**实验1 Environment\_Variable\_and\_SetUID**

**57117134-张家康**

目录

[Task 1: Manipulating environment variables 1](#_Toc49875841)

[Task 2: Passing Environment Variables from Parent Process to Child Process 2](#_Toc49875842)

[Task 3: Environment Variables and execve() 3](#_Toc49875843)

[Task 4: Environment Variables and system() 4](#_Toc49875844)

[Task 5: Environment Variable and Set-UID Programs 4](#_Toc49875845)

[Task 6: The PATH Environment Variable and Set-UID Programs 6](#_Toc49875846)

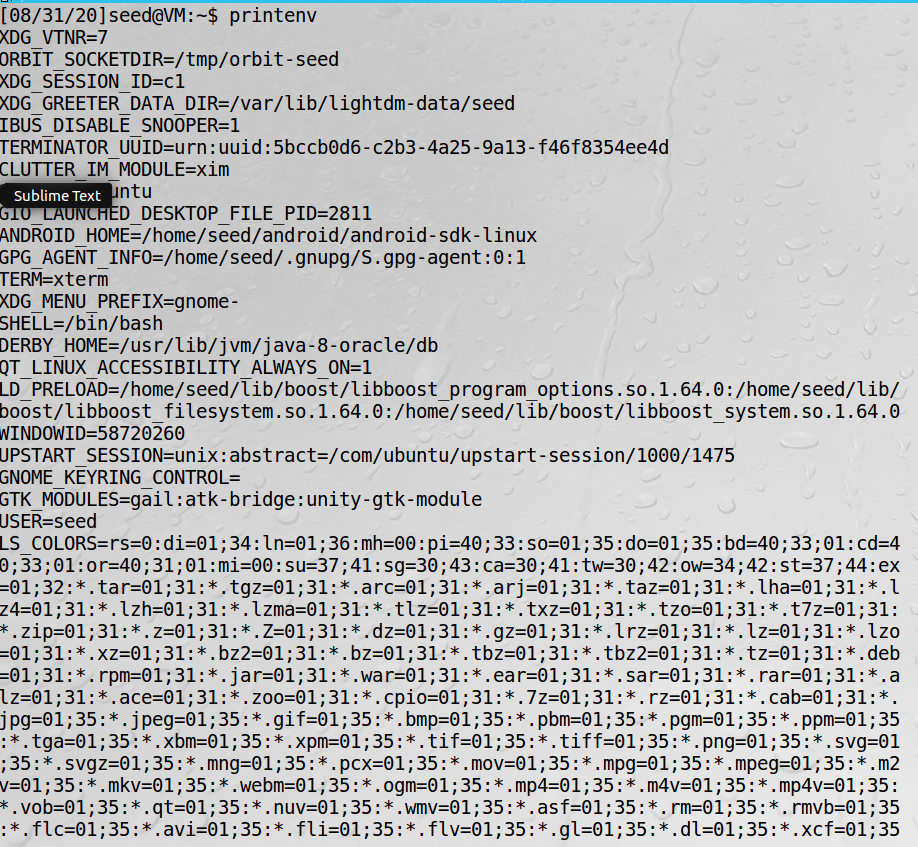
[Task 7: The LD PRELOAD Environment Variable and Set-UID Programs 6](#_Toc49875847)

[Task 8: Invoking External Programs Using system() versus execve() 7](#_Toc49875848)

[Task 9: Capability Leaking 8](#_Toc49875849)

