**计算机学院系统安全**

实验二

细粒度权限管理及实现root能力的分发和管理

1. 实验内容
2. 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。配合第3章 在基于用户权限管理基础上，进行细粒度的权限管理。
3. 利用root的能力机制实现系统加固，有效实现root能力的分发和管理。提供程序比较进行root能力管理前后系统安全性的差异。配合第4章 实现root的多种能力的有效管理，提高root用户权利的合理分发，测试root能力管理的安全性和有效性
4. 实验设计
   1. **Linux密码修改机制，passwd程序功能的仿制实现**

1、passwd程序功能描述

在Linux中，passwd程序是可信任的，修改存储经过加密的密码的影子密码文件（/etc/shadow），passwd程序执行它自己内部的安全策略，允许普通用户修改属于他们自己的密码，同时允许root修改所有密码。为了执行这个受信任的作业，passwd程序需要有移动和重新创建shadow文件的能力，在标准Linux中，它有这个特权，因为passwd程序可执行文件在执行时被加上了setuid位，它作为root用户（它能访问所有文件）允许，然而，许多程序都可以作为root允许（实际上，所有程序都有可能作为root允许）。这就意味着任何程序（当以root身份运行时）都有可能能够修改shadow文件。

2、实验要求

自己编制文件和程序，仿制passwd程序修改/etc/shadow的功能，包括：

a) 自己设置一个类/etc/shadow文件aaa，该文件中约定好内容格式，和读取该文件的程序相配合，文件中包括超级用户及其内容、普通用户及其内容

b)编制程序使得：Root用户能够读取和修改aaa文件中所有用户的内容普通用户仅能够读取和修改aaa文件中属于自己用户的内容

c）普通用户能以root身份执行所编制的类passwd程序

**2.1 root 的capability使用**

1、 修改系统内核，配置capability的能力位，实现几种能力位的设置可验证。

样例：redhat 2.4下的能力

1）函数说明

getcap可以获得程序文件所具有的能力(CAP).

getpcaps可以获得进程所具有的能力(CAP).

setcap可以设置程序文件的能力(CAP).

2、实验要求：

（1）实现３种基本能力位的授权和查看，并分析授权前和授权后的差异；

（2）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，比如cap\_sys\_module，cap\_linux\_immutable

（3）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。

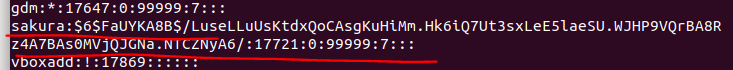
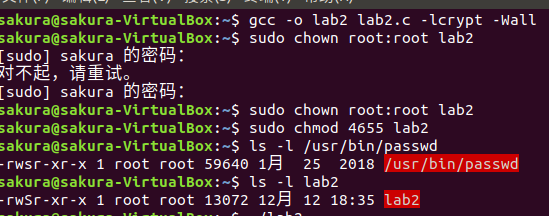
（4）编制攻击程序，测试能力位的安全性。

