哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 计算机系统安全

课程类型： 选修

实验题目：

passwd实现细粒度访问控制及root能力安全使用

学号：1170300728

姓名： 汤添凝

**实验内容：**

2.1 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。

2.2 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。

**实验过程：**

**第一部分：**

**一、密码修改机制，passwd程序功能的仿制实现**

**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

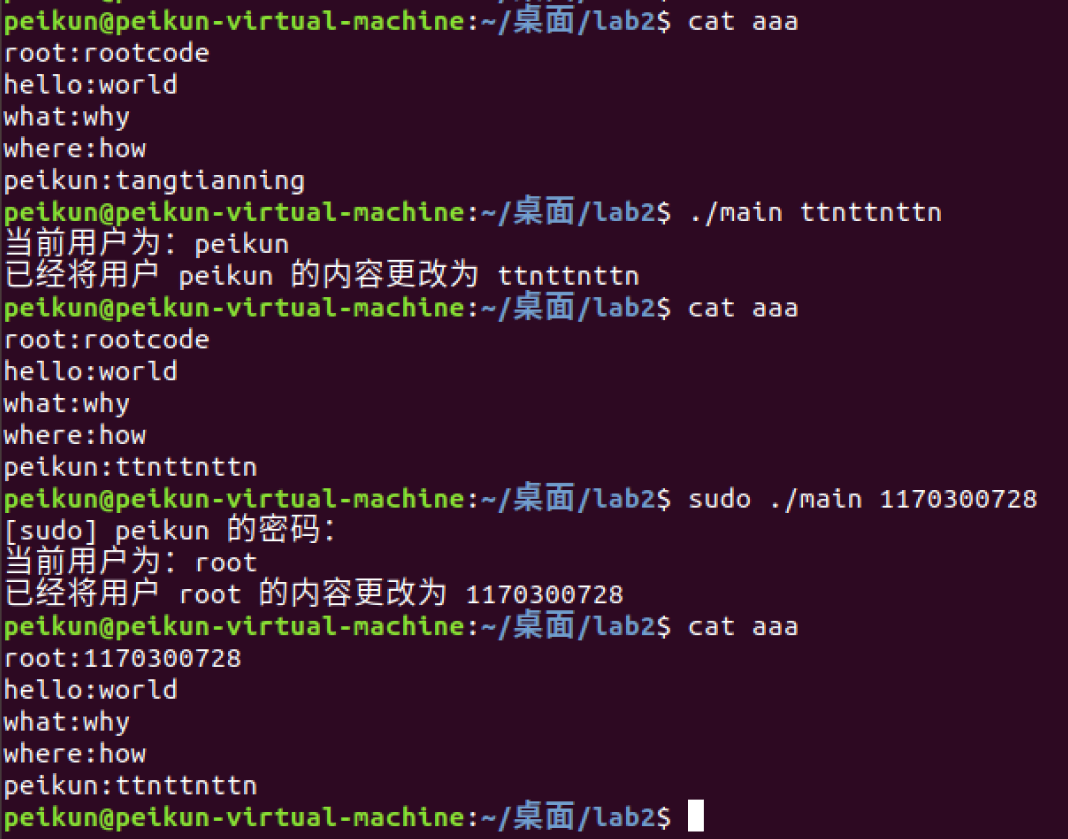
c）普通用户能以root身份执行所编制的类passwd程序

**实验内容：**

1. **自己写一个 aaa 文件**

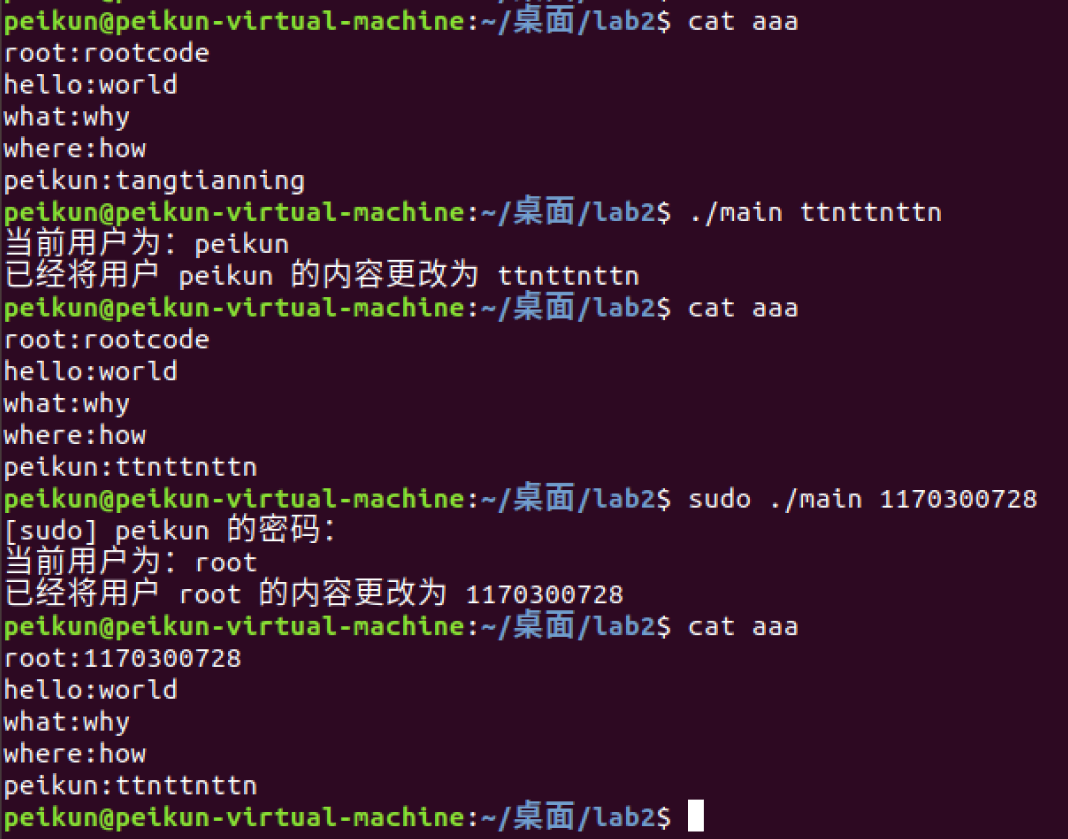
**b) Root用户能够读取和修改aaa文件中所有用户的内容普通用户仅能够读取和**修改aaa文件中属于自己用户的内容

