**Lab1 Report**

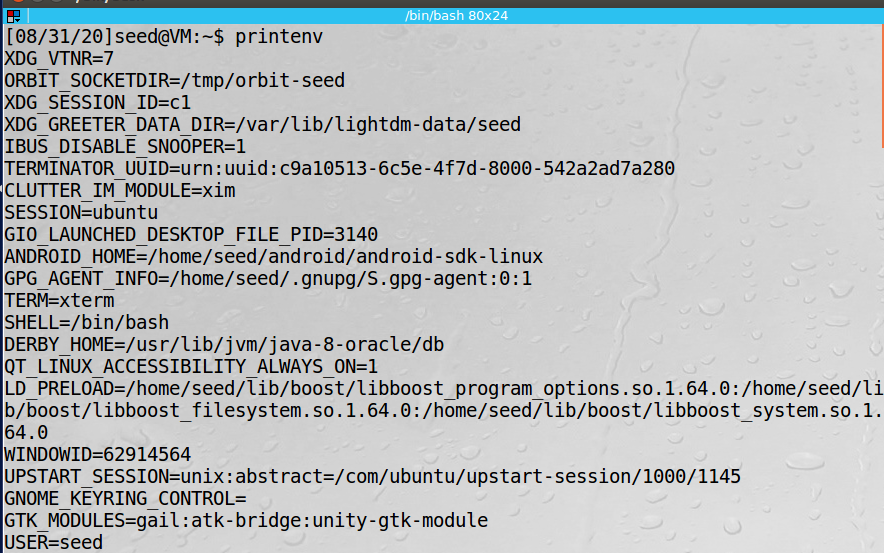
**Environment Variable and Set-UID Program Lab**

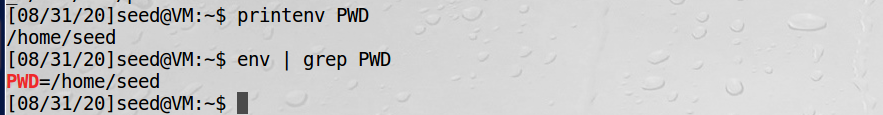
57117107 赵笛言

**LabTasks：**

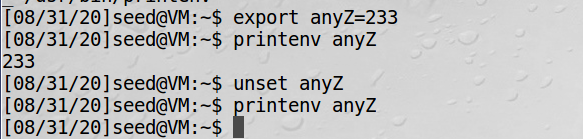
Task1: Manipulating Environment Variables

Use printenv or env command to print out the environment variables.



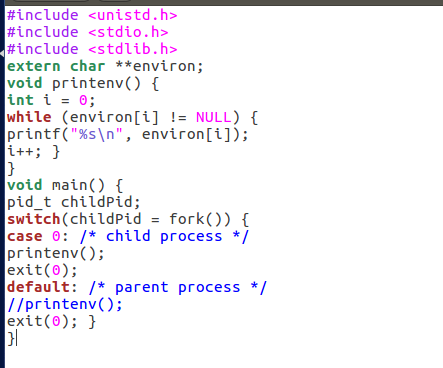


Use export and unset to set or unset environment variables.



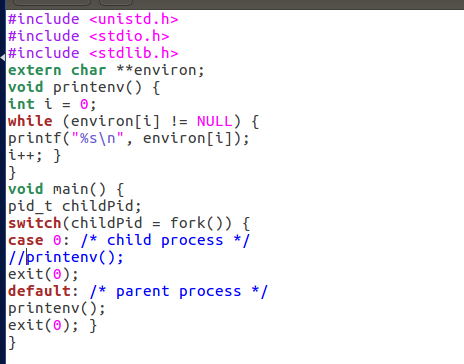
Task2: Passing Environment Variables from Parent Process to Child Process

Step1. Please compile and run the following program ,and describe your observation. B save the output into a file, such as using a.out > child.



