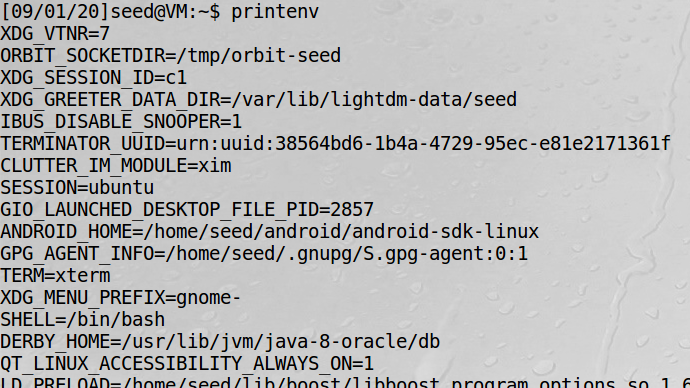
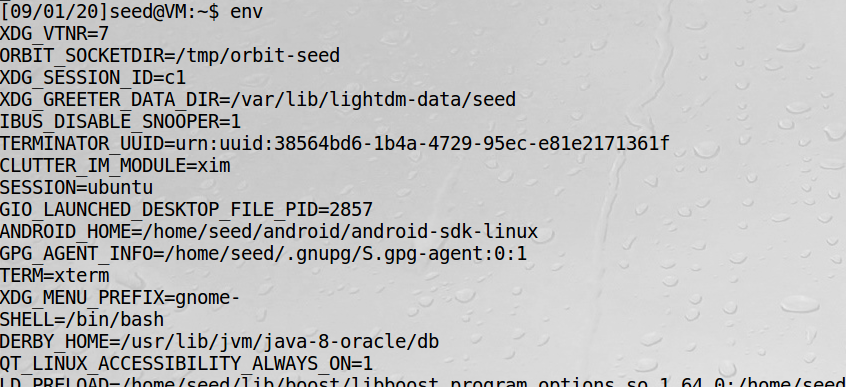
**学号57118127 姓名 孙博**

