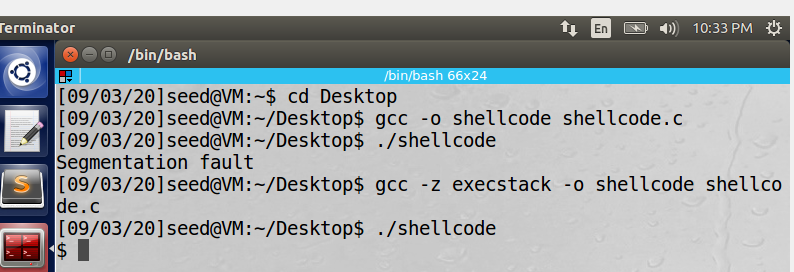
Buffer Overflow Vulnerability Lab

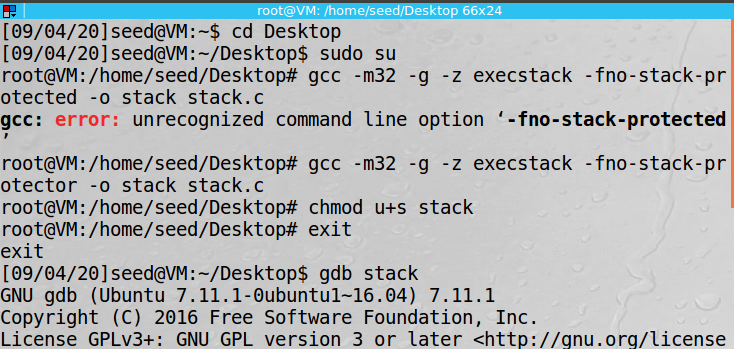
57118138 李嘉怡

**Task 1: Running Shellcode**

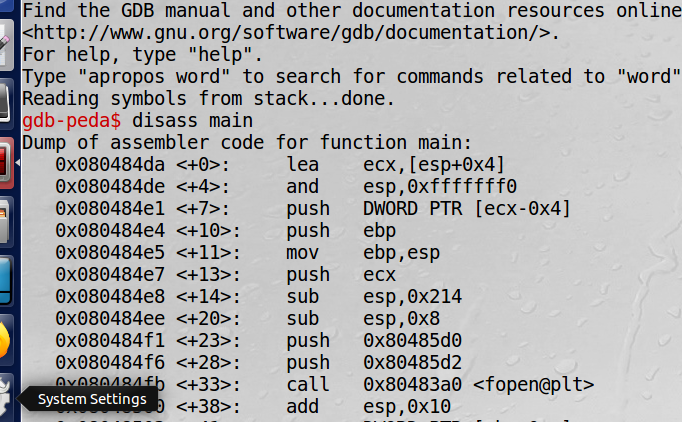


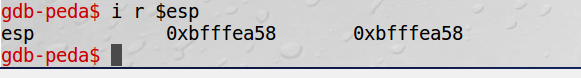
从以上两种调用的对比可知，当没有执行execstack语句时，执行shellcode无法调用shell；当执行execstack允许使用堆栈以后，显示$，说明成功调用shell。

**Task 2: Exploiting the Vulnerability**

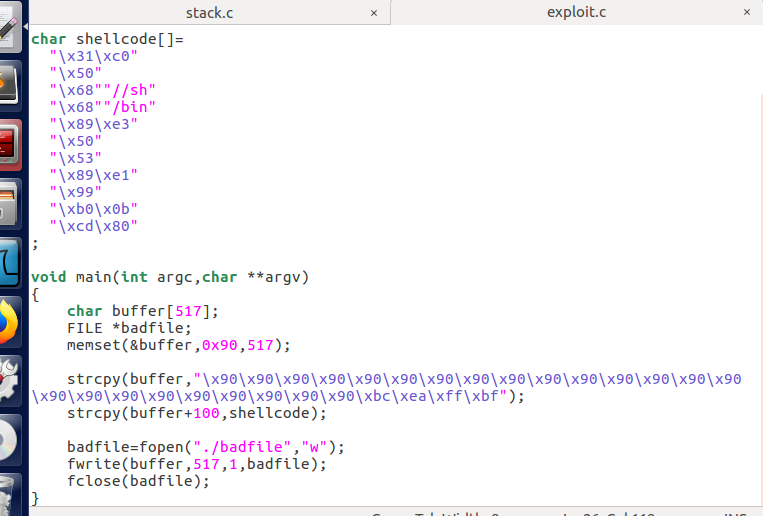


首先，编译易受攻击的程序stack.c并变成setuid程序，在编译代码时需要用 –fno-stack-protector 关闭这种机制， -z execstack 用于允许执行栈。

