**操作系统实验一实验报告**

**基本信息：**

完成人姓名：顾琰 学号：57119117 完成日期：2021 年 7 月 5 日

**实验目的：**

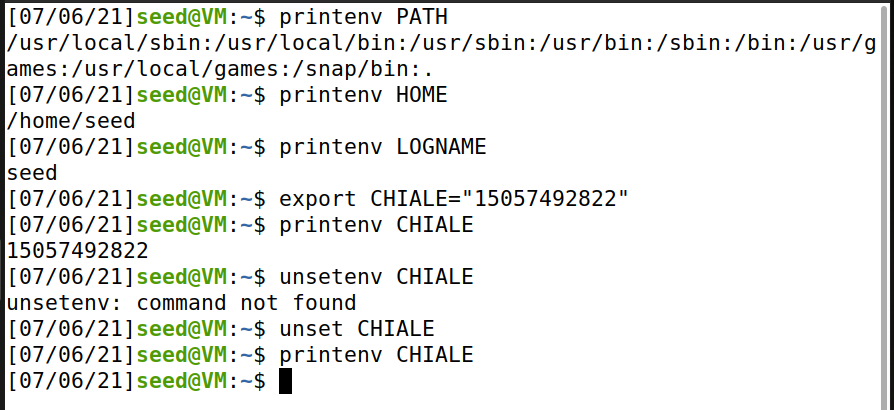
1. 理解环境变量的工作方式。

2. 探究环境变量怎么影响 Set-UID 程序。

**实验内容：**

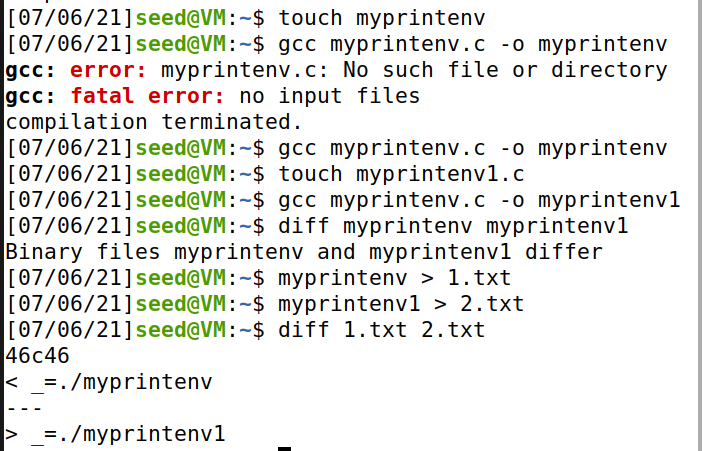
**Task 1: Manipulating Environment Variables**

