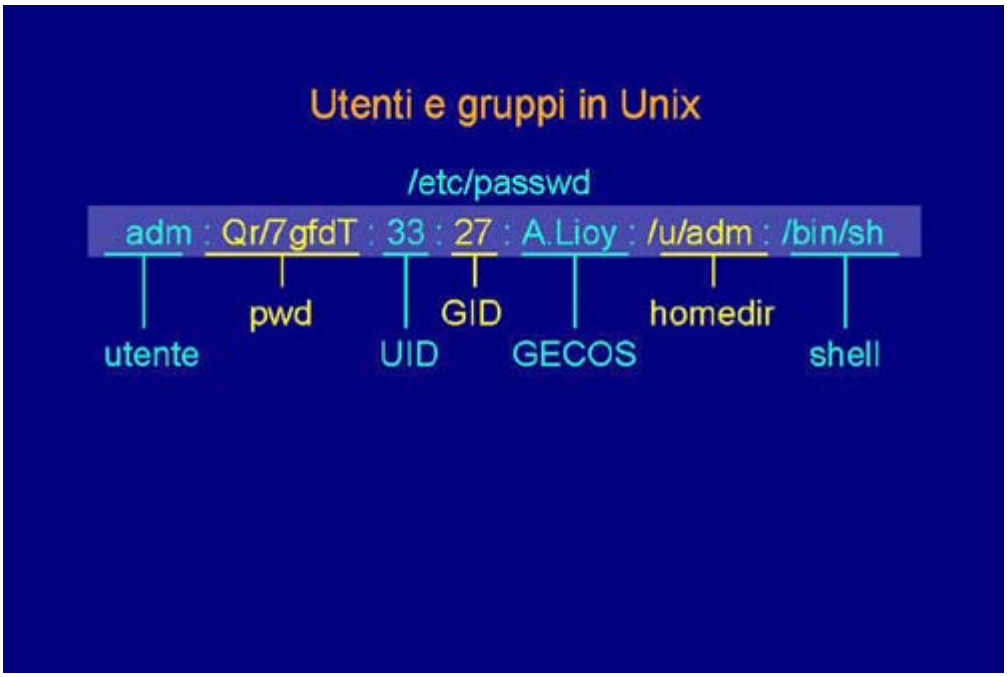
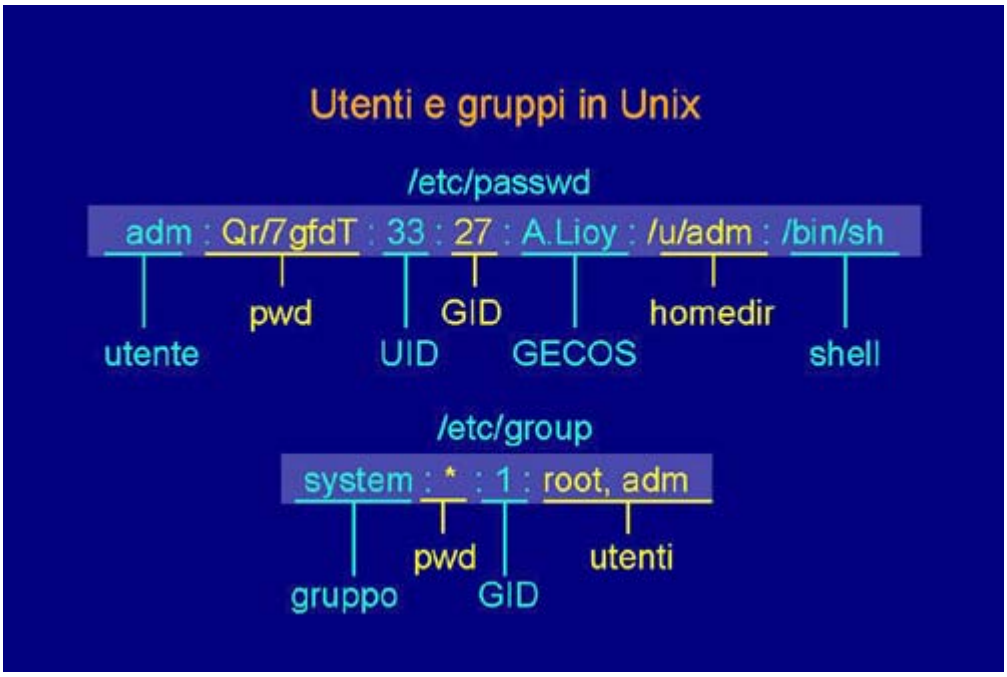
Sicurezza nei sistemi Unix
Utenti e gruppi in Unix (1/2)