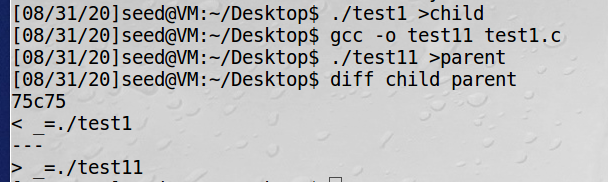
该程序展示了某父进程的子进程的环境变量。这些环境变量输出到文件child中。

Step2. Now comment out the printenv() statement in the child process case , and uncomment the printenv() statement in the parent process case. Compile and run the code again, and describe your observation. Save the output in another file.



将父进程的环境变量存储到文件parent中。

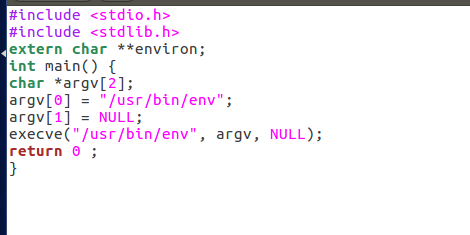
Step3.Compare the difference of these two files using the diff command. Please draw your conclusion.



可以看出除了个人操作中导致的编译文件的不同，其余环境变量均相同。因此可以得出结论，子进程继承了父进程的环境变量。

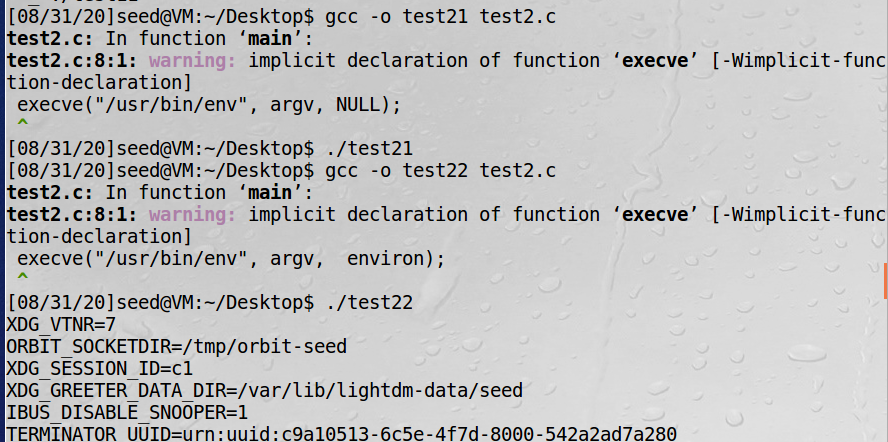
Task3: Environment Variables and execve()

Step1. Please compile and run the following program, and describe your observation. This program simply executes a program called /usr/bin/env, which prints out the environment variables of the current process.

此时输出为空。

Step2. Change the invocation of execve() to the following; describe your observation. execve("/usr/bin/env", argv, environ);





此时输出了环境变量值。

Step3. Please draw your conclusion regarding how the new program gets its environment variables.

对比可得，当execve（）函数第三个参数为null，输出环境变量为空。而当参数为environ，则继承父进程的环境变量。可知execve () 产生的新进程的环境变量在调用时重新赋予。并不类似fork()直接继承父进程环境变量。

Task4: Environment Variables and system()

Using system() ,the environment variables of the calling process is passed to the new program /bin/sh. Please compile and run the following program to verify this.



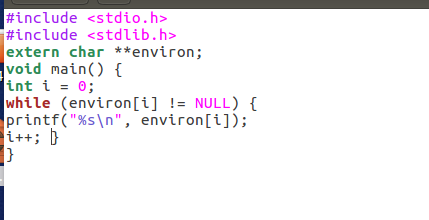
函数定义为int system (const char \* string); system() 会调用fork () 产生子进程，由子进程来调用/bin/sh-c string来执行参数string字符串所代表的命令，此命>令执行完后随即返回原调用的进程。system（）具体运行可分为三个步骤：调用fork（）函数新建一个子进程；在子进程中调用exec函数去执行需求；在父进程中调用wait去等待子进程结束。

Task5: Environment Variable and Set-UID Programs

To understand how Set-UID programs are affected, let us first figure out whether environment variables are inherited by the Set-UID program’s process from the user’s process.

