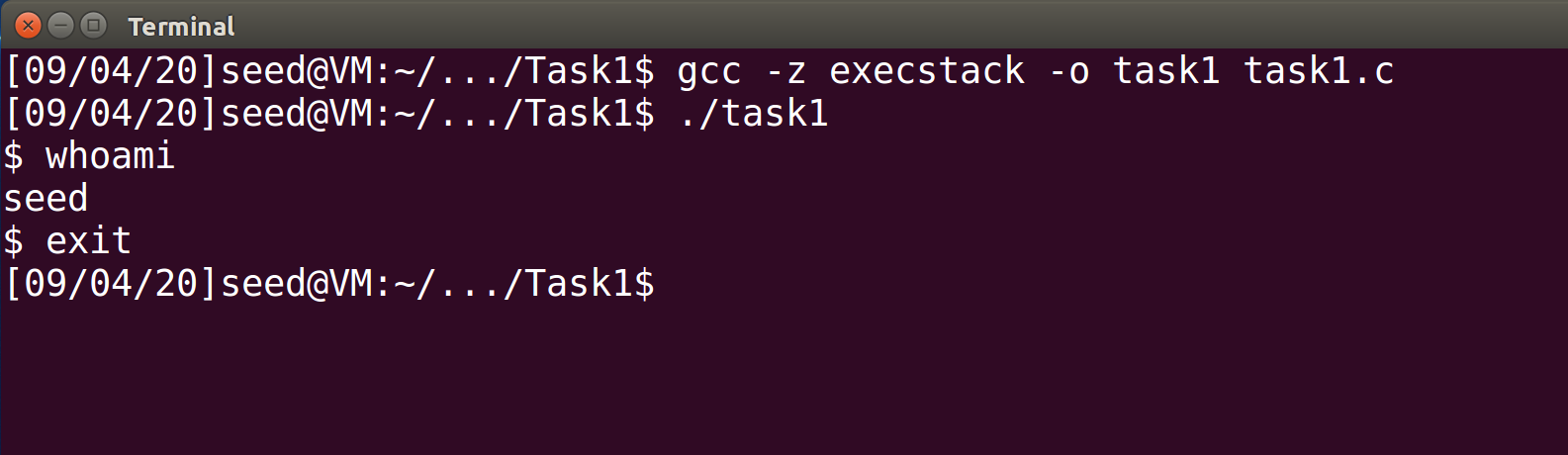
Lab2 Report

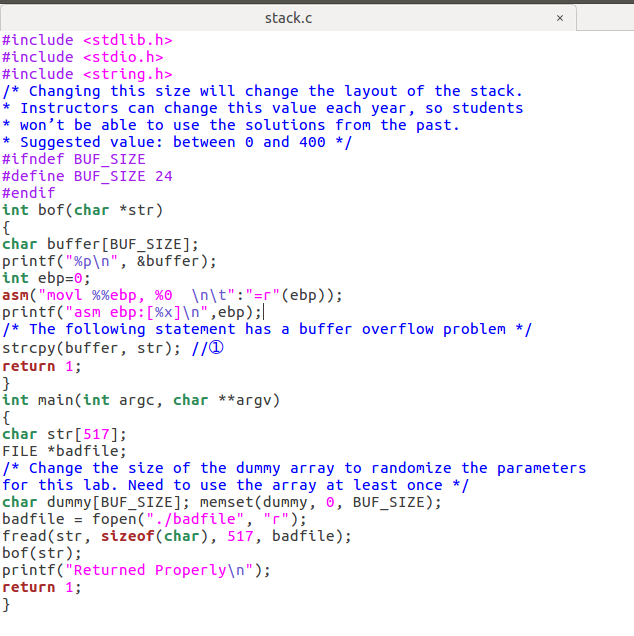
Buffer Overflow Vulnerability Lab

# Task1

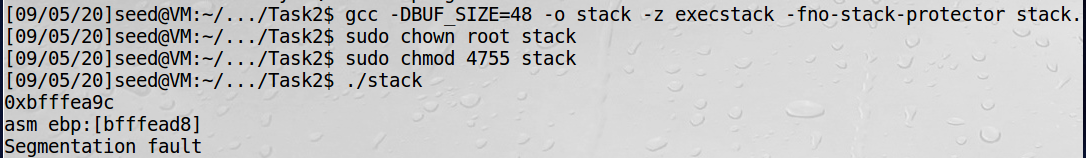


# Task2

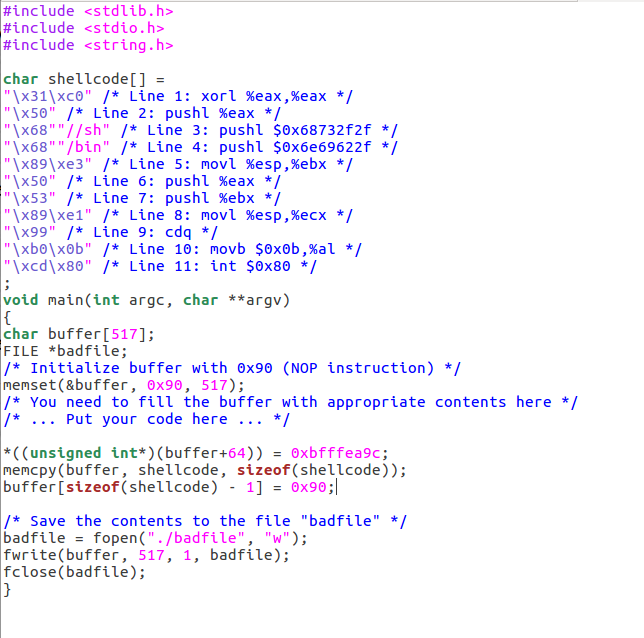
第二个实验在SEED虚拟机中表现很奇怪，使用gdb调试的时候可以正常执行shellcode，直接运行则不可以，查了一下资料是gdb老版本的问题，显示的地址并不是真实地址，并且得到了我的实验核实，因此我将stack.c修改为：



