Lezione 5
Esecuzione con privilegi elevati

Sviluppo di software sicuro (9 CFU), LM Informatica, A. A. 2021/2022
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Università di Modena e Reggio Emilia
http://weblab.ing.unimore.it/people/andreolini/didattica/sviluppo-software-sicuro

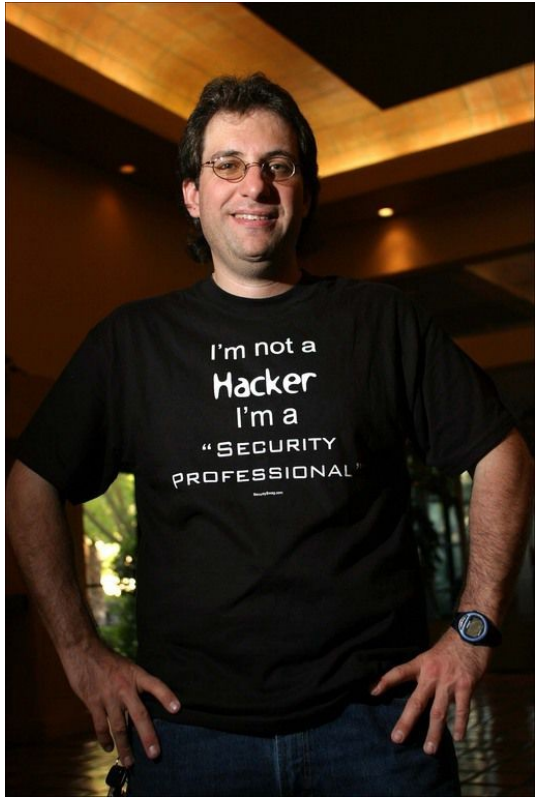
Quote of the day

(Meditate, gente, meditate...)

"The true computer hackers follow a certain set of ethics that forbids them to profit or cause harm from their activities."

Kevin David Mitnick (1963-)

Consulente di sicurezza, penetration tester, scrittore



Lo scenario

(Esecuzione di un comando con privilegi elevati)

Nei sistemi UNIX, può capitare che un processo debba eseguire con privilegi elevati (tipicamente, dell'utente amministratore, ovvero **`root`**).

- Lettura o modifica di un file di sistema altrimenti non accessibile (ad es., **`/etc/shadow`**).
- Interazione con una periferica tramite un file speciale di dispositivo (ad es., **`/dev/sda`**).
- Esecuzione di operazioni privilegiate su una periferica (ad es., invio di datagrammi tramite raw socket).

Elevazione dei privilegi

(Può essere effettuata in due modi: **manuale** oppure automatica)

Elevazione manuale. L'utente digita i comandi opportuni per diventare un amministratore. Dopodiché esegue il comando in questione.

```
$ su -
Password
# passwd andreoli
Changing password for user andreoli.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Problemi dell'elevazione manuale

(Che inducono l'utente a commettere nefandezze di ogni tipo)

Diffusione di credenziali. L'utente conosce la password dell'utente privilegiato. In tal modo, può compromettere le proprietà CIA.

L'utente può diventare root ed ispezionare il sistema, modificare file arbitrari, manomettere servizi.

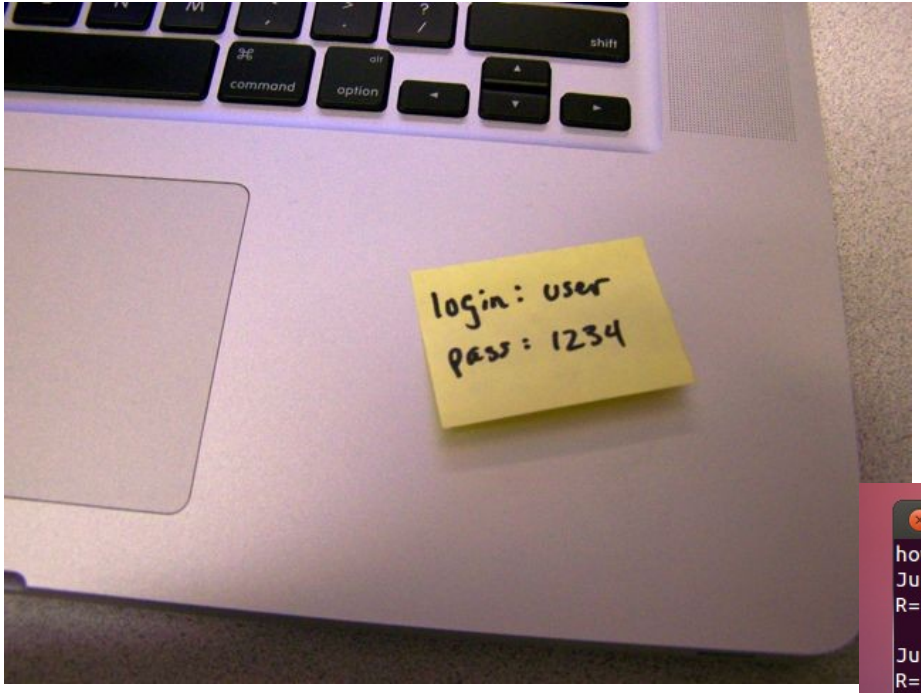
Scomodità d'uso. L'utente deve:

digitare il comando per l'elevazione dei privilegi;
immettere la password dell'utente privilegiato.

→ Alla lunga, l'utente medio si stufa e…

Alcune conseguenze

(Giudicate voi…)



howtogeek@ubuntu: ~

```
 GNU nano 2.2.6        File: /etc/sudoers.tmp              Modified

root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
howtogeek ALL=(ALL) NOPASSWD: ALL

# See sudoers(5) for more information on "#include" directives:


^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos
^X Exit     ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell
```

howtogeek@ubuntu: ~

```
howtogeek@ubuntu:~$ sudo cat /var/log/sudo
Jun 16 07:09:14 : howtogeek : TTY=pts/0 ; PWD=/home/howtogeek ; USE
R=root ;
    COMMAND=/bin/ping howtogeek.com
Jun 16 07:09:19 : howtogeek : TTY=pts/0 ; PWD=/home/howtogeek ; USE
R=root ;
    COMMAND=/usr/bin/apt-get install something
Jun 16 07:09:34 : howtogeek : TTY=pts/0 ; PWD=/home/howtogeek ; USE
R=root ;
    COMMAND=/bin/cat /var/log/sudo
howtogeek@ubuntu:~$
```

Orrore!

(Chiunque abbia accesso al terminale può diventare amministratore)

Elevazione dei privilegi

(Può essere effettuata in due modi: manuale oppure **automatica**)

Elevazione automatica. L'utente esegue il comando in questione. Il SO esegue l'elevazione dei privilegi all'utente opportuno (spesso, **`root`**).

```
$ passwd andreoli
Changing password for user andreoli.
Current password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Quale delle due elevazioni scegliere?

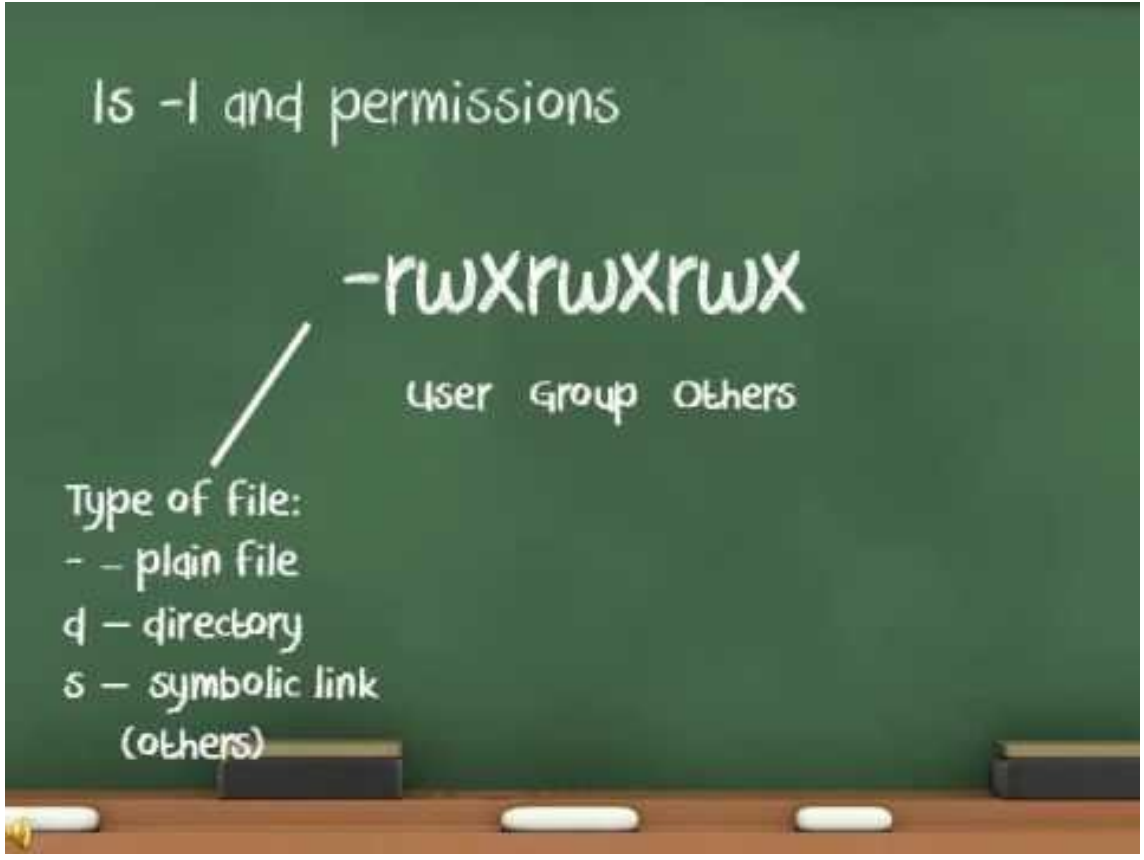
(Manuale o automatica?)

I sistemi UNIX moderni sono configurati per l'elevazione automatica dei privilegi di alcuni programmi di sistema.

Come avviene l'elevazione automatica dei privilegi nel caso più semplice?

Basic UNIX security review

(Un ripasso non fa mai male)



Utenti e gruppi

(Rappresentano persone reali ed automatiche, singole o in gruppo)

Un SO UNIX è organizzato per utenti e gruppi.

Utente: astrazione di un utente (umano oppure automatizzato).

Ha uno **username** (stringa) ed uno **user ID** (intero).

Gruppo: astrazione di un insieme di utenti.

Ha un **groupname** (stringa) ed un **group ID** (intero).

Un utente → più gruppi.

Più utenti → un gruppo.

Permessi di un file

(Regolano l'accesso ad un file da parte di utenti)

In UNIX, "tutto è un file".

File regolari, file speciali di dispositivo, socket, …

Permessi di accesso al file: tre terne di azioni per altrettanti gruppi di utenti.

Gruppi: creatore, gruppo di lavoro, resto del mondo.

Azioni: Read, Write, eXecute.

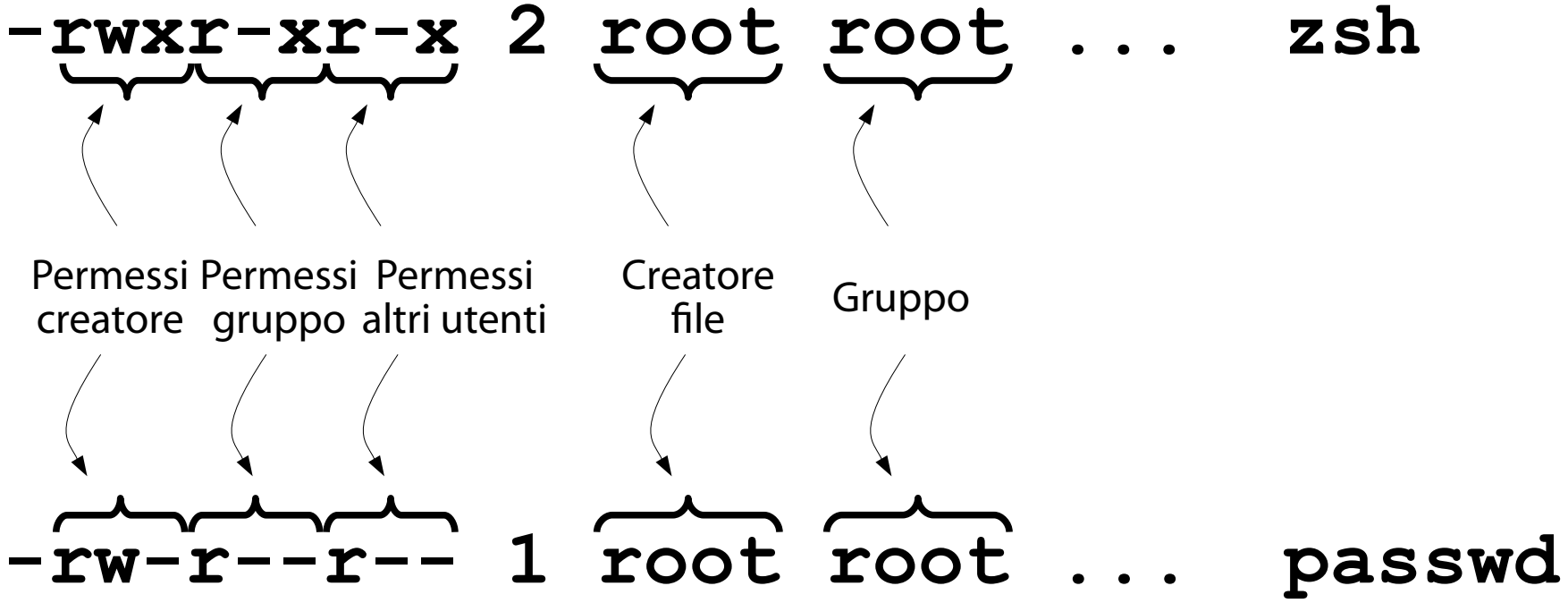
NOTA BENE:

execute su file → si può eseguire.

execute su directory → si può entrare nella directory.

Permessi tipici

(File eseguibile, file di testo)



Rappresentazione dei permessi

(Ottale o simbolica)

Rappresentazione **ottale**.

- Read → 4, Write → 2, eXecute → 1.
- Una terna di azioni → una somma di permessi.
- Permesso finale → tre numeri in ottale.
- rwxrwxr-x → 4+2+1 4+2+1 4+1 → 775 → **`0775`** (ottale)

Rappresentazione **simbolica**.

- Read → r, Write → w, eXecute → x.
- Creatore → u, Gruppo → g, Altri → o.
- Una modifica → utenti ± azioni (u+rwx)
- Permesso finale → modifiche separate da " , ".
- rwxrwxr-x → **`ug+rwx,o+rx`** (simbolica)

I bit di permessi SETUID/SETGID

(Molto rilevanti per la discussione)

In realtà esiste un altro permesso, rappresentato dal bit SETUID.

- Rappresentazione ottale: 4 (primo digit).
- Rappresentazione simbolica: s.

Analogamente, esiste il bit di permesso SETGID.

- Rappresentazione ottale: 2 (primo digit).
- Rappresentazione simbolica: s.

I privilegi di un processo in esecuzione

(Di solito, uguali a quelli dell'utente che ha lanciato il comando relativo)

Un processo in esecuzione assume le credenziali (user ID, group ID) di un utente.

In condizioni normali (bit SETUID e SETGID disattivati):

- User ID → user ID di chi ha lanciato il comando;
- Group ID → group ID primario di chi ha lanciato il comando.

I privilegi di un processo in esecuzione

(Con il SETUID bit attivo)

Se il bit SETUID è attivato:

User ID → user ID del creatore del file.

Se il bit SETGID è attivato:

Group ID → group ID primario del file.

→ È possibile cambiare la persona "effettiva" che esegue il file!

Utente e gruppo reale

(Contengono le credenziali della persona che ha lanciato il comando)

Nei SO UNIX, il descrittore di un processo memorizza una prima coppia di credenziali: **user ID reale** e **group ID reale**. Vale sempre la relazione seguente:

User ID reale → user ID di chi ha lanciato il comando;

Group ID reale → group ID primario di chi ha lanciato il comando.

Utente e gruppo effettivo

(Se SETUID/SETGID = 0, allora Reale → Effettivo)

Nei SO UNIX, il descrittore di un processo memorizza una seconda coppia di credenziali: **user ID effettivo** e **group ID effettivo**.

Se i bit SETUID e SETGID sono disattivati:

- User ID effettivo → user ID di chi ha lanciato il comando;
- Group ID effettivo → group ID primario di chi ha lanciato il comando.

Utente e gruppo effettivo

(Se SETUID/SETGID = 1, allora File → Effettivo)

Nei SO UNIX, il descrittore di un processo memorizza una seconda coppia di credenziali: **user ID effettivo** e **group ID effettivo**.

Se il bit SETUID è attivato:

User ID effettivo → user ID del creatore del file;

Se il bit SETGID è attivato:

Group ID effettivo → group ID del gruppo del file.

Algoritmo di controllo dei permessi

(Dovrebbe essere noto)

Le credenziali effettive del processo sono messe a confronto con i permessi di ogni elemento del percorso di un file.

`/home/andreoli/a.txt` → **`/`**, **`home/`**, **`andreoli/`**, **`a.txt`**

Se le credenziali sono sufficienti per ogni elemento del percorso → permesso di accesso accordato. Altrimenti → permesso negato.

L'utente **`root`** ha sempre accesso ai file.

Come riconoscere il bit SETUID/SETGID?

(Alcuni tratti visuali)

I file eseguibili con bit SETUID/SETGID:

- sono evidenziati in rosso (se SETUID) o in giallo (se SETGID) nell'output di **`ls -l`**;
- hanno il bit di permesso "s" nell'output di **`ls -l`**.

Ad es.:

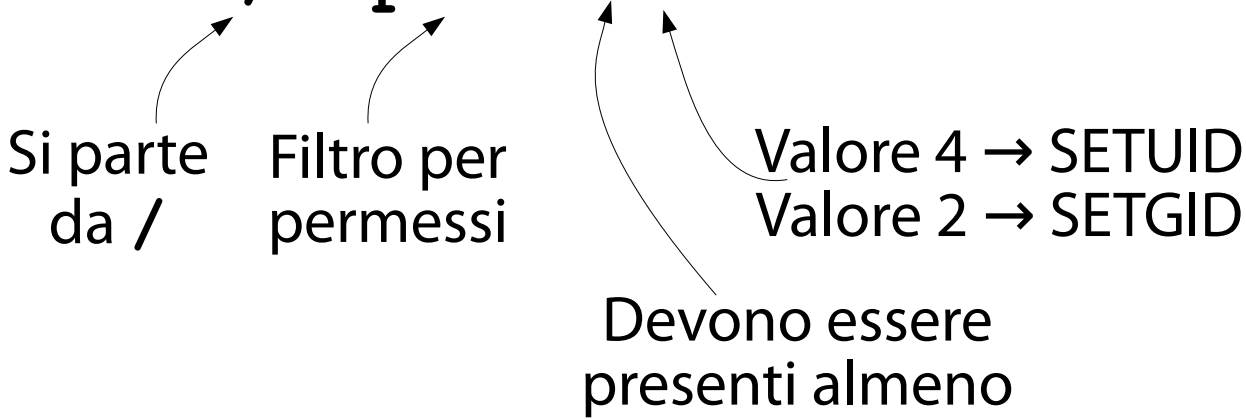
```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 52528 29 ott 17.54 /usr/bin/passwd
```

Individuazione sistematica

(Tramite il comando **`find`**)

Per individuare sistematicamente tutti i file SETUID e SETGID, è possibile usare il comando **`find`** con l'opzione **`-perm`** (filtro in base ai permessi):

```
find / -perm -6000
```



Una osservazione

(Don’t panic if you don’t find any file)

Se non si dovesse trovare un file, è possibile provare con i soli bit

SETUID (4000);

SETGID (2000).

Verifica dei privilegi

(Tramite il comando **ps**)

Per verificare i privilegi di esecuzione, si lanci un comando SETUID e si usi il comando **ps** per estrarre gli ID reali ed effettivi:

```
$ passwd
Changing password for andreoli.
Current password:
[su altro terminale]
$ ps -p $(pgrep -n passwd) -o ruid,rgid,euid,egid
 RUID   RGID   EUID   EGID
 1000   1000      0   1000
   andreoli     root  andreoli
```

PID del processo **passwd** più recente

Stampa gli ID reali ed effettivi

Esercizio

(Individuazione ed esecuzione di un programma SETUID)

Si avvii la macchina virtuale "Nebula".

Ci si autentichi come utente **`level00`**.

Si individui un file SETUID che esegua con le credenziali dell'utente **`flag00`**.

Si esegua il file.

Si verifichi che esegue con le credenziali dell'utente **`flag00`**.

Problemi dell'elevazione automatica

(L'eseguibile mantiene un privilegio enorme…)

Ottenimento di un privilegio enorme. Se non sono presenti altre contromisure, un processo SETUID/SETGID ottiene i pieni poteri dell'utente privilegiato.

Se l'utente privilegiato è **`root`**, il processo può fare di tutto.

