

## Capitolo 3

# Autenticazione e controllo degli accessi

L'autenticazione è il processo mediante il quale viene verificata l'autenticità di un particolare parametro (solitamente una *identità*, anche detta user identity o "userid") dichiarata da un soggetto. Per fare degli esempi, il riconoscimento di un utente all'accesso ad un sistema è composta da una dichiarazione di identità (il suo identificativo utente) e in generale da un meccanismo per verificare questa dichiarazione, mediante quelle che vengono genericamente definite *credenziali di autenticazione*: questo meccanismo è appunto la procedura di autenticazione. Per "controllo degli accessi" si intendono viceversa tutti quei meccanismi che sono in grado, sulla base dell'identità dell'utente, di garantire (con un termine inglese difficilmente traducibile si parla di *enforcement*) che egli possa compiere solo le azioni per cui è stato opportunamente autorizzato.

Dal momento che i paradigmi classici di sicurezza dei sistemi sono basati sull'identificazione certa degli utenti e sull'attribuzione di privilegi appropriati, la sicurezza nell'autenticazione è un fattore fondamentale che può costituire il punto debole di un sistema informativo. Per questo è estremamente importante conoscere fino in fondo le procedure e i meccanismi di autenticazione e controllo degli accessi.

### 3.1 Procedure di autenticazione

La sicurezza delle procedure di autenticazione ha un duplice obiettivo:

5.3.1.1

1. Deve essere molto difficile falsificare delle credenziali valide, sottrarle, ingannare l'utente per fargliele utilizzare in modo improprio, o eludere in qualche modo il meccanismo di autenticazione stesso.
2. Deve essere per converso molto semplice e poco invasivo per l'utente il processo di verifica ed inserimento delle credenziali, per non ottenere il

risultato sgradevole di trasformare l'utente nel primo "nemico" del sistema di autenticazione.

Esistono fondamentalmente tre modi diversi per autenticare l'identità di un soggetto:

1. mediante un *segreto*, ovvero qualcosa che *solo il soggetto sa*;
2. mediante un *token*, ovvero qualcosa che *solamente il soggetto ha*;
3. mediante sistemi di riconoscimento di tipo *biometrico*, ovvero qualcosa che *soltanto il soggetto è*.

In effetti, non importa affatto che si pensi a soggetti umani o a computer: questi schemi sono gli stessi che possono essere utilizzati nella vita di tutti i giorni per autenticare l'identificazione di una persona: la si riconosce (dal viso, dalla voce); più di rado, le viene chiesto un documento o ci si aspetta che utilizzi una chiave per aprire una serratura (due esempi di token); oppure, molto più raramente, le si chiede di comunicare un qualche tipo di codice segreto.

Paradossalmente, nel mondo informatico questa scala di frequenze è invertita: è molto comune identificarsi usando una *password* (il tipo più comune di segreto); molto più di rado si usa una *smart card* o un *token* di altro genere; ed è estremamente raro, per il momento, trovarsi di fronte a richieste di autenticazione biometrica.

Per incrementare la sicurezza, è possibile utilizzare più di uno di questi schemi di autenticazione contemporaneamente: ad esempio, per utilizzare la propria linea di telefonia cellulare è necessaria una scheda SIM (un *token*) e un codice a 4 cifre (un *segreto*). In questi casi si parla di *two factor authentication*. Ovviamente lo scopo è evitare che il semplice furto del token possa consentire ad un malintenzionato di spacciarsi per il soggetto. Sono molto più rari i casi di *three factor authentication*.

### 3.1.1 Autenticazione mediante un segreto

**5.3.2.1** L'autenticazione mediante un segreto è contemporaneamente il sistema di autenticazione più usato e quello più debole.

È il più usato per molteplici motivi: si tratta di un sistema semplice da implementare, poco costoso, che può essere aggiunto con minima difficoltà a qualsiasi protocollo o applicazione preesistente. Può inoltre essere usato ovunque senza bisogno di hardware particolare (al contrario di sistemi biometrici o basati su smart card o chiavi fisiche di vario genere) e senza costi legati ai dispositivi (singolarmente non elevatissimi, ma su grandi popolazioni di utenti spesso proibitivi).

Tuttavia esistono tutta una serie di problematiche che rendono l'uso dei segreti estremamente insicuro:

- *Possibilità di furto dei segreti*: vi sono molti modi con cui un segreto può essere compromesso, mediante l'intercettazione in rete o sul computer utilizzato per il login, o più banalmente mediante il cosiddetto *shoulder surfing* (ovvero osservando l'utente che digita la password, magari una lettera alla volta, sulla tastiera, passandogli dietro alle spalle con aria noncurante). Se avete il cuore robusto, potete passare sulle scrivanie di un'azienda alla domenica e girare le tastiere: sarete sorpresi da quante di queste recano la password incollata sotto. Se non la trovate sotto la tastiera, è sul post-it che c'è incollato al monitor.
- *Replicabilità*: se viene sottratta una chiave fisica, il possessore presto o tardi se ne accorge; una password può essere conosciuta da infinite persone senza che nessuna sappia delle altre. Pertanto l'utente la cui password è stata violata difficilmente se ne accorge.
- *Mancanza di percezione*: legata all'osservazione precedente, gioca contro anche la psicologia dell'utente. È molto diverso chiedere a una persona di consegnare un oggetto fisico che – percettibilmente – rende sicura una postazione (pensate a un badge di accesso, ad esempio) o chiedergli di rivelarci una password.
- *Possibilità di brute forcing e guessing*: qualsiasi segreto, per quanto robusto, può essere indovinato per tentativi (il cosiddetto attacco di forza bruta); per questo, i segreti devono avere una lunghezza sufficiente e contenere una varietà di caratteri. Spesso tuttavia non è necessario provare “tutte le possibili combinazioni” in quanto gli utenti scelgono come segreti parole che sono contenute in un dizionario, ovvero parole che hanno un significato. Questo riduce drammaticamente il numero di tentativi necessari a violare una password

Per tutti questi motivi, la *gestione accurata dei segreti* è di fondamentale importanza per evitare che il sistema di autenticazione venga oltrepassato. Pertanto è necessario impostare, sia mediante i meccanismi di controllo del sistema operativo, sia mediante un'opportuna educazione e responsabilizzazione degli utenti, dei metodi rivolti:

- a incrementare lunghezza e complessità e a limitare l'uso di parole comuni nelle password, per evitare i rischi di brute forcing e di guessing;
- ad evitare pratiche pericolose, quali la scrittura delle password stesse: si può arrivare all'ispezione delle scrivanie se necessario, ma questa abitudine va eradicata nel modo più assoluto;

- a sensibilizzare gli utenti alla necessità di non trasmettere la propria password ad altri, magari facendo notare che può essere usata per compiere azioni dannose “a loro nome”, di cui saranno ritenuti responsabili;
- a favorire un periodico cambio delle password, che consente di tamponare gli effetti del furto, della replicazione e del bruteforcing delle password;

Alcune best practice che è opportuno ricordare, inserire nelle linee guida aziendali e trasmettere in modo opportuno agli utenti sono ad esempio:

- Non usare mai come password il nome dell’utente stesso, nemmeno scritto al contrario, o con variazioni quali numeri o date in fondo.
- La password non deve mai contenere dati immediatamente collegabili all’utente: la password di qualunque utente identifichi il sig. Mario Rossi non dovrà mai essere né *mario* né *rossi*, né contenere la data di nascita, il nome della moglie, dei figli...
- La password dovrebbe avere una lunghezza minima di 8 caratteri.
- Se una password deve essere scritta per non essere dimenticata, non è una buona password.
- Una password dovrebbe contenere sia lettere maiuscole che lettere minuscole che numeri, in modo da rendere difficili gli attacchi per “forza bruta”.
- La password deve essere cambiata frequentemente e non riutilizzata
- Non si deve usare la stessa password per sistemi differenti, in particolare non per sistemi con diversi gradi di sicurezza o diverse entità amministrative.