1. 实验过程

**1.** **密码修改机制，passwd程序功能的仿制实现**

把shadow文件复制一下：然后粘贴在文件myshadow里，用命令：

cat /etc/myshadow查看复制文件。

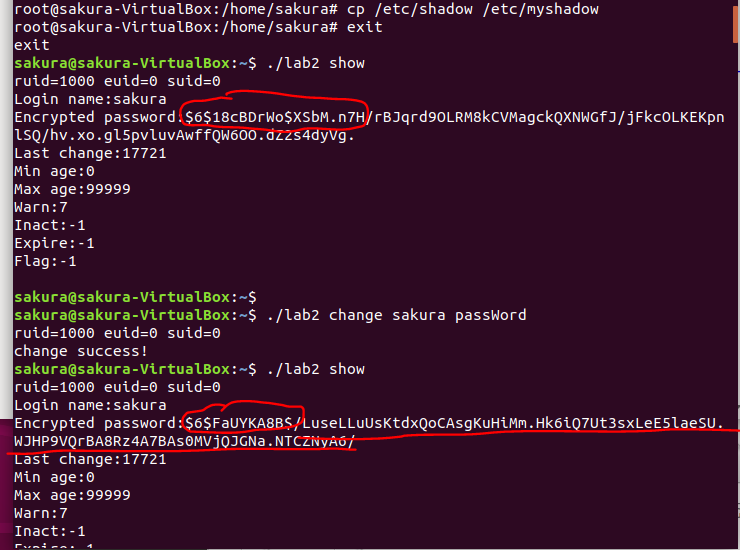


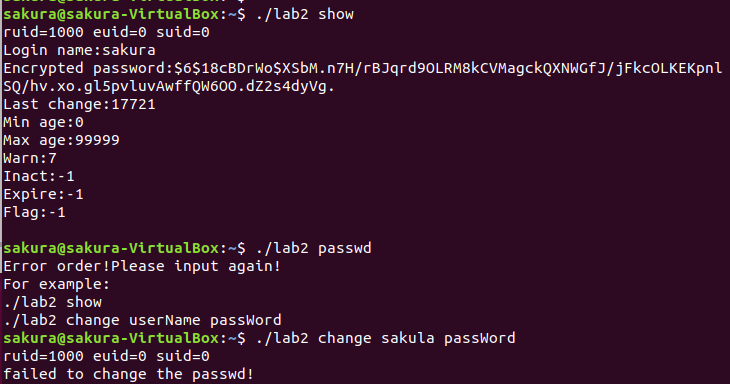
Lab2.c文件根据输入命令：

./lab2 show 是输出全部用户

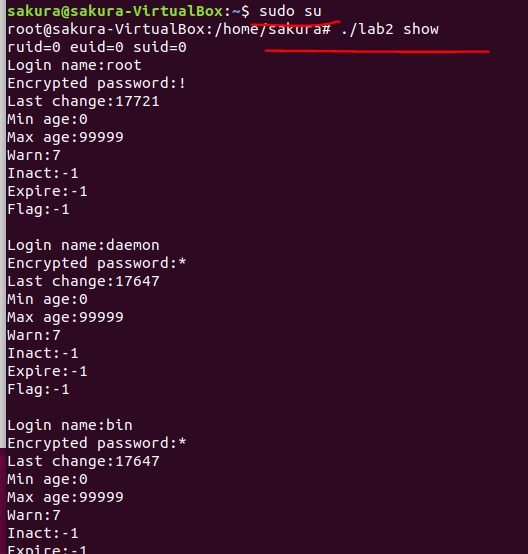
./lab2 change [user] password 是对password进行修改，

默认改为6$FaUYKA8B$。因为shadow其实是加密后的密码，这里用6$FaUYKA8B$来表示新密码。



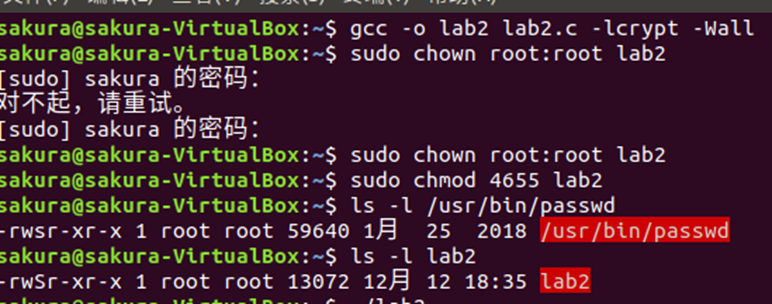


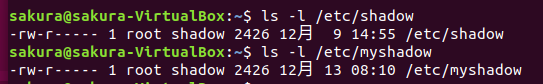
Root用户能够读取和修改aaa文件中所有用户的内容普通用户仅能够读取和修改aaa文件中属于自己用户的内容



普通用户能以root身份执行所编制的类passwd程序

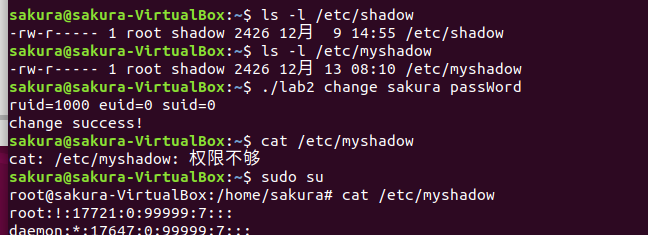
我们可以看到普通用户的euid和suid都为0；

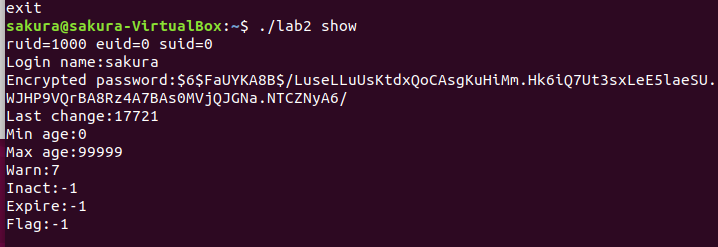


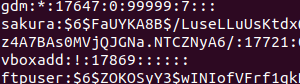


而且shadow文件属于root，在组shadow中，但是lab2却可以修改它。说明用户是用root权限去修改的shadow。

下图中，Sakura用户不可以看shadow文件，没有权限，但是却可以通过执行./lab2去查看权限位和密码加密值。说明普通用户能以root身份执行所编制的类passwd程序。