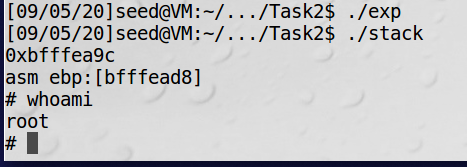
这样的话，在调用bof()时直接打印出buffer地址以及ebp的值，运行可以看到（buffer大小设置为48）：



那么我们把shellcode写到buffer开始的部分，也就是0xbfffea9c，然后把返回地址，也就是0xbfffeadc处的值覆写为buffer的起始地址就可以了，为了做到这一点，我们把shellcode写到badfile的起始位置，把返回地址0xbfffea9c写到buffer的（0xbfffeadc-0xbfffea9c）=64处即可，如图：



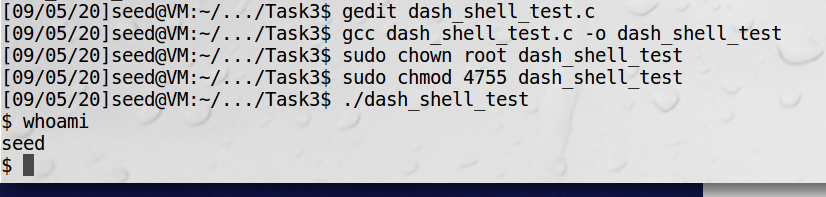
注意memcpy会补0x00，要把它覆写掉，防止strcpy只复制部分字符串，然后生成badfile，运行stack:



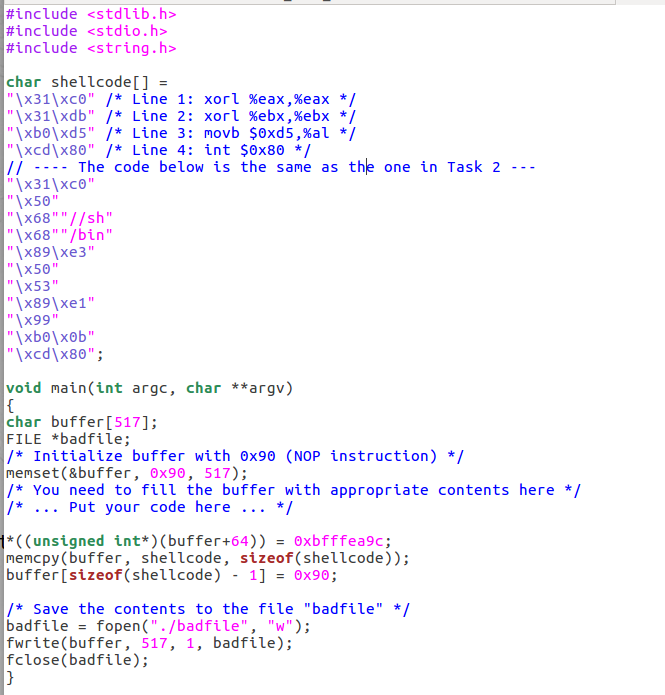
可以看到，得到了root权限的shell

# Task3

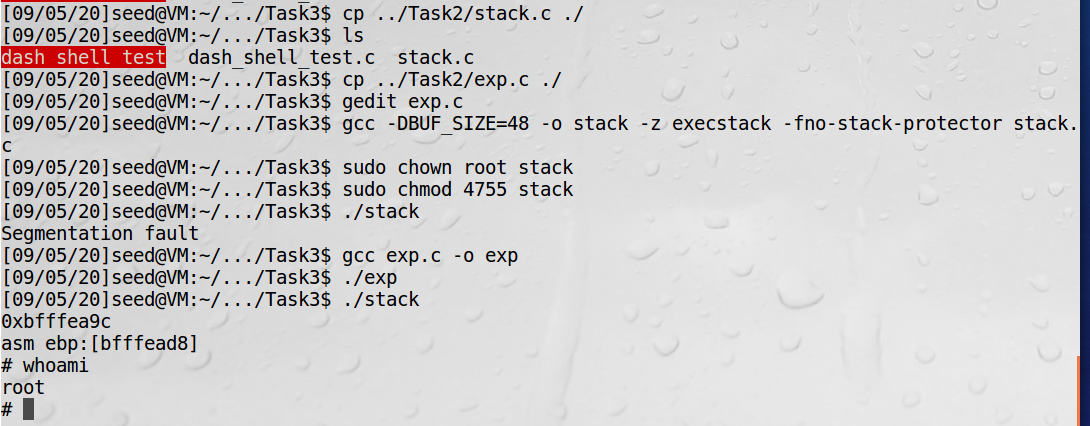
在下图可以看到，即使是Set-UID程序，获得的shell也只有seed的权限：