Problemi dell'elevazione automatica

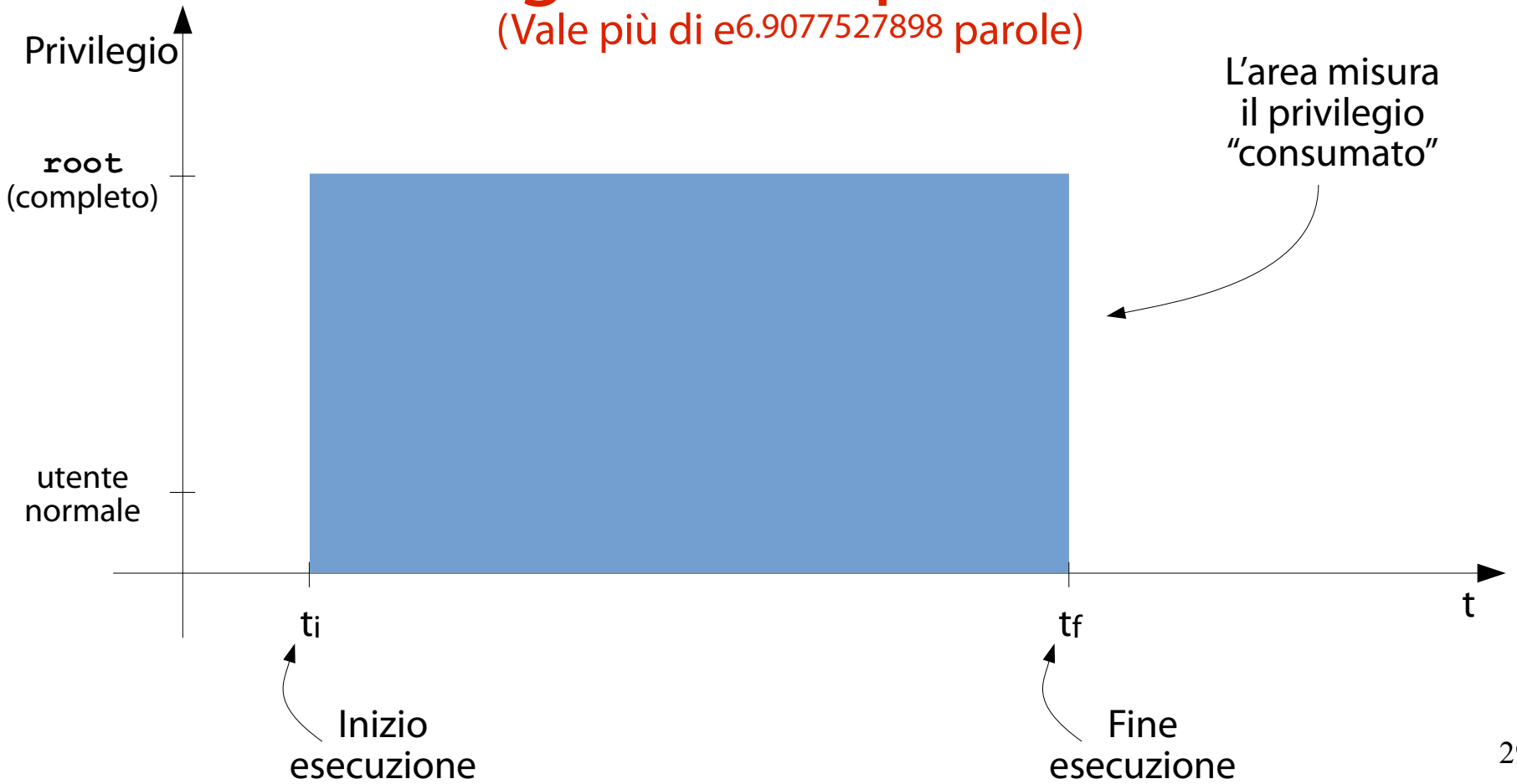
(… per l'intera esecuzione)

Ottenimento per l'intera esecuzione. Se non sono presenti altre contromisure, un processo SETUID/SETGID mantiene il privilegio elevato per l'intera esecuzione.

Una qualunque debolezza nel programma può essere attaccata per ottenere il privilegio (elevato).

Un grafico esplicativo

(Vale più di $e^{6.9077527898}$ parole)



Problemi o debolezze?

(Debolezze)

L'esecuzione automatica con diritti elevati è una funzionalità offerta dal SO UNIX agli utenti.

L'elevazione per tutta la durata dell'esecuzione (anche quando non strettamente necessaria allo svolgimento delle operazioni) è una vera e propria debolezza.

Il conferimento dei pieni poteri di `root` (laddove non sono strettamente necessari) è un'altra debolezza.

Debolezze o vulnerabilità?

(Debolezze, non sfruttabili da sole)

Si può parlare di vulnerabilità? No.

Se un programma è scritto in maniera corretta, la sola elevazione automatica dei privilegi non può dare alcun vantaggio ad un attaccante.

Tuttavia, se unita ad altre debolezze, l'elevazione automatica dei privilegi può avere conseguenze devastanti per la sicurezza di un asset.

Mitigazioni

(Le due più importanti)

Abbassamento e ripristino dei privilegi.

Destrutturazione dei diritti di **`root`**.

Abbassamento e ripristino dei privilegi

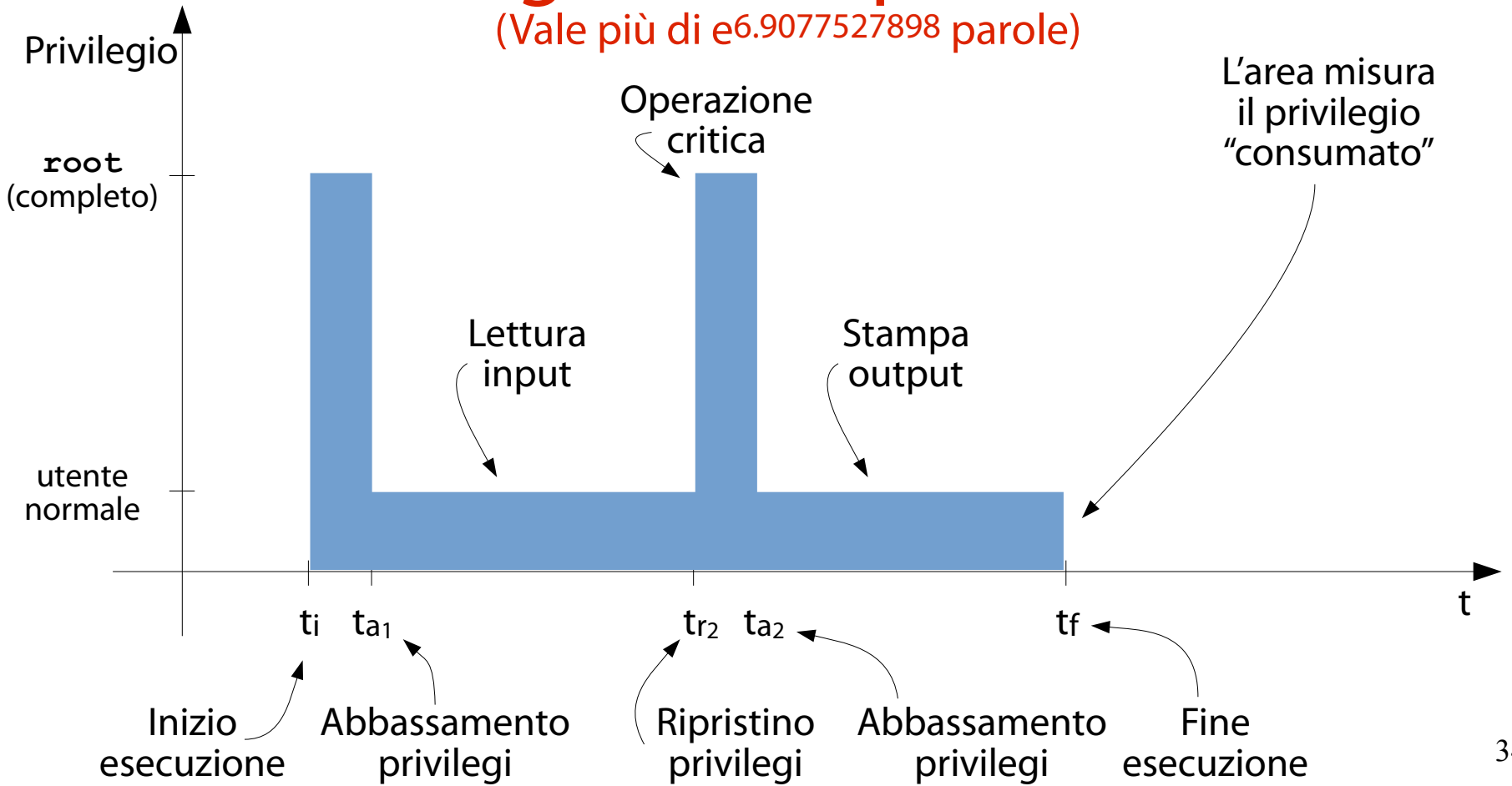
(Privilege drop and restore)

Se l'applicazione non svolge operazioni critiche, può decidere di abbassare i propri privilegi a quelli dell'utente che ha eseguito il comando (**privilege drop**).

Quando l'applicazione svolge operazioni critiche, ripristina nuovamente i privilegi ottenuti tramite l'elevazione automatica (**privilege restore**).

Un grafico esplicativo

(Vale più di $e^{6.9077527898}$ parole)



Alcune domande

(Doverose)

Come si implementano l'abbassamento ed il ripristino dei privilegi in un programma tramite bit SETUID/SETGID?

È disponibile una API?

È una API di sistema (chiamate di sistema, wrapper glibc)?

L'API di sistema è unica o si è evoluta nel tempo?

Una premessa

(Altrettanto doverosa)

Nel seguito della lezione alcuni degli esempi dovranno essere eseguiti nuovamente con privilegi elevati. Si eseguano le istruzioni seguenti.

Impostazione SETUID **`root`**:

```
chown root binario
chmod u+s binario
```

Impostazione SETGID **`root`**:

```
chgrp root binario
chmod g+s binario
```

I primissimi sistemi UNIX

(Ken Thompson and Dennis Ritchie hackin' on a PDP-11 back in the day, ~1970)

Gli user/group ID disponibili

(Solo due: reale ed effettivo)

Nei primissimi Sistemi Operativi UNIX sono previste due sole tipologie di user (o group) ID:

- user/group ID reale;
- user/group ID effettivo.

→ Le stesse tipologie viste nella discussione precedente sui bit SETUID/SETGID.

Recupero user ID reale ed effettivo

(Chiamate di sistema **`getuid()`**, **`geteuid()`**)

La chiamata di sistema **`getuid()`** ritorna l'ID reale del processo invocante.

`man 2 getuid` per tutti i dettagli.

`getuid_unix.c` per un esempio d'uso.

La chiamata di sistema **`geteuid()`** ritorna l'ID effettivo del processo invocante.

`man 2 geteuid` per tutti i dettagli.

`geteuid_unix.c` per un esempio d'uso.

Impostazione user ID effettivo

(Chiamata di sistema **`setuid()`**)

La chiamata di sistema **`setuid(uid)`** permette il cambio dell'ID effettivo di un processo al valore **`uid`**.

`man 2 setuid` per tutti i dettagli.

`setuid_unix.c` per un esempio d'uso.

Algoritmo di funzionamento:

ID effettivo = 0 → ID effettivo = ID reale = uid

ID effettivo ≠ 0 AND uid = ID reale → ID effettivo = uid

ID effettivo ≠ 0 AND uid ≠ID reale → ERRORE

getuid() → reale, setuid() → effettivo???

(Dr. McCoy, Admiral Kirk and Commander Spock are puzzled; teacher gone mad?)

Impostazione user ID effettivo

(Il docente non è mai stato più serio e lucido di così)

Non c'è stato alcun errore di battitura.

Le cose stanno proprio così.

`setuid()` → user ID EFFETTIVO.

Ad ora, cambiare lo user ID effettivo è l'unico modo per influenzare l'algoritmo di controllo dei permessi ed "indebolire", di fatto, l'utente effettivo.

Abbassamento dei privilegi

(Semplicissimo: da processo privilegiato, si esegue **`setuid(getuid())`**)

Per effettuare un abbassamento dei privilegi, è sufficiente eseguire lo statement seguente:

`setuid(getuid());`

`drop_priv_unix.c` per un esempio d'uso.

Cosa succede?

Recupero UID reale **`uid`**.

`uid` → UID effettivo.

`uid` → UID reale.

- Tramite **`getuid()`**.
- Tramite **`setuid()`**. Statement sempre eseguito.
- Tramite **`setuid()`**. Statement eseguito solo in presenza di un processo privilegiato.

Ripristino dei privilegi

(Funzionerà? Oppure fallirà miseramente?)

Domanda: è possibile ripristinare i privilegi elevati (ottenuti ad es. tramite esecuzione con SETUID bit impostato) dopo averli abbassati?

Si può provare ad eseguire lo statement:

```
setuid(uid_privilegiato);
```

nella speranza che il Sistema Operativo permetta nuovamente l'impostazione di uno user ID privilegiato precedentemente posseduto.

Programma di esempio: **`drop_rest_unix.c`**.

Un tentativo di esecuzione

(Questi comandi abilitano l'esecuzione SETUID **`root`**)

Si compili **`drop_rest_unix.c`**.

```
make
```

Si imposti il creatore a **`root`**.

```
chown root drop_rest_unix
```

Si imposti il SETUID bit.

```
chmod u+s drop_rest_unix
```

Si esegua **`drop_rest_unix`**.

```
./drop_rest_unix
```

L'esito del tentativo

(Infausto! `setuid(getuid());` non permette il ripristino)

L'abbassamento dei privilegi tramite lo statement **`setuid(getuid());`** non permette più il ripristino del privilegio elevato acquisito ad inizio esecuzione.

In altre parole, è possibile solamente abbassare i propri privilegi per sempre.

→ Piaga nei primi sistemi UNIX.

Come si può implementare il ripristino?

I sistemi UNIX System V (SVID)

(A DMD 5620 Blit terminal connected to a System V R3 host, ~1986)

Utente e gruppo salvato

(Contengono una copia degli ID effettivi corrispondenti)

Nei SO UNIX System V, il Sistema Operativo usa una terza coppia di credenziali: **user ID salvato** e **group ID salvato**.

Quando parte un processo, le credenziali "salvate" contengono una copia delle credenziali "effettive":

user ID effettivo → user ID salvato;

Group ID effettivo → group ID salvato.

Impostazione user ID effettivo

(Chiamata di sistema **`seteuid()`**)

Viene introdotta la chiamata di sistema **`seteuid(uid)`** che cambia l'ID effettivo di un processo al valore **`uid`**.

`man 2 seteuid` per tutti i dettagli.

`seteuid_sysv.c` per un esempio d'uso.

Algoritmo di funzionamento:

ID effettivo = 0 → ID effettivo = uid (per un qualsiasi uid)
ID effettivo ≠ 0 AND (uid = ID reale OR uid = ID salvato)
→ ID effettivo = uid
ID effettivo ≠ 0 AND NOT (uid = ID reale OR uid = ID salvato)
→ ERRORE

Una riflessione

(**`seteuid(root_uid);`** funziona se **`root_uid`** è l'EUID)

Se un processo:

parte SETUID **`root`**;

abbassa i suoi privilegi con **`seteuid(getuid());`**

invoca **`seteuid(0);`**

allora:

`seteuid(0);` termina correttamente (**`root`** è l'utente effettivo, permesso da **`seteuid()`**);

`seteuid(0);` imposta lo user ID effettivo al valore 0 (**`root`**).

→ Sembra un ripristino di privilegi!

Abbassamento dei privilegi

(Permanente → `setuid();`, Temporaneo → `seteuid();`

Abbassamento permanente dei privilegi:

```
setuid(getuid());
```

Abbassamento temporaneo dei privilegi:

```
seteuid(getuid());
```

Ripristino temporaneo dei privilegi

(Questo funziona, a patto che l'abbassamento sia fatto con **`seteuid()`**)

Nell'ipotesi che:

- un processo parta SETUID (tipicamente, **`root`**);
- abbassi i privilegi con **`seteuid()`**;

è possibile ripristinare i privilegi con la chiamata:

`seteuid(`*`privileged_id`*`);`

dove ***`privileged_id`*** è lo user ID dell'utente privilegiato ottenuto in partenza (tipicamente, 0 per l'utente **`root`**).

Un esempio concreto

(Programma **`drop_rest_sysv.c`**)

Il programma di esempio seguente implementa la strategia ora vista.

`drop_rest_sysv.c`.

Si compili ed esegua il programma da utente normali.

Abbassamento e ripristino da utente normale a utente normale funziona (Captain Obvious …).

Si imposti il SETUID bit e il creatore a **`root`**. Si esegua nuovamente il programma.

Il ripristino dei privilegi funziona!

Un limite di `seteuid()`

(Non è in grado di modificare lo user ID salvato)

A differenza di **`setuid()`** nei primissimi sistemi UNIX, **`seteuid()`** modifica solamente lo user ID effettivo.

- Non modifica quello reale.
- Non modifica quello salvato.

→ Mancanza di controllo diretto da parte dell'utente sullo user ID salvato (che appare come un valore "opaco").

I sistemi UNIX BSD

(The DEC VAX-11/780, a typical minicomputer hosting early BSDs, ~1978)

Impostazione user ID reale ed effettivo

(Chiamata di sistema **`setreuid()`**)

I primi Sistemi Operativi BSD (fino a 4.2BSD) sostituiscono la chiamata di sistema **`setuid(uid)`** con una nuova chiamata di sistema **`setreuid(uid, euid)`**.

`man 2 setreuid` per tutti i dettagli.

`setreuid_bsd.c` per un esempio d'uso.

Funzionamento di `setreuid()`

(Un po' più complicato rispetto alle varianti precedenti)

Parametri di **`setreuid(u, e)`**:

`u` → user ID reale;

`e` → user ID effettivo.

Algoritmo di funzionamento:

u/e = -1 → UID/EUID non viene modificato

UID effettivo = 0 → UID/EUID = u/e (qualunque)

UID effettivo ≠ 0 AND (e = ID reale OR e = ID effettivo)

→ ID effettivo = e

UID reale ≠ 0 AND (u = ID reale OR u = ID effettivo)

→ ID reale = u

Negli altri casi: ERRORE

Abbassamento permanente privilegi

(**`setreuid(uid, uid);`** **`uid`** è lo user ID di un utente non privilegiato)

L'abbassamento permanente dei privilegi si ottiene impostando gli user ID reale ed effettivo ad un valore non privilegiato:

```
setreuid(uid, uid);
```

In seguito, gli UID reale ed effettivo non potranno che assumere il valore **`uid`**.

→ Il ripristino a **`root`** (ID 0) è impossibile.

Esempio: **`drop_priv_bsd.c`**.

Abbassamento temporaneo privilegi

(**`setreuid(euid, uid);`** **`uid`** → reale, **`euid`** → effettivo)

L'abbassamento temporaneo dei privilegi si ottiene invocando **`setreuid()`** con gli user ID reale ed effettivo scambiati:

```
setreuid(geteuid(), getuid());
```

Dopo lo scambio:

- lo user ID reale è da utente privilegiato;
- lo user ID effettivo è da utente non privilegiato.

Ripristino temporaneo privilegi

(**`setreuid(euid, uid);`** **`uid`** → reale, **`euid`** → effettivo)

Il ripristino dei privilegi si ottiene invocando nuovamente **`setreuid()`** con gli user ID reale ed effettivo scambiati:

```
setreuid(geteuid(), getuid());
```

Dopo lo scambio:

- lo user ID reale è da utente non privilegiato;
- lo user ID effettivo è da utente privilegiato.

Un esempio concreto

(Abbassamento e ripristino con doppio scambio UID EUID)

Il programma di esempio **`drop_rest_bsd.c`** effettua l'abbassamento ed il ripristino dei privilegi tramite lo scambio degli ID reale ed effettivo.

Lo si esegua normalmente e, successivamente, SETUID **`root`**.

Domanda

(Che deve venire spontanea ai più attenti)

Nel processo che esegue **`drop_rest_bsd`**, subito dopo l'abbassamento dei privilegi

- lo user ID reale è 0 (**`root`**);
- lo user ID effettivo è ≠ 0 (utente normale).

È mai possibile che il processo non abbia alcun privilegio anche avendo UID reale = 0?

La risposta dovrebbe essere SÌ, poiché i privilegi sono controllati sugli ID effettivi.

- Eccezione: chiamata di sistema **`access()`**.

Risposta

(Fornita dal programma **`d_r_open_bsd.c`**)

L'esempio **`d_r_open_bsd`** prova ad aprire il file **`/etc/shadow`** (disponibile solo a **`root`**) in due occasioni:

- subito dopo l'abbassamento dei privilegi;
- subito dopo il ripristino dei privilegi.

Lo si compili e lo si esegua, normalmente e SETUID **`root`**.

In entrambi i casi, dopo l'abbassamento dei privilegi (anche con UID reale = 0) non è possibile leggere **`/etc/shadow`**!

Un limite di `setreuid()`

(Non gestisce lo user ID salvato)

Nei primi Sistemi Operativi BSD non sono presenti gli ID salvati. Pertanto, la chiamata di sistema **`setreuid()`** non li gestisce.

`setreuid()` supplisce a ciò con lo scambio degli ID reale ed effettivo, ma non è possibile impostare l'ID effettivo a quello di un altro utente specifico.

I sistemi POSIX

(RMS came up with the term, ~1985)

Lo standard POSIX

(Definisce un Sistema Operativo portabile su diverse architetture hardware)

POSIX (**Portable Operating System Interface**) è una famiglia di standard per la definizione di un Sistema Operativo portabile su diverse architetture hardware.

- API di sistema.
- Interfaccia da linea di comando.
- Applicazioni di base.
- Supporto alle applicazioni multi-threaded.
- Supporto alle applicazioni real-time.
- Supporto alla sicurezza.

Privilegio elevato ed appropriato

(Sono subdolamente diversi)

Lo standard POSIX distingue i due concetti di:

- privilegio elevato → pieni poteri di **`root`**;
- privilegio appropriato → una qualunque "capacità" richiesta da una funzione specifica.

Il privilegio appropriato può essere ottenuto e verificato con uno o più meccanismi propri del Sistema Operativo.

- Non esiste solo il SETUID **`root`**…

Standardizzazione di **`setuid()`**

(Tramite l'impostazione della macro **`_POSIX_SAVED_IDS`**)

La semantica della chiamata di sistema **`setuid()`** può essere impostata in funzione del valore della macro **`_POSIX_SAVED_IDS`**.

