哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2018 年秋季学期
计算机学院大三
计算机系统安全课程

Lab 1 实验报告

姓名	李国建
学号	1160300426
班号	1603107
电子邮件	13144602916@163.com
手机号码	13144602916

实验内容：

1. 设想一种场景需要进行普通用户和 root 用户切换，设计程序实现 euid 的安全管理。配合第 3 章 完成进程中 euid 的切换，实现 root 权限临时性和永久性管理，加强程序的安全性。
2. 搭建安全的沙盒环境，在沙盒环境中提供必须的常见工具，并提供程序验证沙盒环境的安全性。配合第 3 章 实现系统中的虚拟化限制方法，实现安全的系统加固，测试虚拟化空间的加固程度。

实验过程：

第一部分：

1.1 设计并实现不同用户对不同类文件的 r、w、x 权限:

（1） 查看系统文件的权限设置

（a） 查看/etc/passwd 文件和/usr/bin/passwd 文件的权限设置，如图所示:

```
root@rocket:/home/rocketeerli# ls -al /etc/passwd /usr/bin/passwd
-rw-r--r-- 1 root root  2650 12月  4 19:56 /etc/passwd
-rwsr-xr-x 1 root root 59640 1月  25  2018 /usr/bin/passwd
```

执行 ls -l 或 ls -al 命令后显示的结果中，最前面的第 2～10 个字符是用来表示权限。第一个字符一般用来区分文件和目录:

三种权限:

Read 权限：控制读文件内容

Write 权限：控制读文件内容

Execute 权限：控制将文件调入内存并执行

r=4，w=2，x=1

例如，rwx 属性则可以表示为 4+2+1=7；rw-属性则可以表示为 4+2=6。

权限分成三组，每组都是“rwx”格式，三组分别代表每个文件的有所有者、所在

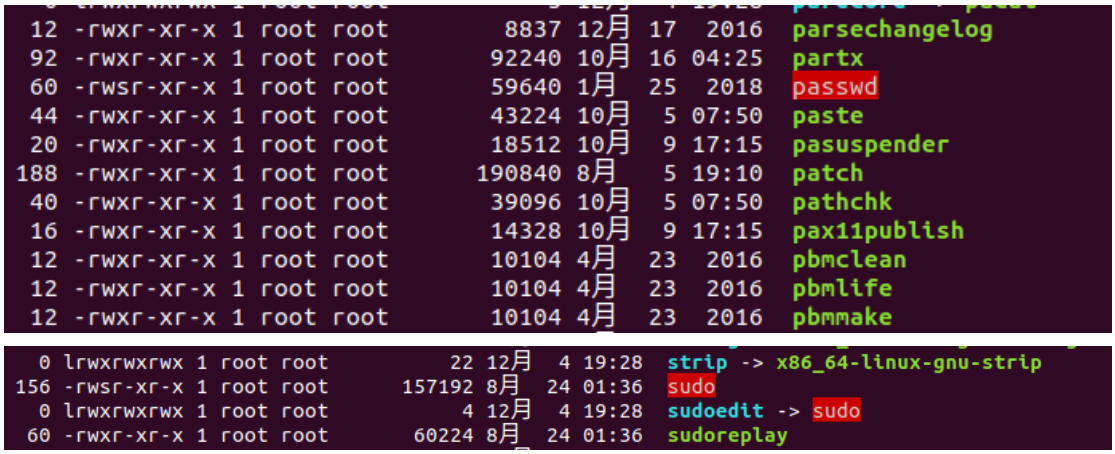
组以及其它组。这样写的目的是能够保证文件所有者对该文件有很高的权限，但是其他用户则没有这么多的权限，包护文件内容以达到安全的目的。

（b） 找到 2 个设置了 setuid 位的可执行程序。

设置 setuid 位的作用：

普通用户在执行命令的时候，会暂时获得所有者的身份，普通用户对这个程序必须有执行权限。

利用 ls -ls /usr/bin 命令，查询 /usr/bin 文件夹下所有文件的权限。结果如下：



如图，图中所示的文件中，第三个权限位设为 s 的就是设置了 setuid 的可执行文件。随便选出了两个： passwd 和 sudo。passwd 这个可执行文件的所有者是 root，但是其他用户对于它也有执行权限，并且它自身具有 SUID 权限。那么当其他用户来执行 passwd 这个可执行文件的时候，产生的进程的就是以 root 用户的 ID 来运行的。

