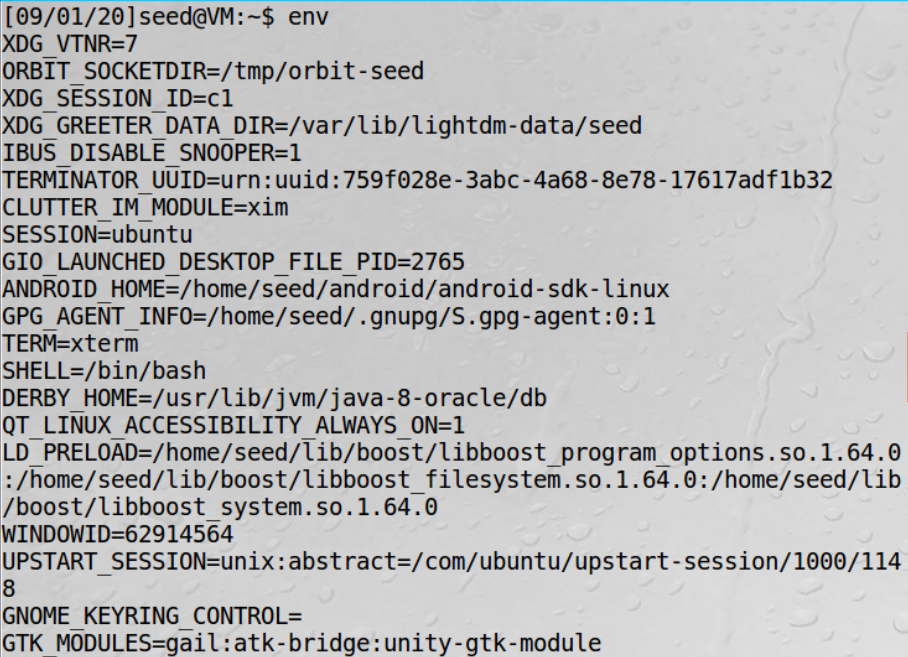
**Lab1实验报告**

57117235李娜

Task1: Manipulating Environment Variables

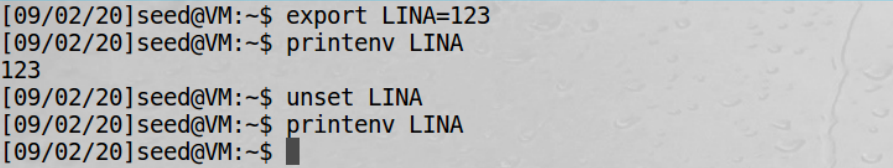
用env（或printenv）输出全部环境变量：



用printenv输出指定的环境变量PWD：



