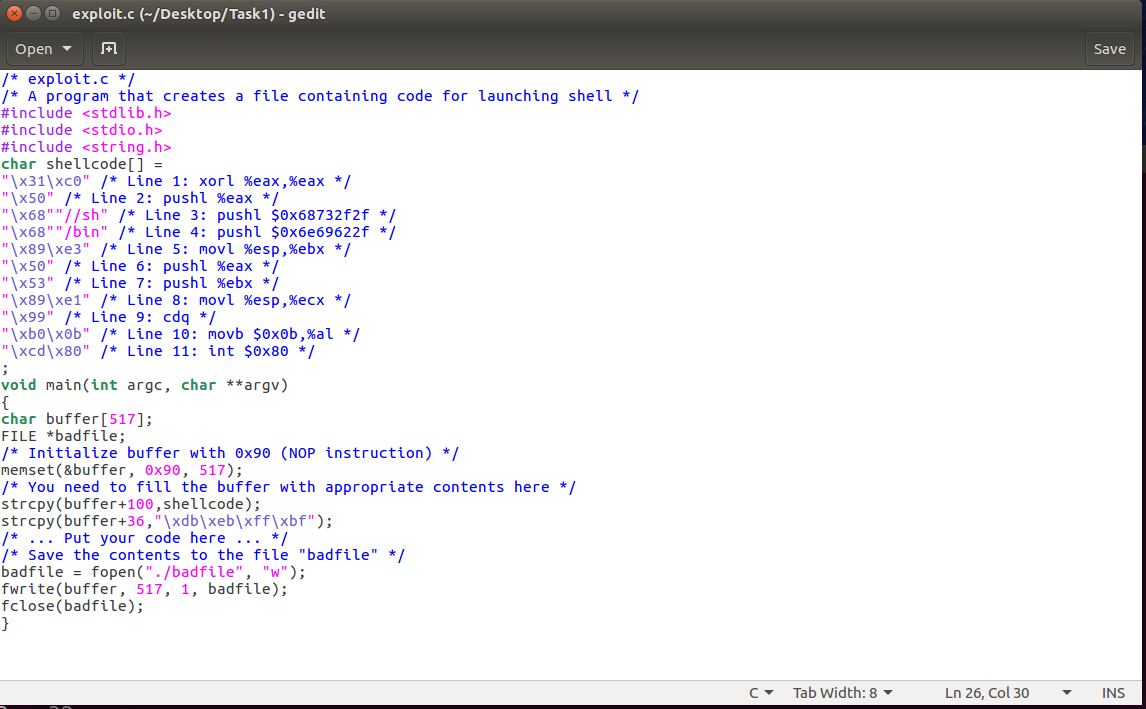




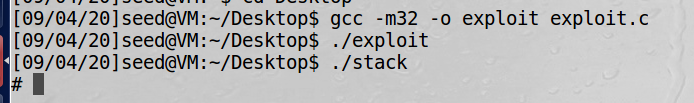




使用gdb计算出buffer与ebp的地址偏移量，填入exlpoit中，并执行

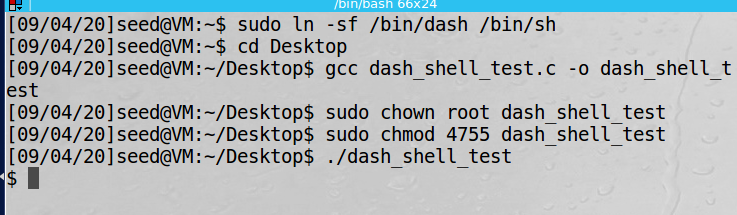


运行发现成功获取了root权限

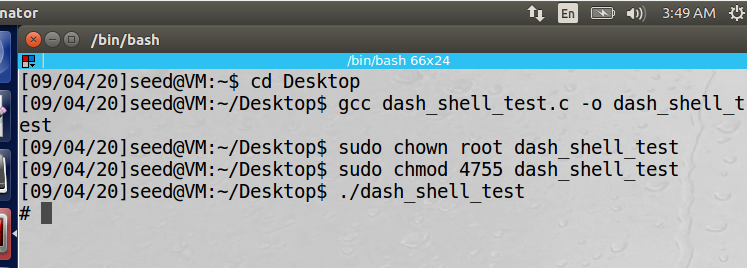


TASK2

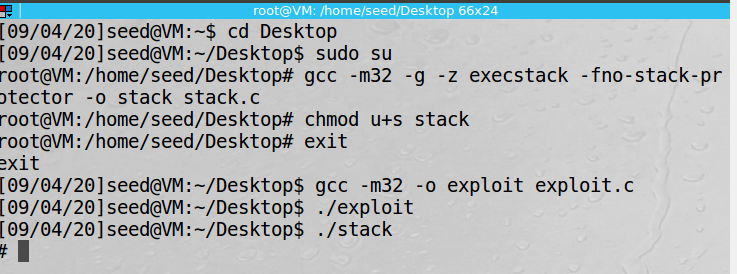
注释掉行①，运行程序。



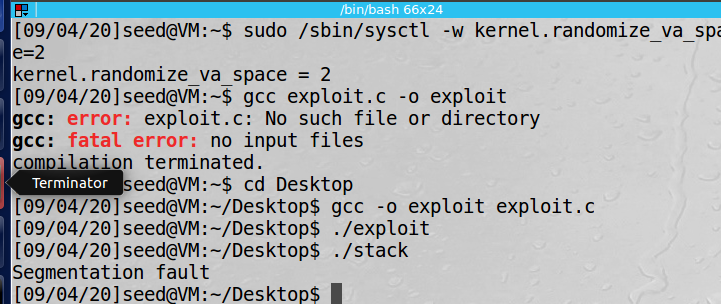
取消注释，运行程序。发现加上setuid(0)之后，可以获得root权限。