Un buon sistema per creare una password ragionevolmente sicura è quello di utilizzare una parola familiare introducendovi alcune sgrammaticature e alterazioni numeriche (“c0mputer”, ad esempio), oppure usando le iniziali di una frase mnemonica (“PmSvN1Cd” in luogo di “Per me si va ne la città dolente”, ad esempio).

### 3.1.2 Autenticazione mediante token

**5.3.3.1** Rispetto all’autenticazione mediante segreti, quella mediante token, ovvero oggetti fisici paragonabili a una “chiave”, presenta innumerevoli ed interessanti vantaggi:

- *Non replicabilità*: se viene sottratto un token, il possessore tipicamente se ne accorge; inoltre, il token non può venire “copiato” facilmente.
- *Percezione fisica della sicurezza*: come dicevamo, la psicologia dell’utente associa molto più facilmente il token al concetto di chiave a cui è abituato

fin dalla nascita. Pertanto è istintivamente molto più restio a consegnare un oggetto a terzi, piuttosto che a condividere una parola chiave.

- *Eliminazione delle problematiche relative alla “scelta della password”* (e, conseguentemente, dei foglietti attaccati al monitor!).
- *Possibilità di brute forcing e guessing*: tipicamente i token usano meccanismi di *challenge/response* a base crittografica, pertanto i tentativi di brute forcing sono sostanzialmente inapplicabili.

Tuttavia, se si opta per una soluzione basata su token bisogna prevedere svariati problemi:

- Si tratta di un sistema più complesso da integrare con meccanismi preesistenti rispetto alle semplici password.
- Ogni token ha un costo non trascurabile, in generale dipendente dal suo livello di sofisticazione.
- Alcuni token necessitano di lettori specifici e/o di porte particolari sui sistemi su cui si deve svolgere l'autenticazione, oppure dell'installazione di software lato client.
- Bisogna comunque decidere appropriate procedure per il rilascio e l'abilitazione dei token, un problema procedurale non indifferente per le strutture medio-grandi.
- I token possono comunque essere sottratti, quindi bisogna prevedere meccanismi per revocare rapidamente token smarriti e possibilmente per impedire che un ladro possa utilizzare un token rubato.

In genere, per ovviare all'ultimo problema, un token prevede un'autenticazione a *due fattori* combinata con una password/PIN. In questo modo, chi venisse a conoscenza del PIN avrebbe comunque bisogno di rubare il token, mentre chi ottiene in maniera fraudolenta il token sarebbe bloccato dalla mancanza del PIN. Tra i tipi più comuni di token ricordiamo:

- le Smart Card, carte elettroniche “intelligenti”, ovvero dotate di un piccolo processore e di un'area di memoria protetta in grado di conservare chiavi crittografiche e di eseguire algoritmi di cifratura e decifratura; in questo modo la chiave non abbandona mai la carta, e usando un meccanismo di tipo “challenge/response” (illustrato più avanti) è possibile autenticare la presenza della tessera dell'utente nel lettore a bordo del PC;
- i token USB, simili alle smart card per concezione ma più facili da utilizzare in quanto non esigono un lettore apposito;
- i generatori di one-time password, piccoli oggetti con un dispositivo che visualizza una password valida per un tempo limitato (ad esempio, 60

secondi): l'utente che si voglia autenticare deve inserire il proprio user-id, la propria password, e infine la password temporanea. Alternativamente si può utilizzare un meccanismo di challenge/response in cui all'utente viene presentato un codice numerico che deve essere digitato sul token; il token elabora quindi una risposta che l'utente copia su PC. Il grande vantaggio di questo metodo è che è facilmente sovrapponibile ad un precedente sistema basato su semplici password, che non richiede hardware o software specifico sul computer client, e che ad esempio può funzionare egregiamente anche nell'ambito web dove i due metodi precedenti non sono funzionali.

Esiste un metodo ibrido tra l'uso di un token e di un segreto, ovvero l'utilizzo di certificati digitali come strumento di autenticazione. Se da un lato un certificato salvato in una smart card è un vero "token", un certificato immagazzinato su un hard disk è meno protetto: può essere copiato, ad esempio, e clonato. Tuttavia, è evidentemente più robusto del "semplice" utilizzo di una password. Chiaramente, nel caso si usi un certificato residente su disco, misura assolutamente imprescindibile è abbinarlo ad una password o passphrase sufficientemente robusta che protegga la chiave privata.



Figura 3.1 Tastiera con riconoscimento utente attraverso SmartCard.

### 3.1.3 Autenticazione biometrica

**5.3.4.1** I processi di autenticazione mediante biometria, ovvero basati sulle caratteristiche fisiche personali di un soggetto, possono sembrare attraenti per molte ragioni:

- Non replicabilità e non sottraibilità: le caratteristiche biometriche non possono essere clonate, né sottratte all'utente.
- Eliminazione delle problematiche relative alla "scelta della password" e alla gestione dei token.
- Semplicità d'uso tipicamente elevata.

Tuttavia, svariati problemi rallentano l'adozione diffusa di tecnologie biometriche:

- Si tratta di un sistema molto complesso da integrare con meccanismi preesistenti.
- Il meccanismo richiede appositi lettori in ogni postazione, che hanno un costo non trascurabile e in genere direttamente proporzionale alla qualità del lettore.
- Alcuni lettori hanno presentato vulnerabilità che consentivano la contraffazione di credenziali mediante metodi più o meno complicati.
- Bisogna comunque attuare appropriate procedure per l'enrollment, ovvero per il primo inserimento dei dati biometrici dell'utente.
- In alcuni casi, possono esserci problemi con gli utenti diversamente abili, che potrebbero sentirsi discriminati da alcuni tipi di autenticazione biometrica.
- Dal momento che la presenza e la cooperazione dell'utente è richiesta per oltrepassare il problema dell'autenticazione, questo può implicare (laddove il sistema protetto abbia un valore sufficiente) possibili rischi per l'incolumità personale dell'utente stesso, trasformando il problema della sicurezza informatica in un problema di sicurezza fisica.

Tra i possibili tipi di autenticazione biometrica si possono ricordare i seguenti.

