Access control

Marco Prandini
DISI
Università di Bologna

Access control basics

- In principle, access control is a decision on whether a ***subject*** can perform a specific ***operation*** on an ***object***
- The most naive mode to express an access control policy would be a complete matrix

Subject / Object	User1	User2	Group3	...	...
File1	read	read write	--	...	...
Dir2	list	modify	--	...	...
Socket3	write	--	read	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Access control matrix partitioned

- Thousands of subjects, millions of objects!
- Most "cells" set to a default value → could be omitted
- Two ways to save space
  - Partition the matrix by subject: *capability lists*
    - A list for each subject
    - It contains only the objects for whom the subject has permissions ≠ default
  - Partition the matrix by object: *access control lists* (ACL)
    - A list for each object
    - It contains only the subjects having permissions ≠ default on the object
    - Explicitly implemented by POSIX and MS Windows
    - Also standard Unix filesystem metadata is an ACL which is limited to list only three subjects
      - The owner user
      - The owner group
      - The implicit group containing all the other users

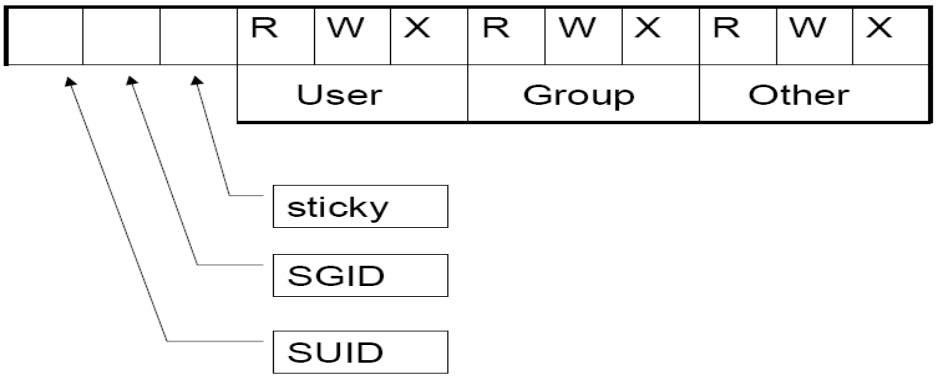
      and the respective permissions

Access control models

- There are two basic access control models
  - DAC (Discretionary access control):
    - Each object has an owner
    - The owner sets permissions
    - ACL are typical of this model
  - MAC (Mandatory access control):
    - There is a centralized policy set by a security manager
    - Ownership of an object does not allow to set permissions
    - Typically expressed as some kind of capability lists in a policy file
- There are more complex models
  - RBAC (Role-based access control):
    - Permissions are granted to *roles*
    - Useful especially if users can dynamically assume different roles based on context (what they are doing, environmental elements, time, …) - often referred to as Rule-based RBAC

Unix Filesystem Authorization

- Each file (regular, directory, link, socket, block/char special) is stored in an i-node
- a set of authorization informations, among other things, is associated to each i-node
  - (exactly one) User owning the file
  - (exactly one) Group owning the file
  - a set of 12 bits representing access rights and special modes



Authorization bits meaning

- The meaning is slightly different for files and directories:

R = read the contents
read a file
list a directory

W = modify contents
write in a file
add/remove directory entries

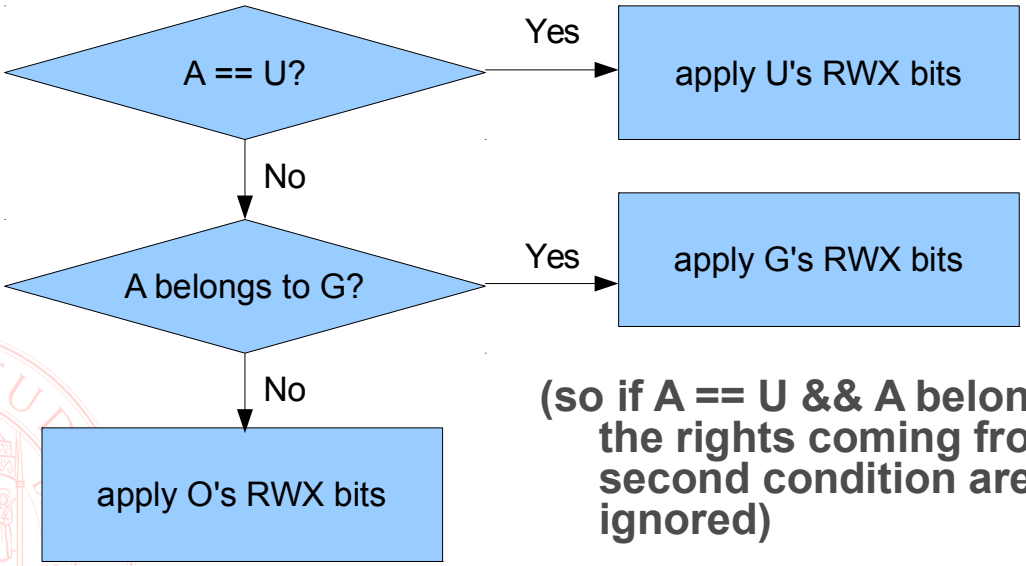
X = execute
run a file as a program
enter a directory

NOTE that 'W' rights on a directory allow a user to delete files on which he/she may not have any right!

