



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2020 年秋季学期 计算机学院大三 计算机系安全课程

Lab 2 实验报告

passwd 实现细粒度访问控制及 **root** 能力位安全应用

姓名	余涛
学号	1180300829
班号	1803202
电子邮件	1063695334@qq.com
手机号码	15586430583

1 实验内容

1.1 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。（5 分）

1.1.1 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。

1.1.2 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

aaa 文件只能允许 root 读写，而不允许其他用户读写，所以需要由 root 用户创建，然后不给其他用户读写的权限。aaa 文件的内容中包含超级用户及其密码、普通用户及其密码，具体格式为：

用户名+空格+密码。如下所示：

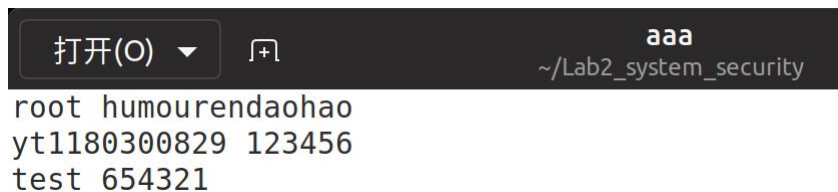
首先创建 aaa 文件并设置权限：

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo touch aaa
[sudo] yt1180300829 的密码:
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l aaa
-rw-r--r-- 1 root root 0 Dec 15 00:51 aaa
yt1180300829@ubuntu:~/Lab2_system_security$ sudo chmod 600 aaa
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l aaa
-rw----- 1 root root 0 Dec 15 00:51 aaa
```

设置超级用户和普通用户的用户名及密码:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo gedit aaa
```

文件内容如下所示:



The screenshot shows a gedit window titled 'aaa' with the path '~/Lab2_system_security'. The file contains the following text:

```
root humourendaohao
yt1180300829 123456
test 654321
```

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo cat aaa
root humourendaohao
yt1180300829 123456
test 654321
```

- 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。

编写一个可以修改密码的程序为 change_password.c 来/usr/bin/passwd 的功能, 具体的功能如下:

- (1) root 用户可以使用”用户名+密码”修改任意用户的密码
- (2) 普通用户可以使用”密码”修改自己的密码, 不能修改别的用户的密码

编译 change_password.c 产生的 change_password 可执行文件需要所有者为 root, 并且设置 setuid 位来实现让普通用户也可以以 root 身份执行 change_password, 具体编译设置过程如下:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo gcc change_password.c -o change_password
[sudo] yt1180300829 的密码:
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l change_password
-rwxr-xr-x 1 root root 17328 Dec 15 00:56 change_password
yt1180300829@ubuntu:~/Lab2_system_security$ sudo chmod 4711 change_password
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l change_password
-rws--x--x 1 root root 17328 Dec 15 00:56 change_password
```

对于 change_password.c, 其首先需要获取进行的 ruid, 这样才能判断执行的用户, 由于获取的仅仅是用户的 ruid 而不是用户的用户名, 所以需要调用 getwuid() 函数来获取 ruid 所对应的用户名:

```

1. //首先获取进程的 ruid, 判断执行的用户
2.     uid_t ruid, euid, suid;
3.     struct passwd* userStruct;
4.
5.     getresuid(&ruid, &euid, &suid);
6.     userStruct = getpwuid(ruid); //由于获取的仅仅是用户的 id, 而不是用户名, 所以需要调用 getpwuid 来获取用户的用户名
7.
8.     printf("该 ruid 的用户名为%s\n", userStruct->pw_name);

```

然后需要根据 argc 来进行参数个数的判断。对于 argc==2, 即参数为一个的情况, 说明是该用户修改自己的密码, 不需要进行权限的判断。对于 argc==3, 即参数为两个的情况, 说明该用户想修改其他用户的密码, 此时就需要判断该用户是否为 root 用户, 如果为 root 用户, 才能允许修改密码, 如果不是, 则设置 errno 位并输出错误信息:

```

1. //根据 argc, 即参数个数进行判断
2.     switch(argc)
3.     {
4.         case 2: //参数只有一个(argc==2)时, 说明只修改该用户自己密码, 无需进行权限判断
5.             changePassword(userStruct->pw_name, argv[1]);
6.             break;
7.         case 3: //参数有两个(argc==3)时, 说明该用户想修改其他用户密码, 需要判断该用户是否为 root
8.             if(strcmp("root", userStruct->pw_name) == 0) //如果该用户为 root, 可以修改密码
9.             {
10.                 changePassword(argv[1], argv[2]);
11.             } else // 如果该用户不为 root, 设置 errno 位
12.             {
13.                 errno = EPERM;
14.                 perror("passwd");
15.             }
16.             break;
17.         default:
18.             errno = EINVAL;
19.             perror("passwd");
20.     }