- *Impronte digitali*, forse il meccanismo più semplice e conosciuto. Il livello di sicurezza è buono ma non ottimale, in quanto è possibile spesso ingannare il lettore con dei "falsi". I lettori più costosi pertanto verificano anche temperatura corporea e presenza di vasi sanguigni.
- *Geometria della mano*: il discorso è molto simile al precedente, anche se i falsi sono più complessi.
- *Viso*: l'identificazione del viso sarebbe molto semplice da usare, anche tramite una comune webcam. Purtroppo ha un'affidabilità molto bassa che la rende scarsamente efficace per l'autenticazione.
- *Iride*: tecnica altamente sicura e molto più difficile da falsificare. L'iride è un piccolo muscolo anulare dell'occhio: la sua pigmentazione e la disposi-

zione delle fibre del muscolo sono caratteristiche uniche per ogni individuo (l'iride ha 266 caratteristiche uniche, l'impronta digitale ne ha 90). Persino le iridi di gemelli monozigoti sono del tutto diverse l'una dall'altra. Il riconoscimento viene effettuato mediante una speciale telecamera che usa un fascio di luce infrarossa a bassa intensità. Persone non vedenti o che fanno uso di lenti a contatto non hanno difficoltà di sorta.

- *Retina*: acquisisce un'immagine interna del corpo della retina dell'occhio. Ogni occhio ha infatti un proprio sistema di vasi sanguigni. Se negli anni '70 la procedura di scansione e i dispositivi erano complessi e anche alquanto invasivi e pericolosi, oggi la scansione della retina non è differente da quella dell'iride.
- *Voce*: il timbro vocale è una caratteristica biometrica identificativa con gli stessi pregi e gli stessi difetti dell'acquisizione dell'immagine del viso (semplicità e scarsa invasività, ma scarsa affidabilità, in quanto il tono di voce è estremamente variabile).



**Figura 3.2** Dispositivo di riconoscimento dell'impronta digitale.

Anche i sistemi di autenticazione biometrica si prestano all'utilizzo in schemi a due fattori. Ad esempio è comune lo schema in cui le "feature" biometriche vengono salvate sul chip di una smart card, e solo la combinazione dell'uso della smart card e della presenza dell'utente può attivare la procedura di autenticazione. Questo schema ha un ulteriore vantaggio in quanto elimina la necessità di un database centralizzato contenente le feature biometriche, che rappresenta un rischio.



Figura 3.3 Riconoscimento dell'impronta digitale combinato con utilizzo di SmartCard.

## 3.2 Sistemi di autenticazione di rete

### 3.2.1 Autenticazione a livello di rete e di host

Sistemi di autenticazione di rete e locali hanno requisiti molto differenti, e i meccanismi che su un sistema locale potrebbero essere adeguati possono rivelarsi inutilizzabili su una LAN, o peggio ancora su Internet.

5.3.5.1

Innanzitutto, il passaggio di credenziali via rete deve essere opportunamente protetto, in genere mediante tecniche crittografiche, contro la possibilità di cattura, di riutilizzo, o di ottenimento a partire delle credenziali del segreto di accesso. Ad esempio, bisogna che siano protette dal rischio di sniffing.

A tal fine sarebbe meglio se le credenziali passate via rete fossero one-time, ovvero valide per una sola volta, oppure di tipo challenge-response. In questo caso il server invia al client un problema (challenge), la cui risposta può essere generata sulla base del segreto o della credenziale che il client ha. Ad esempio, se si usasse per l'autenticazione una smart card con una chiave privata di cifratura, il server potrebbe inviare un piccolo testo e aspettarsi che il client lo cifri con la sua chiave privata, verificandolo quindi con la chiave pubblica. Queste tecniche servono per evitare il riutilizzo delle credenziali mediante attacchi di replay, in cui l'aggressore si limita a ripetere lo scambio di messaggi avvenuto in passato tra client e server.

Inoltre, il paradigma di autenticazione deve gestire l'identità degli utenti e le loro autorizzazioni attraverso sistemi operativi e applicativi multipli e non omogenei, possibilmente consentendo il "single sign on" (illustrato più avanti).

Sia nel caso di autenticazioni a livello di rete, sia a livello di host, se viene usato uno schema basato su un segreto condiviso (password) deve essere posta particolare attenzione a non conservare tale segreto in chiaro su server o su client, e a garantire che sia difficile effettuare un attacco di brute forcing sull'elenco delle password.

In passato i sistemi Unix utilizzavano password cifrate attraverso l'algoritmo *crypt* e registrate in un file leggibile da ogni utente riconosciuto dal sistema (*/etc/passwd*). La loro lunghezza massima era limitata ad 8 caratteri e il sistema presentava numerose debolezze, rendendo poco difficoltosa l'operazione di decifrare le password attraverso programmi di cracking. Attualmente, tutte le moderne distribuzioni di Linux e gran parte dei sistemi UNIX usano le *shadow password*. Con questo termine ci si riferisce al fatto che le password vengono registrate sotto forma di *hash* MD5, in un file leggibile unicamente dall'utente root (*/etc/shadow*). Una password MD5 può superare il tradizionale limite di lunghezza ad 8 caratteri tipico dei vecchi sistemi Unix. Inoltre, grazie all'utilizzo di un valore denominato "salt" che viene combinato con la password per generare l'hash, e grazie alla robustezza e non invertibilità della funzione MD5, un attacco di bruteforcing è molto più complesso. Durante la fase di accesso al sistema viene generato l'hash della password digitata, e viene confrontato con la versione mantenuta nel file che conserva le password. In caso di confronto positivo l'accesso viene consentito.

### 3.2.2 Protocolli per l'autenticazione di rete

**5.3.5.2** Per parlare dei principali protocolli per l'autenticazione di rete, dobbiamo richiamare alcuni semplici concetti relativi al modello di rete di Internet. La pila di protocolli del TCP/IP è fatta per appoggiarsi su uno strato base chiamato "data link layer". Al giorno d'oggi è normale considerare Ethernet, il protocollo per le reti locali, come interfaccia base per TCP/IP, ma sono necessari anche dei protocolli per interfacciare IP con collegamenti di tipo seriale, ad esempio nella comunicazione via modem o via linee ADSL o ATM.

Un primo esempio fu il protocollo **SLIP** (*Serial Line Internet Protocol*), molto semplice ma privo di feature quali controllo di connessione, autenticazione e altri servizi simili. PPP è stato sviluppato per consentire una piena funzionalità di rete su un collegamento seriale, e sebbene sia stato sviluppato per IP supporta molti altri tipi di protocolli di livello superiore. Pubblicato come standard nell'RFC 1171 nel 1990, PPP si è molto evoluto.

PPP è un protocollo orientato alla connessione che realizza un collegamento di layer 2 sopra una varietà di connessioni fisiche: sincrone o asincrone, full-duplex o half-duplex. Come dice il nome, ha una architettura point-to-point tra esattamente 2 dispositivi su un collegamento tale per cui le frame vengono ricevute nello stesso ordine con cui sono inviate. PPP incapsula i messaggi

di livello superiore (ad esempio, i datagrammi IP) dentro frame basati su un protocollo chiamato **HDLC** (*High-Level Data Link Control*). Il protocollo di incapsulamento si chiama NCP, e ne esistono tipi diversi a seconda del protocollo di alto livello (ad esempio, per incapsulare IP dentro HDLC si usa **IPCP**, *IP Control Protocol*, che è un tipo particolare di NCP).