密码更改前， aaa 文件的内容：

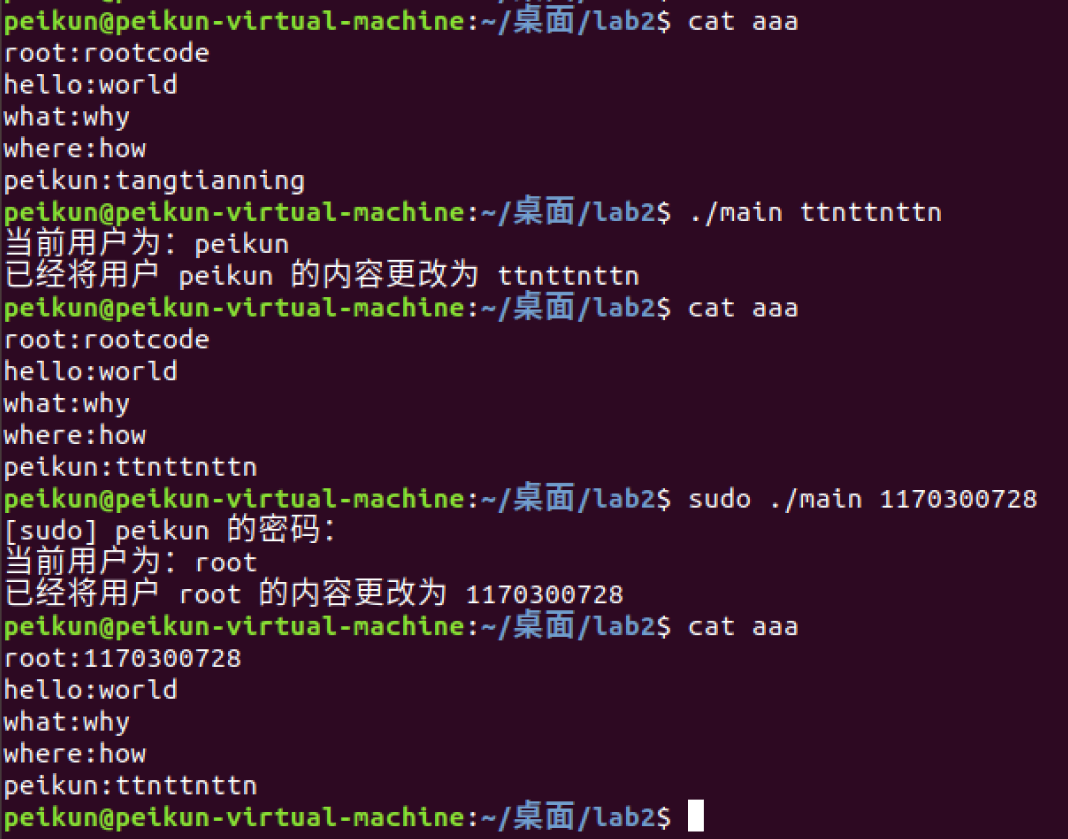


**（1）普通用户仅能够读取和修改aaa文件中属于自己用户的内容：**

更改当前用户的密码，即更改普通用户的密码：

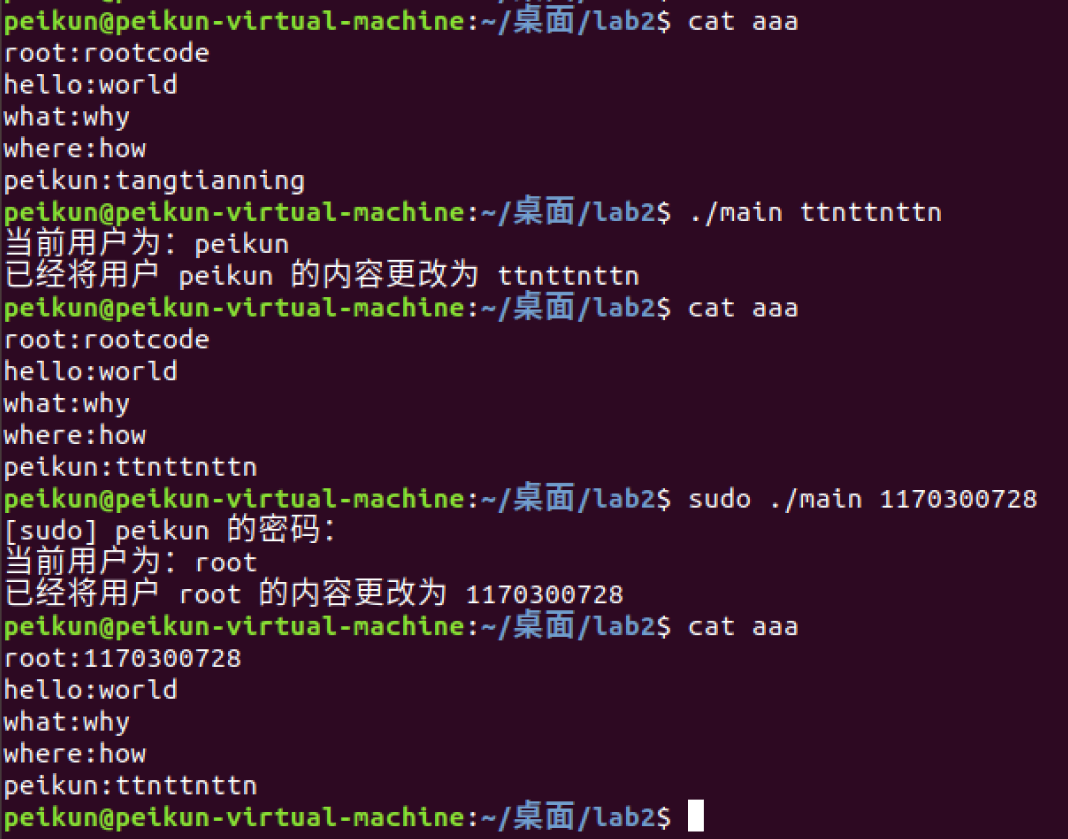


Cat查看运行后的 aaa 文件内容：

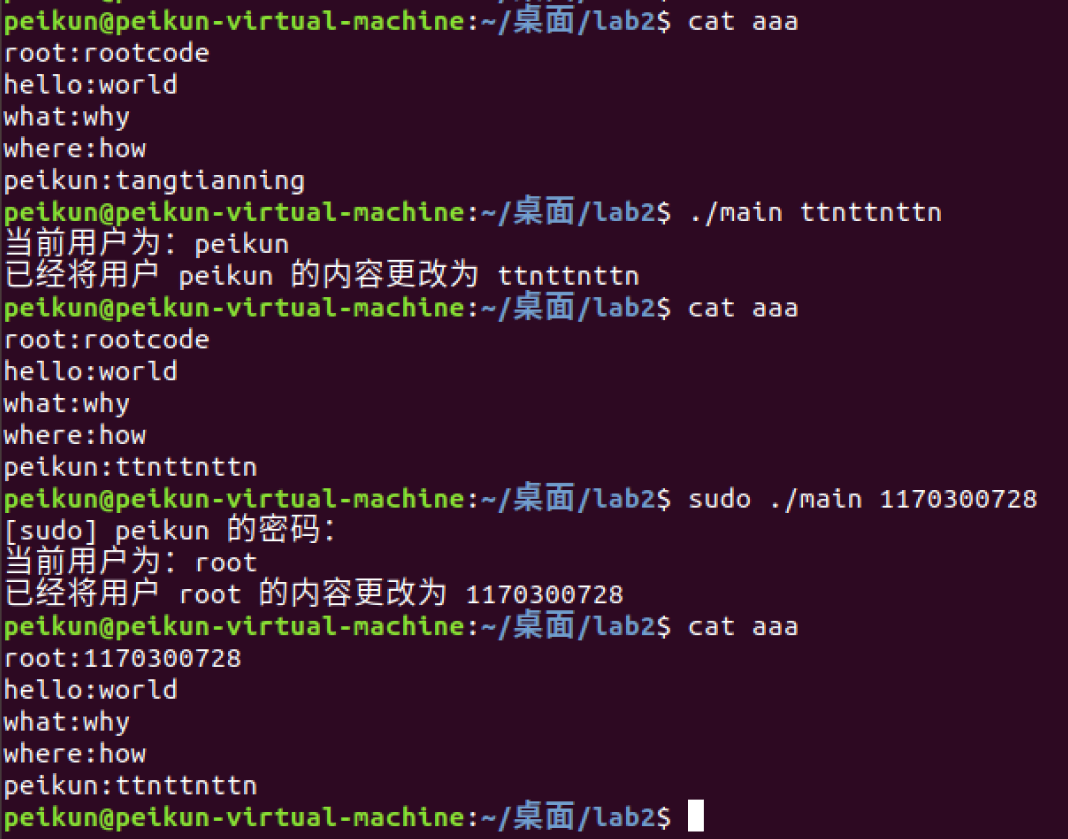


**（2）Root用户能够读取和修改aaa文件中所有用户的内容**

更改 root 用户的密码：

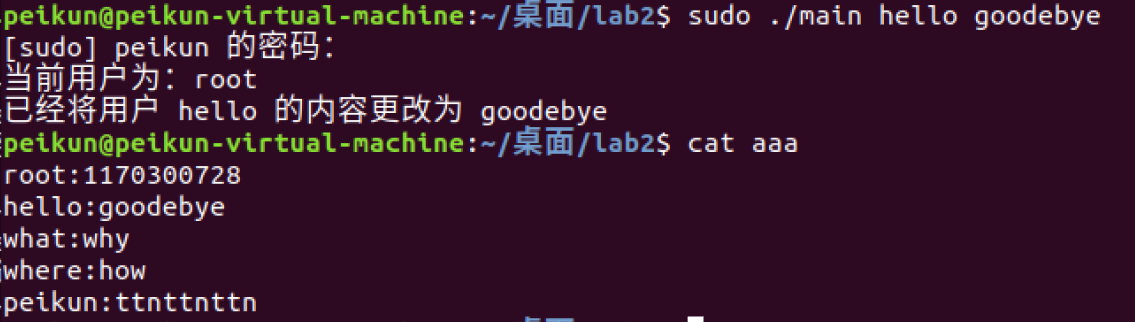


Cat查看运行后的 aaa 文件内容：

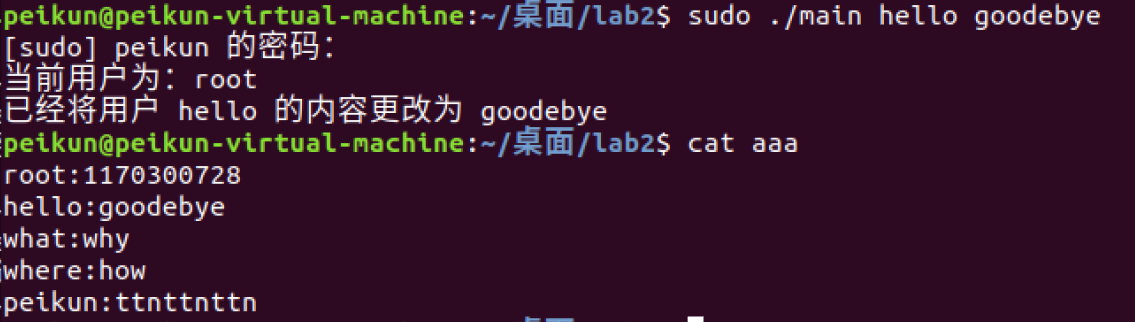


**c) 普通用户能以root身份执行所编制的类passwd程序：**

利用 root 权限，更改其他用户的密码：



Cat查看运行后的 aaa 文件内容：



实验代码：

#include <stdio.h>

#include <cstring>

#include <pwd.h>

#include <unistd.h>

#include <sys/types.h>

// 改变文件中相应用户的内容

void changeFileContext(char \*user, char \*userContext);

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

{

uid\_t ruid, euid, suid;

struct passwd \*user;

