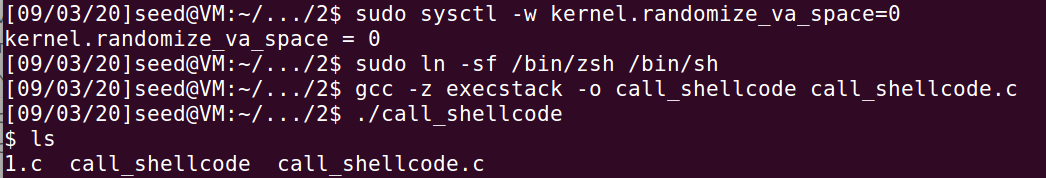
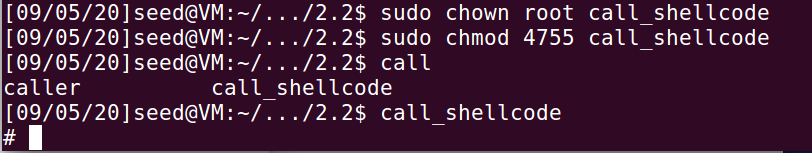
BUF\_SIZE = 33

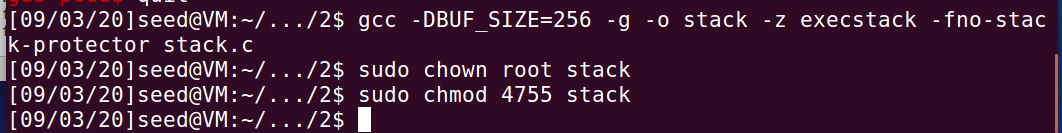
Lab4

Task1

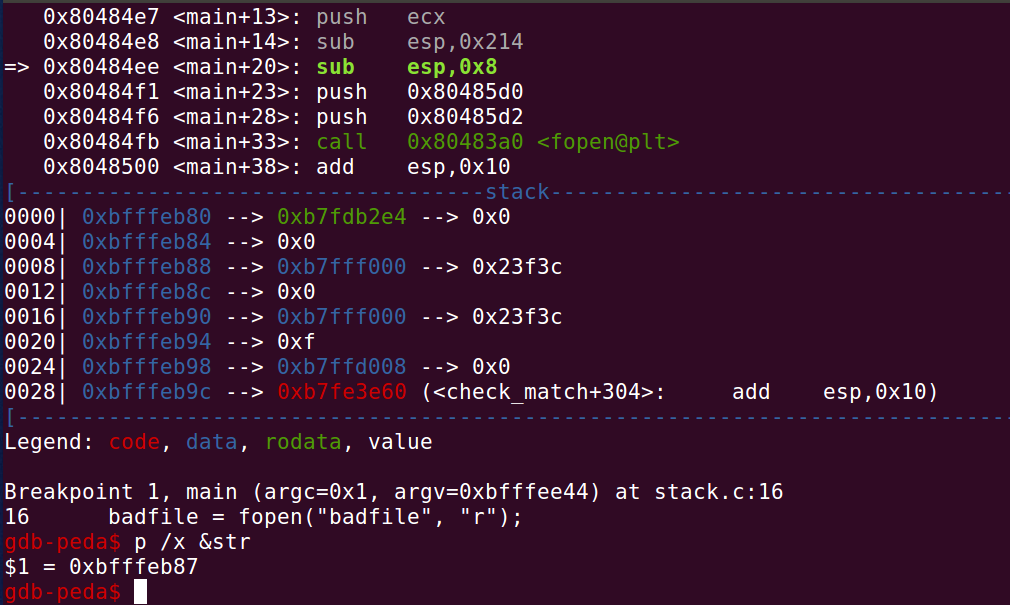


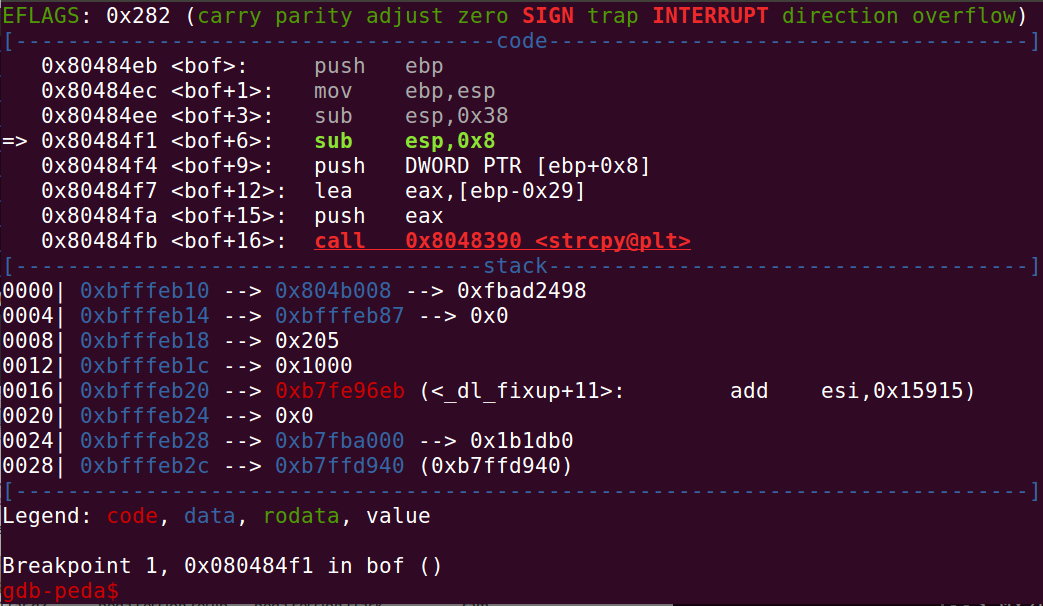


编译以后修改为setuid程序，成功拿到root权限。



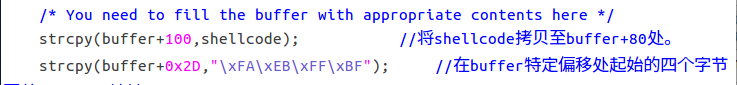
