

**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. 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
2. 查看/etc/passwd文件和/usr/bin/passwd文件的权限设置，如图所示：

C:\Users\13144\Documents\Tencent Files\2482354780\FileRecv\MobileFile\Image\3Z$J_6}(@L[LC6{F{9D`3ZS.png

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。

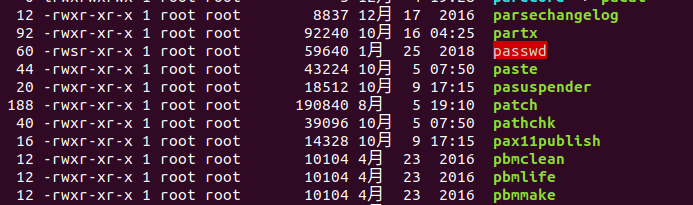
权限分成三组，每组都是 “rwx” 格式，三组分别代表每个文件的有所有者、所在组以及其它组。这样写的目的是能够保证文件所有者对该文件有很高的权限，但是其他用户则没有这么多的权限，包护文件内容以达到安全的目的。

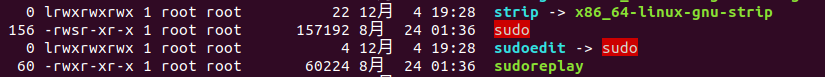
1. 找到2个设置了setuid位的可执行程序。

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。

1. 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。



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。

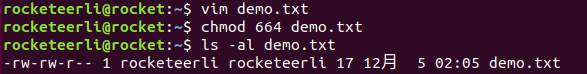
（b）用户A编译了一个可执行文件”cal.exe”，该用户想在系统启动时运行

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：

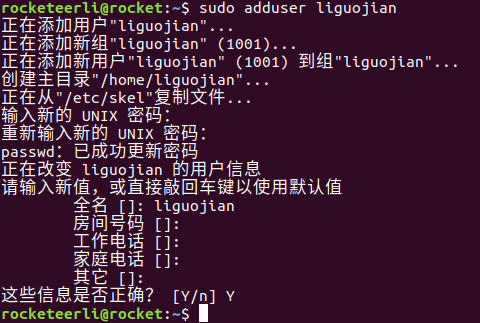


（d）一个root用户拥有的网络服务程序”netmonitor.exe”，需要设置setuid位才能完成其功能。

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。

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：



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");

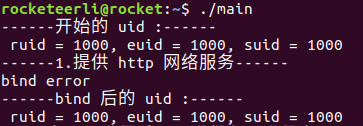
}

getresuid(&ruid, &euid, &suid);

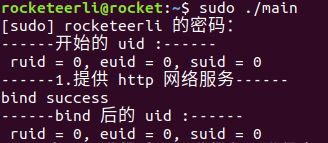
printf("------bind 后的 uid :------ \n ruid = %d, euid = %d, suid = %d\n", ruid, euid, suid);

运行结果：

普通用户调用 http 网络服务：



root 用户调用 http 网络服务：



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。

1. 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：



子进程 uid：



超级用户运行程序：

父进程 uid：

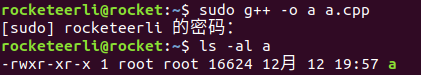


子进程 uid：



1. 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：



然后，调用 chmod u+s a，设置可执行程序的 setuid 位



代码如下：

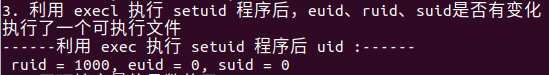
// 3. 利用 execl 执行 setuid 程序后，euid、ruid、suid是否有变化

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);

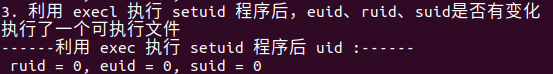
代码运行结果：

普通用户执行：



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：



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。

1. 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：

// 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：

// 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

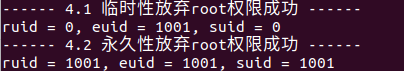
abandonRootTemporary(1001); // 临时性放弃root权限

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：



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。

1. 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);

}

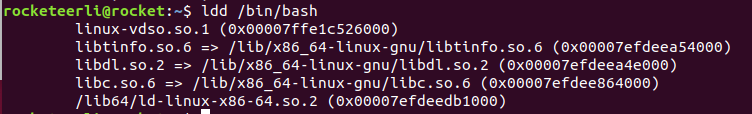
    wait(NULL);

**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



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

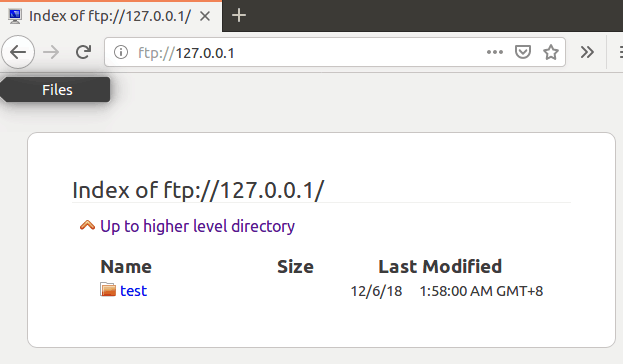
install -C /usr/lib/libmd.so.2 /var/chroot/usr/lib

测试 bash ：

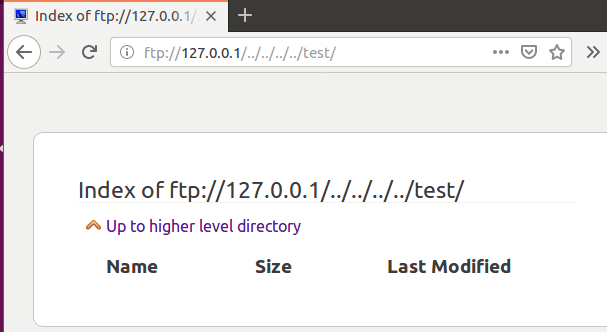


成功

1. **利用chroot实现SSH服务或FTP服务的虚拟化隔离。**
2. 首先按照实验指导书上的教程创建文件夹。
3. 然后创建 ftp 用户。
4. 测试 ftp连接：



1. 虚拟化隔离：

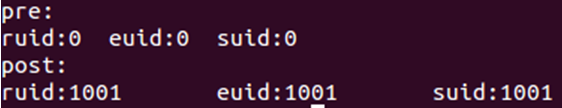


如图所示，可以看到，无论点击多少次返回上一层目录，都还是在这个目录下，即实现了虚拟化隔离。

1. **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：



代码如下：

#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);

}

1. **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。
2. 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。

**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。