然后我们修改exp.c中的shellcode，先改变自己的uid：

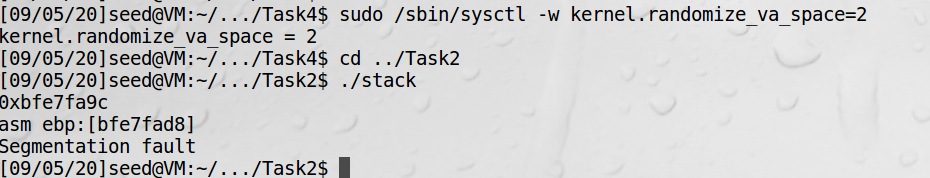


然后编译运行exp.c和stack.c，可以看到，又可以获得root权限了：

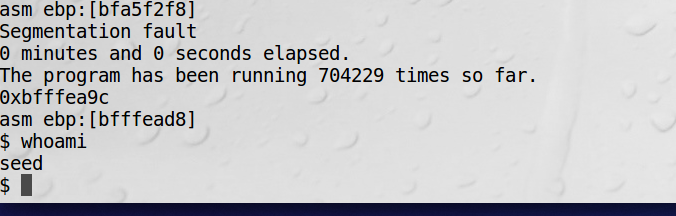


# Task4

开启地址随机化，可以看到之前的攻击失效了：

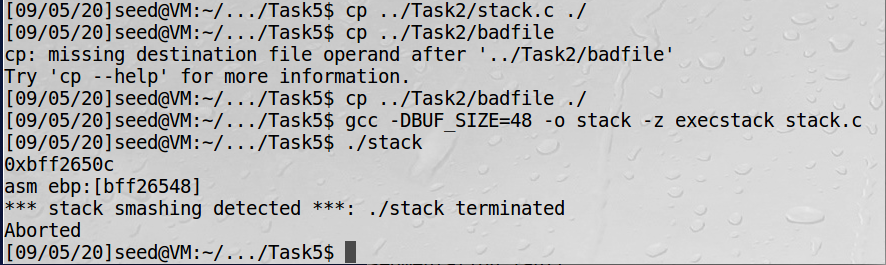


随机跑，跑了一会命中了：

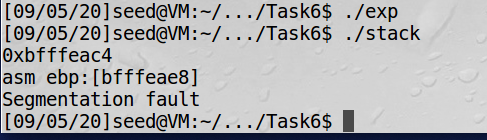


# Task5

由于栈溢出，canary被改写了，通过不了安全检查，程序abort



# Task6

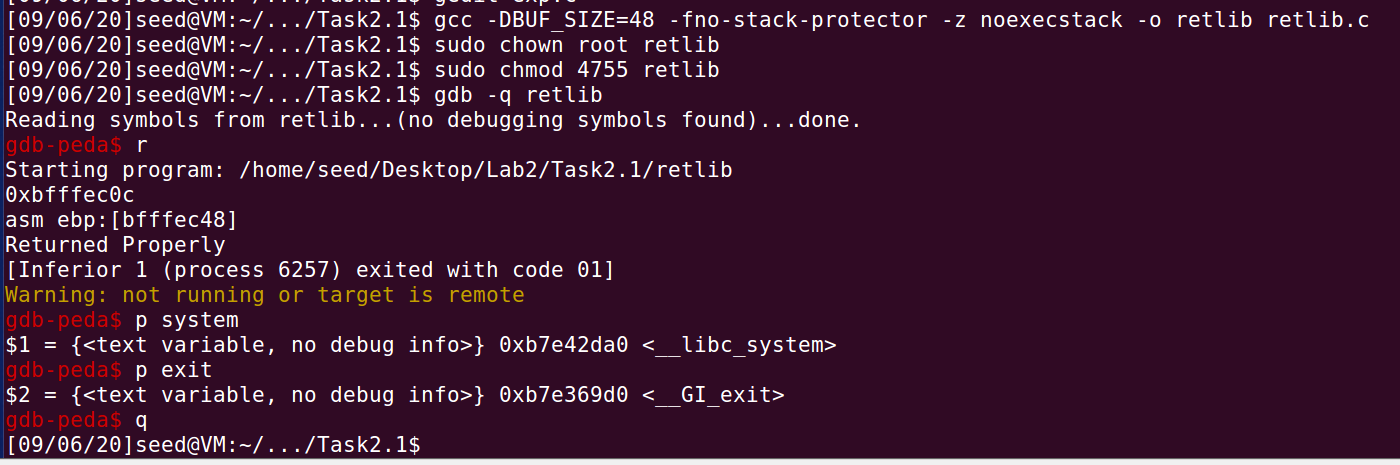


可以看到攻击失效，发生了段错误

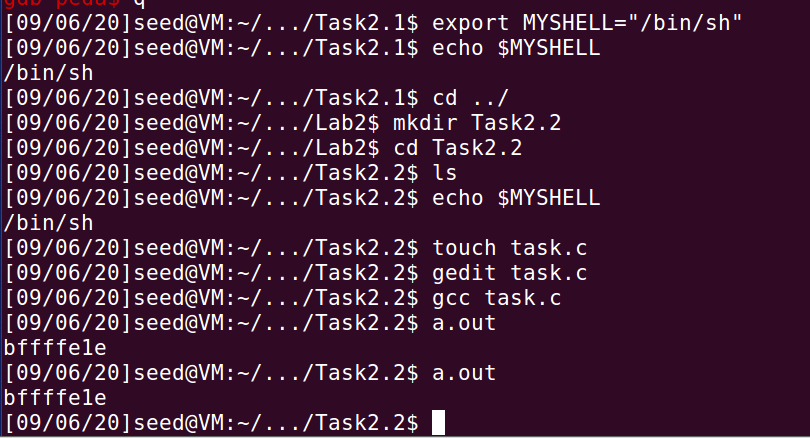
Ret2libc Attack Lab

# Task1

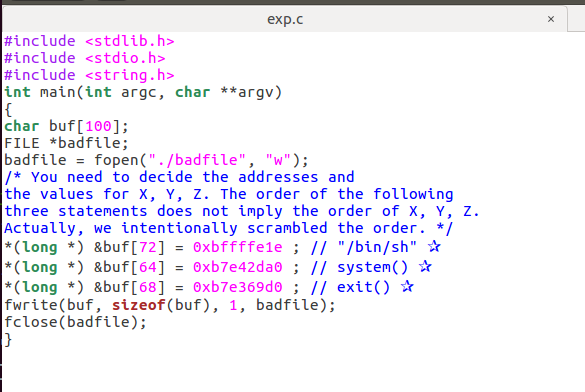
同样为了防止gdb出问题，我们在retlib中打印buffer和ebp，然后执行：



