

**计算机系统安全实验报告**

**实验二：passwd实现细粒度访问控制及root能力位安全应用**

学　　 号 1190201308

班　　 级 1903202

学 生 陈东鑫

日 期 2021年12月1日

**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。**

1、源代码

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pwd.h>

#include <sys/types.h>

#include <sys/stat.h>

void changePassword(const char \*username, const char \*password)

{

    printf("更改 %s 的密码\n", username);

    // 打开待修改文件

    FILE \*fp = NULL;

    fp = fopen("aaa", "r+");

    if (fp == NULL)

    {

        perror("fopen\n");

    }

    // 逐行读取文件内容

    size\_t len = 0;          // 文件总长度

    ssize\_t lineLen;         // 当前行长度

    char \*lineStr = NULL;    // 当前行字符串

    long offset = ftell(fp); // 初始化文件当前偏移位置

    while ((lineLen = getline(&lineStr, &len, fp)) != -1)

    {

        if (strlen(lineStr) == 0)

        {

            continue;

        }

        // 截取每行空格前用户名部分

        char \*lineUsername = strsep(&lineStr, " ");

        // 若匹配，则修改空格后面的密码，剩下行不变

        if (strcmp(username, lineUsername) == 0)

        {

            char buffer[1024] = {0};

            int c = 0;

            // 缓存匹配行后面的部分

            while (!feof(fp))

            {

                buffer[c] = fgetc(fp);

                c++;

            }

            if (c != 0)

            {

                buffer[c - 1] = '\0';

            }

            // 从文件开头查找回到当前行后偏移位置

            fseek(fp, offset, SEEK\_SET);

            fprintf(fp, "%s %s\n", username, password);

            // 向文件写缓存的后面的行

            if (c != 0)

            {

                fprintf(fp, "%s", buffer);

            }

            fclose(fp);

            return;

        }

        offset = ftell(fp); // 更新文件当前偏移，方便只修改指针后边的部分

    }

    printf("passwd: 用户 %s 不存在\n", username);

    return;

}

int main(int argc, char const \*argv[])

{

    // 获得实际用户ID，有效用户ID，保存的用户ID

    uid\_t ruid, euid, suid;

    getresuid(&ruid, &euid, &suid);

    // 根据实际用户uid获得passwd结构体，从而获得实际用户的用户

    struct passwd \*password = getpwuid(ruid);

    char \*ruid\_name = password->pw\_name;

    // 再根据选项数输出被更改密码的账户名

    if (argc == 2)

    {

        printf("%s ", ruid\_name); // 先输出更改密码的用户名

        changePassword(ruid\_name, argv[1]);

    }

    else if (argc == 3)

    {

        if (strcmp(ruid\_name, argv[1]) == 0)

        {

            printf("%s ", ruid\_name);

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

        }

        // 只有root用户能更改其他用户的密码

        else if (strcmp("root", ruid\_name) == 0)

        {

            printf("%s ", ruid\_name);

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

        }

        else

        {

            printf("%s 权限不足\n", ruid\_name);

        }

    }

    else

    {

        printf("用法：user\_passwd [用户名](默认调用用户) [密码]\n");

    }

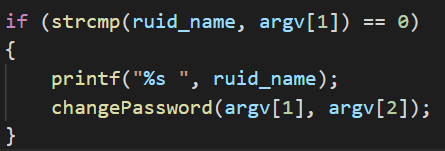
}

2、分析过程

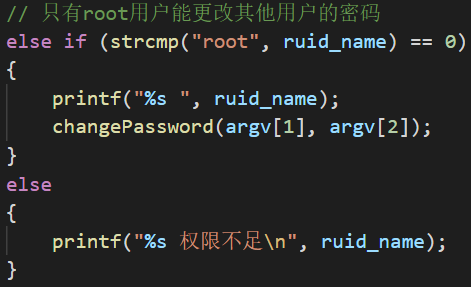
a)首先使用getpwuid通过ruid获得调用的用户的passwd结构体，从而获得用户名，然后根据参数个数采用不同操作。

一个参数：改变自己用户的密码；由于所有用户都有改变自己密码的权限，不需要过多逻辑判断，直接调用changePassword，参数为自己的用户名和参数中的密码。

两个参数：改变第一个参数用户名的密码，密码改为第二个参数；普通用户只有改变自己密码的权限，而root用户有更改所有用户密码的权限，则先比较当前用户名与第一个参数是否相同，若相同则与一个参数相同：



若不同，则只有root用户有权限，若当前用户不是root则提示没有足够权限：



b)changePassword函数：两个参数，第一个参数为待修改用户名，用于在aaa中寻找待修改行，第二个参数为待修改的密码。

首先打开aaa，考虑打开出错情况：



然后getLine逐行读取aaa，使用strsep/strtok函数利用空格分开该行，第一个分割部分为用户名，第二个分割部分为该用户的密码，比较每行的用户名与待修改用户名，若匹配，则准备修改工作。

