

CMPE455/CMSE456 Security of Computer Systems and Networks

Lab 1- Access Control in Linux

Prepared by Alexander Chefranov and Tansel Sarihan in Spring 2019

Contents

Access Control in Linux	1
1. Task	3
2. Fedora Operating System	3
2.1 Bash Terminal.....	3
2.2 List of Some Important and Useful Commands	3
3. Introduction to access control	5
4. Unix file permissions and inodes	5
Example 1	6
Example 2	6
Example 3	7
Example 4	7
Example 5	8
5. Octal representation of permissions	8
Example 6	8
6. Changing file permissions on Linux system, adding users, groups, changing file owner and group	9
Example 7	9
Example 8	10
Example 9	10
Example 10	11
Example 11	11
Example 12	12
Example 14	12
Example 15	12
Example 16	13
Example 17	13

Example 18.....	13
Example 19.....	14
Example 20.....	14
Example 21.....	14
Example 22.....	15
Example 23.....	15
Example 24.....	15
Example 25.....	16
Challenge 1.....	16
7. Command umask	16
Example 26.....	16
Challenge 2.....	16
8. Set UID (SUID)	17
Example 27.....	17
Example 28.....	17
Example 29.....	18
Example 30.....	18
9. Writing a SUID program in C	18
Example 31.....	18
Example 32.....	19
Example 33.....	20
Example 34.....	21
Example 35.....	21
Challenge 3.....	21
Challenge 4.....	22
Challenge 5.....	22
Challenge 6.....	22
10. Linux Extended ACLs	22
Example 36.....	22
11. Conclusion.....	23
References	23

Our lab material is based on [1]. Section 1 describes the task. Sections 2, 3 briefly introduce the most important Linux commands and access control concepts. You will work with Fedora Linux-based operating system in the laboratories. Sections 4-10 contain examples (prepared in Kali Linux (with alex@lenovo in the screenshots) and Fedora (with linuxlab@asus in the screenshots) operating systems) and challenges. Section 11 concludes the material.

1. Task

Read material below in line with running your own examples repeating Examples 1-36 (screenshots) shown in the lab material (use your own user name instead of alex or linuxlab e.g “chmod 701 /home/your_username/”). There are Challenges 1-6 expected to be solved by you. **The lab will be done individually. You must prepare a report on the work done containing screenshots of your variants of the Examples 1-36, Challenges 1-6 solutions, and explanations.** **Due date for reporting will be announced by assistant. You will run your examples and solutions and explain them answering the questions of the Lab Assistant.**

2. Fedora Operating System

Fedora is a Linux distribution developed by the community-supported Fedora Project. It is an open-source and free. Fedora operating system is user friendly because it is developed in collaboration with the user.

2.1 Bash Terminal

We will use the bash terminal frequently for our experiments. Therefore, let's get to know what we will see when we open the bash terminal.

```
[linuxlab@asus ~]$ █
```

When we open the terminal, we see an image as above. “linuxlab” is our user name, “asus” after the “@” sign is the name of the machine we use, and “~” means the user’s home directory.

```
[root@asus linuxlab]# █
```

Same way, “root” is an username(which has all privileges on the system), “asus” is machine name and “#” means we are logged in as administrator(root).

2.2 List of Some Important and Useful Commands

- Fedora uses dnf package manager. If you want to install any open source program or tool, you need to use dnf. For instance, you need to install a text editor to edit your texts or program codes. A text editor “KWrite” can be installed by using “dnf install kwrite” command on terminal.

```
[linuxlab@asus ~]$ sudo dnf install kwrite  
[sudo] password for linuxlab:
```

- **sudo command** - The sudo command makes it easier to manage your Fedora system. Certain commands in Fedora expect to be run only by a privileged user or administrator. The sudo command lets you run a command as if you're the administrator, known as root. For instance, "sudo chmod 666 somefile".

```
[linuxlab@asus ~]$ sudo chmod 666 somefile
[linuxlab@asus ~]$
```

- **su command** – switch user. For example, "su student" command can be used for switching between current user and student user.

```
[linuxlab@asus ~]$ su student
Password:
[student@asus linuxlab]$
```

- **ls command** – lists directory contents of files and directories. For example, when run "ls /home/linuxlab" command, contents of /home/linuxlab directory will be shown on terminal window.

```
[linuxlab@asus ~]$ ls /home/linuxlab
Desktop    Downloads  Pictures  somefile  Videos
Documents  Music      Public    Templates
[linuxlab@asus ~]$
```

- **cd command** – changes the current directory. For example, "cd Desktop" command changes the current directory to /Desktop directory. (in the same way, "cd .." command can also be used to return to the previous directory.)

```
[linuxlab@asus ~]$ cd Desktop
[linuxlab@asus Desktop]$ cd ..
[linuxlab@asus ~]$
```

- **mkdir** – This command creates a new directory. Example usage for creating a new directory with name "myLabWorks": "mkdir myLabWorks".

```
[linuxlab@asus ~]$ mkdir myLabWorks
[linuxlab@asus ~]$ ls
Desktop    Downloads  myLabWorks  Public    Templates
Documents  Music      Pictures    somefile  Videos
[linuxlab@asus ~]$
```

- **rm command** – this command can be used for deleting some file or directories. For example, we can delete the “test.txt” file with "rm test.txt".

```
[linuxlab@asus ~]$ ls
Desktop  Downloads  myLabWorks  Public  Templates  Videos
Documents Music      Pictures    somefile test.txt
[linuxlab@asus ~]$ rm test.txt
[linuxlab@asus ~]$ ls
Desktop  Downloads  myLabWorks  Public  Templates
Documents Music      Pictures    somefile Videos
[linuxlab@asus ~]$
```

3. Introduction to access control

Access control enforces authorization by determining and enforcing which actions are allowed. Some terminology: a subject is an active entity taking actions, such as a user or program, and an object is the (often passive) resource that can be accessed, such as a file or network resource. Access control mediates subjects' access to objects by enforcing a security policy, limiting which actions are and are not allowed. The policy expresses what is allowed, either formally or informally as a set of rules. An access control mechanism is the code or thing that enforces a policy. An access control model is a way of representing and reasoning about a policy or types of policy.