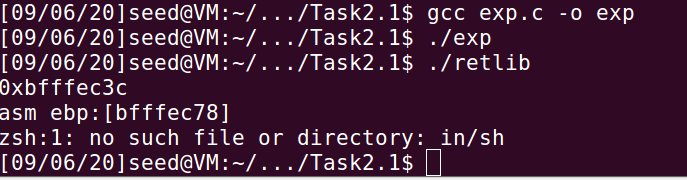
# Task2



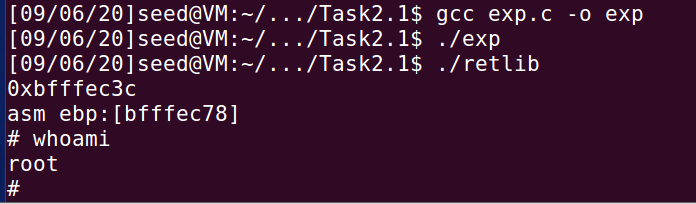
# Task3



然后执行：

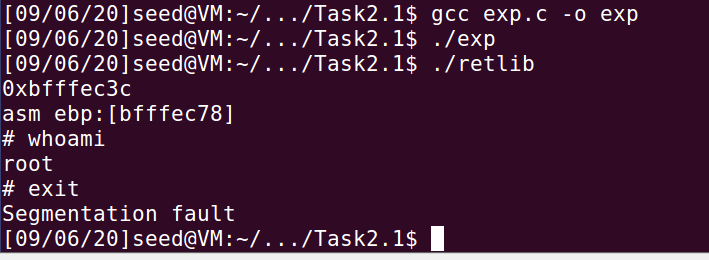


可以看到，传入的命令是in/sh，那我们把/bin/sh的地址从0xbffffe1e改成0xbffffe1c:



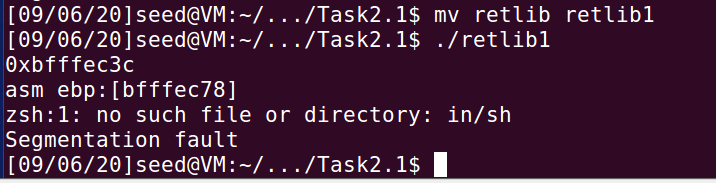
成功getshell

如果不加exit：



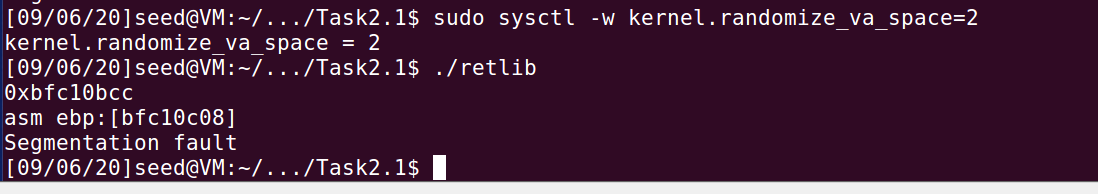
可以成功getshell，但是exit之后，由于返回地址为空，会有sigsegv

如果改个名字：

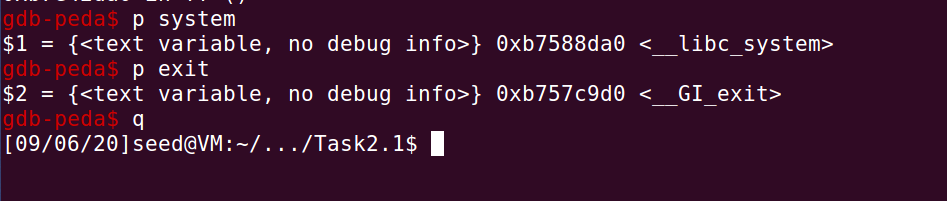


可以看到由于名字长度的变化，所有符号信息出现了一个偏移，包括环境变量，导致传入的命令不对，攻击失效

# Task4

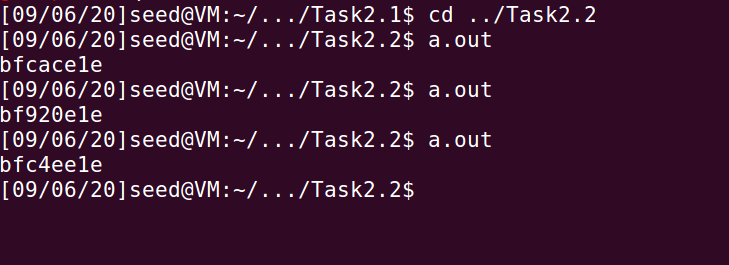


开启地址随机化之后，攻击失效



可以看到system()和exit()的地址都发生了改变，也就是Y,Z发生了改变

然后看环境变量：



环境变量也发生了改变，因此X,Y,Z全部不对，攻击失效