**2.** **root 的capability使用**

（1）实现３种基本能力位的授权和查看，并分析授权前和授权后的差异；

**授权能力1：CAP\_CHOWN 0(允许改变文件的所有权)**

授权前：普通用户无权修改/bin/ls的文件拥有者

授权普通用户可以用/bin/chown程序更改任意文件的owner,如下:

setcap cap\_chown=eip /bin/chown

查看/bin/chown程序的能力值,如下:

getcap /bin/chown

/bin/chown = cap\_chown+eip

切换到test用户,将/bin/ls程序的owner改为test,如下:

su - test

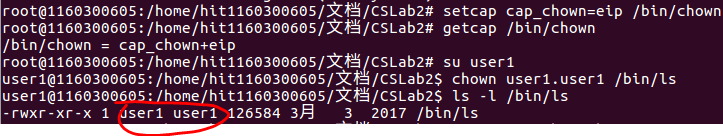
chown test.test /bin/ls

ls -l /bin/ls

-rwxr-xr-x. 1 test test 118736 Jun 14 2010 /bin/ls



授权后：可以看到普通用户user1成功修改了/bin/ls文件的拥有者



1. cap\_chown=eip：

是将chown的能力以cap\_effective(e),cap\_inheritable(i),cap\_permitted(p)三种位图的方式授权给相关的程序文件.

2)如果改变文件名,则能力保留到新文件.

3)用setcap -r /bin/chown可以删除掉文件的能力.

4)重新用setcap授权将覆盖之前的能力.

**授权能力2：CAP\_DAC\_READ\_SEARCH 2(忽略所有对读、搜索操作的限制)**

授权前：普通用户无权查看/etc/shadow文件的内容



授权后：普通用户可以查看/etc/shadow文件的内容

setcap cap\_dac\_read\_search=eip /bin/cat

切换到普通用户

su – test

查看/etc/shadow,如下:

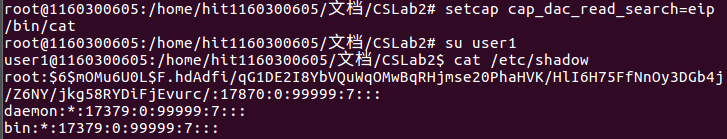
cat /etc/shadow

root:$6$3hJf.BoIVU/cdLKb$JxLXcQScrLS032aFPAQvVc4RzKYNadcIIzxmzAIw.jejrYOHhqdr0oV7sNBL.IhGBo.mMOYEdevlnCp2OGku8.:15094:0:99999:7:::

bin:\*:14790:0:99999:7:::

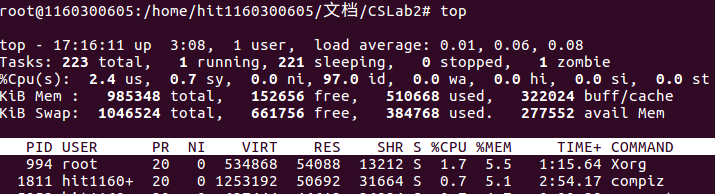
daemon:\*:14790:0:99999:7:::

adm:\*:14790:0:99999:7:::



**授权能力3：CAP\_KILL 5 (允许对不属于自己的进程发送kill信号)**

首先在终端1以root身份启动一个进程top：



授权前：在终端2普通用户试图结束top进程，没有权限不够的提醒，但是终端1 root用户的top进程还在运行。

我们先模拟没有加CAP\_KILL能力的情况,如下:

终端1,用root用户启用top程序,如下:

su - root

top

终端2,用test用户kill之前的top进程,如下:

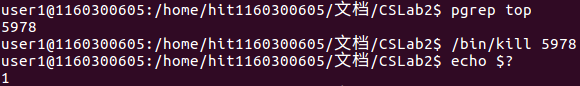
pgrep top

3114

/bin/kill 3114

kill: Operation not permitted

我们发现无法对不属于自己的进程发送信号.



