Unix Access Control

Sicurezza (CT0539) 2023-24
Università Ca' Foscari Venezia

Riccardo Focardi
www.unive.it/data/persone/5590470
secgroup.dais.unive.it

Università
Ca'Foscari
Venezia

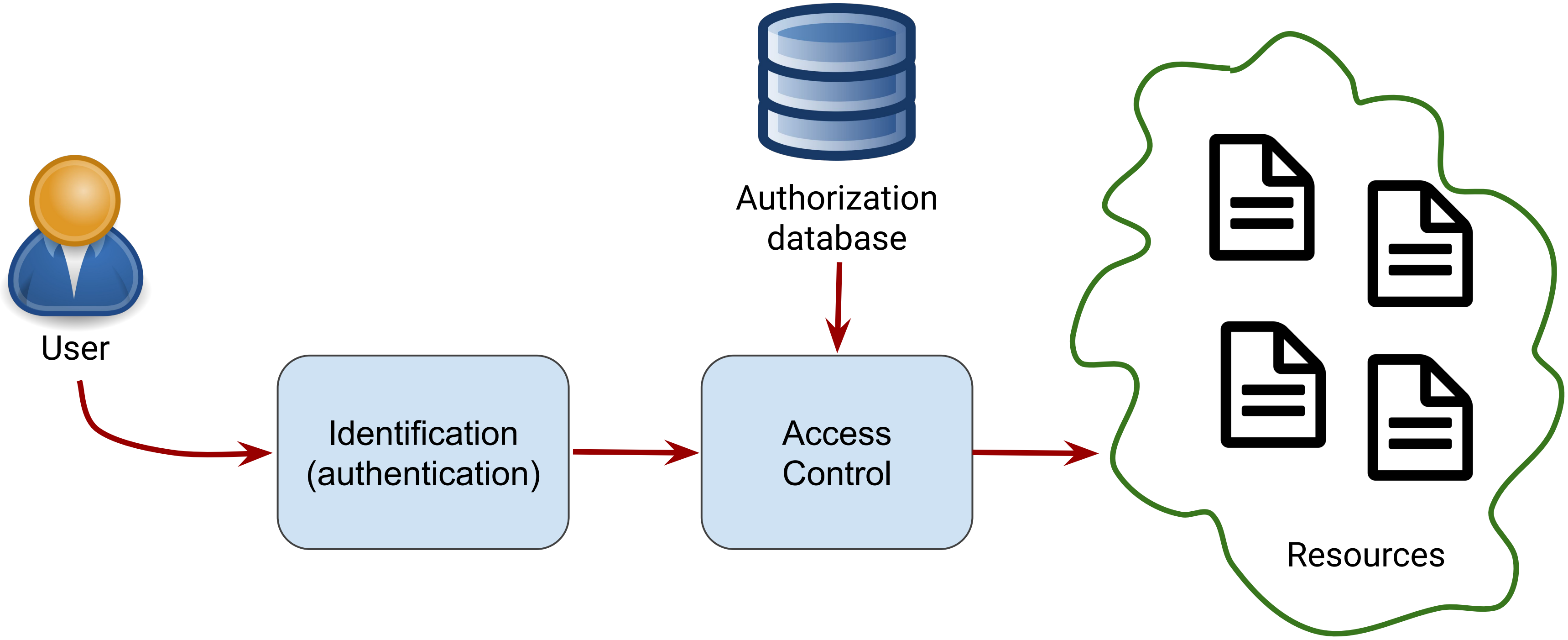
Definition

RFC 4949
Internet Security Glossary

Access Control: *Protection of system resources against unauthorized access*

- The process regulating the use of **system resources** according to a **security policy**
- Access is permitted only by **authorized** entities (users, programs, processes, or other systems) according to that policy.

Access Control



Access control policies (1)

Discretionary access control (DAC): based on the identity of the requestor and on **access rules** (authorizations) stating what requesters are allowed to do

- **Discretionary**: an entity might enable another entity to access some resource

Mandatory access control (MAC): **security labels** indicate how sensitive is a resource while **security clearance** indicate system entities access level

- **Mandatory**: an entity that has clearance to access a resource may not enable another entity to access that resource

Access control policies (2)

Role-based access control (RBAC): based on the **roles** that users have within the system and on rules stating what accesses are allowed to users in given roles

- **Example**: a doctor can access patient's medical data while an administrator can access patient's anagraphic data

Attribute-based access control (ABAC): based on attributes of the user, the resource to be accessed, and current environmental conditions

- **Example**: access to a movie might depend on the kind of subscription, the movie category, possible promotional periods, etc ...

