Corsi Linux – Amministrazione di Sistema

Prima Lezione – Seconda Parte

Gestione degli Utenti e dei Permessi

POuL

have a lot of fun...

UTENTI

- Linux è un S.O. multiutente, questo significa che lo stesso sistema può essere utilizzato contemporaneamente da più utenti.
- Ogni utente fa parte di almeno un **gruppo**.
- Gli utenti, a ognuno dei quali è associato un **account**, possono accedere solo ed esclusivamente a file e programmi che riguardano loro o uno dei gruppi di cui fanno parte.
- L'unica eccezione è l'utente amministratore, a cui è convenzionalmente associato l'**account** *root*, che può letteralmente scavalcare qualsiasi limitazione.

/etc/passwd:

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
adm:x:3:4:adm:/var/adm:/bin/false
lp:x:4:7:lp:/var/spool/lpd:/bin/false
sync:x:5:0:sync:/sbin:/bin/sync
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/bin/false
news:x:9:13:news:/usr/lib/news:/bin/false
uucp:x:10:14:uucp:/var/spool/uucppublic:/bin/false
kdm:x:114:999:added by portage for kdm:/var/lib/kdm-4.6:/sbin/nologin
poul:x:1000:1003::/home/poul:/bin/bash
john:x:1001:1012:John Titor,Viaggi nel Tempo,42,24:/home/bob:/bin/zsh
nobody:x:65534:65534:nobody:/:/bin/false

nome_utente:password:uid:gid:g(e)cos:home:shell

/etc/shadow:

root:$6$24cbiUTm$iuhf73rhfef/OZBM9r6bVmB:15000:0:::::
halt:*:9797:0:::::
nobody:*:9797:0:::::
sshd:!:14743:0:99999:7:::
apache:!:14761:0:99999:7:::
cron:!:14761:0:99999:7:::
kama:6E4Ird3GY$asfiohef9w412234:14761:0:99999:7:::
john:6sjfeog$adsgf4249w412234:14761:0:99999:7:::
kdm:!:15105::::::

**nome_utente:$algo$salt$hash_password: **
**modifica:min_age:max_age:preavviso: **
tempo_riserva:account_exp:riservato

/etc/group:

root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
disk:x:6:root,adm,haldaemon
wheel:x:10:root,poul
mail:x:12:mail
news:x:13:news
uucp:x:14:uucp,poul
audio:x:18:poul
cdrom:x:19:haldaemon,poul
poul:x:1003:
john:x:1012:poul

nome_gruppo:password:gid:lista utenti

/etc/gshadow:

root:x::root
bin:x::root,bin,daemon
daemon:x::root,bin,daemon
audio:::poul
cdrom:x::haldaemon,poul
poul:::

nome_gruppo:password:amministratori:membri

Creazione, modifica ed eliminazione di Utenti e Gruppi

Nome Comando	Descrizione
useradd (groupadd)	Aggiunge un nuovo utente (gruppo) al sistema
usermod (groupmod)	Modifica le informazioni relative ad un utente (gruppo)
userdel (groupdel)	Elimina un utente (gruppo) dal sistema
passwd (gpasswd)	Modifica le informazioni contenute in /etc/shadow (/etc/gshadow)
pwck (grpck)	Verifica la consistenza dei file /etc/passwd (/etc/group) e /etc/shadow (/etc/gshadow)
chsh, chfn, chage	Modificano rispettivamente le informazioni realtive a: shell di default, campo GCOS, scadenza password.

N.B.
Su Debian e distribuzioni derivate sono presenti anche i comandi adduser e addgroup. Sono abbastanza semplici e il loro utilizzo è vivamente consigliato.

Alcuni Esempi:

- `useradd -U -G john,audio -m poul`

  Aggunge un nuovo utente “`poul`”, crea la sua home (popolandola con i file in `/etc/skel`), crea un gruppo col suo stesso nome e lo imposta come gruppo primario e aggiunge l'utente ai gruppi secondari “`john`” e “`audio`”.

- `passwd poul`

  Imposta la password per l'utente “`poul`”.

- `usermod -a -G video poul`

  Aggiunge l'utente esistente “`poul`” al gruppo secondario “`video`”.

- `gpasswd -A poul,john lpd`

  Imposta la lista degli amministratori del gruppo “`lpd`”.

- `userdel -r john`

  Elimina l'utente “`john`” e tutta la sua home.

- `groupdel john`

  Elimina il gruppo “`john`”.