## Task 1: Manipulating environment variables

1. 使用printenv命令输出环境变量，输入命令printenv，得到如下结果（只截取部分）：



使用printenv命令查看PWD的环境变量，输入命令：printenv PWD，得到如下结果：

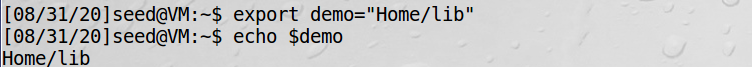


使用env命令查看PWD的环境变量，输入命令：env |grep PWD，得到如下结果：

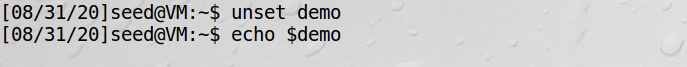


1. 使用export和unset设置或删除环境变量：

使用export设置环境变量，使用echo显示，$符号实际作用是将变量转换成字符，方便输出：

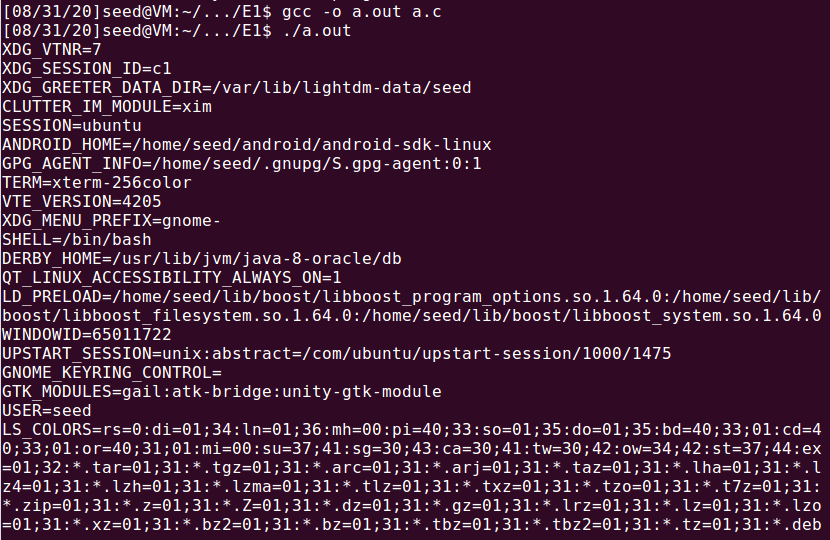


使用unset删除环境变量：

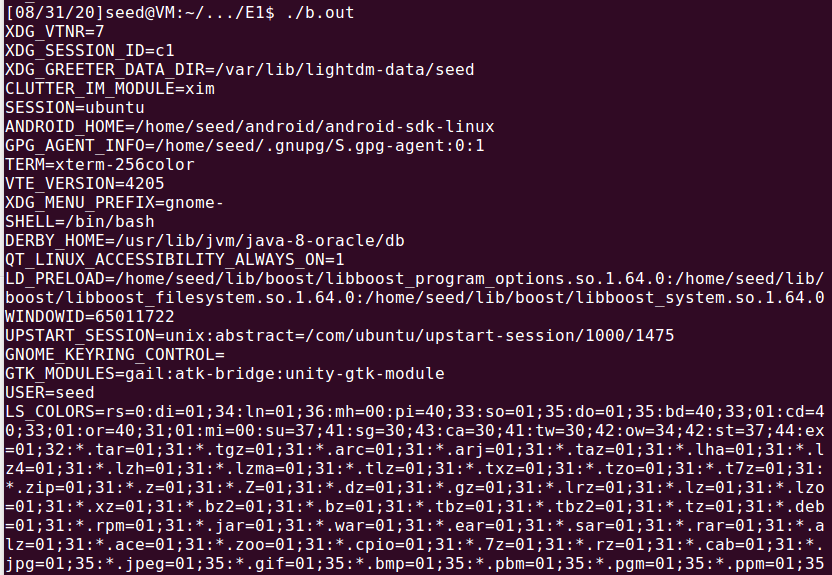


## Task 2: Passing Environment Variables from Parent Process to Child Process

1. 编译a.c文件，并运行a.out，获取环境变量的值，运行结果如下图所示（只截取部分）：

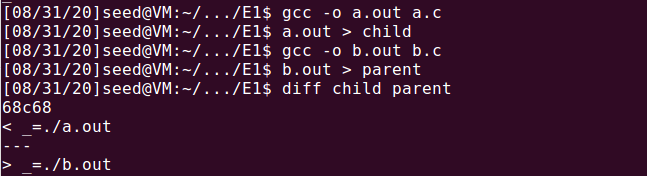


1. 将child process中printenv()注释，将process中parent printenv()取消注释，重新保存为b.c，重新编译并运行b.out，得到结果如下：



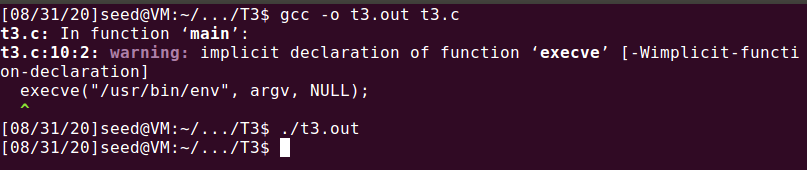
1. 比较两者结果：

将a.out、b.out文件的结果分别保存为child和parent文件，再使用diff命令比较，发现两者除了文件名外完全相同。这说明子进程环境变量会继承父环境变量。结果如图所示：