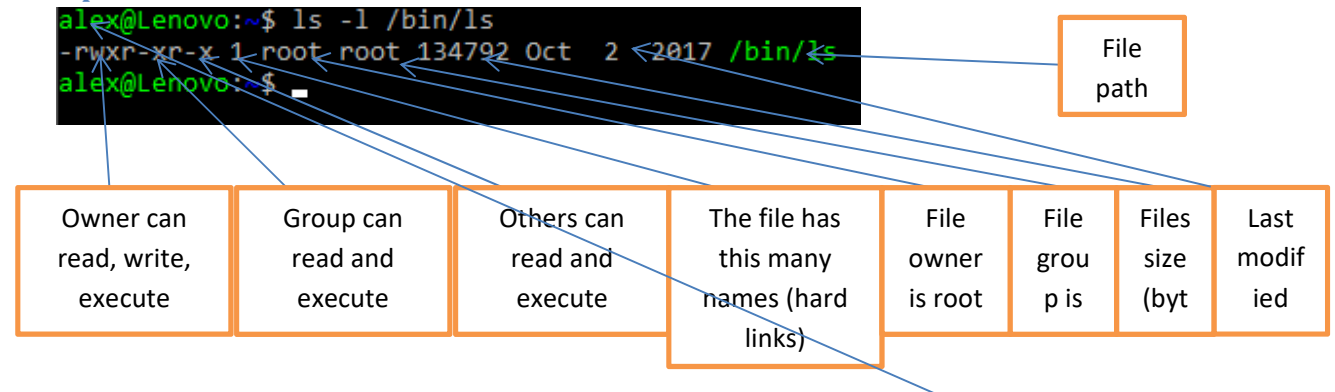
4. Unix file permissions and inodes

The traditional Unix security model is based on the discretionary access control (DAC) model, which enables users to configure who can access the resources that they “own”. Each user can control which other users can access the files that they create. This enables users to grant permissions, without involving a system admin. This is the type of security that has traditionally been built into most consumer OSs such as Windows and Unix. Unix file permissions uses an abbreviated (simplified) form of access control list (ACL). A (full) ACL involves attaching a list of every subject and what they can do to each file (this is how Windows manages file access). For example, a file may have this ACL: “Joe can read, Frank can write, Alice can read, and Eve can read”. Unix simplifies permissions by only defining rules for these three kinds of subjects:

- The user that owns the file (u)
- The file's group (g)
- Other users (o)

ls command is used to display the permissions of the files, -l flag provides detailed output, e.g., ls -l /bin/ls displays permissions of the ls command

Example 1



The meaning for a regular file (as is the case for `/bin/ls`, the first symbol is `-` (dash)):

- **r**: Read the contents of the file
- **w**: Change the contents of the file
- **x**: Execute the file as a process (The first few bytes describe what type of executable it is, a program or a script)

For a directory (the first symbol is `d`):

- **r**: See what files are in the directory
- **w**: Add, rename, or delete files within the directory
- **x**: 'stat' the file (view the file owners and sizes), cd into the directory, and access files within the directory

files within the directory

- **t** (instead of **x**), AKA the "sticky bit": prevents users from deleting or renaming files they do not own

The permissions for each file are stored in the file's inode. An inode is a data structure in Unix filesystems that defines a file. An inode includes an inode number, and defines the location of the file on disk, along with attributes including the Unix file permissions, and access times for the file. View the inode number for this file:

`ls -li /bin/ls`

Example 2

```
alex@Lenovo:~$ ls -li /bin/ls
1970324837335149 /bin/ls
alex@Lenovo:~$
```

File type may be also `l` for links on the files (with the help of the links one and the same data can be accessed using different file names linked to one and the same inode for hard links).

Create a hard link to the `ls` program:

`mkdir /bin/tmp` (create a new directory for hard link)

`ln /bin/ls /bin/tmp/ls`

Now view the details for your new filename, `/bin/tmp/ls`:

`ls -li /bin/tmp/ls`

Create a hard link to some file:

`mkdir tmp` (create a new directory for hard link)

`ln somefile /tmp/somefile_link`

Now view the details for your new filename, `/tmp/somefile_link`:

`ls -li /tmp/somefile_link`

Example 3

```
[linuxlab@asus ~]$ sudo ln /bin/ls /bin/tmp/ls
[linuxlab@asus ~]$ ls -l /bin/tmp/ls
-rwxr-xr-x. 2 root root 157896 May 29 19:33 /bin/tmp/ls
[linuxlab@asus ~]$ ls -l /bin/ls
-rwxr-xr-x. 2 root root 157896 May 29 19:33 /bin/ls
[linuxlab@asus ~]$ ls -i /bin/ls
1311751 /bin/ls
[linuxlab@asus ~]$ ls -i /bin/tmp/ls
1311751 /bin/tmp/ls
[linuxlab@asus ~]$
```

```
~$ ln somefile tmp/somefile_link
~$ ls -l tmp/somefile_link
-rw-r--r-- 2 user user 0 May  7 10:16 tmp/somefile_link
~$ ls -l somefile
-rw-r--r-- 2 user user 0 May  7 10:16 somefile
~$ ls -i tmp/somefile_link
415 tmp/somefile_link
~$ ls -i somefile
415 somefile
```

Thus, we see that the both files share the same inode. Thus, change of one of these files will affect also the other one.

Deleting one of the names simply decrements the link counter. Only when that reaches 0 is the inode actually removed:

```
rm /tmp/somefile_link
```

Deleting one of the file names simply decrements the link counter. Only when link counter reaches 0 is the inode actually removed:

```
rm /bin/tmp/ls
```

Example 4

```
[linuxlab@asus ~]$ rm /bin/tmp/ls
rm: remove write-protected regular file '/bin/tmp/ls'? y
rm: cannot remove '/bin/tmp/ls': Permission denied
[linuxlab@asus ~]$ ls -ld /tmp/
drwxrwxrwt. 14 root root 360 Oct  6 18:38 /tmp/
[linuxlab@asus ~]$
```

You may run into “Permission denied!” Interestingly, in this case as a normal user we can create the link to /bin/ls, but cannot then delete that link since the sticky bit is set for the /tmp/ directory.

```
ls -ld /tmp/
```

Note the “t” in the permissions, and refer to the meaning described above.

You can delete the link as root:

```
sudo rm /bin/tmp/ls
```

Note that in our somefile example, the sticky bit may not be set for the /tmp directory, so we may be able to delete the file with no issues:

```
rm /tmp/somefile_link
```

Example 5

```
~$ rm tmp/somefile_link
~$ dir tmp
~$
```

```
alex@Lenovo:~$ sudo rm /tmp/ls
[sudo] password for alex:
alex@Lenovo:~$ dir /tmp/
alex@Lenovo:~$
```

We see no content in /bin/tmp/ after /bin/tmp/ls removal.