```

然后对于用来修改密码的函数 `changePassword()`，其两个参数分别为用户名 `username` 和密码 `password`，首先需要使用 `getline()` 函数来按行读取 `aaa` 文件，对每一行需要根据空格拆成空格前的内容和空格后的内容，将空格前的内容与 `username` 比较，如果相同就说明需要修改密码的行为该行。然后从该需要修改密码的下一行起，一直读到文件结束，将所有读到的内容保存在字符串 `after` 中，用于修改密码后再次读入。接着将文件的指针移到需要修改密码的行开头，使用 `"username+空格+password+\n"` 的格式来替换该行，然后将刚才保存的 `after` 字符串接着写入文件，写完后使用 `ftruncate()` 函数来删除后续的内容以防止修改前比修改后长从而导致的修改不完全的问题：

```
1. fp = fopen("aaa", "r+"); //打开 aaa 文件
2. if(errno != 0)
3. {
4.     perror("aaa");
5.     return;
6. }
7.
8. ssize_t bytesNum;
9. size_t n = 0;
10. char* line;
11. long line_start = ftell(fp);
12. while((bytesNum = getline(&line, &n, fp)) != -1) //使用 getline 函数按行读取
    aaa 的文件
13. {
14.     if(strlen(line) == 0)
15.     {
16.         continue;
17.     }
18.     char* currentLineUser = strtok(&line, " "); //然后按照空格拆分每一行
19.     if(strcmp(username, currentLineUser) == 0) //将空格前的内容与用户名
        username 比较，相同时说明需要修改该行
20.     {
21.         char after[1024] = {0};
22.         int c = 0;
23.         while(!feof(fp)) //从密码所在行的下一行一直读到文件尾，将读到的内容保存在
            字符串 after 内，用于修改后再次写入
24.         {
25.             after[c] = fgetc(fp);
26.             c ++;
```

```

27.     }
28.     if(c != 0)
29.     {
30.         after[c - 1] = '\0';
31.     }
32.     fseek(fp, line_start, SEEK_SET); //将文件指针移到密码所在行开头
33.     fprintf(fp, "%s %s\n", username, password); //按照新的 username+空格+
    密码写入并加上换行
34.     if(c != 0) //重新写入刚才保存的 after 内容
35.     {
36.         fprintf(fp, "%s", after);
37.     }
38.     long total_length = ftell(fp);
39.     int fd = fileno(fp);
40.     if(ftruncate(fd, total_length)) //写完后使用 ftruncate 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
41.     {
42.         perror("passwd");
43.     }
44.     fclose(fp);
45.     return;
46. }
47. line_start = ftell(fp);
48. }

```

编译设置权限 setuid 位后执行 change_password:

原始密码文件 aaa 如下:

```

yt1180300829@ubuntu:~/Lab2_system_security$ sudo cat aaa
root humourendaohao
yt1180300829 123456
test 654321

```

情况一: 普通用户为 yt19981119, 修改自己密码成功

```

yt1180300829@ubuntu:~/Lab2_system_security$ ./change_password 777777
该ruid的用户名为yt1180300829
将用户yt1180300829的密码修改为777777

```

```

yt1180300829@ubuntu:~/Lab2_system_security$ sudo cat aaa
root humourendaohao
yt1180300829 777777
test 654321

```

情况二: 普通用户为 yt19981119, 修改 test 用户的密码失败

```

yt1180300829@ubuntu:~/Lab2_system_security$ ./change_password test 222222
该ruid的用户名为yt1180300829
passwd: Operation not permitted

```

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo cat aaa
root humourendaohao
yt1180300829 777777
test 654321
```

情况三：root 用户修改 test 用户的密码成功：

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo ./change_password test 222222
该ruid的用户名为root
将用户test的密码修改为222222
```

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo cat aaa
root humourendaohao
yt1180300829 777777
test 222222
```

c) 普通用户能以 root 身份执行所编制的类 passwd 程序。

由于设置了 setuid 位，所以普通用户也能通过 root 身份执行

change_password 程序，查看 change_password 权限如下：

```
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l change_password
-rws--x--x 1 root root 17328 Dec 15 01:26 change_password
```

1.2 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。

1.2.1 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) 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).

注：

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.

能力位: CAP_SYS_NICE 23(允许提升优先级,设置其它进程的优先级)

{

对于普通用户程序的 NICE 优先级,不能超过 ulimit 对它的限制,如下:

```
nice -n -5 ls
```

```
nice: cannot set niceness: Permission denied
```

而 CAP_SYS_NICE 可以帮助普通用户设置一个想要的一个任意优先级.

