Controllo degli accessi in UNIX - parte I

Andrea Lanzi

30 marzo 2015

Controllo degli accessi nei sistemi UNIX

Linee guida generiche stabilite fin dalle prime versioni:

1. Tutti gli oggetti (es. file, processi) hanno dei proprietari. I proprietari hanno pieno controllo (*non necessariamente senza restrizioni!*) sui propri oggetti
2. un utente è il proprietario di tutti gli oggetti che crea
3. l'utente "root" può effettuare operazioni come se fosse il proprietario di un *qualsiasi oggetto*
4. solo all'utente "root" è concesso effettuare speciali operazioni di amministrazione del sistema

Non esiste un *unico* meccanismo di controllo degli accessi (Es. system call "stettimeofday" VS "kill").

Soggetti UNIX:

- utenti
- gruppi
- processi

Oggetti UNIX:

- file

Controllo degli accessi tramite file system

Ogni file ha associati un *proprietario* ed un *gruppo*.

Il proprietario può impostare, *a piacere*, i permessi di accesso a quello specifico file.

Per per stabilire a chi appartiene un file:

ls -l *filename*

```
[16:10:12]srdjan@salvadortmp:$ ls -l
total 16
-rw-r--r-- 1 bob     studente 1332 Apr  9 15:56 appunti.txt
drwx------ 2 srdjan  srdjan    4096 Jan  1  1970 orbit-srdjan
drwxrwxrwt 2 root    root      4096 Apr  9 15:19 VMwareDnD
drwx------ 2 root    root      4096 Apr  9 15:20 vmware-root
```

Controllo degli accessi tramite file system

Ogni file ha associati un *proprietario* ed un *gruppo*.

Il proprietario può impostare, *a piacere*, i permessi di accesso a quello specifico file.

Per per stabilire a chi appartiene un file:

ls -l *filename*

```
[16:10:12]srdjan@salvadortmp:$ ls -l
total 16
-rw-r--r-- 1 bob      studente 1332 Apr  9 15:56 appunti.txt
drwx------ 2 srdjan   srdjan   4096 Jan  1  1970 orbit-srdjan
drwxrwxrwt 2 root     root     4096 Apr  9 15:19 VMwareDnD
drwx------ 2 root     root     4096 Apr  9 15:20 vmware-root
```

Soggetti e oggetti: diritti

Soggetti

- diritti di accesso determinati da UID/GID

Oggetto

- diritti di accesso memorizzati insieme all'oggetto stesso (inode)

Permessi sul file system

Tipologie di permessi di accesso: lettura (`r` o `4`), scrittura (`w` o `2`), esecuzione (`x` o `1`).
Per i file:

- `r`: permette di leggere il file
- `w`: permette di scrivere il file
- `x`: permette l'esecuzione

Per le directory:

- `r`: permette di leggere il contenuto delle directory (elenco dei file)
- `w`: permette di modificare il contenuto delle directory
- `x`: permette l'attraversamento della directory

Processi e proprietario

- Il proprietario del processo può inviare segnali al processo stesso o modificare la priorità di scheduling del processo
- in `Unix` ogni processo possiede le proprie credenziali (UID, GID) che determinano cosa un processo può fare;
- l'utilizzo richiede supporto sia dal processo che dalla risorsa interessata;

Nome	**Descrizione**
`uid, gid`	real UID/GID
`euid, egid`	effective UID/GID
`suid, sgid`	saved UID/GID
`fsuid, fsgid`	effective file access UID/GID [Linux]

L'account di *root*

- È l'account amministratore che è *onnipotente* all'interno di un sistema UNIX (*superuser*)
  - può effettuare qualsiasi operazione *valida*
- root ha UID = 0

Esempi di operazioni privilegiate:

- modifica del *hostname*
- configurazione di interfacce di rete
- spegnimento del sistema
- modifica del proprio UID e/o GUID (Es. programma "*login*")

Permessi di accesso

- quando un processo è creato eredita la credenziali del padre;
- le credenziali possono essere modificate successivamente (e.g. system call);
- solitamente `uid, euid, suid` hanno lo stesso valore;

Cosa succede quando un processo esegue un programma `setuid`?