Cambiamento di identità

Nome Comando	Descrizione
su	Consente di diventare temporaneamente un altro utente.
sudo	Consente di eseguire un comando come un altro utente.
newgrp	Esegue il login con un altro gruppo.
sg	Esegue un comando con un altro gruppo.

su VS sudo

- **su** si usa per eseguire una shell o un qualsiasi altro comando/programma con un altro utente. Eccezion fatta per quando viene usato da root, **su** richiede sempre l'immissione della password dell'utente che si vuole utilizzare

- **sudo** invece è altamente configurabile per consentire solo ad *alcuni* utenti di eseguire solo *alcuni* comandi richiedendo: la password di root, la password dell'utente che lancia **sudo**, la password di un altro utente o nessuna password. Il suo file di configurazione è **/etc/sudoers** e non va mai editato direttamente, ma solo attraverso il comando **visudo**.

/etc/sudoers:

```
Cmnd_Alias     PIPPO = /usr/sbin/halt, /usr/sbin/reboot, /usr/sbin/shutdown

Defaults env_reset,rootpw,insults
Defaults:poul !rootpw

root ALL=(ALL) ALL
poul ALL=(root) PIPPO

%sudo ALL=(ALL) NOPASSWD: ALL
```

Altri comandi utili:

Nome Comando	Descrizione
id	Stampa UID e GID reali ed effettivi.
whoami	Stampa il nome utente effettivo.
w	Mostra una lista degli utenti loggati e cosa stanno facnedo.
groups	Stampa la lista dei gruppi correnti.
last	Mostra la lista degli ultimi login effettuari dall'utente.

PERMESSI UNIX-LIKE

- La gestione dei permessi è implementata a livello di File System, quindi i permessi *unix-like* possono essere utilizzati solo con FS che li supportano. Per esempio con *ext, ReiserFS, btrfs;* ma non con altri come *FAT32 o NTFS.*
- Ad ogni file sono associati un utente e un gruppo proprietari (normalmente referenziati come **proprietario** e **gruppo**).
- Si distinguono 3 tipi di accesso: **lettura**, **scrittura**, **esecuzione**. Che possono essere assegnati (dall'utente **proprietario** o da **root**) a tre categorie di utenti: **proprietario**, **gruppo**, **altri**.
- I permessi hanno un significato diverso a seconda che vengano assegnati ad un file o una directory. Nel caso delle directory forse i loro significati sono meno intuitivi ed è meglio esplicitarli:
  - **Lettura:** Possibilità di leggere la lista di file e dir contenute
  - **Scrittura:** Possibilità di creare, cancellare e rinominare file e dir contenute
  - **Esecuzione:** Possibilità di attraversare la dir
- Esiste, inoltre, una categoria di permessi *speciali* discussa in seguito.

Rappresentazioni dei permessi

- I tipi di rappresentazione largamente utilizzati sono 2: sotto forma di **stringa** e sotto forma di **numero in base 8**.

- La rappresentazione sotto forma di stringa si compone di 9 caratteri:
  `rwxr-xr-x`
  Il primo gruppo di 3 caratteri indica i permessi relativi al proprietario, il secondo quelli relativi al gruppo e l'ultimo quelli che riguardano tutti gli altri utenti. Il "`-`" è un *placeholder* per il permessi disabilitati.

- La rappresentazione sotto forma di numero ottale si compone di 4 cifre:
  `0755`
  La prima cifra (che può anche essere omessa se è 0) è utilizzata per i permessi *speciali* la seconda per i permessi relativi al proprietario e così via. Le singole cifre sono da intendersi come somma dei tre valori numerici assegnati a **lettura**, **scrittura** ed **esecuzione**; cioè, rispettivamente: **4**, **2**, **1**.

Permessi Speciali

- I permessi speciali sono 3 e sono:
  - **SUID** (*Set user identifier*): Un eseguibile su cui sia attivato questo bit, all'atto dell'esecuzione, avrà sempre come UID effettivo quello dell'utente proprietario del file e non quello dell'utente che ne ha richiesto l'esecuzione.
  - **SGID** (*Set group identifier*): Simile al precedente, ma riguarda il GID effettivo. Inoltre, se applicato ad una directory, fa sì che tutti i file creati in quella directory abbiano il GID identico a quello della dir stessa.
  - **Sticky** (*Save text image*): Utilizzato con un directory fa sì che i file in essa contenuti possano essere rinominati o cancellati solo ed esclusivamente dai rispettivi proprietari, impedendo quindi la cancellazione da parte di chi ha solo i diritti di scrittura sulla directory. Il caso di utilizzo con un eseguibile non verrà discusso poiché assolve ad una funzione ormai obsoleta e comunque mai implementata su linux.
- Come anticipato, la rappresentazione in forma numerica avviene utilizzando la prima cifra e assegnando ai permessi una numerazione analoga a quella usata per **lettura**, **scrittura** ed **esecuzione**.
- La rappresentazione in forma letterale, avviene sostituendo la "x" dei permessi di esecuzione con le lettere adeguate. Minuscole o maiuscole a seconda che sia attivo o meno anche il permesso di esecuzione:

```
rwsr-Sr-t
```