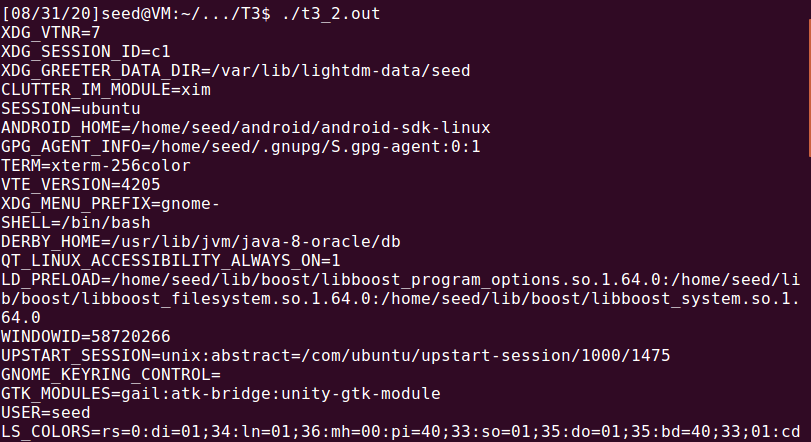


## Task 3: Environment Variables and execve()

1. 编译t3.c，并运行t3.out，输出结果为空，如图所示：



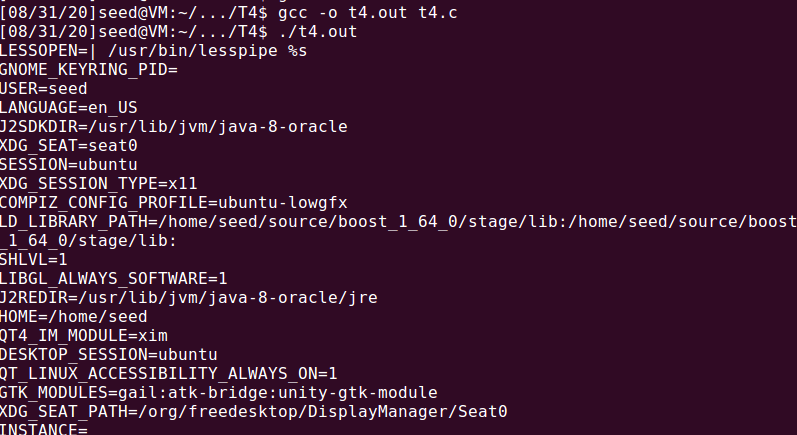
1. 将原语句换为：execve("/usr/bin/env", argv, environ); 重新保存为t3\_2.c并编译为t3\_3.out，运行后得到的结果如下图所示：



1. 从以上实验可以看出，execve()产生的新进程的环境变量又调用时重新赋予，而fork()则是直接继承父进程环境变量。

## Task 4: Environment Variables and system()

1. 编译t4.c文件，并运行t4.out，输出结果如下图所示：



1. 分析：

system函数定义为 int system（const char \* string），该函数调用/bin/sh来执行参数指定的命令，/bin/sh一般是一个软连接，指向某个具体的shell，比如bash，-c 选项是告诉shell从字符串command中读取命令；在该command执行期间，SIGCHLD信号会被暂时搁置，SIGINT和SIGQUIT则会被忽略，意思是进程收到这两个信号后没有任何动作。system（）函数的函数返回值有些复杂。为了更好地理解system（）函数的返回值，需要了解其执行过程，实际上system()函数执行了三步操作：

* 1. fork一个子进程；
  2. 在子进程中调用exec函数去执行command；
  3. 在父进程中调用wait去等待子进程结束。

如果fork失败，system()函数返回-1。

如果exec执行成功，也即command顺利执行完毕，则返回command通过exit或return返回的值。

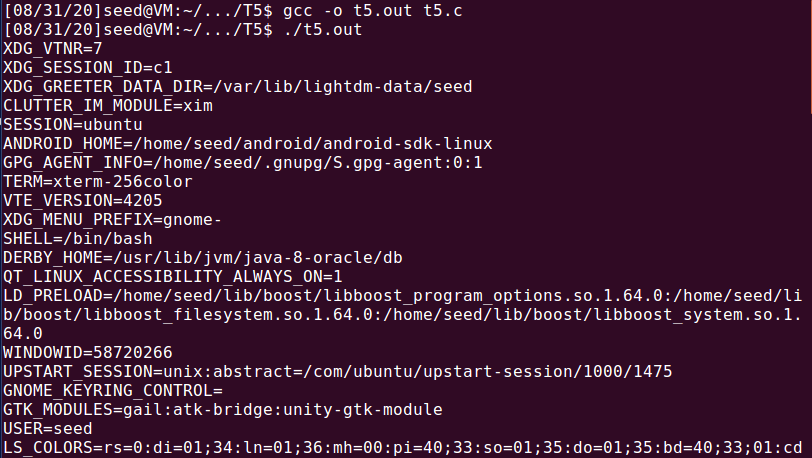
如果exec执行失败，也即command没有顺利执行，比如信号被中断，或者command命令根本不存在，system()函数返回127。