Permessi di accesso

`setuid`

Quando un processo esegue un programma `setuid` i campi `euid` e `fsuid` sono impostati con l'UID del proprietario del file.

`passwd`

- le password sono salvate in un file comune al quale non è possibile accedere da utente non privilegiato;
- per cambiare la password l'utente invoca il programma `/usr/bin/passwd` che ha il flag `setuid` impostato ed il cui proprietario è l'utente con UID = 0 (`root`);
- quando il processo *forkato* dalla *shell* esegue `passwd` le credenziali del processo `euid` e `fuid` sono impostate con valore pari al UID del proprietario del file, cioè 0;
- il processo può accedere al file perché quando il kernel controlla i permessi trova `fsuid` a 0;

Permessi di accesso

programmi `setuid`

Un utente maligno può sfruttare *bug* nel codice per effettuare *privilege escalation*.

`Unix` permette ai processi di acquisire il privilegio `setuid` solo quando necessario e rilasciarlo quando non se ne ha più bisogno.
system call: `getuid`, `geteuid`, `getresuid`, `setuid`, `seteuid`, `setresuid`.

Permessi di accesso
Basati su effective UID/GID: Algoritmo

- real UID/GID: ereditati dal genitore; riconducibili solitamente a UID/GID dell'utente che ha iniziato la sessione di lavoro associata al processo
- effective UID/GID: UID utilizzato quando viene effettuato il controllo dei privilegi del processo; solitamente coincide con RUID

1. se UID del soggetto corrisponde al proprietario del file, si fa riferimento ai bit di permesso relativi al proprietario; *altrimenti*
2. se un GID del soggetto corrisponde al gruppo del file si fa riferimento ai bit di permesso relativi al gruppo; *altrimenti*
3. si utilizzano i bit relativi a "world"

Permessi "speciali"

Per i file:

- **SUID**: associa all'eseguibile in esecuzione i permessi del proprietario del file
- **SGID**: associa all'eseguibile in esecuzione i permessi del gruppo di appartenenza

Per le directory:

- **sticky**: non consente di modificare un file a un utente diverso dal proprietario, neanche dagli utenti che hanno permesso di scrittura sulla directory
- **SGID**: fa in modo che i permessi dei file creati appartengano al gruppo definito SGID

Problema: passwd?

```
srdjan@salvador:~$ ls -l /usr/bin/passwd
-????????? 1 root root 43280 Apr 9 21:54 /usr/bin/passwd

srdjan@salvador:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 2.4K Apr 9 21:54 /etc/shadow
```

Indipendentemente da quali siano i permessi concessi sul eseguibile `passwd`, il programma deve modificare file (`/etc/shadow`) per i quali solamente l'utente `root`, il proprietario, ha il permesso di scrittura (`w`)!.

Soluzione: setuid

```
srdjan@salvador:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 43280 Apr 9 21:54 /usr/bin/passwd
srdjan@salvador:~$ cat /proc/13618/status
Name: passwd
State: S (sleeping)
Tgid: 13618
Pid: 13618
PPid: 13419
TracerPid: 0
Uid: 1000 0 0 0
Gid: 1000 1000 1000 1000
...
```

Funzionamento:

Quando un utente cambia la propria password usando il comando `passwd` il RUID corrisponde all'UID dell'utente che ha eseguito il programma mentre EUID corrisponde al proprietario del file, cioè `root`.

umask

- permessi di default impostati alla creazione di un file, secondo una maschera definita
- maschera di default impostata tramite il comando `umask`
- la maschera indica i permessi che **non** vengono attribuiti; e.g., `umask 022` $\Rightarrow$ **non** viene permessa la scrittura ne al gruppo ne ad altri utenti (permessi reali & umask)

Attributi (1)

Oltre i permessi è possibile definire alcuni attributi ai file che devono essere supportati dai file system (ext2 ext3 ext4).

