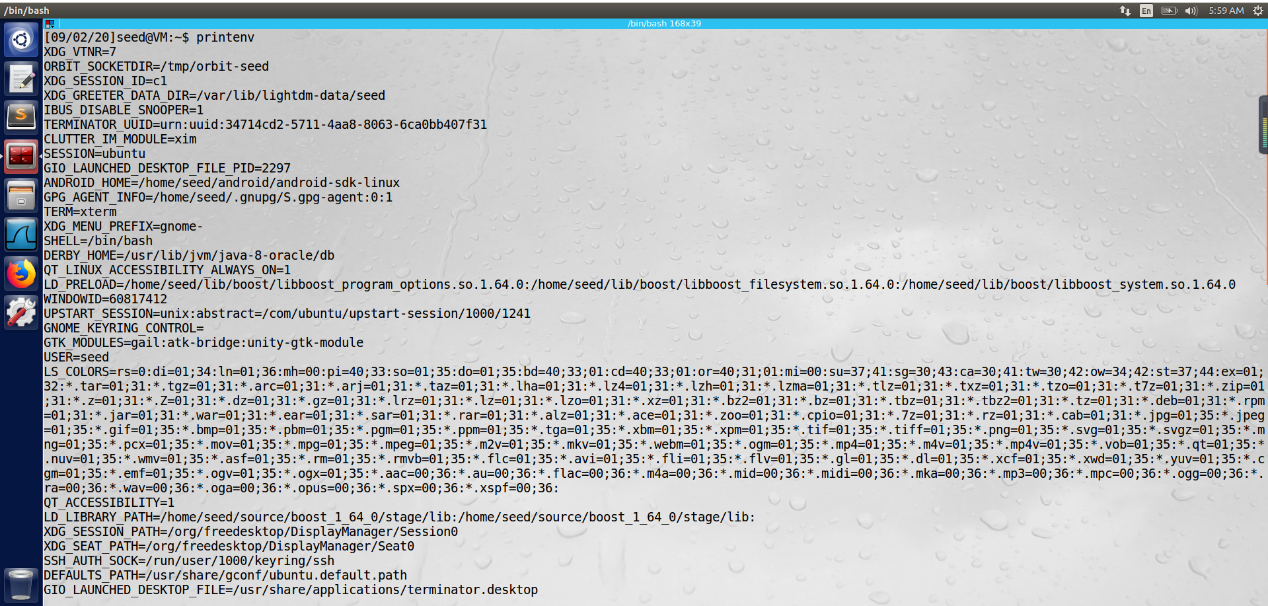
**网络安全综合课程设计实验1**

57117204 姚晓

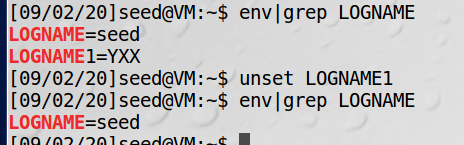
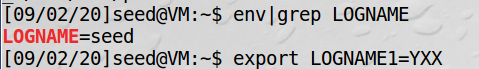
2020/9/3

**Task1:设置和取消设置环境变量**

用printenv命令打印环境变量:



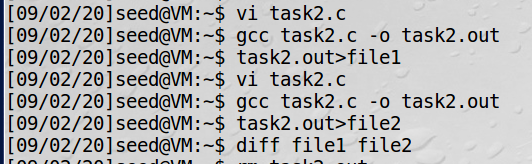
用env|grep命令查看环境变量LOGNAME，发现只有一个LONAME=seed；用export命令设置一个环境变量LOGNAME1=YXX，再使用env|grep命令查看环境变量LOGNAME，发现多了自己设置的新的环境变量；再用unset命令取消设置LOGNAME1，用env|grep命令查看环境变量LOGNAME，发现只有一个LONAME=seed，自己设置的LOGNAME1没有了：