Tuttavia la porzione di PPP che riguarda l'autenticazione è **LCP** (*Link Control Protocol*), il protocollo che gestisce l'attivazione e la terminazione delle connessioni. All'interno di LCP è prevista la possibilità per i dispositivi di negoziare l'uso di un protocollo opzionale di autenticazione. Se entrambi i dispositivi concordano sul protocollo (o se entrambi concordano di non usarlo: si pensi al caso di due PC connessi in un laboratorio con un cavo seriale) lo scambio di messaggi del protocollo di autenticazione avviene prima che il link sia attivato.

I due protocolli di autenticazione base definiti da PPP sono *Password Authentication Protocol* (**PAP**) e *Challenge Handshake Authentication Protocol* (**CHAP**).

*Password Authentication Protocol* (**PAP**) è molto semplice: il dispositivo che si sta connettendo (e che verrà indicato, per semplicità, come *client*) invia all'altro dispositivo (server) un messaggio Authenticate-Request che contiene il suo ID e una password. Il server autentica le credenziali e se decide che sono corrette risponde con un Authenticate-Ack, altrimenti con un Authenticate-Nak.

Sono evidenti le problematiche di sicurezza di un protocollo di questo tipo. Innanzitutto le credenziali sono trasmesse in chiaro. Questo è un grosso problema per un protocollo di autenticazione, perché chiunque possa sorvegliare il traffico può intercettarle e riutilizzarle in futuro. In secondo luogo, solo il client si autentica: cosa succede se il server non è quello che ci si attende?

Un primo miglioramento viene da CHAP, che non trasmette in chiaro la password del client, ma utilizza un metodo a challenge/response. Il server in questo caso usa un three-way handshake (si ritrova la stessa procedura che viene utilizzata nella Shared Key Authentication delle reti wireless): invia a chi si sta connettendo un pacchetto Challenge, contenente un testo random (casuale). Il client unisce la password e il Challenge, genera un *hash MD5* del tutto, e lo reinvia al server in un pacchetto Response. A questo punto il server esegue anche lui la stessa operazione e verifica che i risultati coincidano. Se coincidono, viene inviato un messaggio Success, altrimenti viene inviato un Failure.

Questo schema è robusto contro l'intercettazione: la password non passa mai in rete, e il Challenge cambia ad ogni connessione rendendo la Response inutilizzabile in un attacco di replay. Tuttavia lo schema non consente la mutua autenticazione (continuiamo a non sapere chi è il server) ed è vulnerabile ad attacchi man-in-the-middle.

Microsoft ha creato la sua variante non-standard di CHAP, denominata MS-CHAP, che comprende alcune estensioni per generare chiavi di cifratura e proteggere la connessione. Purtroppo questo protocollo si è rivelato non ro-

busto. Infatti, nella sua versione originale (quella utilizzata da Windows NT) il protocollo utilizza per default un metodo di hashing delle password compatibile con il protocollo LAN Manager, molto più debole di quello standard di NT. Per la precisione, la funzione di hashing di LAN Manager non distingue tra lettere maiuscole e minuscole e suddivide le password superiori a 7 caratteri in due parti da 7 byte, di cui viene fatto separatamente l'hash. Questo rende tutte le password meno resistenti ad attacchi di brute-forcing off-line.

Anche il meccanismo Challenge-Response di MS-CHAP è debole in quanto utilizza i 16 byte di hash della password come chiave aggiungendovi 5 byte vuoti ( $16+5 = 21$ ), e utilizzandoli a gruppi di 7 come chiavi per l'algoritmo DES (7 byte = 56 bit). Cifra così 3 volte il challenge di 8 byte inviato dal server, creando una risposta da 24 byte. Sfruttando il fatto che l'ultimo blocco è stato cifrato con 5 byte nulli e solo 2 byte di chiave, è possibile creare un attacco di bruteforcing molto semplificato e rapido. Per questo è stata poi creata la nuova versione MS-CHAP v2, che oltre a correggere questi problemi consente l'autenticazione mutua, e la generazione di chiavi di cifratura per la comunicazione.

Va anche detto che si sta affermando **EAP**, *Extensible Authentication Protocol*, un'estensione di PPP sviluppata per fornire un meccanismo standard per estendere il protocollo supportando nuovi schemi di autenticazione. Per esempio in EAP è previsto il supporto di token, one-time password, autenticazione usando PKI, certificati e smart-card, e simili.

Gli stessi protocolli sono poi stati utilizzati anche per altre applicazioni, ad esempio la creazione di VPN attraverso il protocollo **PPTP** (*Point to Point Tunneling Protocol*), e nel caso di EAP, per l'autenticazione di utenti anche su reti LAN e WLAN (protocollo 802.1x).

### 3.2.3 Autenticazione per applicazioni distribuite

**5.3.5.3** Le applicazioni distribuite sono un paradigma computazionale non molto in voga, ma che sta subendo un deciso incremento grazie all'introduzione della tecnologia dei cosiddetti web services.

Il concetto di **RPC**, *Remote Procedure Call*, nasce sui sistemi UNIX, per indicare un meccanismo mediante il quale un processo eseguito su un host può chiedere l'esecuzione di una procedura da parte di un processo eseguito su un altro host. Nell'ottica di RPC i servizi di rete sono una collezione di uno o più programmi remoti, ciascuno dei quali implementa una o più procedure remote le quali, assieme ai rispettivi parametri e risultati sono documentate nelle specifiche del protocollo.

Nel seguito si fa riferimento al sistema RPC in ambiente SUN, che è sicuramente tra i più famosi e standardizzati.

Il meccanismo di base prevede che il client invii un messaggio di richiesta di servizio (**call message**) al server e che attenda, bloccandosi, un messaggio

di risposta (**reply** message). Nel primo messaggio sono compresi i parametri attuali della procedura mentre, nel secondo, sono inclusi i risultati della procedura che consentono al processo cliente di riprendere l'esecuzione. Dalla parte del server, il programma che esegue il servizio è momentaneamente sospeso in attesa che arrivi una nuova richiesta. Quando questa arriva il server estrae i parametri della procedura, ne esegue il corpo e quando ha terminato, ottenendo i risultati richiesti, li invia come messaggio di risposta e attende la prossima richiesta.

Sebbene questo modello prevede che solamente uno dei due processi sia attivo ad ogni istante, il protocollo RPC non introduce alcuna restrizione sul tipo di concorrenza implementata. Ad esempio, si può realizzare un protocollo RPC con chiamate asincrone in modo da permettere che cliente possa eseguire dell'altro codice mentre è in attesa della risposta dal server. Un'altra possibilità è data da un server che genera un nuovo processo, clonato da se stesso, per elaborare la richiesta in corso in modo da permettere al server originale di ricevere altre richieste.

Dal momento che queste chiamate attraversano reti potenzialmente insicure, sono evidenti i pericoli di un approccio del genere. L'autenticazione su RPC è dunque parte del protocollo. Viceversa, l'affidabilità delle comunicazioni viene lasciata ai protocolli sottostanti, oppure all'applicazione server che deve controllare ad esempio di non stare rispondendo a un duplicato della transazione precedente.

