**计算机系统安全实验报告**

**实验名称：passwd实现细粒度访问控制及root能力安全使用**

**班级：1703101**

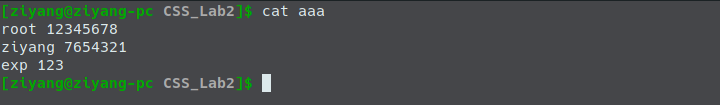
**学号：1170300520**

**姓名：郭子阳**

计算机学院

一 分析passwd程序实现过程，模拟系统中密码修改机制，在自主访问控制系统中实现细粒度的权限管理。

类似/etc/shadow文件aaa格式设置为，用户名+空格+密码，如下：



编写passwd.c文件模拟/usr/bin/passwd的功能，功能描述如下：

* 所有用户都可使用passwd+密码修改自己的密码
* root用户可使用passwd+用户名+密码修改任意用户的密码

编译后的passwd程序的文件所有者是root，并且设置了setuid位，使任何用户都可以以root身份执行，aaa文件设置了只允许root读写，不允许其他用户读写。

passwd程序首先要获取进程的ruid，以判断执行的用户，由于获取的仅仅只是用户的id，而不是用户的用户名，于是需要调用getpwuid()函数获取对应的用户名：

uid\_t ruid, euid, suid;

struct passwd\* userStruct;

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

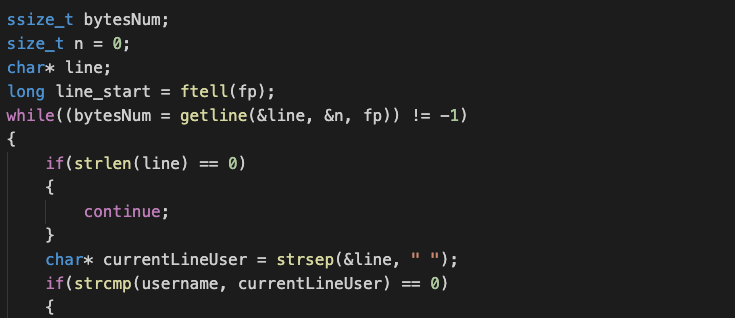
userStruct = getpwuid(ruid);

并根据argc，即参数的个数来判断，如果参数只有一个（argc == 2），说明只是修改该用户自己的密码，则无需进行权限判断；如果参数有两个（argc == 3），说明调用者想修改其他用户的密码，此时需要判断调用用户是否是root，如果是root才允许，否则设置errno位，输出错误信息：



修改密码用的函数changePassword()函数，接受两个参数：username和password。

首先使用getline函数按行读取aaa文件，按照空格拆分每一行，将空格前的内容与username相比较，如果相同，则说明需要修改该行：



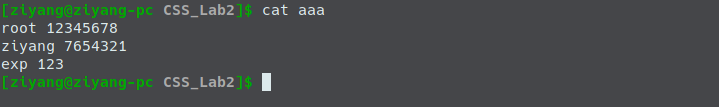
发现密码所在行后，就从下一行开始一直读到文件尾，将读到的内容保存在一个字符串中，用于修改后再次写入。

接着将文件指针移到密码所在行开头，按照username+空格+password的格式写入后加换行，接着将刚刚保存的字符串接着写入到文件，写完后，使用ftruncate()函数删除后续的内容，防止修改前比修改后长而出现修改不完全的情况。

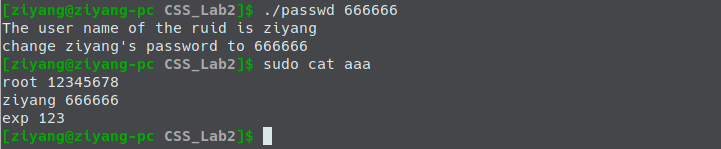


修改密码测试：

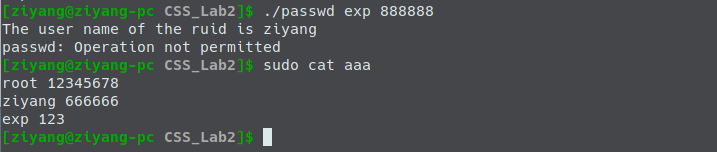
初始密码文件如下：



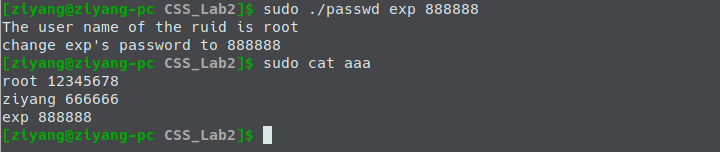
以普通用户身份（ziyang）修改自己的密码：



试图修改exp的密码，会提示Operation not permitted，无法修改：



当以root用户执行时，修改exp的密码为888888:



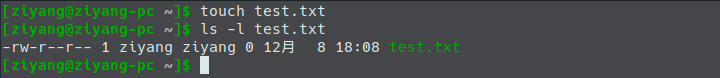
可以看到被成功修改。

二 root的capability位使用

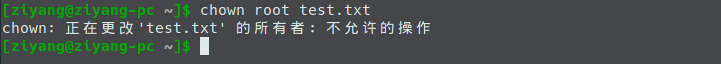
1. cap\_chown能力位

该能力允许用户任意修改文件的拥有者。

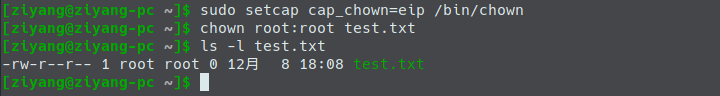
首先创建一个文件，其所属用户为ziyang：



修改能力位之前，试图文件拥有者设置为root：



执行setcap命令后，再尝试修改所有者：

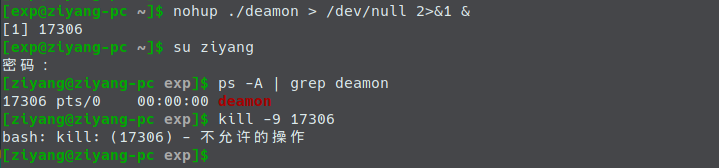


即可成功修改。

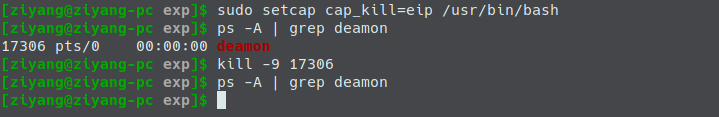
1. cap\_kill能力位

该能力允许用户结束其他用户执行的进行。

切换到用户exp，并在后台执行一个进程，当切换到其他用户（ziyang）时，可以使用ps看到这个进程，但是无法结束它：



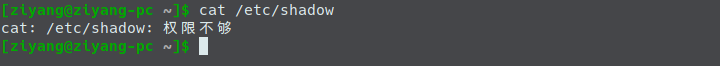
当给bash程序赋予cap\_kill位后（kill信号由bash发出），ziyang即可kill掉其他用户的进程：



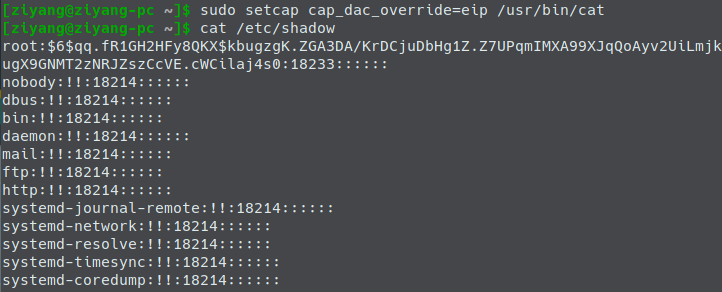
1. cap\_dac\_override能力位

该能力允许用户无视文件能力位设置读写执行文件。

首先尝试使用cat命令读/etc/shadow文件：



给/usr/bin/cat设置了cap\_dac\_override能力位后，即可无视权限读取文件：



1. 系统启动时关闭某能力位，对系统的应用和安全性有何影响，以具体能力位为例说明。

例如关闭cap\_chown位，则任何用户都可以随意改变任意文件的所属用户，相当于可以访问或者执行任何文件。

若关闭cap\_kill位，任何用户可以关闭任何进程，可能关闭系统的重要进程，造成系统不稳定。

若关闭cap\_dac\_override位，则任何用户可以无视权限地读写执行任何文件。

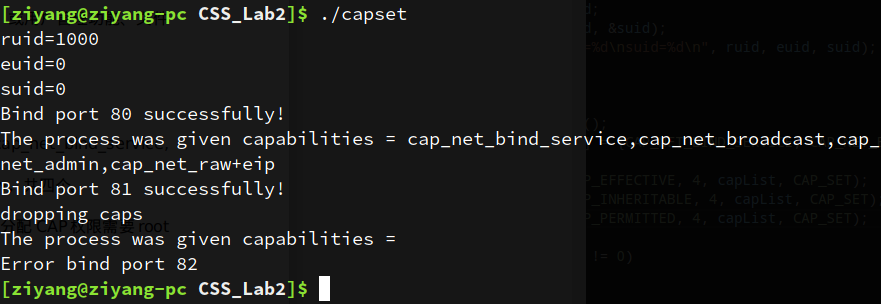
1. 组合系统的部分能力位，实现系统的网络管理功能，或用户管理功能、文件管理功能。

编写capset.c，以实现绑定端口的权限的分配。

绑定较低的端口需要的CAP权限有cap\_net\_bind\_service, cap\_net\_broadcast, cap\_net\_admin, cap\_net\_raw，共四个。

capset.c编译后的setcap需要设置setuid位，是因为分配CAP权限需要root权限。

capset运行结果如下：



capset程序首先会试图绑定80端口，结果是成功绑定，原因是程序设置了setuid位，以root的身份运行。之后将上述提到的四个权限分配给进程：

cap\_t caps = cap\_init();

cap\_value\_t capList[4] = {CAP\_NET\_BIND\_SERVICE, CAP\_NET\_BROADCAST,CAP\_NET\_ADMIN, CAP\_NET\_RAW};

cap\_set\_flag(caps, CAP\_EFFECTIVE, 4, capList, CAP\_SET);

cap\_set\_flag(caps, CAP\_INHERITABLE, 4, capList, CAP\_SET);

cap\_set\_flag(caps, CAP\_PERMITTED, 4, capList, CAP\_SET);

if(cap\_set\_proc(caps) != 0)

{

perror("capset");

return -1;

}

随后尝试绑定81端口，成功绑定。

最后清除caps数组并将caps设置给进程CAP，相当于剥夺上述四个权限：

cap\_clear(caps);

if(cap\_set\_proc(caps) != 0)

{

perror("capset");

return -1;

}

listCaps();

所以最后再尝试绑定端口82时就会失败。

1. 编制攻击程序，测试能力位的安全性。

编制程序tryChangeTime.c，陷入死循环持续尝试修改当前时间：



一旦/usr/bin/date被设置为cap\_sys\_time=eip，那么攻击程序立刻就可以成功修改系统时间，达到了攻击的目的。