Step1. Write the following program that can print out all the environment

variables in the current process.



Step2. Compile the above program, change its ownership to root, and make it a

Set-UID program.



Step 3. In your shell (you need to be in a normal user account, not the root

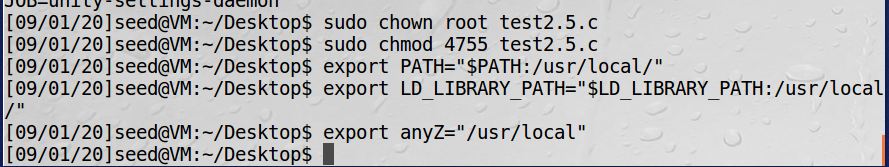
account), use the export command to set the following environment variables

(they may have already exist):

• PATH

• LD LIBRARY PATH

• ANY NAME (this is an environment variable defined by you, so pick whatever name you want).



These environment variables are set in the user’s shell process. Now, run the

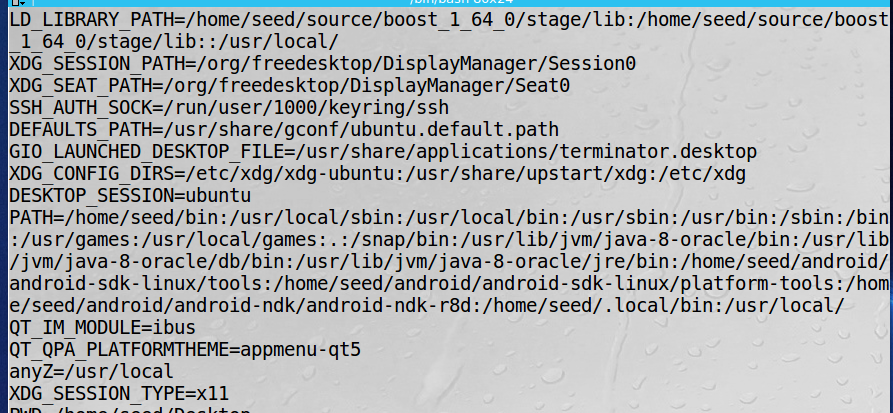
SetUID program from Step 2 in your shell. After you type the name of the program

in your shell, the shell forks a child process, and uses the child process to run the

program. Please check whether all the environment variables you set in the shell

process (parent) get into the Set-UID child process. Describe your observation. If

there are surprises to you, describe them.



三个被定义的环境变量全部被包括在shell中。可以看出我们利用SetUid，使普通权限的使用者获得了修改shell的能力。

Task6: The PATH Environment Variable and Set-UID Programs

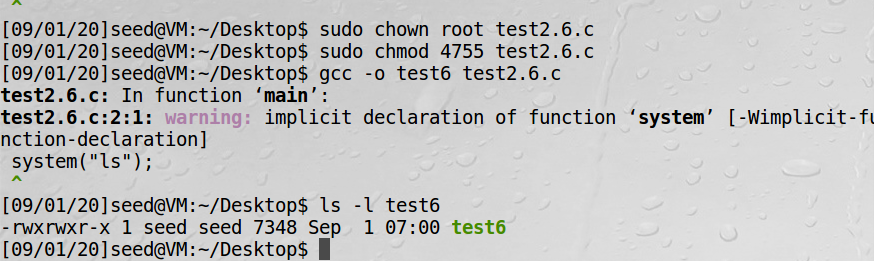
In Bash, you can change the PATH environment variable in the following way：