5. Octal representation of permissions

The stat command can be used to display further information from the inode:

```
stat /bin/ls
```

Example 6

```
alex@Lenovo:~$ stat /bin/ls
  File: /bin/ls
  Size: 134792          Blocks: 264          IO Block: 4096   regular file
Device: 2h/2d  Inode: 1970324837335149  Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2018-09-25 10:22:10.000000000 +0300
Modify: 2017-10-02 20:51:20.000000000 +0300
Change: 2018-10-02 16:19:53.572589600 +0300
 Birth: -
alex@Lenovo:~$
```

Unix file
permissions

File's owner

File's group

Look through this information. Note that the output includes the access rights, along with the last time the file was accessed, modified, and when the inode was last changed.

The output from stat includes the format that the information is stored as, along with a more “human readable” output. As we know, user accounts are referred to by UIDs by the system, in this case the UID is 0, as the file is owned by the root user. Similarly, groups are identified by GID, in this case also 0. The actual permissions are stored as four octets (digits 0-7), in this case “0755”. This translates to the (now familiar) human-friendly output, “-rwxr-xr-x”. For now we will ignore the first octet, this is normally 0, we will come back to the special meaning of this later. Each of the other three octets simply represents the binary for rwx, each represented as a 0 or a 1. The first of the three represents the **user**, then the **group**, then the **other** permission. An easy and quick way to do the conversion is to simply remember:

- $r = 4$
- $w = 2$
- $x = 1$

And add them together to produce each of the three octets. So for example, $rwX = \text{binary } 111 = (4 + 2 + 1) = 7$.

Likewise, $r-x = \text{binary } 101 = (4 + 1) = 5$.

Therefore, “-rwxr-xr-x” = 755.

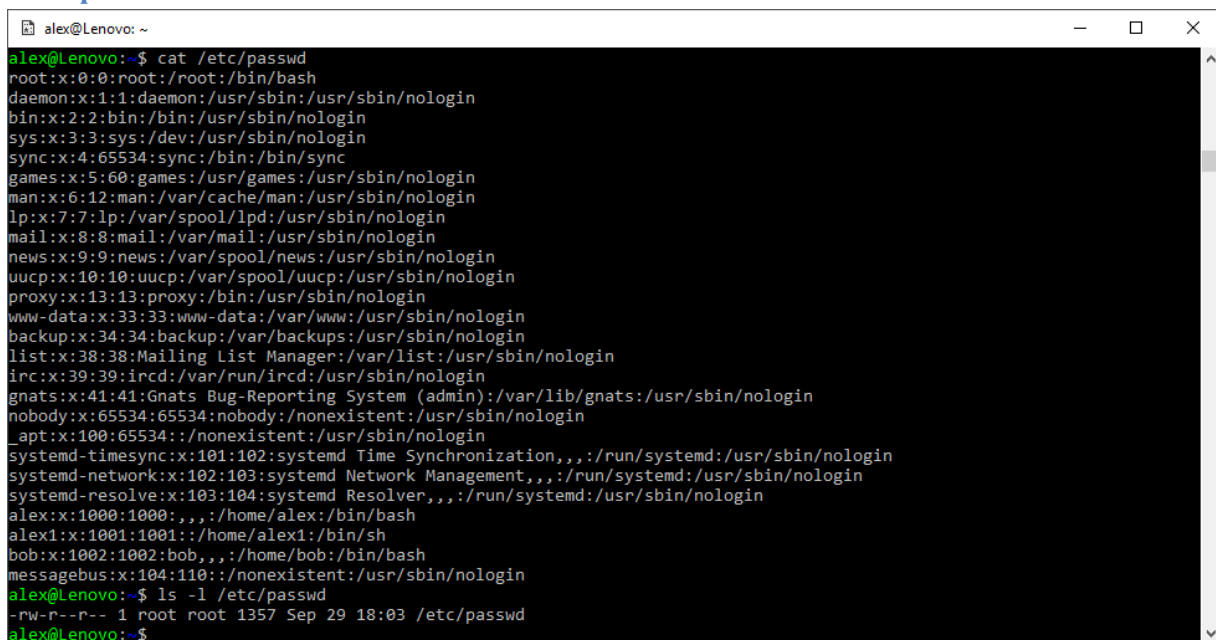
6. Changing file permissions on Linux system, adding users, groups, changing file owner and group

Now, we shall need an opportunity to switch between two users. The list of users can be displayed by

`cat /etc/passwd`

showing contents of the `/etc/passwd` file.

Example 7



```
alex@Lenovo:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
alex:x:1000:1000,,,:/home/alex:/bin/bash
alex1:x:1001:1001::/home/alex1:/bin/sh
bob:x:1002:1002:bob,,,:/home/bob:/bin/bash
messagebus:x:104:110:/nonexistent:/usr/sbin/nologin
alex@Lenovo:~$ ls -l /etc/passwd
-rw-r--r-- 1 root root 1357 Sep 29 18:03 /etc/passwd
alex@Lenovo:~$
```

We see that `/etc/passwd` can be read every user. Let us now create a new user, student with password student. We need for that task to use super user rights with the help of `sudo`. Rights of the users allowed doing `sudo` (from `sudo` group) are enlisted in `/etc/sudoers` file

`cat /etc/sudoers`

Example 8

```
alex@Lenovo: ~  
cat: /etc/sudoers: Permission denied  
alex@Lenovo:~$ sudo cat /etc/sudoers  
[sudo] password for alex:  
#  
# This file MUST be edited with the 'visudo' command as root.  
#  
# Please consider adding local content in /etc/sudoers.d/ instead of  
# directly modifying this file.  
#  
# See the man page for details on how to write a sudoers file.  
#  
Defaults      env_reset  
Defaults      mail_badpass  
Defaults      secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  
  
# Host alias specification  
  
# User alias specification  
  
# Cmnd alias specification  
  
# User privilege specification  
root    ALL=(ALL:ALL) ALL  
  
# Allow members of group sudo to execute any command  
%sudo    ALL=(ALL:ALL) ALL  
  
# See sudoers(5) for more information on "#include" directives:  
  
#includedir /etc/sudoers.d
```

We see that `sudo` group users have the same permissions as `root`. Groups and their members can be viewed by