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

user = getpwuid(ruid);

printf("当前用户为：%s\n", user->pw\_name);

switch (argc)

{

case 2:

changeFileContext(user->pw\_name, argv[1]);

break;

case 3:

if (strcmp(user->pw\_name, "root") == 0)

changeFileContext(argv[1], argv[2]);

break;

default:

break;

}

return 0;

}

// 改变文件中相应用户的内容

void changeFileContext(char \*user, char \*userContext)

{

FILE \*fp;

char \*str\_line = NULL;

size\_t len = 0;

// ssize\_t是signed size\_t

ssize\_t read\_line;

long offset;

char \*p = NULL;

int is\_same\_str;

char buf\_after[1000] = { 0 };

char buf\_before[100][100];

// 打开可读写的文件，该文件必须存在。

fp = fopen("aaa", "r+");

// 得到文件位置指针当前位置相对于文件首的偏移字节数

offset = ftell(fp);

int i = 0;

while ((read\_line = getline(&str\_line, &len, fp)) != -1)

{

// 存储文件内容

strcpy(buf\_before[i], str\_line);

// 判断字符串str2是否是str1的子串。如果是，则返回str2在str1中首次出现的地址；否则，返回NULL。

p = strstr(str\_line, ":");

if (p == NULL)

{

continue;

}

// p-str\_line 指定比较字符的个数

is\_same\_str = strncmp(user, str\_line, p - str\_line);

// 匹配成功，找到该用户的内容

if (!is\_same\_str)

{

int index = 0;

// 检测流上的文件结束符，如果文件结束，则返回非0值，否则返回0

// 读取当前指针之后的所有文件内容

while (!feof(fp))

{

buf\_after[index++] = fgetc(fp);

}

if (index > 0)

{

buf\_after[index - 1] = '\0';

}

fclose(fp);

break;

}

offset = ftell(fp);

i++;

}

// 重新写入文件

fp = fopen("aaa", "w+");

for (int j = 0; j < i; j++)

{

fprintf(fp, "%s", buf\_before[j]);

}

fprintf(fp, "%s:%s\n", user, userContext);

fprintf(fp, "%s", buf\_after);

printf("已经将用户 %s 的内容更改为 %s \n", user, userContext);

