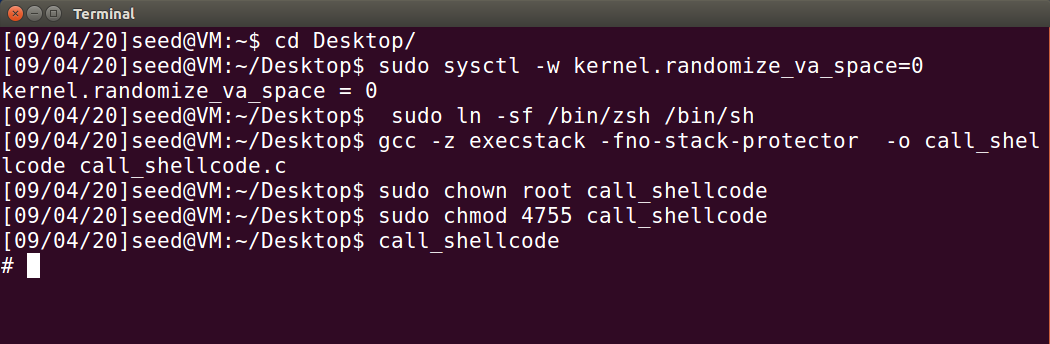
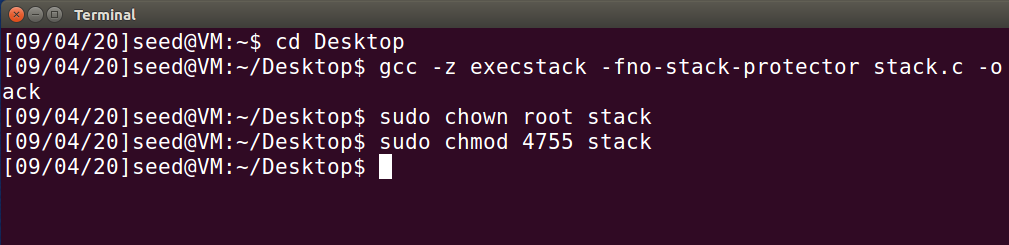
**Task1: Running Shellcode**

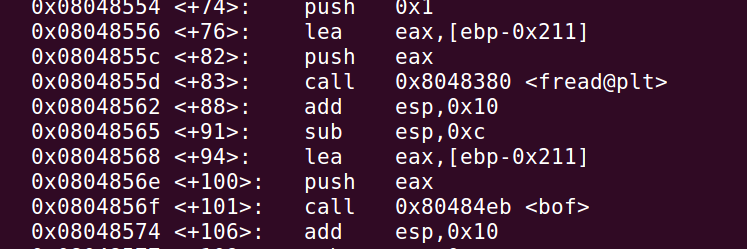
按照要求关闭各种防御措施，编译call\_shellcode.c，将它设定为root所有的set uid程序，并运行，发现成功进入了shell



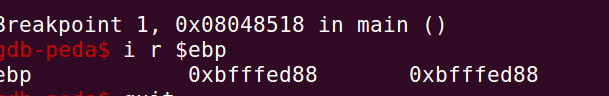
**Task2: Exploiting the Vulnerability**



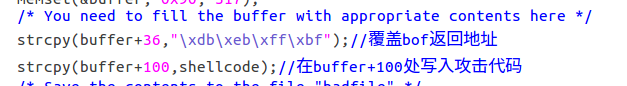
使用disass main发现str在内存中的位置为[ebp-0x211]



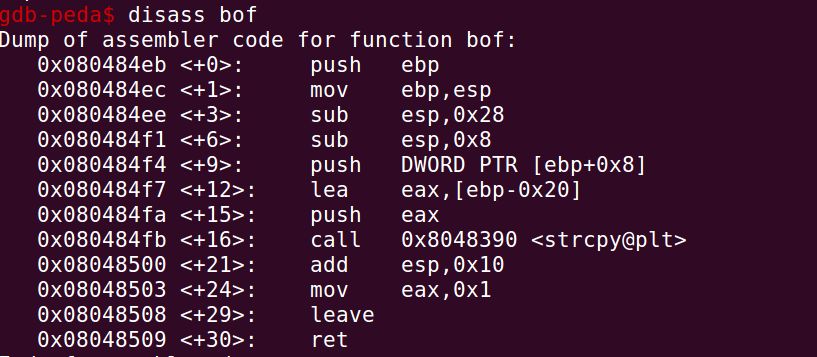
在main函数中设置断点，找出ebp的值为0xbfffed88



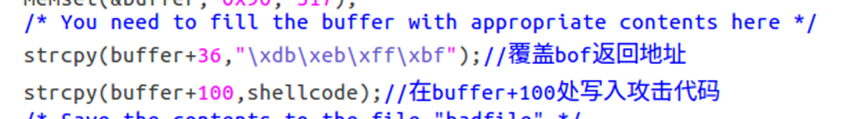
要在buffer+100出写入shellcode，因此计算绝对地址为0xbfffed88-0x211+0x64=0xbfffebdb，填入strcpy函数中。