如果不设置 setuid 位，程序执行就可以不去短暂获取 root 权限了，就达不到相应的功能了。

（2） 设置文件或目录权限

chmod 参数说明：

u：表示文件拥有者，即 user

g：组拥有者，即 group

o：其它用户拥有者,即 other

a：所有用户，即相当于 ugo

：省略不写代表 a，即所有用户

（a）用户 A 具有文本文件"流星雨.txt"，该用户允许别人下载

下载指令可以只设置 r 权限，即设置其他用户对该文件的读权限。

```
rocketeerli@rocket:~$ vim 流星雨.txt
rocketeerli@rocket:~$ ls -al 流星雨.txt
-rw-rw-r-- 1 rocketeerli rocketeerli 20 12月  5 01:45 流星雨.txt
```

创建好文件后，可以看到，权限位最后三位为 r--，说明其他用户已经有了对这个文件的读权限，因此不需要再额外设置了。

（b）用户 A 编译了一个可执行文件"cal.exe"，该用户想在系统启动时运行

可以选择建立一个软连接，将可执行文件“cal.exe”放到/etc/init.d 中，在/etc/rc.d/rc3.d 中建立软链接。然后，由于需要在系统启动时运行，因此需要设置为 root 权限。

（c）用户 A 有起草了文件"demo.txt"，想让同组的用户帮其修改文件

这里需要修改同组用户的权限，即权限位的 4-6 位。修改文件对应的权限是读写权限，因此，需要给同组用户增加读写权限。使用命令：chmod 664 demo.txt

操作如图：

```
rocketeerli@rocket:~$ vim demo.txt
rocketeerli@rocket:~$ chmod 664 demo.txt
rocketeerli@rocket:~$ ls -al demo.txt
-rw-rw-r-- 1 rocketeerli rocketeerli 17 12月  5 02:05 demo.txt
```

（d）一个 root 用户拥有的网络服务程序"netmonitor.exe"，需要设置 setuid 位才能完成其功能。

直接在该文件所在的位置上执行 chmod u+s netmonitor.exe 命令即可成功设置 setuid 位。

1.2

一些可执行程序运行时需要系统管理员权限，在 UNIX 中可以利用 setuid 位实现其功能，但 setuid 了的程序运行过程中拥有了 root 权限，因此在完成管理操作后需要切换到普通用户的身份执行后续操作。

创建一个新的用户：

```
rocketeerli@rocket:~$ sudo adduser liguojian
正在添加用户"liguojian"...
正在添加新组"liguojian" (1001)...
正在添加新用户"liguojian" (1001) 到组"liguojian"...
创建主目录"/home/liguojian"...
正在从"/etc/skel"复制文件...
输入新的 UNIX 密码:
重新输入新的 UNIX 密码:
passwd: 已成功更新密码
正在改变 liguojian 的用户信息
请输入新值，或直接敲回车键以使用默认值
        全名 []: liguojian
        房间号码 []:
        工作电话 []:
        家庭电话 []:
        其它 []:
这些信息是否正确？ [Y/n] Y
rocketeerli@rocket:~$
```

一共使用了三个用户，root(UID=0)，rocketeerli (UID=1000)，liguojian (UID=1001)

(1)设想一种场景，比如提供 http 网络服务，需要设置 setuid 位，并为该场景编制相应的代码。

利用 socket 编程中的 bind 函数进行测试，首先获取建立 socket 套接字之前的实际用户 ID，有效用户 ID，保存的用户 ID。然后建立 socket 套接字，并用 bind() 函数进行绑定，判断绑定是否成功并检查绑定后的实际用户 ID，有效用户 ID，保存的用户 ID。代码如下：

首先获取执行之前的三个 uid ：

```
// 三个 id 分别对应了实际用户 ID，有效用户 ID，保存的用户 ID
    uid_t ruid, euid,suid;
    getresuid(&ruid, &euid, &suid);
    printf("------开始的 uid :------ \n ruid = %d, euid = %d, suid
= %d\n", ruid, euid, suid);
```