`cat /etc/group`

Example 9

```
Select alex@Lenovo: ~  
alex@Lenovo:~$ cat /etc/group  
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:alex  
tty:x:5:  
disk:x:6:  
lp:x:7:  
mail:x:8:  
news:x:9:  
uucp:x:10:  
man:x:12:  
proxy:x:13:  
kmem:x:15:  
dialout:x:20:  
fax:x:21:  
voice:x:22:  
cdrom:x:24:alex  
floppy:x:25:  
tape:x:26:  
sudo:x:27:alex  
audio:x:29:  
dip:x:30:alex  
www-data:x:33:  
backup:x:34:  
operator:x:37:  
list:x:38:  
irc:x:39:  
src:x:40:
```

We see that user `alex` belongs to `sudo` group as well as to `adm` (can monitor system tasks), `cdrom` (can access CDROM), and `dip` (can use tools such as `ppp`, `dip` etc. to dial-up connection) groups. A new user is created by

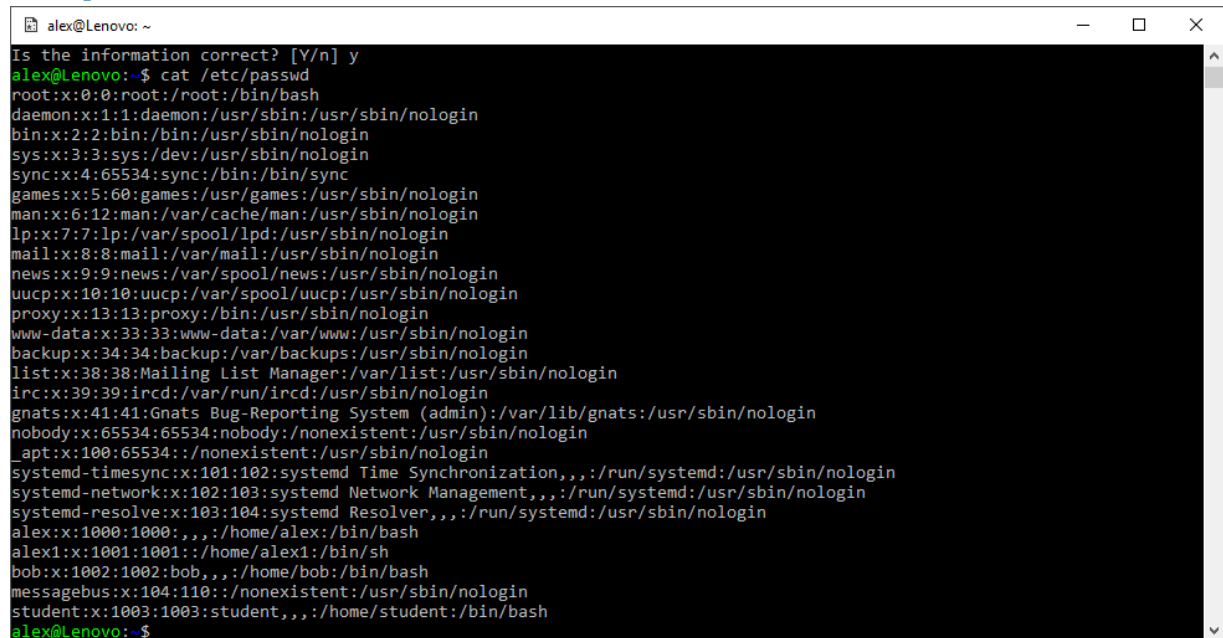
```
sudo adduser student
```

Example 10

```
alex@Lenovo:~$ sudo adduser student
[sudo] password for alex:
Adding user `student' ...
Adding new group `student' (1003) ...
Adding new user `student' (1003) with group `student' ...
Creating home directory `/home/student' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for student
Enter the new value, or press ENTER for the default
    Full Name []: student
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
alex@Lenovo:~$
```

Check that the new user student is in the list of user:

Example 11



```
alex@Lenovo: ~
Is the information correct? [Y/n] y
alex@Lenovo:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
alex:x:1000:1000:::/home/alex:/bin/bash
alex1:x:1001:1001::/home/alex1:/bin/sh
bob:x:1002:1002:bob,,,:/home/bob:/bin/bash
messagebus:x:104:110:/nonexistent:/usr/sbin/nologin
student:x:1003:1003:student,,,:/home/student:/bin/bash
alex@Lenovo:~$
```

Create a file named “mysecret” in your home directory:
cat > ~/mysecret

Example 12

```
alex@Lenovo:~$ cat ? ~/mysecret
cat: '?': No such file or directory
cat: /home/alex/mysecret: No such file or directory
alex@Lenovo:~$ cat > ~/mysecret
It is my secret
Here it is
My secret is here
alex@Lenovo:~$
```

Enter a number of lines of content. Press Ctrl-D when finished entering a “secret” (that others may see). Your first aim is to ensure your “mysecret” file is not visible to other users on the same system. First view the permissions of your newly created file:

```
ls -l ~/mysecret
```

Example 13

```
alex@Lenovo:~$ ls -l ~/mysecret
-rw-rw-rw- 1 alex alex 45 Oct  2 19:24 /home/alex/mysecret
alex@Lenovo:~$
```

Oh no! It’s not so secret...

The chmod command can be used to set permissions on a file. chmod can set permissions based on absolute octal values, or relative changes. So for example, you could use chmod to set permissions on a file based on octet: 770 would give the owner and group rwx, and others no permissions Example:

```
chmod 770 /home/tmp/somefile
```

Example 14

```
alex@Lenovo:~$ mkdir /home/tmp
mkdir: cannot create directory '/home/tmp': Permission denied
alex@Lenovo:~$ sudo mkdir /home/tmp
alex@Lenovo:~$ touch /home/tmp/somefile
touch: cannot touch '/home/tmp/somefile': Permission denied
alex@Lenovo:~$ sudo touch /home/tmp/somefile
alex@Lenovo:~$ ls -l /home/tmp/somefile
-rw-r--r-- 1 root root 0 Oct  2 19:42 /home/tmp/somefile
alex@Lenovo:~$ chmod 770 /home/tmp/somefile
chmod: changing permissions of '/home/tmp/somefile': Operation not per
alex@Lenovo:~$ sudo chmod 770 /home/tmp/somefile
alex@Lenovo:~$ ls -l /home/tmp/somefile
-rwxrwx--- 1 root root 0 Oct  2 19:42 /home/tmp/somefile
alex@Lenovo:~$
```

