DISCLAIMER

Il materiale contenuto nel drive è stato raccolto e richiesto tramite autorizzazione ai ragazzi frequentanti il corso di studi di Informatica dell'Università degli Studi di Salerno. Gli appunti e gli esercizi nascono da un uso e consumo degli autori che li hanno creati e risistemati per tanto non ci assumiamo la responsabilità di eventuali mancanze o difetti all'interno del materiale pubblicato.

Il materiale sarà modificato aggiungendo il logo dell'associazione, in tal caso questo possa recare problemi ad alcuni autori di materiale pubblicato, tale persona può contattarci in privato ed elimineremo o modificheremo il materiale in base alle sue preferenze.

Ringraziamo eventuali segnalazioni di errori così da poter modificare e fornire il miglior materiale possibile a supporto degli studenti.

CoScienze
Associazione

Alfonso Cannavale

ESECUZIONE CON PRIVILEGI ELEVATI

Nei sistemi UNIX può capitare che un processo necessiti di **privilegi elevati** (privilegi di root). Alcuni comandi che richiedono permessi di root sono:

- *passwd*, per modificare il file /etc/passwd;
- *ping*, per aprire socket e porte;
- *mount*, per modificare file system residenti su memorie di massa removibili.

L'elevazione dei privilegi può essere effettuata in due modi:

- **Manuale**
  1. L'utente digita i comandi opportuni per diventare un amministratore (questo passo può essere eseguito solo qualora l'utente è a conoscenza della password dell'admin);
  2. L'utente esegue il comando che gli interessa.

```
$ su –
Password:
$ passwd masucci
Changing password for user masucci.
New password:
Retype new password:
passwd: all authentication tokens updated succesfully
```

  Il comando su consente a un utente eseguire comandi con i privilegi di un altro utente, in questo caso l'utente root. La procedura richiede la conoscenza della password di root ed una volta acquisiti i privilegi di root, l'utente può modificare la sua password (che sarà scritta nel file /etc/passwd).
  Con questo sistema è *necessario conoscere la password di root*. Dopo aver acquisito i privilegi, l'utente può ispezionare il sistema, modificare file e compromettere servizi. Inoltre, è necessario immettere la password di root ogni volta che serve acquisire privilegi elevati. C'è il rischio che per agevolare l'operazione, si decida di non impostare alcuna password per la root, in tal caso, chiunque accede al terminale può diventare amministratore.
  La modalità manuale ha il vantaggio che bisogna conoscere la password per consentire l'accesso e svolgere azioni.
- **Automatica** (*tipicamente usata in UNIX*)
  1. L'utente esegue il comando che gli interessa;
  2. Il SO esegue l'elevazione dei privilegi.

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

  Questa modalità può portare dei problemi se associata a delle debolezze. L'utente richiede di modificare la password mediante il comando *passwd*. Il SO effettua l'elevazione automatica ai privilegi di root, consentendo all'utente di modificare la password (che sarà scritta nel file /etc/passwd).

ORGANIZZAZIONE DEL FILE SYSTEM

Per capire perché il comando *passwd* faccia ottenere i privilegi di root all'utente, bisogna rivedere come è *organizzato il file system* nei sistemi UNIX.

Utenti e gruppi

In UNIX, un **utente** è caratterizzato da:

- *username* (stringa);
- *UID - User IDentification number* (intero), la root è l'utente con UID=0.

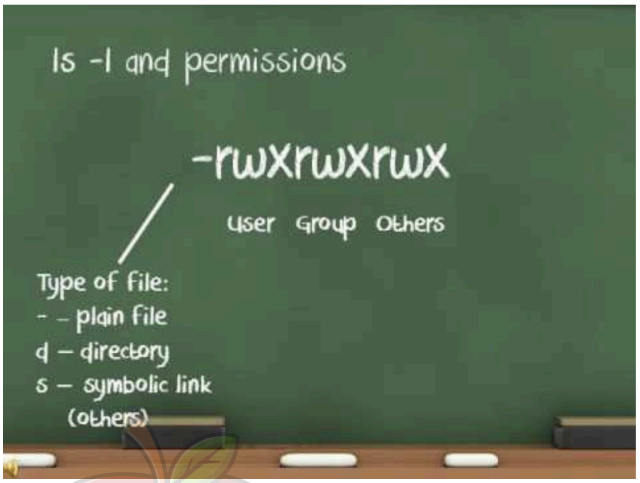
Un **gruppo** (collezioni di utenti con autorizzazione e privilegi simili) è caratterizzato da:

- *groupname* (stringa);
- *GID - Group IDentification number* (intero).