Subjects and objects

Subject: is an entity capable of accessing resources (objects)

- Any user or application actually gains access to an object by means of a **process**
- The process **inherits** the attributes of the user, such as the access rights

Object: is a resource to which access is controlled. An object is an entity used to contain and/or receive information

Examples: pages, segments, files, directories, mailboxes, messages, programs, communication ports, I/O devices.

Access rights

Read: Subject may view information in an object; read access includes the ability to copy or print

Write: Subject may add, modify, or delete data in an object

Execute: Subject may execute an object (e.g. a program)

Delete: Subject may delete an object

Create: Subject may create an object

Search: Subject may search into an object (e.g., a query giving a partial view of the content)

Note: one access right might imply another one, e.g. read $\Rightarrow$ search

Access Matrix

Access matrix: access rights for each subject (row) and object (column)

	README.txt	/etc/shadow	Carol.pdf	/bin/bash
Alice	Read Write	Read Write		Read Write Execute
Bob	Read			Read Execute
Carol	Read		Read Write	Read Execute

NOTE: can be **sparse**!

Access control lists vs. capabilities

Access Control List (ACL): for each object lists subjects and their permission rights (decomposition **by columns**)

- Easy to find which subjects have access to a certain object
- Hard to find the access rights for a certain subject

Capabilities: for each subject, list objects and access rights to them (decomposition **by rows**)

- Easy to find the access rights for a certain subject
- Hard to find which subjects have access to a certain object

Example: ACL

README.txt:

Alice: Read, Write;

Bob: Read;

Carol: Read.

/etc/shadow:

Alice: Read, Write.

	README.txt	/etc/shadow	Carol.pdf	/bin/bash
Alice	Read Write	Read Write		Read Write Execute
Bob	Read			Read Execute
Carol	Read		Read Write	Read Execute

Example: Capabilities

Alice:

README.txt: Read, Write;
/etc/shadow: Read, Write;
/bin/bash: Read, Write, Execute.

Bob:

README.txt: Read;
/bin/bash: Read, Execute.

	README.txt	/etc/shadow	Carol.pdf	/bin/bash
Alice	Read Write	Read Write		Read Write Execute
Bob	Read			Read Execute
Carol	Read		Read Write	Read Execute

Unix Access Control (DAC)

The Unix **kernel** has unrestricted access to the whole machine

Programs (**subjects**) access files and devices (**objects**) through the kernel

Access decisions are based on the object's **userid**/**groupid** and subject's **userid** and groups

⇒ a simplified form of **ACL**

If the user is **root** (userid = 0), access is always granted by the kernel

Users have a **userid**/**groupid** and may belong to several additional groups

Command `id` displays information about user and group id

```
alice:~$ id
uid=1000(alice) gid=1000(alice)
groups=1000(alice),1003(student)
```

Example: add a new user

```
$ docker run --rm -it secunive/sicurezza:ac
root[~]#

root[~]# id                               # display information about user and groups
uid=0(root) gid=0(root)
groups=0(root),0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(di
alout),26(tape),27(video)

root[~]# adduser -D alice                 # creates new user alice with no password
root[~]# echo 'alice:alice' | chpasswd    # change alice’s password to ‘alice’
chpasswd: password for 'alice' changed

root[~]# su - alice                       # switches to user alice

alice[~]$ id                              # display information about user and groups
uid=1000(alice) gid=1000(alice) groups=1000(alice)
```

Example: add a new group

```
root[~]# addgroup student                    # create group student

root[~]# usermod -a -G student alice         # alice is in group student

root[~]# id alice
uid=1000(alice) gid=1000(alice) groups=1000(alice),1001(student)

root[~]# adduser -D bob; echo 'bob:bob' | chpasswd

root[~]# usermod -a -G student bob           # both alice and bob are in group student

root[~]# id bob
uid=1002(bob) gid=1002(bob) groups=1002(bob),1001(student)
```

Unix permissions

File permission is made of **3 triads** defining the permissions granted to the **owner**, to the **group** and to all the **other** users