Thus, the permissions are changed from 644 (rw-) to 770 (rwxrwx---) for /home/tmp/somefile when using sudo command. Or you can make relative changes: u-x would remove the owner (user) the ability to execute the file. Example:

```
chmod u-x /home/tmp/somefile
```

Example 15

```
alex@Lenovo:~$ sudo chmod u-x /home/tmp/somefile
alex@Lenovo:~$ ls -l /home/tmp/somefile
-rw-rwx--- 1 root root 0 Oct  2 19:42 /home/tmp/somefile
alex@Lenovo:~$
```

Likewise, o+w would add other's ability to write to the file
Example:

```
chmod o+w /home/tmp/somefile
```

Example 16

```
alex@Lenovo:~$ sudo chmod o+x /home/tmp/somefile
alex@Lenovo:~$ ls -l /home/tmp/somefile
-rw-rwx--x 1 root root 0 Oct  2 19:42 /home/tmp/somefile
alex@Lenovo:~$
```

Use chmod to give read-write permissions to yourself to your mysecrets file and everyone else no permissions to the file:

```
chmod 660 ~/mysecrets
```

Example 17

```
alex@Lenovo:~$ chmod 660 ~/mysecrets
alex@Lenovo:~$ ls -l ~/mysecrets
-rw-rw---- 1 alex alex 33 Sep 29 12:36 /home/alex/mysecrets
alex@Lenovo:~$
```

Try accessing the file on behalf of the user student:

```
cat /home/alex/mysecrets
```

Example 18

```
alex@Lenovo:~$ su student
Password:
student@Lenovo:/home/alex$ cat /home/alex/mysecrets
cat: /home/alex/mysecrets: Permission denied
student@Lenovo:/home/alex$
```

Create a file "~/myshare", and grant everyone read-write access. Test whether you have correctly set permissions. Also give other necessary permissions to other users for finding file "~/myshare".

Example 19

```
[student@asus linuxlab]$ su linuxlab
Password:
[linuxlab@asus ~]$ touch ~/myshare
[linuxlab@asus ~]$ chmod 666 ~/myshare
[linuxlab@asus ~]$ ls -l ~/myshare
-rw-rw-rw-. 1 linuxlab linuxlab 0 Oct  8 20:44 /home/linuxlab/myshare
[linuxlab@asus ~]$ kwrite ~/myshare
[linuxlab@asus ~]$ cat ~/myshare
It is to be shared with
all the people
[linuxlab@asus ~]$ sudo chmod 701 /home/linuxlab
[sudo] password for linuxlab:
[linuxlab@asus ~]$ su student
Password:
[student@asus linuxlab]$ cat /home/linuxlab/myshare
It is to be shared with
all the people
[student@asus linuxlab]$
```

Create “mygroupshare”, grant only read access to everyone in your group. Test whether you have correctly set permissions.

Example 20

```
[linuxlab@asus ~]$ cat > ~/mygroupshare
It is my group share
People in my group can access it
[linuxlab@asus ~]$ cat ~/mygroupshare
It is my group share
People in my group can access it
[linuxlab@asus ~]$
```

Add user student to group alex by

`sudo usermod -a -G alex student`

Example 21

```
alex@Lenovo:~$ ls -l ~/mygroupshare
-rw-rw-rw- 1 alex alex 54 Oct  3 20:11 /home/alex/mygroupshare
alex@Lenovo:~$ sudo usermod -a -G alex student
[sudo] password for alex:
```

```
alex@Lenovo:~$ cat /etc/group
```

```
alex:x:1000:student
```

```
alex@Lenovo:~$ su student
Password:
student@Lenovo:/home/alex$ cat /home/alex/mygroupshare
It is my group share
People in my group can access it
student@Lenovo:/home/alex$ ls -l /home/alex/mygroupshare
-rw-rw-rw- 1 alex alex 54 Oct  3 20:11 /home/alex/mygroupshare
student@Lenovo:/home/alex$
```

Create a new group mygroup using

`sudo groupadd mygroup`

Example 22

```
[linuxlab@asus ~]$ sudo groupadd mygroup
[sudo] password for linuxlab:
[linuxlab@asus ~]$
```

Change group of mygroupshare from alex to mygroup using

`Sudo chown :mygroup ~/mygroupshare`

To change owner and group of a file use

`chown new_owner:newgroup file`

Example 23

```
alex@Lenovo:~$ sudo chown :mygroup ~/mygroupshare
alex@Lenovo:~$ ls -l ~/mygroupshare
-rw-rw-rw- 1 alex mygroup 54 Oct  3 20:11 /home/alex/mygroupshare
alex@Lenovo:~$
```

Change back mygroupshare group to alex and give alex group only read access to mygroupshare:

`sudo chown :alex ~/mygroupshare`

`sudo chmod g-w ~/mygroupshare`

Example 24

```
alex@Lenovo:~$ sudo chown :alex ~/mygroupshare

alex@Lenovo:~$ ls -l ~/mygroupshare
-rw-rw-rw- 1 alex alex 54 Oct  3 20:11 /home/alex/mygroupshare
alex@Lenovo:~$ sudo chmod g-w ~/mygroupshare
alex@Lenovo:~$ ls -l ~/mygroupshare
-rw-r--rw- 1 alex alex 54 Oct  3 20:11 /home/alex/mygroupshare
alex@Lenovo:~$
```

Create a new group called “staff”, and create a file that you and a fellow classmate (other user) can collaborate on (both edit). Test whether you have correctly set permissions. Both users should be able to edit the file, yet other users should not have write access.

Example 25

```
alex@Lenovo:~$ dir
accessmysecrets accessmysecrets.c acl.txt mygroupshare mysecret mysecrets myshare newfile test
alex@Lenovo:~$ mkdir tmp
alex@Lenovo:~$ dir
accessmysecrets accessmysecrets.c acl.txt mygroupshare mysecret mysecrets myshare newfile test tmp
alex@Lenovo:~$ touch tmp/test1 tmp/test2 tmp/test3
alex@Lenovo:~$ dir tmp
test1 test2 test3
alex@Lenovo:~$
```

Challenge 1.

mkdir test

touch test/test1 test/test2 test/test3

Use a single chmod command to recursively set the permissions for all files contained in the new “test” directory.