**Task1**

Printenv命令



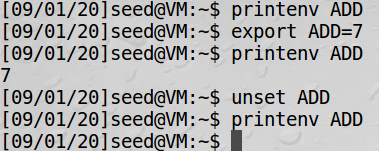
Env命令



当想输出某一个环境变量时

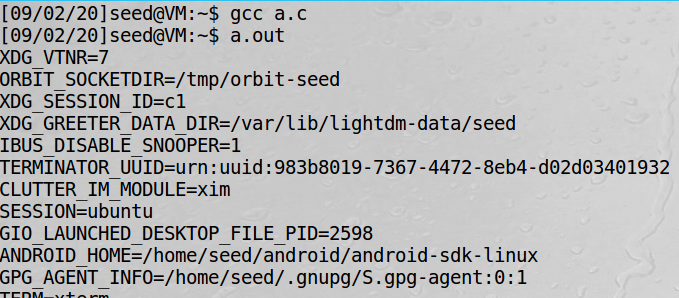


环境变量的增加与删除

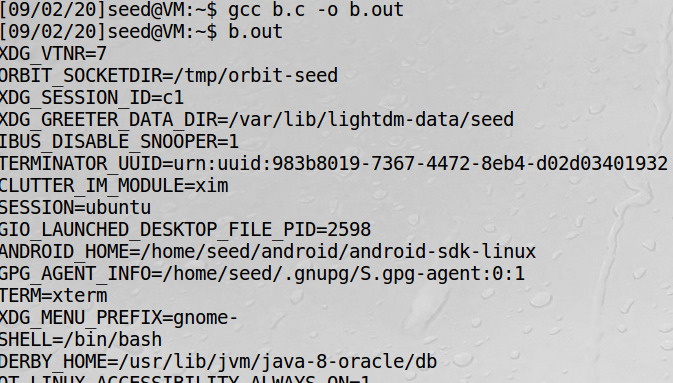


**Task2**

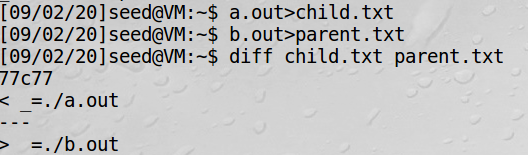
第一步是输出子进程的环境变量



第二步是输出父进程的环境变量



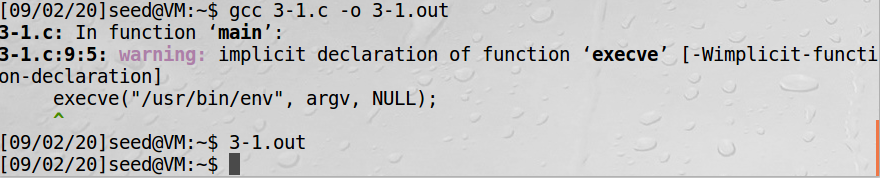
将两次的输出结果保存到txt文件中进行比较

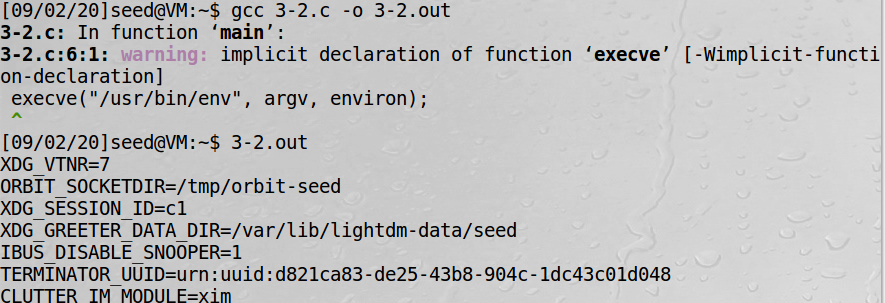


通过比较发现二者的不同仅是由于命名不同带来的，即父进程与子进程输出的环境变量完全相同，因为父进程调用fork()创建子进程，子进程成为父进程的副本，复制父进程的所有进程内存到其内存地址空间中。父、子进程的“数据段”，“堆栈段”和“代码段”完全相同，环境变量也完全相同