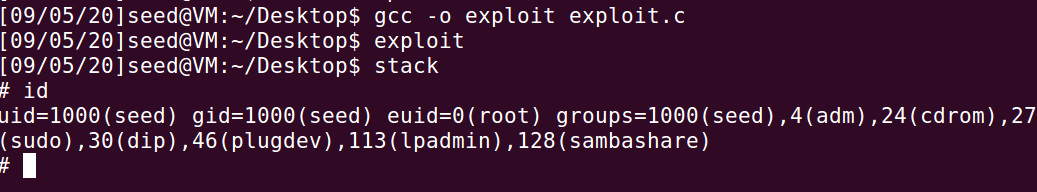
反汇编bof函数，计算出return地址buffer地址+0x20+0x04=buffer+36



填入strcpy函数中

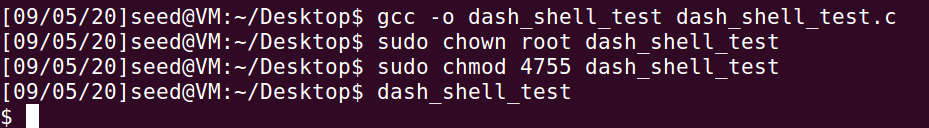


编译并执行exploit.c，生成badfile，在执行stack，成功进入shell

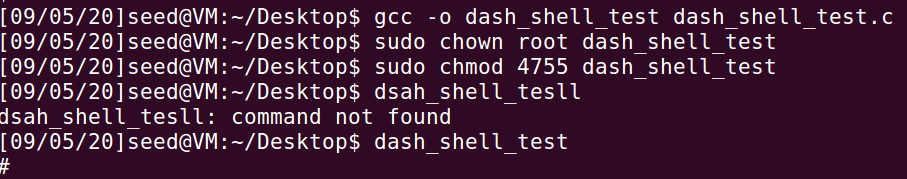


**Task3: Defeating dash’s Countermeasure**

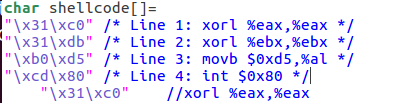
注释掉setuid（0），没有成功的获得root权限



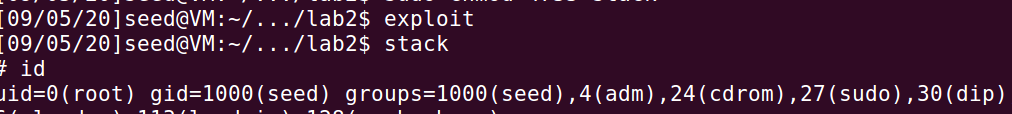
取消注释，再次编译执行，获得root权限



重新编译exploit.c

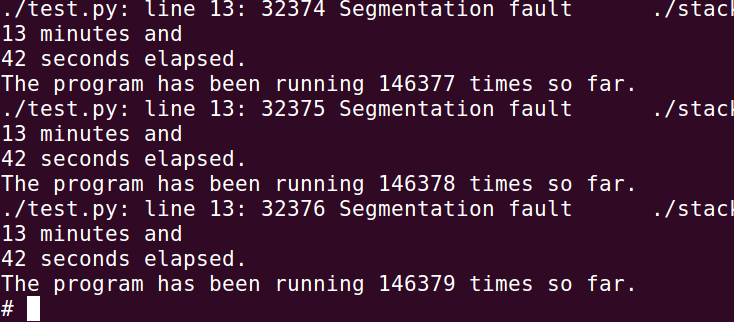


再次进行攻击，成功获得root权限，因为line1-4实现了setuid（0）的作用

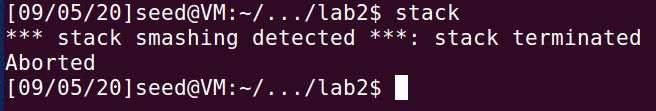


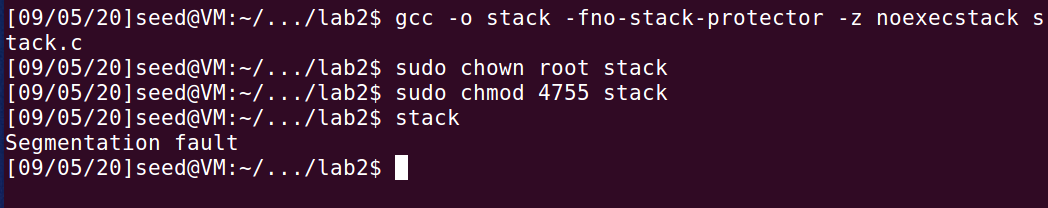
**Task4: Defeating Address Randomization**

使用sudo /sbin/sysctl -w kernel.randomize\_va\_space=2命令打开地址随机化，在执行所给的脚本，等待了13min42s后，获得了root权限。