In questa lezione tratteremo di alcuni concetti di sicurezza tipici dei sistemi Unix. In particolare, questi sistemi definiscono gli utenti abilitati ad utilizzare il sistema in un file chiamato passwd, che si trova all'interno del direttorio etc ed è un file di testo organizzato a righe. In ogni riga di questo file ci sono vari campi separati da :. Il primo campo, quello che qui contiene il valore adm è il nome dell'utente, diciamo il nome mnemonico dell'utente. Il secondo campo, che qui contiene il valore Qr/7gfdT, contiene la password di quell'utente; il fatto che ci sia questa scritta non significa che la password dell'utente sia realmente Qr/7gfdT, ma vuol dire che la password crittografata dell'utente da come risultato quello qui indicato. Il terzo campo indica la UID, ossia l'identificatore univoco, il numero che è stato assegnato a questo utente. In realtà il sistema Unix lavora identificando utenti ed oggetti tramite questi numeri, quindi la scritta adm, il nome mnemonico, è solo un aiuto per l'utente per dover evitare di ricordarsi sempre i numeri univoci. Analogamente, il quarto campo contiene un altro numero, che è il cosiddetto identificatore di gruppo: ossia questo utente appartiene primariamente al gruppo numero 27. Vedremo tra breve dove esistono dei nomi mnemonici associati a questi gruppi numerici.

Utenti e gruppi in Unix (2/2)



Il quinto campo contiene delle informazioni ausiliarie, è detto campo GECOS e prende il nome da un antico sistema Unix in cui questo campo aveva una importanza particolare. Oggigiorno viene normalmente utilizzato per mettere informazioni ausiliarie non trattate automaticamente dal sistema ma di aiuto per il sistemista. Il penultimo campo contiene la cosiddetta home directory, ossia il punto del file system di Unix in cui sono memorizzati i file di questo utente. In questo caso particolare si tratta del direttorio adm all'interno del direttorio u. Infine, l'ultimo campo contiene la cosiddetta shell, ossia il programma che deve essere automaticamente attivato dopo la procedura di login di questo utente. Il sistema Unix utilizza anche un altro file, chiamato group, sempre nel direttorio etc, che serve a fornire la corrispondenza fra identificativi numerici di gruppo e nome mnemonico. Il primo campo contiene, appunto, il nome mnemonico di un gruppo. Il secondo campo contiene una password, che però normalmente in tutti gli attuali sistemi Unix contiene il campo *, nel senso che le password da lungo tempo non vengono più utilizzate per gli accessi ai gruppi. Il terzo campo contiene l'identificativo numerico del gruppo ed il quarto campo contiene l'elenco di tutti gli utenti che appartengono al gruppo in questione. Ovviamente anche questo gruppo è costituito da tante righe, una per ogni gruppo che è stato definito nel sistema. Questi due file rappresentano un problema se non gestiti correttamente, perché per il corretto funzionamento di Unix questi file devono essere leggibili da tutti quanti gli utenti.

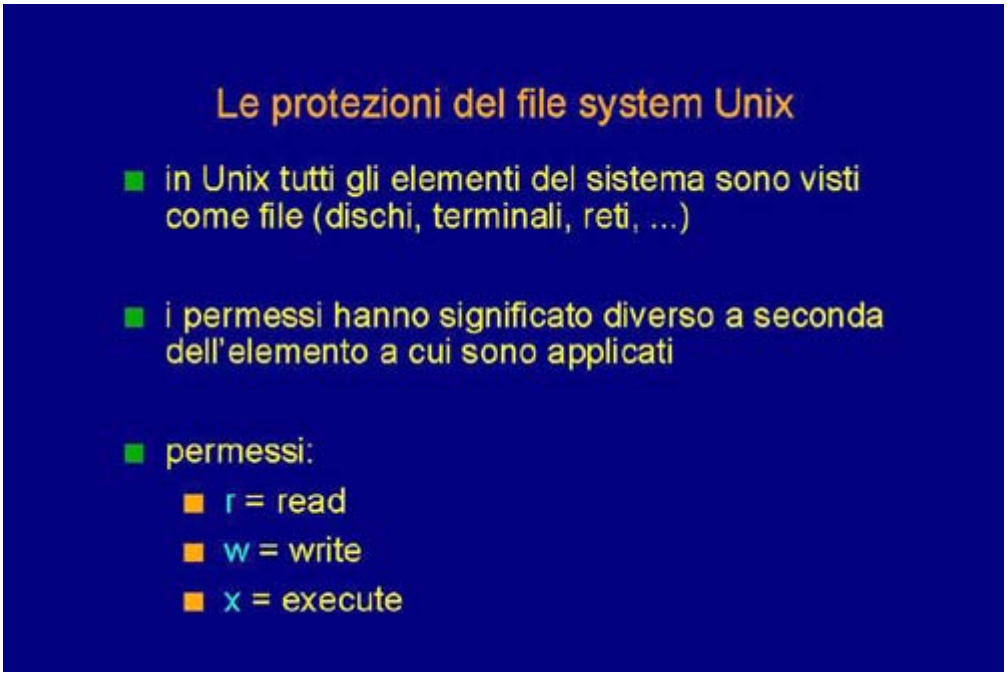
Procedura di login Unix

In particolare, questi file vengono letti durante la procedura di login al sistema. La procedura di login segue i seguenti passi: avendo l'utente fornito uno username e una password, per accedere al sistema tramite lo username indicato, la procedura di login seleziona una riga del file /etc/passwd, quella corrispondente a questo utente. Successivamente la password viene letta, crittografata e confrontata con la password presente nel secondo campo della riga di /etc/passwd selezionata. Se il confronto della password ha avuto successo allora il login è permesso e il sistema si sposta nella home directory, ossia nel punto dove sono memorizzati i file dell'utente. A partire da questa home directory viene attivata una shell, ossia il programma che deve essere attivato automaticamente al login e questa shell viene eseguita, non con i permessi di sistema, ma con i permessi che derivano dagli identificativi univoci di gruppo e di utente. Infine, ogni shell può avere uno o più file di inizializzazione, che sono tipici di questa shell. Questi possono essere file di inizializzazione di sistema oppure personali dell'utente e servono, in qualche modo, a personalizzare il funzionamento della shell.

