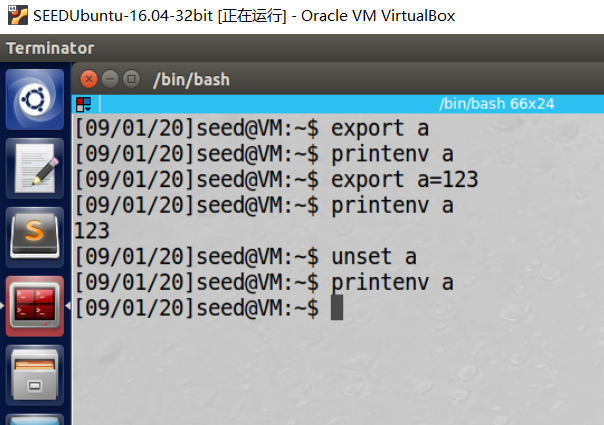
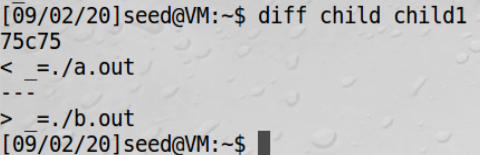
**Lab 1实验报告** 57118110 杨紫瑄

Task 1:

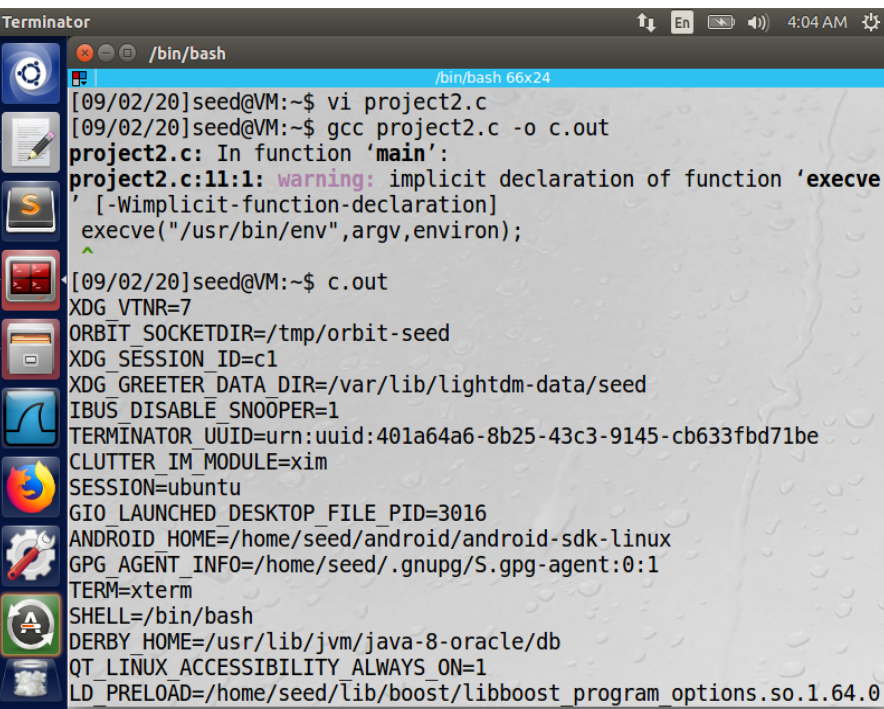


Task 2：



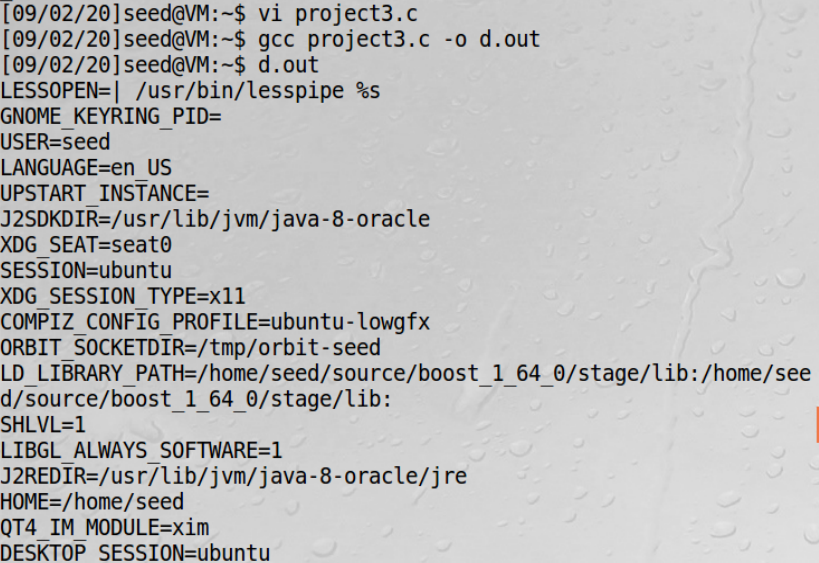
将题目中代码运行后结果分别保存在文件child和child1中，两次可执行文件名分别为a.out和b.out，使