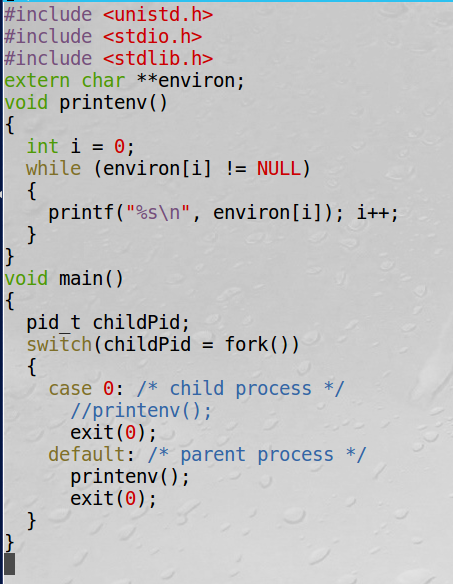


**Task2:父进程的环境变量是否被子进程继承**

编译实验手册的代码（命名为task2.c），将可执行文件（task2.out）的输出保存到file1当中，此时file1当中的数据为子进程的环境变量；然后修改代码为图2所示，编译并运行，将输出保存到file2当中，此时file2当中的数据为父进程的环境变量。

最后用diff命令比较file1和file2的内容，发现没有区别，即父进程的环境变量被子进程所继承：



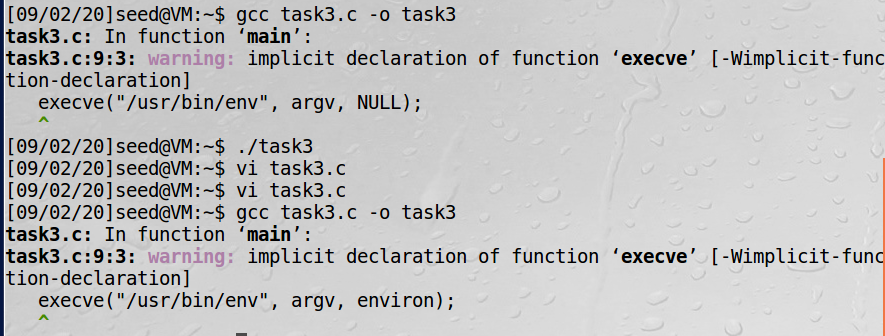


**Task3:用execve()执行新程序，环境变量是否被新程序自动继承**

　编译并运行实验手册上的代码（命名为task3.c），可执行文件为task3，该程序仅执行一个名为/ usr / bin / env的程序，该程序将输出当前进程的环境变量，此时并没有任何环境变量，execve()的第三个参数为NULL，环境变量不被继承；

　然后修改task3.c里execve("/usr/bin/env", argv, NULL)为execve("/usr/bin/env", argv, environ)，再次运行task3，发现输出了许多环境变量，此时execve()的第三个参数为environ，新程序继承了环境变量；