授权后：在终端2普通用户user1成功结束了终端1的top进程

下面我们用CAP\_KILL能力的程序向不属于自己的进程发送信号,如下:

设定kill命令的kill位,如下:

setcap cap\_kill=eip /bin/kill

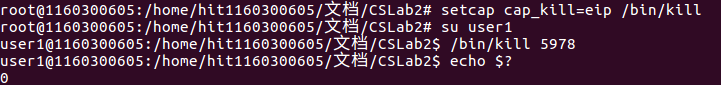
杀掉3114进程,没有问题,如下:

/bin/kill 3114

echo $?

0

注意:普通用户要用/bin/kill这种绝对路径的方式,而不能用kill这种方式.



除了以上的一步步操作，也可以直接运行以下代码：



（2）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，比如cap\_sys\_module，cap\_linux\_immutable

早期linux上信任状模型非常简单，就是"超级用户对普通用户"模型。普通用户的很多操作需要root权限，这通过setuid实现。如果程序编写不好，就可能被攻击者利用，获得系统的控制权。使用能力机制(capability)减小这种风险。系统管理员为了系统的安全可以剥夺root用户的能力，这样即使root用户也将无法进行某些操作。而这个过程又是不可逆的，也就是说如果一种能力被删除，除非重新启动系统，否则即使root用户也无法重新添加被删除的能力。

系统引导时删除部分能力能保护系统。如保护系统工具和日志的完整性。

**CAP\_SYS\_MODULE：插入和删除内核模块：（但是对于在启动时，尚未加载好就完成的攻击束手无策）**

由于普通用户不能插入/删除内核模块,而CAP\_SYS\_MODULE可以帮助普通用户做到这点。对24小时在线运行的服务器，比较担心可加载内核模块的安全性，不需要系统升级等修改内核操作，但又不想完全禁止在系统中使用可加载内核模块或者一些设备的驱动就是一些内核模块。在这种情况下，最好使系统在启动时加载所有的模块，然后禁止加载/卸载任何内核模块。如果把CAP\_SYS\_MODULE从能力边界集中删除，系统将不再允许加载/卸载任何的内核模块。攻击者不能修改系统的内核。系统内核被改动，需要重新启动系统才能使用新内核。

**CAP\_LINUX\_IMMUTABLE：允许修改文件的不可修改(IMMUTABLE)和只添加(APPEND-ONLY)属性**

普通用户不能通过chattr对文件设置IMMUTABLE(chattr +i)和APPEND-ONLY(chattr +a)权限,而通过CAP\_LINUX\_IMMUTABLE可以使普通用户通过自己增减(immutable/append-only)权限.

有些文件用root权限都不能修改，大部分原因是曾经用chattr命令锁定了该文件。如果在系统启动时关闭CAP\_LINUX\_IMMUTABLE能力位，那么普通用户就无法给文件设置不可修改和只添加属性，使攻击者不能删除其攻击轨迹、不能安装后门工具、系统日志文件为“append-only” 、系统工具不被删除和修改。