La shadow password

Se qualcuno è in grado di andare a leggere le password, come abbiamo già detto in passato, può provare a montare un attacco di tipo dizionario, ossia può cercare di indovinare la password. Per evitare questo problema, nel sistema Unix è stato sviluppato il concetto di shadow password, o password ombra, o password nascoste, come preferite. In parole povere si tratta semplicemente di questa cosa: dal file /etc/passwd vengono rimosse le password crittografate, che vengono invece salvate in un altro file accessibile soltanto al sistema. In questo modo si permette agli utenti di leggere /etc/passwd per effettuare le normali corrispondenze, ad esempio fra UID e nome mnemonico dell'utente, ma si toglie il permesso di leggere le password crittografate e quindi di montare attacchi di tipo dizionario. Come vedete, in questo caso se utilizziamo le shadow password quello che capita è che nel file /etc/passwd il secondo campo contiene un asterisco, ossia un indicazione che per questo utente la password non si trova all'interno di questo file. In realtà la password viene memorizzata dentro il file /etc/shadow, ossia il file shadow nel direttorio etc. Questo file è anch'esso organizzato a righe, per ognuna delle quali c'è nome mnemonico dell'utente e la sua password crittografata. La differenza è che il file /etc/passwd è leggibile da qualunque utente del sistema, mentre il file /etc/shadow richiede i cosiddetti privilegi di super-utente, di cui parleremo fra poco.

La protezione del file system Unix



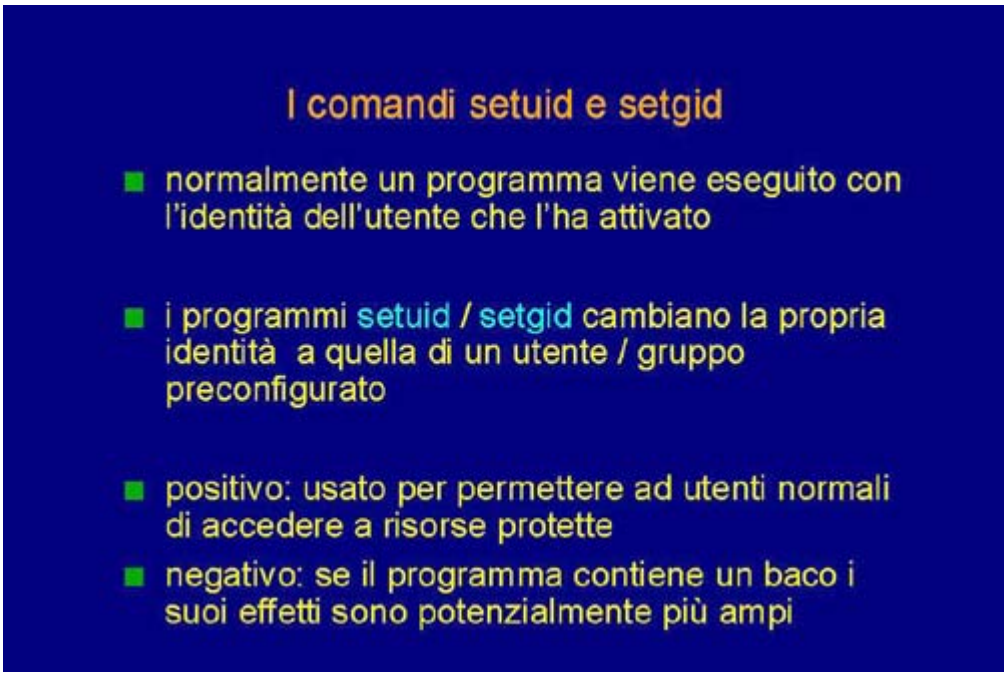
In generale, all'interno del sistema Unix è stata fatta una scelta semplificativa, quella di trattare tutti quanti gli oggetti del sistema come file. Ossia, sia che stiamo parlando veramente di un disco organizzato a file, sia che stiamo parlando di un generico driver di un dispositivo (una scheda di rete, un terminale, una linea seriale, una linea di comunicazione), tutti quanti questi oggetti vengono rappresentati come file. Questo da una parte semplifica la gestione, dall'altra complica un po' il controllo dei permessi di accesso a questi oggetti. I permessi quindi hanno significato diverso a secondo dell'elemento a cui sono applicati. Esistono tre permessi base in Unix: i permessi di lettura, di scrittura e di esecuzione. Questi tre permessi hanno un significato abbastanza chiaro nel caso che stiamo parlando di file, ma nel caso in cui stiamo parlando, ad esempio, di un direttorio, ovviamente il permesso di esecuzione non avrebbe di per sé senso, perché un direttorio non è un programma eseguibile. Allora, in questo esempio, avere il permesso di esecuzione su un direttorio significa avere il permesso di entrare o di attraversare questo direttorio durante uno spostamento nel file system. Analogamente, non esiste un'operazione di permesso di cancellazione, ma viene ottenuto tramite il permesso di scrittura nel direttorio, perché a tutti gli effetti cancellare un file significa rimuovere questo file dal direttorio corrente. Dopo un po' ci si fa l'abitudine, ma è chiaro che non è