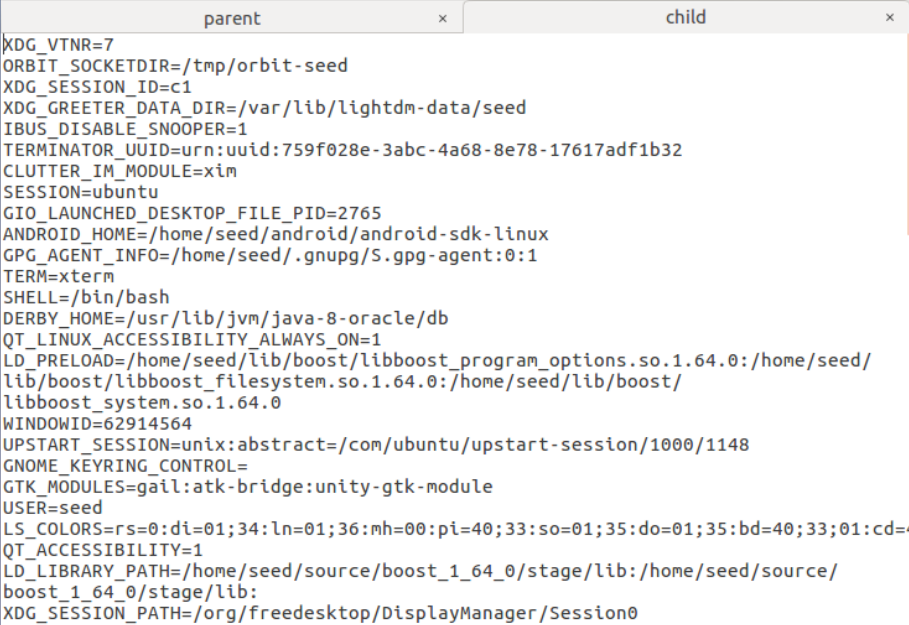
用export和unset命令设置和取消环境变量LINA：



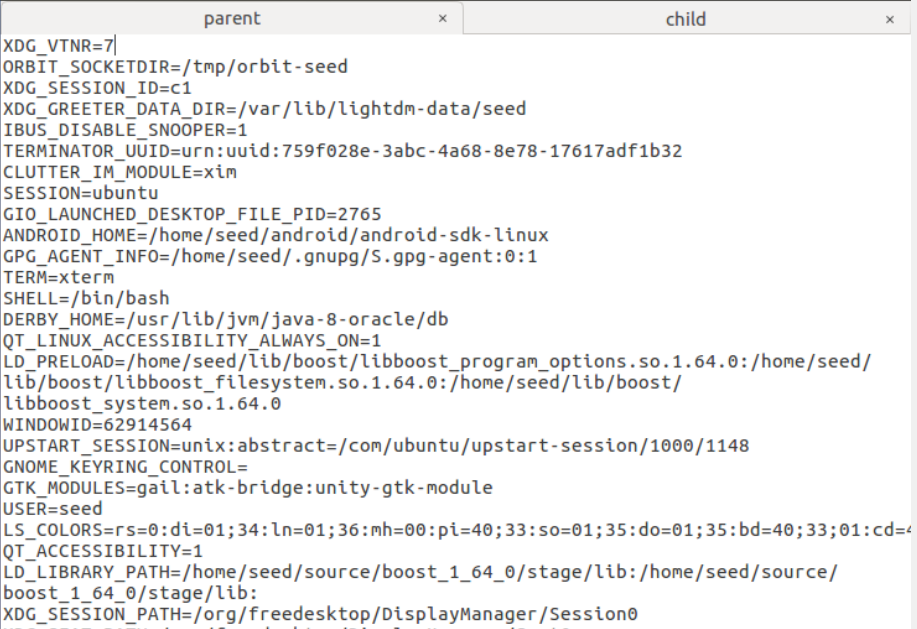
Task2: Passing Environment Variables from Parent Process to Child Process

