1 Lezione del 15-12-25

Continuiamo a parlare di protezione nei S/O.

1.1 Matrice degli accessi

Un sistema di protezione può essere rappresentato sfruttando il modello a **matrice degli accessi**.

Il modello mantiene tutta l'informazinoe che specifica il tipo di accesso che i soggetti hanno sugli oggetti e consente di:

- Rappresentare lo *stato di protezione*;
- Garantire il rispetto dei vincoli di accesso, come specificati dallo stato di protezione, per ogni tentativo di accesso;
- Permettere la modifica controllata dello stato di protezione determinando una *transizione* di stato.

Il meccanismo associato al modello consiste nel:

- Verificare se ogni richiesta di accesso che proviene da un processo che opera in un determinato dominio è consentita oppure no;
- Avere la possibilità di modificare dinamicamente il numero di oggetti e soggetti;
- Consentire ad un processo di cambiare dominio durante l'esecuzione;
- Modificare in modo controllato il cambiamento dello stato di protezione, attraverso le sopranominate transizioni di stato.

Vediamo quindi l'esempio di una matrice degli accessi, con 3 soggetti $\{S_1, S_2, S_3\}$ e 3 oggetti $\{X_1, X_2, X_3\}$:

Soggetti \ Oggetti	X1	X2	X3	S1	S2	S3
S1	read*	read	exec		term	receive
S2		owner write		control receive		term
S3	write exec		read	send	send receive	read

Notiamo come i soggetti (che per noi rappresentano domini di processi) sono a loro volta oggetti di altro soggetti, attraverso le condizioni particolari (*term*, cioè può terminare, ecc...).

Inoltre, vediamo che la matrice degli accessi è una matrice *sparsa*, cosa che chiaramente ci offre delle semplificazioni in termini di implementazione.

Il meccanismo è quindi semplice: un processo che opera nel dominio D_i (per noi le righe), può accedere soltanto agli oggetti specificati nella riga i e con i diritti di processo indicati.

Nello specifico:

- Quando un soggetto i, con dominio D_i, vuole accedere ad un oggetto j, si deve controllare l'entrata i, j-esima della matrice. I diritti di accesso in tale entrata, se specificati, sono quelli che il soggetto avrà sull'oggetto;
- Se usiamo l'approccio **DAC** (*Discretionary Access Control*), le politiche di accesso sono definite dagli utenti stessi, e in particolare dai *properietari* degli oggetti (per cui abbiamo visto una condizione particolare nella tabella superiore, la *owner*).

  Questo significherà che le entrate della matrice degli accessi sono definite dagli utenti.

1.1.1 Propagazione dei diritti d'accesso

La possibilità di copiare un diritto di accesso per un oggetto da un dominio all'altro viene rappresentato in notazione con l'asterisco (**copy flag**).

Nello specifico, un soggetto che possiede il copy flag su una modalità di accesso ad un certo oggetto, potrà trasferire tale modalità agli altri soggetti (cioè copiare la modalità sulla sua colonna).

- Notiamo che la propagazione avviene copiando il solo diritto (*propagazione limitata di un diritto*) oppure il diritto accompagnato da copy flag. Nel secondo caso il soggetto che riceve il diritto può a sua volta trasmetterlo;
- La propagazione, in aggiunta, può comportare la perdita del diritto per il soggetto che lo ha trasmesso (*trasferimento di un diritto*).

La teoria sviluppata da Graham e Denning ci dimostra che con le matrici di accesso e le operazioni appena definite si ha:

- Propagazione limitata e controllata dei metodi di accesso (*confinement*);
- Sicurezza dalla modifica indiscriminata dei diritti di accesso (*sharing parameters*).

1.1.2 Implementazione delle matrici di accesso

Esistono più metodologie secondo le quali possiamo **implementare** una matrice di accesso. Ad esempio, abbiamo già detto che questa sarà rappresentata da una matrice sparsa.