Attributo	Descrizione
A	Non aggiorna la data di accesso (atime)
a	Permette l'aggiunta al file solo in append
c	Dice al kernel di compr. e decompr.
i[1]	immutabile, non permette nessuna modifica al file
s	cancella i dati con altri dati nulli.
u	Recupero file dalla cancellazione
S	ogni scrittura viene eseguita immediatamente.

[1] Invocabile solo da utente root

Attributi (2)

- `chattr [opzioni] [modalità] file...`
- si possono specificare gli attributi con i segni +=- + vengono aggiunti gli attributi specificati, con - vengono rimossi gli attributi specificati, = vengono impostati gli attributi cosi come specificati.
- `lsattr[opzioni] file...`

Opzione	Descrizione
-R	operazione ricorsiva
-a	Elenca tutti i file (anche nascosti)
-d	Elenca anche le directory

chown (1)

- `chown` comando per cambiare proprietario al file.
- `chown [opzioni] [utente][:[gruppo]] file`

–R	lavora in modo ricorsivo
–from=utente:gruppo	cambia i file del utente:gruppo

chown (2)

- **chown srdjan miofile** cambia l'utente del file miofile impostando l'utente `srdjan`
- **chown srdjan:users miofile** cambia l'utente e gruppo del file miofile rispettivamente nell'utente `srdjan` e gruppo `users`
- **chown -R –from=root srdjan miadir** cambia l'owner dei file appartenenti all'utente root nell'utente `srdjan`

chgrp (1)

- chgrp comando per cambiare gruppo al file.
- chgrp [opzioni] gruppo file. . .
- **chgrp users file** questo comando imposta il gruppo users associato al file.
- **chgrp -R users /home/srdjan** questo comando imposta in modo ricorsivo il gruppo users associati ai file contenuti nella directory e sottodirectory.

chmod (1)

- chmod comando per cambiare permessi.
- chmod [opzioni] modalità dei permessi file...

u	Utente proprietario del file
g	Gruppo proprietario
o	Utente diverso
a	Tutti gli utenti indifferentemente

chmod (2)

r	accesso lettura
w	permesso scrittura
x	permesso esecuzione
s	Riguarda file eseguibili e directory (SUID SGID)
t	file eseguibili e directory. STICKY bit

- **chmod -R go-rwx** /* Toglie al gruppo e agli altri utenti la possibilità di scrittura lettura ed esecuzione della directory e nelle directory successive.
- **chmod -R a+rx** /* Assegna a tutti gli utenti il permesso di lettura ed esecuzione partendo dalla directory e considerando le sottodirectory.

sudo VS su

Necessità di ottenere accesso con un altro utente: `sudo` (comando), `su` (login shell).

1. `sudo`: non richiede la condivisione della password;
   - `su`: si;
2. `sudo`: solitamente implica l'esecuzione di un comando;
   - `su`: no;
3. `sudo`: permette la definizione di determinati vincoli (`/etc/sudoers`);
   - `su`: no;
4. `sudo`: mantiene il log di tutti i comandi;
   - `su`: no;

sudo VS su

Necessità di ottenere accesso con un altro utente: `sudo` (comando), `su` (login shell).

1. `sudo`: non richiede la condivisione della password;
   - `su`: si;
2. `sudo`: solitamente implica l'esecuzione di un comando;
   - `su`: no;
3. `sudo`: permette la definizione di determinati vincoli (`/etc/sudoers`);
   - `su`: no;
4. `sudo`: mantiene il log di tutti i comandi;
   - `su`: no;

sudo VS su

Necessità di ottenere accesso con un altro utente: `sudo` (comando), `su` (login shell).

1. `sudo`: non richiede la condivisione della password;
   - `su`: si;
2. `sudo`: solitamente implica l'esecuzione di un comando;
   - `su`: no;
3. `sudo`: permette la definizione di determinati vincoli (`/etc/sudoers`);
   - `su`: no;
4. `sudo`: mantiene il log di tutti i comandi;
   - `su`: no;

sudo VS su

Necessità di ottenere accesso con un altro utente: `sudo` (comando), `su` (login shell).