Hint: “man chmod”

7. Command umask

Remember that our newly created file started with permissions that meant that everyone could read the file. This can be avoided by setting the **user** file-creation mode **mask** (umask) . Every process has a umask: an octal that determines the permissions of newly created files. It works by removing permissions from the default “666” for files and “777” for new executables (based on a logical NOT). That is, a umask of “000” would result in new files with permissions “666”. A umask of “022” (which is the default value) gives “644”, that is “rw- -r- -r-”. The umask system call can be used to set the umask for the current process.

Check the current umask value:

umask

Example 26

```
alex@Lenovo:~$ umask -s
-bash: umask: -s: invalid option
umask: usage: umask [-p] [-S] [mode]
alex@Lenovo:~$ umask -S
u=rwx,g=rwx,o=rwx
alex@Lenovo:~$ umask -p
umask 0000
alex@Lenovo:~$ umask
0000
alex@Lenovo:~$
```

Challenge 2.

Using the umask builtin command, set your umask so that new files are only rw accessible by you (but not to your group or others):

umask XXX

where XXX is the new umask to use.

Test your new umask value by creating a new file and checking its permissions:

touch newfilename

ls -l newfilename

Do the permissions read “rw-----”? If not, change the umask and try again.

8. Set UID (SUID)

Sometimes a user needs to be able to do things that require permissions that they should not always have. For example, the `passwd` command is used to change your password. It needs read and write access to `/etc/shadow`. Clearly not every user should have that kind of access! Also, the `ping` command needs raw network access... Again not something that every user can do. The Unix solution is set UID (SUID). Using SUID, processes can be given permission to run as another user. For example, when you run `passwd`, the program actually runs as root (on most Unix systems). In fact, every process actually has multiple identities, including:

- The real UID (RUID): the user who is running the command
- The effective UID (EUID): the way the process is treated

Take a look at how the effective UID is specified:

`ls -l /usr/bin/passwd`

Example 27

```
alex@Lenovo:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 63736 Jul 27 11:07 /usr/bin/passwd
alex@Lenovo:~$
```

The “s” in the file permissions means that the file UID will be used as the effective UID. Run `stat /usr/bin/passwd`

Then switch to another user (bob) and run `passwd`

Example 28

```
alex@Lenovo:~$ stat /usr/bin/passwd
  File: /usr/bin/passwd
  Size: 63736          Blocks: 128          IO Block: 4096   regular file
Device: 2h/2d  Inode: 1970324837335145  Links: 1
Access: (4755/-rwsr-xr-x)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2018-09-25 10:22:10.000000000 +0300
Modify: 2018-07-27 11:07:37.000000000 +0300
Change: 2018-09-25 10:23:44.512141000 +0300
 Birth: -
alex@Lenovo:~$ su bob
Password:
bob@Lenovo:/home/alex$ passwd
Changing password for bob.
(current) UNIX password:
```

Digit 4 in the octet permissions 4755 means that the file will be executed with UID=root. Thus, the SUID bit is stored in the first (user) permission octet in the inode.

Viewing the processes running from another bash tab with `ps -af`

Example 29

```
alex@Lenovo:~$ ps -af
UID          PID    PPID  C STIME TTY          TIME CMD
alex           4        3  0  09:32 tty1        00:00:00 -bash
root          46        4  0  10:27 tty1        00:00:00 su bob
bob           47       46  0  10:27 tty1        00:00:00 bash
root          55       47  0  10:36 tty1        00:00:00 su alex
alex          56       55  0  10:36 tty1        00:00:00 bash
root          60       56  0  10:38 tty1        00:00:00 su bob
bob           61       60  0  10:38 tty1        00:00:00 bash
root          64       61  0  10:38 tty1        00:00:00 passwd
alex          66       65  0  10:38 tty2        00:00:00 -bash
alex          83       66  0  10:52 tty2        00:00:00 ps -af
alex@Lenovo:~$
```

we see that passwd runs with UID=root in spite of launched by bob.

Find all programs with SUID in /home/ by

```
sudo find /home -perm -4000 -type f -print
```

and check the findings using stat:

Example 30

```
alex@Lenovo:~$ sudo find /home -perm -4000 -type f -print
[sudo] password for alex:
/home/alex/accessmysecrets
alex@Lenovo:~$ stat accessmysecrets
  File: accessmysecrets
  Size: 17032          Blocks: 40          IO Block: 4096   regular file
Device: 2h/2d  Inode: 6192449487819206  Links: 1
Access: (4711/-rws--x--x)  Uid: ( 1000/   alex)   Gid: ( 1000/   alex)
Access: 2018-09-29 19:28:36.212452600 +0300
Modify: 2018-09-29 19:28:36.328067700 +0300
Change: 2018-09-29 19:54:57.398230400 +0300
 Birth: -
alex@Lenovo:~$
```

The program accessmysecrets is described below.

9. Writing a SUID program in C

You are going to create a SUID program, to grant access to the contents of your “mysecret” file to anyone who runs the program, without sharing direct access to the file. Make sure “~/mysecrets” is only accessible by the owner: ls -la should show “rw-----” for that file.

Example 31

```
alex@Lenovo:~$ chmod 600 mysecrets
alex@Lenovo:~$ ls -l mysecrets
-rw----- 1 alex alex 33 Sep 29 12:36 mysecrets
alex@Lenovo:~$
```

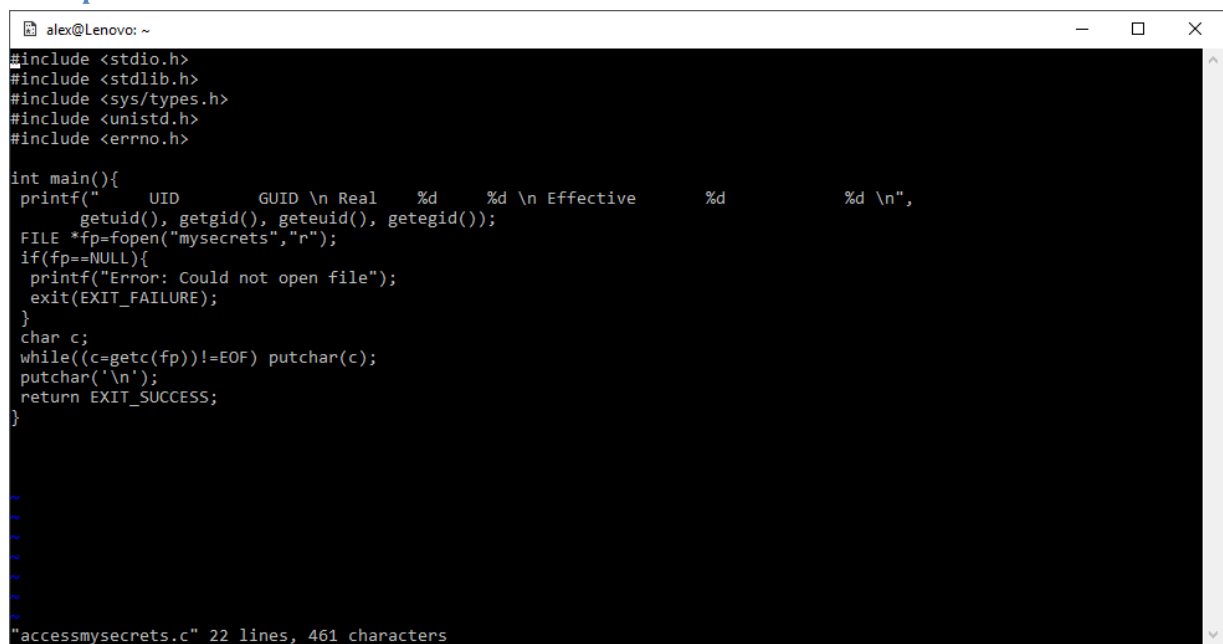
Create a C program by making a new file "accessmysecret.c":

