Materiale

- Cap 4 libro del testo
- Access Control

Controllo degli accessi (Linux)

Controllo degli accessi: far accedere solo la persona autorizzata. Uno degli obbiettivi principali dell'OS è quello di introdurre dei meccanismi che garantiscono confidenzialità, integrità, disponibilità delle informazioni che vengono inseriti del sistema.

Il sistema di protezione è costituito da 3 entità principali:

- Il meccanismo di autenticazione: il sistema verifica l'identità di chi sta usando,
- Controllo degli accessi: verifica chi accede a cosa,
- Logging/Auditing: sono degli strumenti che permettono di verificare che tutto vada apposto registrando tutto quello che succede (logging).

B. Lampson **regole auree** perché tutte le regole iniziano con Au

Autenticazione **Authenticate** -> Controllo degli accessi **Authorize access** -> **Auditing**

chi sei -> cosa fare -> registrare

- **Subject** soggetto: *componenti attivi del sistema* umano/processo che esegue per conto di un utente.
- **Object** oggetto: *parte passiva* sono le risorse.
- **Operazioni di accesso**: *diritti di accesso* cosa può fare l'utente.

Security policy

Alla base di tutto ci sono le regole:

- **Security policy** politiche di sicurezza: **insieme di attività che un utente può fare** nel sistema. Controlla le minacce e vulnerabilità, specificando chi può fare cosa e come.

Access control - ITU-T X.800

Access control è uno strumento di **prevenzione** di uso **non autorizzato** di una risorsa.

- Prevenire accesso non autorizzato.
- Prevenire accesso in una modalità non autorizzata (tipo accesso in lettura ma non in scrittura).

Chi decide la politica di sicurezza di un sistema? Ci sono tre principali metodi

- **DAC** *Discretionary Access Control* accesso discrezionale: è definito dal proprietario della risorsa (file system di os) (quando si crea un file il sistema decide chi può fare cosa, queste informazioni vengono presi da alcuni variabili per sistema).

- **MAC** *Mandatory Access Control* controllo degli accessi mandatorio: politiche centralizzate che setta regole per tutti e tutto. è deciso da es. hardware, sysadmin, app developer. La politica viene definita da

un entità superiore da chi usa la risorsa, quindi, qualcuno altro decide per voi quello che qualcuno altro può fare. (es. processo utente i diritti vengono decisi dall'OS) (ambiente militare).

> Un processo utente ha dei diritti che vengono decisi dall'OS, processo gira in user mode al livello 3, può fare solo quello che stato deciso da chi ha progettato il sistema per il livello 3.

- **RBAC** *Role Based Access Control*: accesso agli oggetti del sistema sono dettati dalle regole che dipendono dal ruolo dell utente. è un pò una fusione tra i due. Ad ogni utente del sistema viene assegnato un ruolo.

La politica di sicurezza si traduce in una matrice, dove le righe sono soggetti, le colonne sono le risorse (oggetti).

Il problema è che richiede molta memoria. Per Unimia siamo sui 10gb.

Prima soluzione ridurre numero di righe, raggruppando utenti per gruppi.

- **ACL** *Access Control List*: è una struttura dati ricavata dalla matrice, contiene un **elenco degli soggetti** e **tipo operazioni che soggetto puo fare**. Si ottiene dividendo la matrice per colonne per ogni oggetto ricorda gli utenti e cosa possono fare. Quindi, ogni file avrà una lista di utenti con loro permessi. Viene salvato in i-node. ACL deve essere salvato in un posto sicuro e protetto che viene gestito dall'OS. Viene usato per controllo degli accessi alle risorse (file system).

> **i-node** è una struttura dati presente sul hdd, è contenuto in una struttura dati chiamato **i-node table**. i-node contiene metadati relativi ai file, data di ultima modifica, creazione ecc. Per ogni file esiste un inode, unix mette ACL dentro i-node. In UNIX tutte le risorse sono file. `i-node_table(i-node(acl))`.

- **Capability** list: introdotto da Dennis and Van Horn. Consente di avere accesso attraverso un token (Controllo degli accessi viene fatto con un token) (usato nei sistemi Windows). Viene caricato nella zona di memoria di kernel e gestito dal kernel il quale permette di essere protetto dagli utenti, perché utenti non hanno accesso al kernel. (utente non può accedere alla zona di memoria del kernel).

> In UNIX kernel ha una zona di memoria (Giga più alto) riservata a cui i processi utenti non possono accedere. Quando si cerca di accedere si crea un Segmentation Fault.

ACL vs Capability

- ACL: la banca ha una lista di authorized persons,
- Capability: es. la banca rilascia dei token.