una cosa facile da imparare e da capire. Questo può portare a degli errori di configurazione che possono essere sfruttati, da persone malintenzionate, per manipolare impropriamente file del nostro sistema.

I permessi

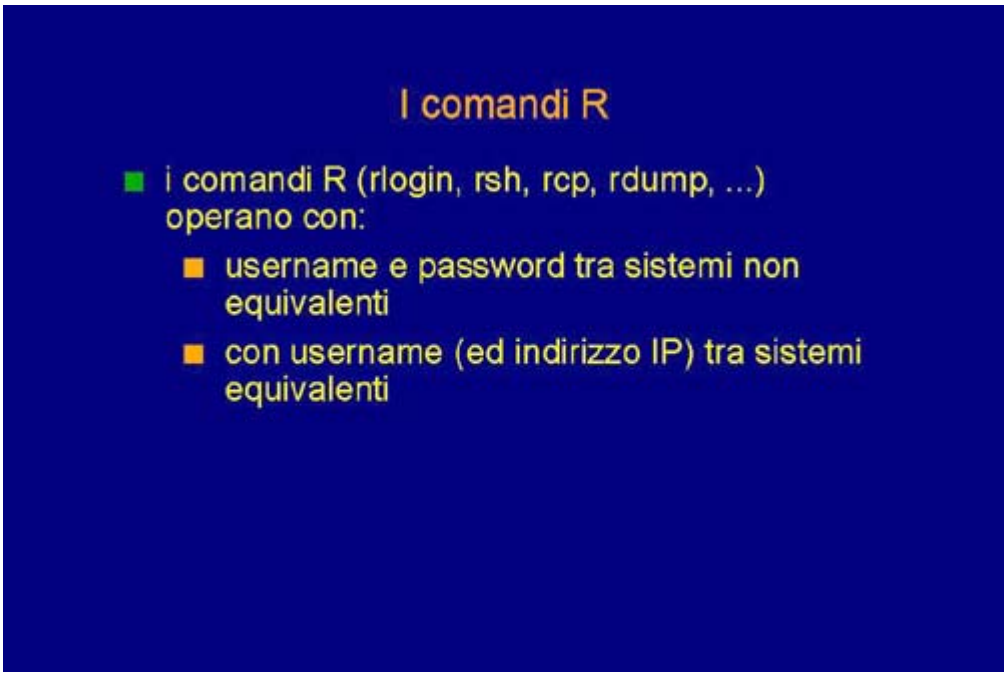
In generale, i permessi sono attribuiti agli oggetti e sono specificati in modo diverso per tre identità che normalmente, colloquialmente, sono chiamati UGO. In realtà UGO è l'acronimo di User, Group e Others: indica che i permessi vengono specificati per il proprietario dell'oggetto, per il gruppo a cui l'oggetto appartiene, ossia in pratica il gruppo degli amici del proprietario, e infine sono specificati degli altri permessi anche per others, ossia per il resto del mondo, tutti gli altri utenti che non sono né il proprietario né il gruppo. Notate che nel caso ipotetico, difficile da realizzarsi ma teoricamente possibile, in cui l'utente ha meno diritti del resto del mondo, si applicano i permessi più restrittivi, quelli di utente, perché non appena si fa un match con una di queste tre entry si applicano i permessi relativi. Bisogna notare che in Unix esiste un utente che può fare tutto senza rispettare alcun tipo di permesso, questo è il cosiddetto super-utente. Il super-utente è chiunque abbia UID pari a zero. Normalmente UID = 0 viene data esclusivamente all'utente chiamato root. Una operazione tipica degli hacker che riescono a manipolare il file /etc/passwd, è quella di assegnare UID = 0 anche ad altri utenti che apparentemente non hanno permessi speciali, perché non hanno il nome di root. Visto che quel che conta non è il nome ma la UID, se questi utenti hanno UID uguale a zero, hanno a tutti gli effetti il permesso di fare qualunque cosa all'interno del nostro sistema. Quindi, prestare particolare attenzione al fatto che nel file /etc/passwd esista un'unica riga con UID pari a zero e questa riga deve corrispondere con l'utente root.

