

Protezione e sicurezza

I meccanismi di protezione offrono un accesso controllato limitando i tipi d'accesso ai file permessi agli utenti. La protezione deve anche assicurare che solo i processi che hanno ottenuto l'autorizzazione dal sistema operativo possano usare i segmenti di memoria, la CPU e altre risorse.

La protezione è assicurata da un meccanismo che controlla l'accesso di programmi, processi o utenti alle risorse di un sistema elaborativo. Questo meccanismo deve offrire uno strumento per specificare quali siano i controlli da impostare, e un sistema che permetta di farli rispettare.

La sicurezza garantisce l'autenticazione degli utenti del sistema per proteggere l'integrità delle informazioni (dati e codice) in esso memorizzate e delle risorse fisiche di cui il sistema elaborativo dispone. Il sistema di sicurezza impedisce gli accessi non autorizzati al sistema, i tentativi dolosi di distruzione o alterazione dei dati, e l'introduzione accidentale d'incoerenze.

CAPITOLO

14

OBIETTIVI DEL CAPITOLO

- Obiettivi e principi relativi alla protezione in un moderno sistema elaborativo.
- Analisi dell'utilizzo di domini di protezione combinati con matrici d'accesso per la specifica di risorse accessibili da un processo.
- Valutazione di sistemi di protezione basati su abilitazione e sul linguaggio.

Protezione

I processi di un sistema operativo devono essere protetti dalle attività altrui. Molteplici meccanismi provvedono al raggiungimento di tale scopo, in modo che file, segmenti di memoria, cpu e altre risorse possano essere adoperati solo dai processi che hanno ottenuto l'autorizzazione dal sistema operativo.

La protezione riguarda il meccanismo per il controllo dell'accesso alle risorse definite da un sistema elaborativo da parte di programmi, processi o utenti. Questo meccanismo deve fornire un metodo per specificare i controlli da imporre e alcuni mezzi per garantirli. È necessario fare una distinzione tra protezione e sicurezza; quest'ultima è una misura della fiducia sul mantenimento dell'integrità di un sistema e dei suoi dati. In questo capitolo tratteremo la protezione. La garanzia della sicurezza è un argomento più generico rispetto alla protezione, ed è trattato nel capitulo 15.

14.1 Scopi della protezione

I calcolatori sono divenuti più complessi e le loro applicazioni sono cresciute a dismisura; di conseguenza è aumentata anche la necessità di proteggere la loro integrità. La protezione era originariamente concepita come un elemento aggiuntivo dei sistemi operativi con multiprogrammazione tale che utenti non fidati potessero condividere tranquillamente uno spazio di nomi logici comuni, come una directory di file, oppure uno spazio di nomi fisici comuni, come la memoria. I moderni concetti di protezione si sono evoluti in modo da aumentare l'affidabilità di qualsiasi sistema complesso che usi risorse condivise.

È necessario disporre di sistemi di protezione per diversi motivi. Tra i più ovvi c'è la necessità di prevenire la violazione intenzionale e dannosa di un vincolo d'accesso da parte di un utente. Tuttavia, ha un'importanza più generale la necessità di assicurare che ogni componente del programma attivo in un sistema impieghi le risorse del sistema in modo coerente con i criteri (policies) stabiliti; per un sistema affidabile, questo requisito è assolutamente necessario.

La protezione può migliorare l'affidabilità rilevando errori latenti nelle interfacce tra sottosistemi componenti. Un'individuazione precoce di errori d'interfaccia riesce spesso a impedire la contaminazione di un sottosistema intatto da parte di un sottosistema malfunzionante. Una risorsa non protetta non può difendersi contro l'uso, o l'abuso, da parte di un utente non autorizzato o incompetente. Un sistema orientato alla protezione offre i mezzi per distinguere tra uso autorizzato e non autorizzato.

Il ruolo della protezione è quello di offrire un meccanismo d'imposizione di criteri che controllino l'uso delle risorse. Questi criteri si possono stabilire in vari modi: alcuni sono stati fissati nella fase di progettazione del sistema, altri sono determinati dalla gestione di un sistema; altri ancora sono definiti dai singoli utenti per proteggere i loro file e programmi. Un sistema di protezione deve avere una flessibilità tale da consentire d'imporre vari tipi di criteri.

I criteri d'uso di una risorsa possono variare secondo l'applicazione e possono cambiare nel tempo. Per queste ragioni la protezione non è più una questione riguardante solamente i progettisti di un sistema operativo; anche i programmatore di applicazioni hanno bisogno di meccanismi per proteggere dagli abusi le risorse create e gestite dai sottosistemi applicativi. In questo capitolo si descrivono i meccanismi di protezione che il sistema operativo dovrebbe fornire, e che anche i progettisti di applicazioni possono impiegarli nella progettazione del proprio sistema di protezione.

I criteri vanno distinti dai meccanismi. I meccanismi determinano come qualcosa si debba eseguire; i criteri decidono che cosa si debba fare. La distinzione tra criteri e meccanismi è importante ai fini della flessibilità. I criteri sono soggetti a cambiamenti in dipendenza dall'ambiente e dal tempo. Nel peggior dei casi qualsiasi cambiamento di criterio richiederebbe un cambiamento anche nel meccanismo sottostante. Utilizzando meccanismi generali si possono evitare tali situazioni.

14.2 Princìpi di protezione

Spesso un principio guida può essere utilizzato attraverso un intero progetto, quale per esempio la progettazione di un sistema operativo. L'osservanza di questo principio semplifica le decisioni relative al progetto, garantendo che il sistema sia coerente e facile da capire. Un principio guida, che nel tempo ha confermato la sua importanza per la protezione, è il principio del privilegio minimo. Esso sancisce che programmi, utenti, e finanche i sistemi, devono ricevere solo le risorse minime necessarie all'esecuzione dei rispettivi compiti.

Si consideri l'analogia di una guardia di vigilanza in possesso di un passe-partout. Se, con questa chiave, la guardia può accedere solamente alle aree pubbliche che controlla, l'uso improprio della chiave provocherà un danno minimo. Se però la chiave è per tutte le aree, qualora venga smarrita, rubata, usata impropriamente, duplicata o in alcun modo alterata, la rilevanza del danno a cui essa espone sarà molto maggiore.

Un sistema operativo fondato sul principio del privilegio minimo implementa le proprie funzionalità, programmi, chiamate di sistema e strutture dati in modo da contenere al minimo il danno derivante dal guasto o dal funzionamento indebito di un componente. L'overflow del buffer di un demone di sistema, per esempio, potrebbe causare un malfunzionamento del demone, ma non dovrebbe permettere l'esecuzione di codice dallo stack del relativo processo, permettendo che un utente remoto guadagni i massimi privilegi e possa accedere all'intero sistema (come troppo spesso accade al giorno d'oggi).

Un tale sistema operativo, inoltre, deve realizzare le chiamate di sistema e i servizi in modo da consentire alle applicazioni il controllo degli accessi con elevata granularità. Il sistema deve fornire meccanismi per attivare i privilegi quando sono necessari e per disattivarli quando non lo sono. Altrettanto utile è la creazione di tracciature di verifica (audit trails) relative all'accesso a tutte le funzioni privilegiate. Un audit trail permette al programmatore, all'amministratore del sistema, o al rappresentante dell'autorità giudiziaria, di ricostruire tutte le attività di protezione e di sicurezza compiute nel sistema.

Applicare il principio del privilegio minimo agli utenti comporta la creazione di un account per ciascun utente, ognuno dei quali prevede i minimi privilegi necessari. Un operatore che debba montare nastri e creare copie di backup sul sistema avrà accesso solo a quei comandi e a quei file che gli sono necessari per ultimare il lavoro. Alcuni sistemi adottano il controllo dell'accesso basato sui ruoli (rb Ac) per fornire questa funzionalità.

Quando un centro di calcolo rispetta il principio del privilegio minimo, gli elaboratori che ne fanno parte possono offrire ciascuno l'esecuzione di specifici servizi, o l'accesso a macchine remote tramite servizi prefissati e in orari predeterminati. Queste restrizioni sono normalmente messe in atto grazie all'attivazione o disattivazione di ciascun servizio e mediante liste di controllo degli accessi, come spiegato nei paragrafi 11.6.2 e 14.6.

Il principio del privilegio minimo può contribuire a creare un ambiente elaborativo più sicuro; spesso, però, non è così. Il sistema Windows 2000, per esempio, malgrado

abbia alla base una complessa struttura di protezione, rivela numerose falle nella sicurezza. In confronto, Solaris è considerato relativamente sicuro, anche se deriva da u NIx che, agli inizi, non ha fatto della protezione una sua priorità. La spiegazione di questa differenza potrebbe risiedere nel fatto che Windows 2000 ha una maggior quantità di servizi e di linee di codice rispetto a Solaris, e quindi più risorse da proteggere. Può anche darsi, però, che lo schema di protezione di Windows 2000 sia incompleto, o protegga aspetti irrilevanti del sistema, lasciandone altri vulnerabili.

14.3 Domini di protezione

Un sistema elaborativo è un insieme di processi e oggetti. Con il nome di oggetti si designano sia gli oggetti fisici, come `cpu`, segmenti di memoria, stampanti, dischi e unità a nastri, sia gli oggetti logici, come file, programmi e semafori. Ogni oggetto ha un nome unico che lo distingue da tutti gli altri oggetti del sistema; è inoltre possibile accedere a ciascuno di loro solo tramite operazioni ben definite e significative. Gli oggetti sono fondamentalmente tipi di dati astratti.

Le operazioni possibili dipendono dall'oggetto: per esempio, una `cpu` può compiere solo elaborazioni; i segmenti di memoria si possono leggere e scrivere, mentre da un lettore di `cd -r OM` o `dvd -r OM` si può soltanto leggere; le unità a nastri si possono leggere, scrivere e riavvolgere; i file di dati si possono creare, aprire, leggere, scrivere, chiudere e cancellare; i file di programmi si possono leggere, scrivere, eseguire e cancellare.

A un processo si deve permettere l'accesso alle sole risorse su cui ha l'autorizzazione. Inoltre, a un dato istante, un processo deve poter accedere solamente alle risorse di cui ha bisogno per eseguire il proprio compito. Questo secondo requisito, noto con il nome di principio della necessità di sapere (need-to-know-principle), è utile per limitare i danni che possono essere causati al sistema da un processo difettoso. Se, per esempio, il processo P invoca la procedura `A()`, alla procedura si deve permettere l'accesso solo alle proprie variabili e ai parametri che le vengono passati; non deve poter accedere a tutte le variabili del processo P. Analogamente, si consideri il caso in cui il processo P richieda la compilazione di un particolare file. Al compilatore non si deve permettere l'accesso a qualsiasi file, ma solo al ben definito sottinsieme di file associato al file da compilare. Viceversa, il compilatore può avere dei file privati, che impiega per scopi di accounting o di ottimizzazione, ai quali il processo P non deve aver accesso. Il principio della necessità di sapere somiglia al principio del privilegio minimo illustrato nel Paragrafo 14.2, in quanto gli scopi della protezione consistono nel minimizzare i rischi di possibili violazioni alla sicurezza.