编译实验代码，将两次打印结果分别输出到child和parent文件中。

child文件内容：



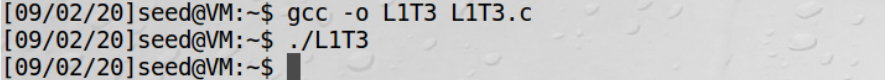
parent文件内容：



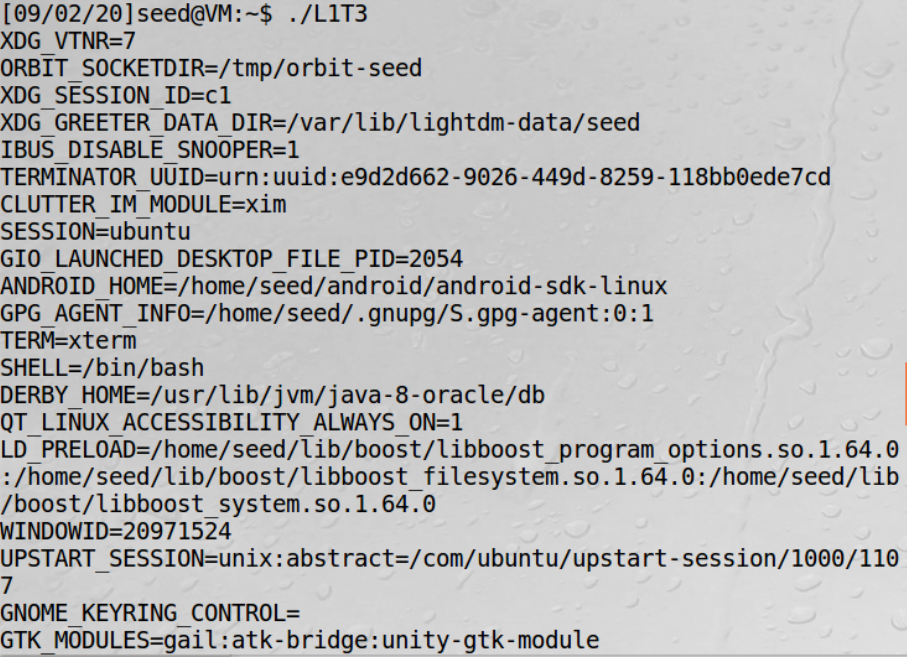
通过比较这两个文件，发现两次输出的结果完全一致，说明通过fork()函数创建的子进程继承了父进程全部的的环境变量。一个现有进程可以调用fork函数创建一个新进程，子进程将获得父进程数据空间、堆、栈等资源的副本，这和实验结果一致。

Task3: Environment Variables and execve()

当execve()函数传入的环境变量参数是NULL时，子进程没有继承环境变量，所以此时没有打印出环境变量：



将enecve()函数参数改成environ，则子进程继承了全部环境变量，运行程序后打印出了全部的环境变量：



Task4: Environment Variables and system()

system()函数被调用时，它会调用shell，shell进程再执行命令。子进程会继承父进程的所有环境变量，运行实验代码后，可以看到打印出了所有的环境变量：

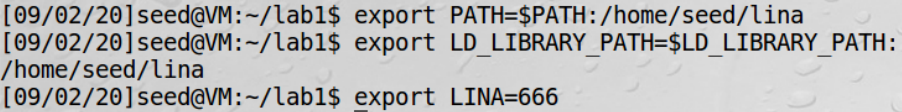


Task5: Environment Variable and Set-UID Programs

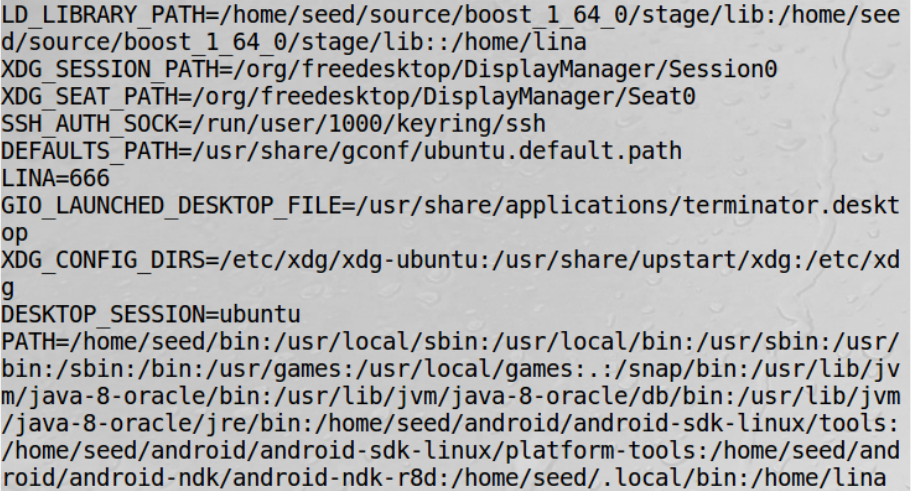
编译实验代码后，用以下命令修改程序权限为root权限，使其变成一个Set-UID程序：



