# UNIVERSITA' DEGLI STUDI DI PERUGIA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Laurea Specialistica in Informatica


Corso di Sicurezza Informatica

Relazione Seminario:

## Security Models

Studente

**Nigro Antonio**

Professore

**Bistarelli Stefano**

Anno Accademico 2008-2009

**INDICE**

*1 The Harrison-Ruzzo-Ullman model..... 1*

*2 Information Flow models..... 4*

*2.1 Entropia ed Entropia condizionata..... 4*

*2.2 A Lattice-based model..... 5*

*2.3 Riconoscere i flussi d'informazione..... 6*

*2.3.1 Compiler-based..... 6*

*2.3.2 Execution-based..... 8*

*2.4 Noninterference models..... 8*

*3 Execution monitor..... 9*

*3.1 Properties of execution..... 10*

*3.2 Safety and Liveness..... 11*

# SECURITY MODELS

Il modello di sicurezza Bell-La Padula ("segretezza" NO READ-UP/NO WRITE-DOWN) è stato considerato il modello di sicurezza in generale per molto tempo, fin quando la sicurezza si è estesa anche in altri ambienti (commerciale) oltre a quello militare, perciò sono stati realizzati altri modelli di sicurezza.

## 1. THE HERRISON-RUZZO-ULLMAN MODEL

Il modello Bell-La padula  non prevede politiche per modificare i diritti di accesso o per la creazione o eliminazione di oggetti e soggetti. Questo è possibile con il modello Harrison-Ruzzo-Ullman realizzato nel 1976. Tale modello ha permesso di rispondere a diverse domande relative ai tipi di protezione che un dato sistema può offrire.

Per descrivere il modello abbiamo bisogno:

- un set di soggetti S;
- un set di oggetti O;
- un set di diritti di accesso R;
- una matrice degli accessi $M=(M_{SO})_{s \in S, o \in O}$; ogni casella di M contiene un sottoinsieme di R specificando i diritti del soggetto s sull'oggetto o.

Esistono sei operazioni per manipolare l'insieme dei soggetti S, degli oggetti O e la matrice M:

- enter r into M
- delete r from M
- create subject s
- delete subject s
- create object o
- delete object o

I comandi hanno il seguente formato:

```
command c(x1,...,xk)
        if r1 in Ms1,o1 and
        if r2 in Ms2,o2 and
        .
        .
        .
        if rm in Msm,om then
                op1
                op2
                .
                .
                .
                opn
end
```

Gli indici s1,...,sm e o1,...,om sono i soggetti e gli oggetti che compaiono nella lista di parametri (x1,...,xk). Le condizioni verificano se sono presenti particolari diritti di accesso. La lista delle condizioni potrebbe essere vuota. Se tutte le condizioni sono verificate, la sequenza di operazioni viene eseguita. Ogni comando contiene al minino una operazione. Per esempio, il seguente comando è usato dal soggetto s per creare un nuovo file f così che s è il proprietario (owener) del file ed ha su di esso i permessi di lettura e scrittura.

```
command create_file(s,f)
        create f
        enter o into Ms,f
        enter r into Ms,f
        enter w into Ms,f
end
```

Il proprietario s del file f può concedere i diritti di lettura sul file ad un altro soggetto p con

```
command grant_read(s,p,f)
        if o in Ms,f then
                enter r in Mp,f
end
```

La matrice degli accessi descrive lo stato del sistema. L'effetto di un comando viene memorizzato come un cambiamento della matrice. Di solito si utilizza una matrice M' per indicare le modifiche sulla matrice degli accessi M.
La politica del sistema è quella di controllare che non esistano vie che permettano di concedere

diritti di accesso indesiderati.

**LEAK e SAFE**

- Una matrice M è detta LEAK (buggata) rispetto ad un permesso r se esiste un comando c che aggiunge il permesso r in una posizione della matrice che non lo conteneva.
- Una matrice M è detta SAFE (sicura) rispetto ad un permesso r se nessuna sequenza di comandi permette di portare lo stato della matrice nello stato LEAK.

Harrison Ruzzo e Ullman hanno enunciato i seguenti due teoremi:

1.*Teorema.* Data una matrice degli accessi M e un diritto r, verificare la sicurezza di M rispetto a r è un problema non decidibile.
*Dim.* Possiamo ridurre il problema di verificare la sicurezza di M al problema dell'arresto della macchina di Turing che sappiamo essere non decidibile (irrisolvibile).