用diff命令将文件child和child1比较后，结果如图，可以得出两次输出的结果除了最后一行可执行文件名的区别外没有其他不同之处，所以能够得出结论父进程的环境变量能够被子进程继承。

Task 3:



Excve()中环境变量不会自动继承，需要通过参数来传递。

Task 4:

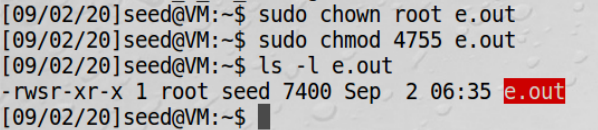


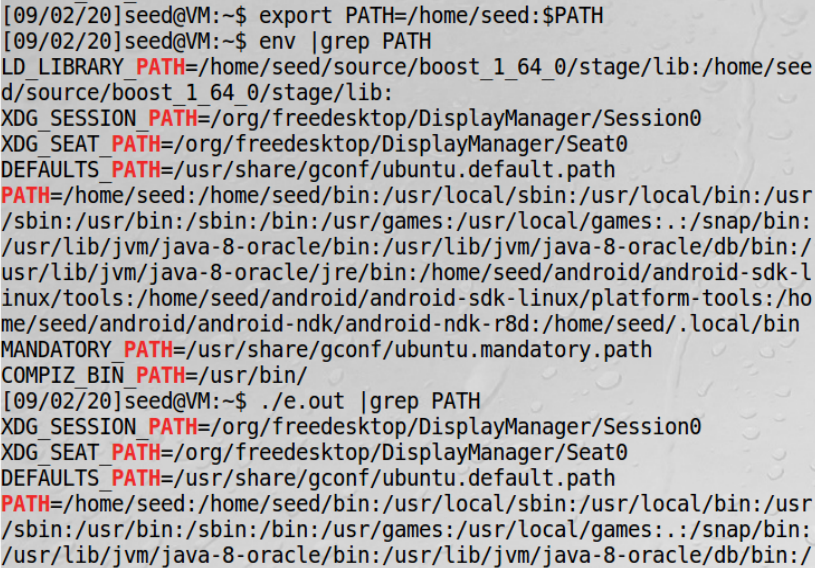
System()中环境变量会被自动传递，不需要通过参数来传递。

Task 5:

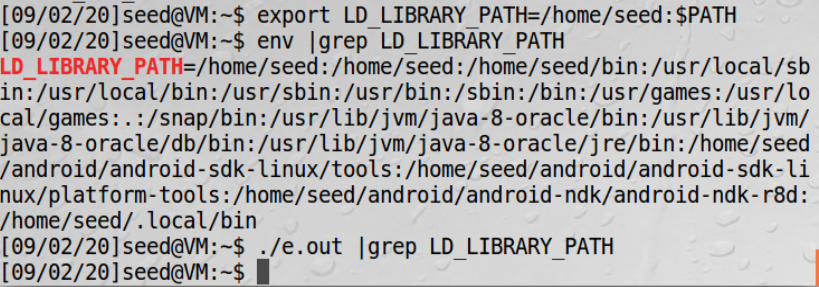


将代码编写到文件project4.c中并设可执行文件为e.out

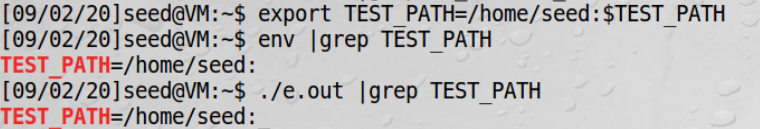




运行发现可以继承父进程PATH的环境变量

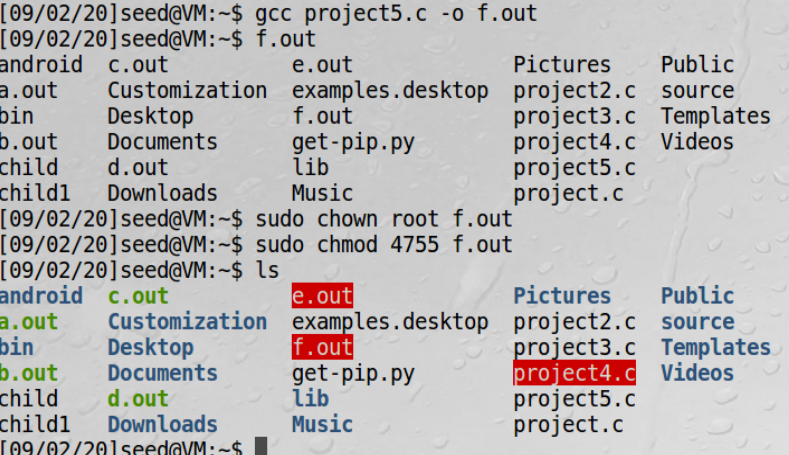


不会继承父进程LD\_LIBRARY\_PATH的环境变量

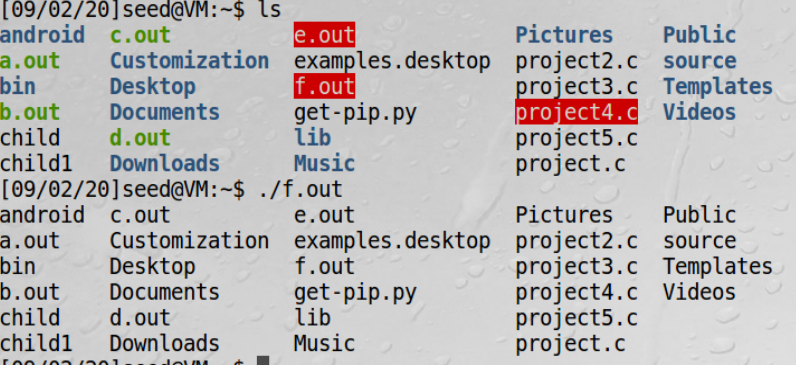


可以继承自定义父进程TEST\_PATH的环境变量

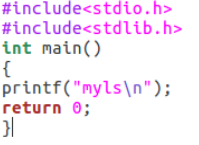
Task 6：



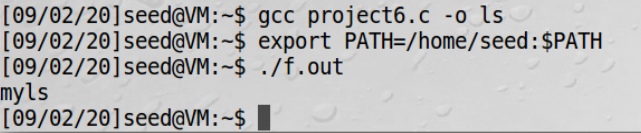
将代码编写到project5.c文件中，设其可执行文件为f.out，并将其设置为SET\_UID程序