$ export PATH=/home/seed:$PATH

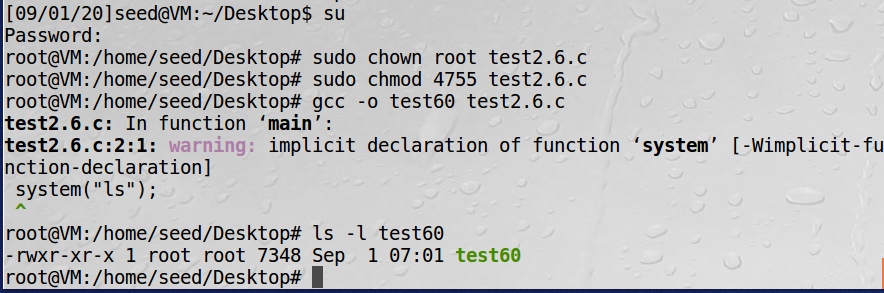
The Set-UID program is supposed to execute the /bin/ls command; however, the programmer only uses the relative path for the ls command, rather than the absolute path；Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.



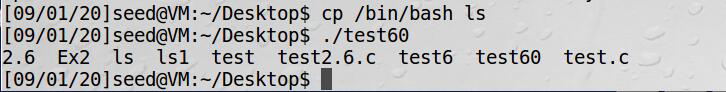
在普通用户权限下，无法将该程序的拥有者直接改至root，如下图所示：

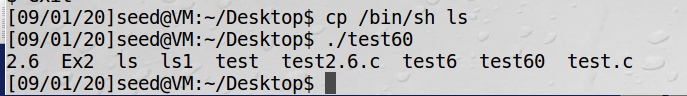


而在root模式下可以更改：

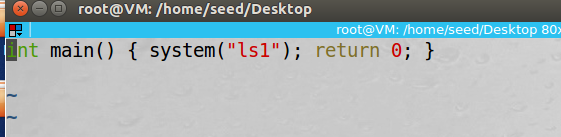


接下来利用cp伪造ls是，我采用的以下俩种方式都没有办法实现：

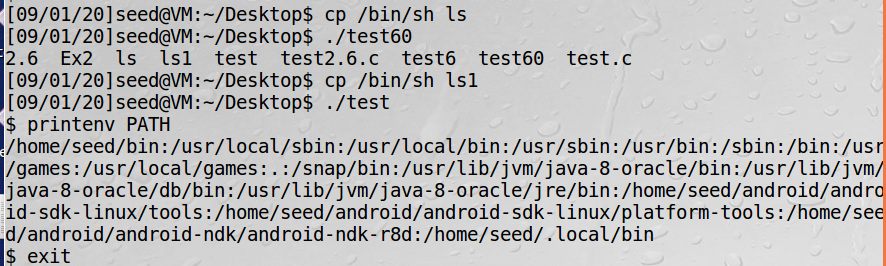




程序仍然实现ls的展示目录功能。为了证明实验可行性，设置了对照组：



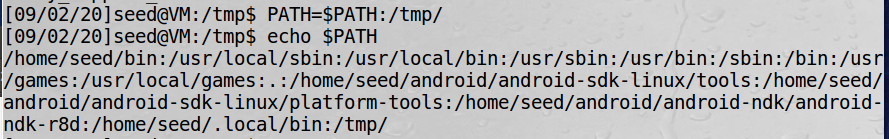
此处无法用以伪造ls以达成攻击目的，但可以验证该操作的可行性，如下：

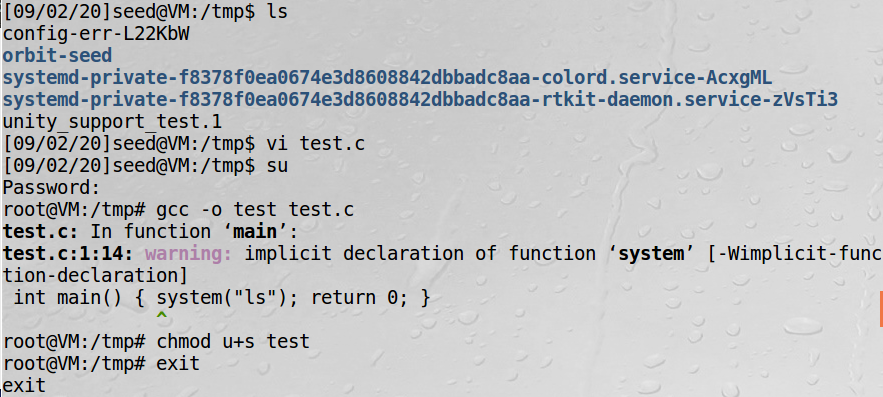


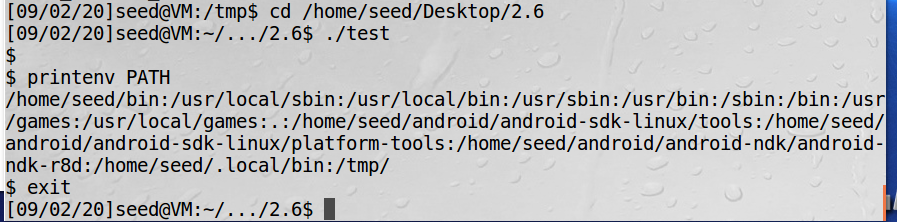
表面上命令为ls1，实际上执行了/bin/sh创建了一个新的shell。

在网上查阅资料后，尝试发现如下操作可以实现伪造ls：

需将/bin/sh 复制至tmp目录下并取名为ls才可以实现伪造，创建新的shell



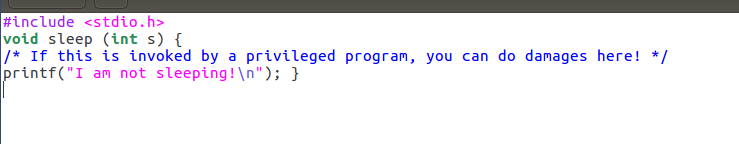


