

## Authentication

*to prove the identity of a user*

- segreti: password, certificati digitali
- oggetti fisici: smart card, chiavi, ...
- biometria: impronte digitali, ...

1

## Authorization

*to determine what the user can do*

*Protection domain*: l'insieme dei diritti posseduti da un processo

Un *diritto* è una coppia (oggetto, azioni permesse)  
es. (/home/matteo/foo.txt: read,write), (/etc/passwd: read)

Ci deve essere un meccanismo per *cambiare* domain  
(es. permettere a un utente di cambiarsi la password)

2

## Protection domains in Unix

Ci sono due protection domains: *kernel* e *user*

In pratica questi due domain sono insufficienti, per cui si definiscono domini di protezione *associati agli utenti*

3

## Protection matrix

*Concettualmente*, esiste una *matrice di protezione*: una relazione fra *utenti* e *oggetti*

		Object							
		File1	File2	File3	File4	File5	File6	Printer1	Plotter2
Domain	1	Read	Read Write						
	2			Read	Read Write Execute	Read Write		Write	
	3						Read Write Execute	Write	Write

4

## Domain switching

Per rappresentare la possibilità di cambiare dominio, trattiamo i "domini" come "oggetti"

Domain	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

5

## Trasferimento dei diritti

Può un utente trasferire un diritto a un altro utente?  
 ⇒ aggiungiamo il permesso "copy right"

L'utente beneficiato può a sua volta trasferire il diritto?  
 ⇒ distinguiamo "copy" e "copy limited"

Una volta trasferito il diritto, l'utente originale lo perde?  
 ⇒ in questo caso si chiama "transfer"

Dobbiamo marcare nella matrice di protezione quali diritti hanno i permessi copy, copy limited, transfer

6

## Audit

Quando un tentativo di accesso fallisce, occorre registrare questo fatto?

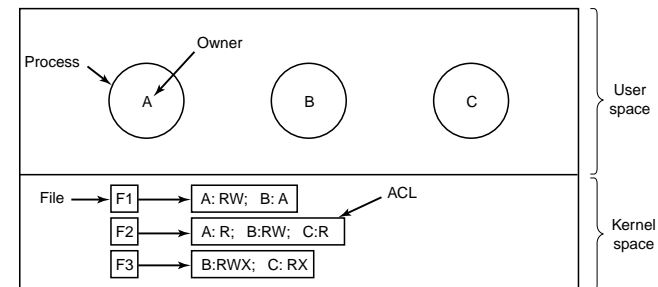
E quando un tentativo di accesso ha successo?

Occorre predisporre una *audit policy*

7

## Implementazione della matrice di protezione

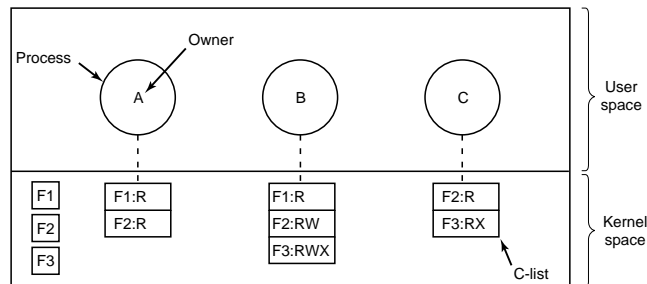
Per righe: *Access Control Lists (ACL)*



8

## Implementazione della matrice di protezione

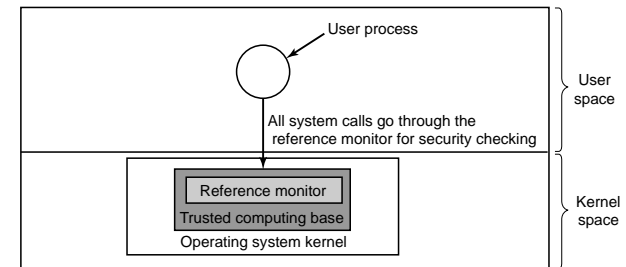
Per colonne: *Capability Lists*



9

## Trusted computing base

TCB: *all hardware and software necessary to enforce the security rules*



10

## Multilevel security

Nelle applicazioni militari, ci sono diversi *livelli* di autorizzazione (*clearance*):