```
setcap cap_sys_nice=eip /usr/bin/nice
```

切换到普通用户,指定优先级,如下:

```
nice -n -5 ls
```

```
log mnt mount.c mounttest pacct psacct psacct.c reboot1 reboot1.c  
test
```

```
[root@localhost zy]# setcap cap_sys_nice=eip /home/tttt/test
```

```
[root@localhost zy]# getcap /home/tttt/test
```

```
/home/tttt/test = cap_sys_nice+eip
```

}

1.2.2 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;

1. cap_chown 能力位:

该能力位能允许用户任意修改文件的所有者。

首先创建一个所有者为用户 yt1180300829 的文件 testfile.txt:

```
yt1180300829@ubuntu:~/Lab2_system_security$ touch testfile.txt  
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l testfile.txt  
-rw-r--r-- 1 yt1180300829 yt1180300829 0 Dec 15 02:17 testfile.txt
```

在执行 setcap 修改能力位前,查看 cap 并尝试修改文件的所有者为 root,失败:

```
yt1180300829@ubuntu:~/Lab2_system_security$ getcap /bin/chown  
yt1180300829@ubuntu:~/Lab2_system_security$ chown root testfile.txt  
chown: 正在更改'testfile.txt'的所有者: 不允许的操作
```

执行 setcap 修改能力位,然后将 testfile.txt 的所有者改为 root,成功:


```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo setcap cap_chown=eip /bin/chown
[sudo] yt1180300829 的密码:
yt1180300829@ubuntu:~/Lab2_system_security$ chown root:root testfile.txt
yt1180300829@ubuntu:~/Lab2_system_security$ ls -l testfile.txt
-rw-r--r-- 1 root root 0 Dec 15 02:17 testfile.txt
```

然后查看 cap:

```
yt1180300829@ubuntu:~/Lab2_system_security$ getcap /bin/chown
/bin/chown = cap_chown+eip
```

2. cap_sys_time 能力位:

该能力位允许用户修改系统时钟。

在执行 setcap 修改能力位前, 查看当前的系统时间, 尝试修改日期, 失败:

```
yt1180300829@ubuntu:~/Lab2_system_security$ date
Tue 15 Dec 2020 02:38:36 AM PST
```

```
yt1180300829@ubuntu:~/Lab2_system_security$ date -s 11/23/2024
date: 无法设置日期: 不允许的操作
Sat 23 Nov 2024 12:00:00 AM PST
```

执行 setcap 修改能力位, 然后修改日期, 成功:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo setcap cap_sys_time=eip /bin/date
[sudo] yt1180300829 的密码:
yt1180300829@ubuntu:~/Lab2_system_security$ date -s 11/23/2024
Sat 23 Nov 2024 12:00:00 AM PST
```

然后查看 cap:

```
yt1180300829@ubuntu:~/Lab2_system_security$ getcap /bin/date
/bin/date = cap_sys_time+eip
```

3. cat_dac_override 能力位:

该能力位允许用户忽视文件能力位而设置读写执行文件:

在执行 setcap 修改能力位前, 使用 cat 来读/etc/shadow 文件, 失败:

```
yt1180300829@ubuntu:~/Lab2_system_security$ cat /etc/shadow
cat: /etc/shadow: 权限不够
```

执行 setcap 修改能力位之后, 使用 cat 来读/etc/shadow 文件, 成功:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo setcap cap_dac_override=eip /bin/cat
[sudo] yt1180300829 的密码:
yt1180300829@ubuntu:~/Lab2_system_security$ cat /etc/shadow
root:!:18155:0:99999:7:::
daemon*:18002:0:99999:7:::
bin*:18002:0:99999:7:::
sys*:18002:0:99999:7:::
sync*:18002:0:99999:7:::
games*:18002:0:99999:7:::
man*:18002:0:99999:7:::
lp*:18002:0:99999:7:::
mail*:18002:0:99999:7:::
news*:18002:0:99999:7:::
uucp*:18002:0:99999:7:::
proxy*:18002:0:99999:7:::
www-data*:18002:0:99999:7:::
backup*:18002:0:99999:7:::
list*:18002:0:99999:7:::
irc*:18002:0:99999:7:::
gnats*:18002:0:99999:7:::
```

然后查看 cap:

```
yt1180300829@ubuntu:~/Lab2_system_security$ getcap /bin/cat
/bin/cat = 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。

cap_sys_module: 允许用户能够加载(或卸载)内核模块的特权操作，如果在系统启动时关闭了该能力位，那么恶意攻击者将无法加载和卸载内核模块，在一定程度上保证了系统安全。

cap_linux_immutable: 允许修改文件的 IMMUTABLE 和 APPEND 属性标志，关闭该能力位，攻击者不能删测其攻击轨迹，也不能安装后门工具，系统日志文件为“append-only”，系统工具就不被删除和修改。

(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。

实现系统的网络管理功能。

使用实验一中绑定端口的函数绑定端口到 80 端口即可，将方法封装到 capset.c 文件中，编译该文件：

```
yt1180300829@ubuntu:~/Lab2_system_security$ gcc capset.c -o capset
```

然后执行该文件，由于没有 CAP_NET_BIND_SERVICE, CAP_NET_BROADCAST, CAP_NET_ADMIN, CAP_NET_RAW 这些能力位，所以执行 capset 绑定端口失败：

```
yt1180300829@ubuntu:~/Lab2_system_security$ ./capset
绑定端口80出现错误
```

所以给 capset 设置组合这些能力位后执行 capset:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo setcap 'CAP_NET_BIND_SERVICE+eip CAP_NET_BROADCAST+eip CAP_NET_ADMIN+eip CAP_NET_RAW+eip' capset
[sudo] yt1180300829 的密码:
```

```
yt1180300829@ubuntu:~/Lab2_system_security$ ./capset
绑定端口80成功!
```

然后查看 capset 的能力位:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo getcap capset
capset = cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw+eip
```

也可再去除这些 capset 的能力位:

```
yt1180300829@ubuntu:~/Lab2_system_security$ sudo setcap -r capset
yt1180300829@ubuntu:~/Lab2_system_security$ ./capset
绑定端口80出现错误
```

代码如下:

```
1. #define _GNU_SOURCE
2.
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <string.h>
6. #include <unistd.h>
7. #include <errno.h>
8. #include <netinet/in.h>
9. #include <arpa/inet.h>
10. #include <sys/types.h>
11. #include <sys/stat.h>
12. #include <sys/socket.h>
13.
14. extern int errno;
15.
16. //绑定端口
17. void bindSocket(int port)
18. {
19.
20.     int server_socket = socket(AF_INET, SOCK_STREAM, 0);
21.     if(server_socket < 0)
22.     {
23.         printf("http 服务执行出现错误!\n");
24.         return;
25.     }
26.
27.     struct sockaddr_in server_sockaddr;
28.     memset(&server_sockaddr, 0, sizeof(server_sockaddr));
29.     server_sockaddr.sin_family = AF_INET;
30.     server_sockaddr.sin_port = htons(port);
```

```

31.     server_sockaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
32.     if ((bind(server_socket, (struct sockaddr *)&server_sockaddr, sizeof(server_sockaddr))) < 0)
33.     {
34.         printf("绑定端口%d 出现错误\n", port);
35.     } else
36.     {
37.         printf("绑定端口%d 成功!\n", port);
38.     }
39. }
40.
41.
42.
43. int main(int argc, char const *argv[]) {
44.     bindSocket(80);
45. }

```

(4) 编制攻击程序，测试能力位的安全性。

编写修改系统时间的攻击 `changetime.c`，陷入无限循环持续修改系统时间：由于前面 `/bin/date` 设置了 `cap_sys_time` 能力位，所以可以直接修改系统时间，实现攻击的目的。

```

yt1180300829@ubuntu:~/Lab2_system_security$ gcc changetime.c -o changetime
yt1180300829@ubuntu:~/Lab2_system_security$ ./changetime
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST
Sat 17 Feb 2024 12:00:00 AM PST

```

`Changetime.c` 的代码如下：

```

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <sys/time.h>
4. #include <sys/types.h>
5. #include <sys/wait.h>
6.
7. int main()
8. {
9.     pid_t pid;
10.    while(1)

```

```
11.     {
12.         sleep(1);
13.
14.         if((pid = fork()) == 0)
15.         {
16.             execlp("date", "date", "-s", "2024-2-17", (int *)0);
17.             return 0;
18.         } else
19.         {
20.             int stat_val;
21.             waitpid(pid, &stat_val, 0);
22.         }
23.     }
24. }
```

2 心得体会

通过本次实验，学会了如何给不同的进程或文件分配不同的能力位，掌握了访问控制系统中实现细粒度的权限管理，理解了系统的 `passwd` 程序实现的原理。并且学会了组合多种能力位和通过能力位对系统进行攻击，收获颇丰。