使用一般用户登录终端，用export命令设置环境变量PATH、LD\_LIBRARY\_PATH和LINA：



运行实验代码，可以看到，程序输出的环境变量包括以上定义的三个：



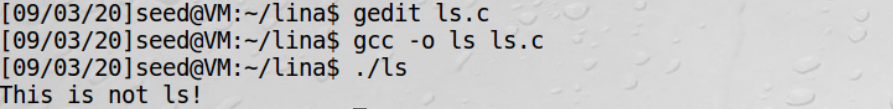
这些环境变量是在shell进程中设置的，shell进程调用程序后，派生了一个子进程，子进程继承了父进程中设置的环境变量。

Task6: The PATH Environment Variable and Set-UID Programs

在/home/seed/lina文件夹下新建ls.c文件，内容如下:



在该文件夹下编译生成可执行文件，执行结果如下：

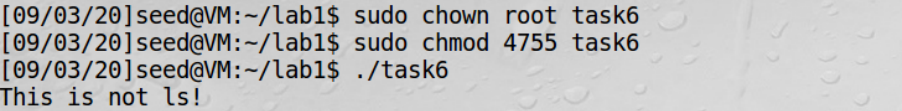


修改环境变量并以普通用户权限执行程序：





设为特权程序，再执行，输出结果和之前一致：

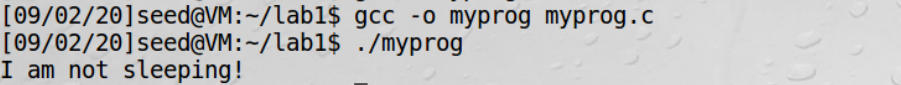


system()函数调用了shell的环境变量，所以执行程序时，按照环境变量PATH的顺序先到/home/seed/lina中找到可执行文件ls。这也说明了，可以通过改变环境变量，让程序执行事先写好的恶意文件，这是很危险的。

Task7: The LD\_PRELOAD Environment Variable and Set-UID Programs

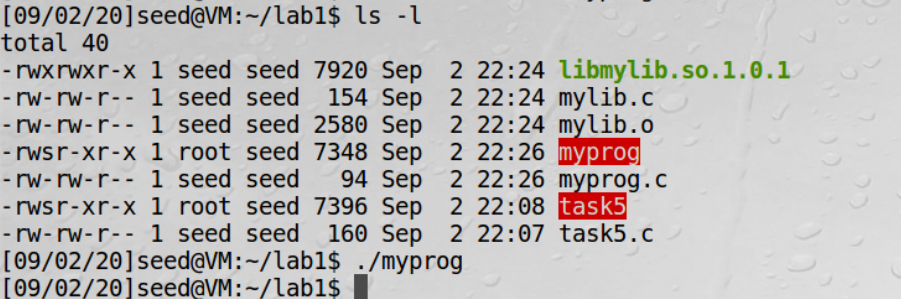
实验用gcc将mylib.c编译生成动态链接，并将其添加到环境变量LD\_PRELOAD中。

以普通用户身份运行程序：



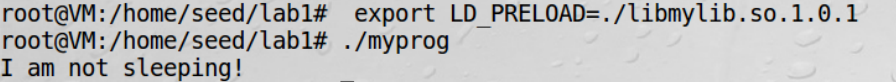
当在shell进程中改变了环境变量后，shell进程再运行程序，子进程继承了父进程的环境变量，链接了设置好的sleep()函数。