2.*Teorema.* Dato un sistema composto di comandi mono-operazionali, una matrice degli accessi M e un diritto r, verificare la sicurezza del sistema rispetto al diritto r è decidibile.
*Dim.* Analizziamo il numero minimo di comandi per conferire un diritto, le operazioni come delete per diritti,oggetti e soggetti non vanno considerate in quanto non hanno effetto sull'espansione dei diritti di accesso, perciò la sequenza più breve per conferire un diritto contiene almeno m=|r|*(|s| +1)*(|o|+1)+1 comandi, dove |r| è il numero di diritti, |s| il numero dei soggetti e |o| il numero degli oggetti del sistema. La verifica consiste nel controllare tutte le sequenze di comandi di lunghezza m, se il diritto è stato conferito si troverà in una di queste squenze.

Un terzo teorema è stato enunciato da Lipton e Snyder nel 1978:

3.*Teorema.* Dato un sistema composto da un numero arbitrario di comandi, se il numero di soggetti è finito allora il sistema è decidibile.

In conclusione si può affermare che anche se non esiste un algoritmo capace di verificare se un sistema complesso è sicuro o meno, questo modello però non esclude la possibilità di prendere decisioni su particolari sistemi di protezione.

# 2. INFORMATION FLOW MODELS

Nel modello Bell-La Padula, l'informazione può fluire da un livello di sicurezza alto ad un livello basso attraverso un covert channel. I modelli Information Flow considerano ogni tipo di flusso d'informazione, non solo il flusso diretto attraverso operazioni di accesso come in Bell-La Padula. Informalmente una transizione di stato causa un fluire di informazione da un oggetto x ad un oggetto y, se possiamo capire meglio x osservando y. Se si sa già tutto su x, non può fluire informazione da x.

Bisogna distinguere due tipi di flusso di informazione:

- flusso esplicito: osservando y dopo l'assegnamento `y:=x`, si conosce il valore di x.
- flusso implicito: osservando y dopo la condizione `IF x=0 THEN y:=1`, è possibile dire qualcosa su x anche se l'assegnamento non è stato eseguito. Per esempio if y=2, sappiamo che x è diverso da 0.

## 2.1 ENTROPIA ED ENTROPIA CONDIZIONATA

L'entropia definisce la quantità d'informazione che può essere dedotta dall'osservazione di un oggetto o variabile. Dato un insieme di valori $\{x_1,...,x_n\}$ che una variabile x può assumere e la probabilità $p(x_i)$ che indica la probabilità per x di assumere i valori $x_i$ con $1 \leqslant i \leqslant n$ . L'**entropia** H(x) è definita come:

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i).$$

Esempio. Dato x che può assumere tutti i valori compresi 0 e $2^w - 1$ con uguale probabilità. Allora l'**entropia** sarà:

$$H(x) = -\sum_{i=1}^{2^w} 1/2^w \log_2(1/2^w) = w.$$

Questa è una parola binaria di lunghezza w che contiene w bit di informazione.

Il flusso d'informazione da x a y viene misurato dal cambiamento dell'**entropia** di x dipendente dal valore di y (**entropia condizionata**). Date due variabili x e y che possono assumere valori $\{x_1,...,x_n\}$ e $\{y_1,...,y_m\}$ con probabilità $p(x_i)$ e $q(y_j)$, data la probabilità congiunta $p(x_i,y_j)$ per x e y di prendere i valori $x_i$ e $y_j$, e data la probabilità condizionata $p(x_i|y_j)$ per x di assumere il valore $x_i$ se y assume il valore $y_j$ l'**entropia condizionata** $H_y(x)$ di x è definita come:

$$H_y(x) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i, y_j)\log_2 p(x_i|y_j).$$

Riscrivendo $p(x_i,y_j)$ come $p(x_i|y_j)q(y_j)$ si ottiene:

$$H_y(x) = -\sum_{j=1}^{m} q(y_j)\sum_{i=1}^{n} p(x_i|y_j)\log_2 p(x_i|y_j).$$

Come esempio consideriamo l'assegnamento IF x=0 THEN y:=1, come prima. Date le variabili binarie x e y con y inizialmente settata a 0 e con i valori di x aventi la stessa probabilità, si ottiene:

$$p(0|0) = p(1|1) = 0,\ \ p(1|0) = p(0|1) = 1 \quad \text{quindi} \quad H_y(x) = 0.$$

Dopo l'esecuzione dell'assegnamento osservando y si può conoscere il valore esatto di x. Tutta l'informazione di x è fluita in y. Se x può assumere i valori 0, 1, 2 con uguale probabilità, si ottiene:

$$q(0) = 2/3,\ \ q(1) = 2/3$$

$$p(0|0) = p(1|1) = p(2|1) = 0,\ \ p(1|0) = p(2|0) = 1/2,\ \ p(0|1) = 1, \ \text{e} \quad H_y(x) = 2/3.$$

## 2.2 A LATTICE-BASED MODEL

I componenti di un modello Information Flow sono:

- un reticolo "lattice" $(L,\leq)$ di livelli di sicurezza;
- un insieme di oggetti etichettati;
- una politica di sicurezza: il flusso d'informazione da un oggetto con etichetta c1 ad un oggetto con

etichetta c2 è permesso se e solo se c1≤c2; ogni flusso d'informazione che viola tale regola non è ammesso.