}

**第二部分：**

**二、root 的capability使用**

**实验内容：**

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。

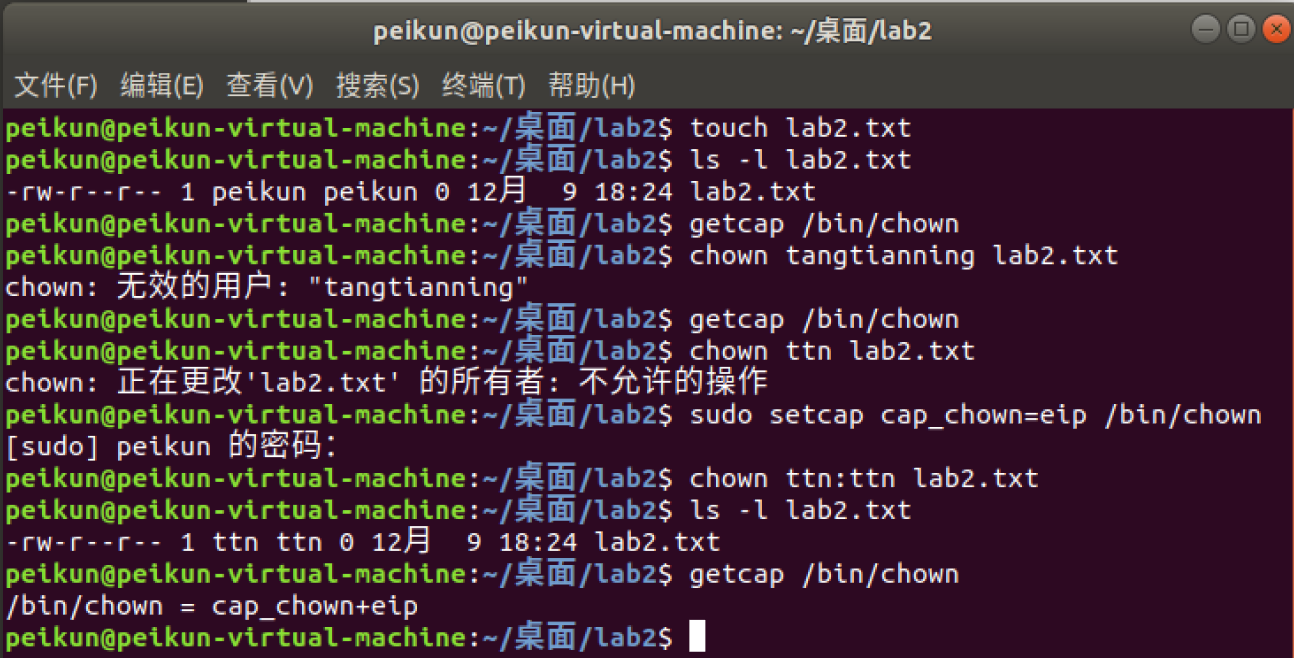
实验过程：

**（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。**

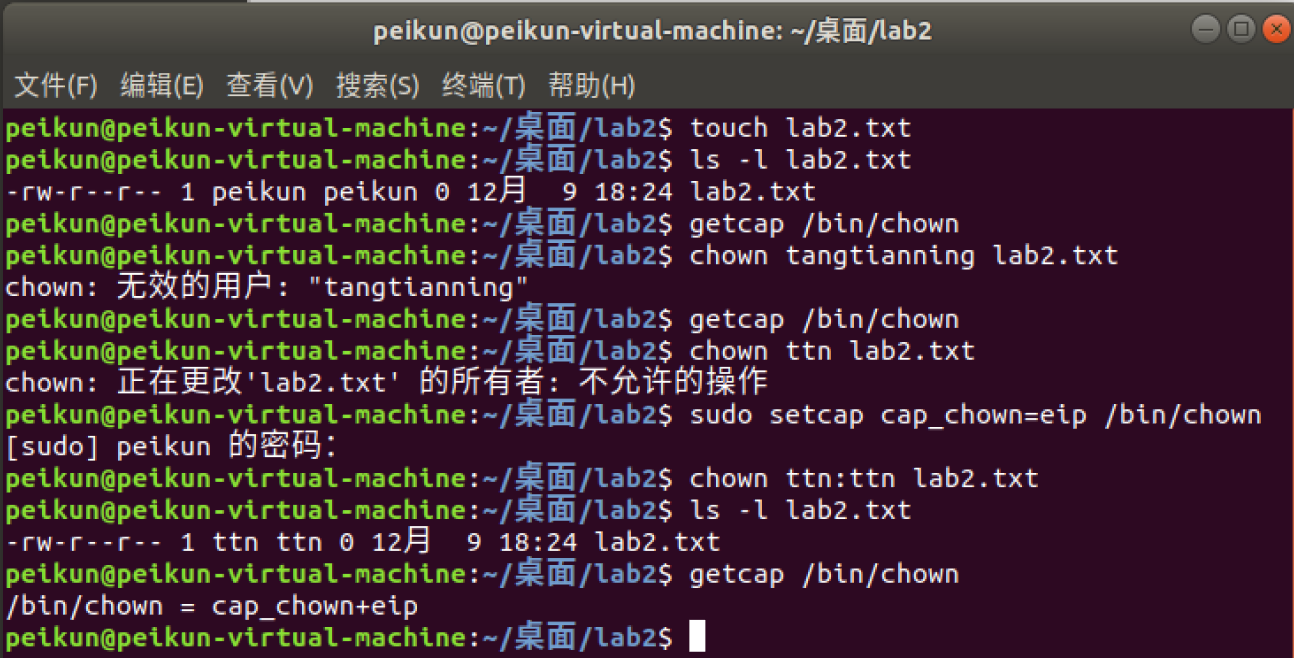
**1.Cap\_chown**

该能力位，设置后，可以修改文件属主的权限。

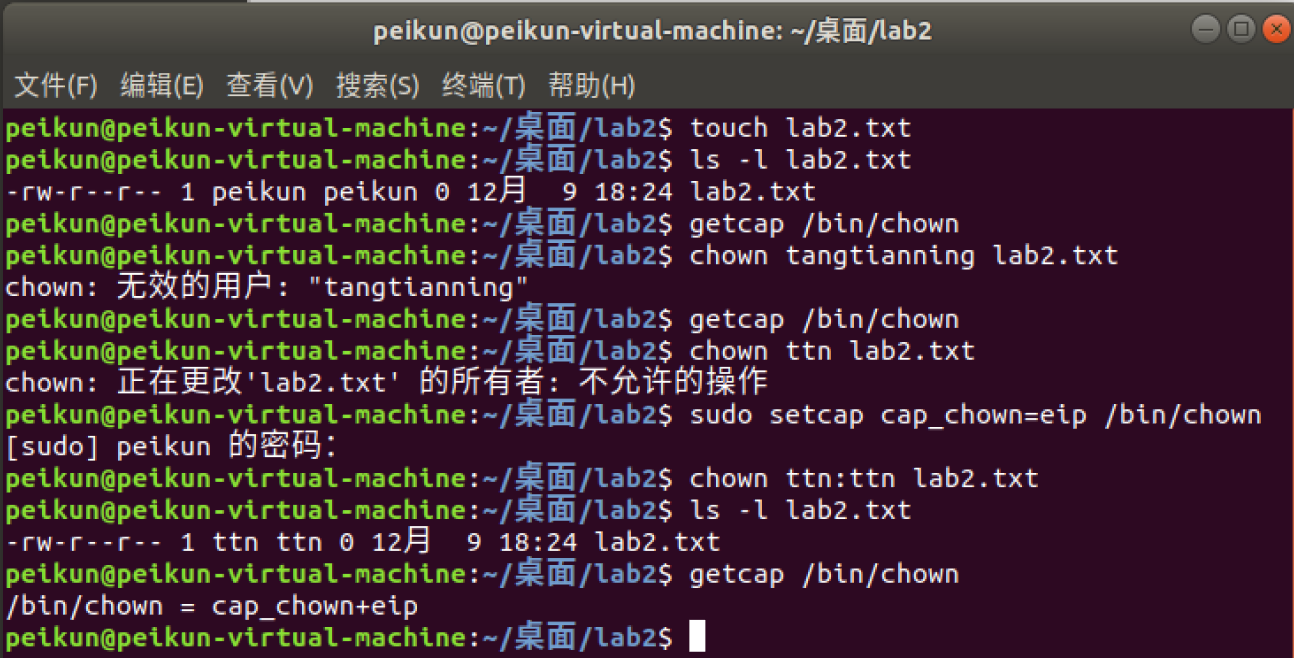
首先创建一个属于用户的文件，这里就创建了一个属于 peikun 的文本文件：



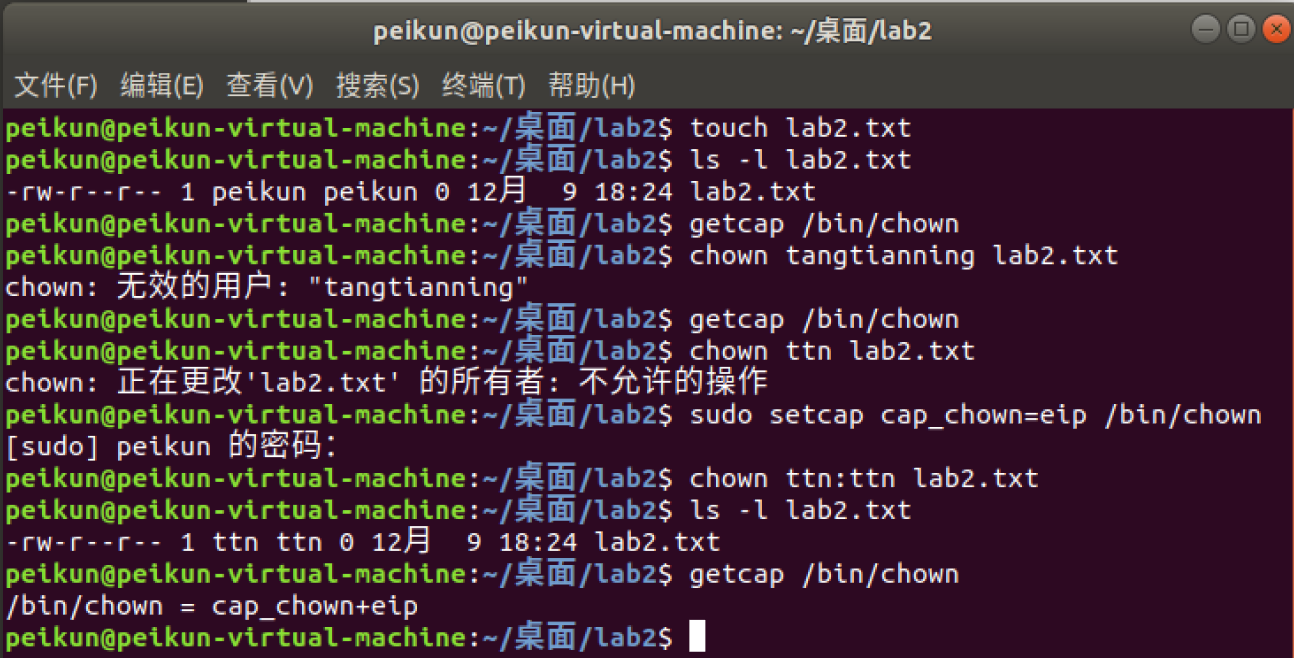
Setcap前，查看当前的能力，尝试修改文件属主：



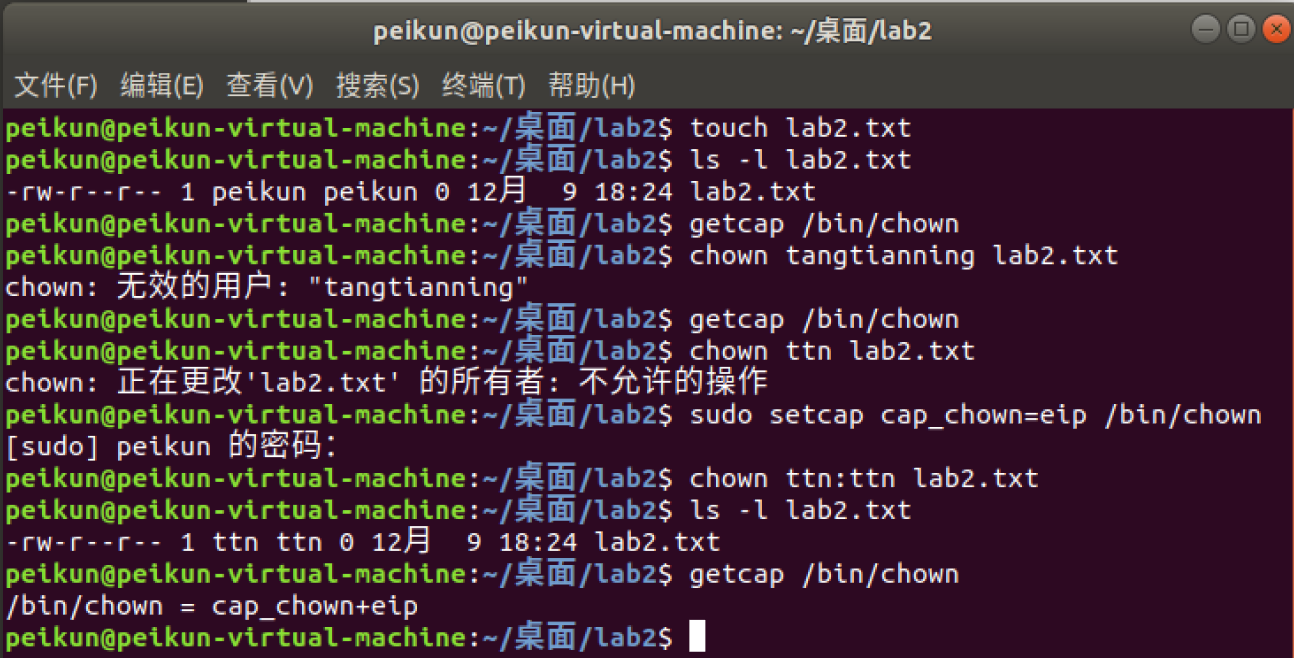
执行 Setcap 命令：



更改文件所有者，由 peikun 更改为 ttn ：



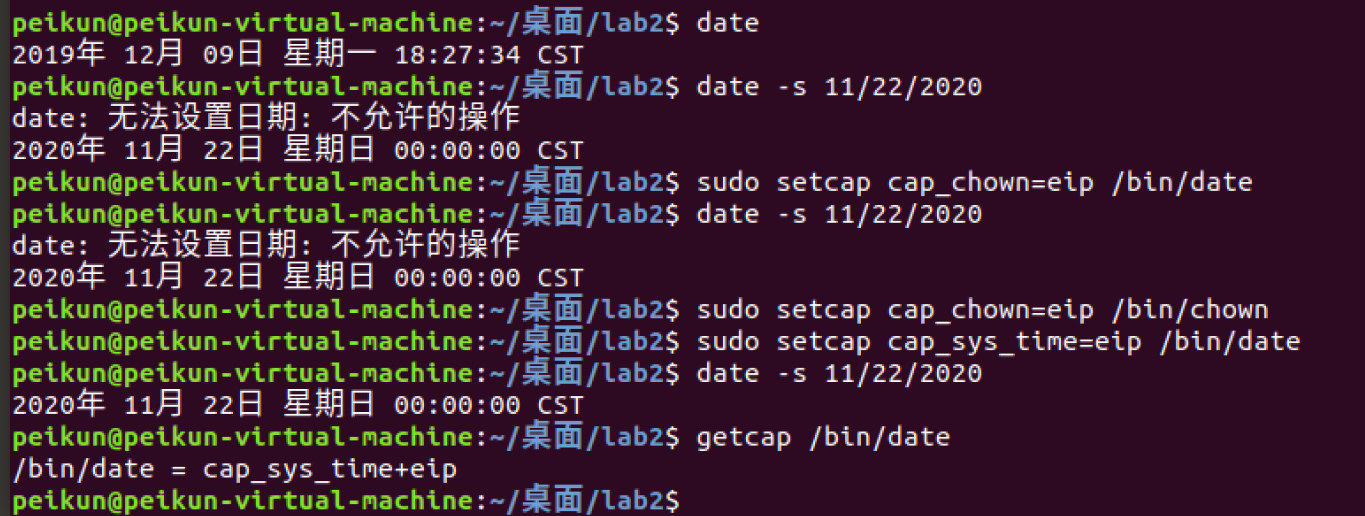