In esempio, vediamo i seguenti metodi:

- **ACL** (*Access Control List*): queste sono liste che elencano, per ogni oggetto, i soggetti e le relative modalità di accesso. Nel nostro esempio, ogni colonna è un ACL.

  Vediamo che in UNIX i bit di protezione rappresentano un'ACL, e che queste ACL sono distribuite sugli i-node. Questo rende l'implementazione UNIX delle ACL un'implementazione *distribuita* (le ACL sono sparse sui vari nodi, cioè gli oggetti);

- **CL** (*Capability List*): queste sono liste che elencano, per ogni soggettto, gli oggetti e le relative modalità di accesso. Nel nostro esempio, ogni riga è una CL.

  In termini di efficineza, la CL è più efficiente delle ACL (quando si genera una richiesta da parte di un certo soggetto, la ricerca va fatta sulla CL del soggetto e non sull'ACL dell'oggeto richiesto, che va prima trovata).

  Il problema delle CL è che i processi cambiano molto più spesso dei file, cioè mantenere ACL distribuite per gli oggetti nel sistema risulta a lungo termine molto meno dispendioso e molto più semplice che creare nuove CL per ogni nuovo oggetto.

  Abbiamo detto che UNIX usa le ACL (9 + 3 bit di protezione) scritte negli inode dei file. In verità, quello che accade nella realtà è che l'ACL di ogni file viene usata una volta sola, cioè quando si fa la prima open su tale file. In seguito, l'ACL di tale file viene memorizzata nella tabella dei file aperti di sistema (e in particolare nella tabella degli inode a cui tale tabella punta). A questo punto, la tabella dei file aperti di processo, cioè quella che indicizziamo coi *filde*, andrà effettivamente a realizzare una CL, dove ad ogni file aperto verrà associata la politica di accesso corrispondente.

1.1.3 ACL multiutente

Il concetto di ACL viene introdotto per sistemi *monoutente*.

Nel caso di sistemi *multiutente*, invece, si definiscono gruppi di utenti che hanno un nome e possono essere inclusi nella ACL. A questo punto, ogni entrata della ACL identifica un soggetto con UID (id utente) e GID (id gruppo), cioè:

```
UID1, GID1 : < diritti soggetto 1 >
UID2, GID2 : < diritti soggetto 2 >
...
```

Il fatto che diverse coppie utente-gruppo possono avere diversi diritti significa che un utente può avere diritti diversi a seconda del gruppo al quale appartiene. Si possono quindi usare le *wildcard* per avere regole più comprensive, come ad esempio che si applicano a tutti gli utenti di un certo gruppo, a allo stesso utente qualsiasi sia il suo gruppo.

1.2 Sicurezza multilivello

Abbiamo introdotto in 21.3.1 le diverse politiche di protezione possibili negli S/O. Riassumendo, la distinzione principale è fra:

- **DAC** (*Discretionary Access Control*), a controllo degli utenti che hanno diritti sui loro file, detti *owner*;
- **MAC** (*Mandatory Access Control*), più rigorosa e controllata, dove il S/O detta le regole su chi può vedere cosa. Queste regole possono essere modificate solo dopo aver ottenuto permessi speciali.

L'uso di sicurezza di tipo MAC viene reso necessario dal fatto che in sistemi DAC a matrice degli accessi c'è comunque il rischio che un soggetto ottenga diritti più elevati di quanto gli spettino.

Abbiamo quindi bisogno di un nuovo modello che si affianchi a quello di Graham-Denning visto prima.

1.2.1 Modello Bell-LaPadula

Questo sarà il modello di **sicurezza multilivello** Bell-LaPadula. In tale modello, ad ogni soggetto è associata una classe $Y = SC(X)$ dove Y è la classe e X il soggetto. L'idea è che, date due classi i e j, i soggetti i abbiano accesso agli oggetti j solo se $i > j$.