然后，执行 http 服务，再查看三个 uid，观察是否变化：

```
    // 1. 提供 http 网络服务，需要设置 setuid 位，否则会失败
    printf("------1.提供 http 网络服务------\n");
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if(server_socket < 0)
    {
        printf("erro \n");
    }
    // bind 绑定
    struct sockaddr_in server_sockaddr;
    memset(&server_sockaddr, 0, sizeof(server_sockaddr));
    server_sockaddr.sin_family = AF_INET;
    server_sockaddr.sin_port = htons(80);
    server_sockaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    int is_bind = bind(server_socket, (struct sockaddr
*)&server_sockaddr,
            sizeof(server_sockaddr));
    if (is_bind < 0)
    {
        printf("bind error \n");
    }
    else
    {
        printf("bind success \n");
    }
    getresuid(&ruid, &euid, &suid);
    printf("------bind 后的 uid :------ \n ruid = %d, euid = %d, suid
= %d\n", ruid, euid, suid);
```

运行结果：

普通用户调用 http 网络服务：

```
rocketeerli@rocket:~$ ./main
------开始的 uid :------
 ruid = 1000, euid = 1000, suid = 1000
------1.提供 http 网络服务------
bind error
------bind 后的 uid :------
 ruid = 1000, euid = 1000, suid = 1000
```

root 用户调用 http 网络服务：

```
rocketeerli@rocket:~$ sudo ./main
[sudo] rocketeerli 的密码:
------开始的 uid :------
 ruid = 0, euid = 0, suid = 0
------1.提供 http 网络服务------
bind success
------bind 后的 uid :------
 ruid = 0, euid = 0, suid = 0
```

可以看到，普通用户是不能执行 http 网络服务，只有超级用户才可以。

(2)如果用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别。

直接在父进程进行 fork()，然后分别查看 fork() 前后的实际用户 ID，有效用户 ID，保存的用户 ID。代码如下：

```
// 2. 用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别
    printf("2. 用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别\n");
    if(fork() == 0)
    {
        getresuid(&ruid, &euid, &suid);
        printf("------子进程 uid:------\n ruid = %d, euid = %d, suid
= %d\n",
                ruid, euid, suid);

        // 3. 利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化
        printf("3. 利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变
化\n");
         execl("./a", "./a", (char *)0);
    }
    else
    {
        getresuid(&ruid, &euid, &suid);
        printf("------父进程 uid:------\n ruid = %d, euid = %d, suid
= %d\n",
                ruid, euid, suid);
    }
```

运行结果如图：

普通用户运行程序：

父进程 uid：

```
2. 用户fork进程后，父进程和子进程中euid、ruid、suid的差别
------父进程 uid:------
 ruid = 1000, euid = 1000, suid = 1000
```

子进程 uid:

```
------子进程 uid:------
 ruid = 1000, euid = 1000, suid = 1000
```

超级用户运行程序:

父进程 uid:

```
2. 用户fork进程后，父进程和子进程中euid、ruid、suid的差别
------父进程 uid:------
 ruid = 0, euid = 0, suid = 0
```

子进程 uid:

```
------子进程 uid:------
 ruid = 0, euid = 0, suid = 0
```

(3)利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化。

首先设置可执行文件为 root 用户的:

```
rocketeerli@rocket:~$ sudo g++ -o a a.cpp
[sudo] rocketeerli 的密码:
rocketeerli@rocket:~$ ls -al a
-rwxr-xr-x 1 root root 16624 12月 12 19:57 a
```

然后，调用 chmod u+s a，设置可执行程序的 setuid 位

```
rocketeerli@rocket:~$ sudo chmod u+s a
rocketeerli@rocket:~$ ls -al a
-rwsr-xr-x 1 root root 16624 12月 12 19:57 a
```

代码如下:

```
// 3. 利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化
printf("3. 利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化\n");
execl("./a", "./a", (char *)0);
```

代码运行结果:

普通用户执行:

```
3. 利用 execl 执行 setuid 程序后，euid、ruid、suid是否有变化
执行了一个可执行文件
------利用 exec 执行 setuid 程序后 uid :------
 ruid = 1000, euid = 0, suid = 0
```

可以看到，普通用户执行了 setuid 的可执行程序后，euid 和 suid 会变为 0.