查看 cap ：



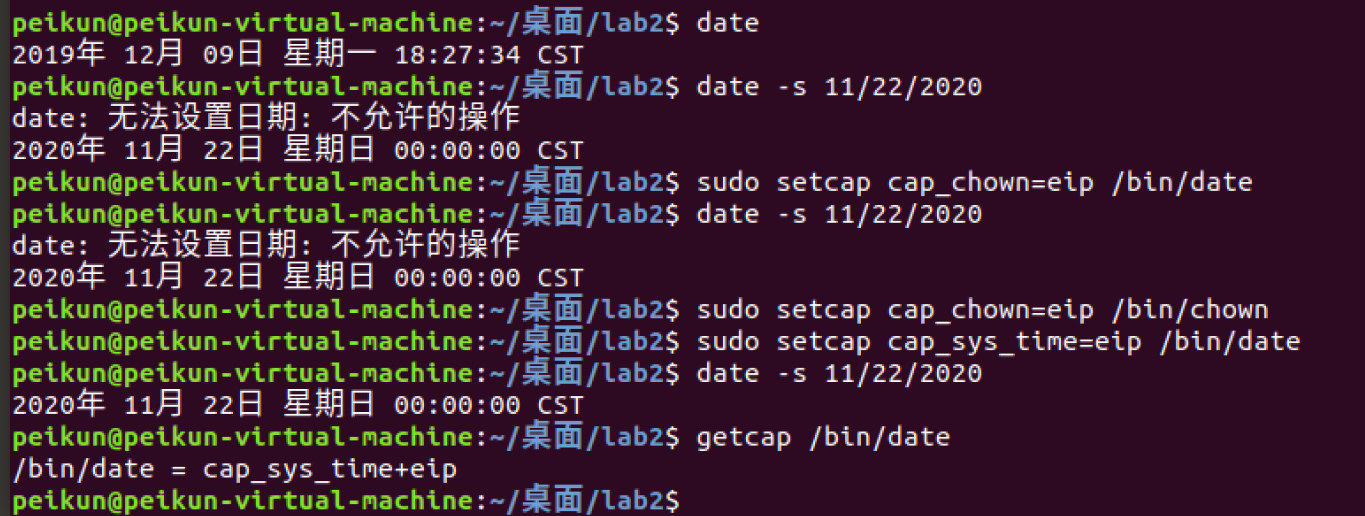
**2.cap\_sys\_time**

该能力位，设置后，允许修改系统时钟

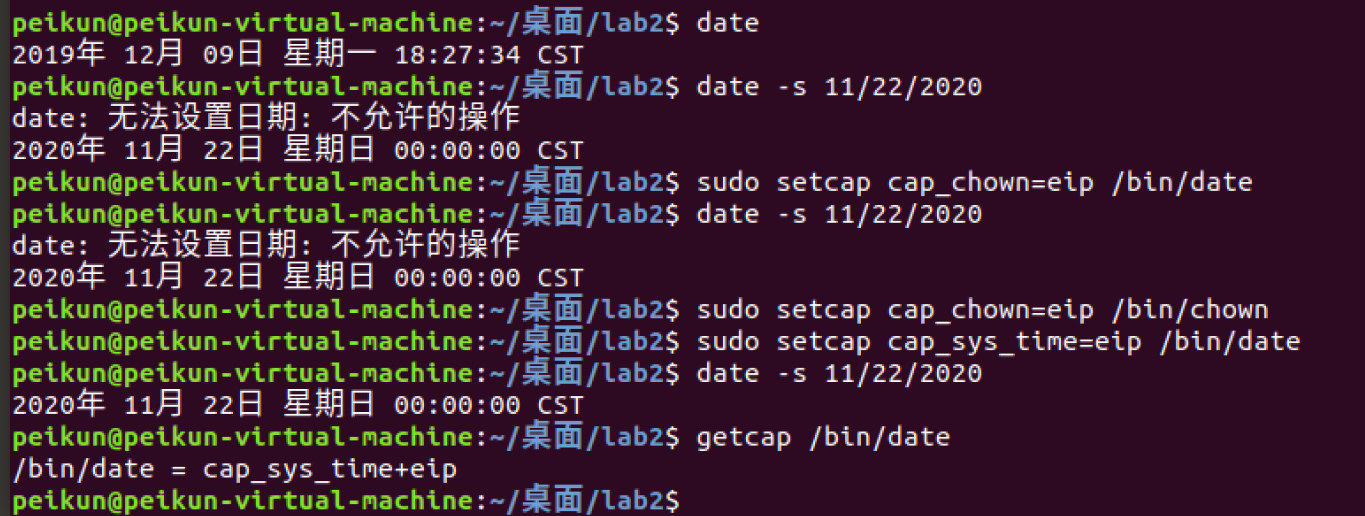
Setcap前，查看当前时间，并尝试修改日期，修改失败：



Setcap：



Setcap 后，修改日期，修改成功，并查看 cap ：



3.cap\_dac\_read\_search，忽略所有对读、搜索操作的限制：

Setcap 前，尝试读 etc/shadow 文件：



Setcap：



Setcap 后，可以查看了：



**（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**

如果关闭了 cap\_chown 能力位，那么所有的程序或用户都可以修改文件属主，那么就相当于可以访问或执行任何文件。

如果关闭了 cap\_sys\_time 能力位，那么所有的程序或用户都可以修改系统的时间，会造成系统的时间错乱。

如果关闭了 cap\_dac\_read\_search 能力位，所有的文件都可以被所有的用户读取，没有不可以读取的文件，这对于一些系统文件来说，是非常不安全的。