Task2



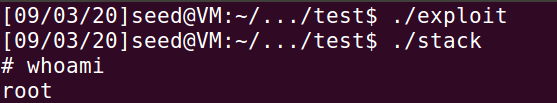


首先得到str的地址为0xbfffeb87，从在bof函数打断点可以看到lea eax,[ebp – 0x29],

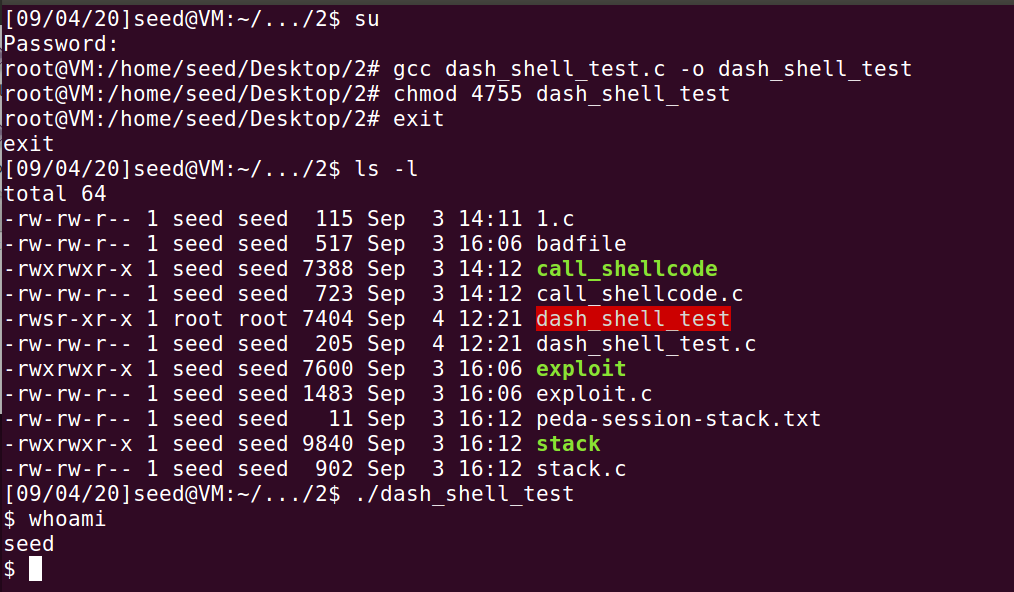
所以可以在给的代码模板中填入



0x29+4=0x2D指向了bof中ret的地址，其后填入shellcode的地址即可，随后根据计算出的shellcode地址应该等于0xbfffeb87+0x64，但是结果不对，查阅资料显示，可能是gdb中的地址空间与实际运行不符，稍微尝试后填入0xbfffebfa即可成功。