Se **`_POSIX_SAVED_IDS`** è definita:

`setuid()` gestisce gli user e group ID salvati.

Se **`_POSIX_SAVED_IDS`** non è definita:

`setuid()` non gestisce gli user e group ID salvati.

`setuid(uid)`

(`_POSIX_SAVED_IDS` definita)

Se il processo ha un privilegio appropriato:

ID effettivo = ID reale = UID salvato = uid

Se il processo non ha un privilegio appropriato:

IF uid = ID reale OR uid = ID salvato → ID effettivo = uid

Negli altri casi: ERRORE

OSS.: gli user ID reale e salvato non vengono modificati.

Esempio d'uso:

programma **`setuid_psi.c`**.

`setuid(uid)`

(`_POSIX_SAVED_IDS` non definita)

Se il processo ha un privilegio appropriato:

ID effettivo = ID reale = uid

Se il processo non ha un privilegio appropriato:

IF uid = ID reale → ID effettivo = uid

Negli altri casi: ERRORE

OSS.: lo user ID reale non è modificato.

Adozione dello standard POSIX

(Nei Sistemi Operativi moderni)

A partire da 4.3BSD, la chiamata di sistema **`setreuid()`** è stata deprecata in favore della definizione POSIX di **`setuid()`**.

- 4.3BSD: senza **`_POSIX_SAVED_IDS`**.
- 4.4BSD: con **`_POSIX_SAVED_IDS`**.

Tutti i Sistemi Operativi moderni implementano **`setuid()`** con **`_POSIX_SAVED_IDS`**.

- Ecco perché non esiste un esercizio per il caso precedente.

Abbassamento dei privilegi

(Permanente → `setuid();`, Temporaneo → `seteuid();`

Abbassamento permanente dei privilegi:

```
setuid(getuid());
```

Abbassamento temporaneo dei privilegi:

```
seteuid(getuid());
```

Ripristino temporaneo dei privilegi

(`seteuid(privileged_id);`)

Nell'ipotesi che:

- un processo parta SETUID (tipicamente, **`root`**);
- abbassi i privilegi con **`seteuid()`**;

è possibile ripristinare i privilegi con la chiamata:

`seteuid(`*`privileged_id`*`);`

dove ***`privileged_id`*** è lo user ID dell'utente privilegiato ottenuto in partenza (tipicamente, 0 per l'utente **`root`**).

Un limite dei sistemi POSIX

(Demanda allo specifico Sistema Operativo la gestione degli ID salvati)

Lo standard POSIX non specifica un meccanismo per la lettura/scrittura esplicita degli ID salvati.

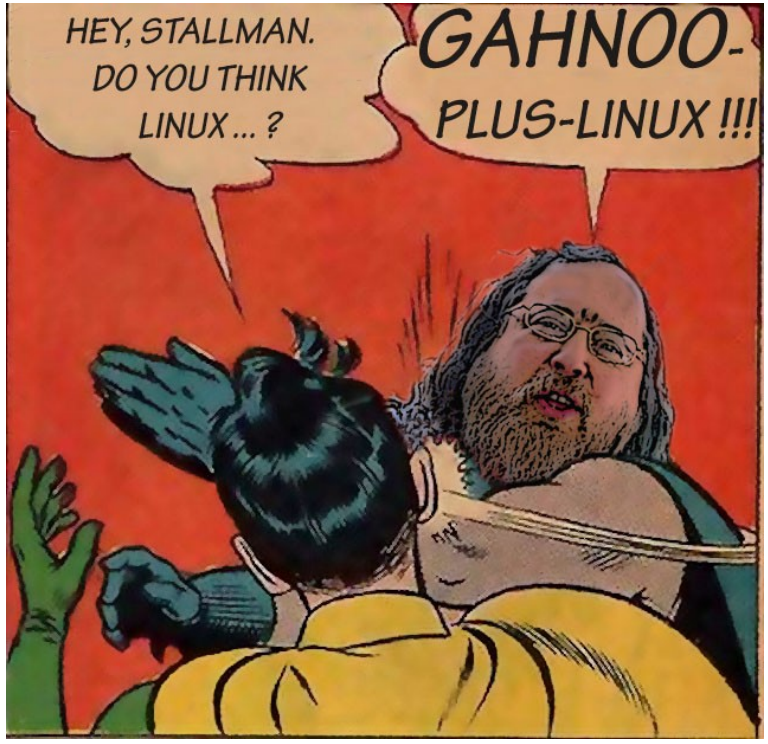
Demanda i dettagli di tale gestione allo specifico Sistema Operativo.

Ad esempio, in un sistema operativo GNU/Linux è possibile leggere lo user ID salvato con il comando seguente (**`PID`** è il PID del processo):

```
grep Uid /proc/PID/status | awk '{ print $4 }'
```

I sistemi UNIX moderni

(RMS and Linus defined and refined them, ~1991-today)



Sistemi Operativi GNU/Linux 1/2

(Come si relazionano con lo standard POSIX?)

I Sistemi Operativi GNU/Linux implementano la **Linux Standards Base** (**LSB**).

LSB è una evoluzione della **Single UNIX Specification** (**SUS**), l'evoluzione attuale dello standard POSIX.

Pur cercando di seguire lo standard POSIX il più possibile, il Sistema Operativo GNU/Linux non ne costituisce una implementazione 1:1.

Perché?

Sistemi Operativi GNU/Linux 2/2

(Come si relazionano con lo standard POSIX?)

Costo.

Certificarsi POSIX/SUS costa (molto) tempo e denaro.

Solo le distribuzioni commerciali lo fanno.

Difetti nello standard.

Lo standard SUS/POSIX presenta diversi difetti. Laddove le funzionalità POSIX/SUS siano giudicate funzionalmente sbagliate, non performanti, insicure,

GNU/Linux sopperisce con i propri meccanismi.

Esempio eclatante: **`setresuid()`** arricchisce e semplifica la semantica di **`setuid()`**.

Recupero degli user ID

(Chiamata di sistema **`getresuid()`**)

È possibile recuperare tutti gli tutti gli user ID del processo invocante tramite la chiamata di sistema **`getresuid()`**.

- Anche lo user ID salvato!

A differenza delle funzioni precedenti, si passano i puntatori degli user ID (che saranno popolati dal kernel).

- **`man 2 getresuid`** per tutti i dettagli.
- **`getresuid_gnulinux.c`** per un esempio d'uso.

Impostazione degli user ID 1/2

(Chiamata di sistema **`setresuid()`**)

È possibile anche impostare tutti gli user ID tramite la chiamata di sistema **`setresuid()`**.

Invocazione:

```
setresuid(u, e, s);
```

dove:

- **`u`** → user ID reale;
- **`e`** → user ID effettivo;
- **`s`** → user ID salvato.

Impostazione degli user ID 2/2

(Chiamata di sistema **`setresuid()`**)

Algoritmo di funzionamento:

u/e/s = -1 → UID/EUID/SUID non viene modificato
UID effettivo = 0 → UID/EUID/SUID = u/e/s (qualunque)
UID effettivo ≠ 0 AND (u = ID reale/effettivo/salvato)
→ ID reale = u
UID effettivo ≠ 0 AND (e = ID reale/effettivo/salvato)
→ ID effettivo = e
UID effettivo ≠ 0 AND (s = ID reale/effettivo/salvato)
→ ID salvato = s
Negli altri casi: ERRORE

Abbassamento permanente privilegi

(**`setresuid(uid, uid, uid);`** **`uid`** è l'UID di un utente non privilegiato)

L'abbassamento permanente dei privilegi si ottiene impostando tutti gli user ID ad un valore non privilegiato:

```
setresuid(uid, uid, uid);
```

In seguito, gli UID reale, effettivo e salvato non potranno che assumere il valore **`uid`**.

→ Il ripristino a **`root`** (ID 0) è impossibile.

Esempio: **`drop_priv_gnulinux.c`**.

Abbassamento temporaneo privilegi

(`setresuid(-1, uid, -1);` uid è l'UID di un utente non privilegiato)

L'abbassamento temporaneo dei privilegi si ottiene impostando lo user ID effettivo ad un valore non privilegiato:

```
setresuid(-1, getuid(), -1);
```

In questo modo si preserva lo user ID salvato e si può effettuare il ripristino in seguito.

NOTA BENE:

```
setresuid(-1, uid, -1) ≡ seteuid(uid);
```

Ripristino temporaneo privilegi

(**`setresuid(-1, uid, -1);`** uid è l'UID di un utente privilegiato)

Il ripristino temporaneo dei privilegi si ottiene impostando lo user ID effettivo ad un valore privilegiato:

```
setresuid(-1, privileged_ID, -1);
```

dove ***`privileged_id`*** è lo user ID dell'utente privilegiato ottenuto in partenza (tipicamente, 0 per l'utente **`root`**).

Un esempio concreto

(Abbassamento e ripristino tramite **`setresuid()`**)

Il programma di esempio seguente:

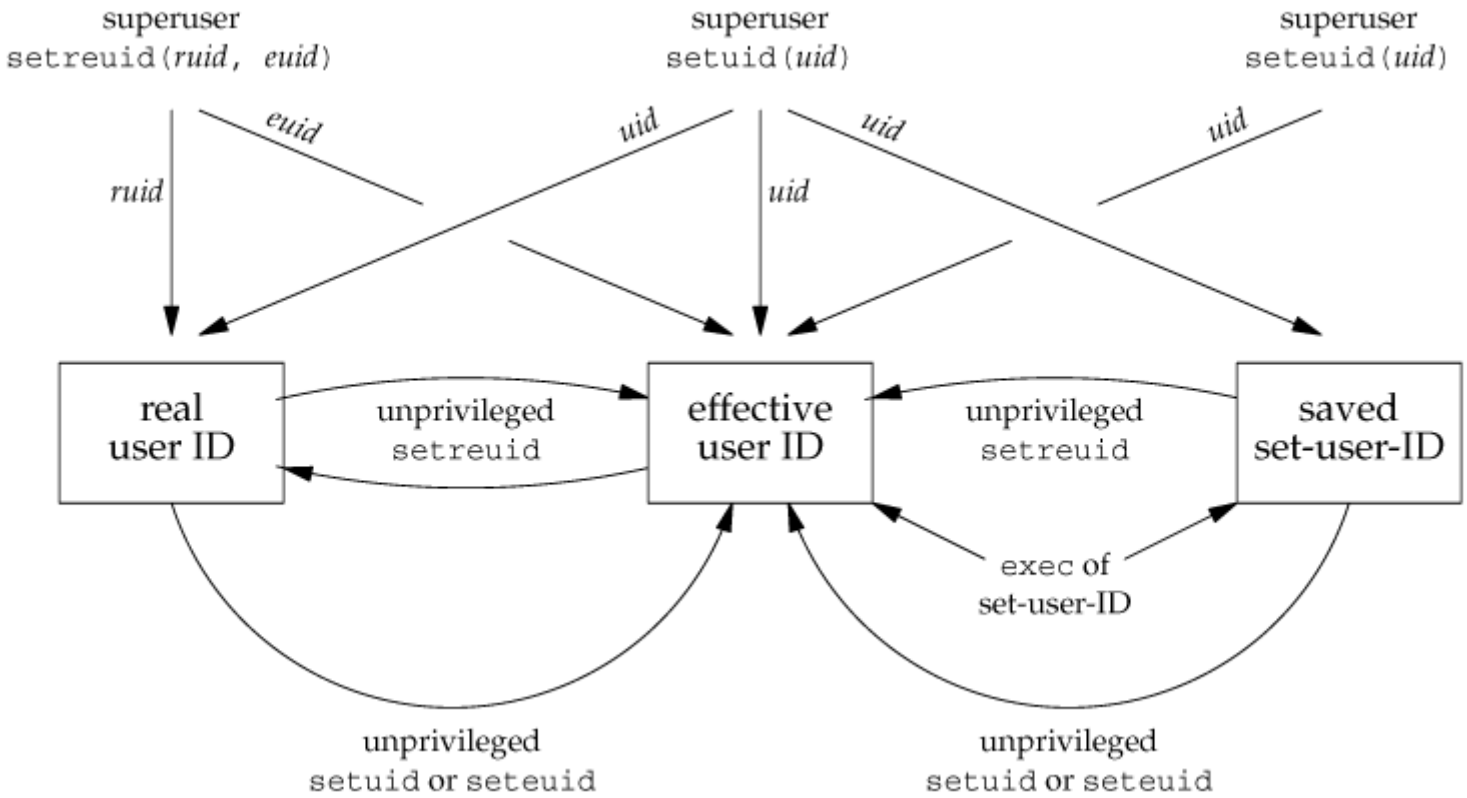
`drop_rest_gnulinux.c`

Implementa la strategia ora vista (emulazione di **`geteuid()`** tramite **`setresuid()`**).

Lo si esegua normalmente e, successivamente, SETUID **`root`**.

Riassumendo

(Un diagramma vale più di 1000 parole)



E l'elevazione dei privilegi via SETGID?

(Dov'è? Perché non è ancora stata presentata?)

La confessione del docente

("Posso andare, Signor Giudice?")

L'API di gestione dei privilegi tramite SETGID è identica concettualmente a quella di gestione dei privilegi tramite SETUID.

- Basta sostituire ovunque uid → gid.
- Una volta capita l'API SETUID, è capita anche l'API SETGID.

L'evoluzione concettuale delle funzioni ora viste è utile per capirne tutte le stranezze.

- Aggiungerci l'elevazione tramite bit SETGID non fa altro che distrarre lo studente dai concetti principali.

Recupero dei group ID

(Concettualmente identico al recupero degli user ID)

Le chiamate di sistema per il recupero dei group ID hanno semantica identica a quelle per il recupero degli user ID, e sono apparse negli stessi Sistemi Operativi.

`getgid()` ritorna il group ID reale del processo.

`getegid()` ritorna il group ID effettivo del processo.

`getresgid()` ritorna i group ID reali, effettivi e salvati del processo.

`man 2 get{,e,res}gid` per tutti i dettagli.

Impostazione dei group ID

(Concettualmente identico all'impostazione degli user ID)

Le chiamate di sistema per l'impostazione dei group ID hanno semantica identica a quelle per l'impostazione degli user ID, e sono apparse negli stessi Sistemi Operativi.

`setgid()` imposta il group ID effettivo del processo (se privilegiato, anche il reale e, se possibile, il salvato).

`setegid()` imposta il group ID effettivo.

`setregid()` imposta group ID reale ed effettivo.

`setresgid()` imposta i group ID reale, effettivo e salvato.

`man 2 set{,e,re,res}gid` per tutti i dettagli.

La logica conseguenza

(Quasi tutte le demo viste prima sono banalmente adattabili)

Poiché la semantica è identica, ci si potrebbe aspettare che tutte le demo viste in precedenza siano banalmente adattabili.

`uid` → `gid`

`chown root` → `chgrp root`

`chmod u+s` → `chmod g+s`

In realtà quasi tutte le demo sono adattabili banalmente.

Purtroppo, una non lo è.

Come “quasi”??? Perché non “tutte”?

(Whatcha talkin’about, Willis?)

I sistemi UNIX che non danno problemi

(**System V**, BSD, POSIX, GNU)

Demo funzionanti:

`drop_rest_sysv.c`

`setegid_sysv.c`

I sistemi UNIX che non danno problemi

(System V, **BSD**, POSIX, GNU)

Demo funzionanti:

- **`setregid_bsd.c`**
- **`drop_priv_bsd.c`**
- **`drop_rest_bsd.c`**
- **`d_r_open_bsd.c`**

I sistemi UNIX che non danno problemi

(System V, BSD, **POSIX**, GNU)

Demo funzionanti:

`setgid_psi.c`

I sistemi UNIX che non danno problemi

(System V, BSD, POSIX, **GNU**)

Demo funzionanti:

- **`getresgid_gnulinux.c`**
- **`drop_priv_gnulinux.c`**
- **`drop_rest_gnulinux.c`**

Il sistema UNIX che dà problemi
(Il primo UNIX)

Demo funzionanti:

- **`getgid_unix.c`**
- **`getegid_unix.c`**
- **`setgid_unix.c`**
- **`drop_priv_unix.c`**

Demo non funzionanti:

- **`drop_rest_unix.c`**

Che cosa ci si aspetta

(Che **`setgid(getgid());`** abbassi permanentemente i privilegi di gruppo)

Poiché si è detto che **`setuid()`** e **`setgid()`** hanno la stessa semantica, ci si aspetta che il seguente statement:

```
setgid(getgid());
```

esegua un abbassamento permanente dei privilegi nel processo che lo esegue.

Che cosa succede in realtà

(`setgid(getgid());` non abbassa permanentemente i privilegi…)

Purtroppo, eseguendo **`drop_rest_unix`** ci si accorge di una tragica realtà.

Lo statement **`setgid(getgid());`** non abbassa permanentemente i privilegi di gruppo.

Li abbassa solo temporaneamente.

Perché?

È necessaria una analisi più approfondita.

Prima dell'abbassamento dei privilegi

(Tutto sembra OK)

Il processo parte SETGID a **`root`**. I suoi GID reale ed effettivo sono stampati qui sotto.

```
Prima di abbassamento: GID reale del processo = 1000
Prima di abbassamento: GID effettivo del processo = 0
```

I valori sono coerenti con un processo SETGID **`root`**.

- GID reale → quello dell'utente normale.
- GID effettivo → **`root`** (SETGID **`root`**).

Dopo l'abbassamento dei privilegi

(Tutto sembra OK)

Dopo l'abbassamento dei privilegi i GID sono i seguenti.

```
Dopo abbassamento: GID reale del processo = 1000
Dopo abbassamento: GID effettivo del processo = 1000
```

Il GID effettivo è modificato correttamente.

Dopo il ripristino dei privilegi

(Qualcosa è andato storto)

Dopo il ripristino dei privilegi i GID sono i seguenti.

```
Dopo ripristino: GID reale del processo = 1000
Dopo ripristino: GID effettivo del processo = 0
```

Il GID effettivo è ripristinato al valore privilegiato.

Ci si aspettava il diniego del ripristino da parte del Sistema Operativo.

Questo è un effetto inatteso.

È necessario indagare ulteriormente.

Una nuova pista

(Per le indagini)

Finora sono stati stampati i soli GID reale ed effettivo.

Non è stato stampato il GID salvato.

Se il GID salvato è privilegiato, il ripristino ha sempre successo.

Bisogna stampare anche il GID salvato e capire che valori assume.

Stampa del GID salvato

(Cercando di non influenzare la semantica di **`setgid()`**)

Per stampare il GID salvato su GNU/Linux, si ricorre al comando visto in precedenza.

```
grep Gid /proc/PID/status | awk '{ print $4 }'
```

Il programma **`drop_rest_unix_dbg.c`** include una **`sleep(1000);`** subito prima dell'abbassamento dei privilegi (in modo tale da permettere all'utente di stampare il GID salvato con calma).

Si compili il programma, lo si imposti SETGID **`root`** e lo si esegua.

Domanda

(Perché non usare **`getresgid()`**?)

Perché non si usa **`getresgid()`**, che è molto più semplice?

Perché bisognerebbe compilare il programma con le estensioni GNU (**`-D_GNU_SOURCE`**).

In tal modo potrebbe essere usata la semantica moderna **`getgid()`**.

- Probabilmente GNU/Linux la adotta comunque…

Invece, così si può compilare ANSI.

- Non è detto che basti per preservare la semantica antica di **`getgid()`**; si è fatto il possibile…

Prima dell'abbassamento dei privilegi

(Tutto sembra OK)

Il processo parte SETGID a **`root`**. I suoi GID sono stampati qui sotto.

- Gid reale → 1000
- Gid effettivo → 0
- Gid salvato → 0

I valori sono coerenti con un processo SETGID **`root`**.

- GID reale → quello dell'utente normale.
- GID effettivo → **`root`** (SETGID **`root`**).
- GID salvato → **`root`** (copia di GID effettivo).

Stampa del GID salvato

(Dopo l'abbassamento dei privilegi)

Per stampare il GID salvato su GNU/Linux, si ricorre al comando visto in precedenza.

```
grep Gid /proc/PID/status | awk '{ print $4 }'
```

Il programma **`drop_rest_unix_dbg2.c`** include una **`sleep(1000);`** subito dopo l'abbassamento dei privilegi (in modo tale da permettere all'utente di stampare il GID salvato con calma).

Si compili il programma, lo si imposti SETGID **`root`** e lo si esegua.

Stampa del GID salvato

(Dopo l'abbassamento dei privilegi)

È possibile (e più semplice) ricorrere a **`ps`**:

```
ps -o ruid,rgid,euid,egid,suid,sgid
   -p $(pgrep -n drop_rest_unix_dbg2)
```

L'attributo **`suid`** stampa l'UID salvato.

L'attributo **`sgid`** stampa il GID salvato.

Dopo l'abbassamento dei privilegi

("Houston, we've had a problem here!")

Dopo l'abbassamento dei privilegi i GID sono i seguenti.

- Gid reale → 1000
- Gid effettivo → 1000
- Gid salvato → 0

Il GID effettivo è impostato correttamente.

Il GID salvato rimane invece a 0 (**`root`**).

→ Problema: ci si aspettava il valore 1000, come da documentazione.

Le conseguenze dell'anomalia

(Il ripristino riesce)

Le conseguenze dell'anomalia sono evidenti.

Poiché il processo non è privilegiato, **`setgid()`** permette l'impostazione del GID a:

group ID reale o group ID salvato.

Purtroppo il group ID salvato è 0 (grazie al mancato aggiornamento).

→ Pertanto, **`setgid(0);`** è perfettamente lecito.

Gid reale → 1000

Gid effettivo → 0

Gid salvato → 0

Cosa dice la documentazione?