L'autenticazione supportata è mutua. Il messaggio del client contiene due campi di autenticazione, *credenziale* e *verificatore*, mentre il messaggio di risposta ha un unico campo di autenticazione, il *verificatore di risposta*. Ciascun campo è costituito a sua volta da un record a due campi, *flavor* e *body*. *Flavor* può essere impostato a vari valori (NULL, UNIX, SHORT o DES), e *body* contiene invece fino a 400 byte di informazioni di autenticazione.

Nel caso UNIX, le informazioni di autenticazione contengono un identificativo di transazione, il nome della workstation client, UID e GID del client al momento. Il verificatore che accompagna la credenziale deve essere NULL. Il server, dopo aver convalidato le credenziali, può rispondere con NULL o con SHORT, nel secondo caso sta fornendo al client un codice che gli consentirà di riautenticarsi in seguito.

Il problema di questa procedura è evidente: non esiste alcun controllo sulle credenziali utente dichiarate nella chiamata RPC. Per questo è preferibile lo schema di autenticazione DES, che richiede l'esecuzione del demone di rete *keyserv* sia sulla macchina client che sulla macchina server. L'utente deve essere dotato di una coppia di chiavi pubblica e privata, di cui la pubblica è disponibile sul server di amministrazione della rete, mentre la privata viene decriptata sulla workstation grazie a una password locale inserita nel comando *keylogin*. L'autenticazione DES risolve anche un altro problema, ovvero il fatto che lo UID di un utente viene mappato univocamente solo su una singola work-

station, o su un gruppo di workstation che condividano gli stessi database di utenti; inoltre sarebbe limitato ai sistemi UNIX. L'autenticazione DES usa invece un nome univoco per identificare l'utente, il cui contenuto è indipendente dal singolo sistema operativo sorgente.

L'autenticazione DES contiene un campo *verificatore* che consente l'autenticazione mutua di client e server. Si tratta di un timestamp cifrato dal client e decifrato dal server, che consente di verificare l'autenticità del client e la validità della transazione (messaggi con un timestamp troppo "vecchio" vengono rifiutati automaticamente).

DES è un algoritmo simmetrico, quindi con una sola chiave segreta: questa chiave viene generata all'inizio della conversazione usando il sistema di scambio chiavi di Diffie-Hellman a 192 bit. Anche nel caso di DES il server può attribuire al client un "nickname" con cui riautenticarsi più rapidamente fino alla scadenza del timestamp.

### 3.2.4 Single Sign on e Kerberos

#### 5.3.5.4

Se un utente opera su una rete complessa ed eterogenea, è esperienza comune la necessità di avere login e password molteplici per diversi sistemi, e anche la necessità continua di riautenticarsi su differenti servizi. Per "Single Sign-on" si intende un'architettura che eviti questo problema, consentendo all'utente di autenticarsi un'unica volta, e in seguito di accedere senza soluzione di continuità a tutti i servizi della rete.

#### 5.3.5.5

Il più famoso e utilizzato protocollo di questo tipo è senza dubbio Kerberos. Nato nel 1983 da una idea originale del MIT sviluppata in collaborazione con Digital Equipment Corporation e IBM, Kerberos ottiene il suo nome dal mitologico cane a tre teste posto a guardia degli inferi nella mitologia greca, a simboleggiare le tre funzioni base del protocollo: autenticazione, autorizzazione ed accounting (AAA). Kerberos è ormai giunto alla sua quinta versione, ed è standardizzato nella RFC 1510.

Kerberos usa un sistema di autenticazione di tipo "trusted third party" in cui i soggetti (utenti e servizi), denominati "principal", si autenticano reciprocamente tramite un server centrale denominato **KDC** (*Key Distribution Center*). Un *principal* è definito come "un'entità alla quale un autenticatore Kerberos può assegnare dei ticket". Ogni principal è identificato da un nome univoco generalmente costituito da tre sezioni, secondo la seguente sintassi: *primary/instance@realm*.

Si veda nel dettaglio il significato di ogni sezione del nome.

- **primary** è la parte che identifica "più in dettaglio" il singolo principal. Per esempio, nel caso di un utente il "primary" può essere il suo username, per una macchina il suo nome, eccetera.

- **instance** è un'informazione che completa *primary*, e che può eventualmente essere omessa. Ad esempio per un utente tale informazione può non esistere (`stefano@KERBEROS.ESEMPIO.ORG`) oppure essere utilizzata per qualificarne le mansioni ed il ruolo (`stefano/autore@KERBEROS.ESEMPIO.ORG`). Attenzione tuttavia: i due *principal* dell'esempio (`stefano@KERBEROS.ESEMPIO.ORG` piuttosto che `stefano/autore@KERBEROS.ESEMPIO.ORG`) sono considerati diversi a tutti gli effetti, quindi dotati di permessi e password separate ed indipendenti. Non sono *alias* l'uno dell'altro: "instance" è un qualificatore a tutti gli effetti. Per una macchina "instance" può rappresentare il nome di dominio completo al fine di distinguere macchine diverse in domini Kerberos eterogenei (ad esempio se sono presenti due macchine nominate "alpha", la prima sulla rete beta e la seconda sulla rete gamma, si può utilizzare `alpha/alpha.beta.esempio.org@KERBEROS.ESEMPIO.ORG` per la prima, e `alpha/alpha.gamma.esempio.org@KERBEROS.ESEMPIO.ORG` per la seconda).
- **realm** indica l'ambiente (rete, o dominio) gestito dal KDC preso in considerazione. Una prassi è usare il nome del dominio in lettere maiuscole.

Il funzionamento di Kerberos è il seguente. Uno dei *principal* (quindi un utente, o un servizio) invia una richiesta di autenticazione al KDC. Il KDC crea un ticket denominato **TGT** (*Ticket Granting Ticket*) e lo invia al *principal* cifrato con la chiave segreta condivisa tra *principal* e KDC. Evidentemente, solo il *principal* è in grado di decifrarlo ed utilizzarlo. Da questo momento in poi, se il *principal* desidera autenticarsi ad un altro *principal* (che può essere indicato come *server*), invia il TGT al KDC, che risponde con delle credenziali valide per l'autenticazione a quel *server*, cifrate con la chiave del richiedente. Le credenziali consistono in un ticket e una chiave di cifratura temporanea (detta anche chiave di sessione).

Il *principal* richiedente a questo punto trasmette un ticket con la sua identità e una parte detta "authenticator" al *server*. L'authenticator include un timestamp ed è cifrato con la chiave di sessione. In questo modo il client prova al *server* che il ticket è stato richiesto di recente e che il ticket è stato generato da un *principal* che conosce la chiave di sessione. Il *server* deve solo ottenere dal KDC la chiave di sessione per quel ticket e verificarla. In questo modo il client e il *server* condividono la stessa chiave di sessione, che viene utilizzata per autenticare il client e per cifrare la comunicazione successiva o per scambiare altre chiavi di sessione. Ogni messaggio cifrato è contenuto in un involucro (ASN.1) che, oltre a contenere il messaggio cifrato, indica l'algoritmo di cifratura utilizzato e la versione della chiave. L'autenticazione può essere eseguita anche in modalità two-way, in modo che anche il *principal-server* si autentichi al *principal-client*.