所以得出结论，环境变量不会被新程序自动继承，只有填入特定参数才可以。

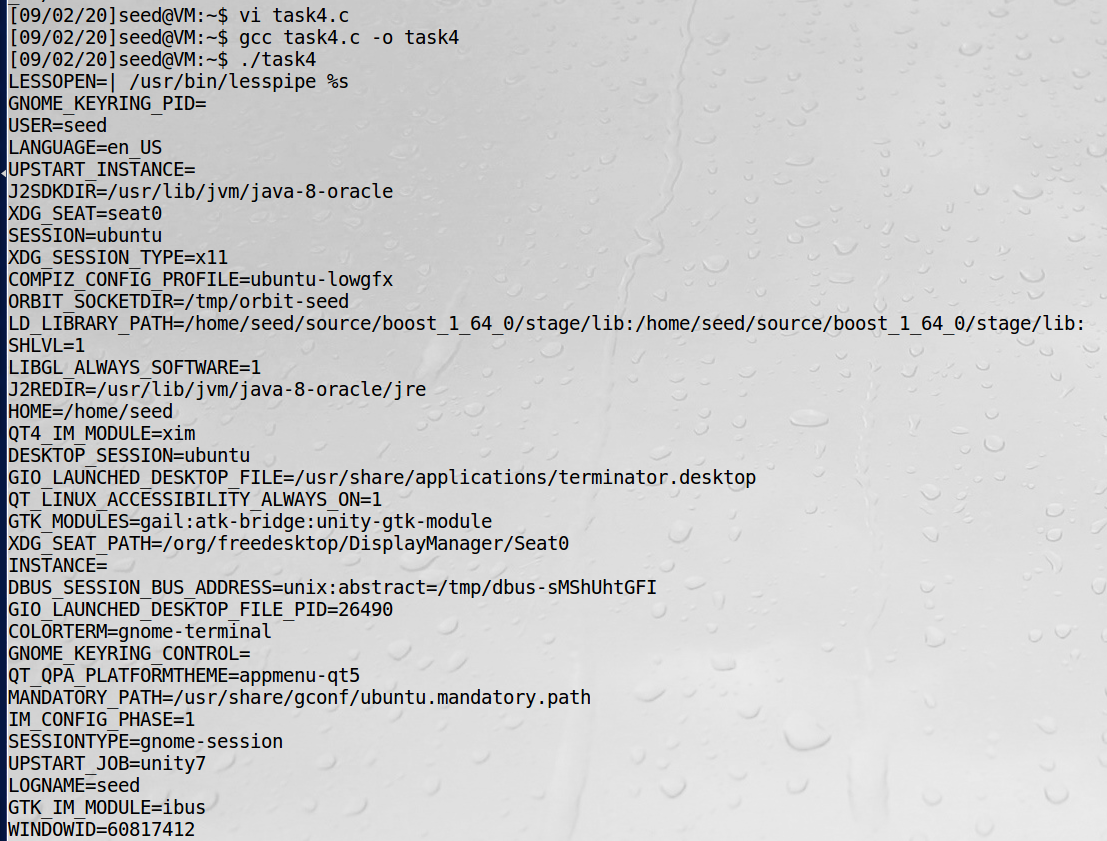




**Task4:** **system（）函数执行新程序时环境变量**

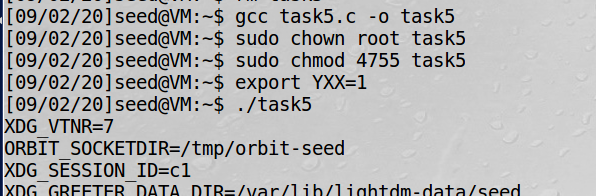
system（）实际上执行“ / bin / sh -c命令”，即，它执行/ bin / sh，并要求shell执行 执行命令。 如果查看system（）函数的实现，您会发现它使用execl（）执行/ bin / sh; execl（）调用execve（），将环境变量数组传递给它。 因此，使用system（），将调用过程的环境变量传递到新程序/ bin / sh。

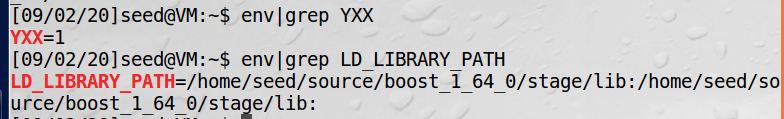
编译并运行实验手册的代码（命名为task4.c），得到以下结果，证明了上述说法：



**Task5:Set-UID程序是否继承用户进程的环境变量**

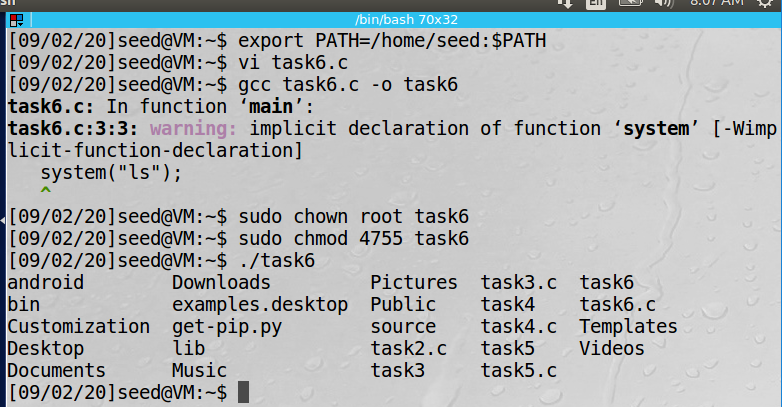
编译并运行实验手册的代码（命名为task5.c），将task5程序设置为Set-UID程序，在我的shell里利用export函数设置一个环境变量YXX=1（没有设置LD\_LIBRARY\_PATH因为已经有了），然后运行task5，用env|grep命令查看YXX和LD\_LIBRARY\_PATH环境变量，能查看到，说明set-UID程序进程会继承用户进程的环境变量。