上为运行/bin/ls程序的结果



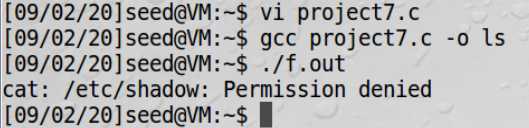
自己编写的ls程序 并将运行后可执行文件命名为ls



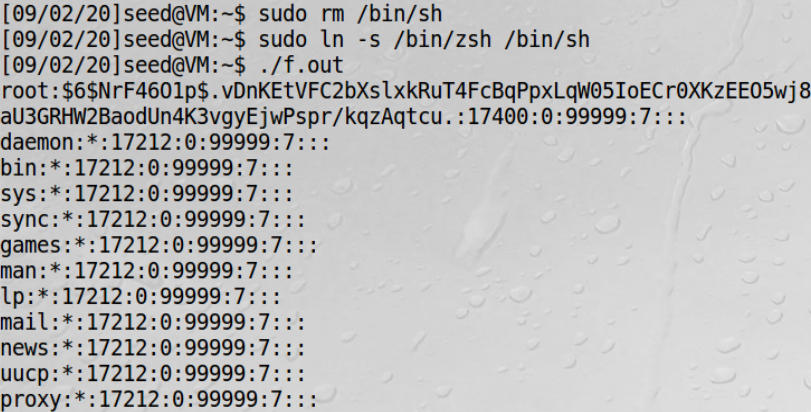
并将ls路径加入到PATH的第一个，再运行上一步的可执行文件f.out，发现执行的是自己编写的ls程序

若换成需要root权限：



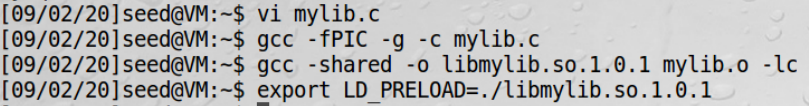


则不能成功

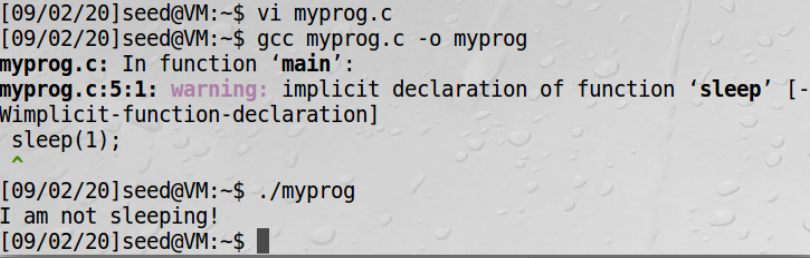


将/bin/sh链接到dash修改成链接zsh，可以看出攻击的效果

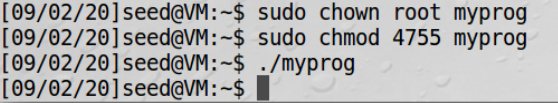
TASK 7：



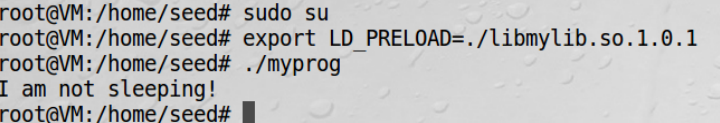
编译成动态库文件并设置LD\_PRELOAD环境变量



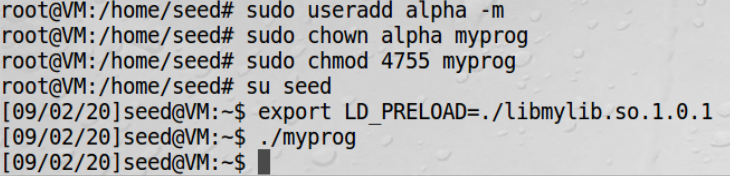
发现调用的是自定义的sleep动态库，此时shell定义的环境变量export LD\_PRELOAD传递给了子进程myprog非特权程序