vi accessmysecret.c

Remember, vi is modal. Press "i" to enter insert mode, then enter this code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
int main()
{
printf(" UID GID \n"
"Real %d Real %d \n"
"Effective %d Effective %d \n",
getuid (), getgid (),
geteuid(), getegid());
FILE *fp = fopen("mysecrets", "r");
if (fp == NULL) {
printf("Error: Could not open file");
exit(EXIT_FAILURE);
}
char c;
while ((c=getc(fp)) != EOF) {
putchar(c);
}
putchar('\n');
return EXIT_SUCCESS;
}
```

Example 32

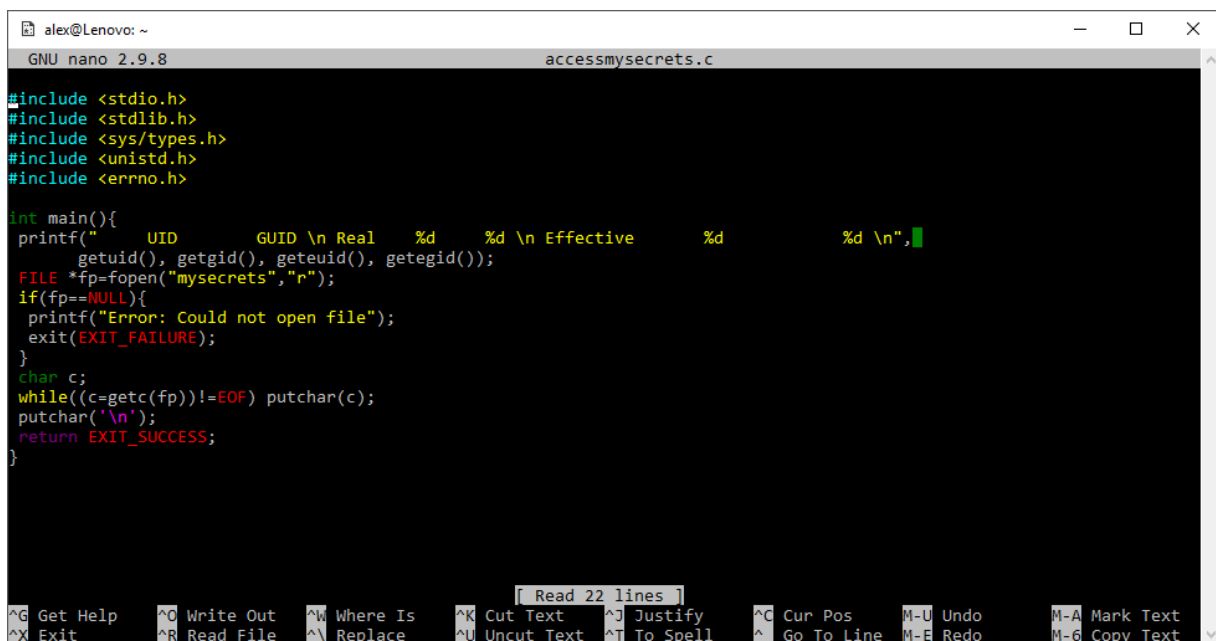


```
alex@Lenovo: ~
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

int main(){
printf("    UID        GUID \n Real    %d    %d \n Effective    %d        %d \n",
      getuid(), getgid(), geteuid(), getegid());
FILE *fp=fopen("mysecrets","r");
if(fp==NULL){
printf("Error: Could not open file");
exit(EXIT_FAILURE);
}
char c;
while((c=getc(fp))!=EOF) putchar(c);
putchar('\n');
return EXIT_SUCCESS;
}
```

"accessmysecrets.c" 22 lines, 461 characters

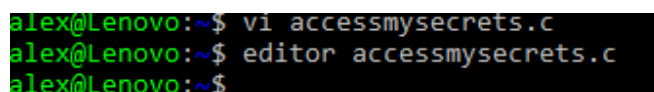
Save your changes and quit (Esc, “:wq”). You may use any other text editor (e.g., editor):



```
alex@Lenovo: ~
GNU nano 2.9.8 accessmysecrets.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

int main(){
    printf("      UID      GUID \n Real      %d      %d \n Effective      %d      %d \n",
           getuid(), getgid(), getuid(), getgid());
    FILE *fp=fopen("mysecrets","r");
    if(fp==NULL){
        printf("Error: Could not open file");
        exit(EXIT_FAILURE);
    }
    char c;
    while((c=getc(fp))!=EOF) putchar(c);
    putchar('\n');
    return EXIT_SUCCESS;
}
```



```
alex@Lenovo:~$ vi accessmysecrets.c
alex@Lenovo:~$ editor accessmysecrets.c
alex@Lenovo:~$
```

Compile the program (which uses the C code to create an executable):