I comandi setuid e setgid



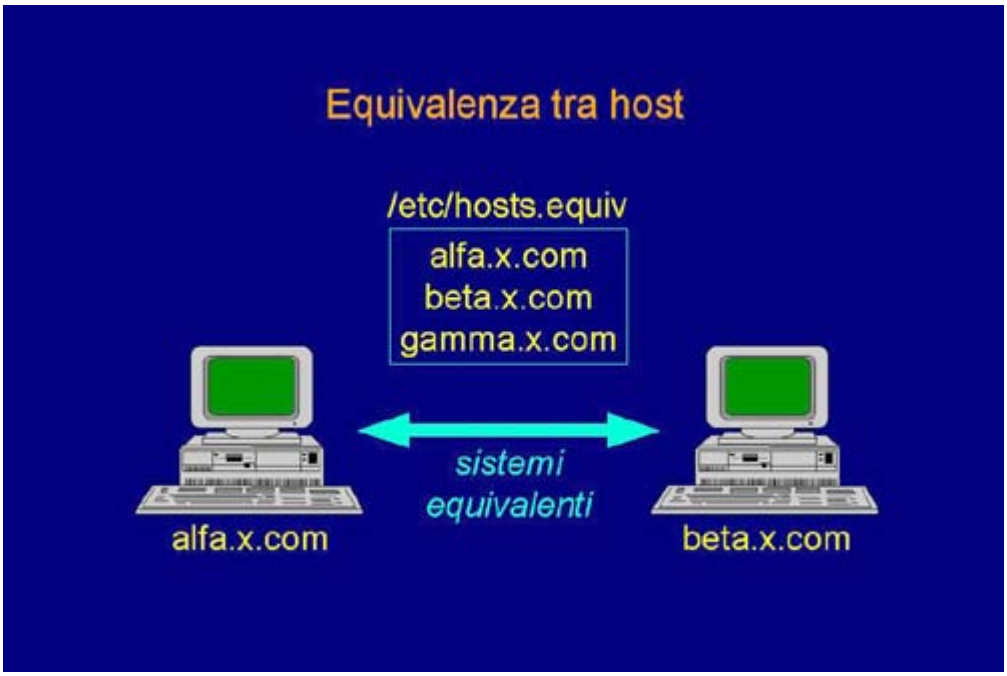
Normalmente, quando un programma viene eseguito da un utente, viene eseguito con i permessi relativi a quel utente. Questo impedirebbe normalmente agli utenti di accedere ed utilizzare delle risorse che richiedono invece i permessi di sistema. Ad esempio, nel caso di un calcolatore multi-utente, i dispositivi di collegamento, come la scheda di rete, devono essere gestiti dal sistema per risolvere i conflitti e ciò vorrebbe dire che nessun utente potrebbe accedere direttamente alla scheda di rete o a dispositivi analoghi, ad esempio pensiamo ad una stampante condivisa. Per risolvere questo problema, Unix ha introdotto i comandi setuid e setgid. Questo significa che al posto di essere eseguiti con i privilegi dell'utente che ha attivato il programma, setuid e setgid sono in grado di cambiare la propria identità a quella di un utente o di un gruppo, nel caso di programma setgid preconfigurato. Tipicamente questo viene usato in modo positivo per permettere ad utenti nomali di accedere a risorse protette. Ad esempio, possiamo dare a più utenti il permesso di controllare lo stato della stampante facendo un comando setuid root, ossia un comando che solo per l'esecuzione di quel particolare programma permette di accedere alla stampante, pur non essendo sistemista. Ma i programmi setuid o setgid possono anche avere un effetto negativo. Ad esempio, se il programma contiene un baco ovviamente i suoi effetti sono potenzialmente molto più ampi se il programma viene eseguito come root, quindi con i massimi privilegi di sistema, invece che con i privilegi tipici di un solo utente. Quindi, questa è un'altra cosa a cui bisogna prestare attenzione: bisogna assicurarsi che i programmi setuid e setgid siano solo quelli definiti dal sistema e che non ne siano stati introdotti degli altri. Inoltre è necessario correggere rapidamente eventuali bachi presenti all'interno di questi programmi.