超级用户执行:

```
3. 利用 execl 执行 setuid 程序后，euid、ruid、suid是否有变化
执行了一个可执行文件
------利用 exec 执行 setuid 程序后 uid :------
 ruid = 0, euid = 0, suid = 0
```

由于文件本身就是属于超级用户的，且 root 的权限是最高的，因此不需要更改 uid。

(4)程序何时需要临时性放弃 root 权限，何时需要永久性放弃 root 权限，并在程序中分别实现两种放弃权限方法。

代码如下：

程序临时性放弃 root 权限：

```c
// 4.1 程序临时性放弃 root 权限
void abandonRootTemporary(uid_t uid_tran)
{
    uid_t ruid, euid,suid;
    getresuid(&ruid, &euid, &suid);
    if (euid == 0)
    {
        // 临时性放弃 root 权限
        int is_seteuid = seteuid(uid_tran);
        getresuid(&ruid, &euid, &suid);
        if(euid > 0)
        {
            printf("------ 4.1 临时性放弃 root 权限成功 ------\n");
        }
        else
        {
            printf("------ 4.1 临时性放弃 root 权限失败 ------\n");
        }
        printf("ruid = %d, euid = %d, suid = %d\n", ruid, euid, suid);
    }
    else
    {
        printf("4.1 无 root 权限，无法放弃 root 权限\n");
    }
}
```

程序永久性放弃 root 权限：

```c
// 4.2 永久性放弃 root 权限
void abandonRootPermanent(uid_t uid_tran)
{
    uid_t ruid, euid,suid;
    getresuid(&ruid, &euid, &suid);
```

```
    if (euid != 0 && (ruid == 0 || suid == 0))
    {
        setuid(0);
        getresuid(&ruid, &euid, &suid);
    }
    if (euid == 0)
    {
        // 永久性放弃 root 权限
        setresuid(uid_tran, uid_tran, uid_tran);
        getresuid(&ruid, &euid, &suid);
        if(ruid > 0 && euid > 0 && suid > 0)
        {
            printf("------ 4.2 永久性放弃 root 权限成功 ------\n");
        }
        else
        {
            printf("------ 4.2 永久性放弃 root 权限失败 ------\n");
        }
        printf("ruid = %d, euid = %d, suid = %d\n", ruid, euid, suid);
    }
    else
    {
        printf("4.2 无 root 权限，无法放弃 root 权限\n");
    }
}
```

代码调用：

```
// 4.两种放弃 root 权限的方式
abandonRootTemporary(1001);  // 临时性放弃 root 权限
abandonRootPermanent(1001);  // 永久性放弃 root 权限
```

程序运行结果：

运行命令 sudo ./main

运行结果：

```
------ 4.1 临时性放弃root权限成功 ------
ruid = 0, euid = 1001, suid = 0
------ 4.2 永久性放弃root权限成功 ------
ruid = 1001, euid = 1001, suid = 1001
```

可以看到临时性放弃 root 权限的时候，只有 euid 变为其他用户的 uid，ruid 和 suid 都还保持为 0。

(5)execl 函数族中有多个函数，比较有环境变量和无环境变量的函数使用的

差异。

有环境变量和无环境变量的区别:

execl()用来执行参数 path 字符串所代表的文件路径，接下来的参数代表执行该文件时传递过去的 argv(0)、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

execlp()会从 PATH 环境变量所指的目录中查找符合参数 file 的文件名，找到后便执行该文件，然后将第二个以后的参数当做该文件的 argv[0]、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

代码如下:

```
// 5. 比较有环境变量和无环境变量的函数使用的差异。
// 5.1 有环境变量的函数使用
if (fork() == 0)
{
    printf("5.1 有环境变量的函数使用\n");
    execlp("a", "./a", (char *)0);
}
wait(NULL);
if (fork() == 0)
{
    // 5.2 无环境变量的函数使用
    printf("5.2 无环境变量的函数使用\n");
    execl("./a", "./a", (char *)0);
}
wait(NULL);
```