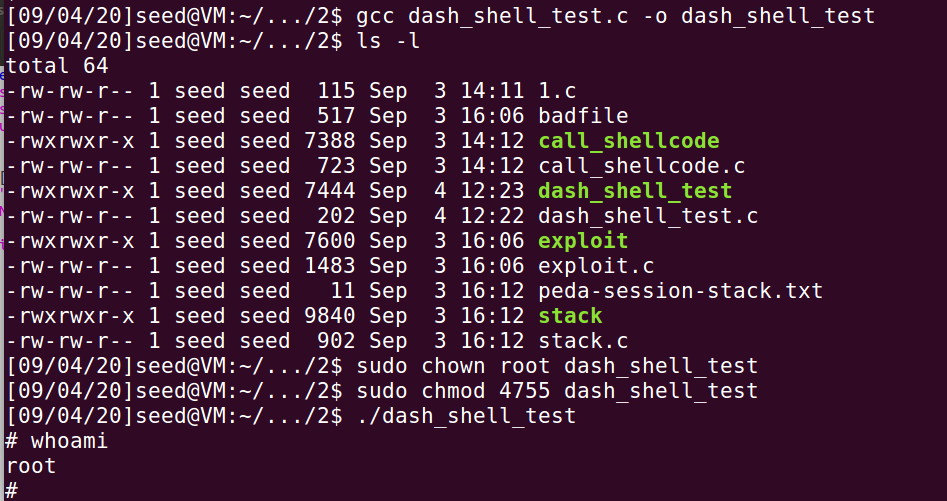


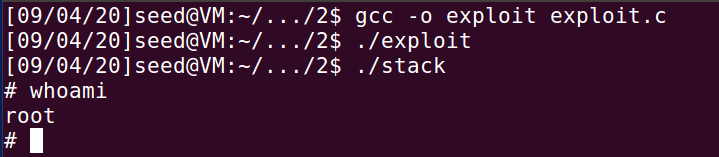
Task3



如上图可以看到更改了shell的链接已经拿不到root权限了，这是因为它的机制启动shell后自动降级的原因。

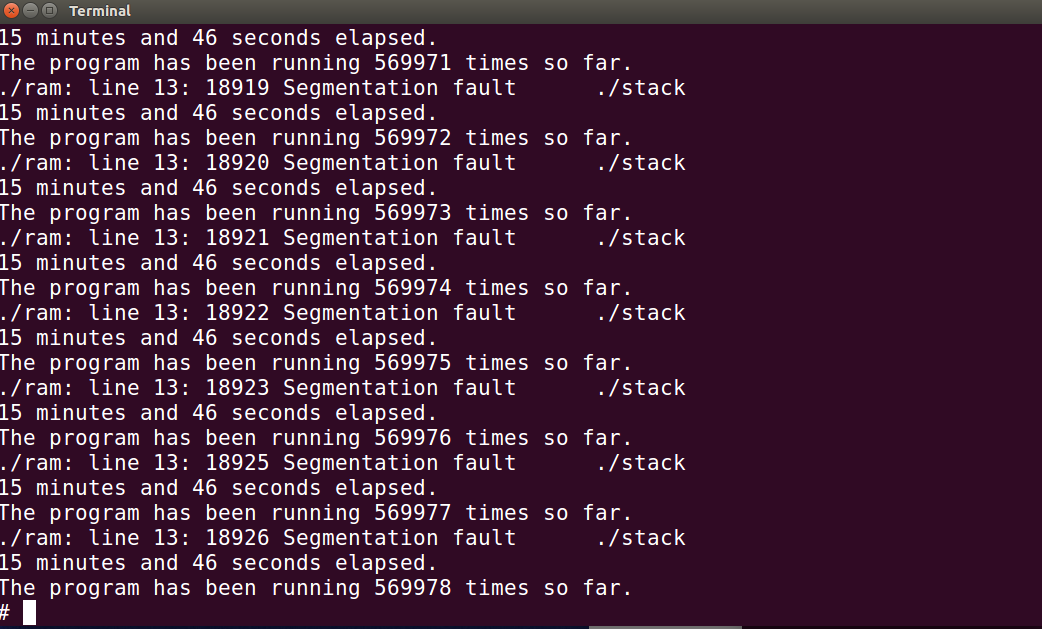






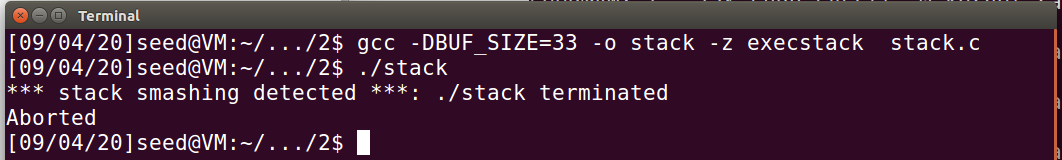
在shellcode中加入setuid(0)后，成功拿到root权限，这是因为权限被降级，通过setuid将自己权限升为root，逃脱了降级。