ⅰ.准备工作：首先保存文件位置指针（在每一次循环会保存上一行的位置指针，则当前循环保存的指针为上一个循环保存的位置，即匹配行之前；找到匹配行后，文件位置指针在匹配行之后），然后利用该指针缓存匹配行后面内容，最后将文件位置指针返回匹配行之前。

ⅱ.开始修改：将“用户名 密码”（第一个和第二个函数参数）写入文件（匹配行之前，写入完成后指针来到匹配行之后），然后将缓存内容写回文件，则aaa只修改了匹配行，而后面的内容没有改变。

若没有找到目标用户名，则系统中没有该用户，打出提示信息。

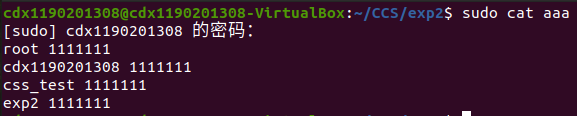
3、实验结果

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

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

IMG_256

创建aaa文件，类似/etc/shadow，用于保存用户名和密码；设置权限如上，只有root用户有读和修改权限，其他用户都不能读和修改。



aaa文件内容每行保存一个用户名和其对应的密码，格式为 [用户名] [密码]。

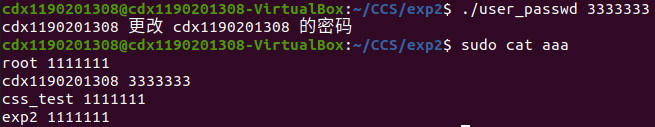
b)编制程序使得：Root用户能够读取和修改aaa文件中所有用户的内容普通用户仅能够读取和修改aaa文件中属于自己用户的内容

编写修改密码程序user\_passwd.c，编译得到user\_passwd，使用方法为passswd ([用户名]) [密码]，修改[用户名]的密码为[密码]，若省略[用户名]，则默认为修改调用user\_passwd的用户。

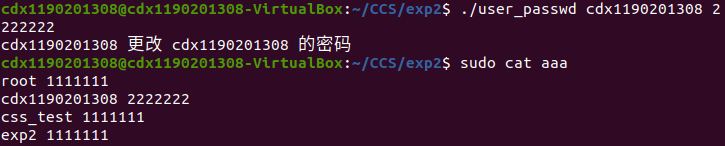
IMG_256

更改user\_passwd的权限如上图，所有用户都有执行权限，设置setuid位，普通用户可以使用root权限调用该程序，修改自己的密码；而root用户可以修改自己和任意用户的密码。

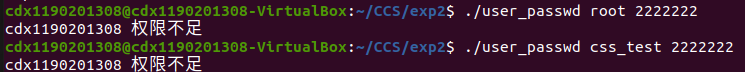
①使用两个参数修改当前账户密码



②使用一个参数修改当前用户密码

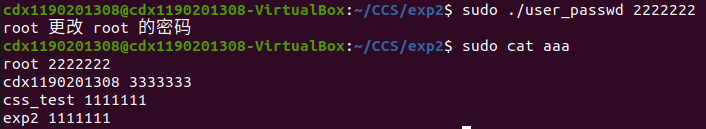


③普通用户尝试修改其他用户密码

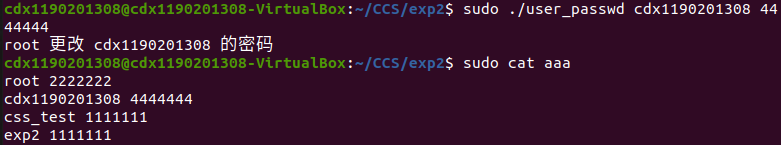


c)普通用户能以root身份执行所编制的类passwd程序

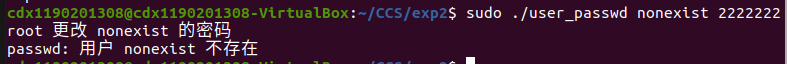
①普通用户以root身份修改自己密码



②普通用户以root身份修改其他用户密码



③修改不存在的用户密码



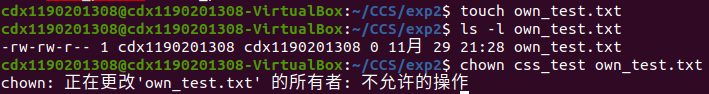
**2.2 利用root的能力机制实现系统加固，有效实现root能力的分发和管理。提供程序比较进行root能力管理前后系统安全性的差异。**

1、学习和理解root 的capability能力位功能。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。以redhat 2.4下的能力为例实现能力位的配置实现。