第二部分：

利用 chroot 工具来虚拟化管理

1) 实现 bash 或 ps 的配置使用。

利用 ldd 命令，查看需要的动态函数库。将/bin/bash 中的 bash 文件，和一些 lib

库的链接文件拷贝到 chroot 的目的文件中

```
rocketeerli@rocket:~$ ldd /bin/bash
        linux-vdso.so.1 (0x00007ffe1c526000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007efdeea54000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007efdeea4e000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007efdee864000)
        /lib64/ld-linux-x86-64.so.2 (0x00007efdeedb1000)
```

利用 install 命令进行拷贝：

mkdir -p /var/chroot/usr/lib
install -C /usr/lib/libskey.so.2 /var/chroot/usr/lib
install -C /usr/lib/libmd.so.2 /var/chroot/usr/lib

测试 bash ：

```
rocketeerli@rocket:~/myftp$ sudo chroot ./
bash-4.4#
```

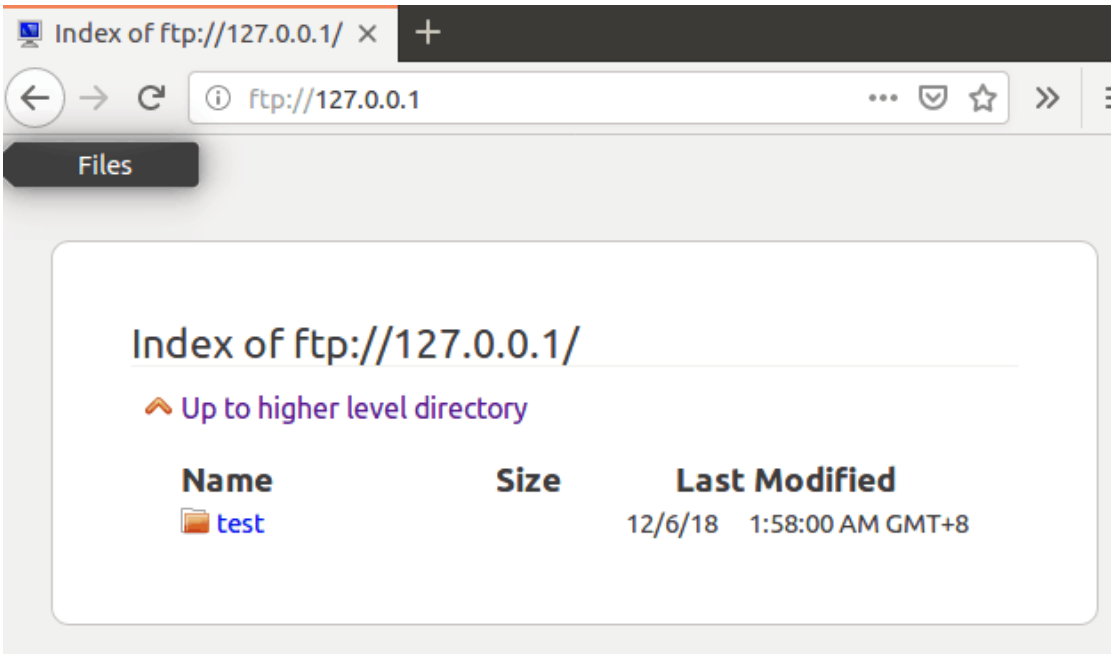
成功

2) 利用 chroot 实现 SSH 服务或 FTP 服务的虚拟化隔离。

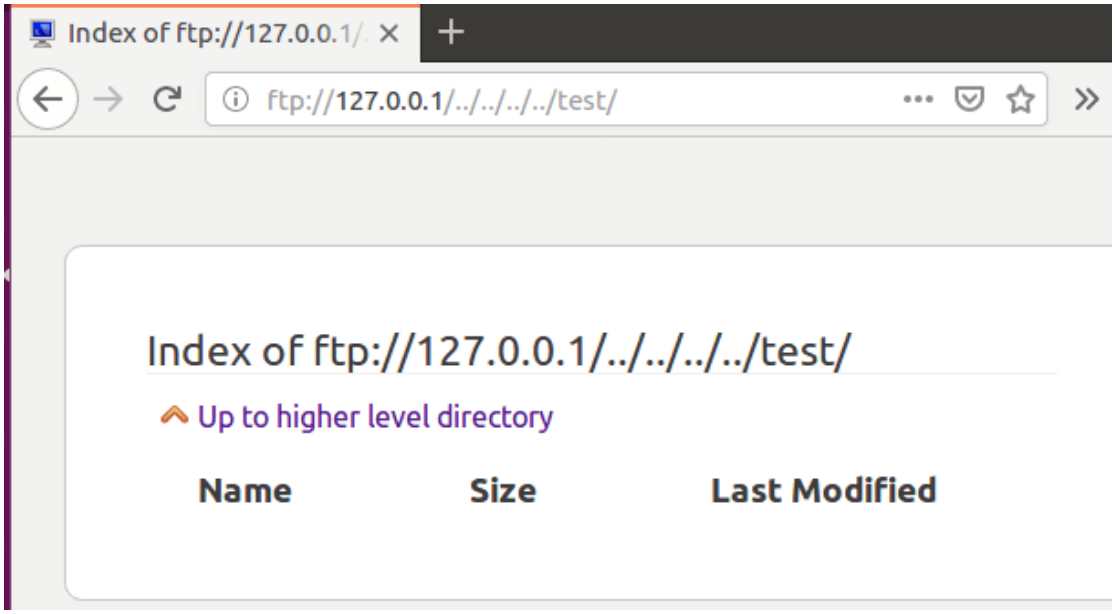
1. 首先按照实验指导书上的教程创建文件夹。

2. 然后创建 ftp 用户。

3. 测试 ftp 连接：



4. 虚拟化隔离：



如图所示，可以看到，无论点击多少次返回上一层目录，都还是在这个目录下，

即实现了虚拟化隔离。

3) chroot 后如何降低权限，利用实验一中编制的程序检查权限的合理性。

chroot 后会将 uid 设置为非 root，这里测试设置成 1001
运行 sudo ./code1 命令：

```
pre:
ruid:0  euid:0  suid:0
post:
ruid:1001       euid:1001       suid:1001
```

代码如下:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    int ruid, euid, suid;
    getresuid(&ruid, &euid, &suid);