将myprog为Set-UID程序，再次执行发现执行的是系统的sleep说明特权程序类型的子进程不继承父进程的动态链接库LD\_PRELOAD环境变量

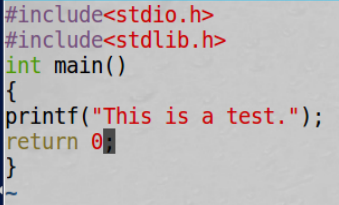


将用户切换为root，再次执行发现执行的是自己设定的sleep动态库，说明此时子进程可以继承父进程的LP\_PRELOAD的环境变量



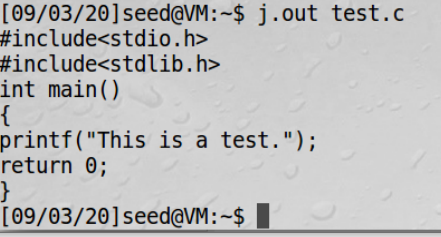
增加一个非root用户，再次执行发现执行的是系统的sleep，说明在两个非root用户之间防御成功

Task 8:

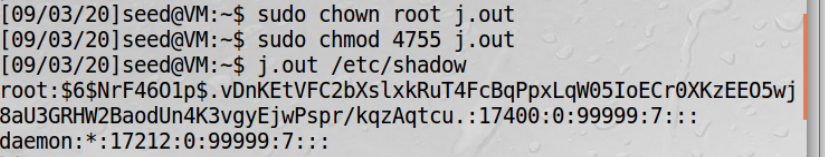


新建test.c文件测试

使用system()情况下：



正常运行后可以看到测试文件内容



将程序升级为特权程序，运行后发现有特权查看密码文件



将文件拷贝到root目录下

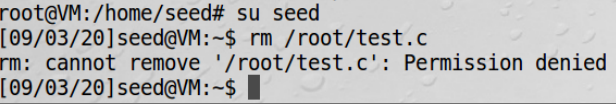




提升程序的特权

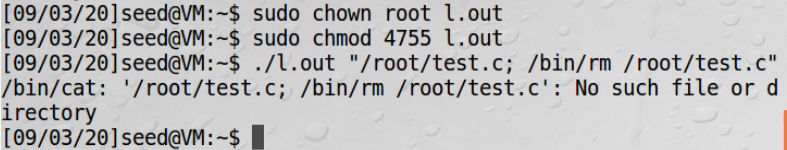


发现文件被删除，权限泄露



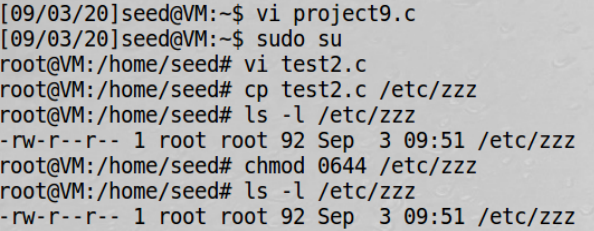
而普通用户是无法删除root目录下的文件的

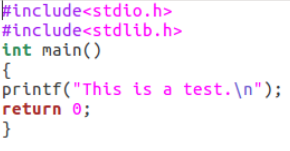
若改用execve():



则可避免通过参数设置泄露权限

Task 9:

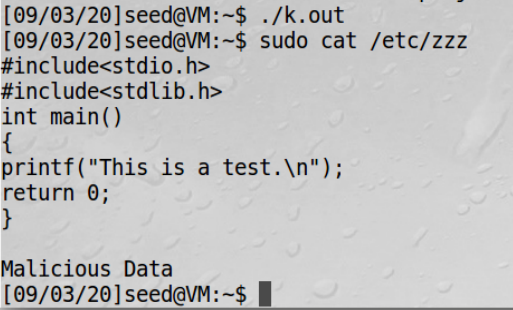


test2.c

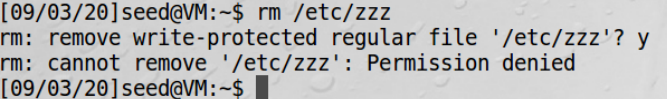
将题目中代码编写到project9.c文件中并设其可执行文件为k.out，新建test2.c文件来创建 /etc/zzz文件



设置root特权



运行后发现文件被修改



而在未设置特权的情况下文件是无法被修改的

实验结论：此次试验使我更加了解环境变量是如何影响程序的，也让我对set-uid程序更加了解，同时也使我对linux语句的使用更加熟练。