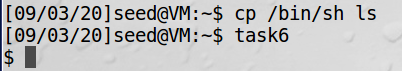


**Task6:PATH环境变量和Set-UID程序**

首先用export命令设置PATH环境变量，然后编译并运行实验手册的代码（命名为task6.c），将task6程序所有者设为root并设置为Set-UID程序：

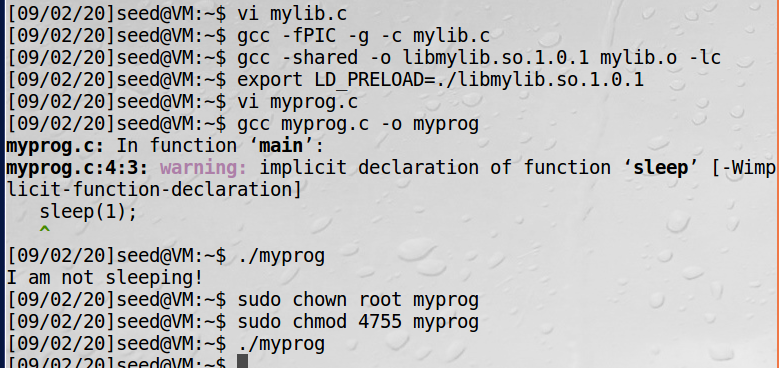


将/bin/sh复制到当前目录并命名为ls，执行task6就可以得到root权限。

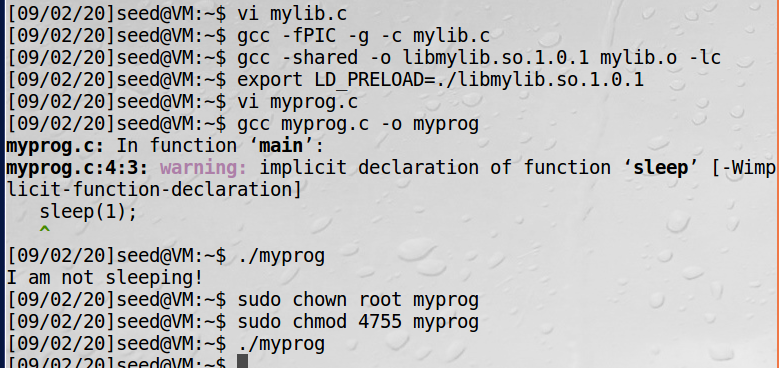


**Task7: LD PRELOAD环境变量Set-UID程序**

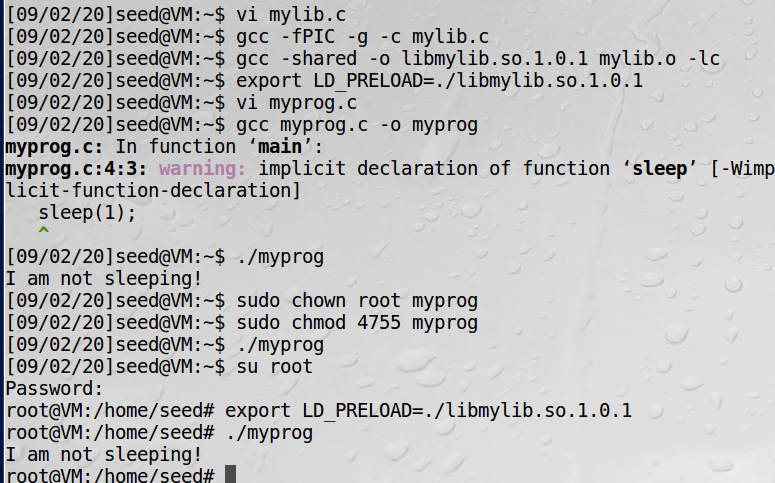
建立一个动态链接库。 创建以下程序，并将其命名为mylib.c。 从根本上覆盖libc中的sleep（）函数，用实验手册里的命令编译该程序，用export命令设置LD\_PRELOAD函数；然后编译实验手册上的myprog.c程序，并在与动态链接库libmylib.so.1.0.1相同的目录下；