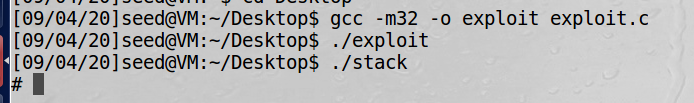




然后完善攻击程序exploit.c。使用gdb反编译找到exploit.c中shellcode数组的地址，得到十六进制0xbfffea58加十进制100后得到的16进制0xbfffeabc，填入exploit.c中。

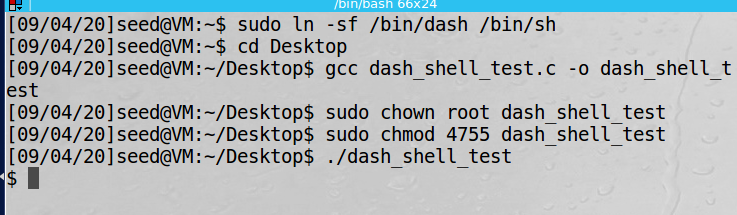


编译exploit.c并运行，再运行stack，出现了#，可见，通过攻击，获得了root权限。

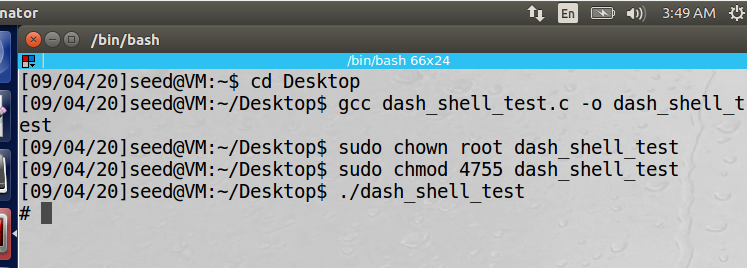


**Task 3: Defeating dash’s Countermeasure**

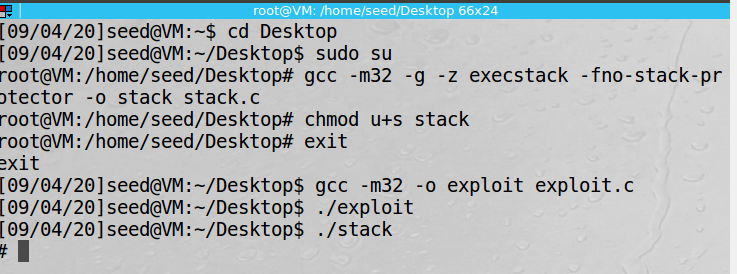
注释掉行①，运行程序。



取消注释，运行程序。发现加上setuid(0)之后，可以获得root权限。

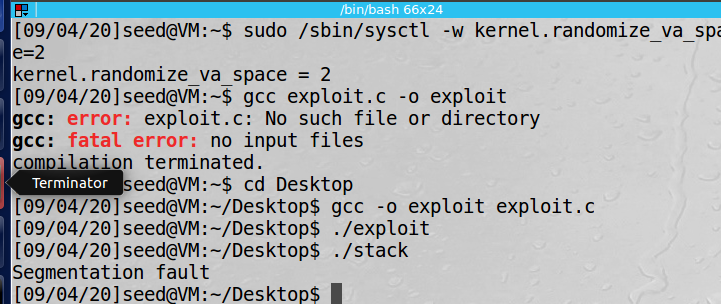


在exploit.c中添加四条语句后，再执行Task2的攻击。可以获得root shell。

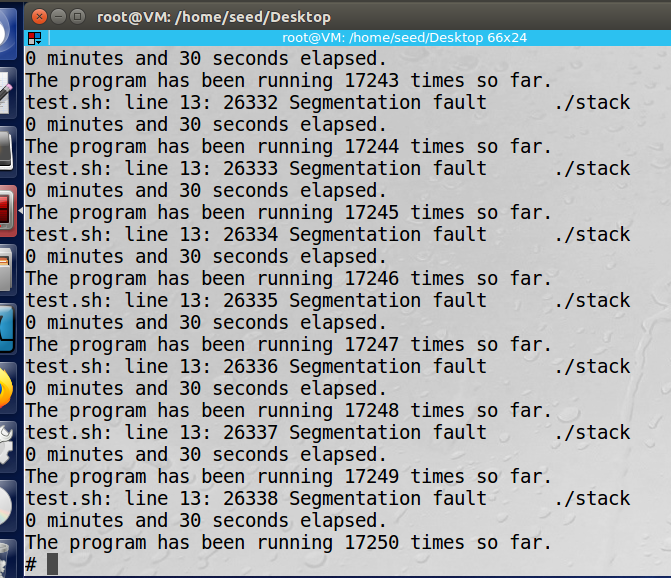


加上四条汇编语句相当于调用了setuid(0)。因为bsh和dsh可以把EUID变成RUID，自动降权成普通用户。如果在shell代码中执行execve()之前调用setuid(0)，可以将受害进程的RUID由seed更改为零，由于是setuid程序，EUID本来就是0，这样RUID=EUID，bsh和dsh就会认为这不是一个setuid程序，不会删除特权了。