NOTE: access to a file is conditioned to having the 'X' right on every directory in its path. 'R' is not needed, unless one needs to discover the file name before accessing it

Rights composition

- When a user “A” wants to perform some operation on a file, the operating system checks authorization rights according to this procedure:



(so if A == U && A belongs to G the rights coming from the second condition are actually ignored)

Control of default permissions

- Every newly created file has:
  - the creator as owner user
  - the active group of the creator as owner group
    - default
    - can be voluntarily changed
    - can be automatically changed (see special bits for directories)
  - permission bits = “all relevant bits” - umask
    - “all relevant bits” = rw-rw-rw- for files, rwxrwxrwx for directories
    - the *umask* then is a bit mask removing unwanted rights
    - since in Linux the default group for a user contains only the user, a safe umask is 006 (remove r and w from others' rights)
      - this is a group-friendly setting that comes in handy when working in collaboration areas of the filesystem
    - check the umask and learn how the shell automatically configures it

Special mode bits / files

The three most important bits (11, 10, 9) indicate a special behaviour related with the owner user, owner group and others respectively

Again, the behaviour is different for files and directories, in a more peculiar way, so let's examine the two cases independently, starting with files

- BIT 11 – SUID (Set User ID)
  - if set on an program, makes the OS create a process upon its execution that runs as the user owning the file instead of the user starting it
- BIT 10 – SGID (Set Group ID)
  - same as SUID, but in this case the group identity of the process is altered
- BIT 9 – STICKY
  - OBSOLETE, suggested the OS to cache a copy of the program

Look for troublesome permissions

- SUID and SGID files are a convenient way to implement front-ends to privileged tasks
  - changing one's password
  - manipulating mail spools
  - etc.
- They must be kept under special surveillance, because anyone that can run them gains higher privileges than what is normally granted
- Periodically use *find* to look for them in the system

  find / -type f -perm +6000

- Other interesting searches are
  - for files that are world-writable (-perm +2)
  - for unowned files, left from deleted accounts (-nouser)

Special mode bits / directories

- Bit 11 is not meaningful for directories
- Bit 10 – SGID
  - if a user belongs to many groups, including the group owning the directory, the latter will be assumed as the active one when the user operates inside the directory when this special bit is set.
  - this allows the creation of "collaboration areas" where user belonging to a common group enjoy the automatic assignment of proper ownership to files they create, without sacrificing the more private default setting when they work in other environments (i.e. the files are normally owned by the group containing the user only)
- Bit 9 – Temp
  - temporary directories, i.e. world-writable places available for various purposes, have an issue: anyone can delete their contents
  - if this bit is set, only the owner of a file can delete it from the directory

Access limitations on filesystems

- Set up separate partitions for storing data vs. system activities
  - prevent the execution of programs, especially with setuid bit
  - prevent the usage of device files or setuid bit

```
/dev/sda5     /nas          ext4     defaults,nosuid,nodev,noexec 1 2
```

- Do the same for world-writable partitions such as `/tmp`

Manage permissions

chmod is used to modify permissions

Symbolic mode:

chmod 'a=r,g-rx,u+rwx' file,

→ "all" can exactly read, block "group" from reading and executing, add to owner read, modify and execute

Numeric mode (octal)

chmod 2770 miadirectory

2770 octal = 010 111 111 000 binary = SGID rwx rwx ---

chmod 4555 miocomando

4555 octal = 100 101 101 101 binary = SUID r-x r-x r-x

Attributes

- Attributes are mainly intended for fs tuning...
  - compressed (c), no dump (d), extent format (e), data journalling (j), no tail-merg-ing (t), undeletable (u), no atime updates (A), synchronous directory updates (D), synchronous updates (S), and top of directory hierarchy (T)
- … but a few are security-relevant
  - append only (a) – good for logfiles
  - immutable (I) – forbids deleting, linking to, renaming, writing to this file; good for critical system files
  - secure deletion (s) – be warned: only zeroes the bytes
- Tools
  - chattr to change the attributes
  - lsattr to display them

POSIX Access Control Lists

- ACL extend the authorization possibilities on the filesystem
- Main advantages:
  - specify an arbitrary list of users and groups with associated permissions, in addition to the owner-user and owner-group
  - inherit the creation mask from the containing directory
  - limit all the granted permissions at once
- Example:

```
user::rw-
user:lisa:rw-          #effective:r--
group::r--
group:toolies:rw-      #effective:r--
mask::r--
other::r--
```

- Tools:
  - `setfacl` to set, `getfacl` to view (ls -l shows a '+' after the permission string if ACL are present for a file)
  - `man acl`