Example: rw-r--r--

Each permission triad is made up of the following characters:

r: the file can be **read** / the directory's contents can be **shown**

w: the file can be **modified** / the directory's contents can be **modified**

x: the file can be **executed** / the directory can be **traversed**

s: the file is **SUID** (**SGID** if s is in the group triad), implies **x**

⇒ Enables the file to run with the **privileges** of its owner (or group)

Example: permissions

```
root[~]# ls -al                    # display files and their permissions
total 12
drwx------ 1 root root 4096 Nov  3 17:13 .
drwxr-xr-x 1 root root 4096 Nov  3 17:13 ..
-rw------- 1 root root  233 Nov  3 17:15 .ash_history

root[~]# pwd                       # current working directory
/root

root[~]# su - alice                # become alice

alice[~]$ pwd                      # current working directory
/home/alice

alice[~]$ ls /root                 # try to list the content of directory /root
ls: cannot open directory '/root': Permission denied
```

Example: permissions

```
alice[~]$ ls -al                                  # display files and their permissions
total 12
drwxr-sr-x 2 alice alice 4096 Nov  3 17:14 .
drwxr-xr-x 1 root  root  4096 Nov  3 17:14 ..
-rw------- 1 alice alice   36 Nov  3 17:15 .ash_history

alice[~]$ ls -al ..                               # display .. files and their permissions
total 16
drwxr-xr-x 1 root  root  4096 Nov  3 17:14 .
drwxr-xr-x 1 root  root  4096 Nov  3 17:13 ..
drwxr-sr-x 2 alice alice 4096 Nov  3 17:14 alice
drwxr-sr-x 2 bob   bob   4096 Nov  3 17:14 bob

alice[~]$ ls -al ../bob                           # try to list files in /home/bob
total 8
drwxr-sr-x 2 bob  bob  4096 Nov  3 17:14 .
drwxr-xr-x 1 root root 4096 Nov  3 17:14 ..
```

Example: permissions

```
alice[~]$ which ls                         # show the location of the binary program
/bin/ls
alice[~]$ ls -al /bin/ls                   # display its permissions
lrwxrwxrwx 1 root root 20 Nov  3 17:11 /bin/ls -> ../usr/bin/coreutils

alice[~]$ ls -al /usr/bin/coreutils        # it's a link, check the real permissions
-rwxr-xr-x 1 root root 1074184 May  3  2019 /usr/bin/coreutils

alice[~]$ ls -al / | grep bin              # display permissions of /bin and /sbin
drwxr-xr-x   1 root root 4096 Nov  3 17:11 bin
drwxr-xr-x   1 root root 4096 Nov  3 17:11 sbin

alice[~]$ ls -al /bin/su                   # display permissions of /bin/su
-rwsr-xr-x 1 root root 36488 May 10  2019 /bin/su
alice[~]$ su - bob                         # it is SUID root: passwords, setuid, ...
Password:
bob[~]$
```

Managing permissions

Unix permissions can be altered using the **chmod** command

Example: `chmod 600 myfile`
set permissions to `rw-------`

`600` is interpreted as an **octal** number, each digit corresponding to the three permission bits

6 is 110 which is `rw-`
0 is 000 which is `---`

Owner and group can be set using the **chown** command

⇒ non-root users can change the group (to one they belong to) but **not** the ownership.

Example:
`chown alice:student myfile`

changes the group to student, OK if alice is in group `student`

Example: managing permissions

```
bob[~]$ echo "message for Alice" > test.txt  # create file for alice

bob[~]$ chown alice:alice test.txt           # try to change owner and group to alice
chown: changing ownership of 'test.txt': Operation not permitted

bob[~]$ chown bob:alice test.txt             # try to change group to alice
chown: changing ownership of 'test.txt': Operation not permitted

bob[~]$ chown bob:student test.txt           # try to change group to student
bob[~]$ ls -l                                # check that group is now student
total 4
-rw-r--r-- 1 bob student 18 Nov  3 17:21 test.txt

bob[~]$ chmod 640 test.txt                   # change permissions
bob[~]$ ls -l
total 4
-rw-r----- 1 bob student 18 Nov  3 17:21 test.txt     # readable by group student!
```