ACL

La banca ha una lista di persone autorizzate (ACL), la banca deve prendere in carico di verificare l'identità delle persone che si presentano per accedere alle cassette di sicurezza.

Deve memorizzare e proteggere, verificare l'identità, per aggiungere nuova persona owner deve venire con l'utente, cioè, non si può delegare un'altra persona. Per rimuovere una persona basta toglierlo dalla lista.

- Facile implementazione,
- non è efficiente per controlli di sicurezza,

- difficile trovare i file accessibili da una persona (devo scorrere tutti i-node per avere tutta la lista dei file accessibili da quella persona).

ACL è lo strumento principale che viene usato per controllo degli accessi al file system.

ACL viene usata per fare security checking. Un utente chiede di accedere ad una risorsa e si va a controllare la lista per vedere s'è autorizzata o meno. solitamente l'OS conosce il processo e quindi l'OS deve accedere al i-node e recuperare ACL.

Capability

è responsabilità di Alice che deve proteggere token "la chiave". se si vuole aggiungere una nuova persona al gruppo, basta dare la chiave. è complicato la revoca, cioè, per revocare bisogna cambiare la chiave.

- **Runtime security check** più efficienti (perché il processo si presenta con il token (capability)),
- si può delegare senza molta difficolta,
- difficile trovare chi ha accesso ad un file, se voglio sapere gli utenti che possono accedere ad una risorsa devo controllare tutti gli utenti.

ACL vs capabilities

Le ACL sono di facile implementazione ma non sono particolarmente performanti nell'effettuare il security checking (il sistema operativo deve conoscere l'identità del soggetto e per ogni operazione richiesta deve accedere all'ilist usando l'inumber, recuperare l'inode e quindi l'ACL, a questo punto deve scorrere l'ACL fino a trovare i permessi del soggetto e infine verificare che questo abbia i permessi necessari). Per sapere a quali oggetti può accedere un certo soggetto bisogna scorrere tutti gli inode del sistema.

Le capabilities sono più complesse da realizzare perché non devono essere falsificabili ma sono più veloci nell'effettuare il security checking (un processo richiede una certa operazione presentando la capability, questa viene verificata direttamente). Per sapere quali permessi hanno i vari utenti su un certo oggetto bisogna scorrere tutte le capability degli utenti presenti nel sistema.

Anche se le capacility sono più efficienti, storicamente si sono preferite le ACL, soprattutto per gestire gli accessi al file system.

Reference Monitor

Componente specifica dell'OS che fa access control è il **Reference Monitor** è un insieme di componenti sono distribuite in diversi parti: nel kernel, file system, drivers, ecc. mirate a garantire la sicurezza dell'intero sistema.

Proprietà

- **Valida ogni accesso ad una risorsa**. Quindi, ogni volta che un processo chiede di accedere ad una risorsa questo accesso viene tradotto in una chiamata al reference monitor che autorizza o meno l'accesso.
- Deve essere immune contro le compromissioni.
- Dovrebbe essere dimostrato la sua correttezza.

utente -> entra -> diventa un soggetto -> ogni volte che deve accedere ad un oggetto, inoltra la richiesta -> viene intercettato dal reference monitor (contemporaneamente logga tutto) -> secondo le security policy nega o accetta.

Come viene utilizzato questi sistemi in UNIX

UNIX ha dotato una politica di accesso discrezionale DAC per quanto riguarda il file system.

`users` utenti sono tutti coloro che hanno un account, salvato nel `/etc/passwd` passwd contiene elenco degli utenti autorizzati ad accedere a quel sistema.

```
nome utente: password (non ce più): user id: group id: nome esteso: home
directory: tipo di shell

username:x:uid:gid:name:homedir:shell
```

`uid` è importante, è un numero a 16-bit, ogni utente è un numero.

superuser `root` ha `uid` numero `0`. superuser non si applica il controllo degli accessi, reference monitor viene bypassato. Ma non può fare: non può conoscere password di utente, ma può cambiarlo. Non può scrivere su file read-only ma può montarlo.

Le password sono memorizzate in `/etc/shadow` (è stato spostato da passwd a shadow)

`group` gruppi sono contenuti nel file `/etc/group`

```
groupname:password:gid:list_of_users
```

ogni utente può appartenere a qualsiasi numero di gruppi ma ha un gruppo primario il suo `gid` è scritto nel file `passwd`.

```
sudo adduser username
```

per modificare utente

```
usermod options username
usermod -a -G username

groupadd options groupname
```

UID e GID di un processo UNIX

Effective EUID/EGID - Real RUID/RGID

I processi unix sono contraddistinti da un pid, è un numero che li identifica all'interno del sistema. Un nuovo processo viene generato attraverso il syscall fork().