使用export设置环境变量，使用printenv显示，使用unset删除环境变量。



**Task 2: Passing Environment Variables from Parent Process to Child Process**

把手册所给程序编译并执行，将其内容保存到文本文件中去，其内容为环境变量的值。

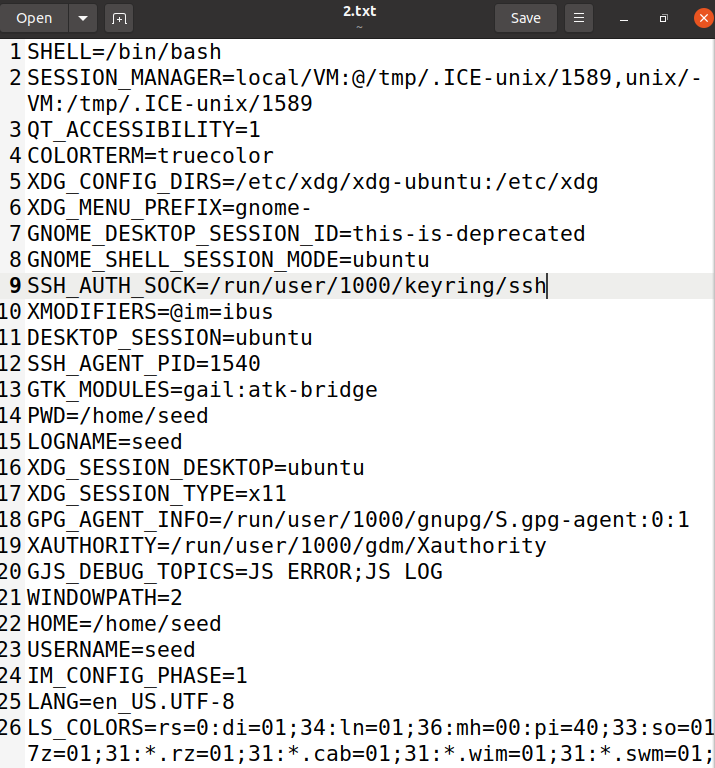


使用diff命令比较，无任何输出，表明两者的内容完全相同。

实验结果表明在Unix环境下使用fork函数，子进程环境变量会继承父进程环境变量。

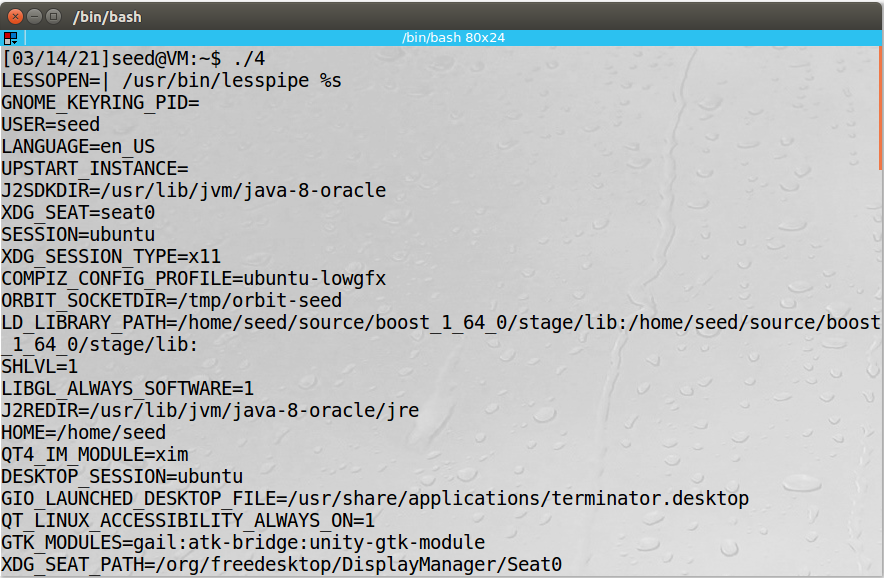
**Task 3: Environment Variables and execve()**

把手册所给程序编译并执行，分别输出到两个文本文件中，结果发现第一个执行结果为空，第二个执行结果为当前的环境变量。



参数2是利用指针数组来传递给执行文件，并且需要以空指针结束；参数3是传递给执行文件的新环境变量数组。成功不会返回，失败返回-1。所以当execve()函数的参数3:新的环境变量数组，设置为NULL的时候，打印信息为空；当参数3为外部环境变量的数组environ时，其传递给新的program，所以可以打印出环境变量信息。

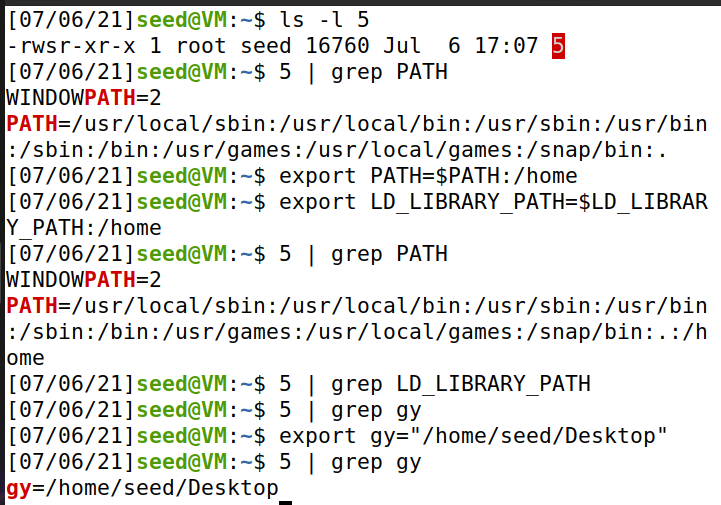
**Task 4: Environment Variables and system()**



通过printenv命令比较发现二者相同。

结论：通过system命令执行的新程序的环境变量与调用程序相同。

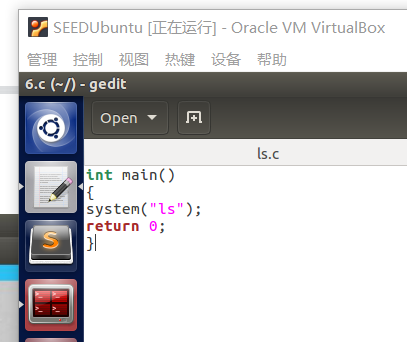
**Task 5: Environment Variable and Set-UID** **Programs**