I comandi R



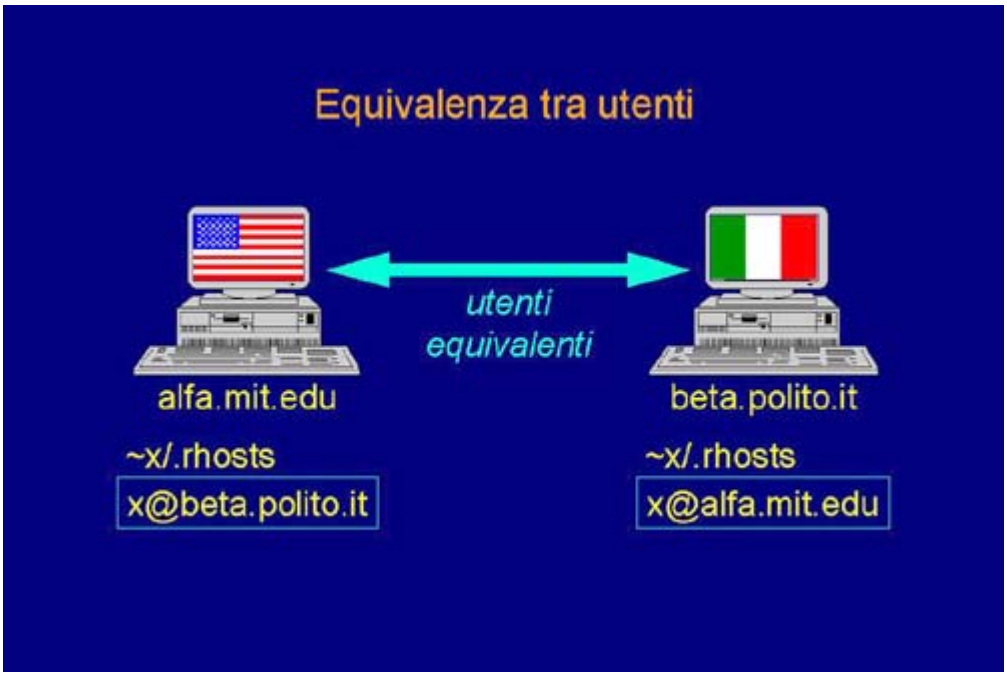
Quando da un sistema Unix si desidera svolgere delle operazioni su un altro sistema, esistono vari modi di farlo. Tipicamente si può ricorrere ad un collegamento in emulazione di terminale, come telnet, o un trasferimento di file, come FTP. Ma nel caso che siamo fra elaboratori collegati, tutti quanti, alla medesima rete locale, è abbastanza usuale usare i cosiddetti comandi R: un insieme di comandi che permettono l'esecuzione di una procedura remota su altri sistemi della nostra rete. Per esempio, con rlogin facciamo un emulazione di terminale, con rsh eseguiamo un particolare comando su un sistema remoto, rcp mi permette di copiare dei file, rdump mi permette di utilizzare un'unità nastro remota per fare un dump e così via. I comandi R possono funzionare richiedendo username e password, per permettere l'operazione sul sistema remoto, nel caso che i due sistemi coinvolti nello scambio di dati non siano equivalenti. Ma può lavorare anche in un modo che evita all'utente di dover ogni volta introdurre la password, semplicemente chiedendogli lo username e ovviamente l'indirizzo IP, questo fra sistemi equivalenti. Questa può essere la causa di un grosso problema di sicurezza all'interno dei sistemi Unix.

Equivalenza tra host



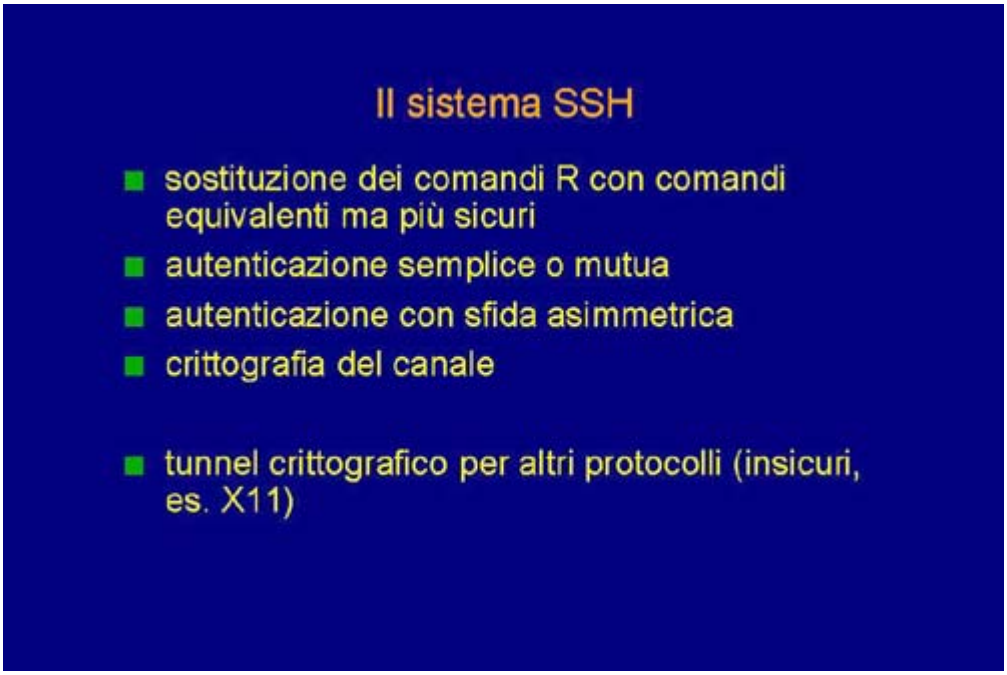
Il problema è legato alla definizione di sistemi equivalenti, ossia sistemi che possono permettere l'esecuzione di questi comandi R senza richiedere la password all'utente. È possibile definire due sistemi come completamente equivalenti, se su entrambe queste macchine viene introdotto nel direttorio etc il file chiamato hosts.equiv. Questo file contiene l'elenco di tutte le macchine che sono da considerarsi equivalenti dal punto di vista del login. Se un utente ha fatto login sulla macchina alfa e quindi si è già autenticato mostrando username e password, anche i sistemi beta e gamma accetteranno di eseguire per conto suo dei comandi, senza richiedergli ulteriormente la password. Poiché questa equivalenza tra host è definita in base al nome del calcolatore e quindi per traduzione in base al suo indirizzo IP, se qualcuno è in grado di assumere l'indirizzo IP di uno di questi sistemi, cosa che come abbiamo visto in passato è molto facile da fare, allora può a tutti gli effetti entrare facendo finta di essere un utente equivalente a quelli dei sistemi.