Un processo a sua volta ha un `gid` e un `uid` associato:

- Real uid e gid `RUID`/`rgid` identifica il soggetto che l'ha generato e viene **ereditato dal processo figlio dal processo genitore**, generalmente uid e gid dell'utente che è loggato.
- Effective uid e gid `EUID`/`egid` viene **ereditato dal processo genitore or dal file associato al processo** che viene mandato in esecuzione. Questo succede solo per file `setuid`, quando `execve` prende il nome del file. Tipicamente RUID e GUID sono uguali.

Il commando ls viene letto dal shell (la shell è un processo), se il commando è corretto la shell fa fork() generando una copia di se stesso, quindi, abbiamo due shell in esecuzione:

```
if command == correct then fork()
if (pid == 0) execve (/bin/ls)
```

A questo punto il processo figlio fa esegue il `syscall execve` che prende il file eseguibile.

In questo caso ls eredita gui/uid di shell, cioè, dell'utente che ha eseguito la shell.

Ogni processo ha zona **codice**, **dati**, **heap**, **stack**, `execve` sostituisce la zona di memoria del codice di shell con il codice del ls (nel nostro caso).

Directory ha una tabella che contiene i nome dei file e i-node number

Per **ogni oggetto in UNIX esistono solo tre utenti**. Owner, gruppo primario dell'owner, resto del mondo. `user | group | other` .

Ogni soggetto può fare read `r`, write `w`, execute `x`. Usa 3 bit per tipo, 9 bit in totale.

```
rwx | rwx | rwx
777

r-x | r-- | --x
542

rw- | r-- | r--
644
```

```
type | users | group | others

d | rwx | r-- | -r-

- file
d dir
l link
```

Comandi per cambiare ACL:

```
chmod 0754 filename
rwxr-xr--

chmod u+wrx,g+rx,g-w,o+r,o-wx filename
u user
g group
o other
+ aggiungi
- rimuovi
```

Cedere ownership: (tipicamente fatto da root)

```
chown newower:newgroup filename
```

`execve` **potrebbe modificare** `EUID` del processo: quando si fa fork si eredita RUID ed EUID del genitore e quando il processo figlio fa execve è possibile che modifichi l'EUID (prende quello del file). Es. utente 1000 manda in esecuzione il commando `ls`, `ls` ha come `uid 0` perché è di `root`, quindi il processo ls che l'utente ha eseguito avrà `RUID 1000` ed `EUID 0`. Questo succede solo ai file `setuid`.

Quando reference deve controllare ACL

1. Recupera **i-node del file**,
2. Controlla chi è il **proprietario tramite uid** dell'owner,
3. Confronta con **effective uid**, se sono uguali **recupera ACL** dall'i-node,
4. Se owner ha `EUID` **applica permessi** dell'owner, altrimenti prova con group e other.

Importante! per il controllo degli accessi, utilizza **EUID** e non RUID.

Significato di diritti ACL di un directory

Si utilizzano gli stessi caratteri che si utilizzano per i file ma con significati diversi.

```
read dir r: diritti per accedere in lettura al contenuto del directory, possibile
fare elenco dei file / cercare file in directory. Senza non si può vedere il
contenuto della directory (non si può fare ls).

write dir w: diritti per aggiungere file oppure rimuovere file dal directory. (per
cancellare è necessario sia i diritti di write ed execute)

execute dir x: è possibile utilizzare la directory. possibile posizionarsi
all'interno del directory e utilizzare la directory, possibile attraversare la
directory.
```

L'OS utilizza delle **maschere** per assegnare i diritti ACL. Se un utente crea un file il sistema assegnerà un set di diritti predefiniti, eventualmente l'utente potrà modificare secondo le sue esigenze.

ACL+ di UNIX "raffinato"

Sono state introdotte delle ACL access control lists (diverse) che consente di aggiungere delle ulteriori informazioni di raffinamento.

Per dare permessi ad un utente che non è del tuo gruppo senza aprire al mondo

permette di definire permessi per particolari utenti

```
getfacl filename // mostra i diritti supplementari

setfacl -m ugo:user_or_group_name:permissions // -m per aggiungere/modificare

setfacl -m u:student:7 script.sh // dà diritti rwx per utente student, u = user

setfacl -m g:studentGroup:r-x script.sh // g = group, per rimuovere -x

se ce rwxrwxr-x+ "+" significa che ci sono diritti supplementari
```