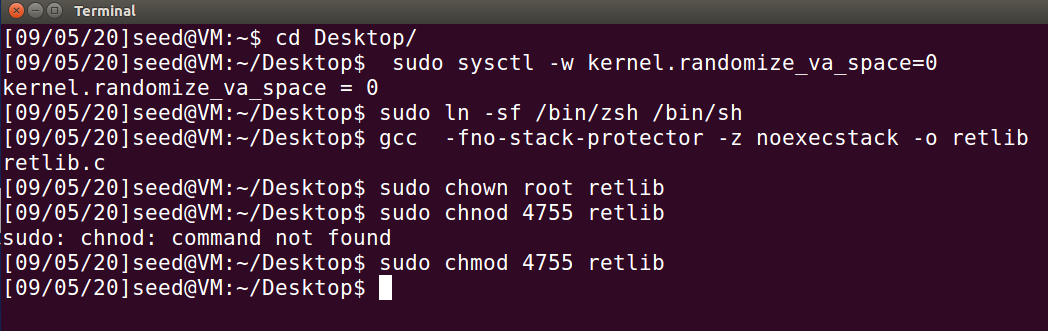


**Task5: Turn on the Stack Guard Protection**

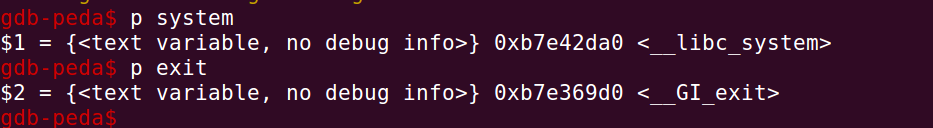


**Task6: Turn on the Non-executable Stack Protection** 

准备工作，按要求设置环境并编译retlib.c

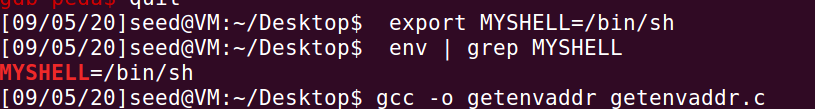


**Task1: Finding out the addresses of libc functions**

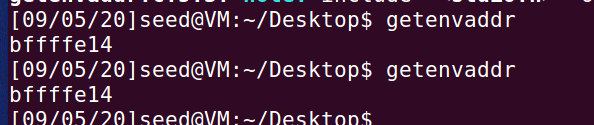


**Task2: Putting the shell string in the memory**

设置环境变量，编译getenvaddr.c

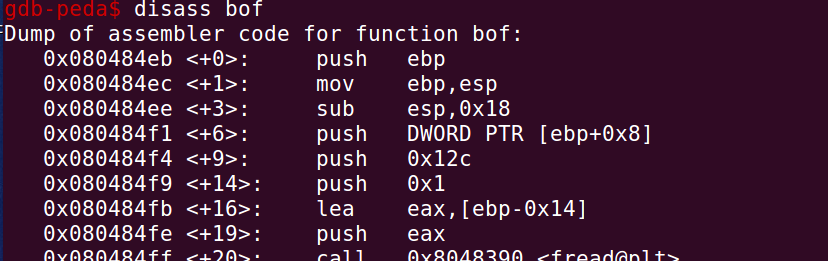


运行并得到“/bin/sh”的地址



**Task3: Exploiting the buffer-overﬂow vulnerability**

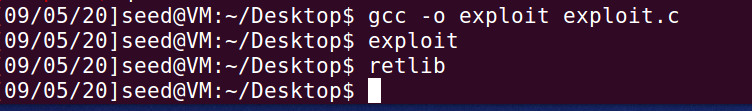
使用gdb,反汇编bof函数



发现buffer到ebp为0x14，而return地址为ebp+4,所以Y=0x14+4=24，那么Z=24+4=28,X=28+4=32,再将前两步中所得到的地址填入对应位置



编译exploit.c并运行，再运行retlib，发现没有出现#就结束了，不知道原因

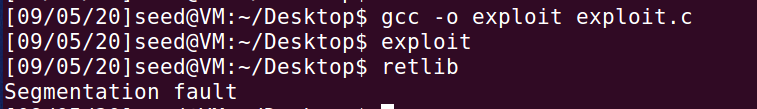


**Attack variation 1**

注释掉exit()地址

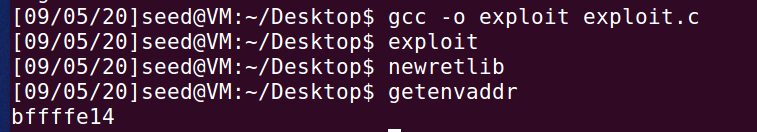


重新编译执行，攻击失败



**Attack variation2**

将exit()的注释取消，发现好像是成功了，环境变量的地址没改变（有问题）



**Task4: Turning on address randomization**

XYZ的值没有变化，所对应的3个address的值都发生改变。