Task4



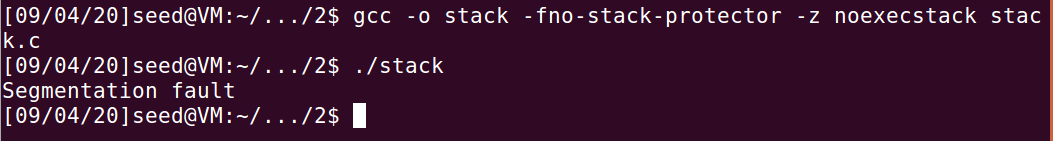
再不断的尝试后，等了很久得到了shell。

Task5



开启了栈保护后程序运行错误。

Task6

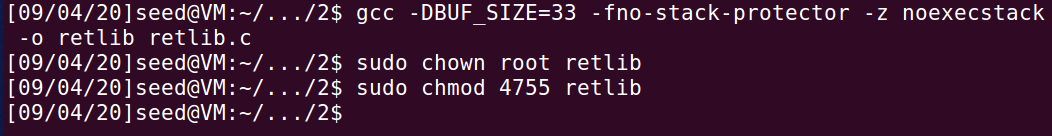


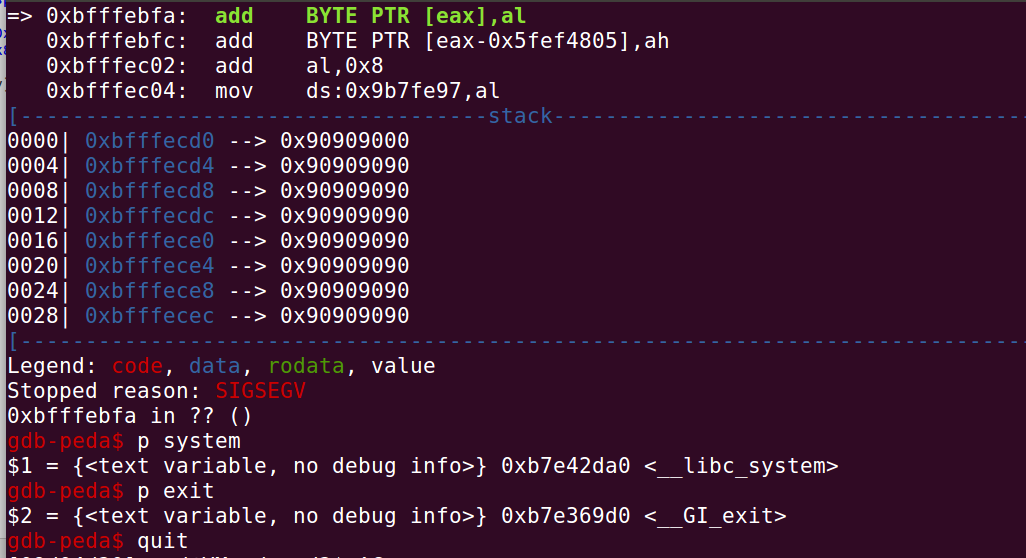
开启栈不可执行后，直接提示段错误。

BUF\_SIZE = 33

Lab5

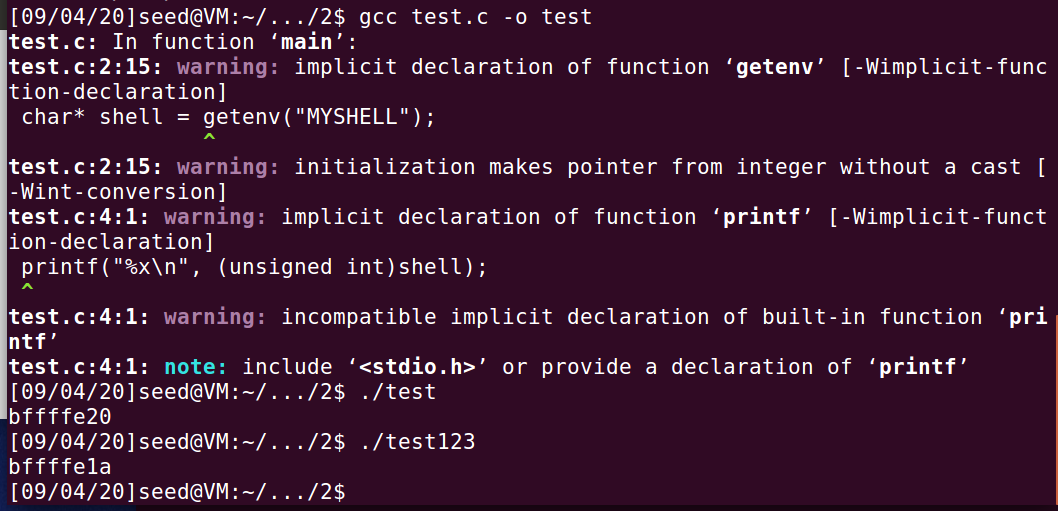
Task1





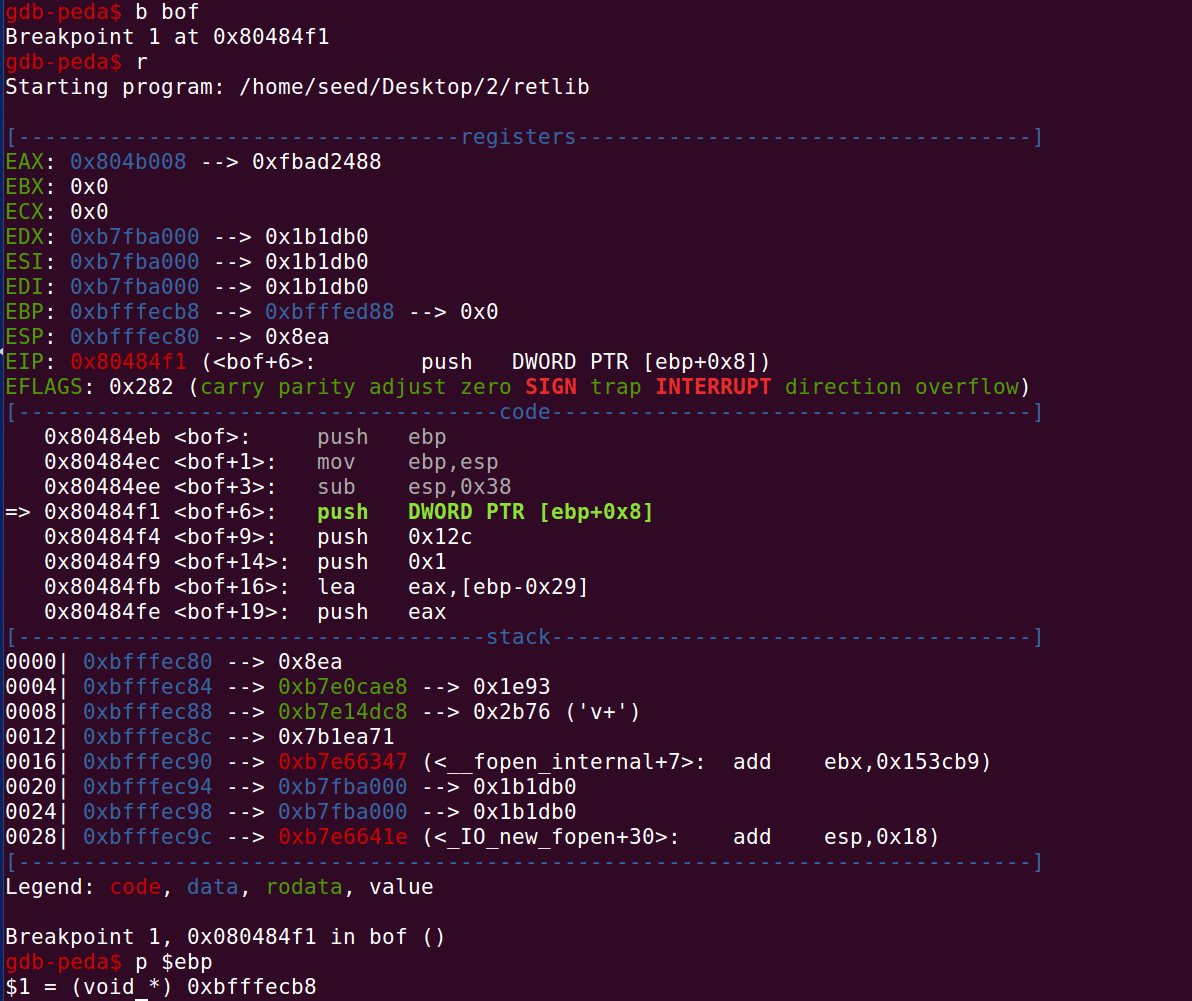
如图所示得到函数的地址

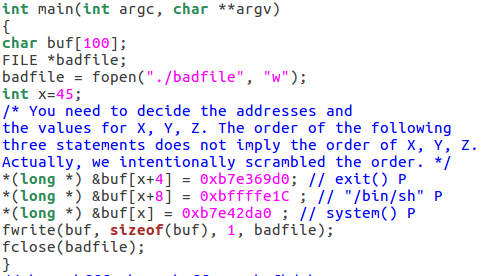
Task2



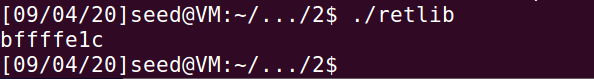
从这里可以看出名字不同该环境变量的地址也不同，所以可以事先改成目标程序的名字测出它的该环境变量地址。