14.3.1 Struttura dei domini di protezione

Per facilitare lo schema appena descritto, un processo opera all'interno di un dominio di protezione, che specifica le risorse accessibili dal processo. Ogni dominio definisce un insieme di oggetti e i tipi di operazioni che si possono compiere su ciascun

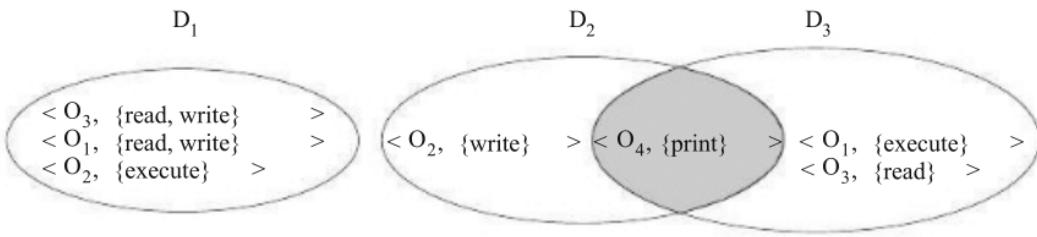


Figura 14.1 Sistema con tre domini di protezione.

oggetto. La possibilità di eseguire un'operazione su un oggetto è detta diritto d'accesso. Un dominio è dunque un insieme di diritti d'accesso, ciascuno dei quali è composto di una coppia ordinata \langle nome oggetto, insieme di diritti \rangle . Per esempio, se il dominio D ha il diritto d'accesso \langle file F , $\{read, write\}$ \rangle , un processo in esecuzione nel dominio D può leggere e scrivere il file F , ma non può eseguire altre operazioni su quello stesso oggetto.

I domini possono condividere diritti d'accesso. Nella Figura 14.1, per esempio, sono illustrati tre domini: D_1 , D_2 e D_3 . Il diritto d'accesso $\langle O_4, \{print\} \rangle$ è condiviso da D_2 e D_3 , implicando che un processo in esecuzione su uno dei due domini può stampare l'oggetto O_4 . Occorre notare che un processo, per leggere e scrivere l'oggetto O_1 , deve essere in esecuzione nel dominio D_1 . D'altra parte, soltanto i processi che si trovano nel dominio D_3 possono eseguire l'oggetto O_1 .

L'associazione tra un processo e un dominio può essere statica, se l'insieme delle risorse disponibili per un processo è fisso per tutta la durata del processo, o dinamica. Com'è prevedibile, la determinazione dei domini di protezione dinamici è più complicata della determinazione dei domini di protezione statici.

Se l'associazione tra processi e domini è fissa, per aderire al principio del privilegio minimo è necessario disporre di un meccanismo che permetta di modificare il contenuto di un dominio. Ciò deriva dal fatto che l'esecuzione di un processo si può dividere in più fasi e, ad esempio, il processo può richiedere l'accesso in lettura in una fase e l'accesso in scrittura in un'altra. Se un dominio è statico, occorre definire un dominio che contenga sia l'accesso per la lettura sia quello per la scrittura. Tuttavia questa disposizione fornisce più diritti di quanti siano necessari in ciascuna delle due fasi, poiché è disponibile il diritto d'accesso per la lettura nella fase in cui è necessario il solo accesso per la scrittura e viceversa; quindi il principio del privilegio minimo è violato. È necessario permettere che il contenuto di un dominio sia modificato, in modo che il dominio rifletta sempre i minimi diritti d'accesso necessari.

Se l'associazione è dinamica, deve essere disponibile un meccanismo per permettere a un processo di passare da un dominio all'altro (domain switching). Si può anche voler modificare al contenuto di un dominio. Se non è possibile modificare il contenuto di un dominio, si può ottenere lo stesso effetto creando un nuovo dominio con il contenuto modificato e passando al nuovo dominio quando sia necessario modificare il contenuto del dominio originario.

Si noti che un dominio si può realizzare in diversi modi.

- Ogni utente può essere un dominio. In questo caso l'insieme d'oggetti cui si può accedere dipende dall'identità dell'utente. Il cambio di dominio avviene quando cambia l'utente, generalmente quando un utente chiude una sessione di lavoro e un altro la comincia.
- Ogni processo può essere un dominio. In questo caso l'insieme d'oggetti cui si può accedere dipende dall'identità del processo. Il cambio di dominio avviene quando un processo invia un messaggio a un altro processo e quindi attende una risposta.
- Ogni procedura può essere un dominio. In questo caso l'insieme d'oggetti cui si può accedere corrisponde alle variabili locali definite all'interno della procedura. Il cambio di dominio avviene quando s'invoca una procedura.

Il cambio di dominio è approfondito nel paragrafo 14.4.

Si consideri l'ordinaria duplice modalità di funzionamento (di sistema e utente) dei sistemi operativi. Quando si esegue un processo in modalità di sistema, esso può impiegare istruzioni privilegiate e quindi acquisire il controllo completo del calcolatore. D'altra parte, se è eseguito in modalità utente, il processo può impiegare solo istruzioni non privilegiate; di conseguenza, può essere eseguito solo all'interno del proprio spazio di memoria predefinito. Questi due modi proteggono il sistema operativo, che è eseguito nel dominio di sistema, dai processi utenti, che si eseguono nel dominio dell'utente. In un sistema operativo con multiprogrammazione due domini di protezione sono insufficienti, poiché gli utenti richiedono anche la protezione reciproca. Serve quindi uno schema più elaborato: lo illustreremo esaminando due influenti sistemi operativi, *UNIX* e *MULTICS*, e osservando come realizzano questi concetti.

14.3.2 Un esempio: *UNIX*

Nel sistema operativo *UNIX* un dominio è associato all'utente. Il cambio di dominio corrisponde al cambio temporaneo dell'identificatore dell'utente. Questo cambio si compie tramite il file system. A ogni file sono associati un identificatore di proprietario e un bit di dominio (noto con il nome di setuid bit). Quando il setuid bit è on, e un utente esegue quel file, lo userID è impostato a quello del proprietario del file; quando il setuid bit è off lo userID non cambia. Ad esempio, quando un utente, con userID uguale ad A, avvia l'esecuzione di un file posseduto da B, il cui setuid bit è off, l'identificatore del processo viene impostato ad A. Quando il setuid bit è on, lo userID viene impostato a quello del proprietario del file, in questo caso B. Quando il processo termina, cessa anche quest'impostazione temporanea dello userID.

Per cambiare domini nei sistemi operativi in cui i domini sono definiti dagli identificatori degli utenti s'impiegano anche altri metodi. Quasi tutti i sistemi hanno bisogno di un tale meccanismo, che si usa per rendere disponibile a tutti gli utenti una funzione il cui utilizzo è normalmente privilegiato. Per esempio, è auspicabile che tutti gli utenti possano ottenere il permesso di accedere a una rete senza che ciascuno debba

scrivere il proprio programma di interfacciamento con la rete. In tal caso in un sistema u NIx si attiva il bit setuid di un programma di rete con la conseguenza che l'identificatore dell'utente cambia quando il programma è in esecuzione: l'identificatore di un utente con il privilegio d'accesso alla rete (come root , l'utente più importante) sostituisce il corrente identificatore dell'utente. Un problema che si pone con questo metodo è che se un utente riesce a creare un file con identificatore root e con il setuid bit posto a on, può assumere l'identità root e fare qualsiasi operazione sul sistema.

Un metodo alternativo, usato in altri sistemi operativi, prevede l'inserimento dei programmi privilegiati in una directory speciale. In questo caso il sistema operativo è progettato in modo da cambiare al momento dell'esecuzione l'identificatore dell'utente di ogni programma residente in questa directory, rendendolo equivalente a root o all'identificatore dell'utente proprietario della directory. Ciò elimina il problema di sicurezza che si verifica quando intrusi creano programmi per manipolare la funzione setuid e nascondere i programmi presenti nel sistema per usi futuri. Questo metodo è comunque meno flessibile di quello usato nello u NIx .

Ancora più restrittivi, e quindi più protettivi, sono i sistemi che semplicemente non permettono di modificare l'identificatore dell'utente. In questi casi è necessario ricorrere a speciali tecniche per permettere agli utenti di accedere a funzioni privilegiate. Per esempio, si può attivare un processo demone nella fase d'avviamento del sistema ed eseguirlo con uno speciale userID . Gli utenti quindi eseguono un programma distinto che invia richieste a questo processo ogni volta che hanno bisogno di usare la relativa funzione. Questo metodo è impiegato dal sistema operativo TOPS-20.

In ciascuno di questi sistemi la scrittura dei programmi privilegiati si deve realizzare con molta cura: qualsiasi svista può causare una totale mancanza di protezione del sistema. Di solito questi programmi sono i primi a essere attaccati dalle persone che tentano di penetrare in un sistema; sfortunatamente tali attacchi hanno spesso successo; per esempio, la sicurezza è stata infranta in molti sistemi u NIx attraverso la funzionalità del setuid. La sicurezza è trattata nel capitulo 15.

14.3.3 Un esempio: mULTICS

Nel sistema Mu LTIC S i domini di protezione sono organizzati gerarchicamente in una struttura ad anelli, numerati da 0 a 7. Ciascun anello corrisponde a un dominio (Figura 14.2). Si supponga che D_i e D_j siano due anelli di dominio. Se $j < i$, D_i è un sottoinsieme di D_j ; ciò significa che un processo in esecuzione nel dominio D_j ha più privilegi di quanti ne abbia uno in esecuzione nel dominio D_i . Un processo in esecuzione nel dominio D_0 ha più privilegi di tutti. Se ci sono due soli anelli, questo schema corrisponde alla modalità d'esecuzione di sistema e utente, dove la modalità di sistema corrisponde a D_0 e la modalità utente a D_1 .

Mu LTIC S ha uno spazio d'indirizzi segmentato; ogni segmento è un file ed è associato a uno degli anelli. Un descrittore del segmento include un elemento che identifica il numero dell'anello. Inoltre contiene tre bit d'accesso per il controllo di lettura, scrittura ed esecuzione. L'associazione tra segmenti e anelli è una scelta che riguarda le policy e che non esamineremo.

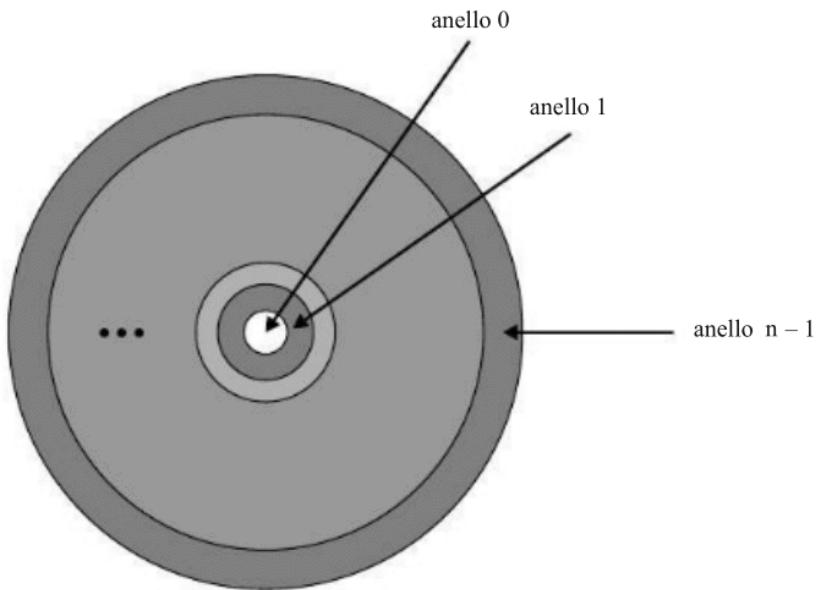


Figura 14.2 Struttura ad anelli del sistema multic S.

A ogni processo è associato un contatore del numero corrente d'anello , che identifica l'anello in cui il processo è attualmente in esecuzione. Quando un processo è in esecuzione nell'anello i , non può accedere a un segmento associato all'anello j , tale che $j < i$, ma può accedere a un segmento associato all'anello k , tale che $k \geq i$. comunque, il tipo d'accesso è limitato dai bit d'accesso associati a quel segmento.

In Mu LTIc S il cambio dei domini avviene quando un processo passa da un anello all'altro invocando una procedura in un anello diverso. Naturalmente questo cambio si deve compiere in modo controllato, altrimenti un processo potrebbe iniziare l'esecuzione nell'anello 0, senza che sia garantita alcuna protezione. Per permettere un cambio di dominio controllato, occorre modificare il campo concernente l'anello del descrittore del segmento inserendovi le seguenti informazioni.