Example: managing permissions

```
bob[~]$ su - alice                          # switch to alice
Password:
alice[~]$ cat /home/bob/test.txt            # try to read test.txt as alice
message for Alice

alice[~]$ exit                              # exits alice's shell (back to bob)
bob[~]$ exit                                # exits bob's shell (back to root)
root[~]# adduser -D carol                   # add user carol
root[~]# su - carol                         # switch to carol
carol[~]$ id                                # display carol's groups
uid=1003(carol) gid=1003(carol) groups=1003(carol)

carol[~]$ ls -l /home/bob/test.txt          # display test.txt permissions
-rw-r----- 1 bob student 18 Nov  3 17:21 /home/bob/test.txt

carol[~]$ cat /home/bob/test.txt            # try to read test.txt as carol
cat: /home/bob/test.txt: Permission denied
```

SUID and SGID

SUID: When **s** appears in place of **x** in the owner triad, the program will be run with the **privileges** of the owner

Example: system utility requiring root permissions such as `/bin/su`

NOTE: SUID is **risky**: a vulnerability would give root access to the attacker!
⇒ we will discuss mitigations ...

SGID: When **s** appears in place of **x** in the group triad, the program will be run with the **privileges** of the group

Example: access to `/etc/shadow` by `/sbin/unix_chkpwd`

NOTE: When a directory d has SGID set then all files or directories **created** inside d will be owned by the same common (SGID) group

Example: messing up /bin/su permissions

```
root[~]# ls -al /bin/su                    # display /bin/su permissions
-rwsr-xr-x 1 root root 36488 May 10  2019 /bin/su

root[~]# chmod 755 /bin/su                 # disable SUID root

root[~]# ls -al /bin/su                    # display /bin/su permissions
-rwxr-xr-x 1 root root 36488 May 10  2019 /bin/su

root[~]# su - alice                        # switch from root to alice
alice[~]$ su - bob                         # switch to alice to bob
Password:
setgid: Operation not permitted

alice[~]$ exit
root[~]# chmod 4755 /bin/su                # re-enable SUID root
root[~]# ls -al /bin/su                    # display /bin/su permissions
-rwsr-xr-x 1 root root 36488 May 10  2019 /bin/su
```

Example: SGID

```
root[~]# cd /tmp/Challenge2/                # set current directory to /tmp/Challenge2/

root[/tmp/Challenge2]# ./pwdChallenge     # check the pwdChallenge program
Insert password: AAAAAAAAAAAAAAA
Authenticated!

root[/tmp/Challenge2]# cat pwd.txt        # display the password
AAAAAAAAAAAAAAA

root[/tmp/Challenge2]# ls -al             # display the permissions
total 28
drwxr-xr-x 1 root root  4096 Nov  3 21:53 .
drwxrwxrwt 1 root root  4096 Nov  3 21:53 ..
-rw------- 1 root root    15 Nov  3 17:59 pwd.txt
-rwx------ 1 root root 13128 Mar 26  2020 pwdChallenge
```

Example: SGID

```
root[/tmp/Challenge2]# addgroup challenge                # create group challenge
root[/tmp/Challenge2]# chown root:challenge pwd*         # change group to challenge
root[/tmp/Challenge2]# ls -al
total 36
drwxr-xr-x 1 root root       4096 Nov  3 21:53 .
drwxrwxrwt 1 root root       4096 Nov  3 21:53 ..
-rw------- 1 root challenge    15 Nov  3 17:59 pwd.txt
-rwx------ 1 root challenge 13128 Mar 26  2020 pwdChallenge

root[/tmp/Challenge2]# chmod 2755 pwdChallenge           # SGID! NOTE: 2754 is not enough
root[/tmp/Challenge2]# chmod 640 pwd.txt                 # change pwd.txt permissions
root[/tmp/Challenge2]# ls -al                            # display new permissions
total 36
drwxr-xr-x 1 root root       4096 Nov  3 21:53 .
drwxrwxrwt 1 root root       4096 Nov  3 21:53 ..
-rw-r----- 1 root challenge    15 Nov  3 17:59 pwd.txt
-rwxr-sr-x 1 root challenge 13128 Mar 26  2020 pwdChallenge
```

Example: SGID

Now alice can run the program but **cannot access the password file**

⇒ SGID let the program access the file by inheriting the group privileges

```
root[/tmp/Challenge2]# su - alice

alice[~]$ cd /tmp/Challenge2/

alice[/tmp/Challenge2]$ ./pwdChallenge
Insert password: AAAAAAAAAAAAAAA
Authenticated!

alice[/tmp/Challenge2]$ cat pwd.txt
cat: pwd.txt: Permission denied
```

Sticky bit

In shared folders such as /tmp it is useful to give **full access** to any user

Use Case: applications add their (private) temporary folders and files to /tmp

NOTE: full access would make it possible for any user to **delete** files owned by other users!