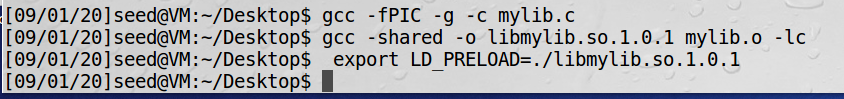


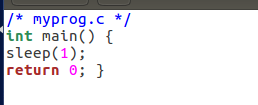
Task7: The LD PRELOAD Environment Variable and Set-UID Programs

Step1. First, we will see how these environment variables influence the behavior of dynamic loader / linker when running a normal program.

Build a dynamic link library；compile the above program using the following commands；set the LD PRELOAD environment variable；compile the following program myprog, and in the same directory as the above dynamic link library libmylib.so.1.0.1:



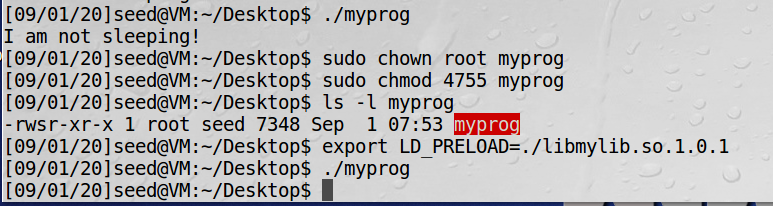




Step 2. After you have done the above, please run myprog under the following conditions, and observe what happens.

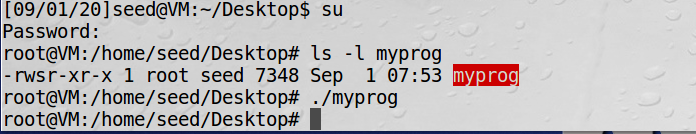
• Make myprog a regular program, and run it as a normal user.

• Make myprog a Set-UID root program, and run it as a normal user.