- top secret
- secret
- confidential
- unclassified

Un sistema è *multilevel* se deve poter essere usato da persone di diverso livello di clearance

11

## Il modello Bell-La Padula

- simple security property (no read up)
- star property (no write down)

12

## Orange book security

Livello D: no security (MS-DOS, Win 95/98/Me)

Livello C1: multiuser

- authentication
- discretionary access control (DACL)
- protected mode OS

Livello C2: DACL a livello di singolo utente

⇒ Unix raggiunge C1 ma non C2 (perché non ha le ACL)

Livello B: multilevel

- ogni utente e oggetto ha un livello di *clearance*

Livello A: formally verified

13

## Covert channels

La “star property” non è realizzabile in pratica

14

## Autenticazione in Unix: /etc/passwd

Database delle password: /etc/passwd

Password crittate *one-way*

Salt

/etc/shadow

15

## Autorizzazione in Unix

Users are identified by UID (an integer)

Groups are identified by GID (an integer)

Every user belongs to one or more groups

```
/etc/passwd
root:x:0:0::/root:/bin/bash
ftp:x:14:50::/home/ftp:
matteo:x:500:100:Matteo Vaccari:/home/matteo:/bin/bash
```

```
/etc/groups
root::0:root
ftp::50:
users::100:
project-X::1234:matteo,marco,guido
```

16

Every file has a UID and a GID, and RWX permissions for the UID, the GID and everyone else

```
-rw-r--r-- 1 matteo users 5 May 30 11:21 pippo.txt
```

Inizialmente UID e GID sono quelli dell'utente che ha creato il file

Posso cambiare UID (e GID) con il comando `chown(1)`

```
chown <newuser> <file> <file> <file> ...
chgrp <newgroup> <file> <file> <file> ...
```

Per esempio:

```
chown guest pippo.txt pluto.txt topolino.c
chgrp other pippo.txt pluto.txt topolino.c
```

Oppure, in un colpo solo:

```
chown guest.other pippo.txt pluto.txt topolino.c
```

17

## I gruppi

Sono il meccanismo principale per gestire la collaborazione in Unix

`newgrp(1)` is used to change the current group ID during a login session.

```
$ touch aaa
$ ls -l aaa
-rw-r--r-- 1 matteo users 0 Jun 9 10:49 aaa
$ newgroup project-X
$ touch bbb
$ ls -l bbb
-rw-r--r-- 1 matteo project-X 0 Jun 9 10:49 bbb
$
```

`chgrp(1)` e `newgrp(1)` mi permettono di usare solo i gruppi a cui appartengo

(`chown(1)` è riservato a root)

18

## Permission bits for gurus

la rappresentazione `rw-r--r--` è per pivelli

```
rw- r-- r-- pivelli
110 100 100 binario
6 4 4 ottale
```

Allora diciamo che un file ha il **modo** 644

(Nota: in C la costante ottale si indica con il prefisso "0", es. 0644)

19

## Problema

dati questi permessi:

```
-rw----- 1 root root 605 Dec 2 2002 /etc/shadow
```

come può un utente cambiare la sua password?

20

## Il bit SETUID

S\_ISUID 04000 set user ID on execution

Un normale eseguibile: modo 711

```
-rwx--x--x 1 root bin 46700 May 28 2002 /bin/ls
```

Un eseguibile setuid: modo 4711

```
-rws--x--x 1 root bin 36800 Jun 10 2002 /usr/bin/passwd
```

Quando un processo fa una `exec(2)` di un eseguibile setuid, il suo UID *effettivo* diventa quello del file eseguibile