以普通用户身份运行具有root权限的程序：



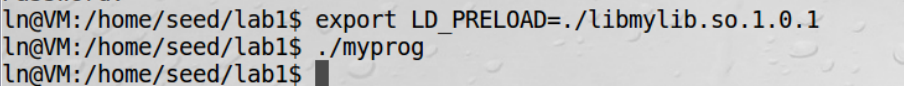
Set-UID程序未继承设置的环境变量。

在root账户下设置环境变量，并运行程序：



在root账户下设置环境变量后运行程序，子进程继承了该环境变量。

设置程序为用户1的Set-UID程序，在用户2的shell中运行程序：

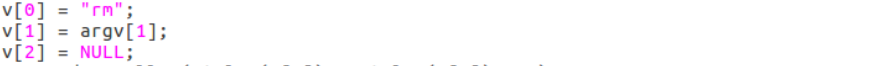


该子进程未继承设置的环境变量。

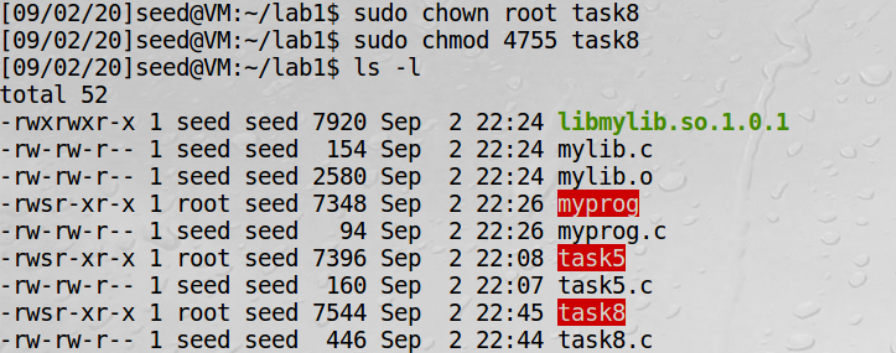
由此可以得出结论，环境变量LD\_PRELOAD可能不能被子进程继承。

Task8: Invoking External Programs Using system() versus execve()

将实验代码中的cat命令改成rm命令：



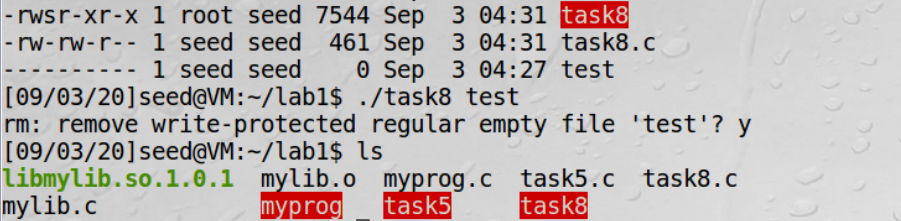
设置实验代码task8的权限，使其成为一个特权程序：



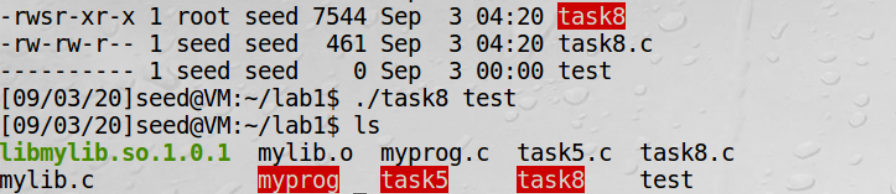
新建一个test，并设置它的权限为000，即不可读不可写不可执行：



使用system()函数，test文件可以被移除：



使用函数execve()，传入的环境变量为空，运行程序，文件test并未被移除：

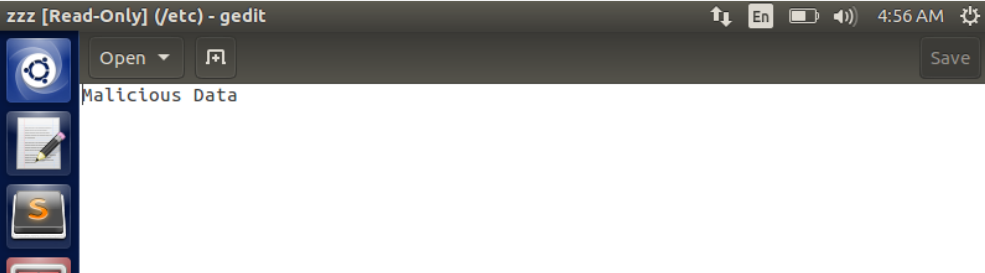


Task9: Capability Leaking

设置实验代码task9为Set-UID程序，新建空白文件/etc/zzz，并执行task9：



再打开/etc/zzz文件，发现文件写入指定内容：



开始以root权限打开文件后，恢复成用户权限，子进程的文件还未关闭，仍可以执行写操作，导致特权泄露。