- Un intervallo d'accesso. una coppia d'interi b_1 e b_2 tali che $b_1 \leq b_2$.
- Un limite. un intero b_3 tale che $b_3 > b_2$.
- Una lista degli ingressi (o gates). Identifica i punti d'accesso (ingressi) da cui si possono invocare i segmenti.

Se un processo in esecuzione nell'anello i invoca una procedura (segmento) con intervallo d'accesso (b_1, b_2) , la chiamata è ammessa se $b_1 \leq i \leq b_2$ e il numero corrente d'anello del processo rimane i . Altrimenti s'invia un segnale di eccezione al sistema operativo e la situazione viene gestita come segue.

- Se $i < b_1$, si può eseguire la chiamata, poiché si ha un trasferimento a un anello (dominio) con meno privilegi. Tuttavia, se si passano parametri che si riferiscono a segmenti in un anello inferiore (vale a dire segmenti che non sono accessibili alla procedura invocata), questi segmenti devono essere copiati in un'area alla quale può accedere anche la procedura invocata.

- Se $i > b_2$, si può eseguire la chiamata solo se b_3 è maggiore o uguale a i e la chiamata è indirizzata a uno dei punti d'accesso stabiliti nella lista. Questo schema permette a processi con diritti d'accesso limitati di invocare, ma solo in modo attentamente controllato, procedure che si trovano in anelli inferiori e che hanno più diritti d'accesso.

Lo svantaggio maggiore legato alla struttura (gerarchica) ad anelli consiste nel fatto che non consente l'applicazione del principio della necessità di sapere. In particolare, se un oggetto deve essere accessibile nel dominio D_j , ma non nel dominio D_i , allora deve essere $j < i$, ma ciò significa che ogni segmento accessibile in D_i è accessibile anche in D_j .

Il sistema di protezione di $M^u L T I^c S$ è in generale più complesso e meno efficiente di quello adoperato nei correnti sistemi operativi. Se la protezione interferisce con la comodità d'uso del sistema, o ne riduce significativamente le prestazioni, il suo impiego deve essere attentamente valutato rispetto agli scopi del sistema. Per esempio, si potrebbe volere un complesso sistema di protezione in un calcolatore usato in un'università per gestire i voti degli studenti, e usato anche dagli studenti per i loro compiti. Un simile sistema di protezione non sarebbe adatto a un calcolatore che s'impiega per elaborazioni numeriche, in cui le prestazioni sono della massima importanza. Sarebbe quindi preferibile separare i meccanismi di protezione dai criteri di protezione, permettendo allo stesso sistema di avere una protezione complessa o semplice secondo le necessità dei propri utenti. Per separare i meccanismi dai criteri è necessario un modello di protezione più generale.

14.4 matrice d'accesso

Il modello generale di protezione qui descritto si può considerare in modo astratto come una matrice, chiamata matrice d'accesso. Le righe della matrice rappresentano i domini, e le colonne gli oggetti. Ciascun elemento della matrice consiste di un insieme di diritti d'accesso. Poiché le colonne definiscono esplicitamente gli oggetti, si possono omettere i nomi degli oggetti dai diritti d'accesso. L'elemento $access(i, j)$ definisce l'insieme di operazioni che un processo in esecuzione nel dominio D_i può richiamare sull'oggetto O_j .

Per spiegare questi concetti si consideri la matrice d'accesso riportata nella Figura 14.3, in cui sono presenti quattro domini e quattro oggetti: tre file (F_1, F_2, F_3) e una stampante. Quando viene eseguito nel dominio D_1 , un processo può leggere i file F_1 ed F_3 . Un processo in esecuzione nel dominio D_4 ha gli stessi privilegi di un processo in esecuzione nel dominio D_1 , ma in più può scrivere anche sui file F_1 e F_3 . Solo un processo in esecuzione nel dominio D_2 può accedere alla stampante.

Lo schema della matrice d'accesso offre un meccanismo che permette di specificare diversi criteri. Il meccanismo consiste nella realizzazione della matrice d'accesso e nella garanzia che le proprietà semantiche sottolineate siano sempre valide. Più specificamente, occorre assicurare che un processo in esecuzione nel dominio D_i possa

oggetto dominio \ F _i	F ₁	F ₂	F ₃	stampante
D ₁	read		read	
D ₂				print
D ₃		read	execute	
D ₄	read write		read write	

Figura 14.3 matrice d'accesso.

accedere solo agli oggetti specificati nella riga i e solo nel modo indicato dagli elementi della matrice d'accesso.

con la matrice d'accesso si possono attuare i criteri riguardanti la protezione. Tali criteri implicano la scelta dei diritti da inserire nell'(i, j)-esimo elemento. Occorre anche stabilire il dominio in cui avviene l'esecuzione di ogni processo. Quest'ultimo criterio è generalmente stabilito dal sistema operativo.

Normalmente sono gli utenti a decidere il contenuto degli elementi della matrice d'accesso. Quando un utente crea un nuovo oggetto O_j, si aggiunge la colonna O_j alla matrice d'accesso con gli elementi di inizializzazione stabiliti dal creatore. L'utente può decidere di inserire alcuni diritti in determinati elementi della colonna j e altri diritti in altri elementi, secondo le necessità.

La matrice d'accesso fornisce un meccanismo adeguato alla definizione e realizzazione di uno stretto controllo sia per l'associazione statica sia per l'associazione dinamica tra processi e domini. Quando un processo passa da un dominio all'altro, si esegue un'operazione (switch) su un oggetto (il dominio). Il passaggio da un dominio all'altro si può controllare inserendo i domini tra gli oggetti della matrice d'accesso. Analogamente, quando si modifica il contenuto della matrice d'accesso, si esegue un'operazione su un oggetto: la matrice d'accesso. Anche in questo caso si possono controllare le modifiche considerando la stessa matrice d'accesso come un oggetto. In effetti, poiché si può modificare singolarmente, ogni elemento della matrice d'accesso si deve considerare come un oggetto da proteggere. A questo punto si devono considerare solo le operazioni possibili su questi nuovi oggetti, domini e matrice d'accesso, e occorre decidere come debbano essere eseguite dai processi.

I processi devono poter passare da un dominio all'altro. Il passaggio dal dominio D_i al dominio D_j è permesso se e solo se l'operazione $\text{switch } \in \text{access}(i, j)$. Quindi, com'è illustrato nella Figura 14.4, un processo in esecuzione nel dominio D₂ può passare al dominio D₃ oppure al dominio D₄. Un processo del dominio D₄ può passare al dominio D₁, e uno del dominio D₁ può passare al dominio D₂.

oggetto dominio \	F ₁	F ₂	F ₃	stampante	D ₁	D ₂	D ₃	D ₄
D ₁	read		read			switch		
D ₂				print			switch	switch
D ₃		read	execute					
D ₄	read write		read write		switch			

Figura 14.4 Matrice d'accesso della Figura 14.3 con domini come oggetti.

Permettere la modifica controllata del contenuto degli elementi della matrice d'accesso richiede altre tre operazioni: copy , owner e control . Esamineremo queste operazioni nel seguito.

La possibilità di copiare un diritto d'accesso da un dominio (o riga) a un altro della matrice d'accesso è indicata da un asterisco (*) apposto sul diritto d'accesso. Il diritto copy permette di copiare il diritto d'accesso solo all'interno della colonna (cioè per l'oggetto) per la quale il diritto stesso è definito. Per esempio, nella Figura 14.5 (a), un processo in esecuzione nel dominio D₂ può copiare l'operazione read in un elemento qualsiasi associato al file F₂. Quindi, la matrice d'accesso della Figura 14.5(a)

oggetto dominio \	F ₁	F ₂	F ₃
D ₁	execute		write*
D ₂	execute	read*	execute
D ₃	execute		

(a)

oggetto dominio \	F ₁	F ₂	F ₃
D ₁	execute		write*
D ₂	execute	read*	execute
D ₃	execute	read	

(b)

Figura 14.5 Matrice d'accesso con diritti copy .

si può modificare nella matrice d'accesso illustrata nella Figura 14.5(b). Questo schema ha due ulteriori varianti.

1. un diritto viene copiato da $\text{access}(i, j)$ ad $\text{access}(k, j)$ e viene successivamente rimosso da $\text{access}(i, j)$; quest'azione è un trasferimento di un diritto piuttosto che una copiatura.
2. La propagazione del diritto copy può essere limitata. ciò significa che quando si copia il diritto R^* da $\text{access}(i, j)$ ad $\text{access}(k, j)$, si crea solo il diritto R e non R^* . un processo in esecuzione nel dominio D_k non può copiare ulteriormente il diritto R .

un sistema può scegliere uno tra questi tre diritti copy , oppure può fornirli tutti e tre identificandoli come diritti separati: copy , transfer e limited copy .

Occorre anche un meccanismo che permetta di aggiungere nuovi diritti e rimuoverne altri; queste operazioni sono controllate dal diritto owner . Se $\text{access}(i, j)$ contiene il diritto owner , un processo in esecuzione nel dominio D_i può aggiungere e rimuovere qualsiasi diritto di qualsiasi elemento della colonna j . per esempio, nella Figura 14.6(a) il dominio D_1 è il proprietario di F_1 e quindi può aggiungere e cancellare qualsiasi diritto valido nella colonna di F_1 . Analogamente, il dominio D_2 è proprietario di F_2 e F_3 , quindi può aggiungere e rimuovere qualsiasi diritto valido che si trovi all'interno di queste due colonne. così, la matrice d'accesso della Figura 14.6(a) si può modificare nella matrice d'accesso illustrata nella Figura 14.6(b).

I diritti copy e owner permettono a un processo di modificare gli elementi di una colonna. Occorre anche un meccanismo per modificare gli elementi di una riga. Il diritto control si può applicare solo a oggetti di dominio. Se $\text{access}(i, j)$ contiene il diritto control , allora un processo in esecuzione nel dominio D_i può rimuovere qualsiasi diritto d'accesso dalla riga j . Si supponga, prendendo come esempio la Figura 14.4, di inserire il diritto control in $\text{access}(D_2, D_4)$. Qundi un processo in esecuzione nel dominio D_2 può modificare il dominio D_4 , com'è illustrato nella Figura 14.7.

I diritti copy e owner forniscono un meccanismo per limitare la propagazione dei diritti d'accesso, però non forniscono strumenti adeguati per impedire la propagazione delle informazioni. Il problema di garantire che nessuna informazione, tenuta inizialmente in un oggetto, possa migrare all'esterno del proprio ambiente d'esecuzione si chiama problema della reclusione (confinement problem). Tale problema è in generale insolubile (si vedano le Note bibliografiche in proposito).

Queste operazioni sui domini e sulla matrice d'accesso non sono di per sé importanti, ma spiegano come il modello della matrice d'accesso consenta la realizzazione e il controllo dei requisiti di protezione dinamica. Nuovi oggetti e nuovi domini si possono creare dinamicamente e inserire nel modello della matrice d'accesso. Tuttavia, in questo paragrafo è spiegato soltanto il meccanismo di base; chi progetta e usa il sistema deve scegliere i criteri riguardanti quali domini, e in che modo, abbiano accesso a determinati oggetti.

oggetto dominio \ F ₁	F ₁	F ₂	F ₃
D ₁	owner execute		write
D ₂		read* owner	read* owner write
D ₃	execute		

(a)

oggetto dominio \ F ₁	F ₁	F ₂	F ₃
D ₁	owner execute		
D ₂		owner read* write*	read* owner write
D ₃		write	write

(b)

Figura 14.6 matrice d'accesso con diritti owner .

14.5 Realizzazione della matrice d'accesso

È necessario implementare efficacemente la matrice d'accesso. Tale matrice generalmente è sparsa, ossia la maggior parte dei suoi elementi è vuota. Tuttavia, a causa del modo in cui si usa la funzione di protezione, le tecniche standard di strutturazione dei dati per la rappresentazione delle matrici sparse non sono particolarmente utili

oggetto dominio \ F ₁	F ₁	F ₂	F ₃	stampante	D ₁	D ₂	D ₃	D ₄
D ₁	read		read			switch		
D ₂				print			switch	switch control
D ₃		read	execute					
D ₄	write		write		switch			

Figura 14.7 matrice d'accesso della Figura 14.4 modificata.

per quest'applicazione. Nel seguito descriveremo e confronteremo vari metodi per implementare la matrice d'accesso.