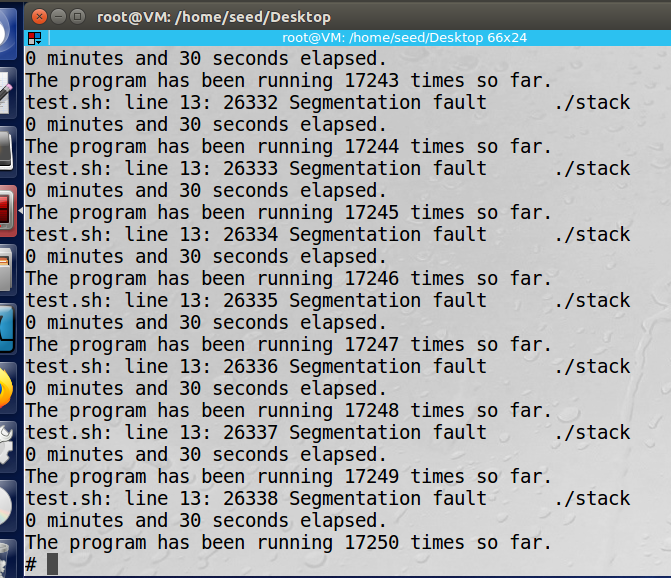
在exploit.c中添加四条语句后，再执行Task2的攻击。可以获得root shell。



TASK4

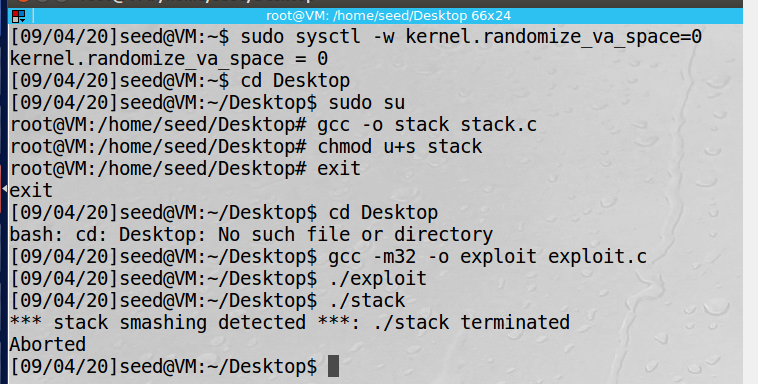


打开地址随机化后，发现无法通过攻击获得root权限了。因为猜测地址是缓冲区溢出攻击的关键步骤之一，缓冲区地址随机化使得猜测准确的地址变得困难，猜中的成功率很低，攻击难以成功。



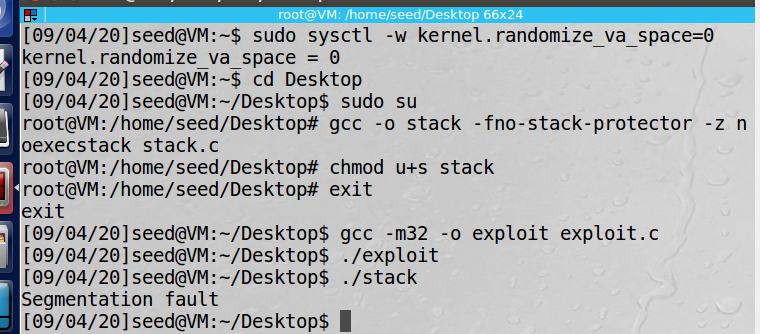
最终发现在重复攻击后最终获取到root权限

TASK5



关闭地址随机化，不禁用stackguard保护机制重新编译stack.c，再次进行Task2的攻击，发现去掉-fno-stack-protector后，在运行的时候报stack smashing detected