L'intera procedura viene gestita in modo trasparente agli utenti. Strutturalmente, il KDC è diviso in due servizi indipendenti: *Authentication Server*

(AS), che gestisce il database delle chiavi dei principal e che fornisce il TGT, e *Ticket Granting Server* (TGS), che verifica di volta in volta i TGT rispondendo con dei ticket di sessione.

Kerberos può essere utilizzato anche per autenticazioni tra ambienti (realm) diversi. Ogni realm ha un proprio KDC che autentica i propri principal. Il nome di ogni principal comprende anche quello del realm a cui appartiene. Per far sì che un principal di un realm si possa autenticare con un principal di un altro realm, si devono stabilire delle chiavi inter-realm relative ai KDC dei realm interessati. Nel KDC del realm A viene memorizzata la chiave inter-realm del KDC del realm B e viceversa, in maniera tale che il KDC del realm A divenga a sua volta un principal del realm B (e viceversa). A questo punto un principal del realm A può richiedere l'autenticazione verso un principal del realm B: il KDC del realm A fornisce al principal che desidera autenticarsi un TGT valido per il realm B (cifrato con la chiave inter-realm in maniera tale che il KDC del realm B possa decifrarlo).

### 3.3 Sistemi di controllo d'accesso

#### 3.3.1 Teoria del controllo d'accesso

**5.3.6.1** Esistono due fondamentali paradigmi per la gestione dei controlli d'accesso alle risorse:

- Modello **DAC** (*Discretionary Access Control*): ogni oggetto del sistema appartiene ad un utente (owner), e su di esso risultano possibili soltanto determinate operazioni a seconda dei permessi che sono stati attribuiti a tale oggetto. L'owner di ogni oggetto amministra i permessi per gli altri utenti. Esiste poi tipicamente un super utente (amministratore) che è in grado di modificare i permessi e le ownership dei file e di eseguire qualsiasi altra operazione senza sottostare a vincoli.
- Modello **MAC** (*Mandatory Access Control*): in questo caso non esiste la figura dell'owner. Ogni dato o risorsa ha un suo livello di classificazione di sicurezza, solitamente definito mediante una struttura composta da un livello d'accesso (normale-segreto-segretissimo, ad esempio), e da etichette (finanze-ricerca-commerciale, ad esempio). Ogni utente del sistema può accedere a determinati livelli: per cui, nell'esempio, un utente che ha accesso a "Segreto.Commerciale" non potrà accedere a "Segretissimo.Commerciale" o a "Normale.Ricerca", ma potrà accedere a "Normale.Commerciale", ad esempio. Una figura di amministrazione (il Security Manager) assegna la classificazione agli oggetti, e i permessi agli utenti.

I meccanismi di gestione degli accessi dei moderni sistemi operativi sono in gran parte riconducibili al modello DAC. I sistemi UNIX e tutti i sistemi Windows sono chiaramente orientati a questo. Il modello DAC è gestibile e semplice, tuttavia mostra numerosi problemi: il primo è che la gestione dell'architettura è totalmente demandata agli utenti. Se questo da un lato semplifica il processo amministrativo, dall'altro crea un guazzabuglio di permessi non omogenei che un aggressore potrebbe trovare il modo di sfruttare. Inoltre, l'esistenza di una sola classe di super-user implica che alcune applicazioni, dovendo eseguire operazioni particolari, debbano avere i permessi di quel super-user, e quindi in caso di compromissione diano all'attaccante un accesso altamente privilegiato al sistema.

I sistemi MAC sono implicitamente più sicuri, ma molto meno intuitivi e molto più complessi da amministrare. Esistono tuttavia delle implementazioni di sistemi MAC nel mondo UNIX, a partire da SELinux (Security Enhanced Linux) e TrustedBSD, ma anche TrustedSolaris e altri. Vengono tipicamente usati in ambito militare. Un formalismo molto usato per i sistemi MAC è quello denominato di Bell-LaPadula, in cui esistono (come nel nostro esempio di cui sopra) livelli di sicurezza e label, e vengono forzate due proprietà molto semplici:

- No Read Up (se si può accedere a “Segreto” non si può leggere “Segretissimo”, e questo è intuitivo).
- No Write Down (se si può accedere a “Segreto” non si può scrivere a livello “Normale”, altrimenti si potrebbero declassificare inavvertitamente delle informazioni).

Esiste un ulteriore modello, chiamato **RBAC**, *Role-Based Access Control*, in cui i privilegi non vengono garantiti ai singoli utenti, ma a dei “ruoli”, corrispondenti in genere alle funzioni aziendali rappresentate nel sistema informativo.

### 3.3.2 Access Control List e Capabilities

Si è già più volte detto, in varie combinazioni, che il controllo d'accessi stabilisce una matrice di permessi tra utenti opportunamente autenticati e risorse del sistema. A seconda di come si vuole descrivere questa matrice, si utilizza l'approccio delle Capability o delle Access Control List.

In particolare, le capability sono equivalenti a una lettura per righe della matrice, in cui sostanzialmente a ogni utente si associano le cose che può fare su qualche risorsa del sistema. Viceversa, in una access control list ad ogni risorsa sono associati gli elenchi degli utenti che possono accedervi, e i permessi con cui possono accedere. La protezione dei file in Windows e UNIX segue i

paradigmi delle ACL, tuttavia esistono delle estensioni (POSIX Capabilities) che consentono di utilizzare questo secondo paradigma nei sistemi Linux-U<sub>n</sub>ix.

Tipicamente nelle implementazioni ogni utente acquisisce le proprie capability all'atto del login, e non vengono in seguito riverificate (early binding), mentre una ACL viene tipicamente verificata ad ogni accesso (late binding). Questo rende la revoca meno semplice in un contesto di capabilities. Inoltre, dal momento che le capabilities vengono assegnate per utente, è spesso difficile essere opportunamente granulari. D'altro canto, le ACL di un sistema molto usato diventano rapidamente ingestibili, costringendo l'amministratore ad assegnarle e revocarle per blocchi.

Va ricordato che sia ACL sia Capabilities possono essere amministrare per utenti o per ruoli.

### 3.3.3 Gestione pratica dei controlli d'accesso

**5.3.6.3** Le versioni di Windows client (*Windows 9X* ed *ME*), consentono di definire degli "utenti" attraverso lo strumento Utenti del Pannello del Controllo; non si tratta tuttavia di un sistema di protezione ma semplicemente di un modo per consentire l'utilizzo dello stesso PC da parte di più utenti con impostazioni personalizzate. Semplicemente premendo il tasto Esc alla richiesta di login e password, Windows termina comunque la fase di avvio consentendo l'accesso all'utente. Inoltre, il file system di questi sistemi è il FAT, che non contiene informazioni sui diritti di accesso. Pertanto, una volta avviato il PC, l'accesso è consentito a tutte le cartelle.

Le versioni server di Windows, *WindowsNT*, *2000* e *2003*, e le nuove versioni client (*2000 Professional* e *XP*) sono più protette, consentendo all'amministratore di indicare chi può accedere al sistema e con quali privilegi, e obbligando gli utenti al login (anche se sulle versioni client questo può essere disabilitato dall'amministratore).