```
gcc accessmysecrets.c -o accessmysecrets
```

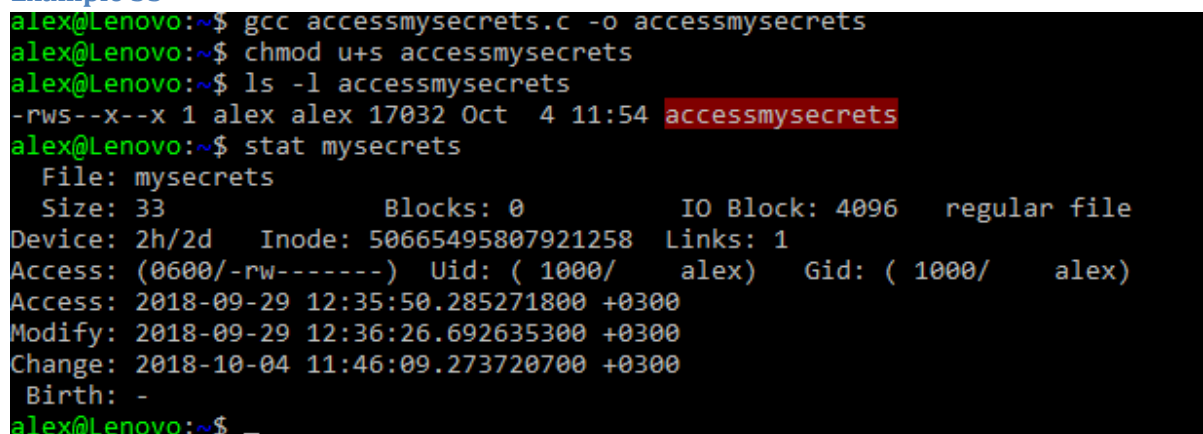
Set the permissions for the file (using chmod) to setuid:

```
chmod u+s accessmysecrets
```

Check the permissions include SUID:

```
ls -l accessmysecrets
```

Example 33



```
alex@Lenovo:~$ gcc accessmysecrets.c -o accessmysecrets
alex@Lenovo:~$ chmod u+s accessmysecrets
alex@Lenovo:~$ ls -l accessmysecrets
-rws--x--x 1 alex alex 17032 Oct  4 11:54 accessmysecrets
alex@Lenovo:~$ stat mysecrets
  File: mysecrets
  Size: 33          Blocks: 0          IO Block: 4096   regular file
Device: 2h/2d  Inode: 50665495807921258 Links: 1
Access: (0600/-rw-----)  Uid: ( 1000/   alex)   Gid: ( 1000/   alex)
Access: 2018-09-29 12:35:50.285271800 +0300
Modify: 2018-09-29 12:36:26.692635300 +0300
Change: 2018-10-04 11:46:09.273720700 +0300
 Birth: -
alex@Lenovo:~$
```

Run the program:
./accessmysecrets

Example 34

```
alex@Lenovo:~$ accessmysecrets
bash: accessmysecrets: command not found
alex@Lenovo:~$ ./accessmysecrets
      UID      GUID
Real    1000      1000
Effective      1000      1000
it is my secret
I want to define
alex@Lenovo:~$
```

Note that the program outputs its real and effective identity.
Change to another user, and execute the program:
/home/yourusername/accessmysecrets

Example 35

```
alex@Lenovo:~$ su student
Password:
student@Lenovo:/home/alex$ /home/alex/accessmysecrets
      UID      GUID
Real    1003      1003
Effective      1000      1003
it is my secret
I want to define
student@Lenovo:/home/alex$
```

Note that the effective ID is that of the owner of the program. You should also see the contents of the mysecrets file, even though you don't have access to the secrets file directly.

Challenge 3

Switch to another user and use the SUID accessmysecrets program to get read access to any one of the owner user's files!

Hint: there is a security problem with this code.

Another hint: think about hard links.

Solution:

There is a security problem caused by not using an absolute filename when opening the file, it opens "mysecrets" rather than "/home/user/mysecrets". Remember, any user can create a hard link to a file (therefore they can make a "copy" of the SUID program wherever they like).

Make a hard link to the SUID program in a directory that the attacker can write to, then also make a hard link to any file the SUID user owns, and name it "mysecrets" in the same directory as the program, then when you execute the program it will write out the contents of the file.

You can exploit this vulnerability as follows:

su - student

In /home/user/accessmysecrets /tmp/access

In /home/user/someotherfile /tmp/mysecrets
/tmp/access

Challenge 4

Modify the program to correct the above vulnerability.

Challenge 5

Modify the program so that only the first line of the mysecrets file is displayed to others.

Challenge 6

Modify the program so that the script checks the UID and only continues for a specific user (for example, if the user is root).

Hint: “man getuid

10. Linux Extended ACLs

We have explored standard Unix permissions. Modern Linux systems (and some other Unix-based systems) now have more complete (and complicated) ACL support. As previously mentioned, an access control list (ACL) is attached to an object (resource) and lists all the subjects (users / active entities) that are allowed access, along with the kind of access that is authorised.

Set a file ACL on your mysecrets file, using the setfacl command:

```
setfacl -m u:student:r ~/mysecrets
```

Example 36

```
[linuxlab@asus ~]$ sudo setfacl -m u:student:r ~/mysecrets
[linuxlab@asus ~]$ getfacl ~/mysecrets
getfacl: Removing leading '/' from absolute path names
# file: home/linuxlab/mysecrets
# owner: linuxlab
# group: linuxlab
user::rw-
user:student:r--
group:---
mask:r--
other:---
```

This grants the “student” user read access to the file.

Note that the stat program is not usually ACL aware, so won't report anything out of the usual:

```
stat ~/mysecrets
```

The ls program can be used to detect File ACLs:

```
ls -la ~/mysecrets
```

```
-rw-r-----+ 1 cliffe users 22 Feb 28 11:47 mysecrets
```

Note that the output includes a “+”. This indicates an ACL is in place.

Use getfacl to display the permissions:

```
getfacl ~/mysecrets
```

Use Linux File ACLs to grant one or more specific users (other class members) read access to your mysecrets file.

Using ACLs, grant any other group (you choose) read-write access to your mygroupshare file.
Remove the group permission you just added.

Example: `setfacl -x g:staff file`

11. Conclusion

At this point you have:

1. Learned about file permissions, hard links, and inodes
2. Learned about octal representations of permissions
3. Changed Unix file permissions to grant access to specific users and groups, using `chmod`
4. Used `umask` to change the permissions applied to new files
5. Learned about Set UID (SUID), become more familiar with C, and compiled a SUID C program
6. You may have also done some more programming of your own
7. Used Linux Extended ACLs to configure more advanced security policies

Well done!

References

1. Z. Cliffe Schreuders. Access controls and Linux/Unix file permissions,
z.cliffe.schreuders.org/edu/ADS/Access%20Controls.pdf