Lab 04 - Access control

Objectives

- Mandatory Access Control
- Discretionary Access Control
- Unix Permissions & ACLs
- Windows ACLs

Access Control Models

An access control system consists of a set of rules regarding whether subjects (e.g. users) are allowed to do an action (e.g. read / write / delete) on the system's objects (e.g. file / application).

There are several models available:

- **Mandatory Access Control** (**MAC**): access policies are controlled by a central authority (e.g. system administrator); example implementations include AppArmor, SELinux and Windows's Mandatory Integrity Control;
- **Discretionary Access Control** (**DAC**): subjects may own objects, allowing them to propagate their access to others; this is the access scheme used by Posix Permissions (chmod), Linux ACLs and Windows File Access Control;
- **Role-based Access Control** (**RBAC**): subjects are allowed access to objects based on their role; this is a flexible generalization that allows the implementation of both MAC and DAC rules; widely used in enterprise applications;

Unix Permissions

00. Setup

- Open a lab VM instance on openstack [https://cloud-controller.grid.pub.ro], image: ***ISC 2020***, flavor: ***m1.small***, availability zone: ***any***;
- Please record your screen using asciinema [https://www.mankier.com/1/asciinema] while working:

```
# install the package
$ sudo apt update -y && sudo apt install -y asciinema
# start recording (optionally, use --append to append to an existing one)
$ asciinema rec [--append] lab04_${LDAP_USERNAME}.cast

# echo your name in the terminal
$ echo "Andrei Popescu"
# work... then:
# stop recording
$ exit

# upload recording
$ ASCIINEMA_API_URL=https://asciinema.cs.pub.ro asciinema upload lab04_${LDAP_USERNAME}.cast
```

- Don't forget to submit the lab's feedback form [https://forms.office.com/r/GZzRJVqQuy]. Double check at form responses [https://ctipub-my.sharepoint.com/:x:/g/personal/mihai_chiroiu_upb_ro/Ee4pZApRKA5Iq9JgR2r652QB0FVL4J9EFtTBtva3jX-1Lw?e=ylQ7lu] ;)

01. Warmup

- Create 3 new users: **bugs**, **daffy** and **tweety**. Create their home directories and set the default shell that the user accounts will log into to /bin/bash.
  - **Hint:** man useradd.
- Check that the users and their home directories were created and to what groups they belong to.
  - **Hint:** /etc/passwd & /etc/group
- Add common names / descriptions for all 3 users:

```
bugs: Bugs Bunny
daffy: Daffy Duck
tweety: Tweety
```

- **Hint:** man usermod, -c.
- Are the user accounts created so far active? No, because no passwords were defined for them. Check out the shadow file, which stores passwords.
  - **Hint:** /etc/shadow.
- You can notice the exclamation mark/s after the usernames. It is a placeholder for the actual password. Once you set the password, the encrypted password will appear here. So, set a password for each of the newly created users. In case your creativity level is low, here are some password examples: bugs4, daffy5, tweety6. Check the shadow file again.
- Create a new group called **friends**, assign the users daffy and tweety to this group and remove the groups daffy and tweety.
  - **Hint:** man groupadd, man usermod

02. Permissions

- Log in as daffy, create a file in daffy's home directory and check out its default permissions.
- Change the file's permissions so that tweety will be able to modify its content but bugs will not be allowed to either modify or see the content.
- Now change it to be the other way: bugs can read and write to the file, while tweety cannot do either.

Remember that, for directories, the execute permission is required as well as read in order for that directory to be browseable and traversable.

03. Special Permissions

In addition to the regular POSIX permissions of read, write and execute there are 3 special permissions. They hold the same place value as the regular permissions and are:

```
SETUID - set user ID on execute
SETGID - set group ID on execute
StickyBit - puts the directory in sticky mode
```

The SETUID and SETGID permissions allow users and groups who are not the owner or group of a file to execute that file as though they were. When the Sticky Bit is set on a directory, only that directory's owner or root can delete or rename the directory's files.