Equivalenza tra utenti



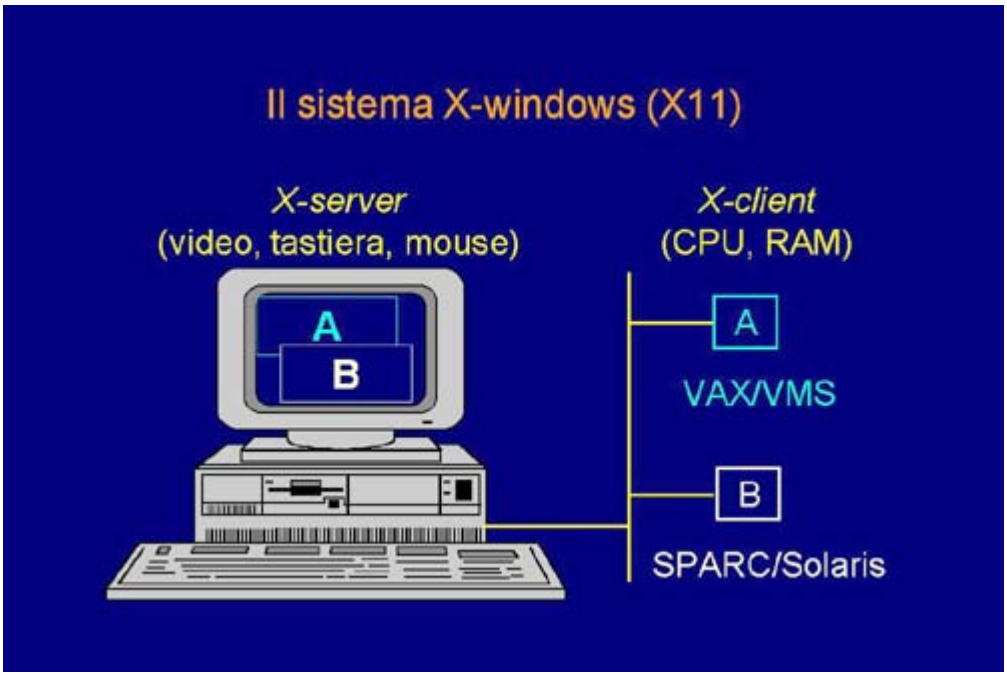
Esiste anche un altro tipo di equivalenza per i comandi R ed è l'equivalenza che non definisce tutti gli utenti di un sistema equivalenti a tutti gli utenti dell'altro sistema, ma definisce soltanto un equivalenza fra utenti specifici. Ad esempio, supponiamo di avere due calcolatori, uno situato in America e uno situato in Italia, ma di avere casualmente un utente che è il medesimo su entrambi i sistemi. Questo utente potrebbe desiderare di eseguire comandi R senza dover fornire la password ogni volta. Allora, è possibile definire non entrambi i sistemi come equivalenti, ma soltanto i due utenti come equivalenti. Per far questo occorre che l'utente, che abbiamo supposto avere login x, nella sua home directory (x indica il direttorio di login) abbia scritto un file chiamato .rhosts. All'interno di questo file deve essere messo il nome dell'utente e del calcolatore da considerare equivalente all'utente locale. Quindi, questo indica che l'utente x sul calcolatore alfa.mit.edu è da considerarsi equivalente all'utente x sul calcolatore beta.polito.it. L'equivalenza può essere monodirezionale o bidirezionale. Per averla bidirezionale occorre che anche sul calcolatore beta.polito.it venga messo un file analogo, che dichiari che l'utente x sul sistema americano è equivalente all'utente locale. Se questi file .rhosts non sono adeguatamente protetti, una persona malintenzionata può introdurre altre righe. Questo file può essere composto da più righe di testo e quindi introducendo altre persone queste possono essere dichiarate equivalenti ad un utente e quindi possono entrare su quel sistema senza fornire la password. Bisogna prestare particolare attenzione alle protezioni e al contenuto dei file .rhosts.

