# Sistemi 11

Eravamo rimasti all’esecuzione di una syscall da parte di un processo: il Sistema operativo deve verificare che sia sensata e che rispetti la politica di sicurezza. Un utente non parla direttamente col kernel, per questo si passa attraverso l’autenticazione (verifica dell’identità), l’autorizzazione (applicazione di una politica di sicurezza: cioè decidere se accettare/rifiutare la richiesta sulla base dell’identità di chi fa la richiesta) e l’accounting (contabilità): il logging e la gestione del consumo di risorse (è importante tenere traccia di chi e come consuma le risorse).

Come ci si identifica in un sistema UNIX? Come in tanti altri sistemi, il modo di default è usare l’username e password. Per questi esistono due file, “etc/passwd” e “etc/shadow”. In passwd Non ci sono le password, ad ogni riga corrisponde il nome di login, una x che indica che la password sta in shadow, l’identificatore dello user, l’identificatore del gruppo, il nome dell’utente, la user home directory e la shell corrispondente. Le password inizialmente stavano in passwd (o meglio venivano salvati gli hash delle password, hash che non devono essere invertibili e devono essere lenti), tuttavia così un qualunque utente poteva prendersi questi hash e fare password cracking, per questo motivo le password sono state spostate su un altro file, shadow, che a differenza di passwd non è leggibile da tutti, ma solo da root (passwd è leggibile da tutti, ma scrivibile solo da root). L’hash è generato fondendo la password con un numero generato randomicamente, chiamato “sale”.

Ogni utente ha un proprio UID, così ogni gruppo ha il proprio GID. L’UID zero corrisponde all’utente root, l’amministratore del sistema (che non è la radice del file system). Tradizionalmente i processi si dividono in privilegiati con UID = 0 e non privilegiati con UID != 0. Tuttavia, non è esattamente il massimo dare solo massimo e minimo privilegio, quindi col tempo sono state introdotte le Linux Capabilities, che permettono di avere una maggiore granularità dei privilegi (un processo che apre una porta non necessariamente deve poter fare anche il reboot del sistema).

In generale, l’amministratore di sistema coincide con root e può quindi fare tutto, tuttavia è pericoloso rimanere sempre come root, quindi, anche se siamo amministratore di sistema conviene di più girare come utente e diventare root solo quando dobbiamo compiere azioni particolari che richiedono modalità supervisore. Questo modo di fare è detto principio di minimo privilegio.   
Nota: questo non si applica nei telefoni, anche se siamo il proprietario di uno smartphone non siamo l’utente root (a meno che non lo rootiamo).

Ci sono due approcci generali all’autorizzazione, la prima è l’access control list: a ogni oggetto associo una lista di utenti specificando cosa può fare il singolo utente (in Unix viene ridotta/ottimizzata coi 9 bit rwx per proprietario, gruppo e altri). Il secondo è quello del capabilities: per ogni oggetto ci sono delle “chiavi” che ne permettono l’uso (e solo alcuni utenti le hanno e altri no).

In ogni caso, da qualche parte devo avere un elenco di informazioni sulle operazioni permesse da un certo soggetto verso un certo oggetto, c’è solo da decidere come memorizzare le informazioni e dove memorizzarle. C’è inoltre da considerare cosa fare quando aggiungiamo o eliminiamo utenti: questa problematica è il motivo per cui in Unix si usa l’access control list ridotta, se usassimo quella completa ogni volta che eliminiamo un utente dovremmo eliminarlo dalla access control list di TUTTI i file del sistema e un discorso analogo si applicherebbe quando ne aggiungiamo uno nuovo.

Indipendentemente da come memorizziamo queste informazioni, si distingue tra discretionary access control, in cui il proprietario della risorsa decide chi può accedere a quella risorsa (applicato dalla maggior parte dei sistemi operativi), e il mandatory access control, in cui le regole sono stabilite dal sistema (viene usato principalmente in ambito militare per documenti top secret). L’eccezione è che anche col discretionary access control, su Linux c’è il root che è Dio e può fare tutto sbattendosene il belino.

Come abbiamo accennato, ciascun processo non dovrebbe avere più privilegi di quelli che li servono per funzionare. In generale è importante considerare il ruolo: lo stesso utente può avere ruoli diversi in momenti diversi (a volte possiamo essere root e altre no). In Unix, per aumentare i privilegi si usava storicamente “su”, oggi si usa “sudo”.  
Un esempio è “login”, che parte come root per verificare la nostra identità e droppa i privilegi non appena ha verificato che noi siamo noi.

Come funziona sudo? Quando lo lanciamo ci viene chiesta la nostra password (non quella di root) e se siamo tra gli utenti che possono usare root (e sappiamo la password) il comando che lanciamo viene lanciato come root, ma solo quello, poi si ritorna a essere utenti normali (su invece chiedeva la password di root e faceva switchare a modalità super user ed era un po’ più pericoloso).

Il motivo per cui il fatto che sudo chieda la nostra password anziché quella di root è un bene, è dovuto al fatto che così nei sistemi a più amministratori si può cambiare la propria password senza rovinare la vita agli altri.

Ogni processo ha tre UID (Linux ne mette un quarto ma non ce ne frega una mazza).  
Il real UID è quello vero, che ci dice chi può chiamare kill, l’effective UID è l’identità usata per determinare i permessi di accesso alle risorse e il saved UID. Al login sono tutti e tre uguali.  
Tramite setuid un processo può modificare l’effective UID, facendolo diventare come il real o il saved (se il chiamante è privilegiato può modificare tutti e tre come vuole).

Un nuovo processo eredita gli id del parent e l’execve non cambia l’id.

Se il file eseguibile ha il bit set-userid a 1, l’effective UID e il Saved UID diventano quelli del proprietario del file (in genere, root). Questo è il modo con cui sudo rende se stesso temporaneamente root (quando ci chiede la password in realtà il comando è già root).

Il motivo per cui kill guarda il real UID è perché così se uno avvia un processo con sudo, poi per killarlo non serve essere root, basta essere noi stessi, anche come utente sfigato, perché tanto il real UID coincide.

Chroot è una syscall che permette di modificare il significato di “/” nella risoluzione dei percorsi assoluti (non i relativi): la modifica è per il processo e tutti i (futuri) figli, è necessario che il processo sia privilegiato quando la chiama. I FD non vengono toccati.   
Il vantaggio è che così facendo il processo vedrà solo una fettina di file system (non potrà tornare indietro rispetto alla sua root e accedere al resto del file system). Questo permette di creare le cosiddette chroot jail: limitano la visibilità del file system a una directory. È un’applicazione del minimo-privilegio, ma se il processo rimane privilegiato e/o la syscall non è correttamente associata a una chdir è possibile evadere dalla prigione.

Per esempio, in caso di bug, in incApache un client può sbirciare fuori da /www/…