Il modello di Bell-LaPadula definisce quindi regole su come le informazioni possono circolare:

1. Proprietà di sicurezza: un soggetto in esecuzione al livello di sicurezza k può **leggere** solo oggetti al suo livello o a livelli inferiori:

$$\text{read:} \quad SC(O) \leq SC(S)$$

2. Di contro, un soggetto in esecuzione al livello di sicurezza k può **scrivere** solo oggetti al suo livello o a livelli superiori:

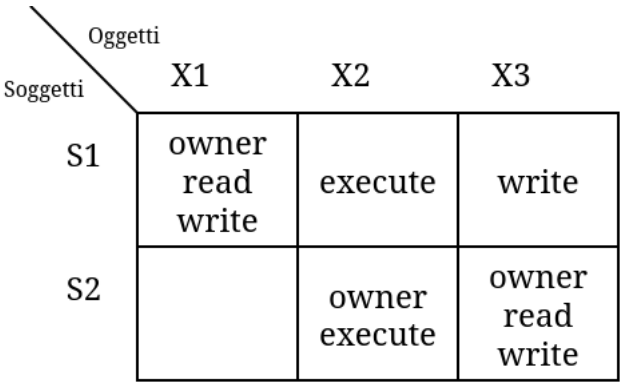
$$\text{write:} \quad SC(S) \leq SC(O)$$

   Questo potrebbe sembrare poco intuitivo: Bell-La Padula non risolve il problema dell'integrità delle informazioni (paradossalmente posso scrivere dove non posso leggere), ma garantisce la sicurezza. Infatti, non si possono leggere informazioni di livello più alto al nostro, e non si possono scrivere informazioni in livelli più bassi (potenzialmente rendendole disponibili a chi non dovrebbe poterle leggere).

1.2.2 Cavalli di Troia

Vediamo un esempio pratico per capire l'utilità della seconda regola del modello Bell-LaPadula, in un caso dove il modello a matrice degli accessi fallisce.

Partiamo dalla seguente matrice degli accessi:



Soggetti \ Oggetti	X1	X2	X3
S1	owner read write	execute	write
S2		owner execute	owner read write

In questo caso abbiamo 3 oggetti:

- X_1, che è un file riservato del soggetto S_1;
- X_2, che è il nostro trojan in possesso di S_2;
- X_3, che è un file di appoggio usato nell'attacco da S_2.

La conformazione attuale della matrice degli accessi prevede che S_1 può eseguire X_2. Quando X_2 è in esecuzione nel dominio di S_1, può leggere da X_1 e scrivere in X_3. Questo, ad esempio, gli permetterebbe di copiare i contenuti di X_1 in X_3. S_2 potrebbe quindi successivamente leggere X_3, ed avere accesso ai file originariamente di proprietà solo di X_1.

Tale problematica non si sarebbe posta fosse stato adottato il modello di Bell-LaPadula. Sarebbe infatti bastato porre:

- $SC(S_1) = SC(O_1) = R$
- $SC(S_2) = SC(O_2) = SC(O_3) = P$

con $R > P$. A questo punto, S_1 non avrebbe potuto scrivere su O_3, per la seconda regola, nemmeno eseguendo il trojan O_2.

1.2.3 Modello BiBa

Il modello BBa è un alternativa al modello di Bell LaPadula, che non assicura la sicurezza ma l'integrità dei dati (nessuno tocca dati che non gli spettano). I 2 modelli sono complementari e non possono essere realizzati contemporaneamente.

In particolare, le regole implementate dal modello BiBa sono:

1. Proprietà di integrità: un soggetto in esecuzione al livello di sicurezza k può **leggere** solo oggetti al suo livello o a livelli superiori

$$\text{read:} \quad SC(O) \geq SC(S)$$

2. Di contro, un soggetto in esecuzione al livello di sicurezza k può **scrivere** solo oggetti al suo livello o a livelli inferiori

$$\text{write:} \quad SC(S) \geq SC(O)$$