14.5.1 Tabella globale

La realizzazione più semplice della matrice d'accesso è una tabella globale costituita di un insieme di triple ordinate $\langle \text{dominio}, \text{oggetto}, \text{insieme dei diritti} \rangle$. Ogni volta che si esegue un'operazione M su un oggetto O_j del dominio D_i , si cerca una tripla $\langle D_i, O_j, R_k \rangle$ nella tabella globale, tale che $M \in R_k$. Se si trova questa tripla, l'operazione può continuare, altrimenti si verifica una condizione di eccezione (errore).

Questa tipo di realizzazione ha, però, parecchi svantaggi. Generalmente la tabella è grande e non può essere conservata nella memoria centrale, perciò sono richieste ulteriori operazioni di I/O. Per gestire questa tabella spesso si usano tecniche di memoria virtuale; inoltre è difficile anche trarre vantaggio da speciali raggruppamenti di oggetti o domini. Per esempio, se chiunque può leggere un particolare oggetto, esso deve avere un elemento distinto in ogni dominio.

14.5.2 Liste d'accesso per oggetti

Ogni colonna della matrice d'accesso si può realizzare come una lista d'accesso per un oggetto (Paragrafo 11.6.2). Naturalmente gli elementi vuoti si possono scartare. Per ogni oggetto la lista risultante è composta di coppie ordinate $\langle \text{dominio}, \text{insieme dei diritti} \rangle$ che definiscono tutti i domini il cui insieme di diritti d'accesso per quell'oggetto risulta non vuoto.

Questo metodo si può estendere facilmente definendo una lista più un insieme di default di diritti d'accesso. Quando nel dominio D_i si tenta un'operazione M su un oggetto O_j , si cerca un elemento $\langle D_i, R_k \rangle$, con $M \in R_k$, nella lista d'accesso relativa all'oggetto O_j . Se si trova quest'elemento, l'operazione è permessa; altrimenti, si controlla l'insieme di default. Se M si trova in questo insieme, l'accesso è permesso, altrimenti l'accesso viene negato e si verifica una condizione di eccezione. Per motivi di efficienza, conviene controllare prima l'insieme di default e poi cercare nella lista d'accesso.

14.5.3 Liste delle abilitazioni per domini

Anziché associare le colonne della matrice d'accesso agli oggetti formando liste d'accesso, è possibile associare ogni riga della matrice al suo dominio. La lista delle abilitazioni (capability list) per un dominio è una lista di oggetti insieme con le operazioni ammesse su quegli oggetti. Un oggetto è spesso rappresentato dal suo nome fisico o indirizzo, detto abilitazione (capability). Per eseguire l'operazione M sull'oggetto O_j , il processo esegue l'operazione M , specificando l'abilitazione (puntatore) per l'oggetto O_j come parametro. Il semplice possesso dell'abilitazione indica che l'accesso è permesso.

La lista delle abilitazioni è associata a un dominio, ma non è mai direttamente accessibile a un processo che si trova in esecuzione in quel dominio: è un oggetto protetto, mantenuto dal sistema operativo e al quale gli utenti possono accedere solo indirettamente. La protezione basata sulle abilitazioni si fonda sul presupposto che non è mai permessa la migrazione delle abilitazioni in qualsiasi spazio di indirizzi direttamente accessibile a un processo utente, dove queste potrebbero essere modificate. Se tutte le abilitazioni sono sicure, è sicuro anche l'oggetto da esse protetto contro accessi non autorizzati.

Le abilitazioni sono state originariamente proposte come un tipo di puntatore sicuro, per soddisfare la necessità di protezione delle risorse dovuta alla diffusione dei calcolatori multiprogrammati. L'idea di un puntatore intrinsecamente protetto (dal punto di vista dell'utente di un sistema) fornisce la base per una protezione che si può estendere fino a livello delle applicazioni.

Per fornire una protezione intrinseca occorre distinguere le abilitazioni dagli altri oggetti, e interpretarle attraverso una macchina astratta su cui si eseguono programmi di livello superiore. Generalmente le abilitazioni si distinguono dagli altri dati in uno dei due modi seguenti.

- Ogni oggetto ha un'etichetta (tag) che ne indica il tipo: abilitazione o dati accessibili. Le etichette non devono essere direttamente accessibili ai programmi applicativi. Per imporre tale limite si può ricorrere al supporto dell'hardware o del firmware. Anche se per distinguere tra abilitazioni e altri oggetti è sufficiente un bit, spesso se ne adoperano di più. Questa estensione permette all'hardware di applicare a tutti gli oggetti etichette indicanti i rispettivi tipi. Quindi l'hardware può distinguere interi, numeri in virgola mobile, puntatori, valori booleani, caratteri, istruzioni, abilitazioni e valori non inizializzati grazie alle rispettive etichette.
- Come alternativa, lo spazio d'indirizzi associato a un programma si può dividere in due parti: una contenente i dati e le istruzioni del programma, accessibile al programma; l'altra, contenente la lista delle abilitazioni, accessibile solo al sistema operativo. Lo spazio di memoria segmentato è un utile supporto di questo metodo (paragrafo 8.4).

Sono stati sviluppati parecchi sistemi di protezione basati sulle abilitazioni, brevemente descritti nel paragrafo 14.8. Anche il sistema operativo Mach fa uso di un tipo di protezione basata sulle abilitazioni.

14.5.4 meccanismo chiave-serratura

Lo schema chiave-serratura (lock-key scheme) rappresenta un compromesso tra le liste d'accesso e le liste di abilitazioni. Ogni oggetto ha una lista di sequenze di bit uniche, chiamate serrature (lock); analogamente, ogni dominio ha una lista di sequenze di bit uniche, chiamate chiavi (key). Un processo in esecuzione in un dominio può accedere a un oggetto solo se quel dominio ha una chiave che corrisponde a una delle serrature dell'oggetto.

Come le liste delle abilitazioni, anche la lista delle chiavi per un dominio deve essere gestita dal sistema operativo per conto del dominio. Gli utenti non possono esaminare o modificare direttamente la lista delle chiavi o delle serrature.

14.5.5 Confronto

Come è ovvio, la scelta di una tecnica di implementazione della matrice degli accessi comporta vari trade-off. L'uso di una tabella globale è relativamente semplice; tuttavia, tale tabella può essere piuttosto grande e spesso non può avvantaggiarsi dalla esistenza di gruppi speciali di oggetti o domini. Le liste d'accesso corrispondono direttamente alle necessità degli utenti. Quando un utente crea un oggetto, può specificare quali domini abbiano accesso a quell'oggetto e quali operazioni siano permesse. Tuttavia, poiché le informazioni sui diritti d'accesso per un particolare dominio non sono localizzate, è difficile stabilire l'insieme dei diritti d'accesso per ogni dominio. Inoltre ogni accesso all'oggetto deve essere controllato, il che richiede una ricerca nella lista d'accesso. In un grande sistema con lunghe liste d'accesso, questa ricerca può causare un notevole spreco di tempo.

Le liste delle abilitazioni non corrispondono direttamente alle necessità degli utenti, ma sono utili per localizzare le informazioni per un dato processo. Un processo che tenti un accesso deve presentare la relativa abilitazione, quindi il sistema di protezione deve verificare soltanto che l'abilitazione sia valida. Tuttavia la revoca delle abilitazioni può essere inefficiente; questo problema è trattato nel Paragrafo 14.7.

Il meccanismo chiave-serratura rappresenta un compromesso tra questi due schemi. Il meccanismo può essere efficace e flessibile, a seconda della lunghezza delle chiavi, che possono essere passate liberamente da dominio a dominio. Inoltre i privilegi d'accesso si possono revocare in modo semplice ed efficace modificando alcune chiavi associate all'oggetto; anche questo problema è trattato nel paragrafo 14.7.

La maggior parte dei sistemi adopera una combinazione di liste d'accesso e liste di abilitazioni. Quando un processo tenta per la prima volta di accedere a un oggetto, si fa una ricerca nella lista d'accesso. Se l'accesso viene negato, si verifica una condizione di eccezione, altrimenti si crea un'abilitazione che si associa al processo. I riferimenti successivi si servono dell'abilitazione per dimostrare rapidamente che l'accesso è consentito. Dopo l'ultimo accesso, l'abilitazione viene distrutta. Questa strategia è usata nel sistema MULTRICS e nel sistema CAL.

Come esempio di come opera questa strategia, si consideri un file system in cui a ogni file è associata una lista d'accesso. Quando un processo apre un file, si fa una ricerca nella directory per trovare il file, si controlla il permesso d'accesso e si assegnano i buffer per l'I/O. Tutte queste informazioni si registrano in un nuovo elemento della tabella dei file associata al processo. L'operazione riporta un indice in questa tabella per il file appena aperto, tramite il quale si compiono tutte le operazioni sul file. Quindi l'elemento della tabella dei file punta al file e ai suoi buffer. Quando il file viene chiuso, si cancella l'elemento della tabella dei file. Poiché la tabella dei file è mantenuta dal sistema operativo, gli utenti non possono alterarla accidentalmente, quindi gli utenti possono accedere ai soli file che sono stati aperti. Poiché l'accesso

viene controllato al momento dell'apertura del file, la protezione è assicurata. Nel sistema operativo uNIX si adopera questa strategia.

Il diritto d'accesso si deve ancora controllare per ogni accesso e l'elemento della tabella dei file contiene l'abilitazione per le sole operazioni ammesse. Se si apre un file per la lettura, nell'elemento della tabella dei file viene inserita un'abilitazione d'accesso alla lettura. Se si tenta di scrivere in quel file, il sistema rileva questa violazione della protezione confrontando l'operazione richiesta con l'abilitazione presente nell'elemento della tabella dei file.

14.6 Controllo dell'accesso

Nel paragrafo 11.6.2 abbiamo descritto l'uso dei controlli dell'accesso ai file in un file system. A tutti i file e le directory è assegnato un proprietario, un gruppo o, in alcuni casi, una lista di utenti, e a ciascuna di tali entità sono assegnate informazioni di controllo dell'accesso. u na funzionalità simile può essere realizzata per altri aspetti del sistema; Solaris 10 ne è un buon esempio.

Solaris 10 migliora la protezione offerta dal sistema operativo applicando il principio del minor privilegio tramite il controllo dell'accesso basato sui ruoli (role-based access control, rb Ac). Questa funzionalità si basa sui privilegi. Si dice privilegio il diritto di eseguire una chiamata di sistema o di sfruttare un'opzione di tale chiamata (come l'apertura di un file con accesso in scrittura). I privilegi possono essere assegnati ai processi, limitandoli al minimo indispensabile per svolgere il compito cui sono preposti. Inoltre, privilegi e programmi possono essere assegnati sulla base di ruoli. u n utente può avere un ruolo assegnato o può assumere un ruolo tramite l'uso di una password. In questo modo, un utente può assumere un ruolo che gli attribuisce un certo privilegio; ciò permette all'utente di eseguire un programma per portare a termine un compito specifico, come rappresentato nella Figura 14.8. Questa organizzazione attenua i rischi per la sicurezza del sistema, in particolare quelli legati ai superuser e ai programmi e setuid.

Si noti l'analogia di questa tecnica con la matrice d'accesso, trattata nel paragrafo 14.4. Questa relazione sarà esplorata più compiutamente negli esercizi che concludono il capitolo.

14.7 Revoca dei diritti d'accesso

In un sistema di protezione dinamica talvolta può essere necessario revocare i diritti d'accesso a oggetti condivisi da diversi utenti. A proposito della revoca si possono presentare diverse questioni.