Il sistema SSH



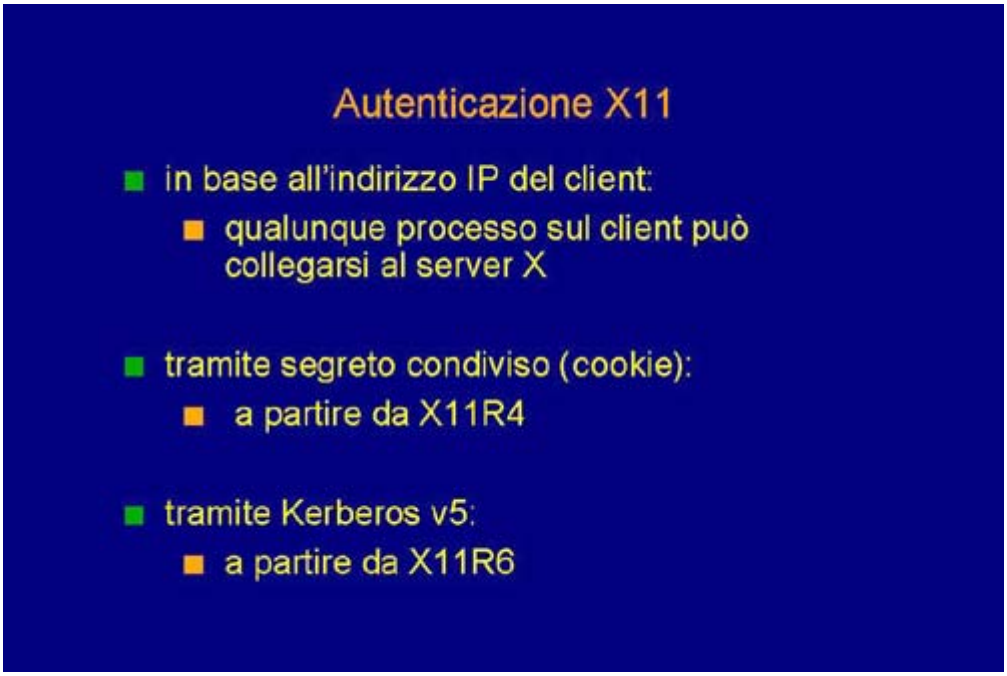
Una soluzione ancora migliore è non utilizzare i comandi R, visto che sono intrinsecamente insicuri. Esiste per Unix una sostituzione completa dei comandi R chiamato SSH. SSH è un programma sviluppato in Finlandia, ma diffuso ormai in tutto quanto il mondo, che sostituisce i comandi R con comandi equivalenti ma più sicuri. In pratica, permette di fare autenticazione semplice, quindi solo l'utente, o mutua, ossia anche il calcolatore mostra le proprie credenziali. L'autenticazione può essere fatta tramite una sfida asimmetrica, ossia usando chiavi pubbliche e chiavi private, che vengono assegnate sia all'utente sia ai sistemi a cui vogliamo collegarci. Inoltre, il canale di comunicazione tra l'utente ed il sistema viene crittografato, quindi anche un eventuale attaccante che potesse osservare la rete non può leggere i nostri dati. Una particolarità molto interessante del sistema SSH è l'esistenza di un tunnel crittografico, ossia SSH può veicolare al suo interno altri protocolli. Tipicamente questo viene fatto per protocolli insicuri, quali il sistema X11 che andiamo adesso ad esaminare.

Il sistema X-windows



Il sistema X11 è un altro dei prodotti del famoso progetto Athena del MIT, di cui abbiamo già parlato a proposito del sistema Kerberos. Il sistema X11 è un sistema distribuito, ossia che funziona su una rete di calcolatori, grafico a finestre. Vedete qui l'illustrazione: nella terminologia di X-windows si parla di X-client per quei programmi che stanno svolgendo del calcolo, che quindi usano una certa CPU, una certa RAM e hanno bisogno del servizio di input/output nei confronti dell'utente. Il servizio di input/output viene fornito dal X-server, che è un programma che gira su una stazione personale dell'utente. Il X-server, in pratica, offre il servizio di video, tastiera, mouse ai programmi remoti. Trattandosi di un programma distribuito in rete, esso è anche eterogeneo, ossia le architetture dei programmi coinvolti possono essere diverse, come qui illustrato.

Autenticazione X11



L'autenticazione all'interno del sistema X normalmente è molto debole, nel senso che esistono tre metodologie base. Quella utilizzata più di tutte è la prima che vedete indicata, perché è la più antica.

È una autenticazione in base all'indirizzo IP del client: l'utente che controlla l'X-server, decide quali sono i client che possono accedere, ma purtroppo la granularità è solo a livello di indirizzo. Questo significa che qualunque processo che sia in esecuzione sul X-client può collegarsi al server e quindi può leggere quello che noi stiamo battendo sulla tastiera, o può visualizzare l'intero schermo e, a questo punto, può sottrarci qualunque tipo di dato. A partire dalla versione X11R4 è stato introdotto un altro tipo di autenticazione, i cosiddetti cookie, biscottini magici, che sono un segreto condiviso tra client e server. Solo i client che mostrano di conoscere questo cookie hanno diritto di accesso al server. A partire dalla versione X11 release 6, è stata anche introdotta l'autenticazione con Kerberos versione 5. Le autenticazioni con cookie e con Kerberos sono sicuramente più forti della normale autenticazione basata sull'indirizzo IP. Purtroppo, nella maggior parte dei casi, le implementazioni correnti utilizzano solo gli indirizzi e quindi il sistema X11 è potenzialmente fonte di grossi problemi all'interno del nostro sistema Unix.