(Dice una cosa apparentemente contraddittoria con l'esito dell'esperimento)

"If the calling process is privileged, the real GID and saved GID are also set."

"Error EPERM: The calling process is not privileged and gid does not match the real group ID or saved group ID of the calling process."

Questo dice la documentazione.

Cosa se ne deduce? 1/2

(È molto difficile emulare uno UNIX primordiale su GNU/Linux)

GNU/Linux implementa la versione POSIX di **`setgid()`** a livello di kernel.

Si parla di “privilegi appropriati” (does this ring any bells to you?).

`setgid()` gestisce anche il group ID salvato.

→ È impossibile fornire una emulazione precisa di uno UNIX primordiale.

E, infatti, la demo che funziona su un sistema UNIX primordiale fallisce su GNU/Linux.

Cosa se ne deduce? 2/2

(Il processo non sembra essere privilegiato)

Se il processo è privilegiato, **`setgid()`** imposta anche gli ID reale e salvato.

L'evidenza sperimentale dice che l'ID salvato non è, invece, impostato.

→ Se ne deduce che il processo non è privilegiato nel senso descritto dalla documentazione.

Il problema

(Anche se SETGID **`root`**, **`drop_rest_unix_dbg`** non è privilegiato!)

La parola "privilegiato" nella documentazione si riferisce ad uno user ID effettivo pari a 0 (**`root`**).

Il programma **`drop_rest_unix_dbg2`** NON è privilegiato in tal senso.

- Non è eseguito dall'utente **`root`**.
- Non è SETUID **`root`** (è solo SETGID **`root`**).

Se il programma non è privilegiato, **`setgid()`** non aggiorna lo user ID salvato! → Problema!

Un (timido) tentativo di soluzione

(Si imposta SETUID **`root`** il programma)

Si può provare allora a rendere privilegiato il programma (ad es., SETUID **`root`**).

Per fare impostare a **`setgid()`** il group ID salvato.

`chown root:root drop_rest_unix_dbg2`

Lo si continua a mantenere SETGID **`root`**.

Per valutare l'abbassamento e ripristino dei privilegi di gruppo.

`chmod ug+s drop_rest_unix_dbg2`

Si esegua: **`./drop_rest_unix_dbg2`**

Il risultato

(Un lieve miglioramento; il group ID salvato è impostato correttamente)

Dopo l'abbassamento dei privilegi i GID sono i seguenti.

Gid reale → 1000
Gid effettivo → 1000
Gid salvato → 1000

Il group ID salvato è impostato correttamente (1000).

Fallirà il ripristino?

Un ulteriore programma di esempio

(drop_rest_unix_dbg3)

Il programma **drop_rest_unix_dbg3** rimuove lo statement **sleep(1000);** per verificare il ripristino.

Lo si imposti SETUID e SETGID **root**.

```
chown root:root drop_rest_unix_dbg3
chmod ug+s drop_rest_unix_dbg3
```

Si esegua il programma.

```
./drop_rest_unix_dbg3
```

Il risultato

(Nessun miglioramento; il ripristino va a buon fine)

Dopo il ripristino dei privilegi i GID sono i seguenti.

- Gid reale → 0
- Gid effettivo → 0
- Gid salvato → 0

Il ripristino non fallisce ancora.

Il problema

(Poiché **`drop_rest_unix_dbg3`** è SETUID **`root`**, l'ultima **`setgid();`** è OK)

Poiché il programma **`drop_rest_unix_dbg3`** è SETUID **`root`**, l'ultima chiamata **`setgid();`** può impostare un ID privilegiato (0).

→ Il ripristino ha successo.

→ Indipendentemente dalla modifica del group ID salvato.

Un altro tentativo di soluzione

(Si abbassa il privilegio SETUID prima di ripristinare il privilegio SETGID)

Si può provare allora ad abbassare i privilegi SETUID subito prima del ripristino dei privilegi SETGID.

In tal modo, l'ultima **`setgid();`** non è più in grado di impostare un GID privilegiato.

Un altro programma di esempio

(L'ennesimo: **`drop_rest_unix_dbg4`**)

Il programma **`drop_rest_unix_dbg4`** implementa la strategia ora vista.

Lo si imposti SETUID e SETGID **`root`**.

```
chown root:root drop_rest_unix_dbg4
chmod ug+s drop_rest_unix_dbg4
```

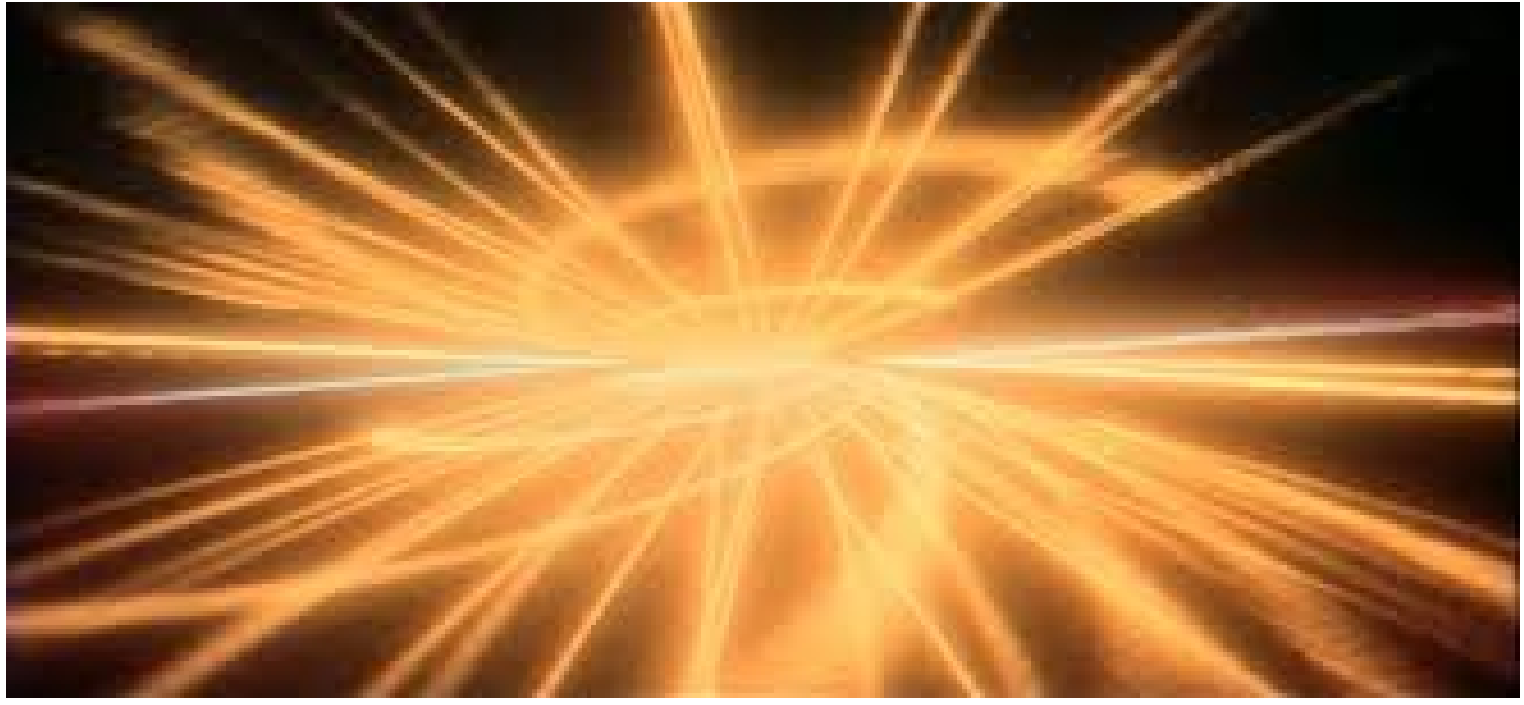
Si esegua il programma.

```
./drop_rest_unix_dbg4
```

Il ripristino di privilegio, finalmente, fallisce!

Amazing, eh?

(This is just the beginning)



So this is how it feels

(When you have effective `root`)

Inibizione del meccanismo SET[UG]ID

(Impedire l'esecuzione di binari con pieni poteri di `root`)

Il meccanismo di elevazione dei privilegi basato su SETUID/SETGID fornisce i pieni poteri di **`root`**.

Per tale motivo si preferisce inibirlo il più possibile.

Obiettivo: impedire l'esecuzione di shell (o altri programmi che si comportano come shell) con privilegi di **`root`**.

Opzione di mount **`nosuid`**

(Inibisce l'elevazione dei privilegi SETUID/SETGID su alcuni file system)

Alcuni file system sono montati con l'opzione **`nosuid`**. In essi, l'elevazione via SETUID/SETGID è inibita. Per individuare tali file system, si digiti:

```
mount | grep nosuid
```

Tutti i file system "virtuali" e temporanei impediscono l'esecuzione di programmi SETUID o SETGID.

Un esempio concreto

(Copia di un binario SETUID **root** in **/tmp** e successiva esecuzione)

Ad esempio, si copi **drop_rest_gnulinux** in **/tmp**.

```
cp drop_rest_gnulinux /tmp
```

Lo si renda SETUID **root**.

```
chown root /tmp/drop_rest_gnulinux
chmod u+s /tmp/drop_rest_gnulinux
```

Lo si esegua dalla **/tmp**.

```
/tmp/drop_rest_gnulinux
```

Il risultato dell'esperimento

(L'esecuzione è identica a quella in assenza di SETUID **`root`**)

Il programma **`drop_rest_gnulinux`** stampa gli user ID dell'utente normale che lo ha lanciato.

Non vi è traccia di ID privilegiati.

Ciò implica che il meccanismo SETUID sia stato inibito.

Abbassamento privilegi di BASH

(Impedisce l'esecuzione di script con privilegi elevati)

Di default, molte shell "moderne" (ad es., BASH) non onorano i bit SETUID/SETGID.

Né nei loro binari (**`/bin/bash`**).

Né negli script eseguiti (**`~andreoli/script.sh`**).

Nello specifico, non appena esegue BASH abbassa i propri privilegi effettivi (UID, GID) a quelli dell'utente che ha invocato la shell.

Motivazione dell'abbassamento

(Si impediscono delle falle di sicurezza mostruose)

Lasciare una BASH SETUID **`root`** è da criminali.

Esistono diversi modi di provocare esecuzione arbitraria di codice.

Tenere traccia di tutti questi attacchi è un compito arduo, se non impossibile.

Un esempio su tutti

(Da galleria degli orrori)

Si supponga di avere installato l'eseguibile **`shar`** per la creazione di archivi autoestranti.

Impostando il separatore degli input (variabile di ambiente **`IFS`**) al valore **`a`** è possibile far fare alla lettera 'a' le veci del carattere '**`<SPAZIO>`**' (separatore degli input).

→ **`shar`** è interpretato come **`sh r`**.

Se **`sh`** è una shell SETUID **`root`** e/o **`r`** è uno script SETUID **`root`** scrivibile dall'attaccante, permettere l'esecuzione privilegiata comporta l'esecuzione di codice arbitrario con i diritti di **`root`**.

Un esempio concreto

(Esecuzione di una BASH SETUID e lettura degli ID reale ed effettivo)

Ad esempio, si copi **`/bin/bash`** in una directory non montata **`nosuid`**.

Va bene una sottodirectory di **`$HOME`**.

```
cp /bin/bash $HOME
```

La si renda SETUID **`root`**.

```
chown root ./bash
chmod u+s ./bash
```

La si esegua e si stampino gli ID reale ed effettivo.

```
$HOME/bash
ps -p $$ -o ruid,rgid,euid,egid
```

Il risultato dell'esperimento

(L'esecuzione è identica a quella in assenza di SETUID **`root`**)

Il programma **`bash`** stampa gli user e group ID reale ed effettivo dell'utente normale che lo ha lanciato.

Non vi è traccia di ID privilegiati.

Ciò implica l'abbassamento dei privilegi da parte di **`bash`**.

Un altro esempio concreto

(Esecuzione di o script BASH SETUID e lettura degli ID reale ed effettivo)

Ad esempio, si crei uno script **scr.sh** con il seguente contenuto:

```
#!/bin/bash
id -u      ← Stampa user ID effettivo
id -g      ← Stampa group ID effettivo
```

Lo si renda SETUID **root** e lo si esegua.

```
chown root ./scr.sh
chmod 4755 ./scr.sh
./scr.sh
```

Il risultato dell'esperimento

(L'esecuzione è identica a quella in assenza di SETUID **`root`**)

Lo script **`scr.sh`** stampa user e group ID effettivo dell'utente normale che lo ha lanciato.
Non vi è traccia di ID privilegiati.
Ciò implica ancora l'abbassamento dei privilegi da parte di **`bash`**.

Un piccolo trucchetto

(Poco noto ai più…)

L'opzione **`-p`** di BASH (privileged mode) inibisce l'abbassamento dei privilegi.

Provare per credere (con la BASH SETUID **`root`**):

```
./bash -p
```

La BASH esegue con i diritti di root!

Oops…

A scanso di equivoci

(Altrimenti sembra che **`bash -p`** esegua sempre come **`root`**)

È necessario che un utente con diritti di amministratore abbia, in precedenza, reso SETUID **`root`** una BASH.

Ciò è possibile solo se si è in possesso di credenziali di **`root`**.

Solo se questa condizione è verificata è possibile l'elevazione di privilegi tramite **`bash -p`**.

Cancellazione SETUID nelle copie

(Impedisce la copia di binari pericolosi in cartelle locali scrivibili)

La copia di un file SETUID/SETGID perde il bit SETUID/SETGID.

Altrimenti, si potrebbe copiare un binario SETUID in una cartella locale e modificarlo a piacimento.

Magari permettendo l'esecuzione di una shell.

Un esempio concreto

(Copia di un programma SETUID **`root`** in una directory locale)

Ad esempio, si copi il file **`/usr/bin/passwd`** in una directory locale.

```
cp /usr/bin/passwd .
```

Se ne esaminino i metadati.

```
ls -l passwd
```

Il risultato dell'esperimento

(L'esecuzione è identica a quella in assenza di SETUID **`root`**)

I metadati del file **`passwd`** sono privati dei bit SETUID e SETGID.

```
-rwxr-xr-x 1 andreoli andreoli 52528 28 mar 18.05 passwd
```

Notate la presenza della **`x`**
(e non della **`s`**).

La chiamata di sistema `ptrace()`

(Meccanismo per l'osservazione ed il controllo di un processo)

La chiamata di sistema **`ptrace()`** permette ad un processo padre "tracciante" di osservare e controllare l'esecuzione di un processo figlio "tracciato".

→ Meccanismo per l'implementazione di debugger (ad es., **`gdb`**) e tracciatori vari (ad es., **`strace`**).

Esecuzione passo passo.

Ispezione e modifica registri.

…

Inibizione SET[UG]ID con `ptrace()`

(Impedisce la modifica di processi tracciati con privilegi maggiori dei traccianti)

Se il processo tracciato ha privilegi superiori al processo tracciante, il kernel del Sistema Operativo riduce le funzionalità della chiamata **`ptrace()`**.

→ Impedisce l'aggancio di un tracciante ad un processo già in esecuzione.

→ Se un tracciante fa partire un processo tracciato ex-novo, quest'ultimo non può elevare i suoi privilegi.

Motivazione

(Impedire l'iniezione di codice in processo privilegiato)

L'obiettivo è impedire la lettura e la modifica di un programma privilegiato tramite un debugger.

→ Inserendo (ad esempio) codice che invoca una shell.

→ Modificando (sempre ad esempio) il registro puntatore istruzioni all'indirizzo iniziale di tale codice.

Un esempio concreto

(Impedisce la modifica di processi tracciati con privilegi maggiori dei traccianti)

Si faccia partire tramite debugger la BASH SETUID **`root`** vista in precedenza.

```
gdb ./bash
b main
r
```

Su un altro terminale si stampino gli ID reali ed effettivi della BASH appena eseguita:

```
ps -p $(pgrep -n bash) -o ruid,rgid,euid,egid
```

Il risultato dell'esperimento

(L'esecuzione è identica a quella in assenza di SETUID **`root`**)

La BASH SETUID **`root`** esegue con gli user ID reali ed effettivo dell'utente (normale) che ha lanciato il debugger.

Non vi è traccia di ID privilegiati.

Ciò implica la riduzione delle funzionalità di **`ptrace()`**.

So this is how it feels

(When SETUID and SETGID are being inhibited)

Gestione dei privilegi negli altri linguaggi

(È possibile? Con la stessa varietà del C?)

Tutti i meccanismi ora visti sono stati concepiti per l'uso in programmi scritti nel linguaggio C.

È possibile abbassare e ripristinare i privilegi in altri ambienti di programmazione?

Linguaggi di scripting dinamici 1/2

(Python, Ruby, Lua, …)

Proprio come in BASH, nei moderni sistemi UNIX, i bit SETUID/SETGID sono ignorati sugli script di ogni genere.

- Script: inizia con la stringa **`#!/path/to/interpreter`**

L'unico programma che si può impostare SETUID **`root`** è proprio l'interprete.

- Bisogna essere **`root`** per farlo.
- È una pessima idea (gli script eseguono come **`root`**).

Linguaggi di scripting dinamici 2/2

(Python, Ruby, Lua, …)

Avendo a disposizione un interprete SETUID **`root`**, è possibile gestire il privilegio all'interno dello script tramite le funzioni di libreria fornite con l'interprete.

Un esempio concreto

(Abbassamento e ripristino privilegi tramite interprete SETUID **`root`**)

Si copi l'interprete Python in una directory locale.

```
cp /usr/bin/python3 .
```

Si renda SETUID **`root`** l'interprete.

```
chown root python3
chmod u+s python3
```

Si esegua lo script **`drop_rest.py`** (che abbassa e ripristina i privilegi con **`setresuid()`**).

```
./python3 drop_rest.py
```

Il risultato dell'esperimento

(L'abbassamento ed il ripristino dei privilegi funziona)

Lo script **`drop_rest.py`** abbassa e ripristina correttamente i privilegi di esecuzione.

Un altro esempio concreto

(Abbassamento e ripristino privilegi tramite script SETUID **`root`**)

Si renda SETUID **`root`** lo script **`drop_rest.py`**.

```
chown root drop_rest.py
chmod u+s drop_rest.py
```

Si esegua lo script **`drop_rest.py`** con l'interprete Python standard.

```
python3 drop_rest.py
```

Il risultato dell'esperimento

(L'abbassamento ed il ripristino dei privilegi è inibito)

Lo script **`drop_rest.py`** stampa sempre gli ID reale, effettivo e salvato dell'utente che ha lanciato il comando.

Non vi è traccia di ID privilegiati.

Ciò implica l'inibizione dei privilegi SETUID/SETGID.

Java

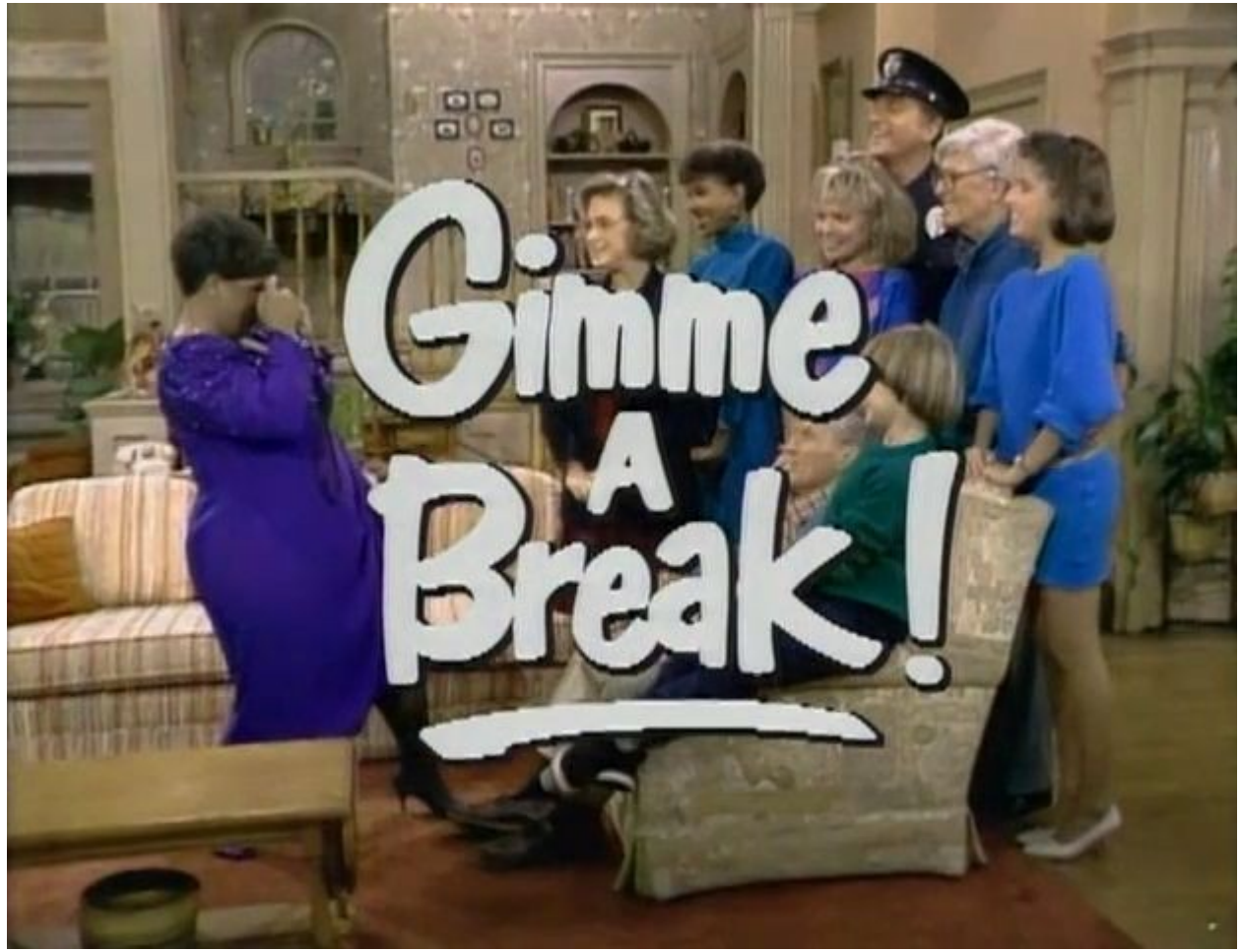
(Non supporta la gestione dei privilegi SETUID/SETGID)

L'ambiente di esecuzione di Java non supporta la gestione dei privilegi via bit SETUID/SETGID.