如果command为NULL，则system()函数返回值非0，一般为1。

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

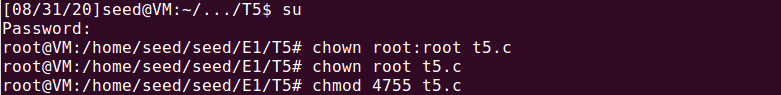
1. 在当前进程中打印出所有的环境变量：

保存并编译t5.c，并运行t5.out，得到的结果如下图所示：



1. 将上述程序的所有权改为root，并使它成为一个Set-UID程序：

先切换为root账户，使用chown root:root t5.c将此c文件权限改为root权限



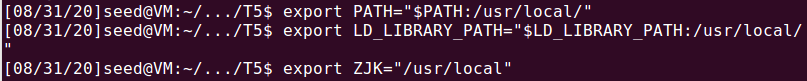
1. 使用一般用户登录终端，使用export命令设置如下环境变量：PATH 、LD\_LIBRARY\_PATH、ANY\_NAME：

export PATH="$PATH:/usr/local/"

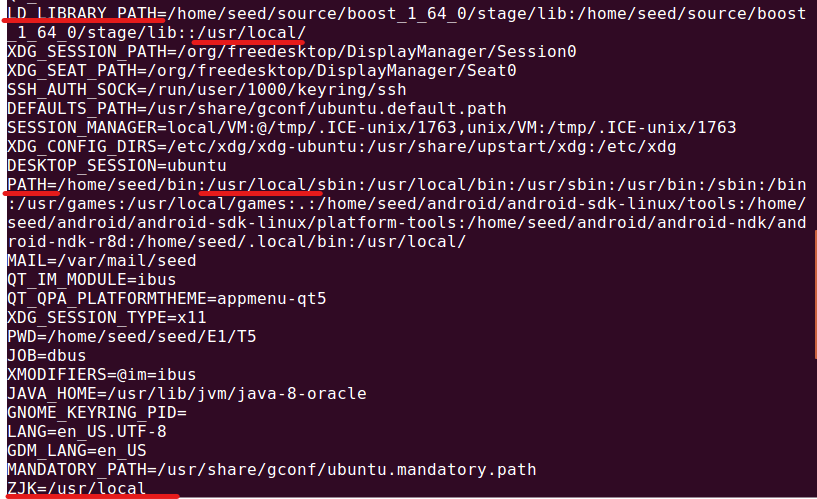
export LD\_LIBRARY\_PATH="$LD\_LIBRARY\_PATH:/usr/local/"

export ZJK="/usr/local"

如下图所示：



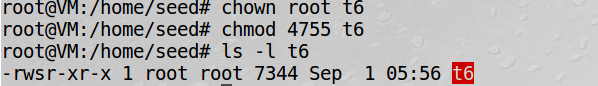
1. 执行已经赋予Set-UID的t5.c程序，得到如下结果：



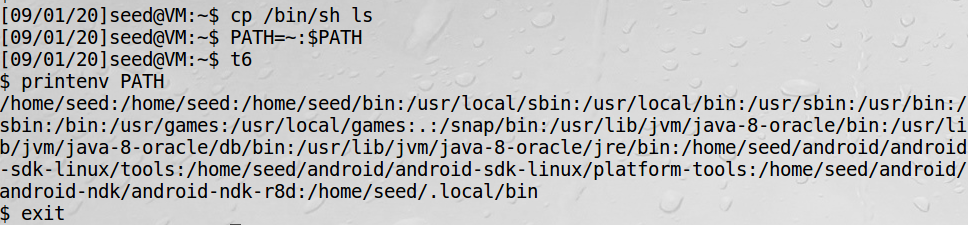
由上图可见，以上三个被定义的环境变量全部被包括在shell中。

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

保存代码为t6.c文件，切换为root用户，将其编译为t6.out，并设置其所有者为root，赋予SUID特殊权限。使用ls -l t6语句查看文件的权限，验证操作确实成功完成。如下图所示：