Sticky bit: When **t** appears in place of **x** in the **other** triad, the directory forbid users to delete files that they do not own

Example: `/tmp` permissions are usually set as:

```
drwxrwxrwt 1 root root
```

Example: sticky bit

```
root[~]# ls -al /tmp/                           # display the sticky bit permissions
total 28
drwxrwxrwt 1 root root  4096 Nov  3 21:53 .
drwxr-xr-x 1 root root  4096 Nov  3 22:12 ..
drwxr-xr-x 1 root root  4096 Nov  3 21:53 Challenge2
-rwsr-xr-x 1 root root 12864 Nov  3 21:19 privilegeDropTest

root[~]# su - alice                             # switch to alice
alice[~]$ rm /tmp/privilegeDropTest             # try to remove privilegeDropTest
rm: remove write-protected regular file '/tmp/privilegeDropTest'? y
rm: cannot remove '/tmp/privilegeDropTest': Operation not permitted
root[~]# chmod 777 /tmp                         # remove the sticky bit
root[~]# su - alice                             # switch to alice
alice[~]$ rm /tmp/privilegeDropTest             # try to remove privilegeDropTest
rm: remove write-protected regular file '/tmp/privilegeDropTest'? y
alice[~]$ ls -al /tmp/privilegeDropTest         # check that the file has been deleted
ls: cannot access '/tmp/privilegeDropTest': No such file or directory
```

ACLs, Capabilities and privilege drop

Access Control Lists (ACLs) define different permissions on a **per-user/per-group** basis. They have higher priority over Unix permissions

Linux Capabilities: instead of SUID permission, assign only the root capabilities that are **necessary** to perform the administrative task

⇒ no full root access if vulnerable!

SUID is **risky**: a vulnerability would give root access to the attacker!

Privilege drop: use root privileges at the beginning and then **drop** to standard user privileges

IDEA: when the user id is set back to the “real” one it cannot be set back again to root (setuid is “one-way”)

Example: privilege drop

```c
int show_uid() {
    printf("Effective user id is: %d\n",geteuid());
    printf("Real user id is:      %d\n",getuid());
    return getuid(); // returns the real user id
}

int main () {
    int myuid;

    myuid = show_uid();

    printf("[-] Trying to open shadow file (need to be root)\n");
    if( open("/etc/shadow",O_RDONLY) < 0 )
        die("Failed to open shadow");

    printf("[-] Trying privilege drop\n");
    if ( setuid(myuid)<0 ) die("Failed to set original uid\n");
```

Privileged access (requires SUID root)

Drops privileges as soon as possible

Example: privilege drop

```
    ...

    show_uid();

    printf("[-] Checking that shadow cannot be opened\n");
    if( open("/etc/shadow",O_RDONLY) >= 0) die ("I could open shadow?");

    printf("[-] Trying to set back uid 0 (root)\n");
    if ( setuid(0)<0 ) die("Failed to set root uid");

    show_uid();

    printf("[-] Trying to open shadow file (need to be root)\n");
    if( open("/etc/shadow",O_RDONLY) < 0 ) die("Failed to open shadow");
}
```

Once dropped root privileges cannot be re-acquired

Example: privilege drop

```
alice[/tmp]$ ls -al /tmp/privilegeDropTest
-rwsr-xr-x 1 root root 12864 Nov  3 21:10 /tmp/privilegeDropTest

alice[/tmp]$ ./privilegeDropTest
[*] Effective user id is: 0
[*] Real user id is:      1000
[-] Trying to open shadow file (need to be root)
[*] Done!
[-] Trying privilege drop
[*] Done!
[*] Effective user id is: 1000
[*] Real user id is:      1000
[-] Checking that shadow cannot be opened
[*] Done!
[-] Trying to set back uid 0 (root)
[=] ERROR: Failed to set root uid: Operation not permitted
```