（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；

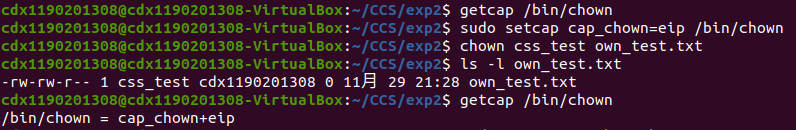
1.CAP\_CHOWN位：修改文件属主的权限

授权前：创建文件own\_test.txt，拥有者为cdx1190201308，现在尝试使用chown将其拥有者修改为css\_test，修改失败。



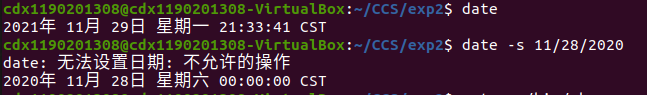
授权后：将cap\_chown的权限授权给chown，再次尝试，发现修改成功。

使用getcap查看chown的权限，发现其权限为cap\_chown+eip。



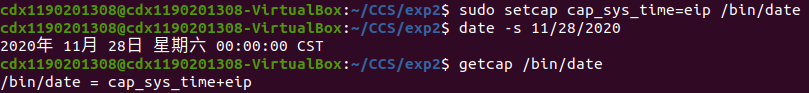
2.CAP\_SYS\_TIME：允许改变系统时钟

授权前：尝试使用date -s 修改时间，修改失败。



授权后：将cap\_sys\_time的权限授权给date，可以修改时间。

使用getcap查看date的权限，发现其权限为cap\_sys\_time+eip。



3.CAP\_DAC\_OVERRIDE：忽略文件的DAC访问限制

授权前：尝试使用普通用户读/etc/shadow，权限不够无法读。

IMG_256

授权后：将cap\_dac\_override的权限授权给cat，使用cat读/etc/shadow文件，发现可以读。

使用getcap查看cat权限，发现其权限为cap\_dac\_override+eip。



（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

a)CAP\_SYS\_MODULE:允许插入和删除内核模块

系统启动时关闭CAP\_SYS\_MODULE能力位，系统故没有修改系统内核模块能力，则敌人不能对系统发动涉及修改内核攻击。所以，对24小时在线运行的服务器来说，是不需要系统升级等修改内核操作的，要去除系统的CAP\_SYS\_MODULE能力。

b)CAP\_LINUX\_IMMUTABLE:允许修改文件的IMMUTABLE和APPEND属性标志

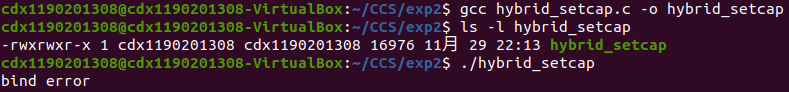
系统启动时关闭CAP\_LINUX\_IMMUTABLE能力位，系统故没有修改文件的IMMUTABLE和APPEND属性标志能力，攻击者不能删除其攻击轨迹、不能安装后门工具、系统日志文件为“append-only” 、系统工具不被删除和修改。

（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。

a)源代码见附件

b)系统网络管理功能：

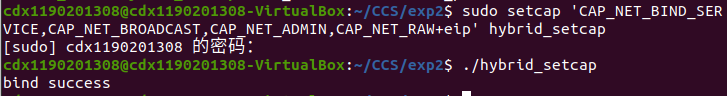
截取实验一的1.1.2的提供http服务部分代码，编写为hybrid\_setcap.c，原理为绑定1024以下端口号需要root权限，而使用普通用户无法绑定，如下图。



使用

sudo setcap 'CAP\_NET\_BIND\_SERVICE,CAP\_NET\_BROADCAST,CAP\_NET\_ADMIN,CAP\_NET\_RAW+eip' hybrid\_setcap

授予hybrid\_setcap权限CAP\_NET\_BIND\_SERVICE（允许绑定到小于1024的端口）CAP\_NET\_BROADCAST（允许网络广播和多播访问），再次调用程序发现绑定成功，并且能够进行网络广播通信。



（4）编制攻击程序，测试能力位的安全性。

a)源代码

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

    while(1)

    {

        sleep(1);

        pid\_t pid;

        if((pid = fork()) == 0)

        {

            execlp("date", "date", "-s", "2020-11-28", (int \*)0);

            return 0;

        }

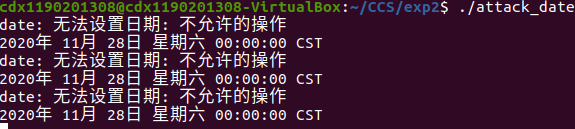
        wait(NULL);

    }

}

b)攻击案例

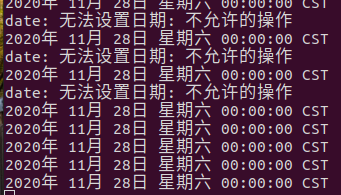
授权date 权限cap\_sys\_time前，程序不断尝试修改日期，修改无效



sudo setcap cap\_sys\_time=eip /bin/date

当我们在另一个终端使用setcap授予date权限后，发现程序尝试成功

IMG_256



**3.心得体会**

通过实验学习了细粒度访问控制的实现方法和机制，及其与粗粒度访问控制的区别，了解了root能力位安全应用方法，并使用其加强对系统的加固，模拟了对能力位防御的攻击以及对攻击的防御，理解了能力位的能力。