1. `sudo`: non richiede la condivisione della password;
   - `su`: si;
2. `sudo`: solitamente implica l'esecuzione di un comando;
   - `su`: no;
3. `sudo`: permette la definizione di determinati vincoli (`/etc/sudoers`);
   - `su`: no;
4. `sudo`: mantiene il log di tutti i comandi;
   - `su`: no;

sudo VS su

Necessità di ottenere accesso con un altro utente: `sudo` (comando), `su` (login shell).

1. `sudo`: non richiede la condivisione della password;
   - `su`: si;
2. `sudo`: solitamente implica l'esecuzione di un comando;
   - `su`: no;
3. `sudo`: permette la definizione di determinati vincoli (`/etc/sudoers`);
   - `su`: no;
4. `sudo`: mantiene il log di tutti i comandi;
   - `su`: no;

Esercizio

0xA0

~~Creare l'utente *foobar*~~ *Utilizzando l'utente* `bob` *presente sul sistema*, impostare la sua *umask* in modo tale che possa creare file e directory leggibili e scrivibili solo da lui (i.e. negando qualsiasi altro permesso a *group* e *other*).

Modificare ora la `umask` dell'utente `bob` in modo che solo lui abbia il diritto di modificare e leggere i nuovi file che crea. Tale modifica deve persistere al riavvio del sistema.

Esercizio

0xA1

Permettere a *bob* di creare la directory `/usr/local/bob`

- **senza** aggiungerlo al gruppo *root*
- senza creare la directory come utente *root*
- senza cambiare proprietario/permessi della directory `/usr/local`

Tale modifica deve persistere ad eventuali reboot del sistema.

Esercizio

0xA2

Creare la directory */usr/local/bob/tmp* e permettere l'accesso, la scrittura e la lettura in tale directory a tutti (user, group, other). Fare in modo che soltanto gli utenti proprietari dei file creati nella directory abbiamo la possibilità di cancellarli.

Esercizio

0xA3

Creare la directory */home/bob/suid-files/* e cercare tutti i file *suid root* e copiarli nella directory appena creata impostando `bob` come proprietario e gruppo di appartenenza di ciascun file.

Esercizio

0xA4

Creare un file *appendMe* in */home/bob/* in modo tale che sia solamente possibile aggiungere dati e non eliminarne.

Esercizio

0xB1

- Scrivere un programma `C` che scriva su `stdout` le credenziali del processo relativo all'esecuzione del vostro programma.
- Si modifichi il proprietario del programma come `root`, `bob`, `alice` e si attribuiscano gli opportuni permessi in modo tale che tale programma sia eseguito con le credenziali dell'utente `root`, `bob`, `alice`.
- Eseguire il programma con diversi utenti e vedere come cambiano i valori di: RUID, EUID, SUID.

Esercizio

0xB2

- Individuare ID dell'utente `eve`.
- Creare un nuovo gruppo *malicious* e aggiungere l'utente `eve` al gruppo appena creato.
- Usando l'account di `eve`, creare un file testuale `/tmp/evil.txt`. Modificare i permessi di tale file per fare in modo che solamente i membri del gruppo *malicious* possano leggerne il contenuto.
- Scrivere un programma `C`, che utilizzi la system call `setsgid`, che verrà utilizzato per generare un programma che consenta all'utente `bob` di leggere il contenuto di `/tmp/evil.txt`.

Hint

Proprietario del file eseguibile: `root`; setuid (`root`) su file eseguibile generato.

Link e materiale utile

- Unix history (`http://www.bell-labs.com/history/unix/`)
- "Unix and Linux System Administration Handbook", Evi Nemeth - Garth Snyder - Trent R. Hein - Ben Whaley, ED. Pretience Hall, 4th ed.
- "The Linux Programming Interface", Michael Kerrisk, ED. no starch press
- "Linux System Administration", Tom Adelstein & Bill Lubanovic, ED. O'Reilly
- "Setuid Demistified", H. Chen, D. Wagner, and D. Dean, 2002.
- "Linux Command Line and Shell Scripting Bible", Richard Blum, ED. Whiley