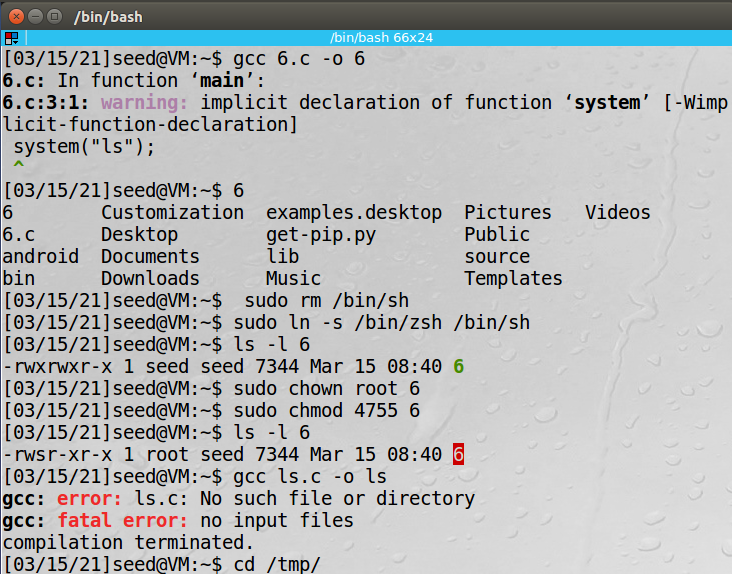
由结果可知，PATH变量中增加了新输入的环境变量，LD\_LIBRARY\_PATH变量没有包含在结果中，自定义的 gy 变量出现在结果中。经过我们修改的 PATH 与我们额外添加的 gy 环境变量都被 5 Set\_UID 程序所继承。

**Task 6: The PATH Environment Variable and Set-UID** **Programs 、**

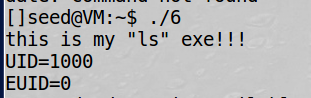
创建一个 Set-UID 的程序(命名为 system\_ls)来运行”ls”命令:



为防止 Set-UID 程序在 /bin/dash 内被重新恢复为 uid，我们将/bin/sh 链接到 zsh:



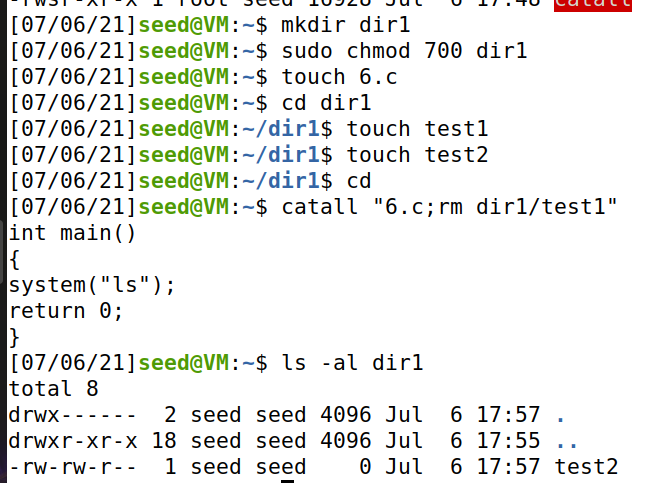
更改 path 环境变量为我们所写程序的目录:/tmp/



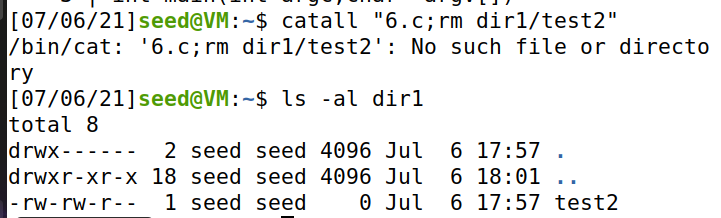
根据实验手册，由于dash的反制措施，所以先把shell软链接到zsh，然后更改环境变量，将shell复制到当前目录并设其名称为ls，然后重新运行6，从图中得知，已经运行了具有root权限的shell，所以set-uid程序所运行的子程序具有root权限。

**Task 8: Invoking External Programs Using system() versus execve()**

将所给程序编译，设置为set-uid程序，新建一个只有root有权限的文件夹dir1



在dir1中写入两个文件：test1和test2，在普通用户权限下使用task8删除test1。在没有root权限的情况下删除成功。接下来注释掉system(command)语句，使用execve（）来调用该命令。然后尝试按照同样的方法删除test2。



删除失败，原来的攻击方法已经失效。

结论：system函数会重新开启一个shell来执行用户的命令，在程序为set-uid程序时会造成巨大安全隐患；而execve函数则是把用户的输入作为参数，其具有更高的安全性。

**实验体会：**

本次实验是Set-UID程序漏洞实验，我总结如下:

①system()会调用 shell 执行命令，并且把全部的环境变量传给新进程。

②execve()不调用 shell 执行命令，环境变量通过其第三个参数传递给新进程(NULL 代

表新进程环境变量为空)。

③Set-UID 程序会将除了 LD\_\*的其余环境变量传递给新进程。

我学会了通过下述方法进行攻击:

①在已经提权的 system()中通过‘;’将我们自己的命令注入进去执行。

②通过改变环境变量使得已经提权的 system()执行我们自己写的不同路径的同名程序。

③利用提权程序未释放的变量对普通权限无法操作的文件进行操作。