L'utente *Administrator* è la chiave per la gestione di gruppi e utenti in queste versioni di Windows. Viene creato automaticamente durante l'installazione del sistema operativo ed è un utente privilegiato, autorizzato alla creazione di altri utenti e gruppi di utenti, all'assegnazione di password, alla concessione di autorizzazioni di accesso a cartelle e file del computer, all'accesso senza restrizioni alle cartelle di qualsiasi utente. La password dell'utente Administrator viene impostata durante l'installazione del sistema operativo ed è indispensabile conoscerla per effettuare il primo login al sistema. Dal momento che l'utente Administrator ha tutto questo potere, è buona norma riservare tale account alla gestione del sistema, e utilizzare account normali per tutte le operazioni quotidiane.

Nel caso dei sistemi operativi UNIX e di Linux, i file system prevedono sempre la presenza di permessi, e l'accesso al sistema viene sempre preceduto dall'autenticazione. I comandi base per la gestione degli utenti sono:

```
useradd [opzioni] nome_utente
```

Questo comando crea un utente, o con l'opzione `-D` lo modifica. È possibile aggiungere le seguenti opzioni:

- `c commento` = un commento sull'utente;
- `d home_dir` = specifica la home directory dell'utente. Il default è `/home/nome_utente`;
- `e expire_date` = data di scadenza dell'account, nella forma `YYYY-MM-DD`;
- `f inactive_days` = numero di giorni dopo la scadenza della password oltre i quali l'account viene cancellato;
- `g initial_group` = il nome o il GID del gruppo iniziale dell'utente;
- `G group,[...]` = il nome di gruppi supplementari di cui l'utente farà parte, separati da virgole;
- `m` = crea la home directory, se non esiste;
- `s shell` = specifica la shell usata dall'utente;
- `u uid` = l'UID (User ID) dell'utente, nel caso ne volessimo specificare uno.

Tutte le informazioni sono salvate nel file `/etc/passwd` (a parte la password, che come dicevamo viene salvata in `/etc/shadow`). Le righe sono del tipo

```
username:x:UID:GID::/home/username:/bin/bash.
```

Se esiste una directory `/etc/skel` tutti i file ivi contenuti saranno salvati per default nella nuova directory di un utente.

```
groupadd [-g GID [-o]] nome_gruppo
```

dove l'opzione `-g` è opzionale e permette di specificare il **GID** (*Group ID*) nel caso lo si voglia indicare. Ogni gruppo ha un proprio GID univoco, e se volete assegnarlo a mano evitate quelli tra 0 e 99 che sono riservati per uso di sistema.

```
passwd [username]
```

Questo comando serve per modificare la password di un utente. Se dato senza parametri, serve a modificare la propria. Solo root può modificare la password altrui.

Per assegnare i permessi è necessario capire il modo in cui Linux e gli UNIX li visualizzano e li gestiscono.

Ad esempio, il comando `ls -l` visualizza per ogni file una riga di permessi con questa struttura:

```
-rwx rwx rwxuser  group  nomefile
```

`r` significa “accesso in lettura”, `w` significa “accesso in scrittura” e `x` significa “accesso per l’esecuzione”, o nel caso di directory “permesso di visualizzare il contenuto”. Il primo blocchetto di permessi riguarda l’utente proprietario (visualizzato come `user` nell’esempio), il secondo blocchetto riguarda il gruppo (visualizzato come `group`) ed il terzo blocchetto “tutti gli altri utenti del sistema”. Una directory verrà segnalata con una `d` iniziale.

Per modificare i permessi su un file si può usare il comando:

```
chmod XYZ file
```

dove:

`X` rappresenta un qualificatore (`a` - tutti, `u` - l’utente owner, `g` - il gruppo owner)

`Y` rappresenta un’operazione (+ per aggiungere, - per togliere)

`Z` rappresenta un permesso o più (r, w, x)

Ad esempio:

```
chmod u+r file per dare al proprietario il permesso in lettura su file
```

```
chmod g-w file per togliere al gruppo il permesso di scrittura su file
```

Alternativamente si può usare:

```
chmod ABC file
```

dove invece di `A`, `B` e `C` vengono inserite le tre triplete di accesso, codificate in binario, ad esempio:

```
rw- r-x r-- diventa 110 101 100 ovvero 654
```

```
chown user:group file
```

Questo comando assegna il file *file* all’utente *user* e al gruppo *group*. Entrambi i comandi hanno altri switch, ad esempio per attivare la ricorsione, che possono essere appresi mediante la funzione di help `man chown` o `man chmod`.

### 3.3.4 Controllo d'accesso nei database

5.3.6.4

I database sono ovviamente un elemento chiave da considerare nel processo di sicurezza aziendale, in quanto costituiscono il cuore del sistema informativo. Essi custodiscono dati che possono essere un bersaglio molto appetibile (dall'abusato esempio dei "numeri di carta di credito" a dati finanziari, strategici o commerciali).

Non si può pretendere di descrivere estensivamente i modelli di controllo d'accesso nei database, ma focalizzandosi sugli **RDBMS** (*Relational Database Management System*), ovvero sulle basi di dati relazionali che al giorno d'oggi sono sicuramente le più utilizzate, verrà descritta l'architettura di sicurezza di base di un sistema di questo tipo che utilizzi il linguaggio SQL.

Un database è costituito principalmente da tabelle di dati. Generalmente, i permessi assegnati agli utenti del database riguardano quindi il diritto di accedere alle tabelle, e agli oggetti costruiti su di esse (viste, stored procedure...) con privilegi granularmente diversi.

Bisogna in generale distinguere quindi in due fasi il processo di autenticazione e controllo d'accesso: innanzitutto l'utente viene autenticato in fase di connessione, e in seguito ogni suo comando viene verificato per stabilire se ha i privilegi necessari a compierlo.

Il controllo degli accessi viene eseguito tramite le informazioni memorizzate all'interno di alcune tabelle di sistema dette anche **grant tables**. Per fare un esempio, qui e nel seguito, si farà riferimento all'architettura di MySQL, un robusto database sviluppato come software libero.

**Tabella 3.1** Tabelle di sistema di un RDBMS.

| TABELLA      | INFORMAZIONI  |
|--------------|---|
| user         | Determina a quali utenti è consentita la connessione  |
| db           | Memorizza i database ai quali un utente può accedere, e le operazioni che può eseguire su di essi |
| host         | Specifica da quali host un utente può connettersi   |
| tables_priv  | Indica i privilegi di ogni utente a livello di tabella  |
| columns_priv | Indica i privilegi di ogni utente a livello di colonne  |

Nella Tabella 3.2 sono riportati tutti i privilegi insieme con il relativo contesto di applicazione.

**Tabella 3.2** Privilegi e contesto di applicazione.

| PRIVILEGIO | CONTESTO                     |
|------------|------------------------------|
| select     | Tabella                      |
| insert     | Tabella                      |
| update     | Tabella                      |
| delete     | Tabella                      |
| index      | Tabella                      |
| alter      | Tabella                      |
| create     | Database, Tabella, Indice    |
| drop       | Database, Tabella            |
| grant      | Database, Tabella            |
| references | Database, Tabella            |
| reload     | Amministrazione del server   |
| shutdown   | Amministrazione del server   |
| process    | Amministrazione del server   |
| file       | Accesso file del server host |

È da ricordare che per le operazioni sui dati (ad esempio INSERT, UPDATE, ecc.) il DBMS verifica innanzitutto la presenza di privilegi globali (tabella “user”), nel qual caso l’accesso è sempre consentito; oppure di privilegi specifici su quel database consentiti per quell’host (incrocio tra tabelle host e database); o ancora di privilegi specifici per la tabella, o la colonna coinvolta.