    printf("pre:\nruid:%d\teuid:%d\tsuid:%d\n", ruid, euid, suid);
    chroot("/home/rocketeerli/myftp");
    setresuid(1001, 1001, 1001);
    getresuid(&ruid, &euid, &suid);
    printf("pre:\nruid:%d\teuid:%d\tsuid:%d\n", ruid, euid, suid);
```

```
}
```

4) 在 chroot 之前没有采用 cd xx 目录，会对系统有何影响，编制程序分析其影响。

没有在监狱目录 chroot，此时创建的监狱将会拥有当前目录的访问权限，超出了监狱的范围。

（1） 测试没有改变目录，此时的目录为/home/lovebear/acg,结果打印出上级的目录，说明可以访问所在目录的信息及文件。

```
pre:
ruid:0  euid:0  suid:0
```

```
(unreachable)/home/
(unreachable)/home/
(unreachable)/home
(unreachable)/
(unreachable)/
(unreachable)/
```

（3） 在代码中加一行 chdir，使 chroot 前目录已经变为监狱目录，结果表明，最上层为/，而对于其他目录，是透明的。

```
pre:
ruid:0  euid:0  suid:0
/
/
/
/
/
/
```

代码如下：

```
#include <sys/stat.h>
int main()
{
    uid_t ruid, euid, suid;
    char *pwd;
    getresuid(&ruid, &euid, &suid);
    printf("pre:\nruid:%d\teuid:%d\tsuid:%d\n", ruid, euid, suid);
    chroot("/home/rocketeerli/myftp");
    pwd = getcwd(NULL, 80);
    printf("%s\n", pwd);
    int i = 0;
    for(i = 0; i < 5;i++)
    {
```

```
        if(chdir(..) < 0)
        {
            exit(-1);
        }
        pwd = getcwd(NULL, 80);
        printf("%s\n", pwd);
        free(pwd);
    }
}
```

总结

本次实验，让我对用户的权限管理有了初步的了解，对三种 uid 的使用，有了深刻的认识。而且，对 chroot 虚拟化隔离有了进一步的认识。学习到了很多。