Comandi per gestire proprietari e permessi

Nome Comando	Descrizione
chown	Cambia il proprietario del file (solo root)
chmod	Cambia i permessi del file (solo root e proprietario)
umask	Imposta la maschera di permessi di default per i file creati in una detarminata directory

Alcuni esempi:

- `chown -R poul:john project/`
  Cambia il proprietario (`poul`) e il gruppo (`john`) della dir "`project`" e ricorsivamente a tutti i file le dir che contiene.

- `chmod -R ugo-w+x project/`
  Toglie i permessi di scrittura e aggiunge quelli di esecuzione a Utente, Gruppo e altri utenti; ricorsivamente alla dir "`project`" e a tutti i file e le dir che contiene.

- `chmod 4775 program`
  Imposta la i permessi a `4775` per il file "`program`"

- `umask 022` ; `umask u=rwx,g=rx,o=rx`
  Imposta la maschera di permessi di default per la dir corrente a `0755`

POSIX Capabilities

- I privilegi normalmente riservati a **root** vengono suddivisi in una serie di ***capability*** che possono essere abilitate o disabilitate singolarmente su ogni eseguibile o thread al fine di assegnargli solo i privilegi necessari al suo funzionamento.

- Una lista di tutte le capability disponibili, nonché un approfondimento sul loro funzionamento può essere trovato nel relativo manuale[1]

Nome Comando	Descrizione
setcap	Imposta le capability su uno o più file
getcap	Legge le capability di uno o più file

[1] `man 7 capabilities`

Sets

- Le ***capability*** che si possono assegnare ad un file/thread possono essere all'interno di 3 differenti ***set:***

Set	File	Thread
Permitted	Indica quali *CAP* debbano essere inserite automaticamente nel *set* **Permitted** del nuovo programma.	Stabilisce quali *CAP* possono essere inserite nel *set* **Effective** del thread
Inheritable	Stabilisce quali *CAP* possono essere ereditate da un programma dopo una *execve*	Stabilisce quali *CAP* possono essere ereditate da un programma dopo una *execve*
Effective	Non è un *set* ma un singolo bit che, se attivato, fa sì che le *CAP* **Permitted** di un nuovo programma vengano inserite anche nel *set* **Effective**	È l'insieme di *CAP* effettivamente usato dal *kernel* per eseguire i controlli

Alcuni esempi:

- `setcap cap_net_raw=ep /bin/ping`

  Dando all'esegubile *ping* l'abilità di usare i *raw socket* consente a qualsiasi utente di utilizzarlo senza bisogno che sia attivato su di esso il SUID bit.

- `setcap cap_chown,cap_dac_override,cap_fowner+ep\ /usr/bin/passwd`

  Attiva le capability necessarie per consentire a chiunque di cambiare la propria password anche se su *passwd* viene disabilitato il SUID bit.

RBAC
Role-based access control

- In un sistema **RBAC** l'obbiettivo è di garantire ad i singoli *utenti* solo ed esclusivamente i permessi strettamente necessari al loro funzionamento e nulla più. Anche in modo complesso e capillare.

- La gestione è semplificata poiché i vari **permessi** vengono assegnati a dei **ruoli**. Questi **ruoli** poi possono essere attribuiti a diversi **soggetti** (e.g. un gruppo di *utenti*).

- Alcuni sistemi consento di limitare i privilegi dello stesso utente *root* e creare set di **policy** che basino i loro vincoli non solo sull'*UID*, ma anche sul programma in esecuzione, e sugli *indirizzi IP* locali o remoti. Permettendo di ridurre al minimo i danni che un attaccante potrebbe arrecare, se riuscisse a sfruttare una vulnerabilità in un servizio.

Panoramica sugli RBAC più diffusi:

- **SELinux:** basato su un progetto originale dell'*NSA*, recentemente è stato inserito nei sorgenti ufficiali del kernel Linux. È supportato da molte distribuzioni e sono disponibili diversi file di policy adatti ai software più diffusi. Nonostante questo rimane molto complesso e di difficile utilizzo.