- Immediata o ritardata. Occorre stabilire se la revoca ha effetto immediatamente o con ritardo. Se la revoca è ritardata, occorre stabilire quando avverrà.
- Selettiva o generale. Quando si revoca un diritto d'accesso a un oggetto, occorre stabilire se la revoca vale per tutti gli utenti che hanno un diritto d'accesso a

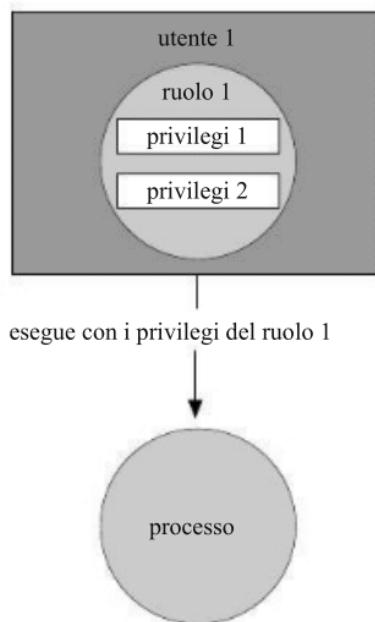


Figura 14.8 Controllo dell’accesso basato sui ruoli in Solaris 10.

quell’oggetto, oppure se si può specificare un gruppo di utenti a cui si debbano revocare i diritti d’accesso.

- Parziale o totale. Occorre stabilire se si può revocare un sottoinsieme di diritti associati a un oggetto, oppure se si devono revocare tutti i diritti d’accesso a quest’oggetto.
- Temporanea o permanente. Occorre stabilire se l’accesso si può revocare in modo permanente, cioè il diritto d’accesso non sarà più disponibile, oppure se può essere nuovamente ottenuto.

disponendo di un sistema basato su liste d’accesso, effettuare una revoca è facile. Si cercano i diritti d’accesso da revocare nella lista d’accesso, e li si cancellano dalla lista. La revoca è immediata e può essere generale o selettiva, totale o parziale, permanente o temporanea.

La revoca delle abilitazioni, invece, è molto più difficile. poiché le abilitazioni sono distribuite su tutto il sistema, occorre prima trovarle e poi revocarle. Tra gli schemi che realizzano la revoca delle abilitazioni ci sono i seguenti.

- Riacquisizione. Le abilitazioni vengono cancellate periodicamente da ogni dominio. Se un processo intende usare un’abilitazione, può scoprire che quell’abilitazione è stata cancellata. Il processo allora può tentare di riacquisirla. Se l’accesso è stato revocato, il processo non è più in grado di riacquisire l’abilitazione.
- Puntatori all’indietro. Insieme a ciascun oggetto si conserva una lista di puntatori a tutte le abilitazioni a esso associate. Quando si richiede una revoca, si possono seguire questi puntatori, modificando le abilitazioni secondo le necessità. Questo

schema era adottato nel sistema Mu LTIC S, ed è abbastanza generale, anche se la sua realizzazione è onerosa.

- Riferimento indiretto. Le abilitazioni non puntano direttamente agli oggetti. Ogni abilitazione punta all'elemento di una tabella globale che a sua volta punta all'oggetto. La revoca si realizza cercando nella tabella globale l'elemento desiderato e cancellandolo. Quando si tenta l'accesso, si riscontra che l'abilitazione punta a un elemento illegale della tabella. Gli elementi della tabella si possono riutilizzare senza difficoltà per altre abilitazioni, poiché sia l'abilitazione sia l'elemento della tabella contengono il nome unico dell'oggetto. L'oggetto dell'abilitazione e dell'elemento nella tabella devono corrispondere. Questo schema, adottato nel sistema c AL, non permette la revoca selettiva.
- Chiavi. Una chiave è una sequenza unica di bit associabile a ogni abilitazione. Tale chiave viene definita al momento della creazione dell'abilitazione e non può essere modificata né esaminata dal processo proprietario dell'abilitazione stessa. Esiste una chiave principale (master key) associata a ogni oggetto, che si può definire o sostituire con l'operazione set-key . Quando si crea un'abilitazione, a questa si associa il valore corrente della chiave principale. Quando l'abilitazione viene esercitata, si confronta la sua chiave con la chiave principale. Se le due chiavi coincidono, l'operazione può continuare, altrimenti si verifica una condizione di eccezione. La revoca sostituisce la chiave principale con un valore nuovo tramite l'operazione set-key , che invalida tutte le abilitazioni precedenti per quest'oggetto. Questo schema non permette la revoca selettiva, poiché a ogni oggetto è associata solo una chiave principale. Se a ogni oggetto si associa una lista di chiavi, si può realizzare la revoca selettiva. Infine, tutte le chiavi si possono raggruppare in una tabella globale di chiavi. Un'abilitazione è valida solo se la sua chiave coincide con una delle chiavi della tabella globale. La revoca si realizza rimuovendo dalla tabella la chiave coincidente. In questo schema una chiave si può associare a più oggetti, e più chiavi si possono associare a ciascun oggetto, offrendo la massima flessibilità. Negli schemi basati sulle chiavi, le operazioni di definizione, inserimento in liste e cancellazione dalle liste delle chiavi stesse non devono essere a disposizione di tutti gli utenti. In particolare, è ragionevole permettere soltanto al proprietario di un oggetto di impostare le chiavi per quell'oggetto. Questa scelta, in ogni modo, riguarda un criterio (policy), che il sistema di protezione può realizzare ma non deve definire.

14.8 Sistemi basati su abilitazioni

In questo paragrafo si esaminano due sistemi di protezione basati su abilitazioni. Questi sistemi sono diversi nel grado di complessità e nel tipo di criteri realizzabili su di essi. Nessuno dei due è molto usato, ma entrambi sono interessanti banchi di prova delle teorie sulla protezione.

14.8.1 Un esempio: Hydra

Il sistema Hydra è un sistema di protezione basato sulle abilitazioni, caratterizzato da una notevole flessibilità. Il sistema fornisce un prefissato insieme di possibili diritti d'accesso tra cui sono presenti le principali forme d'accesso come il diritto per lettura, scrittura o esecuzione di un segmento di memoria. Inoltre il sistema offre agli utenti (del sistema di protezione) gli strumenti necessari per dichiarare altri diritti. I diritti definiti dagli utenti sono interpretati esclusivamente dai programmi dell'utente, ma il sistema fornisce la protezione degli accessi nell'uso di questi diritti e dei diritti definiti dal sistema. Queste funzioni costituiscono un notevole passo avanti nella tecnologia della protezione.

Le operazioni sugli oggetti sono definite in modo procedurale, e le procedure che realizzano tali operazioni sono a loro volta oggetti, accessibili indirettamente attraverso abilitazioni. I nomi delle procedure definite dagli utenti si devono comunicare al sistema di protezione, se deve gestire oggetti del tipo definito dagli utenti. Quando si comunica la definizione di un oggetto al sistema Hydra, i nomi delle operazioni sul tipo diventano diritti ausiliari . Questi diritti ausiliari si possono descrivere in un'abilitazione per un'istanza del tipo. Affinché un processo possa eseguire un'operazione su un oggetto tipizzato, l'abilitazione per quell'oggetto deve contenere tra i diritti ausiliari il nome dell'operazione invocata. Questo limite permette di distinguere i diritti d'accesso istanza per istanza e processo per processo.

Il sistema Hydra offre anche l'amplificazione dei diritti . Questo schema permette di certificare che una procedura è fidata per agire su un parametro formale di un tipo specificato, per conto di qualsiasi processo che abbia un diritto d'esecuzione della procedura. I diritti posseduti da una procedura fidata sono indipendenti dai diritti posseduti dal processo chiamante e possono anche superarli. Tuttavia tale procedura non deve essere considerata universalmente fidata (per esempio, alla procedura non è permesso di agire su altri tipi), e l'affidabilità non deve essere estesa a qualsiasi altra procedura o altro segmento di programma che può essere eseguito da un processo.

L'amplificazione permette alle procedure di accedere alle variabili che rappresentano il tipo di dato astratto. Se, per esempio, un processo possiede un'abilitazione a un oggetto tipizzato A, quest'abilitazione può comprendere un diritto ausiliario a richiamare una generica operazione P, ma non può comprendere nessuno dei cosiddetti diritti kernel, come i diritti per lettura, scrittura o esecuzione, sul segmento che rappresenta A. Tale abilitazione offre a un processo un mezzo per accedere indirettamente, tramite l'operazione P, alla rappresentazione di A, ma solo per scopi specifici.

d'altra parte, quando un processo invoca l'operazione P su un oggetto A, l'abilitazione d'accesso ad A può essere amplificata quando il controllo passa al corpo del codice di P. Quest'amplificazione può essere necessaria per conferire a P il diritto d'accesso al segmento di memoria che rappresenta A, così da realizzare l'operazione che P definisce sul tipo di dato astratto. Si può permettere al corpo del codice di P di leggere o scrivere direttamente nel segmento di A, anche se il processo chiamante non può farlo. Al termine di P, si riporta l'abilitazione per A al suo stato originale non amplificato. Questo è un tipico caso in cui i diritti posseduti da un processo per accedere

a un segmento protetto devono cambiare dinamicamente, secondo il compito da svolgere. La regolazione dinamica dei diritti si esegue per garantire la coerenza delle astrazioni definite dai programmatori. Nel sistema operativo Hydra l'amplificazione dei diritti si può stabilire in modo esplicito nelle dichiarazioni dei tipi di dato astratti.

Quando un utente passa un oggetto come argomento a una procedura, può essere necessario assicurare che la procedura non possa modificare l'oggetto. Questa limitazione si può realizzare facilmente passando un diritto d'accesso senza diritto di modifica (scrittura). Tuttavia, se l'amplificazione è possibile, il diritto di modifica può essere ripristinato, e il requisito di protezione dell'utente può quindi essere aggirato. In generale, naturalmente, un utente può supporre che una procedura esegua correttamente il proprio compito. Tale ipotesi, però, non sempre è corretta, poiché possono verificarsi errori fisici o logici. Il sistema Hydra risolve questo problema limitando le amplificazioni.

Il meccanismo di chiamata di procedura del sistema Hydra è stato progettato come una diretta soluzione al problema dei sottosistemi mutuamente sospetti. Questo problema è definito come segue. Si supponga che per un programma possa essere invocato come servizio da diversi utenti (per esempio, una procedura di ordinamento, un compilatore, un videogioco). Quando gli utenti invocano questo programma di servizio, corrono il rischio che il programma non funzioni bene e possa danneggiare i dati forniti, o trattenere, senza averne l'autorità, qualche diritto d'accesso ai dati, da adoperare in seguito. Analogamente, il programma di servizio può avere alcuni file privati, per esempio file di accounting, cui il programma utente chiamante non deve accedere direttamente. Il sistema Hydra fornisce meccanismi per affrontare questo problema in modo diretto.

Un sottosistema di Hydra è costruito sopra un kernel di protezione e può richiedere la protezione dei propri componenti. Un sottosistema interagisce con il kernel tramite un insieme di primitive stabilite dal kernel, che definiscono i diritti d'accesso alle risorse definite dal sottosistema. I criteri relativi all'uso di queste risorse da parte dei processi utenti possono essere definiti da chi progetta il sottosistema, ma si implementano tramite l'uso della protezione degli accessi standard offerta dal sistema di abilitazioni.

Un programmatore può servirsi direttamente del sistema di protezione, dopo aver studiato le sue caratteristiche. Il sistema Hydra offre un'ampia libreria di procedure definite dal sistema che i programmi utenti possono impiegare. Un utente del sistema Hydra può incorporare esplicitamente le chiamate dirette a queste procedure di sistema nel codice del proprio programma, oppure servirsi di un traduttore di programmi interfacciato ad Hydra.