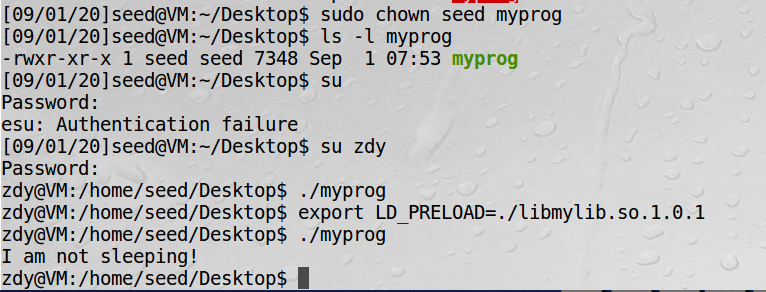


以上俩种执行情况如图所示，第一种输出“I am not sleeping！”，后者等待一秒无输出。

• Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.



• Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD PRELOAD environment variable again in a different user’s account (not-root user) and run it.

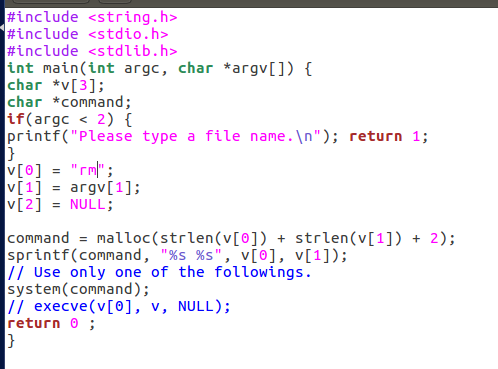


LD\_PRELOAD环境变量是Unix动态链接库的世界中的一个环境变量，通过该变量可以定义在程序运行前优先加载的动态链接库。在该实验中，mylib.c通过定义了一个假的sleep函数，生成了一个libmylib.so.1.0.1链接库。然后将该链接库添加到LD\_PRELOAD环境变量上。当程序执行时调用了这个链接库，则会执行mylib.c即输出字符串而不是停顿一秒。

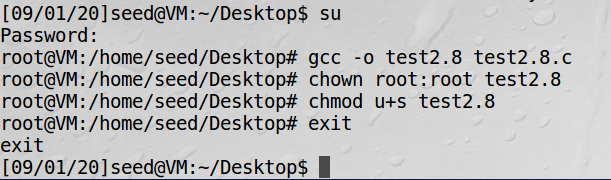
只有当加载了这个自创库以及是用户自己创建的程序自己去运行时，才会使用LD\_PRELOAD环境变量，将会重载sleep函数，否则的话忽略LD\_PRELOAD环境变量，不会重载sleep函数，即不会输出字符串。

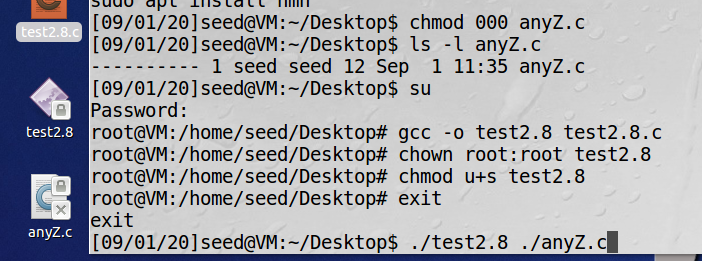
Task8: Invoking External Programs Using system() versus execve()

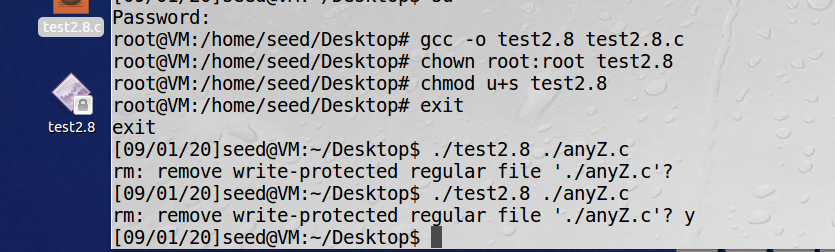
Step 1: Compile the above program, make it a root-owned Set-UID program. The program will use system() to invoke the command. If you were Bob,can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?