È possibile (ma molto macchinoso) usare JNI (Java Native Interface) per effettuare chiamate di libreria/sistema C dall'interno di una classe Java.

Per chi è interessato (non costituisce programma di esame):

https://www.nag.com/industryArticles/CallingCLibraryRoutinesfromJava.pdf


Gimme
A
Break!

Destrutturazione dei privilegi di `root`

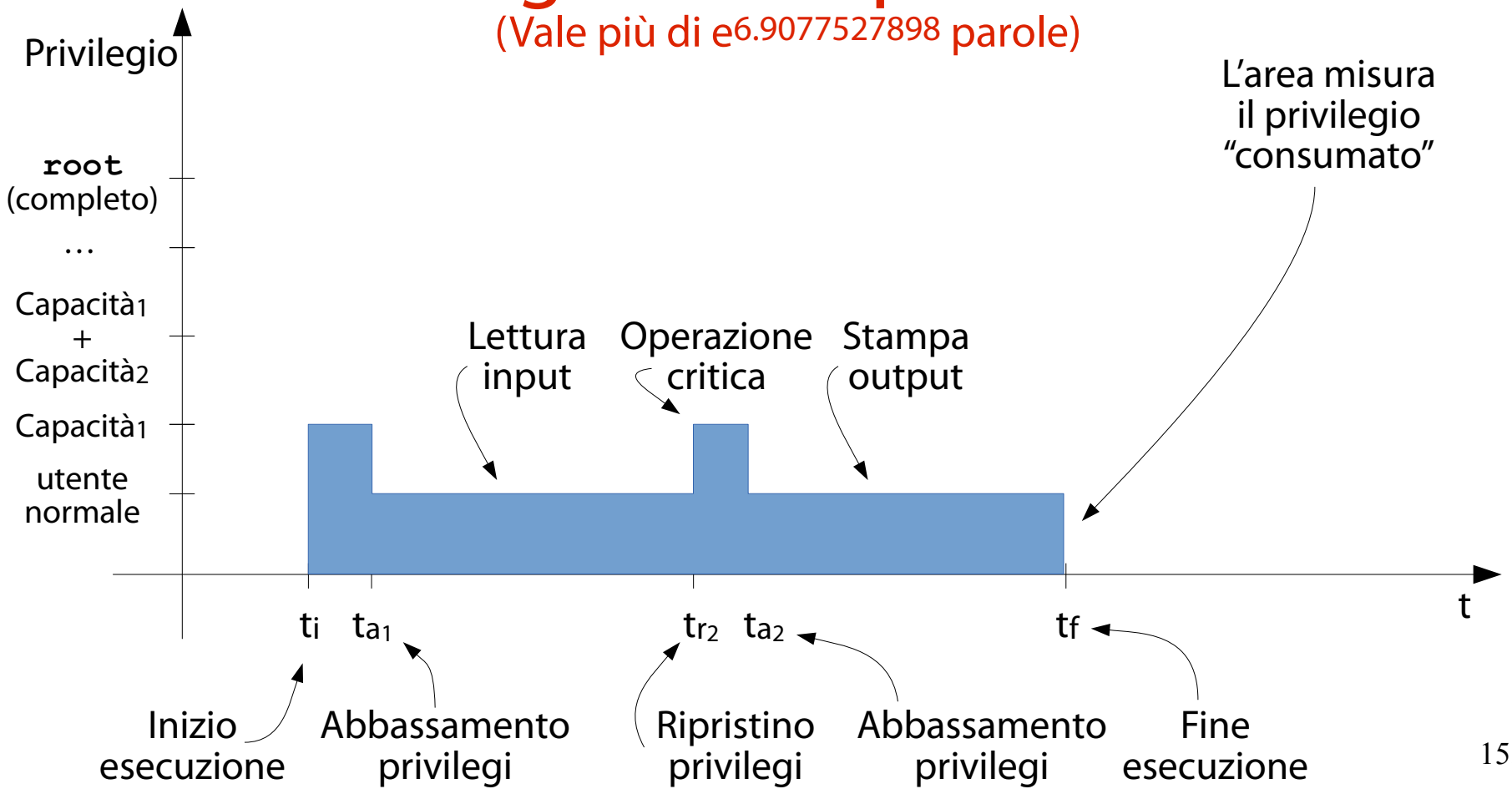
(Capacità di root = capacità$_1$ + capacità$_2$ + … + capacità$_n$)

Il SO GNU/Linux fornisce il meccanismo delle **capability**. In sostanza, i pieni privilegi dell'utente `root` sono suddivisi in tanti singoli sotto-privilegi acquisibili separatamente dai processi.

→ Non è più necessario acquisire i pieni poteri. Si acquisiscono i poteri strettamente necessari per risolvere uno specifico problema.

Un grafico esplicativo

(Vale più di $e^{6.9077527898}$ parole)



Alcune domande

(Doverose)

Come si implementano l'abbassamento ed il ripristino dei privilegi in un programma tramite capability?

È disponibile una API?

È una API di sistema (chiamate di sistema, wrapper glibc)?

L'API di sistema è unica o si è evoluta nel tempo?

Risposte

(Confortanti (mica tanto...))

A differenza del meccanismo basasto su bit SETUID/SETGID (che ha risentito di un periodo di maturazione lungo 40 anni), l'API delle capability è nuova e ben definita.

Il meccanismo delle capability è, tuttavia, più complicato di quello basato bit SETUID/SETGID.

Quali capability esistono?

(**`man 7 capabilities`** le spiega tutte)

Di capability ne esistono tante.

`man 7 capabilities` per tutti i sordidi dettagli.

Esempi di capability:

forgiare e spedire pacchetti di rete arbitrari.
impostare interfacce di rete.
montare e smontare dischi.
riavviare il sistema.
…

Capability di processo

(**Permesse**, effettive, ereditabili)

Capability permesse (permitted capability).

Queste sono le capability cui è concesso l'uso al processo (dal nucleo) durante la sua esecuzione.

Alcune di queste capability possono essere superflue; il processo può abbandonarle.

Attenzione! Una volta abbandonate, il processo non può più riprenderle!

Capability di processo

(Permesse, **effettive**, ereditabili)

Capability effettive (effective capability).

Queste sono le capability non abbandonate, con cui il processo può operare durante l'esecuzione.

Il processo può rilasciare temporaneamente i propri privilegi disabilitando le opportune capability permesse.

Il processo può recuperare i pieni privilegi attivando nuovamente le opportune capability permesse.

Capability di processo

(Permesse, effettive, **ereditabili**)

Capability ereditabili (inheritable capability). Queste sono le capability trasferibili all'insieme delle capability permesse quando il processo carica in memoria una nuova immagine con la chiamata di sistema **`execve()`**.

Capability di file

(La naturale estensione del bit SETUID)

Un file può avere (o no) impostate le capability.
Se le capability sono impostate, esse sono usate dal kernel del Sistema Operativo per stabilire gli insiemi di capability possedute da un processo che esegue quel file.

Un file espone tre insiemi di capability.

Capability di file

(**Permesse**, effettive, ereditabili)

Capability permesse (permitted capability). Queste capability sono inserite nell'insieme delle capability permesse al processo all'inizio della sua esecuzione.

Capability di file

(Permesse, **effettive**, ereditabili)

Capability effettive (effective capability).

Questo è un singolo bit.

Se è abilitato, durante il lancio del processo le capability permesse sono tutte abilitate di default.

Capability di file

(Permesse, effettive, **ereditabili**)

Capability ereditabili (inheritable capability).

Queste capability sono messe in AND bit a bit con le capability ereditabili del processo.

Il risultato è l'insieme di capability che il processo farà ereditare se deciderà di lanciare un nuovo programma.

Logicamente, tale insieme conterrà l'intersezione delle capability:

- offerte dal processo;
- presenti nel file.

Elevazione privilegi con le capability

(Che capability ottiene un processo quando va in esecuzione?)

Il kernel del Sistema Operativo stabilisce le capability a disposizione di un processo durante il suo avvio, nel momento in cui è caricata in memoria l'immagine di un file eseguibile.

Alcune definizioni

(Necessarie per illustrare le operazioni del nucleo)

Siano:

P → un insieme di capability del processo prima del caricamento dell'immagine

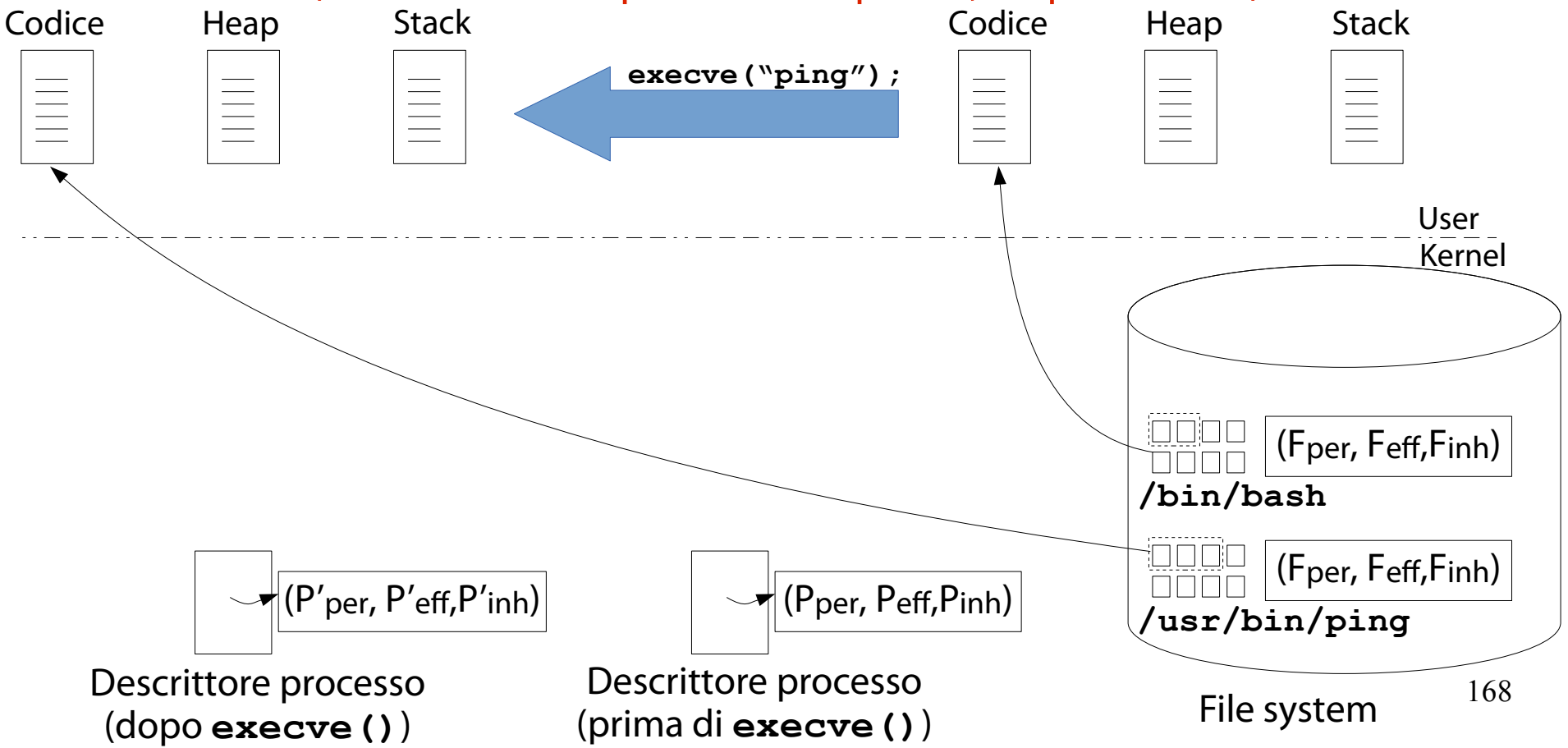
P' → un insieme di capability del processo dopo il caricamento dell'immagine

F → un insieme di capability del file

M → una maschera di capability denominata **capability bounding set** (inizialmente contenente tutte le capability) da cui il processo può depennare quelle che non vuole far ereditare ai suoi figli.

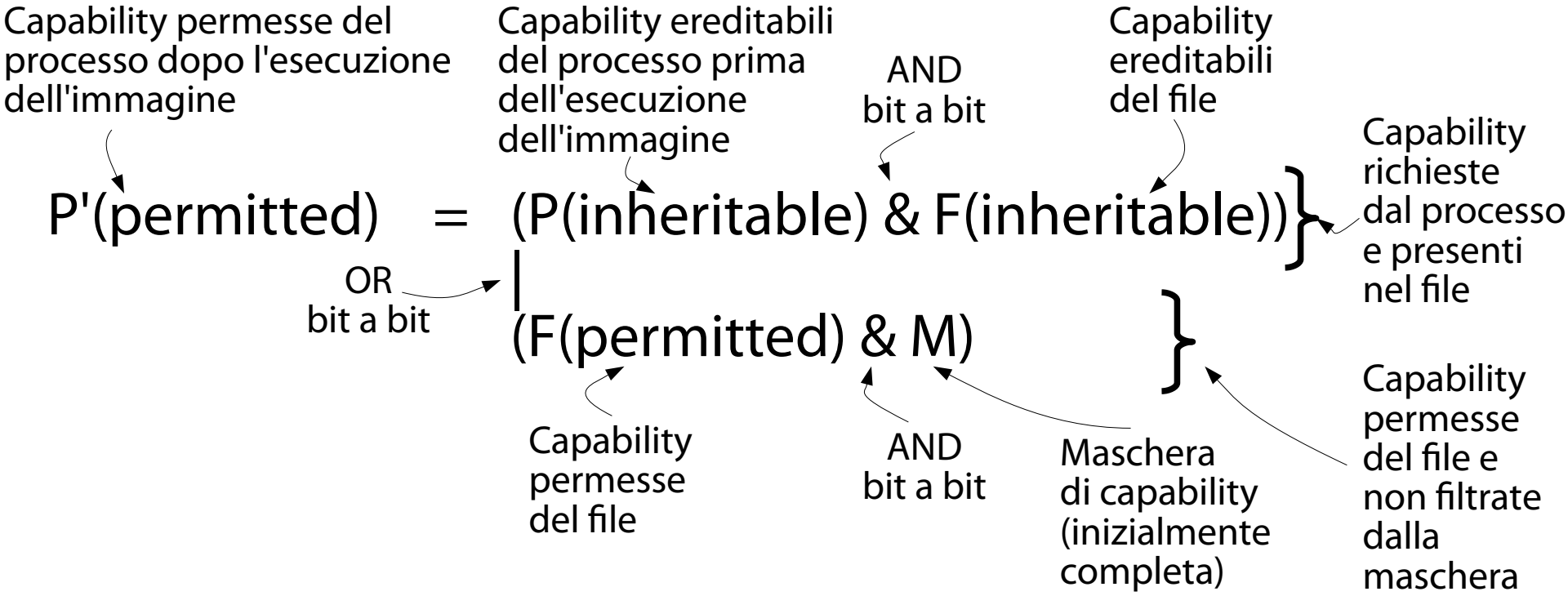
Un diagramma illustrativo

(Potrebbe valere più di 10000 parole, in questo caso)



Calcolo dell'insieme P'(permitted)

(Le capability permesse del processo, dopo l'esecuzione dell'immagine)



Una opportuna semplificazione

(P'(permitted) = F(permitted))

Nel caso più comune, un processo **`bash`** esegue l'immagine di un comando.

Il processo **`bash`** non ha capability.

→ P(inheritable) = 0

→ P(inheritable) & F(inheritable) = 0

La maschera M ha tutti i bit impostati ad 1.

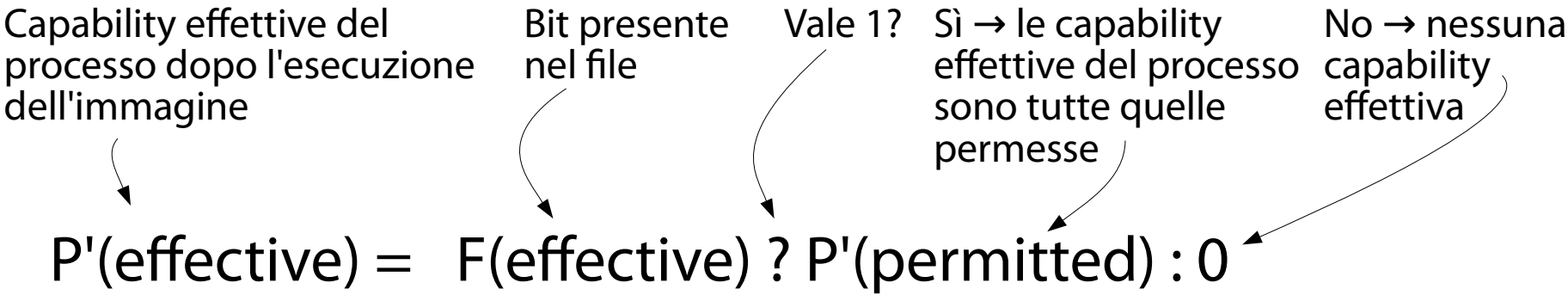
→ F(permitted) & M = F(permitted) & 11…1 = F(permitted)

In definitiva:

P'(permitted) = 0 | F(permitted) = F(permitted)

Calcolo dell'insieme P'(effective)

(Le capability effettive del processo, dopo l'esecuzione dell'immagine)



Calcolo dell'insieme P'(inheritable)

(Le capability ereditabili del processo, dopo l'esecuzione dell'immagine)

Capability ereditabili del processo dopo l'esecuzione dell'immagine

Sono esattamente quelle ereditabili prima della esecuzione dell'immagine

$$P'(\text{inheritable}) = P(\text{inheritable})$$

Come riconoscere file con capability?

(Alcuni tratti visuali)

I file eseguibili con capability:

possono essere evidenziati in rosso nell'output di **`ls -l`**;

NON hanno il bit di permesso "s" nell'output di **`ls -l`**.

Ad es.:

```
$ ls -l /usr/bin/ping
-rwxr-xr-x 1 root root 52528 29 ott 17.54 /usr/bin/ping
```

Lettura di capability di file

(Il comando **`getcap`**)

Il comando **`getcap`** mostra le capability di un file. Il suo uso è molto semplice:

```
getcap [OPZIONI] file...
```

Ad esempio:

```
getcap /usr/bin/ping
```

Memento!

(Comando **`getcap`** e Debian)

Nel Sistema Operativo Debian GNU/Linux:

`getcap` è considerato un programma di amministrazione di sistema essenziale all'avvio (è in **`/sbin`**);

un utente normale non ha **`/sbin`** nel **`PATH`**.

→ Scrivendo **`getcap`** si ottiene probabilmente un "command not found".

Bisogna eseguire esplicitamente il comando **`/sbin/getcap`**.

Il formato delle capability di getcap

(Una sequenza di clausole separate dal carattere `,`)

In generale, le capability mostrate dal comando **`getcap`** sono rappresentate da un insieme di clausole separate dal carattere `,` (virgola).

`comando = clausola`$_1$`,clausola`$_2$`,…,clausola`$_N$

Ogni clausola è composta da:

- un elenco di capability;
- un'azione (aggiunta, rimozione, reset+aggiunta);
- un elenco di insiemi di capability su cui compiere l'azione.

`clausola`$_i$ `= <`*`capability`*`><`*`azione`*`><`*`insiemi`*`>

Elenco delle capability

(Un elenco di nomi validi di capability separato da “,”, oppure **`all`**)

L'elenco delle capability è separato dal carattere **`,`** (virgola). L'elenco può anche essere nullo.

Per specificare tutte le capability viene usata la parola chiave **`all`**.

Azioni sulle capability

(+ aggiunge; – rimuove; = pulisce prima di reimpostare)

L'azione sulle capability è uno dei tre caratteri seguenti:

- **+** → aggiunge le capability
- **–** → rimuove le capability
- **=** → rimuove le capability considerate dagli insiemi effective, permitted ed inheritable, prima di impostarle.

Insiemi delle capability

(**`e`** → effettive; **`p`** → permesse; **`i`** → ereditate)

Gli insiemi delle capability sono uno o più dei seguenti insiemi:

- **`e`** → insieme delle capability effettive (effective)
- **`p`** → insieme delle capability permesse (permitted)
- **`i`** → insieme delle capability ereditate (inherited)

Quali capability ha **`ping`**?

(In qualità di file eseguibile su disco)

Quali capability ha il file **`/usr/bin/ping`**?

```
/usr/bin/ping = cap_net_raw+ep
```

Viene selezionata la capability **`CAP_NET_RAW`**.

Viene accesa nell'insieme permitted.

- Sarà attivata sull'insieme permitted del processo, quando inizia l'esecuzione.

Viene acceso il bit effective.

- Le capability permesse del processo sono tutte abilitate di default.

La capability **CAP_NET_RAW**

(Invio/ricezione diretta di buffer; bind ad un qualunque indirizzo (proxy trasp.))

Che privilegi fornisce **CAP_NET_RAW**?

```
man 7 capabilities
```

Ping usa questo

È permessa la creazione di un **raw socket** per l'invio e la ricezione diretta di buffer verso e da una scheda di rete.

Casi d'uso: creazione di datagrammi non standard (spesso, "costruiti male").