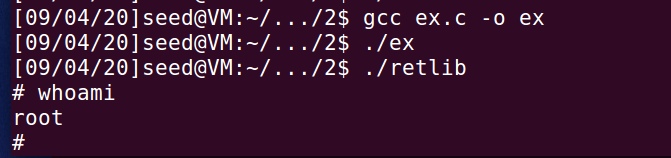
Task3



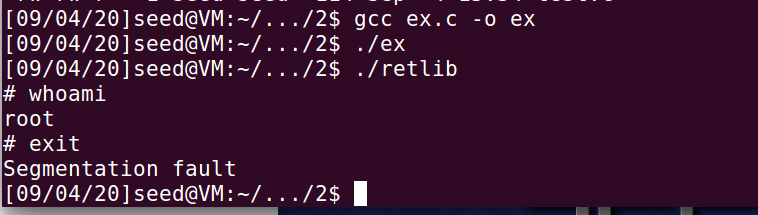


通过前面的实验可以确认数组内容的函数地址，而buf 中的x为ebp+4，从上图gdb中，在bof中设置断点，故ebp距离buffer为0x29，故ebp+4=45。



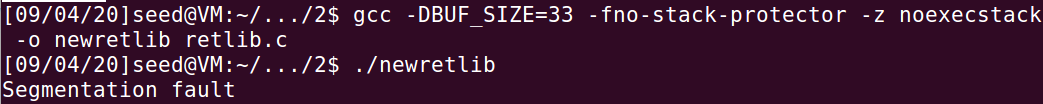


Attack variation 1:



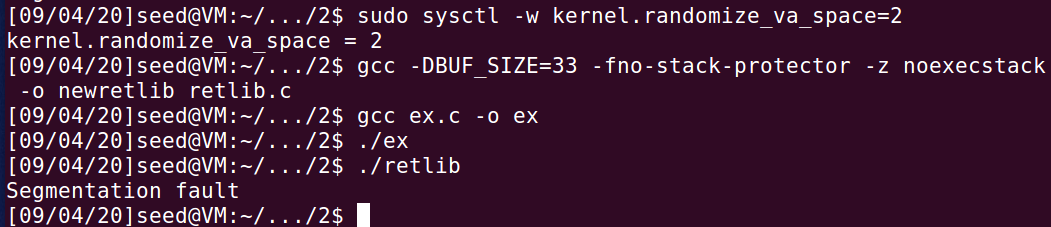
如果不加入exit函数，退出shell后会出现段错误

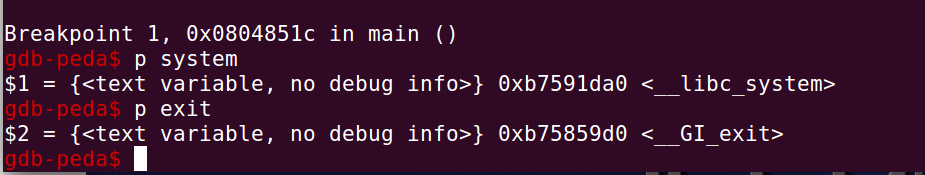
Attack variation 2:

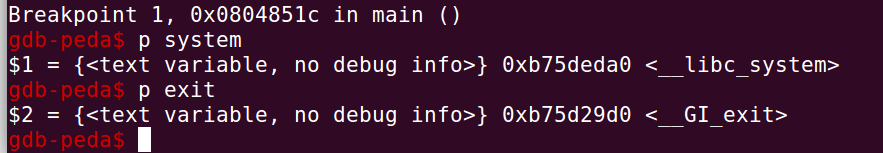


改变名字后在运行会发生段错误，这是由于环境变量的地址改变了。

Task4







System与exit地址变动

开启地址随机化后可以发现函数地址会不停的变化，所以会发生段错误。