- **AppArmor:** è simile al precendente, ma nasce come alternativa più semplice da usare. Per esempio è dotato di una *learning mode* che consente di generare automaticamente le policy. Deve, probabilmente, parte della sua fortuna al fatto che è stato integrato in Ubuntu Linux.

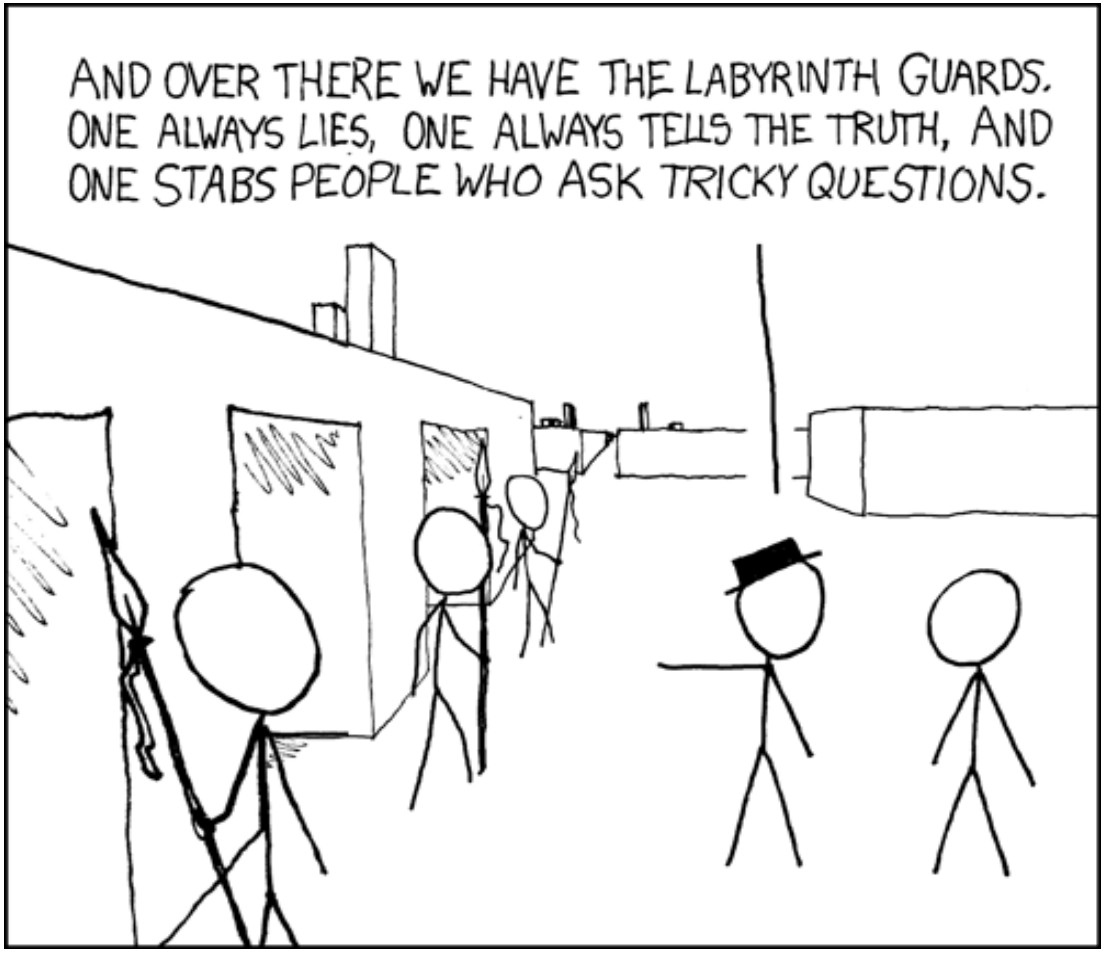
- **grsecurity:** oltre all'*RBAC* (che può funzionare anche in *learning mode*) include un ampio ventaglio di patch al codice del kernel per migliorarne la sicurezza sotto molti punti di vista (rendendo in molti casi inoffensive, o comunque difficilmente sfruttabili, la maggior parte delle vulnerabilità). Poichè però le modifiche sono molto estese ed invasive non è stato incluso nella mainline del kernel Linux.

Appunti di Informatica Libera

di Daniele Giacomini

http://www.informaticalibera.net/

Domande?



POuL

have a lot of fun...

http://www.poul.org

- **Mailing List**: http://www.poul.org/cgi-bin/mailman/listinfo/mailinglist
- **IRC**: **#polimi @ irc.azzurra.org**
- **Sede**: Venerdì 17:00-18:00