È permesso l'aggancio (**bind**) ad un qualunque indirizzo di rete (transparent proxy).

Come trovare file con capability

(Comando **`getcap`**, argomento "*")

Per scoprire i file eseguibili con capability si può eseguire **`getcap`** con argomenti "wildcard" che rappresentano gli eseguibili in questione:

```
getcap {/usr,}/{s,}bin/*
```

Se si vuole ricercare l'intero file system, si può usare l'opzione **`-r`** (recursive search):

```
getcap -r /
```

Capire che cosa fa l'azione "="

(Con un esempio, si spera semplice)

Esempio: che cosa fa **`all=p`**?

- **`all`** → considera tutte le capability
- = → rimuove tutte le capability dagli insiemi **`e`**, **`p`**, i prima di impostarle
- **`p`** → imposta le capability richieste nell'insieme **`p`** (permitted)

Lettura di capability di processo

(Il comando **`getpcaps`**)

Il comando **`getpcaps`** mostra le capability di un processo in esecuzione. Il suo uso è molto semplice:

```
getpcaps pid [pid ...]
```

Ad esempio, su un terminale si esegua **`ping`**.

```
ping 8.8.8.8
```

Su un altro terminale si esegua **`getpcaps`**:

```
getpcaps $(pgrep -n ping)
```

Quali capability ha **`ping`**?

(In qualità di processo in esecuzione)

Quali capability ha il processo **`ping`**?

```
Capabilities for `24547': = cap_net_raw+p
```

Viene usata la capability **`CAP_NET_RAW`**.

Viene accesa nell'insieme permitted.

- É stata fornita al processo ad inizio esecuzione.

Non è presente l'insieme effective.

- La capability è acquisibile (ma non ancora acquisita, oppure rilasciata).

Che cosa se ne deduce?

(Ipotesi di funzionamento "alla cieca" (senza codice sottomano))

Senza analizzare il codice, che cosa si può ipotizzare sulla gestione dei privilegi?

`ping`

parte privilegiato (**`CAP_NET_RAW`**);
rilascia temporaneamente i privilegi;
riprende temporaneamente i privilegi per creare un socket raw;
rilascia per sempre i privilegi;
invia i buffer e stampa le risposte;
ad un certo punto esce.

Impostazione di capability

(Il comando **`setcap`**)

Il comando **`setcap`** imposta le capability di un file. Il suo uso è molto semplice:

```
setcap [OPZIONI] capability file
```

Esattamente come nel caso di **`chown root ...`**, il comando **`setcap`** ha bisogno di privilegi elevati per poter eseguire.

Un esempio concreto
(Aggiunta di una capability ed esecuzione)

Si copi il file eseguibile **`/usr/bin/ping`** in una directory locale.

```
cp /usr/bin/ping .
```

Si abiliti **`cap_net_raw+ep`** su **`./ping`**:

```
setcap cap_net_raw+ep ./ping
```

Si esegua il comando: **`./ping`**
Con la capability, **`./ping`** funziona.

Un esempio concreto
(Rimozione di una capability ed esecuzione)

Si rimuovano tutte le capability su **`./ping`** (opzione **`-r`** di setcap):

```
setcap -r ./ping
```

Si esegua il comando: **`./ping`**
Senza la capability, **`./ping`** NON funziona (poiché non riesce a creare il socket raw).

Una domanda arguta

(Di quelle che piacciono tanto al docente)

Non si poteva semplicemente "spegnere" la capability con il comando seguente?

```
setcap cap_net_raw-ep ./ping
```

Certo che si poteva! Si immettano i comandi:

```
setcap cap_net_raw-ep ./ping
setcap cap_net_raw-ep ./ping
./ping
```

→ Il comando non funziona (giustamente).

Una domanda ancora più arguta

(Di quelle che piacciono tantissimo al docente)

Se le capability sono rimosse, perché il comando **`ls -l`** mostra ancora **`./ping`** in rosso?

In quei Sistemi Operativi che hanno BASH configurata in modo tale da colorare un eseguibile di rosso in presenza di capability, ovviamente.

```
$ ls -l ./ping
-rwxr-xr-x 1 root root 52528 29 ott 17.54 ./ping
```

Attributi estesi di un file

(Metadati arbitrari usabili dalle applicazioni)

Nei Sistemi Operativi moderni (GNU/Linux incluso) è possibile associare metadati arbitrari ad un file.

Le applicazioni fanno l'uso che vogliono di tali medadati.

Attributo esteso: è una coppia di stringhe (nome, valore) associabile ad un file.

I nomi degli attributi estesi sono organizzati in maniera gerarchica.

Gestione degli attributi estesi

(Comandi **`getfattr`**, **`setfattr`**)

Gli attributi estesi sono gestiti dai comandi **`getfattr`** e **`setfattr`**.

Visione degli attributi:

```
getfattr -n nome file
```

Modifica degli attributi:

```
setfattr -n nome -v valore file
```

Cancellazione degli attributi:

```
setfattr -x nome file
```

Capability tramite attributi estesi

(Attributo esteso **`security.capability`**)

La libreria **libcap** memorizza le capability di file tramite un attributo esteso di nome **`security.capability`**.

Si visionino le capability possedute da **`./ping`**:

```
$ getfattr -n security.capability ./ping
# file: ping
security.capability=0sAAAAAgAAAAAAAAAAAAAAAAAAA=
```

Che cosa se ne deduce?

(La rimozione della capability non implica la cancellazione dell'attributo esteso)

La rimozione delle capability non implica la rimozione dell'attributo esteso che le contiene.

La shell **`bash`** colora di rosso un eseguibile:

- SETUID;
- contenente l'attributo esteso **`security.capability`**.

Un piccolo esperimento

(Cancellazione delle capability di un file e visione dell'attributo esteso)

Si provi a cancellare le capability di **`./ping`**:

```
setcap -r ./ping
```

Si visualizzi l'attributo esteso:

```
$ getfattr -n security.capability ./ping
ping: security.capability: No such attribute
```

Si visualizzi **`./ping`** in formato lungo:

```
$ ls -l ./ping
-rwxr-xr-x 1 root root 52528 29 ott 17.54 ./ping
```

Che cosa se ne deduce?

(La cancellazione delle capability rimuove anche l'attributo esteso)

La cancellazione delle capability forza anche la cancellazione dell'attributo esteso associato: **`security.capability`**.

API di gestione delle capability

(Fornita dalla libreria **`libcap`**)

La libreria **`libcap`**:

definisce tipi di dato opachi per rappresentare singole capability o insiemi di capability;
definisce funzioni per la creazione, l'ottenimento, la modifica e l'impostazione di capability di processo e di file.

Per usare queste definizioni:

si includa **`<sys/capability.h>`**;
si linki l'eseguibile con **`-lcap`**.

Una osservazione

(Si spera gradita)

Per semplificare la trattazione, ci si limita alla gestione delle capability di processo.
La gestione delle capability di file serve per analizzare il funzionamento di **`getfattr`** e **`setfattr`**.

Esula dallo scopo primario del presente corso.

Tipo di dato opaco **`cap_t`**

(Contiene i tre insiemi permitted, effective, inheritable)

Il tipo di dato opaco **`cap_t`** punta ad una struttura dati (**`struct _cap_struct`**) contenente tre array di bit.

- Un array di bit per le capability permitted.
- Un array di bit per le capability effective.
- Un array di bit per le capability inheritable.

La gestione degli array di bit è trasparente all'utente; se ne occupa **`libcap`** internamente.

Tipo di dato opaco **`cap_value_t`**

(Rappresenta una specifica capability)

Il tipo di dato opaco **`cap_value_t`** è un intero che rappresenta una specifica capability.

`CAP_NET_RAW`
`CAP_CHOWN`
`CAP_KILL`
…

Tipo di dato opaco `cap_flag_t`

(Rappresenta l'insieme di capability su cui operare)

Il tipo di dato opaco **`cap_flag_t`** è un enumeratore che rappresenta uno specifico insieme di capability.

- **`CAP_EFFECTIVE`**
- **`CAP_INHERITABLE`**
- **`CAP_PERMITTED`**

Tipo di dato opaco `cap_flag_value_t`

(Rappresenta l'insieme su cui operare)

Il tipo di dato opaco **`cap_flag_value_t`** è un enumeratore che rappresenta la specifica operazione da eseguire.

- **`CAP_SET`**: imposta la capability
- **`CAP_CLEAR`**: rimuove la capability

Funzione **`cap_init()`**

(Crea una struttura dati **`_cap_struct`** vuota)

La funzione di libreria **`cap_init()`** crea una struttura dati **`_cap_struct`** vuota e ritorna il suo puntatore **`cap_t`**.
Ritorna NULL in caso di errore.

`man cap_init` per tutti i dettagli.

```
cap_t = cap_init();
```

Funzione `cap_get_proc()`

(Crea una `_cap_struct` vuota e la riempie con le capability del processo)

La funzione di libreria **`cap_get_proc()`** crea una struttura dati **`_cap_struct`** vuota, la riempie con le capability attualmente possedute dal processo invocante e ritorna il suo puntatore **`cap_t`**.

`man cap_get_proc` per tutti i dettagli.

```
cap_t = cap_get_proc();
```

Funzione `cap_set_flag()`

(Imposta o rimuove le capability di uno o più insiemi di capability)

La funzione di libreria **`cap_set_flag()`** imposta o rimuove una o più capability da un insieme di capability.
Ritorna -1 in caso di errore.

`man cap_set_flag` per tutti i dettagli.

Il prototipo di `cap_set_flag()`

(Non esattamente immediato)

Il protitipo della funzione **`cap_set_flag()`** è il seguente.

```
int cap_set_flag(cap_t cap_p,
   cap_flag_t flag,
   int ncap, const cap_value_t *caps,
   cap_flag_value_t value);
```

A che cosa servono tutti questi argomenti?

Argomenti di **cap_set_flag()** 1/3

(cap_t cap_p, cap_flag_t flag)

cap_t cap_p punta alla **_cap_struct** contenente i tre vettori di bit (capability effective, permitted, inheritable) che si intende modificare.

cap_flag_t flag identifica lo specifico insieme di capability su cui operare.

CAP_EFFECTIVE, **CAP_INHERITABLE**, **CAP_PERMITTED**

Argomenti di `cap_set_flag()` 2/3

`(int ncap, const cap_value_t *caps)`

`const cap_value_t *caps` è un array di capability che si vogliono aggiungere o rimuovere.

`CAP_NET_RAW`, **`CAP_CHOWN`**, **`CAP_KILL`**, …

`int ncap` contiene la lunghezza di tale array.

Argomenti di `cap_set_flag()` 3/3

(`cap_flag_value_t value`)

`cap_flag_value_t value` specifica l'azione che si intende compiere.

- **`CAP_SET`**, **`CAP_CLEAR`**

Esempio d'uso

(Alla fine, neanche troppo complesso)

Ad esempio, per impostare la capability **`CAP_NET_RAW`** nel sottoinsieme effective di un insieme di capability:

```
cap_value_t caplist[1];
cap_t cap_p;

cap_p = cap_get_proc();
caplist[0] = CAP_NET_RAW;
ret = cap_set_flag(cap_p, CAP_EFFECTIVE,   1,
                   caplist, CAP_SET);
```

Funzione `cap_set_proc()`

(Prova ad impostare le capability per il processo invocante)

La funzione di libreria **`cap_set_proc()`** prova ad impostare per il processo invocante le capability memorizzate nella **`_cap_struct`** puntata da **`cap_t`**.
Ritorna -1 in caso di errore.

`man cap_set_proc` per tutti i dettagli.

```
ret = cap_set_proc(cap_t);
```

Funzione **`cap_free()`**

(Dealloca la memoria della struttura dati **`_cap_struct`**)

La funzione di libreria **`cap_free()`** dealloca la struttura dati **`_cap_struct`** puntata da **`cap_t`.** Ritorna -1 in caso di errore.

`man cap_free` per tutti i dettagli.

```
ret = cap_free(cap_t);
```

Abbassamento permanente

(Piuttosto complicato da effettuare)

Se un processo ha la capability **`CAP_SETPCAP`** (la capacità impostare e rimuovere capability a piacere), può spegnere bit nel capability bounding set (M), di fatto disattivando per sempre la capability associata.

Chiamata di sistema **`prctl()`**

(Analogo di **`ioctl()`** per i processi; non fa parte di **`libcap`**)

La chiamata di sistema **`prctl()`** esegue operazioni di basso livello su un processo.
Ritorna -1 in caso di errore.

`man 2 prctl` per tutti i dettagli.

L'operazione **`PR_CAPBSET_DROP`** spegne una capability del bounding set (M) per sempre.
→ Quella capability non è più usabile.

```
ret = prctl(PR_CAPBSET_DROP, CAP_CHOWN);
```

Un programma di esempio
(Autenticazione di un utente)

Il programma di esempio seguente:

`check_password_caps.c`

Legge uno username ed una password da terminale, controlla la validità della password (aprendo **`/etc/shadow`**) e stampa un messaggio di successo o di errore.

Lo si compili.

`make`

CAP_DAC_READ_SEARCH

(La capability usata nell'esempio)

La capability **CAP_DAC_READ_SEARCH** permette al suo fortunato possessore di aggirare tutti i controlli dei permessi su file e directory.

→ Si riesce a leggere **/etc/shadow**.

Tale capability va impostata sul binario di esempio.

```
setcap CAP_DAC_READ_SEARCH=p check_password_caps
```

L'effetto di tale impostazione

(Che capability ha il processo **`check_password_caps`** all'avvio?)

$$
\begin{aligned}
P'(\text{permitted}) &= (P(\text{inheritable}) \,\&\, F(\text{inheritable})) \\
&\quad \mid (F(\text{permitted}) \,\&\, M)) \\
&= 0 \mid F(\text{permitted}) = F(\text{permitted}) \\
&= \text{CAP\_DAC\_READ\_SEARCH} \\
P'(\text{effective}) &= F(\text{effective}) \;?\; P'(\text{permitted}) : 0 \\
&= 0 \\
P'(\text{inheritable}) &= P(\text{inheritable}) = 0
\end{aligned}
$$

Una riflessione

(Che cosa implicano tali insiemi di capability per il processo?)

Il processo check_password_caps ha la capability permitted **`CAP_DAC_READ_SEARCH`**.
Poiché non è stato impostato il bit effective sul file, tale capability non diventa subito effettiva.
In altre parole, può ancora essere scartata definitivamente.

Controllo delle capability del file

(Prima dell'esecuzione del comando, giusto per sicurezza)

Prima di eseguire il comando, per sicurezza si può controllare la corretta impostazione della capability sul file.

```
$ getcap check_password_caps
check_password_caps = cap_dac_read_search+p
```

Esecuzione del comando

(Chiede username e password; restituisce il risultato dell'autenticazione)

Si esegua il comando **`check_password_caps`**.

```
$ ./check_password_caps
Username: andreoli
Password:
Successfully authenticated: UID=1000
```

“It all starts here.”

(By understanding privileges)

Coesistenza di bit SET[UG]ID e capability

(In uno specifico file)

La (confusa) realtà odierna è quella di un Sistema Operativo GNU/Linux che supporta sia i bit SETUID e SETGID, sia le capability su file.

Provare per credere.

```
$ cp /usr/bin/ping .
$ su
# chown root ping
# setcap cap_net_raw+ep ping
# ls -l ping; getcap ping
```

Coesistenza di bit SET[UG]ID e capability

(In uno specifico processo in esecuzione)

Se un processo privilegiato SETUID/SETGID esegue una delle operazioni seguenti, che cosa succede alle sue capability?

Caso 1. Abbassamento permanente dei privilegi.

Caso 2. Abbassamento e ripristino temporaneo dei privilegi.

Caso 3. Caricamento di una immagine eseguibile.

Abbassamento permanente dei privilegi

(Comporta l'azzeramento delle capability permitted ed effective)

Un processo in esecuzione con almeno uno user ID a 0 (**`root`**) imposta tutti i suoi ID a valori non nulli.

Abbassamento permanente dei privilegi.

→ Tutte le capability negli insiemi permitted ed effective sono azzerate.

Il processo non ha più modo di acquisirle.

Abbassamento temporaneo dei privilegi

(Comporta l'azzeramento delle capability effective)

Un processo in esecuzione con lo user ID effettivo a 0 (**`root`** effettivo) imposta lo user ID effettivo ad un valore non nullo.

Abbassamento temporaneo dei privilegi.

→ Tutte le capability nel solo insieme effective sono azzerate.

Quando il processo ripristina lo user ID effettivo a 0, l'insieme delle capability permitted è copiato nell'insieme delle capability effective.

Caricamento di un eseguibile

(Due sottocasi)

Un processo in esecuzione con lo user ID reale o effettivo a 0 (**`root`** o **`root`** effettivo) carica l'immagine di un nuovo eseguibile.

OPPURE

Un processo in esecuzione con nessun privilegio particolare carica l'immagine di un nuovo eseguibile SETUID/SETGID **`root`**.

Caricamento di un eseguibile

(Capability inheritable e permitted del file sono tutte attive)

→ L'intero insieme di capability inheritable del file è considerato attivo (tutti i bit ad 1).

Al termine del caricamento, il processo eredita tutte le capability di chi lo carica.

→ L'intero insieme di capability permitted del file è considerato attivo (tutti i bit ad 1).

Al termine del caricamento, se anche il bounding set M ha tutti i bit ad 1, il processo ha a disposizione tutte le capability e può scegliere quali scartare e quali usare.

Caricamento di un eseguibile

(Se caricante o caricato sono privilegiati → Capability effective del file = 1)

Se, inoltre:

lo user ID effettivo dal processo caricante è 0 (**`root`** effettivo);

OPPURE

il file eseguibile caricato è SETUID **`root`**;

→ la capability effective del file è considerata attiva (bit ad 1).

Tutte le capability permitted sono anche effective. Il processo ne scarta nessuna.

Capabilities are ignored with SETUIDs

(Facepalm. Because expressing how dumb that was in words just doesn't work.)

Domanda

(Si spera che la risposta sia "sì")

Anche un utente con una conoscenza di base dei dettagli implementativi intuisce il livello di confusione generato dalla convivenza di bit SETUID/SETGID e capability.

Domanda da 1M Eur: è possibile disabilitare i tre casi ora visti, di fatto abilitando un sistema basato su sole capability?

Il meccanismo "securebits"

(Incluso nel kernel Linux a partire dalla versione 2.6.26 nel 2008)

Il meccanismo **securebits** permette di disabilitare ognuno dei tre casi ora visti.

- Solo a partire dal kernel 2.6.26 in poi.
- Solo se è attivo nel kernel il supporto per le capability.

Ogni processo eredita dal padre (in seguito ad una **`fork()`**) un insieme di sei bit, gestibili con la chiamata di sistema **`prctl()`** e le opzioni seguenti.

- **`PR_GET_SECUREBITS`**: lettura securebit.
- **`PR_SET_SECUREBITS`**: impostazione securebit.

I securebit disponibili

(Inibiscono i Casi 1, 2, 3)

Flag	Significato
`SECBIT_KEEP_CAPS`	Non azzera le capability permitted quando un processo privilegiato abbassa i privilegi permanentemente (inibisce il Caso 1). `SECBIT_NO_SETUID_FIXUP` deve essere uguale a 0 (altrimenti l'effetto è nullo). Questo flag è azzerato con una `exec()`.
`SECBIT_NO_SETUID_FIXUP`	Non azzera le capability effective quando un processo con EUID = 0 abbassa i privilegi temporaneamente (inibisce il Caso 2).
`SECBIT_NOROOT`	Non assegna capability dopo il caricamento di una immagine, a meno che il file caricato non abbia capability (inibisce il Caso 3).

I securebit disponibili

(Impediscono le modifiche sui corrispondenti bit precedenti)

Flag	Significato
SECBIT_KEEP_CAPS_LOCKED	Se impostato, **SECBIT_KEEP_CAPS** non può più essere modificato.
SECBIT_NO_SETUID_FIXUP_LOCKED	Se impostato, **SECBIT_NO_SETUID_FIXUP** non può più essere modificato.
SECBIT_NO_ROOT_LOCKED	Se impostato, **SECBIT_NO_ROOT** non può più essere modificato.

Esempio di uso

(Tutto sommato, semplice)

Per fare in modo che un processo caricato possa ottenere solo le sue capability di file, si esegua nel processo caricante:

```
ret = prctl(PR_SET_SECUREBITS,
   /* SECBIT_KEEP_CAPS off */
   SECBIT_KEEP_CAPS_LOCKED |
   SECBIT_NO_SETUID_FIXUP |
   SECBIT_NO_SETUID_FIXUP_LOCKED |
   SECBIT_NOROOT | SECBIT_NOROOT_LOCKED);
```

Una osservazione
(Molto pertinente)

I due securebit seguenti sembrano molto simili:

`SECBIT_KEEP_CAPS`;
`SECBIT_NO_SETUID_FIXUP`.

Perché queste due funzionalità non sono state condensate in una sola?

Ad esempio, **`SECBIT_NO_SETUID_FIXUP`**.

SECBIT_KEEP_CAPS e **PR_SET_KEEPCAPS**

(Ecco la spiegazione)

Il bit **SECBIT_KEEP_CAPS** sostituisce la precedente implementazione tramite il bit **PR_SET_KEEPCAPS** (presente dal kernel 2.2.18).
I due bit innescano la stessa operazione.
La loro unica differenza è:

- un processo non ha bisogno della capability **CAP_SETPCAP** per **PR_SET_KEEPCAPS**;
- un processo ha bisogno della capability **CAP_SETPCAP** per **SECBIT_KEEP_KEEPCAPS**.

Esercizio finale

(“Un nuovo inizio col botto…”)

Si propone un ultimo esercizio, più realistico dei precedenti: l’analisi della gestione dei privilegi del software **`ping`**.
L’analisi è condotta sulle seguenti moderne distribuzioni GNU/Linux: Debian, Ubuntu, Fedora, Gentoo, Arch.