**Task 4: Defeating Address Randomization**

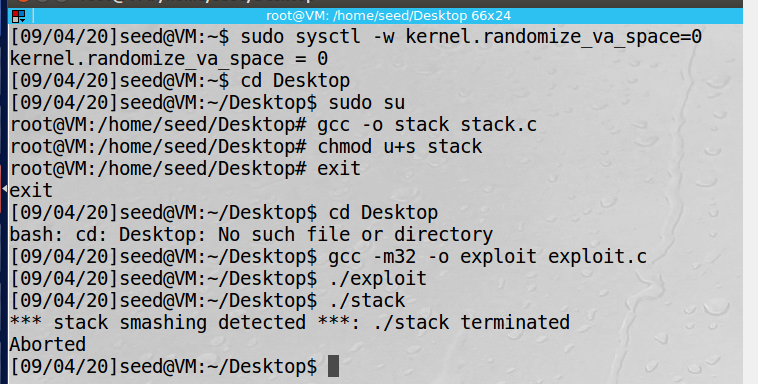


打开地址随机化后，发现无法通过攻击获得root权限了。因为猜测地址是缓冲区溢出攻击的关键步骤之一，缓冲区地址随机化使得猜测准确的地址变得困难，猜中的成功率很低，攻击难以成功。



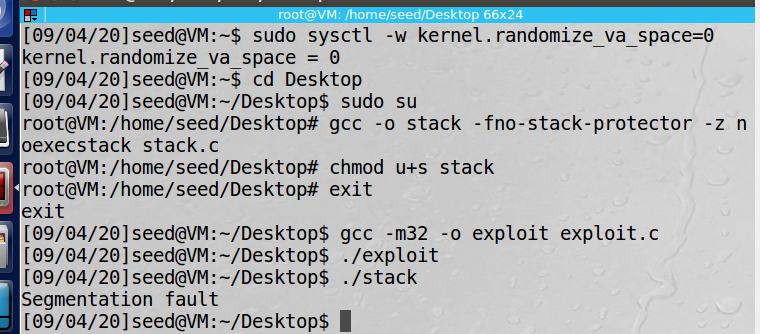
用brute-force的方法运行脚本30s，共17250次以后，获得了对shell的访问(执行了恶意代码)。

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



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错误问题，原因是默认开启了GCC的检测“缓存溢出”的保护机制。这种机制能够在栈中加一个值来观测，如果栈溢出了会被发现。通过这种保护机制可以终止缓冲区溢出的程序，达到预防攻击的目的。

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



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。因为shellcode一般都是在缓冲区里,缓冲区是堆栈形成的，只要操作系统限制堆栈中的数据只可读写，不可执行，一旦堆栈中的数据被执行立即报告错误，并退出。那么溢出成功后也不能执行shellcode。