将手册中所给的程序v[0]改为”rm”,再对其赋予root权限：

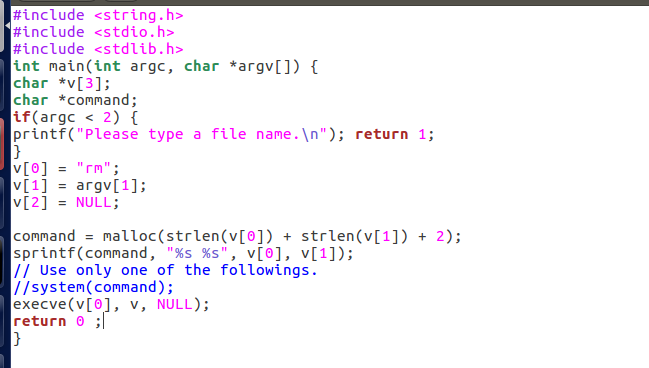


随便创建一个文件anyZ.c，并将其权限设置为000（不可读不可写）：  


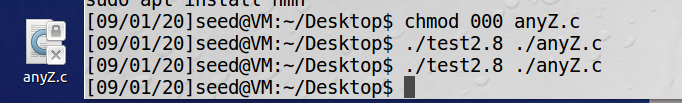


可以看出，虽然该文件对seed用户不可读不可写，但依然通过test2.8程序将其成功删除了。

Step 2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it a root-owned Set-UID. Do your attacks in Step 1 still work? Please describe and explain your observations.



改变test2.8.c并同样赋予root权限后，再进行同样的攻击，此时：

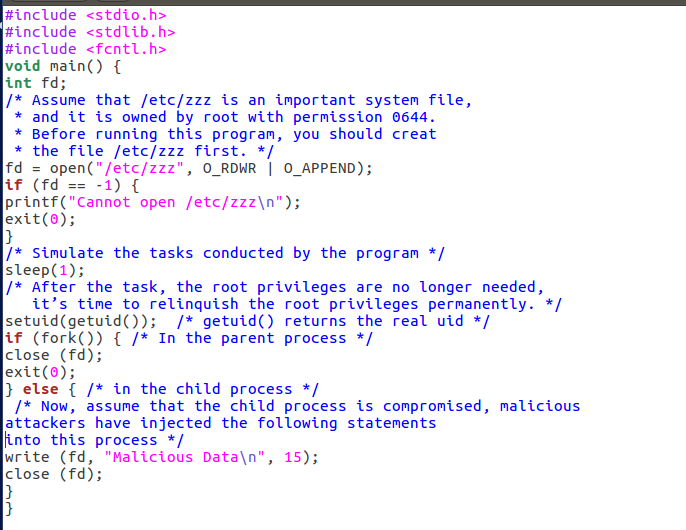


无法删除。

可以得知，system（）函数在用户输入时存在安全隐患，很容易被利用以获得更高的权限来对文件等进行操作，而execve（）函数对输入的识别更准确，有更高的安全性。

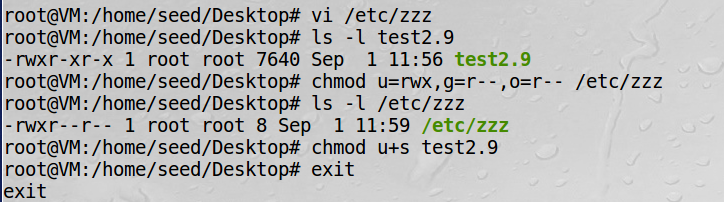
Task9: Capability Leaking

Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user, and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observation.

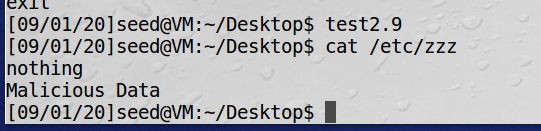


程序如图，对其设置root权限。

再创建/etc/zzz文件，（里面写入nothing），然后将其设置为仅root用户可写入：



此时在seed用户普通权限下运行程序，可以看见成功写入：



可以知道，在取消权限前并没有关闭文件，会导致seed用户可以借机进行root权限下的写操作。造成了权限泄露。