**（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。**

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)) { //赋予能力

perror("capset()");

return EXIT\_FAILURE;

}

listCaps();

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");//预计绑定成功

}

printf("dropping caps\n");

cap\_clear(caps); // resetting caps storage 重置能力位

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

perror("capset()");

return EXIT\_FAILURE;

}

listCaps();

is\_bind = bind(server\_socket, (struct sockaddr \*)&server\_sockaddr,

sizeof(server\_sockaddr));//尝试绑定

if (is\_bind < 0)

{

printf("bind error \n");//预计绑定失败

}

else

{

printf("bind success \n");

}

cap\_free(caps);

return 0;

}

思路与实验一的程序一样，测试网络功能，例如bind，发现有能力位的时候bind 成功，之后drop能力，再尝试bind函数，绑定失败。



**（4）编制攻击程序，测试能力位的安全性。**

编写攻击attack程序：

#include <stdio.h>

#include <unistd.h>

#include <sys/time.h>

#include <sys/wait.h>

int main()

{

system ("getcap /bin/\*");

if (fork() == 0)

{

printf("当前系统时间：\n");

execlp("date", "date", NULL);

}

else

{

wait(NULL);

printf("修改后的系统时间：\n");

execlp("date", "date", "-s", "2020-6-7", NULL);

}

wait(NULL);

return 0;

}