14.8.2 Un esempio: sistema Cambridge CAP

Un diverso orientamento alla protezione basata sulle abilitazioni è stato seguito nella progettazione del sistema Cambridge CAP. Il sistema di abilitazioni di CAP è più semplice e a prima vista meno potente di quello di Hydra. Tuttavia, con un esame più attento, è possibile capire che anche questo sistema si può usare per offrire una prote-

zione sicura agli oggetti definiti dagli utenti. c Ap ha due tipi di abilitazioni. Il tipo ordinario si chiama abilitazione dati , e si può impiegare per fornire l’accesso agli oggetti, ma gli unici diritti forniti sono quelli ordinari di lettura, scrittura o esecuzione dei singoli segmenti di memoria associati all’oggetto. Le abilitazioni dati sono interpretate dal microcodice della macchina c Ap.

Il secondo tipo è la cosiddetta abilitazione software , che è protetta, ma non interpretata, dal microcodice di c Ap. L’interpretazione spetta a una procedura protetta (cioè privilegiata) che può essere scritta da un programmatore di applicazioni come parte di un sottosistema. A una procedura protetta è associato un particolare tipo di amplificazione di diritti. Quando si esegue il corpo del codice di tale procedura, un processo acquisisce temporaneamente i diritti di lettura o scrittura del contenuto dell’abilitazione software stessa. Questo particolare tipo di amplificazione di diritti corrisponde a una realizzazione delle primitive seal e unseal sulle abilitazioni. Naturalmente questo privilegio rimane soggetto alla verifica del tipo per assicurare che solo le abilitazioni software per uno specifico tipo astratto possano passare a una particolare procedura. Nessun codice gode di totale fiducia, tranne il microcodice della macchina c Ap. Per avere riferimenti in merito si vedano le Note bibliografiche alla fine del capitolo.

L’interpretazione di un’abilitazione software è lasciata esclusivamente al sottosistema, che la esegue per mezzo delle proprie procedure protette. Questo schema permette di realizzare un gran numero di criteri di protezione. benché un programmatore possa definire le proprie procedure protette, la sicurezza del sistema nel suo complesso non può essere compromessa (anche se tali procedure contengono errori). Il sistema di protezione di base non permetterebbe a una procedura protetta non verificata, definita dall’utente, di accedere a qualsiasi segmento di memoria, o abilitazione, che non appartenga all’ambiente di protezione in cui la procedura stessa risiede. La conseguenza più grave dovuta a una procedura protetta non sicura è la violazione della protezione del sottosistema del quale quella procedura è responsabile.

I progettisti del sistema c Ap hanno notato che l’uso delle abilitazioni software permette di fare notevoli economie nella formulazione e nella realizzazione di criteri di protezione adeguati ai requisiti delle risorse astratte. Tuttavia un progettista di sottosistemi che voglia usare questa funzione non può semplicemente studiare un manuale, come nel caso del sistema Hydra, ma deve apprenderne i principi e le tecniche di protezione, poiché il sistema non offre alcuna libreria di procedure.

14.9 Protezione basata sul linguaggio

Il grado di protezione fornito negli attuali sistemi elaborativi è di solito ottenuto attraverso il kernel del sistema operativo, che si occupa di controllare e convalidare ogni tentativo d’accesso a una risorsa protetta. Poiché una completa convalida degli accessi è potenzialmente una fonte di notevole sovraccarico, o si dispone del supporto dell’hardware per ridurre il costo di ogni convalida, o si deve accettare un compromesso rispetto agli obiettivi della protezione. Se la flessibilità di realizzazione dei cri-

teri di protezione è limitata dai meccanismi di supporto disponibili, o se gli ambienti di protezione sono resi più grandi di quanto è necessario per assicurare una maggiore efficienza, soddisfare tutti questi obiettivi è difficile.

Gli scopi della protezione sono stati perfezionati con l'aumentare della complessità dei sistemi operativi e soprattutto con il tentativo di fornire interfacce utente di livello superiore. I progettisti dei sistemi di protezione si sono basati in modo determinante su idee nate nell'ambito dei linguaggi di programmazione e, soprattutto, sui concetti di tipo di dati astratti e di oggetto. Attualmente i sistemi di protezione non riguardano solo l'identità di una risorsa a cui si tenta di accedere, ma anche la natura funzionale di quell'accesso. Nei sistemi di protezione più recenti la azione di controllo sulle funzioni da invocare va oltre un insieme di funzioni definite dal sistema, come gli ordinari metodi per l'accesso ai file, per comprendere anche funzioni definibili dagli utenti.

Anche i criteri concernenti l'uso delle risorse variano secondo l'applicazione e, con il passare del tempo, possono essere soggetti a cambiamenti. Per questi motivi la protezione non può più essere considerata come un esclusivo interesse del progettista di un sistema operativo; dovrebbe essere uno strumento disponibile al progettista di applicazioni per proteggere le risorse di un sottosistema d'applicazione da manomissioni o effetti dovuti a errori.

14.9.1 Controllo realizzato dal compilatore

A questo punto entrano in gioco i linguaggi di programmazione. Specificare il controllo degli accessi desiderato per una risorsa condivisa significa fare un'asserzione dichiarativa su tale risorsa. Questo tipo di dichiarazione si può integrare in un linguaggio di programmazione estendendone la funzione di tipizzazione. Quando insieme alla tipizzazione dei dati si dichiara anche la protezione, i progettisti dei sottosistemi possono specificare i propri requisiti di protezione, come anche la necessità di utilizzare altre risorse del sistema. Tali specificazioni si dovrebbero fornire direttamente nella fase di stesura dei programmi, e nello stesso linguaggio in cui si scrivono i programmi. Questo metodo presenta importanti vantaggi:

1. le necessità di protezione si devono semplicemente dichiarare e non programmare come una sequenza di chiamate di procedure di un sistema operativo;
2. i requisiti di protezione si possono stabilire indipendentemente dalle funzioni fornite da uno specifico sistema operativo;
3. i progettisti dei sottosistemi non devono fornire i meccanismi di controllo dei criteri;
4. la notazione dichiarativa è naturale, perché i privilegi d'accesso sono strettamente connessi al concetto linguistico di tipo di dati.

L'implementazione di un linguaggio di programmazione può fornire diverse tecniche per imporre protezioni, ma ciascuna di loro deve dipendere in qualche misura dalle funzioni di supporto offerte dalla macchina sottostante e dal suo sistema operativo.

Si supponga, per esempio, che un linguaggio sia impiegato per generare codice da eseguire sul sistema Cambridge e Ap. In questo sistema ogni riferimento alla memoria effettuato sull'hardware sottostante avviene indirettamente tramite abilitazioni. Questo limite impedisce a un processo qualsiasi di accedere a una risorsa esterna al proprio ambiente di protezione. Tuttavia un programma può imporre limitazioni arbitrarie su come una risorsa può essere usata durante l'esecuzione di un particolare segmento di codice. Tali limiti si possono realizzare in modo pressoché immediato ricorrendo alle abilitazioni software offerte da e Ap. L'implementazione di un linguaggio di programmazione potrebbe fornire procedure protette standard per interpretare le abilitazioni software che realizzano i criteri di protezione specificabili nel linguaggio stesso. Questo schema permette ai programmatore di specificare i criteri di protezione, senza costringerli a occuparsi dei dettagli riguardanti i relativi meccanismi di supporto.

Anche se un sistema non offre un kernel di protezione potente come quelli di Hydra o e Ap, esistono meccanismi che permettono la realizzazione delle specifiche di protezione presenti in un linguaggio di programmazione. La differenza principale è dovuta al fatto che la sicurezza di questa protezione non è grande quanto quella gestita da un kernel di protezione, poiché il meccanismo si deve basare su un maggior numero di assunzioni sullo stato operativo del sistema. Un compilatore può separare i riferimenti per i quali non è possibile avere violazioni di protezione da quelli per i quali la violazione può avvenire, e trattarli in modi diversi. La sicurezza offerta da questo tipo di protezione si fonda sul presupposto che il codice generato dal compilatore non venga modificato prima o durante la sua esecuzione.

Di seguito sono elencati i vantaggi dei meccanismi basati esclusivamente su un kernel, rispetto a quelli che si hanno con i meccanismi forniti da un compilatore.

- Sicurezza. Il controllo effettuato da un kernel offre un grado di sicurezza del sistema di protezione maggiore di quello offerto dalla generazione, da parte di un compilatore, di codice per il controllo della protezione. In uno schema supportato dal compilatore, la sicurezza è basata sulla correttezza del traduttore, su qualche meccanismo di gestione della memoria che protegge i segmenti dai quali si esegue il codice compilato e, in ultima analisi, sulla sicurezza dei file dai quali si carica il programma. Alcune di queste considerazioni valgono anche per un kernel di protezione supportato esclusivamente dal software, ma in questo caso in misura minore poiché il kernel può risiedere in segmenti fissati di memoria fisica e può essere caricato solo da un file designato. In un sistema di abilitazioni con etichette, in cui tutto il calcolo degli indirizzi è eseguito dall'hardware o da un microprogramma fisso, si può avere una sicurezza ancora maggiore. La protezione basata sull'hardware è anche relativamente immune dalle violazioni della protezione verificabili a causa di malfunzionamenti sia fisici sia logici.
- Flessibilità. La flessibilità di un kernel di protezione ha alcuni limiti per quel che riguarda la realizzazione di criteri di protezione definiti dagli utenti, anche se può fornire al sistema le funzioni necessarie per l'imposizione dei propri criteri. Con un linguaggio di programmazione i criteri di protezione si possono dichiarare, così come si può fornire il controllo richiesto tramite l'implementazione del linguag-

gio. Se un linguaggio non offre una sufficiente flessibilità, si può estendere o sostituire, interferendo con il sistema in servizio meno di quanto non accadrebbe modificando il kernel del sistema operativo.

- **Efficienza.** La maggior efficienza si ha quando la protezione è gestita direttamente dall'hardware (o dal microcodice). Il controllo basato sul linguaggio ha il vantaggio di poter verificare il controllo degli accessi in maniera statica nella fase di compilazione. Inoltre, poiché un compilatore intelligente può adattare il meccanismo di controllo alla specifica necessità, spesso si può evitare l'overhead fisso delle chiamate del kernel.

ripiegando, la specificazione della protezione in un linguaggio di programmazione permette la descrizione ad alto livello di criteri di allocazione e uso di risorse. L'implementazione di un linguaggio di programmazione può fornire gli strumenti per la realizzazione della protezione quando non è disponibile il controllo automatico gestito dall'hardware; inoltre può interpretare le indicazioni di protezione per generare chiamate a qualsiasi sistema di protezione sia fornito dall'hardware e dal sistema operativo.

Un modo per rendere disponibile la protezione ai programmi applicativi prevede l'uso delle abilitazioni software come oggetti di calcolo. Questo concetto è basato sull'idea che alcuni componenti di programmi possano avere il privilegio di creare o esaminare queste abilitazioni. Un programma che crea un'abilitazione può eseguire un'operazione primitiva (`seal`) che sigilla una struttura dati, rendendo il contenuto di quest'ultima inaccessibile a qualsiasi componente di programma che non possieda i privilegi `seal` o `unseal`. Questi componenti potrebbero copiare la struttura dati, o passarne l'indirizzo ad altri componenti del programma, ma non possono ottenere l'accesso al suo contenuto. Tali abilitazioni software si introducono per portare un meccanismo di protezione all'interno del linguaggio di programmazione. L'unico problema che s'incontra seguendo tale metodo riguarda l'uso delle operazioni `seal` e `unseal`; tale uso richiede infatti un orientamento procedurale alla specifica della protezione. Per rendere disponibile l'ambiente di protezione ai programmatore di applicazioni sembra preferibile una notazione non procedurale o dichiarativa.

È necessario un meccanismo dinamico sicuro di controllo degli accessi, per poter distribuire tra i processi utenti le abilitazioni alle risorse del sistema. Per contribuire all'affidabilità generale di un sistema, il meccanismo per il controllo dell'accesso deve essere utilizzabile in modo sicuro, e per essere utile nella pratica deve essere anche ragionevolmente efficiente. Questo requisito ha portato allo sviluppo di un certo numero di costrutti di linguaggio che consentono ai programmatore di dichiarare vari limiti riguardanti l'uso di specifiche risorse (si vedano le Note bibliografiche per gli appropriati riferimenti).