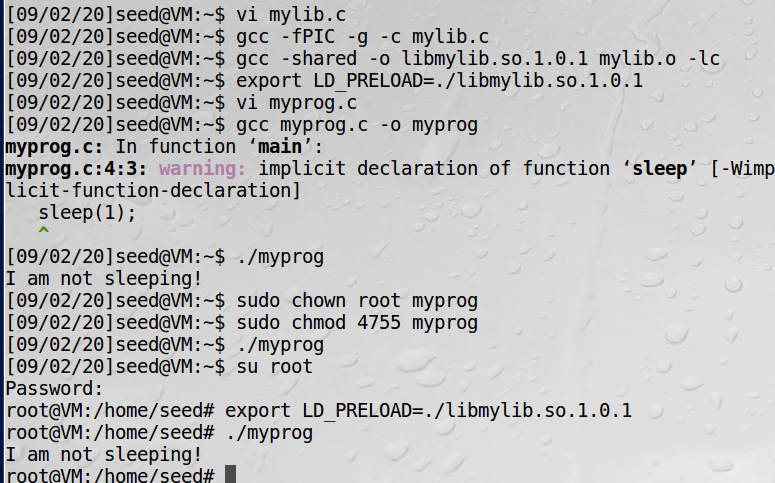
1. 使myprog成为常规程序，然后以普通用户身份运行它，运行的是自己设置的sleep函数：



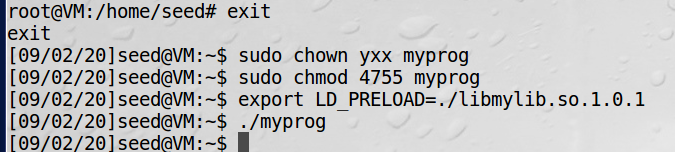
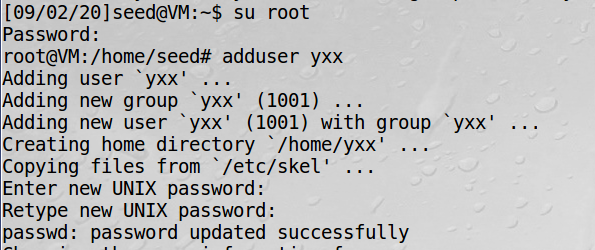
1. 将myprog设置为Set-UID根程序，然后以普通用户身份运行它，并不能运行自己设置的sleep函数，因为没有root权限，运行的是libc中的sleep()函数：



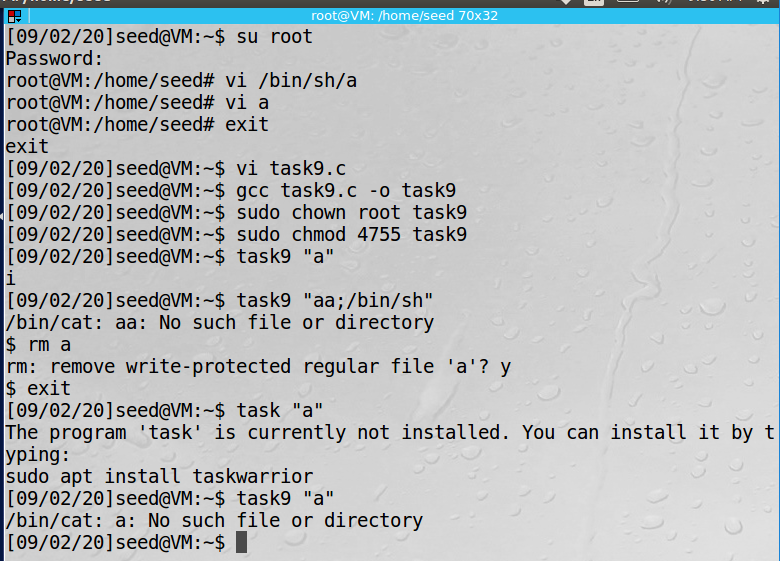
3、使myprog成为Set-UID根程序，再次在根帐户中导出LD PRELOAD环境变量并运行它，可以运行自己设置的sleep函数：