Riassunto, esistono:

1. *privilegi a livello globale*: riguardano tutti i database del server e sono memorizzati nella tabella user;
2. *privilegi a livello di database*: riguardano tutti gli oggetti di un certo database e sono memorizzati nella tabella db;
3. *privilegi a livello di tabella*: riguardano la singola tabella di un singolo database e vengono memorizzati nella tabella tables\_priv;
4. *privilegi a livello di colonna*: riguardano la singola colonna di una singola tabella e sono memorizzati nella tabella columns\_priv.

I privilegi vengono concessi e revocati usando i comandi SQL grant e revoke, o alternativamente inserendo oppure cancellando direttamente dei record

nelle tabelle sopraelencate. Tuttavia nel secondo caso sarà necessario indicare a MySQL di ricaricare le tabelle dei privilegi con i comandi `mysqladmin flush-privileges` o `mysqladmin reload`.

In MySQL esistono inoltre dei nomi mnemonici come ad esempio `ALL_PRIVILEGES`, che è l'equivalente dei diritti di `SELECT`, `INSERT`, `UPDATE`, `INDEX`, `ALTER`, `CREATE`, `DROP`, `DELETE` e `GRANT`. Tuttavia è importante fare attenzione a non assegnare involontariamente privilegi troppo ampi: è spesso buona procedura assegnare i privilegi uno a uno, in modo da capire esattamente cosa si sta facendo.

Se invece si prende in considerazione SQL Server (il DBMS di Microsoft), si osserva che prima di poter accedere alle risorse del database è necessario passare attraverso l'autenticazione di sistema del server Windows sottostante. Fondamentalmente esistono due modalità di autenticazione:

- *Windows Authentication Mode*: modalità di default, agli utenti locali non occorre fornire login e password per connettersi a SQL Server perché l'autenticazione viene mutuata da Windows. In questi casi l'amministratore del database deve semplicemente specificare gli utenti/gruppi di sistema operativo che hanno il permesso di autenticarsi (e chiaramente i loro privilegi).
- *Mixed Mode*: è possibile creare account di login apposta per SQL Server che non hanno nulla a che vedere con quelli di sistema, che vengono salvati nella tabella `sysxlogins`.

Per quanto riguarda i privilegi, SQL Server dalla versione 2000 utilizza un sistema basato su ruoli. Un ruolo è una collezione di privilegi, e assegnare un ruolo a un utente o un gruppo attribuisce tutti quei privilegi all'utente o al gruppo. Fondamentalmente un ruolo è uno strumento per semplificare la vita dell'amministratore. Esistono varie categorie di ruoli:

- *public role*: questo ruolo stabilisce i permessi di default per gli utenti di un database; non può essere eliminato ed è presente in tutti i database, inclusi quelli di sistema.
- *fixed server roles*: sono ruoli predefiniti globali e trasversali ai vari database. Questi ruoli non sono modificabili e non è possibile crearne altri dello stesso tipo. Vanno associati agli user id per attribuire queste facoltà amministrative. Nella Tabella 3.3 sono elencati con una sommaria descrizione.
- *fixed database roles*: anche questi sono ruoli predefiniti, ma esistono ciascuno nel contesto dei singoli database. Generalmente vengono concessi agli account utente; i permessi associati non possono essere modificati. Nella Tabella 3.4 viene riportata una lista di questi ruoli insieme con l'indicazione delle loro caratteristiche.

- *user defined database roles*: ogni amministratore può definire nuovi ruoli a livello di singolo database per raggruppare i permessi che caratterizzano l'attività di un certo gruppo di utenti specifico per la sua applicazione.
- *application roles*: consentono di restringere l'accesso ai dati in base all'applicazione in uso per l'accesso, permettendo di delegare ad essa la responsabilità dell'autenticazione dell'utente. L'amministratore crea il ruolo e gli assegna i singoli permessi necessari all'applicazione, mentre quest'ultima attiverà il ruolo durante l'esecuzione, attraverso l'invocazione di una procedura registrata (`sp_setapprole`) così da ottenere i permessi. I ruoli applicativi non possono essere attribuiti ai singoli utenti; non si sommano i privilegi dell'utente che esegue l'applicazione con quelli del ruolo dell'applicazione (l'applicazione perde all'attivazione tutti i permessi normalmente associati con il login dell'utente che l'ha eseguita). I ruoli applicativi esistono soltanto nel contesto di un singolo database, per cui se l'applicazione deve iniziare una transazione che interessa un altro database bisognerà opportunamente provvedere.

In SQL Server a `grant` e `revoke` si aggiunge il privilegio `deny`, che può negare l'accesso ad oggetti anche quando esiste una `grant` che lo consentirebbe. Sostanzialmente consente di negare con granularità maggiore dei privilegi rispetto a un `grant` troppo esteso.

**Tabella 3.3** Ruoli di SQL Server: fixed server roles.

| RUOLO         | DESCRIZIONE   |
|---------------|---|
| sysadmin      | È in grado di compiere qualsiasi attività   |
| serveradmin   | Amministra le opzioni di configurazione del server; può eseguire shutdown e restart                             |
| setupadmin    | Gestisce ed amministra i server collegati e le procedure di startup   |
| securityadmin | Gestisce le impostazioni di sicurezza e possiede il permesso CREATE DATABASE.<br>Gestisce i login e le password |
| processadmin  | Può terminare i processi di SQL Server  |
| dbcreator     | Può creare, modificare e ripristinare qualsiasi database di SQL Server  |
| diskadmin     | Gestisce i file su disco  |
| bulkadmin     | Permette agli utenti diversi dagli amministratori di sistema di eseguire le istruzioni bulkadmin                |

**Tabella 3.4** Ruoli di SQL Server: fixed database roles.

| <b>RUOLO</b>      | <b>DESCRIZIONE</b>   |
|-------------------|--|
| db_owner          | Esegue tutte le attività di manutenzione e configurazione del singolo database                             |
| db_accessadmin    | Aggiunge o rimuove le autorizzazioni di accesso al database  |
| db_datareader     | È in grado di leggere tutti i dati da tutte le tabelle   |
| db_datawriter     | È in grado di aggiungere, cancellare e modificare dati in tutte le tabelle                                 |
| db_ddladmin       | Può eseguire comandi DDL (Data Definition Language)  |
| db_securityadmin  | Attribuisce l'appartenenza ai ruoli e gestisce i permessi  |
| db_backupoperator | Autorizzato ad eseguire il backup del database   |
| db_denydatareader | Chi ha questo ruolo non può leggere i dati di nessuna tabella del database                                 |
| db_denydatawriter | Chi ha questo ruolo non può aggiungere, modificare o cancellare dati in nessuna delle tabelle del database |