Questi costrutti forniscono meccanismi per tre funzioni:

1. distribuire, in modo sicuro ed efficiente, le abilitazioni tra i processi clienti: in particolare i meccanismi assicurano che un processo utente si servirà di una risorsa solo se gli è stata concessa una specifica abilitazione;

2. specificare il tipo di operazioni che un processo particolare può compiere su una risorsa assegnata (per esempio, a un lettore di file si deve permettere soltanto di leggere i file, mentre a uno scrittore si possono concedere sia lettura sia scrittura): non dovrebbe essere necessario concedere lo stesso insieme di diritti a ogni processo utente e un processo non dovrebbe avere la possibilità di ampliare il proprio insieme di diritti d'accesso, tranne che se abbia ricevuto l'autorizzazione da parte del meccanismo di controllo degli accessi;
3. specificare l'ordine in cui un processo particolare può compiere le diverse operazioni su una risorsa (per esempio, un file deve essere aperto prima di essere letto): deve essere possibile dare a due processi limitazioni diverse sull'ordine in cui possono compiere le operazioni sulla risorsa assegnata.

L'incorporazione dei concetti di protezione nei linguaggi di programmazione, intesa come strumento pratico per la progettazione dei sistemi, è in una fase iniziale. La protezione probabilmente acquisterà un crescente interesse da parte dei progettisti di nuovi sistemi con architetture distribuite, e requisiti sempre più severi sulla sicurezza dei dati; sarà quindi riconosciuta più diffusamente anche l'importanza d'idonee notazioni di linguaggio in cui esprimere i requisiti di protezione.

14.9.2 Protezione nel linguaggio Java

Poiché Java è stato pensato per l'esecuzione in ambiente distribuito, la macchina virtuale Java, o JV M (Java virtual machine), è dotata di molti meccanismi di protezione. I programmi Java sono composti da classi, ognuna delle quali è un insieme di campi di dati e di funzioni (chiamate metodi) che operano su quei campi. La JV M carica una classe come risposta a una richiesta di creazione di istanze (o oggetti) di quella classe. Una tra le caratteristiche più originali e utili del linguaggio Java è la gestione del caricamento dinamico di classi non fidate da una rete e dell'esecuzione all'interno della stessa JV M di classi che si ritengono mutuamente sospette.

Proprio per queste caratteristiche, il problema della protezione è di vitale importanza. Le classi eseguite dalla stessa JV M possono provenire da diverse fonti e possono avere diversi gradi di affidabilità. Quindi, è insufficiente imporre la protezione a livello del processo della JV M. Intuitivamente, il fatto che una richiesta d'apertura di file sia consentita dipenderà generalmente da quale classe ha fatto la richiesta d'apertura. Il sistema operativo non ha queste informazioni.

Le decisioni di protezione sono gestite all'interno della JV M. Quando la JV M carica una classe, la assegna a un dominio di protezione che fornisce i permessi per quella classe. Il dominio di protezione al quale si assegna una classe dipende dall'URL da cui la classe è stata caricata e da eventuali firme digitali sul file della classe (le firme digitali sono trattate nel paragrafo 15.4.1.3). Un file che permette la configurazione dei criteri di protezione determina i permessi che si danno al dominio (e alle sue classi). Per esempio, le classi prelevate da un server fidato si potrebbero mettere in un dominio di protezione che permette a tali classi di accedere ai file

nelle directory iniziali degli utenti, mentre le classi prelevate da un server ritenuto non fidato potrebbero non avere alcun permesso d'accesso ai file.

Per una JV M può essere difficile stabilire quale sia la classe responsabile di una richiesta d'accesso a una risorsa protetta. Gli accessi avvengono spesso in modo indiretto, per mezzo di librerie di sistema o altre classi. Si consideri per esempio una classe cui non sia permesso aprire connessioni di rete. Potrebbe chiamare una libreria di sistema per richiedere il caricamento dei contenuti di un ur L. La JV M deve decidere se aprire a no una connessione di rete per questa richiesta. Ma quale classe si dovrebbe considerare per determinare se si debba concedere o no la connessione, l'applicazione o la libreria di sistema?

L'orientamento seguito nel linguaggio Java è quello di richiedere alla classe di libreria di permettere esplicitamente una connessione di rete. Più in generale, per poter accedere a una risorsa protetta, uno dei metodi nella sequenza delle chiamate che ha portato alla richiesta deve esplicitamente asserire il privilegio di accedere alla risorsa. In questo modo, il metodo si assume la responsabilità della richiesta e, presumibilmente, eseguirà anche tutti i controlli necessari ad assicurare la sicurezza della richiesta stessa. Chiaramente, non tutti i metodi sono abilitati ad asserire privilegi; lo possono fare solo se la classe d'appartenenza è in un dominio di protezione che ha esso stesso il permesso di esercitare quel privilegio.

Questo metodo di realizzazione si chiama ispezione dello stack . Ogni thread nella JV M ha uno stack associato per le sue attuali invocazioni di metodi. Quando un chiamante può non essere fidato, un metodo esegue una richiesta d'accesso all'interno di un blocco doPrivileged() , per accedere direttamente o indirettamente a una risorsa protetta. doPrivileged() è un metodo statico della classe AccessController al quale si passa una classe con un metodo run() da invocare. Quando l'esecuzione entra nel blocco doPrivileged() , si annota l'elemento dello stack per questo metodo per indicare questo fatto, e successivamente si eseguono le istruzioni nel blocco. Quando, in seguito, si richiede l'accesso a una risorsa protetta, o da parte di questo metodo o da parte di un metodo da esso chiamato, s'invoca checkPermissions() per richiedere l'ispezione dello stack, allo scopo di determinare se l'accesso richiesto debba essere concesso. L'ispezione consiste nell'esame degli elementi presenti nello stack del thread chiamante, partendo da quello inserito più recentemente e procedendo verso il meno recente. Se prima si trova un elemento che ha l'annotazione doPrivileged() , checkPermissions() permette immediatamente l'accesso. Se invece si trova prima un elemento per il quale l'accesso non è consentito nell'ambito del dominio di protezione della classe del metodo, checkPermissions() genera un'eccezione AccessControlException . Se l'ispezione esaurisce lo stack senza trovare né un tipo d'elemento né l'altro, allora la concessione dell'accesso dipende dalla implementazione (alcune implementazioni della JV M permettono l'accesso, altre lo negano).

La procedura d'ispezione dello stack è illustrata nella Figura 14.9. In quest'esempio, il metodo gui() di una classe nel dominio di protezione untrusted applet compie due operazioni: prima una get() e poi una open() . La prima corrisponde all'invocazione del metodo get() di una classe nel dominio di protezione u RL loader, che

dominio di protezione:	applet non fidata	caricatore di URL	interconnessione
permesso della socket:	nessuno	*.lucent.com:80, connect	qualsiasi
classe:	gui: ... get(url); open(addr);	get(URL u): ... doPrivileged { open('proxy.lucent.com:80'); } <request u from proxy> ...	open(Addr a): ... checkPermission(a, connect); connect(a); ...

Figura 14.9 Ispezione dello stack.

ha i permessi per aprire sessioni nei siti nel dominio lucent.com , e in particolare per il server proxy.lucent.com per individuare gli ur L. Per questa ragione, l’invocazione di get() dall’applet non fidata andrà a buon fine: la chiamata a checkPermissions() nella libreria di rete troverà l’elemento dello stack relativo al metodo get() che ha eseguito la sua open() in un blocco doPrivileged(). Tuttavia, l’invocazione della open() da parte dell’applet non fidata risulta invece in un’eccezione, poiché la chiamata a checkPermissions() non trova alcuna annotazione doPrivileged() prima di raggiungere l’elemento dello stack relativo al metodo gui() .

Ovviamente, affinché l’ispezione dello stack possa funzionare, un programma non deve poter modificare le annotazioni sul suo stesso elemento dello stack, né compiere altre manipolazioni che interferiscano con l’ispezione dello stack. Questa è una delle differenze più importanti tra il linguaggio Java e molti altri linguaggi (compreso il C++). Un programma scritto in Java non può accedere direttamente alla memoria. Piuttosto, può manipolare solo oggetti per i quali ha un riferimento. I riferimenti non si possono contraffare e le manipolazioni si compiono per mezzo d’interfacce ben definite. La conformità a questi criteri è imposta da un raffinato insieme di controlli della fase di caricamento e della fase d’esecuzione. Ne segue che un oggetto non può manipolare il proprio stack, poiché non può ottenere un riferimento né a essa né ad altri componenti del sistema di protezione.

Più in generale, i controlli del linguaggio Java nella fase di caricamento e nella fase d’esecuzione impongono la sicurezza dei tipi (type safety) delle classi. La sicurezza dei tipi garantisce che le classi non possano trattare gli interi come puntatori, scrivere oltre la fine di un array, accedere alla memoria in modi arbitrari. Un programma può accedere a un oggetto soltanto attraverso i metodi definiti per quell’oggetto dalla sua classe. Su ciò si fonda il sistema di protezione del linguaggio Java, poiché permette a una classe di encapsulare efficacemente i propri dati e metodi e di proteggerli da altre classi caricate nella stessa JV M. Per esempio, una variabile si può definire private , in modo che solo la classe che la contiene possa accedervi, oppure si può definire protected , in modo che possano accedervi solo la classe che la contiene, le sue sottoclassi e le classi dello stesso package. La sicurezza dei tipi garantisce l’imposizione di queste limitazioni.

14.10 Sommario

I sistemi elaborativi contengono molti oggetti, che devono essere protetti contro i possibili abusi. Gli oggetti possono essere hardware (come la memoria, il tempo di cpu o i dispositivi di I/O) o software (come i file, i programmi e i semafori). Un diritto d'accesso è un permesso per eseguire un'operazione su un oggetto. Un dominio è un insieme di diritti d'accesso. I processi vengono eseguiti in domini e possono usare tutti i diritti d'accesso del dominio per accedere agli oggetti e manipolarli. Durante il suo ciclo di vita un processo può essere vincolato a un dominio di protezione o può essergli consentito di passare da un dominio a un altro.

La matrice d'accesso è un modello generale di protezione; fornisce un meccanismo per la protezione senza imporre uno specifico criterio di protezione al sistema o ai suoi utenti. La separazione tra criteri e meccanismi è un'importante caratteristica di progettazione.

La matrice d'accesso è sparsa e normalmente si realizza per mezzo di liste d'accesso associate a ciascun oggetto, oppure per mezzo di liste di abilitazioni associate a ciascun dominio. Si può inserire la protezione dinamica nel modello della matrice d'accesso considerando i domini e la stessa matrice d'accesso come oggetti. La revoca dei diritti d'accesso in un modello di protezione dinamico è di solito più facile da realizzare con lo schema delle liste d'accesso che con le liste di abilitazioni.

I sistemi reali sono molto più limitati e tendono a fornire protezioni solo per i file. UNIX è un caso tipico, poiché per ogni file fornisce le protezioni per lettura, scrittura ed esecuzione, distinte per proprietario, gruppo e per chiunque. Il sistema MULTICS, oltre alla protezione dei file, impiega una struttura dei domini ad anelli. Hydra, il sistema Cambridge CAP e Mach sono sistemi con abilitazioni che estendono la protezione agli oggetti software definiti dagli utenti. Solaris 10 realizza il principio del privilegio minimo attraverso il controllo dell'accesso basato sul ruolo, una forma di matrice d'accesso.

La protezione basata sul linguaggio offre un controllo delle richieste e dei privilegi più selettivo di quello ottenibile con il sistema operativo. Per esempio, una singola JVM può eseguire molti thread, ognuno in un diverso dominio di protezione. La JVM controlla le richieste di risorse attraverso un raffinato meccanismo di ispezione dello stack e attraverso la sicurezza dei tipi offerta dal linguaggio.