Un sistema è detto sicuro se non esistono flussi d'informazione non ammessi. Il vantaggio di un modello così strutturato è che può coprire tutti i tipi di flussi. Lo svantaggio è che può diventare molto difficile realizzare un sistema sicuro. Infatti come già detto in precedenza verificare la sicurezza di un sistema è un problema non decidibile.

## 2.3 RICONOSCERE I FLUSSI DI INFORMAZIONE

Distinguiamo l'analisi delle politiche di sicurezza tra *statica* e *dinamica*. Nel primo caso il sistema è considerato come un oggetto statico, nel secondo caso il sistema viene analizzato a tempo di esecuzione. La differenza tra i due tipi di analisi e che nell'analisi *dinamica* bisogna assicurare che il flusso di informazione, che in teoria è possibile effettuare, non si verifichi durante l'esecuzione, di conseguenza l'analisi *statica* tende a produrre un sistema restrittivo.
Due meccanismi che permettono l'analisi del flusso di informazione sono il Compiler-based che lavora a tempo di compilazione quindi staticamente e l'Execution-based che effettua un controllo dinamico. Entrambi analizzano il codice di un programma.
Bisogna distinguere i costrutti da:

- costrutti di assegnamento
- costrutti condizionali e di iterazione.

*Definizione*. Un insieme di costrutti è certificato rispetto ad una politica di information flow se il flusso d'informazione che passa attraverso l'insieme di costrutti non viola la politica.

Si utilizzerà la notazione $\underline{x} \leqslant \underline{y}$ per indicare che l'informazione può fluire da un elemento di classe x ad un elemento di classe y.

### *2.3.1 Compiler-based*

Il meccanismo Compiler-based determina se i flussi di informazione in un programma possano violare la politica di information flow, questo controllo non è preciso in quanto percorsi di flusso d'informazione sicuri potrebbero essere marcati come violazioni della politica, ma il meccanismo è sicuro poiché tutti i percorsi non autorizzati lungo i quali l'informazione potrebbe fluire saranno

individuati.

Esempio. Consideriamo il seguente costrutto condizionale, meccanismo Compiler-based:

```
if x=1 then y:=a;
else y:=b;
```

Si hanno flussi di informazione da x e a in y oppure da x e b in y, così se la politica dice che $\underline{a} \leqslant \underline{y}$ , $\underline{b} \leqslant \underline{y}$ e $\underline{x} \leqslant \underline{y}$ allora il flusso d'informazione è sicuro. Ma se la politica richiede che $\underline{a} \leqslant \underline{y}$ è ammesso solo quando un'altra variabile z=1, il meccanismo Compiler-based deve determinare anche se z=1 prima di certificare il costrutto. Questo, è tipicamente irrealizzabile. Quindi tale meccanismo potrebbe non certificare il costrutto.

Si assume che i flussi ammessi sono forniti al meccanismo di controllo da una fonte esterna, ad esempio un file. Le caratteristiche dei flussi ammessi riguardano le classi di sicurezza dei costrutti del linguaggio. Il programma coinvolge le variabili, così i costrutti del linguaggio devono relazionare le variabili alle classi di sicurezza.

Una possibile scelta è assegnare ogni variabile esattamente ad una classe di sicurezza. Ma è possibile scegliere anche un approccio più liberale in cui i costrutti del linguaggio specificano l'insieme di classi dal quale l'informazione potrebbe fluire nella variabile.

Per esempio :

x: integer class {A,B}

indica che x è una variabile intera e che i dati possano fluire dalle classi di sicurezza A e B in x. Le classi sono assegnate staticamente. Guardando le classi come un lattice, si ottiene che le classi di x devono essere l'insieme più piccolo dei least upper bound delle classi A e B, quindi si ha $lub\{A, B\} \leqslant \underline{x}$.

Le classi Low e High sono due classi distinte e rappresentano rispettivamente il greatest lower bound e il least upper bound di un lattice. Tutte le costanti sono di classe Low.