root用户可以删除系统保留的能力。却不能再恢复被删除的能力，只有init进程能够添加能力。通常，一个能力如果从能力边界集中被删除，只有系统重新启动才能恢复。

用户可通过shell命令行设置能力。例如：禁止加载/卸载任何内核模块，CAP\_SYS\_MODULE能力的值是16，可用下列命令完成：

[root@]# echo 0xFFFEFFFF >/proc/sys/kernel/cap-bound

（3）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。

设置网络服务能力：CAP\_NET\_BIND\_SERVICE、CAP\_NET\_BROADCAST

CAP\_NET\_BIND\_SERVICE 允许绑定到小于1024的端口即公认端口，通常这些端口的通讯明确表明了某种服务的协议。

CAP\_NET\_BROADCAST 允许网络广播和多播访问

拥有root的文件管理能力：CAP\_DAC\_OVERRIDE、CAP\_DAC\_READ\_SEARCH

CAP\_DAC\_OVERRIDE 1 忽略对文件的所有DAC访问限制

CAP\_DAC\_READ\_SEARCH 2 忽略所有对读、搜索操作的限制

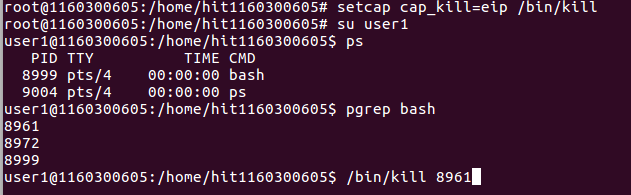
拥有root的用户管理能力：CAP\_SETGID、CAP\_FSETID

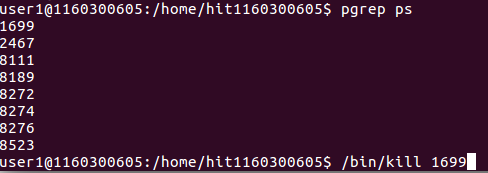
CAP\_FSETID 4 允许设置setuid位

CAP\_SETGID 6 允许改变组ID

（4）编制攻击程序，测试能力位的安全性。

尝试一：开放cap\_kill位并尝试以普通用户user1的身份杀死系统进程ps





在kill到8111的时候一下子黑屏要重新登陆了



尝试二：开放cap\_kill位并尝试以普通用户user1的身份杀死user2的bash进程以使其强制下线

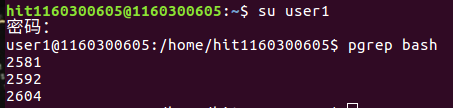
打开一个终端并输出bash进程号，下图看出终端1的bash进程号为2581



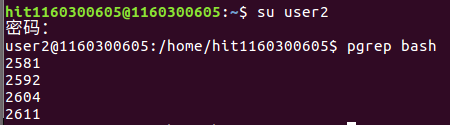
打开另一个终端并输出bash进程号，下图看出终端2的bash进程号为2592



在终端1登录user1，并再次查看bash进程号，下图看出user1的bash进程号为2604



在终端2登录user2，并再次查看bash进程号，下图看出user2的bash进程号为2611



Cap\_kill能力位在前面的实验中已设置好，尝试在终端1杀掉user2的bash进程



此时终端2显示user2已退出



该实验说明如果cap\_kill权限位开放，那么一个普通用户只要通过pgrep bash推知另一个用户的bash进程号，就可以使另一个用户强制退出，说明能力位还是有其不安全性。

1、如果我们fork()了子进程,那么子进程继承父进程的所有能力.

2、不能单独设定CAP\_EFFECTIVE,CAP\_INHERITABLE位图,必须要和CAP\_PERMITTED联用,且CAP\_PERMITTED一定要是其它两个位图的超集.

3、如果两次调用cap\_set\_proc函数,第二次调用的值力值不能少于或多于第一次调用.如第一次我们授权chown,setuid能力,第二次只能是chown,setuid不能是其它的能力值.

4、普通用户不能给进程设定能力.

1. 实验收获

学到了很多知识，对密码机制更理解了。

练习了系统编程。

1. 参考

参考网站：

<https://blog.csdn.net/lisemi/article/details/53163557>

<https://blog.csdn.net/T146lLa128XX0x/article/details/81977249>

六、附录：

**Lab2.c:**

#define \_GNU\_SOURCE

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <shadow.h>

#include <sys/types.h>

#include <pwd.h>

#include <crypt.h>

#include <stdlib.h>

int main(int argc,char\*\*argv)

{

char judgeInput[2][7]= {"show","change"};

uid\_t ruid,euid,suid;

int temp;

int showBranch,changeBranch;

showBranch=!strcmp(judgeInput[0],argv[1]);

changeBranch=!strcmp(judgeInput[1],argv[1]);

/\*---------------------判断输入是否合法-------------------------\*/

if(!((argc==2 && showBranch) || (argc==4 && changeBranch)))

{

printf("Error order!Please input again!\n");

printf("For example:\n./lab2 show\n./lab2 change userName passWord\n");

return -1;

}

/\*------------------------------------------------------------\*/

if((temp=getresuid(&ruid,&euid,&suid))==-1)

perror("getresuid failed!\n");

else

printf("ruid=%d euid=%d suid=%d\n",ruid,euid,suid);

/\*---------------------获得当前用户名---------------------\*/

struct passwd\* cur;

cur=getpwuid(getuid());

/\*---------------------------------------------------------------\*/

/\*------------------------------------show branch-----------------------------------\*/

if(showBranch)

{

struct spwd\* p;

FILE \*fs;

fs=fopen("/etc/myshadow","r");

if(!ruid)

{

while((p=fgetspent(fs))!=NULL)

{

printf("Login name:%s\nEncrypted password:%s\nLast change:%ld\nMin age:%ld\nMax age:%ld\nWarn:%ld\nInact:%ld\nExpire:%ld\nFlag:%ld\n\n",p->sp\_namp,p->sp\_pwdp,p->sp\_lstchg,p->sp\_min,p->sp\_max,p->sp\_warn,p->sp\_inact,p->sp\_expire,p->sp\_flag);