Esercizi di ripasso

- 14.1 Quali sono le principali differenze tra liste delle abilitazioni e liste d'accesso?
- 14.2 Un file nel borghes b7000/b6000 Mcp può essere contrassegnato come dato sensibile. Quando un tale file viene cancellato, l'area in cui era memorizzato viene sovrascritta da bit casuali. Per quali scopi un tale schema può essere utile?
- 14.3 In un sistema di protezione ad anello il livello 0 ha l'accesso più ampio agli oggetti, e il livello n (con $n > 0$) ha diritti di accesso più bassi. I diritti di ac-

cesso di un programma a un dato livello nell’anello sono considerati un insieme di abilitazioni. Che relazione c’è tra le abilitazioni a un oggetto per un dominio a livello j e un dominio a livello i (per $j > i$)?

- 14.4 Il sistema rc-4000, come altri sistemi, ha definito un albero dei processi tale che tutti i discendenti di un processo possono ottenere risorse (oggetti) e diritti di accesso solo dai loro antenati. Un discendente non potrà quindi mai fare niente di quello che gli antenati non possono fare. La radice dell’albero è il sistema operativo, che ha la capacità di fare ogni cosa. Assumete che l’insieme dei diritti di accesso sia rappresentato tramite una matrice di accesso A. A(x,y) definisce i diritti di accesso del processo x a un oggetto y. Se x è discendente di z, che relazione c’è tra A(x,y) e A(z,y), per un oggetto arbitrario y?
- 14.5 Quali problemi di protezione possono nascere se viene utilizzata uno stack condiviso per il passaggio di parametri?
- 14.6 Considerate un ambiente elaborativo in cui a ogni processo e a ogni oggetto del sistema sia associato un numero univoco. Supponete che un processo con numero n possa accedere a oggetti con numero m solo se $n > m$. Quale tipo di struttura di protezione avremmo?
- 14.7 Considerare un ambiente elaborativo in cui un processo ottiene il privilegio di accedere a un oggetto soltanto n volte. Suggerite uno schema per implementare una tale politica.
- 14.8 Se tutti i diritti di accesso di un oggetto vengono cancellati, l’oggetto non è più accessibile. A questo punto, lo stesso oggetto dovrebbe essere cancellato, e lo spazio da lui occupato dovrebbe essere restituito al sistema. Suggerite un’efficiente implementazione di questo schema.
- 14.9 Quali difficoltà si incontrano nella protezione di un sistema in cui gli utenti hanno il permesso di eseguire le loro operazioni di I/O?
- 14.10 La lista delle abilitazioni è solitamente mantenuta nello spazio di indirizzi dell’utente. In che modo il sistema assicura che l’utente non possa modificare il contenuto della lista?

Esercizi

- 14.11 Considerate la struttura di protezione ad anelli di MULTICS. Se dovessimo realizzare le chiamate di sistema di un tipico sistema operativo, memorizzandole in un segmento associato all’anello 0, quali sarebbero i valori da memorizzare nel campo concernente l’anello del descrittore del segmento? Che cosa succede durante una chiamata di sistema, quando un processo in esecuzione in un anello superiore invoca una procedura dell’anello 0?

- 14.12 Grazie alla matrice per il controllo dell'accesso si può determinare se un processo sia abilitato a passare, per esempio, dal dominio A al dominio b, e se possa godere dei privilegi d'accesso del dominio b. Valutate se questa soluzione sia da considerarsi equivalente all'inclusione dei privilegi d'accesso del dominio b in quelli del dominio A.
- 14.13 Considerate un sistema in cui i videogiochi possano essere usati dagli studenti solo tra le 22 e le 6, dai membri della facoltà tra le 17 e le 8 e dal personale del centro di calcolo a tutte le ore. Suggerite uno schema per realizzare efficacemente questo criterio.
- 14.14 Date quali caratteristiche dell'hardware di un calcolatore siano necessarie per ottenere un'efficiente manipolazione delle abilitazioni. Date se queste caratteristiche si possano adoperare per la protezione della memoria.
- 14.15 Evidenziate i punti di forza e le vulnerabilità di una matrice d'accesso realizzata tramite le liste d'accesso associate agli oggetti.
- 14.16 Evidenziate i punti di forza e le vulnerabilità di una matrice d'accesso realizzata tramite le abilitazioni associate ai domini.
- 14.17 Spiegate perché i sistemi basati sull'abilitazione, come Hydra, siano più flessibili nell'applicare i parametri di protezione rispetto alla struttura di protezione ad anelli.
- 14.18 Esamineate la necessità dell'amplificazione dei diritti nel sistema Hydra. Quali tratti specifici contraddistinguono questa pratica da quella delle chiamate attraverso più anelli in una struttura ad anelli?
- 14.19 Date che cos'è il principio della necessità di sapere, e perché è importante che un sistema di protezione aderisca a questo principio.
- 14.20 Chiariete quale tra i seguenti sistemi consente ai progettisti di moduli di rendere effettivo il principio della necessità di sapere.
- a. La struttura di protezione ad anelli di MULTICS.
 - b. Le funzionalità del sistema Hydra.
 - c. Lo schema di ispezione dello stack della JV M.
- 14.21 Descrivete in che modo il modello di protezione del linguaggio Java verrebbe meno se si permettesse a un programma scritto in Java di alterare direttamente le annotazioni nel suo elemento dello stack.
- 14.22 In che cosa si assomigliano la funzionalità della matrice d'accesso e il controllo dell'accesso basato sui ruoli? In che cosa differiscono?
- 14.23 In che modo il principio del privilegio minimo aiuta a creare sistemi di protezione?
- 14.24 Come possono persistere, nei sistemi che adottano il principio del privilegio minimo, carenze di protezione tali da causare violazioni alla sicurezza?

Note bibliografiche

Il modello di protezione della matrice d’accesso tra domini e oggetti è sviluppato in [Lampson 1969] e [Lampson 1971]. [Popek 1974] e [Saltzer e Schroeder 1975] presentano eccellenti rassegne sull’argomento della protezione. [Harrison et al. 1976] si serve di una versione formale della matrice di accesso per dimostrare matematicamente alcune proprietà di un sistema di protezione.

Il concetto di abilitazione è un’evoluzione di quello di codeword, di Iliffe e Jodeit, impiegato nel calcolatore della Rice University, [Iliffe e Jodeit 1962]. Il termine capability (abilitazione) è stato introdotto da [Dennis e Horn 1966].

Il sistema Hydra è descritto in [Wulf et al. 1981]. Il sistema cAp è descritto in [Needham e Walker 1977]. [Organick 1972] discute il sistema di protezione ad anelli del sistema MuLTIC S.

La revoca è trattata in [Redell e Fabry 1974], [Cohen e Jefferson 1975] e [Ekanadham e Bernstein 1979]. Il principio di separazione tra criteri e meccanismi di protezione è stato sostenuto dal progettista del sistema Hydra, [Levin et al. 1975]. Il problema della reclusione è stato trattato per la prima volta in [Lampson 1973] ed esaminato successivamente in [Lipner 1975].

L’uso di linguaggi di livello più alto per specificare il controllo degli accessi è stato suggerito per la prima volta da [Morris 1973], che ha proposto l’uso delle operazioni seal e unseal, descritte nel Paragrafo 14.9. [Kieburz e Silberschatz 1978], [Kieburz e Silberschatz 1983] e [McGraw e Andrews 1979] hanno proposto diversi costrutti di linguaggio per la gestione degli schemi dinamici generali per la gestione delle risorse. [Jones e Liskov 1978] considera il problema dell’incorporazione di uno schema statico di controllo degli accessi in un linguaggio di programmazione che prevede i tipi di dati astratti. Un coinvolgimento minimo del sistema operativo nell’attuare la protezione è stato perorato dal Progetto Exokernel ([Ganger et al. 2002], [Kaashoek et al. 1997]). L’estensibilità del codice di sistema per mezzo di meccanismi di protezione basati sul linguaggio è stata discussa in [Bershad et al. 1995b]. Altre tecniche di attuazione della protezione comprendono il cosiddetto sandboxing [Goldberg et al. 1996] e l’isolamento dei malfunzionamenti software [Wahbe et al. 1993]. Le tematiche dell’abbassamento dei costi di gestione legati alla protezione e della concessione dell’accesso a livello utente ai dispositivi di rete sono state trattate in [McCanne e Jacobson 1993], e in [Basu et al. 1995].

Un’analisi più dettagliata dell’ispezione dello stack, che comprende il confronto con altri approcci alla sicurezza dell’ambiente Java, si trova in [Wallach et al. 1997] e [Gong et al. 1997].

Bibliografia

[Basu et al. 1995] A. Basu, V. Buch, W. Vogels e T. von Eicken, “u-Net: A user-Level Network Interface for Parallel and Distributed Computing”, proceedings of the ACM Symposium on Operating Systems Principles, 1995.

- [Bershad et al. 1995] N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers e c. Chambers, "Extensibility, Safety and Performance in the SPIN Operating System", Proceedings of the ACM Symposium on Operating Systems Principles, p. 267–284, 1995.
- [Cohen e Jefferson 1975] E. S. Cohen e d. Jefferson, "Protection in the Hydra Operating System", Proceedings of the ACM Symposium on Operating Systems principles, p. 141–160, 1975.
- [Dennis e Horn 1966] J. Dennis e E. C. V. Horn, "Programming Semantics for Multiprogrammed Computations", Communications of the ACM, Vol. 9, Num. 3, p. 143–155, 1966.
- [Ekanadham e Bernstein 1979] K. Ekanadham e A. J. Bernstein, "Conditional capabilities", IEEE Transactions on Software Engineering, Vol. SE-5, Num. 5, p. 458–464, 1979.
- [Ganger et al. 2002] G. R. Ganger, D. R. Engler, M. F. Kaashoek, H.M. Brinceno, R. Hunt e T. Pinckney, "Fast and Flexible Application-Level Networking on Exokernel Systems", ACM Transactions on Computer Systems, Vol. 20, Num. 1, p. 49–83, 2002.
- [Goldberg et al. 1996] I. Goldberg, D. Wagner, R. Thomas e E. A. Brewer, "A Secure Environment for Untrusted Helper Applications", Proceedings of the 6th USENIX Security Symposium, 1996.
- [Gong et al. 1997] L. Gong, M. Mueller, H. Prafullchandra e r. Schemers, "Going beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1997.
- [Harrison et al. 1976] M. A. Harrison, W. L. Ruzzo e J. D. Ullman, "Protection in Operating Systems", Communications of the ACM, Vol. 19, Num. 8, p. 461–471, 1976.
- [Iliffe e Jodeit 1962] J. K. Iliffe e J. G. Jodeit, "A Dynamic Storage Allocation System", Computer Journal, Vol. 5, Num. 3, p. 200–209, 1962.
- [Jones e Liskov 1978] A. K. Jones e b. H. Liskov, "A Language Extension for Expressing Constraints on Data Access", Communications of the ACM, Vol. 21, Num. 5, p. 358–367, 1978.
- [Kaashoek et al. 1997] M. F. Kaashoek, D. R. Engler, G. R. Ganger, H. M. Brinceno, R. Hunt, D. Mazieres, T. Pinckney, R. Grimm, J. Jannotti e K. Mackenzie, "Application Performance and Flexibility on Exokernel Systems", Proceedings of the ACM Symposium on Operating Systems Principles, p. 52–65, 1997.
- [Kieburz e Silberschatz 1978] R. B. Kieburz e A. Silberschatz, "Capability Managers", IEEE Transactions on Software Engineering, Vol. SE-4, Num. 6, p. 467–477, 1978.
- [Kieburz e Silberschatz 1983] R. B. Kieburz e A. Silberschatz, "Access Right Expressions", ACM Transactions on Programming Languages and Systems, Vol. 5, Num. 1, p. 78–96, 1983.