**Task3**

将两次代码分别输入、编译，得结果如下

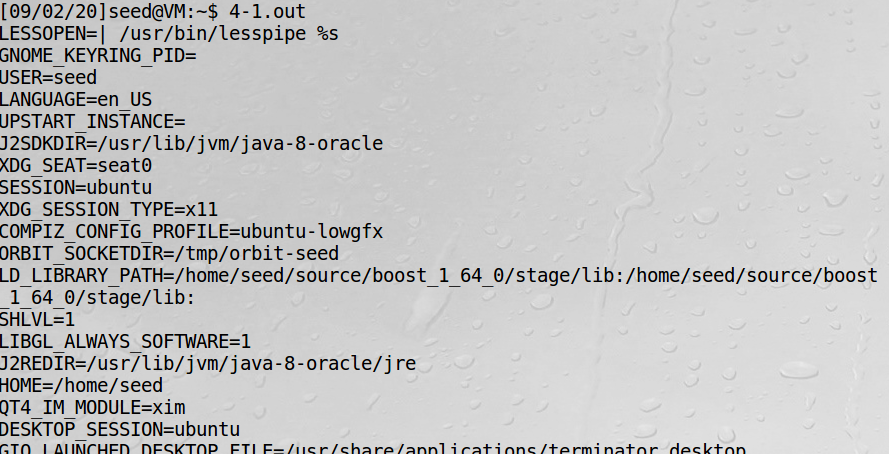




execve()函数第三个参数为传递给执行文件的新环境变量数组。空指针时无输出；为环境变量字符数组时输出环境变量。

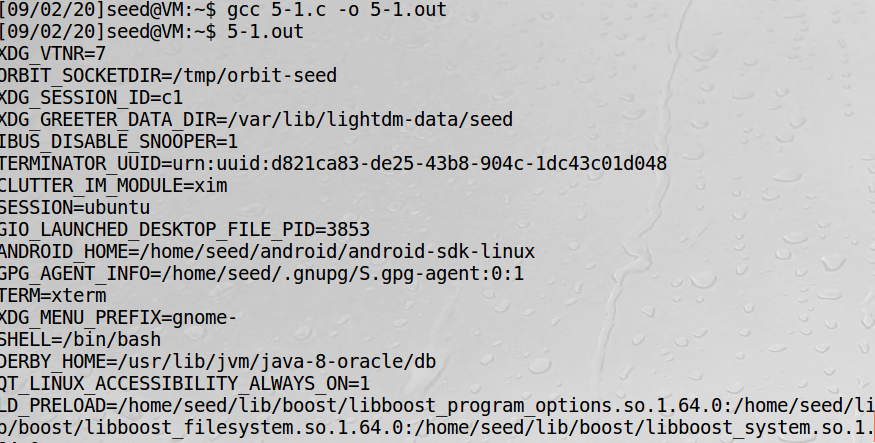
**Task4**

调用system()函数部分输出如下：



**Task5**

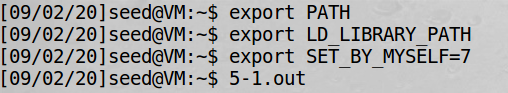
编译并运行输出环境变量的代码如下



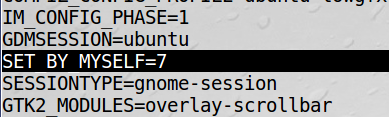
给程序添加root权限



在用户shell中添加自己设置的环境变量并重新运行set-uid程序



发现输出的环境变量中有自己设置的环境变量



**Task6**

编译程序，将其所有者更改为root，并将其设置为Set-UID程序



确认获取到root权限



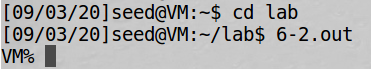
将/bin/sh链接到zsh



新建文件夹lab，将可执行文件和/bin/sh重命名的ls粘贴到lab中，修改PATH到lab文件中



运行得如下结果

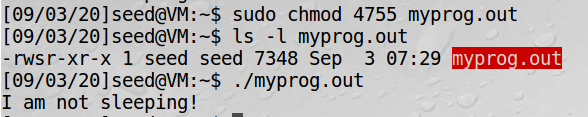


**Task7**

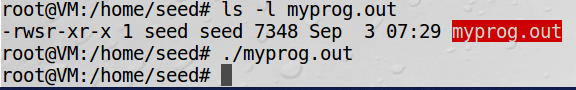
普通使用者的权限以普通程序运行myprog，结果如下，发现链接到mylib



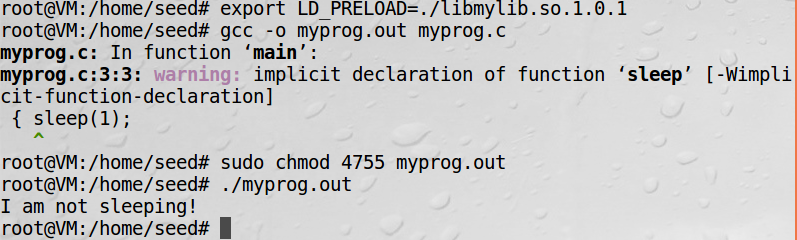
普通使用者的权限以set-uid程序运行myprog，结果如下



在root下编译myprog.out，并设置成set-uid程序，在普通用户下执行，发现是链接到真正的libc中



在root下编译myprog.out，并设置成set-uid程序，在root下执行，发现链接到的是mylib.c



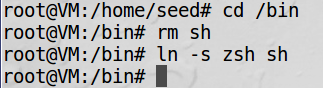
在普通用户下编译并设置成set-uid程序，再在另一用户下执行，结果如下



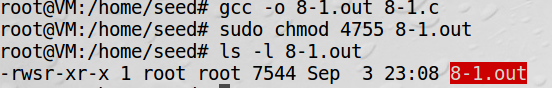
搭载同一用户下设置set-uid程序并执行则可以输出结果，**说明只有用户自己创建的自己才能去运行，才会使用LD\_PRELOAD环境变量，链接到自己的sleep函数，否则忽略LD\_PRELOAD环境变量，不会链接到自己的sleep函数**

**Task8**

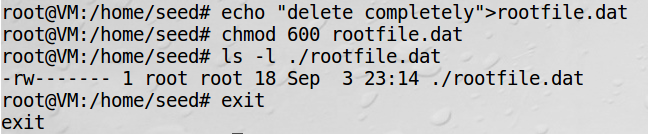
可以删除不可写的文件 先将sh链接到zsh上，防止攻击失败



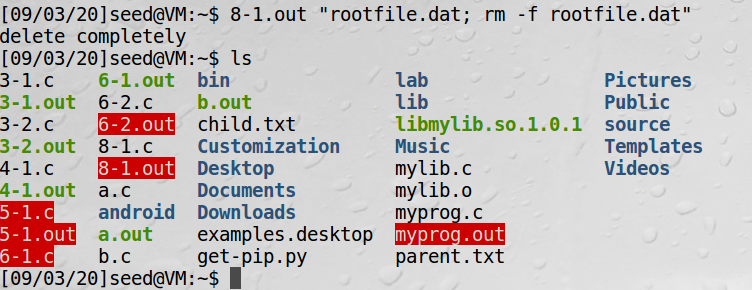
运行已给文件，并给予set-uid权限



创建rootfile文件作为删除标记 该文件并没有删除权限



在执行8-1.out时，在程序后添加rm -f rootfile.dat，使system会继续执行该命令并删除该文件



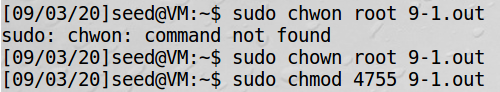
文件已经删除

注释掉system()语句，取消execve()语句的注释;程序使用execve()调用该命令，再重复上述操作，不能删除文件。

**由于system()工作方式是将后续的命令字符都传递给shell，交由shell来执行，这样很可能被攻击者利用shell的漏洞进行攻击;而execve()则是创建子进程，参数是传递给子进程的，因此可以防止此类攻击。**

**Task9**

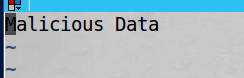
编译以下程序，将其所有者更改为root，并使其成为Set-UID程序。



创建/etc/zzz，再以普通用户运行9-1.out



打开/etc/zzz，发现已经被修改



**因为代码中的fork函数会在创建子进程。子进程拥有父进程的数据、堆栈的拷贝。同时拥有的父进程的SetUID权限。**