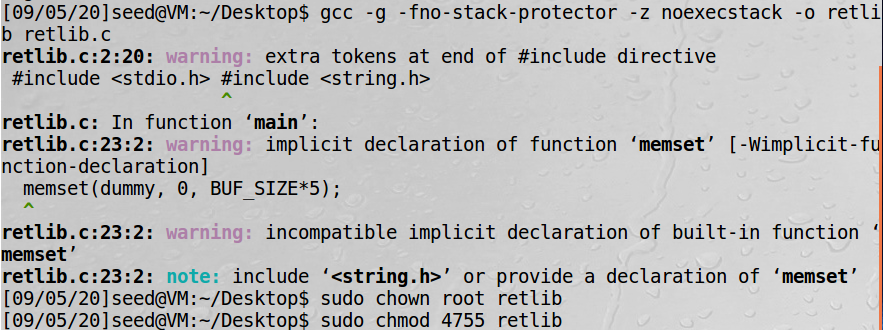
TASK6

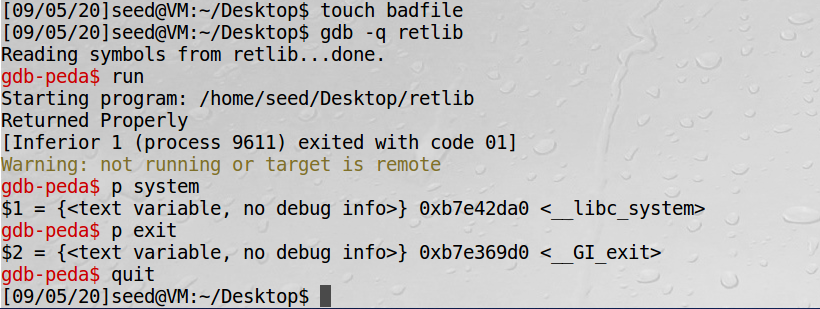


关闭地址随机化，使堆栈不可执行，重新编译stack.c，再次进行Task2的攻击，发现无法通过攻击获得root权限了。

Ret2lic

TASK1

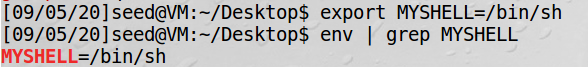




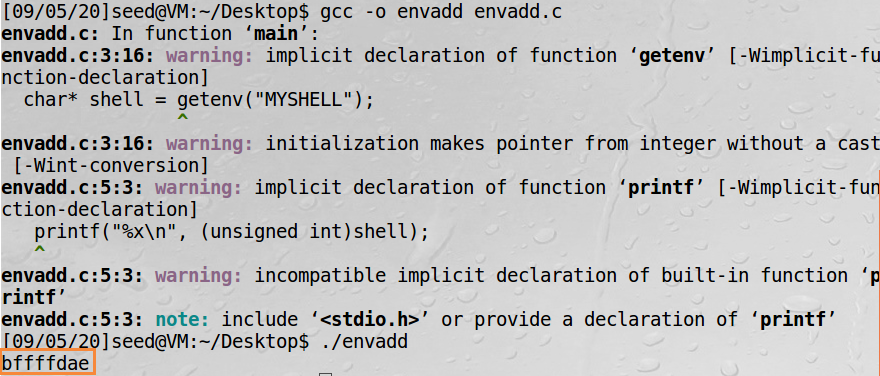
得到system()的地址为0xb7e42da0，exit()的地址为0xb7e369d0

TASK2

首先定义一个环境变量MYSHELL，值为/bin/sh



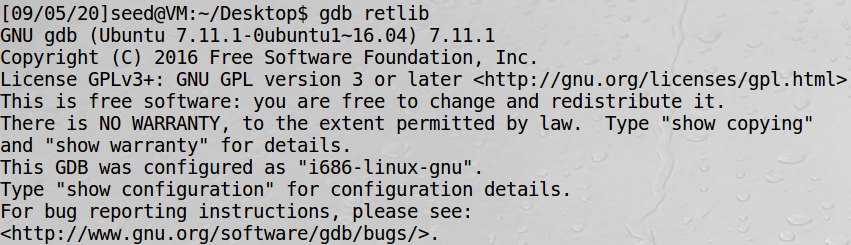
将环境变量的地址作为传递给system()系统调用的参数，要获取环境变量的地址

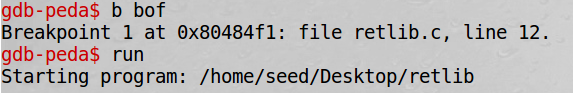


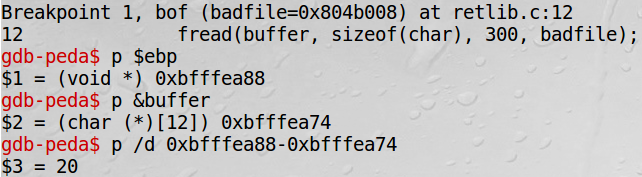
TASK3

我们需要覆盖bof函数的返回地址，将其指向system()函数，并且将/bin/sh作为system()的参数，最后执行exit()函数范围，故我们必须知道bof函数的返回地址