Example: *chmod 4762 myfile* translates to:

```
setuid = on
setgid = off
sticky bit = off
user = read + write + execute
group = read + write
other = write
```

In addition to setting permissions numerically, you can use addition and substraction operators:

```
chmod u+w = add write to *user*
chmod g-rw = remove read and write from *group*
chmod o-rwx = remove read, write and execute from *other*

chmod u+s = add setuid
chmod g-s = remove setgid
chmod o+t = add sticky bit

chmod a+w = add write to *all*
chmod a-wx = remove write and execute from *all*
```

More examples:

```
chmod u=rwx,go=r = set read, write, execute on *user* and read, write on *group* and *other*
chmod go=  = remove all permissions on *group* and *other*
```

Another useful option is **-R**. It allows you to modify objects **recursively**, changing permissions on all objects in a directory and its subdirectories.

```
chmod -R 755 myfolder
```

Tasks

- Log in as daffy and create a folder called daffysfolder. Set the SETUID, SETGID and Sticky Bit permissions. List
- (*ls -l*) the permissions. What do you notice?
- Remove execute for all, but leave the special permissions (SETUID, SETGIT, Sticky Bit). What changes do you notice?
- Remove the special permissions (SETUID, SETGIT, Sticky Bit).
- Assign the minimum group permissions for daffysfolder so that users from other groups can browse the folder and read the files within it.
- Change the owner and group for daffysfolder to *bugs*.

- Log in as tweety and create a folder called tweety_likes_to_share. Set the permissions for this folder in such a way that tweety can share the files with bugs and daffy. This means that bugs and daffy can browse the folder, read the content of any files, but cannot modify, rename or delete any other files than their own.

- As tweety, create a file called sensitivedata.txt and write a line of text in this file. Set no permissions for group and other for this file. Switch users to bugs, and try to read the data from this file using vi. Bugs has permission to run vi, but not to read sensitivedata.txt. So when vi attempts to read the file a "permission denied" error message will be displayed. However, if you set the SUID bit on the vi, bugs is granted access to the file. How does it work? The UNIX system doesn't think bugs is reading file via vi, it thinks "root" is the user and hence the access is granted. Test this and use **ps** to monitor what is going on.

Linux Access Control Lists

Imagine a system with the following users: *student00*, *student01*, *student02*, *student03*, *student04*, *student05* and *student06*. In that system, users *student01* and *student02* are members of a group called *sysop*. The user *student00* creates a new file called *script00.sh*. For this new file, the owner (*student00*) has read, write and execute permissions, the group *sysop* has read and execution permissions, and the rest of the users only have the read permission. Now, we want to give to *student05* the following permissions: read and write (but not execute permission).

With traditional Linux permission we cannot give this particular set of permissions to *student05* because neither as a member of others nor as a member of *sysop* that user would have the desired permissions. Therefore, we need a much more sophisticated system for controlling the permissions for files and directories, Access Control Lists (ACLs), supported by both Windows and Linux.

For Linux, **ACL (Access Control Lists)** provide a finer-grained control over which users can access specific directories and files than do traditional Linux permissions. Using ACLs, you can specify the ways in which each of several users and groups can access a directory or file.

Displaying access permissions

The `getfacl` command displays the file name, owner, group and the existing ACL for a file.

```
student@isc-v2:~$ getfacl my-script.sh
# file: my-script.sh
# owner: student
# group: student
user::rw-
group::rw-
other::r--
```

Setting ACLs of files

The `setfacl` command sets ACLs of files and directories. The `-m` option adds or modifies one or more rules in a file or folder's ACL.

```
setfacl -m ugo:user_or_group_name:permissions file_or_folder_name
```

Examples:

```
    setfacl  -m  u:student04:7  script00.sh  => Adds (or modifies) a rule to the ACL for the script00.sh file that gives student04 read, write and execute permissions to that file.
    setfacl  -m  u:student04:rw-  script00.sh  => Adds (or modifies) a rule to the ACL for the script00.sh file that gives student04 read and write and execute permissions to that file.
    setfacl  -m  g:sysop:r-x  script00.sh  => Adds (or modifies) a rule to the ACL for the script00.sh file that gives sysop read and execute permissions to that file.
    setfacl  -m  o::6  script00.sh  => Adds (or modifies) a rule to the ACL for the script00.sh file that gives others read and write permissions to that file.
    setfacl  -m  u:student04:rx  script00.sh  => Adds (or modifies) a rule to the ACL for the script00.sh file that gives student04 read and execute permissions to that file.
    setfacl  -m  u:student04:rx  folder00 => Adds (or modifies) a rule to the ACL for the folder00 folder that gives student04 read and execute permissions to that folder.
    setfacl  -m  u:student06:5  script00.sh folder00 => Adds (or modifies) a rule to the ACL for the folder00 folder and file  script00.sh that gives student06 read and execute permission
```