⇒ eseguo `/usr/bin/passwd` con i permessi di root

Pericolo! `passwd` deve essere scritto con *molta* attenzione

... infatti si rifiuta di cambiare la password di altri utenti

21

## Ancora su setuid

Ciascun processo ha un

- *real* UID
  - usato per sapere chi è il proprietario originale
- *effective* UID
  - usato per decidere che cosa può fare

Normalmente:

effective UID == real UID

Quando eseguo un programma setuid:

effective UID != real UID

`/usr/bin/passwd`

- usa il real UID per sapere di chi bisogna cambiare la password
- ha bisogno di effective UID == 0 per potere cambiare `/etc/shadow`

22

## Chiamate di sistema relative alle autorizzazioni in Unix

```
int getuid(void);
```

returns the **real** user ID of the current process

```
int geteuid(void);
```

returns the **effective** user ID

```
int getgid(void);
```

returns the **real** group ID of the current process

```
int getegid(void);
```

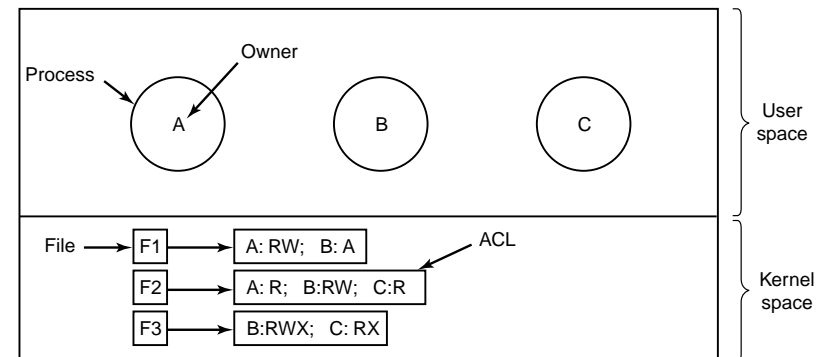
returns the **effective** group ID

```
int setuid(int uid);
```

sets the effective user ID of the current process

23

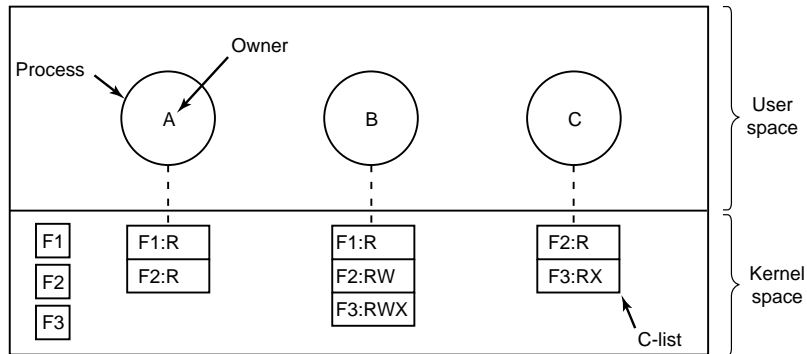
## Access Control Lists (ACL)



Each ACL is a column of the protection matrix

24

## Capability lists



Each capability list is a row of the protection matrix

25

## Un problema di Unix

root può fare tutto

gli utenti normali possono fare poco

se un programma deve fare una sola cosa da root, come ad es. aprire una socket su un port privilegiato (<1024) devo eseguirlo come root

⇒ pericolo! se un attaccante trova un exploit...

Il meccanismo di *capabilities* serve a dare a un processo solo le capacità che gli servono

26

CAP_CHOWN	Allow for the changing of file ownership
CAP_DAC_OVERRIDE	Override all DAC access restrictions
CAP_DAC_READ_SEARCH	Override all DAC restrictions regarding read and search
CAP_KILL	Allow the sending of signals to processes belonging to others
CAP_SETGID	Allow changing of the GID
CAP_SETUID	Allow changing of the UID
CAP_SETPCAP	Allow the transferring and removal of current set to any PID
CAP_LINUX_IMMUTABLE	Allow the modification of immutable and append-only files
CAP_NET_BIND_SERVICE	Allow binding to ports below 1024
CAP_NET_RAW	Allow use of raw sockets