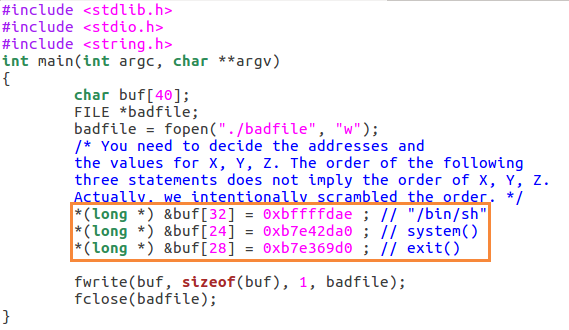
操作同上一个实验的task 2



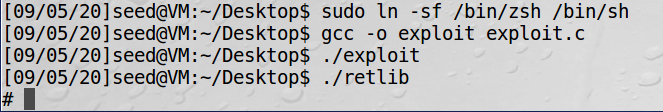




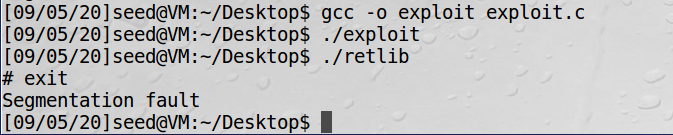
buffer到return address的距离为20+4=24，所以exploit.c中system()在buffer数组中的下标为24。可将exit()的地址作为system()的返回地址，所以exit()在buffer数组中的下标为28，参数/bin/sh在buffer数组中的下标为32。对应的地址前面两个task已经求过了。



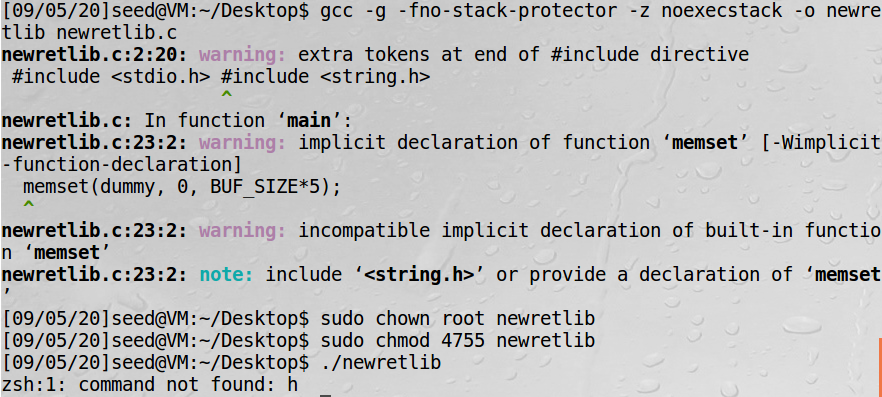
编译并执行代码，成功得到root shell



exit()函数是必要的吗？是必要的，否则system()的返回地址就可能是任意值，当system()返回时会导致程序崩溃



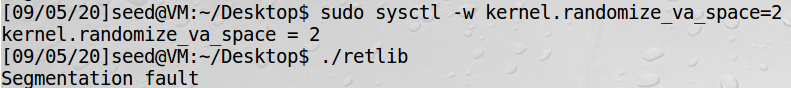
将retlib的名字改为newretlib，重新实验，实验失败了



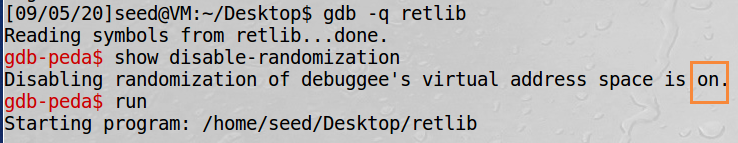
原因是环境变量的地址可能有所不同（比如地址会因为程序名字的改变而发生改变，主要是字符数）。程序名不同，传入栈的字符长度可能不同，从而导致环境变量的地址可能不同。

TASK4

打开地址随机化，再次尝试之前的攻击，发现攻击失败



原因是地址随机化打开后，之前的三个地址就会发生变化



两次运行retlib，两次的system和exit的地址不同