Removing rules

The `-x` option removes rules in a file or folder's ACL.

Examples:

```
    setfacl  -x  u:student04  script00.sh  => Removes a rule that gives student04 permission to access the files script00.sh.
    setfacl  -x  g:sysop  script00.sh  => Removes a rule that gives sysop permission to access the files script00.sh.
    setfacl  -x  u:student04  folder00 => Removes a rule that gives student04 permission to access the folder foldert00.
    setfacl  -x  u:student06:5  script00.sh folder00 => Removes a rule that gives student06 permission to access the folder folder00 and the file script00.sh.
```

01. Setup

Create 2 additional users: **alice**, and **bob**. Create the group **nice-people** and add both *alice* and *bob* to it.

02. Getfacl

Create a folder called *important-files* in the home folder of the user *student*. Display the ACL of *important-files*. At the moment, are there any differences between using `ls -la` and `getfacl`?

03. Setfacl

Login as **alice**.

- Try to add a new folder called *alice-files* inside *important-files*. Can you create this folder? Why?
- Add a new rule to the ACL of the folder *important-files* that gives **alice** read, write and execute permissions to that folder.
- Display the ACL of the *important-files* folder. Display the permissions of *important-files* using `ls -la`. Do you see anything different?
- Login again as **alice** and try again to create the *alice-files* folder. Did it work?

04. Test ACLs

Login as **bob**.

- Try to create a file called *bob.txt* in the *alice-files* directory. Did it work?
- Add a rule to the ACL of the folder *alice-files* that gives to the **nice-people** group read, write and execute permissions to that folder. Try again to create the *bob.txt* file.

05. More rules

Login as **alice** and create a file called *alice.txt* in *alice-files*.

- Login as **bob** and try to modify *alice.txt*. Did it work?
- Add a rule specifying that any file created in the *alice-files* directory can be modified by the group *nice-people*.
- As **alice**, create a new file named *alice2.txt*. Can **bob** modify this file?

06. Removing ACLs

- Remove all the rules related to the **nice-people** group in the ACL of the *alice-files* directory.
- Login as **bob** and check if you can still modify *alice2.txt*.
- Remove the ACL of the *alice-files* directory.

Windows

Reboot to Windows.

For online lab: watch the presentation.

01. Setup

Create users **jack**, **john**, **outsider**. The password should be "student". Create a new group called **jgroup** and add **jack** and **john** to it.

Download this file hierarchy: movies.zip and extract it to "C:\Users\Public". List the content of Movies.

- **Hint:** net user, net localgroup

02. Windows ACLs (cacls)

Change the permisions for *Up* so that **outsider** has full permissions and **jack** has only read permissions. Log in as Jack. Is he able to edit *Up\Carl.txt*?

- **Hint:** cacls
- **Hint**: For testing, you can use runas (the equivalent of "su") to quickly start a program as another user, e.g.:

```
runas /User:jack cmd.exe
```

03. Setting ACLs

Edit the permissions for *Storks* recursively in such a way that **outsider** has no access. Login as **outsider** and check if he is unable to access the content of *Storks*.

04. Complex ACLs

Grant full rights to **jgroup** for *Zootopia*. Edit the rights for *Zootopia\Judy.txt* so that only **jack** can write and **john** to read, and for *Zootopia\Nick.txt* so that only **john** can write and **jack** to read. Check if the commands were correct.

11. [10p] Feedback

Please take a minute to fill in the feedback form [https://forms.gle/5Lu1mFa63zptk2ox9] for this lab.

isc/labs/04.txt · Last modified: 2021/03/28 20:20 by florin.stancu