Individuazione del percorso di **`ping`**

(Indipendente dalla distribuzione GNU/Linux)

Qual è il percorso completo del comando **`ping`**?

```
$ which ping
```

Debian GNU/Linux, Ubuntu GNU/Linux, Gentoo GNU/Linux:
`/bin/ping`

Fedora GNU/Linux, Arch GNU/Linux:
`/usr/bin/ping`

Osservazioni

(**`ping`** ha percorsi diversi al variare delle distribuzioni GNU/Linux)

Il comando **`ping`** è installato su percorsi diversi al variare della distribuzione utilizzata.
→ Diaspora UNIX.
→ Ogni distribuzione, pur rispettando in generale la Linux Standard Base, incentiva iniziative volte al miglioramento.
Esempio: riunificazione dei binari esistenti nell'unica directory **`/usr/bin`**.

Individuazione del pacchetto software

(Debian, Ubuntu, Fedora GNU/Linux)

Come si chiama il pacchetto software fornente **`ping`**?

Debian, Ubuntu GNU/Linux:

```
$ dpkg -S /bin/ping
iputils-ping: /bin/ping
```

Fedora GNU/Linux:

```
$ rpm -qf /bin/ping
iputils-201603008-3.fc25.x86_64
```

Individuazione del pacchetto software

(Gentoo, Arch GNU/Linux)

Come si chiama il pacchetto software fornente **`ping`**?

Gentoo GNU/Linux:

```
$ equery belongs /bin/ping
* Searching for /bin/ping ...
net-misc/iputils-20151218 (/bin/ping)
```

Arch GNU/Linux:

```
$ pacman -Qo /bin/ping
/usr/bin/ping is owned by iputils
20180629.f6aac8d-4
```

Osservazioni

(Il nome del pacchetto software è sempre lo stesso: **`iputils`**)

Numero di versione a parte, il nome del pacchetto software è lo stesso al variare delle distribuzioni GNU/Linux.

→ **`iputils`**

Arch GNU/Linux sembra avere la versione più recente (sempre una buona cosa per la sicurezza).

Scaricamento dell'albero sorgente

(Debian, Ubuntu GNU/Linux)

Il sorgente di **`iputils`** si ottiene scaricando il pacchetto sorgente fornito dalla distribuzione.

Debian, Ubuntu GNU/Linux:

```
$ apt-get source iputils
iputils-ping: /bin/ping
```

Scaricamento dell'albero sorgente

(Fedora GNU/Linux)

Il sorgente di **`iputils`** si ottiene scaricando il pacchetto sorgente fornito dalla distribuzione.

Fedora GNU/Linux:

```
$ dnf download --source
iputils-201603008-3.fc25.x86_64
mkdir iputils; cd iputils
rpm2cpio ../iputils*rpm | cpio -idmv
tar xf iputils*tar.gz
```

Scaricamento dell'albero sorgente

(Gentoo GNU/Linux)

Il sorgente di **`iputils`** si ottiene scaricando il pacchetto sorgente fornito dalla distribuzione.

Gentoo GNU/Linux:

```
$ sudo ebuild /usr/portage/net-misc/iputils/iputils-20151218.ebuild fetch
$ sudo ebuild /usr/portage/net-misc/iputils/iputils-20151218.ebuild unpack
```

Scaricamento dell'albero sorgente

(Arch GNU/Linux)

Il sorgente di **`iputils`** si ottiene scaricando il pacchetto sorgente fornito dalla distribuzione.

Arch GNU/Linux:

```
$ asp checkout iputils
$ cd iputils/trunk
$ makepkg -o
```

Osservazioni

(Il nome del pacchetto software è sempre lo stesso: **`iputils`**)

In alcuni casi lo scaricamento dell'albero sorgente è semplicissimo.

- Debian, Ubuntu GNU/Linux.

Un singolo comando, lanciato da utente normale:

- scarica il pacchetto sorgente;
- spacchetta il pacchetto sorgente;
- estrae l'albero sorgente nella directory corrente.

Osservazioni

(Il nome del pacchetto software è sempre lo stesso: **`iputils`**)

In altri casi lo scaricamento dell'albero sorgente è semplice, ma un po' più scomodo.

Fedora GNU/Linux.

Più comandi, lanciati da utente normale:

scaricano il pacchetto sorgente;
spacchettano il pacchetto sorgente in una directory;
estraggono l'archivio sorgente in una sotto-directory.

Osservazioni

(Il nome del pacchetto software è sempre lo stesso: **`iputils`**)

In altri casi, lo scaricamento dell'albero sorgente è ancora più scomodo (ma più flessibile per il processo di compilazione).

Arch GNU/Linux.

Più comandi, alcuni lanciati da amministratore, altri lanciati da utente normale:

recuperano uno script di installazione del pacchetto;
usano lo script per scaricare il pacchetto sorgente;
Usano lo script per estrarre l'archivio sorgente in una sotto-directory.

Osservazioni

(Il nome del pacchetto software è sempre lo stesso: **`iputils`**)

Infine, lo scaricamento dell'albero sorgente può diventare un vero e proprio incubo.

- Gentoo GNU/Linux.

Più comandi, tutti lanciati da amministratore:

- usano uno script di installazione per scaricare il pacchetto sorgente in una directory di sistema;
- Usano uno script di installazione per estrarre l'archivio sorgente in una sotto-directory di sistema.

Individuazione dell'albero sorgente

(Debian, Ubuntu, Fedora GNU/Linux)

L'albero sorgente di **`iputils`** è spacchettato in una directory ben precisa.

Debian, Ubuntu GNU/Linux:

```
$ cd ./iputils-20121221
```

Fedora GNU/Linux:

```
$ cd ./iputils/iputils-s20160308
```

Individuazione dell'albero sorgente

(Gentoo, Arch GNU/Linux)

L'albero sorgente di **`iputils`** è spacchettato in una directory ben precisa.

Gentoo GNU/Linux:

```
$ cd /var/tmp/portage/net-misc/iputils-20151218/work/iputils-s20151218
```

Arch GNU/Linux:

```
$ cd src/iputils
```

Individuazione dei sorgenti di **`ping`**

(Indipendente dalla distribuzione GNU/Linux)

Dando un'occhiata all'albero sorgente, il linguaggio di implementazione sembra essere il C.

Per trovare tutti i sorgenti di **`ping`**, in prima istanza si possono cercare i file contenenti tale stringa:

```
find . -regex '^.*ping.*\.[ch]$'
```

I file sorgenti individuati

(Tutto sommato, pochi)

I file individuati da **`find`** sono i seguenti:

- **`ping.h`**
- **`ping.c`**
- **`ping_common.c`**
- **`ping6_common.c`**
- **`arping.c`**

Osservazioni

(C'è un "intruso", **`arping.c`**; ping sembra supportare IPv4 e IPv6)

Un file sorgente (**`arping.c`**) implementa il comando **`arping`** (**`ping`** a livello 2 tramite ARP).

Non si considera nell'analisi.

Il comando **`ping`** sembra supportare entrambi i protocolli IPv4 e IPv6.

Individuazione dei meccanismi

(Indagine tramite comando **`grep`**)

Ora bisogna capire se:

- non è usato alcun meccanismo di elevazione automatica dei privilegi;
- è usato il meccanismo SETUID/SETGID;
- è usato il meccanismo delle capability.

A tal scopo, si può usare il comando **`grep`** con opportune opzioni per la ricerca di espressioni regolari specifiche.

Indagine meccanismo SETUID/SETGID

(`grep -nHiE '[gs]et.*[ug]id' ping*`)

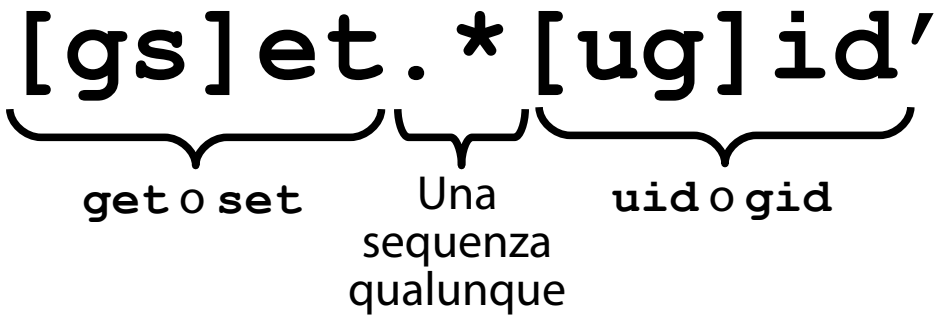
Il comando seguente:

```
grep -nHiE '[gs]et.*[ug]id' ping*
```

ricerca case-insensitive tutte le righe dei sorgenti di **`ping`** che verificano l'espressione regolare

```
'[gs]et.*[ug]id'
```

e stampa nome file e numero di riga.



Che cosa si è scoperto?

(Abbassamento permanente; abbassamento e ripristino temporanei)

Uso di **`setuid(getuid());`**
→ Abbassamento permanente dei privilegi.

Uso di **`seteuid(uid);`**
→ Abbassamento e ripristino temporaneo dei privilegi.

Indagine meccanismo capability

(grep -nHiE '(cap_.*)|prctl' ping*)

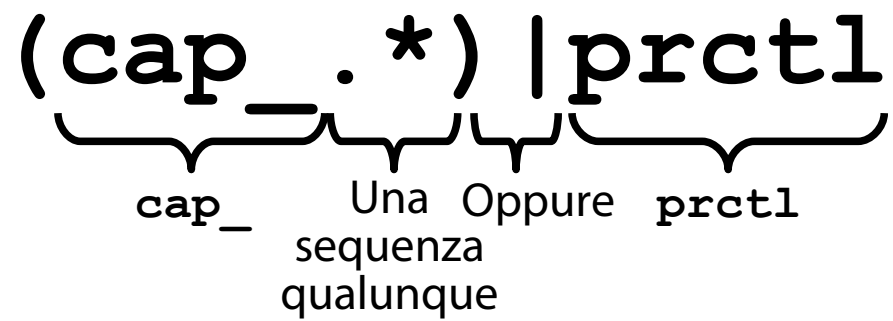
Il comando seguente:

grep -nHiE '(cap_.*)|prctl' ping*

ricerca case-insensitive tutte le righe dei sorgenti di **ping** che verificano l'espressione regolare

'(cap_.*)|prctl'

e stampa nome file e numero di riga.



Che cosa si è scoperto?

(Abbassamento e ripristino temporanei)

Uso dell'intera API di gestione delle capability.

Da **`cap_init();`** a **`cap_free();`**.

→ Abbassamento e ripristino temporaneo dei privilegi.

Presenza di **`prctl()`**:

```
prctl(PR_SET_KEEPCAPS, 1);
prctl(PR_SET_KEEPCAPS, 0);
```

Investigation time

(Even Mrs. Fletcher needs some time on this difficult case)

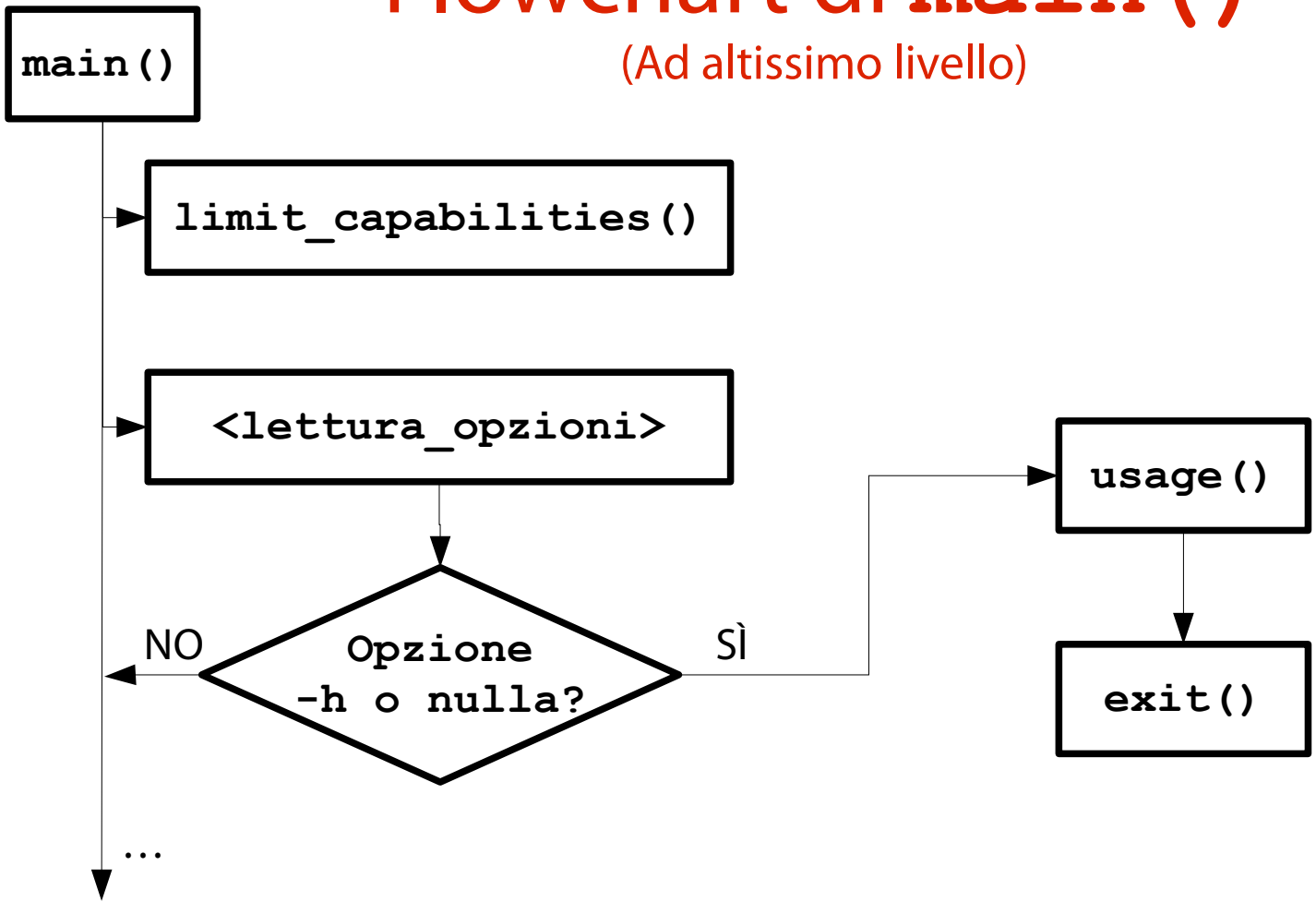
Analisi di `main()`

(Rapida, tanto per capire come sono gestiti i privilegi)

Si scorra rapidamente la funzione **`main()`** di **`ping.c`** (implementazione TCP/IP v4).
L'obiettivo è quello di capire a grandi linee come sono gestiti i privilegi nel percorso di codice coinvolto nell'invio di un datagramma.
In seguito si approfondisce l'analisi sui singoli meccanismi coinvolti.

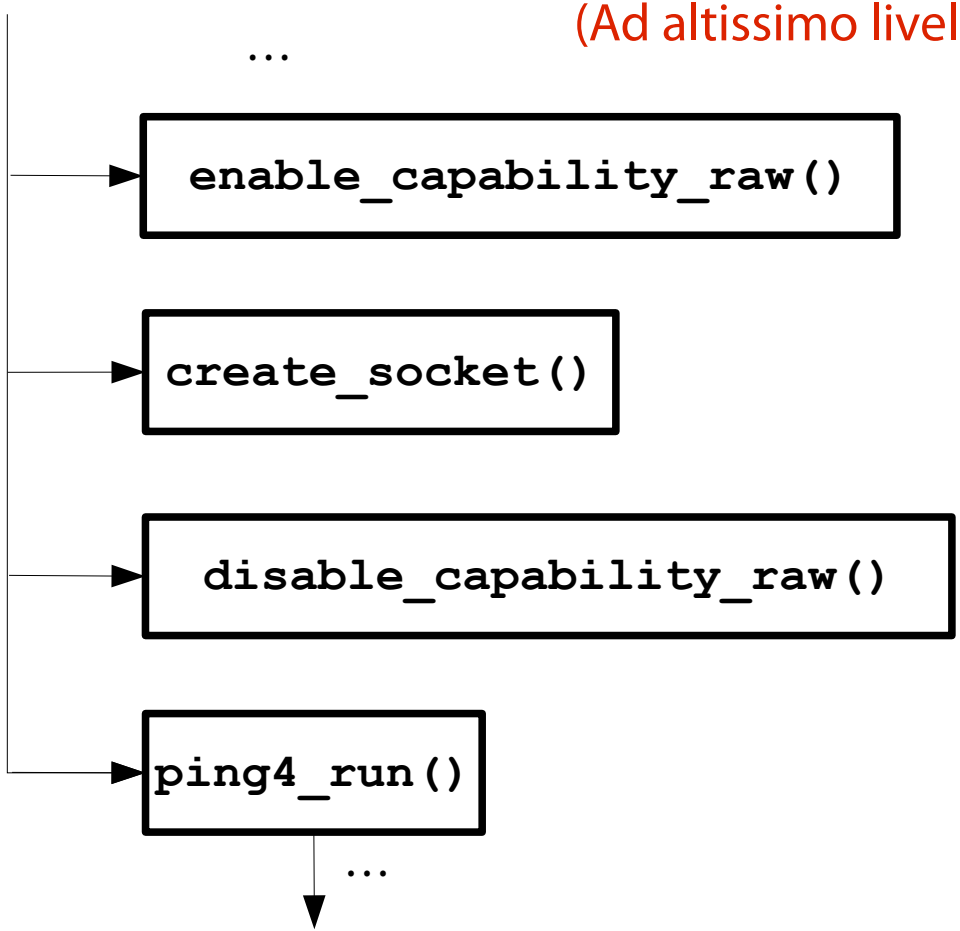
Flowchart di `main()`

(Ad altissimo livello)



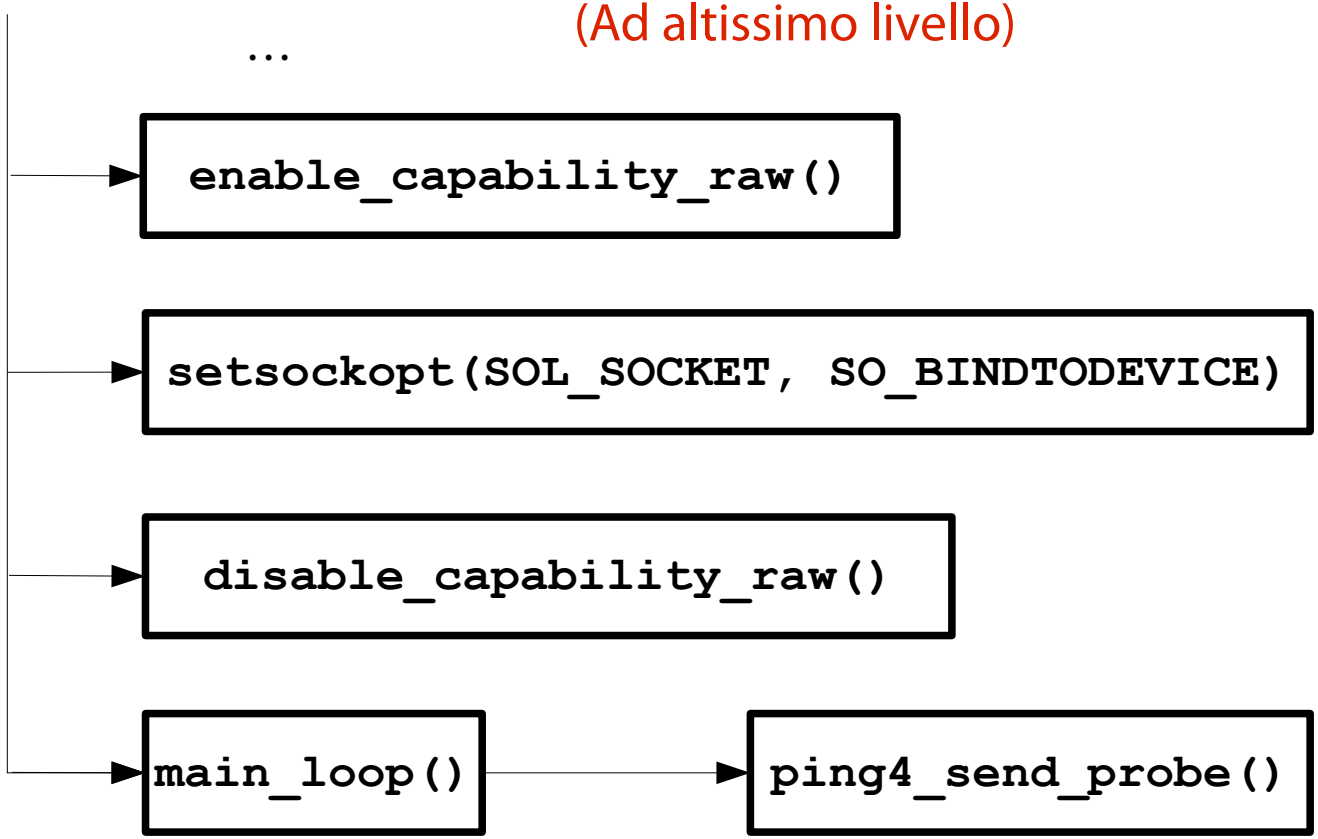
Flowchart di `main()`

(Ad altissimo livello)



Flowchart di `main()`

(Ad altissimo livello)



Riassumendo

(Al massimo)

`ping` sembra rilasciare subito i privilegi.
Li prende solo per creare un socket speciale (tramite cui sarà spedito un datagramma creato a mano).
Li rilascia subito dopo.

`limit_capabilities()`

(Che cosa fa?)