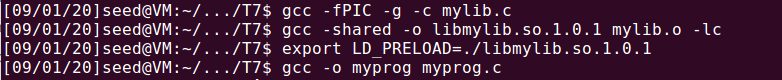
将/bin/sh复制到程序当前目录，命令为ls，设置环境变量，并运行t6，结果如下图所示：



当前目录我们自己编造了一个ls，所以程序就会直接执行伪造的ls。sh原本的作用是创建一个新shell，在执行此命令后我们就会一直停留在子进程中，知道我们主动退出这个程序，我们才会回到原来的权限

## Task 7: The LD PRELOAD Environment Variable and Set-UID Programs

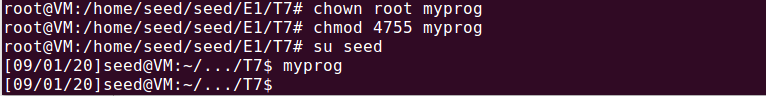
编译mylib.c，设置LD\_PRELOAD环境变量，编译myprog程序，在链接库libmylib.so.1.0.1的相同目录下：



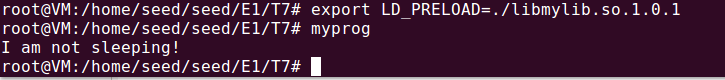
1. 以普通用户运行myprog程序，得到的结果如下图所示：



1. 以普通用户运行拥有root权限的myprog程序，无输出，如下图所示：



1. 以root用户更改环境变量，并运行，输出结果如下图所示：



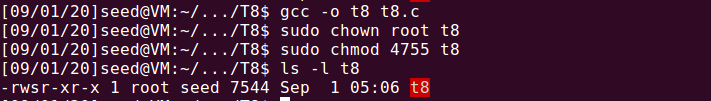
1. 使myprog成为一个Set-UID user1程序，在user2用户（非root用户）中再次设置LD\_PRELOAD环境变量，运行myprog程序，输出结果如下图所示：



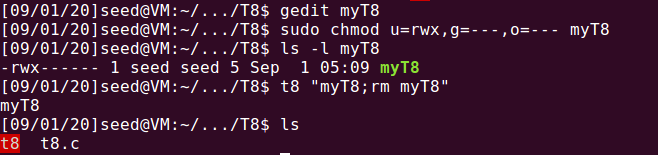
导致以上四次试验不同的原因就在于LD\_PRELOAD环境变量。LD\_PRELOAD环境变量是Unix动态链接库的世界中的一个环境变量，它可以影响程序的运行时的链接，它允许你定义在程序运行前优先加载的动态链接库。这个功能主要是用来有选择性的载入不同动态链接库中的相同函数。在该实验中，mylib.c通过sleep函数，生成了一个libmylib.so.1.0.1链接库。然后将该链接库添加到LD\_PRELOAD环境变量上。

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

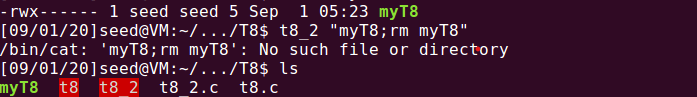
1. 保存t8.c，并编译为t8，设置其为Set-UID程序：



新建一个名为MY的文件，并设置其权限为仅root用户可读、写、执行。执行t8，输入命令“t8 “myT8;rm myT8”，运行后发现文件myT8被删除。如下图所示：

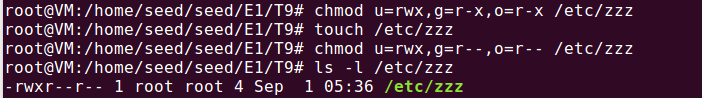


1. 注释掉system(command)语句，该程序将使用execve()来调用该命令。重复步骤1，发现步骤1中的攻击方式已经失效。



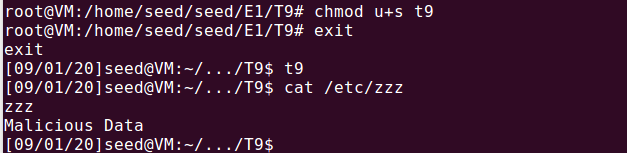
## Task 9: Capability Leaking

在root用户权限下，创建/etc/zzz，设置其为只有root用户可以写入。



保存代码为t9.c文件，将其编译为t9，并赋予SUID特殊权限。

执行t9，可以查看/etc/zzz文件，并没有权限限制。



分析其原因是因为其在取消权限前并没有关闭文件，导致seed用户仍然可以进行root用户才可以执行的写入操作。