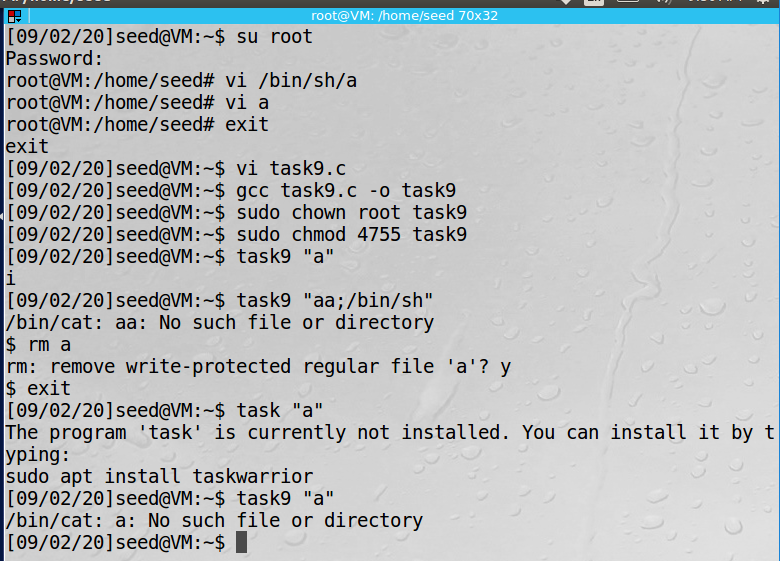
4、添加一个用户yxx，将myprog设置为Set-UID yxx程序，即所有者为yxx，再次在用户seed的帐户中导出LD PRELOAD环境变量并运行它，不能运行自己的sleep函数，因为没有yxx用户的权限。



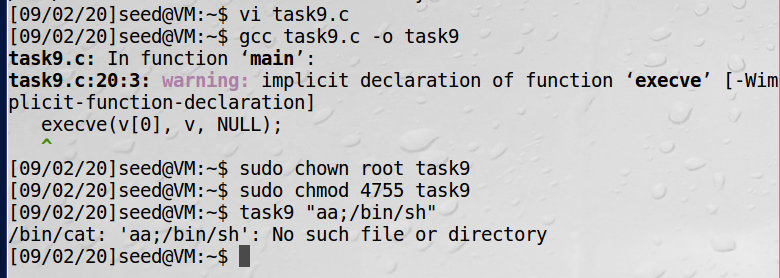
**Task8:** **使用system（）与execve（）调用外部程序**

首先用root账户在/bin/sh文件夹内建一个新文件a：

编译实验手册的代码并命名为task9(命名错误没有改)，此时该程序利用system()调用外部程序，将task9所有者设置为root，并设为Set-UID程序，用task9 ”a”能看到当前路径下的a文件，然后用task9 “aa;/bin/sh”，会报错没有aa文件，并且拥有了root权限，此时就可以用rm命令删除a文件，删除后退出，再用task9查看发现没有该文件了即删除成功：



按实验手册修改task9.c的代码，此时该程序利用execve()调用外部程序，将task9所有者设置为root，并设为Set-UID程序，然后用task9 “aa;/bin/sh”，会报错没有aa文件，不能拥有root权限。



**Task9: Capability Leaking**

撤消特权时，常见的错误之一是功能泄漏。当该进程仍然具有特权时，它可能已经获得了一些特权。当特权降级时，如果程序没有清除这些功能，则非特权进程仍可以访问它们。换句话说，尽管该进程的有效用户ID变为非特权，但该进程仍具有特权，因为它具有特权功能。编译实验手册里的程序并命名为task10，将其所有者更改为root，并使其成为Set-UID程序。用root账户新建一个文件/etc/zzz，里面内容为yxxxxxxxxxxx，以普通用户身份运行task9程序，发现被插入了malicious data。