Se il Sistema Operativo supporta le capability:

recupera le capability del processo;
controlla se le capability permitted contengono i bit **`CAP_NET_RAW`** e **`CAP_NET_ADMIN`**;
se esistono, prepara una nuova maschera con le due succitate capability e la imposta nel kernel;
impedisce la cancellazione delle capability in seguito all'abbassamento permanente dei privilegi con **`setuid()`**;
…

`limit_capabilities()`

(Che cosa fa?)

Se il Sistema Operativo supporta le capability:

…

abbassa i privilegi SETUID permanentemente con la chiamata di sistema **`setuid()`**;

abilita nuovamente la cancellazione delle capability in seguito all'abbassamento permanente dei privilegi tramite **`setuid()`**.

`limit_capabilities()`

(Che cosa fa?)

Se il Sistema Operativo non supporta le capability:

- abbassa temporaneamente i privilegi tramite la chiamata di sistema **`seteuid()`**.

`enable_capability_raw()`

(Che cosa fa?)

Se il Sistema Operativo supporta le capability:

abilita la capability **`CAP_NET_RAW`** tramite la funzione wrapper **`modify_capability()`** a due parametri.

Se il Sistema Operativo non supporta le capability:

eleva i privilegi a quelli dell'utente effettivo tramite la funzione wrapper **`modify_capability()`** ad un parametro.

`disable_capability_raw()`

(Che cosa fa?)

Se il Sistema Operativo supporta le capability:

disabilita la capability **`CAP_NET_RAW`** tramite la funzione wrapper **`modify_capability()`** a due parametri.

Se il Sistema Operativo non supporta le capability:

abbassa i privilegi a quelli dell'utente reale tramite la funzione wrapper **`modify_capability()`** ad un parametro.

`modify_capability()`

(Che cosa fa?)

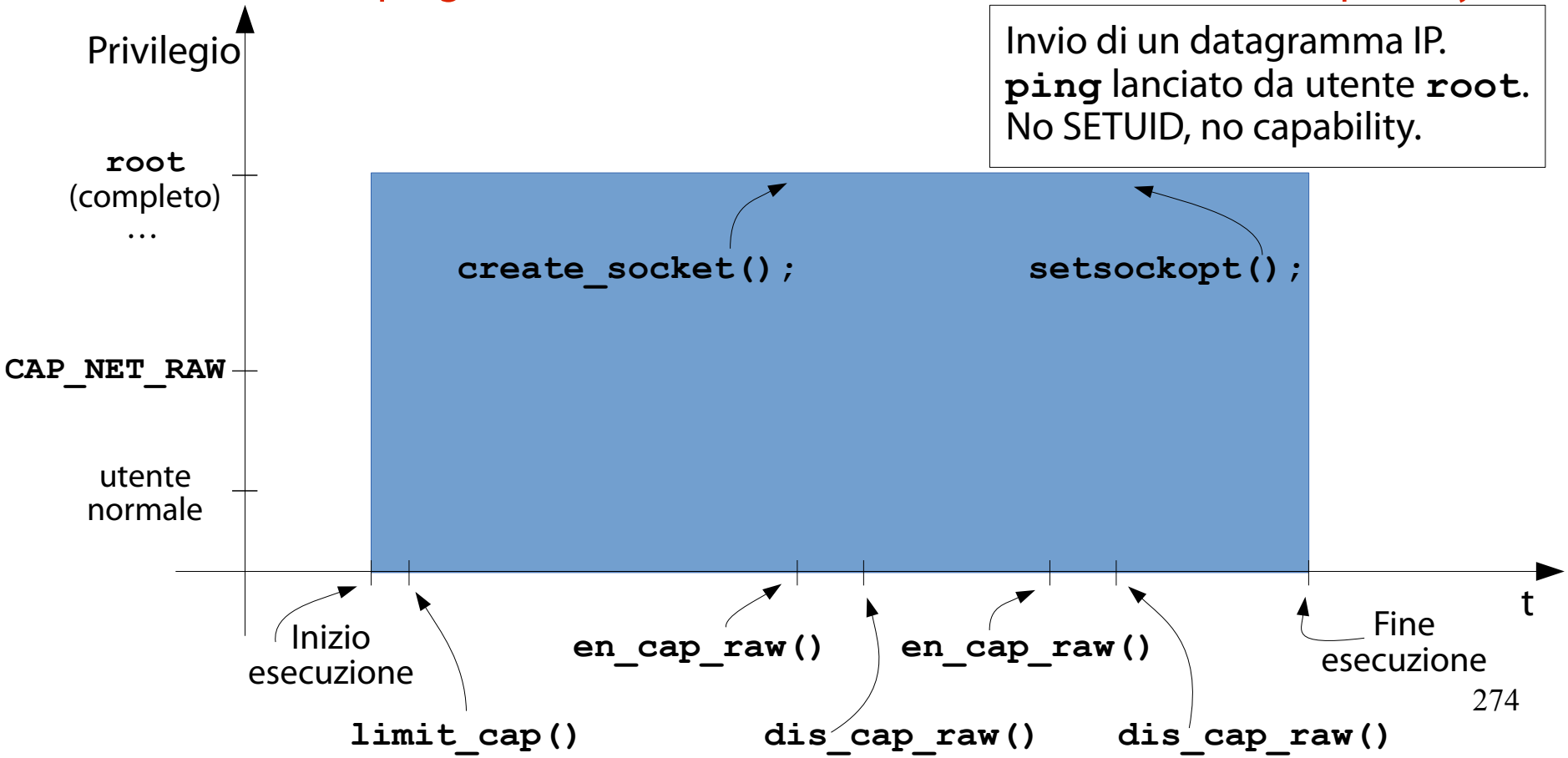
La **`modify_capability()`** è una macro che accetta uno o due parametri a seconda del tipo di meccanismo di elevazione dei privilegi usato.

Se sono definite le capability, usa l'API delle capability per impostare o rimuovere uno specifico bit.

Se non sono definite le capability, usa l'API classica per abbassare o ripristinare temporaneamente i privilegi.

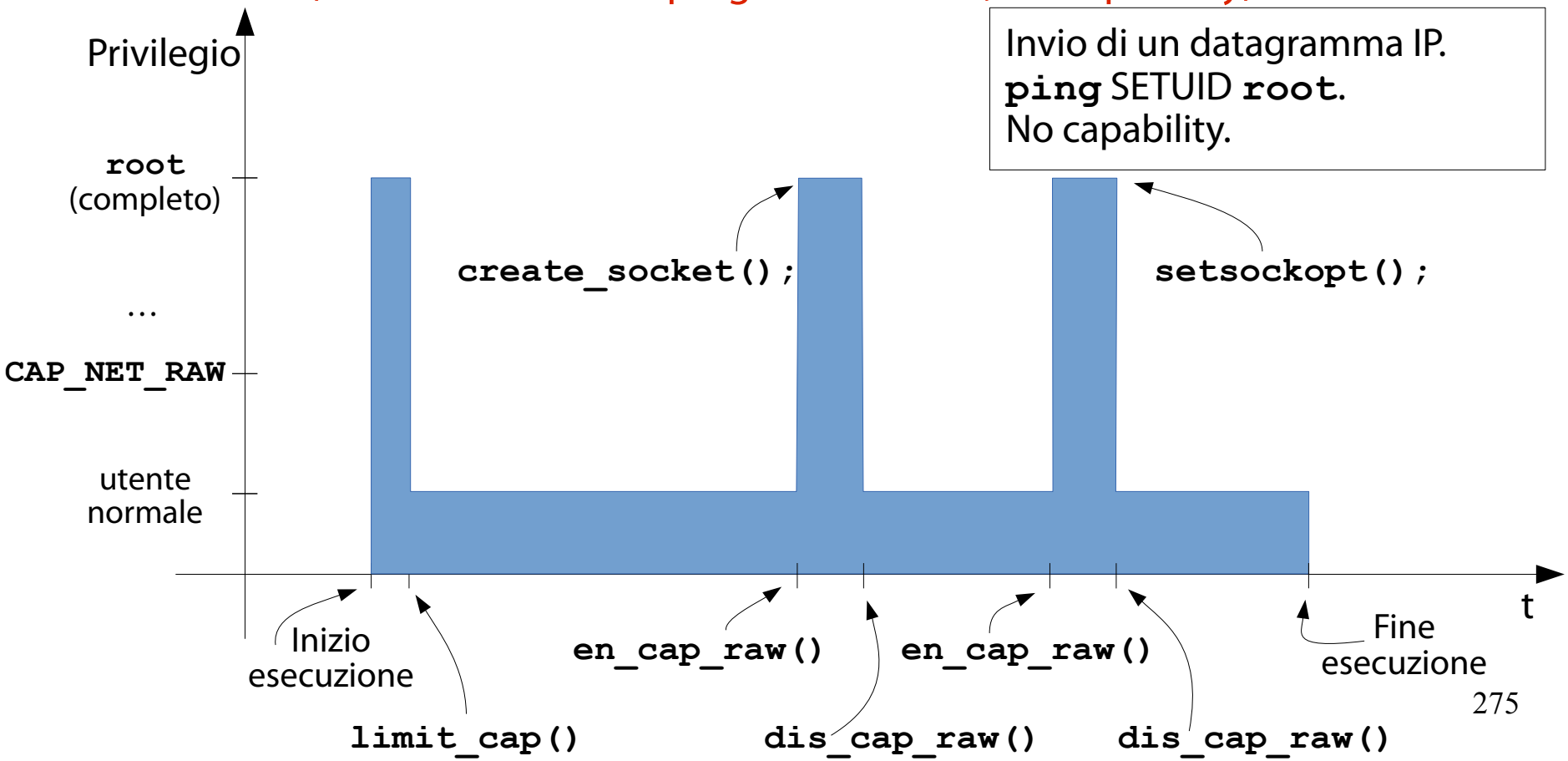
Andamento del privilegio

(Primo scenario: ping lanciato da utente root, non SETUID root, no capability)



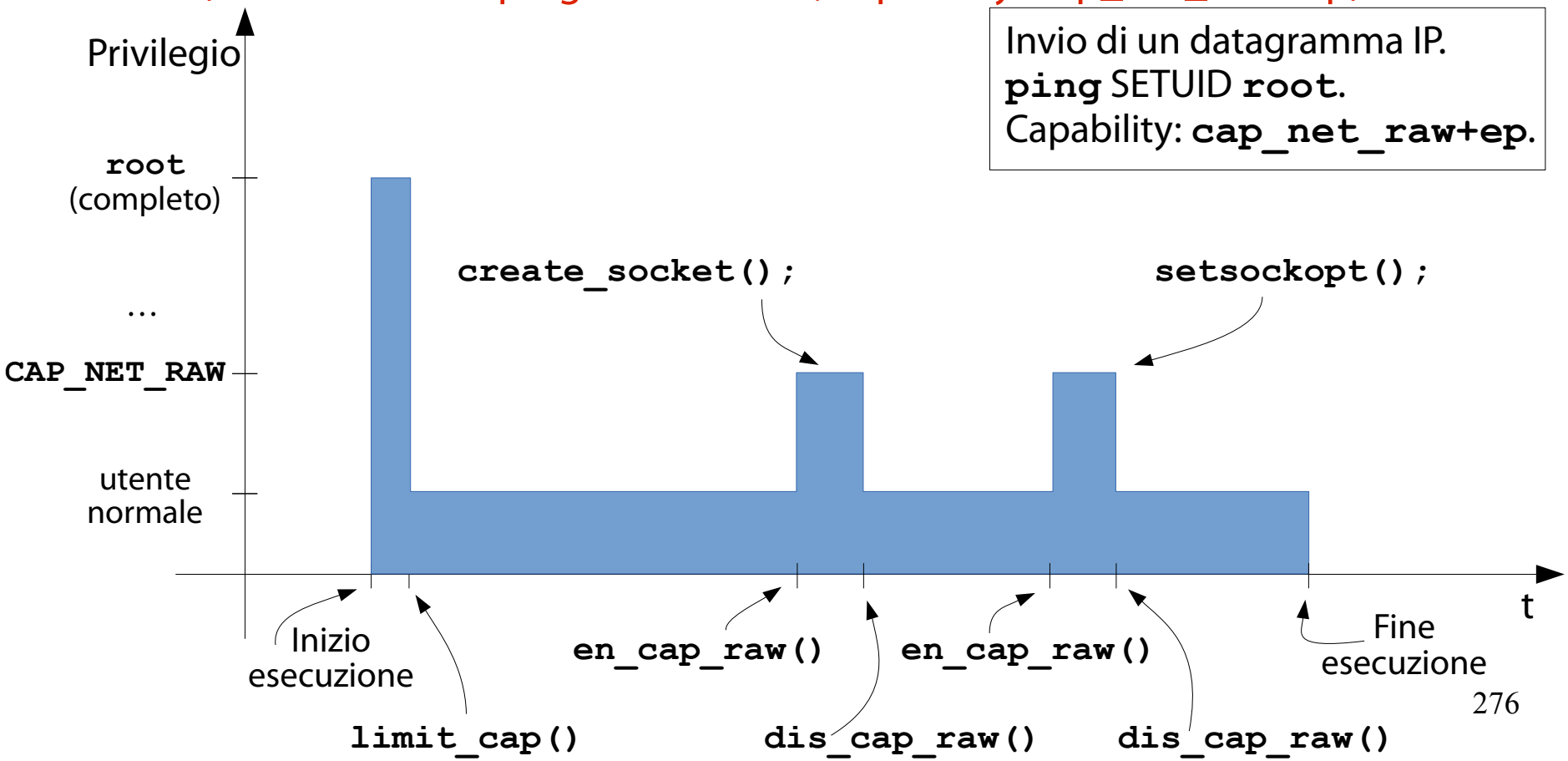
Andamento del privilegio

(Secondo scenario: ping SETUID root, no capability)



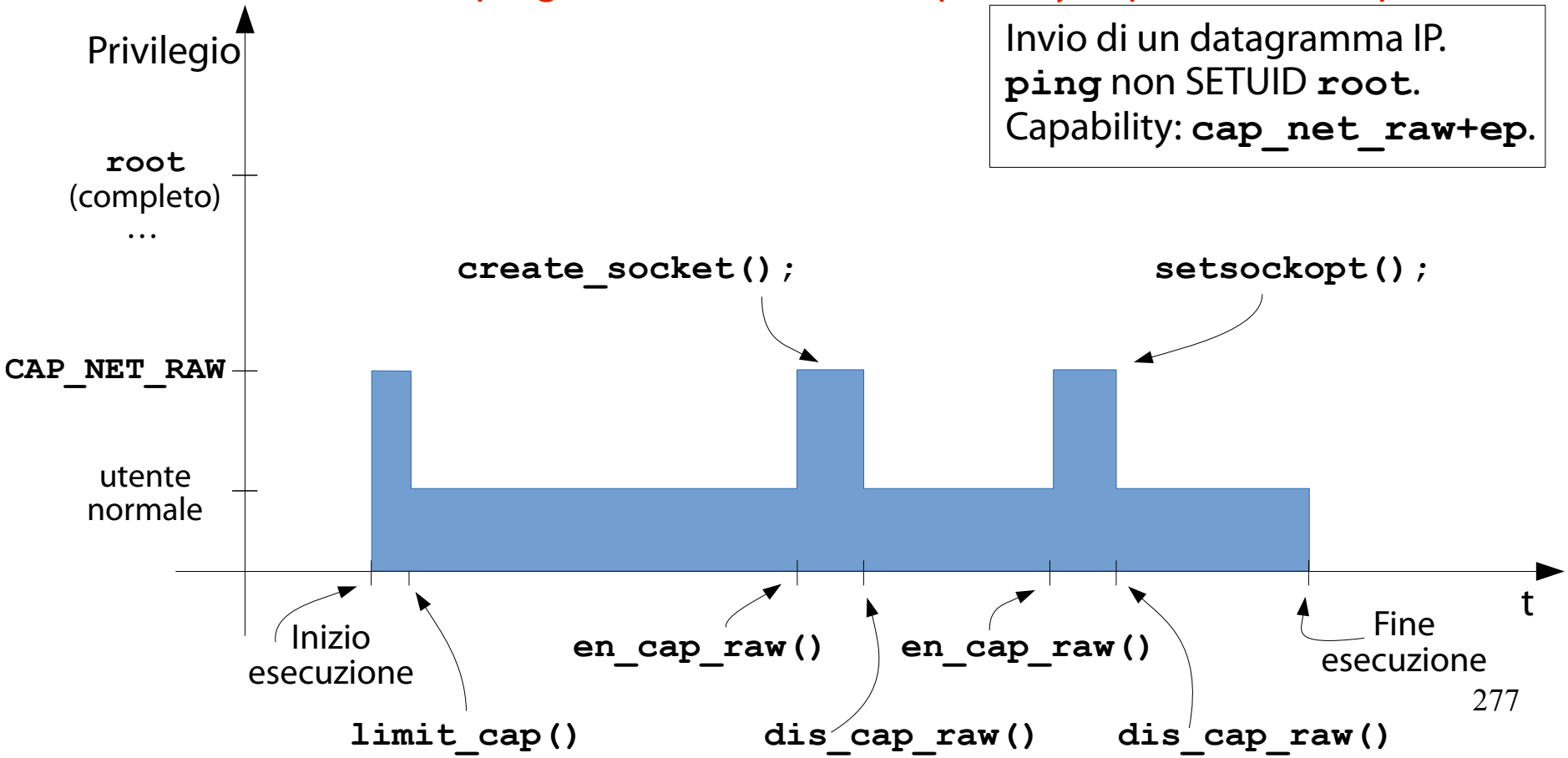
Andamento del privilegio

(Terzo scenario: ping SETUID root, capability cap_net_raw+ep)



Andamento del privilegio

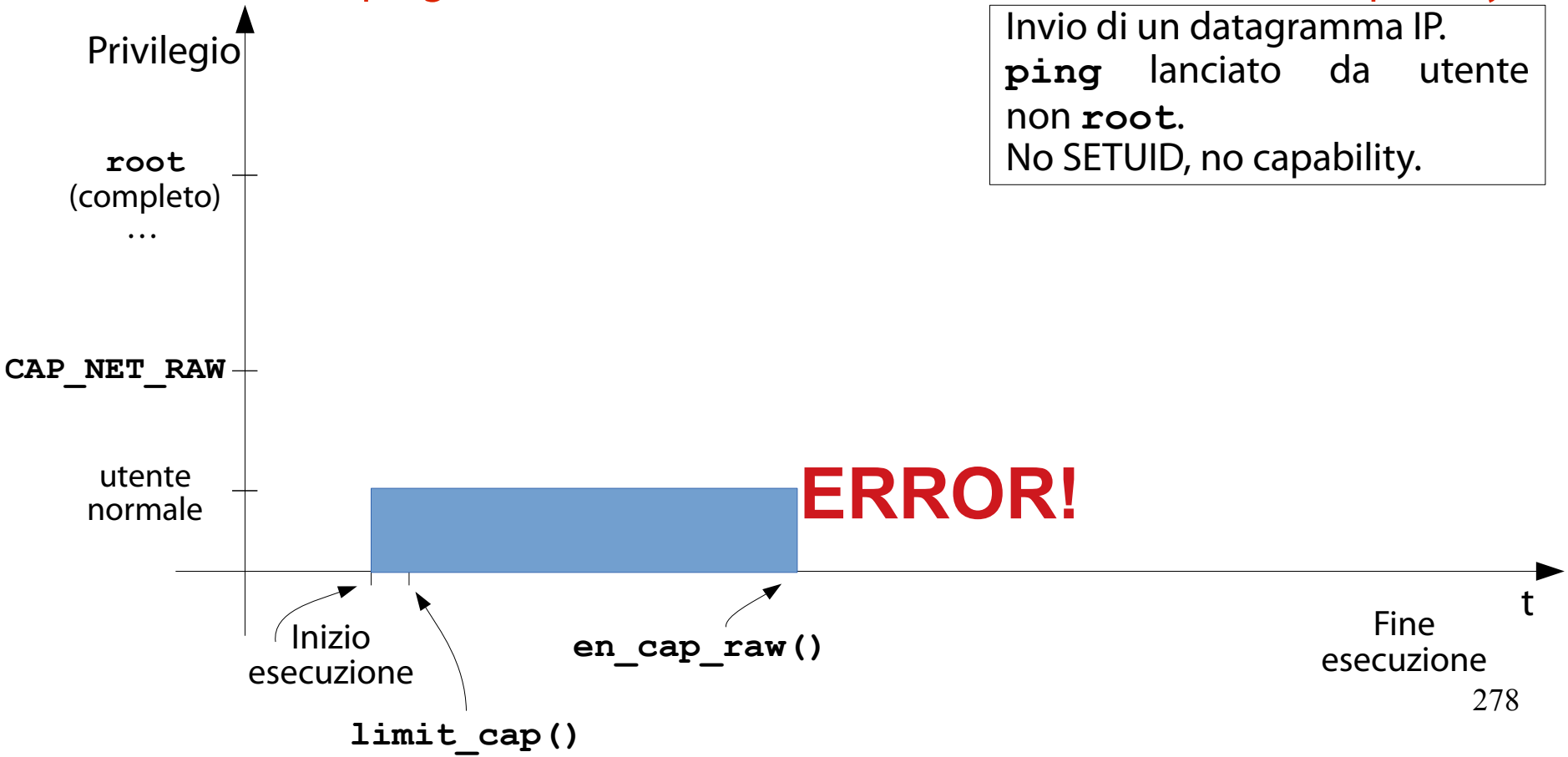
(Quarto scenario: ping non SETUID root, capability cap_net_raw+ep)



Andamento del privilegio

(Quinto scenario: ping lanciato da utente ≠ root, non SETUID root, no capability)

Invio di un datagramma IP.
`ping` lanciato da utente non **`root`**.
No SETUID, no capability.



Un esercizio per casa

(Facoltativo, per chi si vuole divertire)

Si analizzi la gestione dei privilegi del comando **`dumpcap`**.