Esempio. Consideriamo il seguente costrutto di assegnamento:

```
x:=y+z;
```

il requisito per far sì che il flusso di informazione sia sicuro è $lub\{\underline{y}, \underline{z}\} \leqslant \underline{x}$.

Nel caso in cui il costrutto è di tipo iterativo

```
while f(x1,...,xn) do
        S;
```

i requisiti per rendere il costrutto sicuro sono:

1.l'iterazione deve terminare;

2.S sia sicuro;

3. $lub\{\underline{x1},\ldots,\underline{xn}\} \leqslant glb\{\underline{y} | y \text{ è l'obbiettivo di un assegnamento in } S\}$.

*2.3.2 Execution-based*

Il meccanismo Execution-based ha come obbiettivo quello di prevenire i flussi di informazione che violano la politica. Esso raggiunge tale obbiettivo per costrutti che contengono flussi di tipo esplicito, il controllo per i flussi di tipo implicito è complicato e si fa uso della Fenton's Data Mark Machine. Tale macchina che prende il nome dal suo creatore prevede che ogni variabile sia associata ad una classe di sicurezza o tag, essa include anche un tag per il contatore di programma (PC). Quest'ultima inclusione permise a Fenton di trattare i flussi impliciti come espliciti.

Esempio. Date due variabili x e y. Il requisito per certificare un particolare costrutto y operazione x è che $\underline{x} \leqslant \underline{y}$.
Il costrutto condizionale

if x=1 then y:=a;

causa un flusso da x a y. Supposto che quando $x \neq 1$, $\underline{x} = High$ e $\underline{y} = Low$. Se i flussi sono verificati solo quando sono espliciti, e $x \neq 1$, il flusso implicito non sarà controllato. Quindi il costrutto potrebbe essere certificato come conforme alla politica.

## 2.4 NONINTERFERENCE MODELS

I modelli Noninterference sono un alternativa ai modelli Information Flow. Essi prevedono un formalismo differente per descrivere quello che un soggetto conosce sullo stato del sistema. Il soggetto s1 non interferisce con il soggetto s2 se le azioni di s1 non influenzano ciò che s2 può vedere (s1 e s2 appartengono a due livelli differenti). Oggigiorno, i modelli Information Flow e Noninterference sono argomenti di ricerca piuttosto che le basi di una metodologia pratica per realizzare dei sistemi sicuri.

# 3. EXECUTION MONITORS

Come già detto verificare la sicurezza di un sistema rappresenta un problema non decidibile. Non esiste un algoritmo generale che risolve tutte le istanze di questo problema. Adesso, le nostre investigazioni teoriche seguiranno una strada differente. Inizieremo dai meccanismi tipici di access control in uso oggigiorno e definiremo le politiche che questi meccanismi possono analizzare. Consideriamo tre classi di politiche di sicurezza (Schneider, 2000):

- le politiche di Access Control definiscono delle restrizioni sulle operazioni principali che possono essere eseguite su un oggetto;
- le politiche di Information Flow limitano le operazioni principali che possono dedurre informazione su di un oggetto osservando il comportamento del sistema;
- le politiche di Availability limitano le operazioni principali negando l'utilizzo di una risorsa agli altri.

I meccanismi di access control oggigiorno sono presenti in firewalls, sistemi operativi, architetture middleware come CORBA, o in web services essi hanno in comune il fatto che osservano (monitorano) l'esecuzione del sistema ed intervengono se un passo d'esecuzione è vietato dalla politica di sicurezza. Il termine Execution Monitoring (EM) è stato introdotto da Schneider per meccanismi che monitorano i passi d'esecuzione di un sistema e termina l'esecuzione se si verifica una violazione della politica di sicurezza.

Gli Execution Monitors hanno due importanti limitazioni. Primo, essi non hanno un modello del sistema target così non possono predire i risultati di una possibile continuazione dell'esecuzione che stanno osservando. I compilatori, per esempio, lavorano analizzando una rappresentazione statica del target e possono dedurre informazioni su tutte le sue possibili esecuzioni. Questi metodi non sono tuttavia meccanismi EM. Secondo, i meccanismi EM non possono modificare un target prima di eseguirlo. Gli In-line reference monitors e la programmazione Reflection-oriented non rientrano nella categoria degli EM.

Un In-line reference monitor si ottiene modificando un applicazione, includendo le funzionalità di un reference monitor. Mentre la programmazione Reflection-oriented rappresenta un'estensione del paradigma della programmazione object-oriented che permette di monitorare l'esecuzione di un programma e di modificarne la sua struttura e il comportamento.

