


Lab 04 - Access control

Objectives

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

Preparation

You may use the UPB's  OpenStack cloud to instantiate a Virtual Machine to be used for this lab! [Read these instructions if you wanna know how!](#).

Overview

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

Standard Linux access control is mainly done at the filesystem level by using a couple of bit flags (read / write / execute) representing actions allowed for a specific subject (user / group / others) to do on an object (filesystem nodes).

Standard UNIX actions are **Read**, **Write** and **Execute**, though their effects are different between files and directories:

- **read**: allows opening for reading a file; for directories, allows listing their contents (e.g., `ls`);
- **write**: allows opening a file for writing; for directories, allows creating & deleting its children;
- **execute**: allows executing a file; for directories, allows changing working directory inside (e.g., `cd`).

The  verification algorithm is also simple:

- check if the user is the **owner** of the file; if true \Rightarrow apply the permissions in the owner slot;
- check if the user is in the file's **group**; if true \Rightarrow apply the permissions in the group slot;
- else: apply permissions in the **others** slot!



Fun fact: if you have `077` (no permissions for owner, all permissions for group + others) and the owner try to actually read or write the file, the command fails :| others will have full access!

The basic tool to read permissions is `ls -l`, and for altering them: `chmod`. To change the owner / group of a filesystem object, use `chown`:

```
# Read The Friendly Manuals:
man chmod
man chown
```

Special Permissions & Examples

In addition to the regular POSIX permissions of read, write and execute there are 3 special permissions (available using an additional 3-bit subject structure). 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 subtraction 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,gor+ = 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
# same goes for chown:
chown -R student:teachers myfolder
```

Linux Access Control Lists

Imagine a system with the following users: *mike*, *dave*, *john*, *steve*, *mark*. In that system, users *mike* and *dave* are members of a group called *sysop*. The user *mike* creates a new file called *runme.sh*. For this new file, the owner (*mike*) 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 *mark* the following permissions: read and write (but not execute permission), but not to the others!

With traditional Linux permission we cannot give this particular set of permissions to *mark* 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-vm:~$ getfacl runme.sh
# file: my-script.sh
# owner: mark
# group: sysop
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> PATH...
```

Examples:

```
setfacl -mf u:mark:7 runme.sh # => Adds (or modifies) a rule to the ACL for the runme.sh file that
setfacl -m u:mark:rw- runme.sh # => Adds (or modifies) a rule to the ACL for the runme.sh file that
setfacl -m g:sysop:r-x runme.sh # => Adds (or modifies) a rule to the ACL for the runme.sh file th
setfacl -m o:6 runme.sh => Adds (or modifies) a rule to the ACL for the runme.sh file that gives
setfacl -m u:mark:rx runme.sh # => Adds (or modifies) a rule to the ACL for the runme.sh file that
setfacl -m u:mark:rx myfolder/ # => Adds (or modifies) a rule to the ACL for the folder00 folder t
< >
```

Removing rules

The `=x` option removes rules in a file or folder's ACL, e.g.:

```
setfacl -x u:mark runme.sh => Removes mark's rule on runme.sh
```

Tasks

00. Setup

All tasks will be solved inside a Docker container (available on Docker Hub):

```
docker pull ropubisc/acl-lab # to update image
docker run --rm --name acl-lab -it ropubisc/acl-lab # to run the container
```

If you wish to open multiple terminals inside the same container, find the container's name and use `docker exec`:

```
docker container ls
# note the container's name or hash -> copy it!
docker exec -it "<CONTAINER_ID>" bash
```



Since you are running the containers locally, you may be tempted to cheat by entering using the root user...

This defeats the purpose of the lab, so: **don't do that!**

[25p] 01. Security through obscurity

- Open the container. Try to read the files in `/etc/secret/`. There is a flag in there... can you read it?
- Go to `/usr/local/isc/`. There is a **very hidden** file made up of **numbers!** Can you try to guess it?
 - *Hint: you may want to filter the output a bit.. stderr redirection, maybe?*
- Finally, run `giff-me-flag`
 - *Hint 1: no execute bit – read the other tasks, you are allowed to use any existing accounts ;)*
 - *Hint 2: it expects a secret in argv[1]!... can you "reverse engineer" its strings?*
- Total: **3 flags!**

[25p] 02. The old userswitcheroo

- Inside the container, you have many existing users!
- One has the password `hunter2`. The others have further instructions (text files) inside their home directories!
- Main objective: read the flag inside `/home/.not_for_your_eyes` by using the good ol' `su*` commands!
 - *Hint: yeeep, just listen to the .txts and search through sudo's manual pages, you can't become root no matter how hard you try!*
 - *Hint: you may need to do some unusual "path traversals"!*
- Total: **1 flag!**

[25p] 03. Specials

- Go back as being the hacker!
- Retrieve the flag from `t413nt`'s home directory!
 - *Hint: use your mad Python skillz :P*
 - *Hint: code injection! try to simulate the resulting value of `expr` (on a notepad)!*
- Total: **1 flag!**

[25p] 04. Linux ACLs

- Enter as `student` (inside the container, `ofc!`); guess the password!
- Run `copy-t3h-f14gz` – it's not working properly.. fix the permissions (*no source core? you should have no problems with it :P*)!
 - *Hint: "reverse engineer" it, again!*
- Total: **2 flags!**