}

}

else

{

while((p=fgetspent(fs))!=NULL)

{

if(!strcmp(cur->pw\_name,p->sp\_namp))

printf("Login name:%s\nEncrypted password:%s\nLast change:%ld\nMin age:%ld\nMax age:%ld\nWarn:%ld\nInact:%ld\nExpire:%ld\nFlag:%ld\n\n",p->sp\_namp,p->sp\_pwdp,p->sp\_lstchg,p->sp\_min,p->sp\_max,p->sp\_warn,p->sp\_inact,p->sp\_expire,p->sp\_flag);

}

}

fclose(fs);

return 0;

}

/\*-----------------------------------------------------------------------------------\*/

/\*------------------------------------change branch----------------------------------\*/

char newPassword[10];

char \*cryptedPassword;

char userTochange[10];

strcpy(userTochange,argv[2]);

strcpy(newPassword,argv[3]);

struct spwd store[100],\*p;

int total=0;

int i;

FILE \*fs;

fs=fopen("/etc/myshadow","r");

int flag=0;

/\*----------------------------root 用户可以随意更改任何人的passwd----------------------\*/

if(!ruid)

{

while((p=fgetspent(fs))!=NULL)

{

store[total]=\*p;

store[total].sp\_namp=malloc(strlen(p->sp\_namp)+1);

store[total].sp\_pwdp=malloc(strlen(p->sp\_pwdp)+1);

strcpy(store[total].sp\_namp,p->sp\_namp);

if(!strcmp(p->sp\_namp,userTochange))

{

cryptedPassword=crypt(newPassword,"$6$FaUYKA8B$CUuItcF");

strcpy(store[total].sp\_pwdp,cryptedPassword);

flag=1;

}

else

{

strcpy(store[total].sp\_pwdp,p->sp\_pwdp);

}

total++;

}

}

/\*----------------------------------------------------------------------------------\*/

/\*------------------------------普通用户只能更改自己的passwd--------------------------\*/

else

{

if(!strcmp(cur->pw\_name,userTochange))

{

while((p=fgetspent(fs))!=NULL)

{

store[total]=\*p;

store[total].sp\_namp=malloc(strlen(p->sp\_namp)+1);

store[total].sp\_pwdp=malloc(strlen(p->sp\_pwdp)+1);

strcpy(store[total].sp\_namp,p->sp\_namp);

if(!strcmp(p->sp\_namp,userTochange))

{

cryptedPassword=crypt(newPassword,"$6$FaUYKA8B$CUuItcF");

strcpy(store[total].sp\_pwdp,cryptedPassword);

flag=1;

}

else

{

strcpy(store[total].sp\_pwdp,p->sp\_pwdp);

}

total++;

}

}

/\*----------------要修改的不是当前用户则直接退出----------------------\*/

}

/\*------------------------------------------------------------------------------------------------\*/

fclose(fs);

if(!flag)

{

printf("failed to change the passwd!\n");

return -1;

}

/\*------------------------修改成功，写回--------------------------------\*/

else

{

printf("change success!\n");

fs=fopen("/etc/myshadow","w");

for(i=0; i<total; i++)

putspent(&store[i],fs);

fclose(fs);

}

return 0;

}

设置能力位再清除：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

#undef \_POSIX\_SOURCE

#include <sys/capability.h>

extern int errno;

void whoami(void)

{

printf("uid=%i euid=%i gid=%i\n", getuid(), geteuid(), getgid());

}

void listCaps()

{

cap\_t caps = cap\_get\_proc();

ssize\_t y = 0;

printf("The process %d was give capabilities %s\n",(int) getpid(), cap\_to\_text(caps, &y));

fflush(0);

cap\_free(caps);

}

int main(int argc, char \*\*argv)

{

int stat;

whoami();

stat = setuid(geteuid());

pid\_t parentPid = getpid();

if(!parentPid)

return 1;

cap\_t caps = cap\_init();

cap\_value\_t capList[5] ={ CAP\_NET\_RAW, CAP\_NET\_BIND\_SERVICE , CAP\_SETUID, CAP\_SETGID,CAP\_SETPCAP } ;

unsigned num\_caps = 5;

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

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

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

if (cap\_set\_proc(caps)) {

perror("capset()");

return EXIT\_FAILURE;

}

listCaps();

printf("dropping caps\n");

cap\_clear(caps); // resetting caps storage

if (cap\_set\_proc(caps)) {

perror("capset()");

return EXIT\_FAILURE;

}

listCaps();

cap\_free(caps);

return 0;

}