## 3.1 PROPERTIES OF EXECUTION

L'esecuzione di un sistema target è costituita da una sequenza di passi. La natura precisa di questi passi dipenderà dal target attuale. Esempi tipici sarebbero le operazioni di accesso alla memoria o a file. Nella nostra discussione generale $\Psi$ indica l'insieme di tutte le sequenze di passi finite e infinite, e $\Sigma_S$ indica le sequenze rappresentanti l'esecuzione di un sistema target S. Una politica di sicurezza p è definita come un predicato sull'insieme delle esecuzioni. Un sistema target S soddisfa la politica di sicurezza p se $p(\Sigma_S)$ è uguale a true.
$\Sigma$ indica un insieme di esecuzioni. Una politica di sicurezza p per essere analizzata da un execution monitor deve essere specificata da un predicato della forma:

$$p(\Sigma) : (\forall \sigma \in \Sigma : \hat{p}(\sigma))$$

dove $\hat{p}$ è un predicato su esecuzioni individuali. Questa osservazione fornisce un collegamento alla letteratura sulla verifica del linear-time della programmazione concorrente.
Un insieme $\Gamma \subset \Psi$ di esecuzioni è detto proprietà se i membri di un elemento è determinato dal singolo elemento, non da altri elementi dell'insieme. Una politica di sicurezza deve quindi essere una proprietà per avere un meccanismo di analisi che rientra in EM.

Comunque, non tutte le politiche di sicurezza sono una proprietà. Alcune politiche non possono essere definite come un predicato su esecuzioni individuali. Per esempio, le politiche di Information Flow richiedono che un utente di basso livello non possa distinguere una esecuzione dove un utente di alto livello è attivo da un'altra esecuzione dove l'utente di alto livello non è attivo.

Inoltre, non tutte le proprietà sono analizzabili da meccanismi EM. I meccanismi di analisi che rientrano nella categoria EM non possono prevedere i passi successivi quando viene presa una decisione su un'esecuzione. Consideriamo una esecuzione $\sigma$ che è conforme alla politica di sicurezza ma ha un prefisso $\sigma'$ che non lo è. Informalmente, l'esecuzione passa attraverso un stato "non sicuro" ma probabilmente gli è permesso terminare. Come semplice esempio consideriamo una politica che richiede un matching del comando 'close file' per ogni comando 'open file' . Un execution monitor deve proibire un prefisso non sicuro e fermare l'esecuzione. Per tali politiche EM rappresenterebbe un approccio conservativo che ferma più esecuzioni del necessario.

### 3.2 SAFETY AND LIVENESS

Tra le proprietà delle esecuzioni ci sono due classi ampie dal significato particolare.

- Le proprietà safety: nulla di sbagliato può accadere.
- Le proprietà liveness: eventualmente qualcosa di buono accadrà.

Esiste una relazione chiusa tra safety e il tipo di politica che si vuole analizzare mediante l'execution monitor. Definiremo formalmente le proprietà safety caratterizzando i loro complementi. Nella definizione i primi i passi di una sequenza $\sigma \in \Psi$ saranno rappresentati da σ[..i]. Una proprietà Γ è detta proprietà safety se per ogni esecuzione finita o infinita σ

$$\sigma \notin \Gamma \Rightarrow \exists i (\forall \tau \in \Psi : \sigma[..i]\tau \notin \Gamma).$$

Se un'esecuzione σ non è sicura, devono esserci alcuni punti i nell'esecuzione dopo i quali non sia più possibile ritornare ad una continuazione sicura dell'esecuzione.

Se l'insieme delle esecuzioni per una politica di sicurezza non è una proprietà safety, allora esiste un'esecuzione non sicura che potrebbe essere estesa dai passi successivi in una esecuzione sicura. Come già discusso in precedenza tali proprietà non hanno un meccanismo di analisi appartenente alla categoria EM. Così, se una politica non è una proprietà safety, non è analizzabile. Detto in altre parole gli execution monitors analizzano politiche di sicurezza che sono proprietà safety.

Ritornando alle tre classi di politiche di sicurezza si può affermare:

- le politiche Access Control definiscono proprietà safety; esecuzioni parziali che terminano con la prova di un'operazione inaccettabile saranno vietate.
- le politiche Information Flow non definiscono insiemi di esecuzioni che sono proprietà; di conseguenza Information Flow non può essere una proprietà safety e non può essere analizzato da EM.
- le politiche Availability non definiscono proprietà safety; qualche esecuzione parziale potrebbe essere estesa così che permetta un accesso alla risorsa.

Le politiche Availability che si riferiscono al Maximum Waiting Time (MWT) sono proprietà safety. Non appena un'esecuzione resta in attesa superando il valore corrispondente a MWT qualsiasi estensione sarà considerata una violazione della politica.