Queste informazioni possono essere identificate con il comando *id*.

File e permesso
L'accesso ai file è regolato da ***permessi***, definiti come tre terne di azioni -> una terna per ciascuna tipologia di utenti, dove le tipologie di utenti sono: (i) *proprietario* (colui che ha creato il file o la directory), (ii) *gruppo di lavoro* (gruppo a cui appartiene il file o la directory) e (iii) *altri utenti* (tutti gli altri utenti che non sono né il proprietario né fanno parte del gruppo di lavoro del file o della directory).
In un sistema UNIX, le azioni che possono essere eseguite su un file sono: ***R**ead* (), ***W**rite* ed *e**X**ecute*. Queste azioni definiscono i permessi di accesso per il file e possono essere assegnate separatamente per il proprietario del file, il gruppo di lavoro e gli altri utenti.

- Nota: eXecute su un file indica che il file può essere eseguito, mentre, eXecute su una directory indica che si può entrare in essa.



rwxrwxrwx: questi nove caratteri rappresentano i permessi del file. Si dividono in tre gruppi di tre lettere, ognuno dei quali rappresenta un tipo di utente: rwx per lo user, rwx per il gruppo e rwx per gli altri utenti.

Type of file, indica il tipo di file:

- \- (trattino): File normale.
- d (directory): Directory.
- s (link simbolico): Collegamento simbolico ad un altro file.

Rappresentazione dei permessi
I permessi vengono rappresentati con:

- ***Rappresentazione ottale***
  - Read → 4, Write → 2, eXecute → 1
  - Una terna di azioni viene rappresentata come una somma di permessi.
  - Esempio: rwxrwxr-x → 4+2+1 (user) 4+2+1 (gruppo) 4+1 (other) → 775
- ***Rappresentazione simbolica***
  - Read → r, Write → w, eXecute → x
  - Creatore (user) → u, Gruppo → g, Altri → o
  - Ogni modifica ai permessi è indicata con: tipologia di utente ± azioni, ad esempio u+rw aggiunge i permessi di lettura e scrittura al creatore.
  - La differenza tra l'utilizzo del + e del meno nel permesso finale è che il + aggiunge i permessi specificati, mentre il meno li rimuove.
  - I permessi finali sono separati da una virgola, ad esempio ug+rwx, o+rx. Ad esempio, rwxrwxr-x può essere rappresentato come ug+rwx,o+rx, il che significa che il creatore e il gruppo hanno tutti i permessi (lettura, scrittura ed esecuzione), mentre gli altri utenti hanno solo il permesso di lettura ed esecuzione.

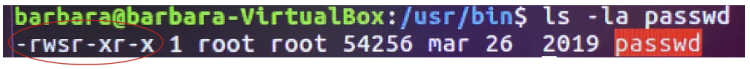
I permessi possono essere impostati con il comando: *chmod nuovi-permessi nome-file*. Esempio, se si esegue *chmod 777 nomefile*, vengono dati tutti i permessi possibili su quel file.

Si evince quindi, che a ciascun file sia associata una ***stringa di permessi*** costituita da 9 bit. In realtà, ci sono altri tre bit aggiuntivi associati ai permessi: **SETUID**, **SETGID**, STICKY, che portano da 9 a 12 bit.
Quando SETUID e SETGID sono posti uguali a 1, **consentono una elevazione dei privilegi**. La rappresentazioni *ottale/simbolica* è: per ***SETUID: 4/s***, mentre per ***SETGID: 2/s***.

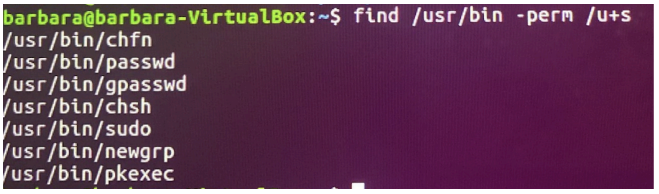
Impostare SETUID

Per impostare a 1 il bit SETUID di un file, si utilizza il comando: *chmod u+s file*.

- Per stabilire se questo bit è settato a 1 (è "accesso") per un determinato file, si invoca il comando *ls -l nomefile* → in output se è presente il permesso "s" nella parte relativa ai permessi dell'utente e de il file è evidenziato in rosso allora SETUID è impostato a 1.



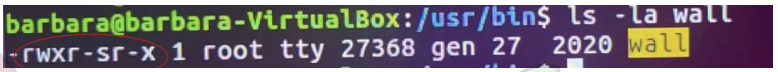
- Per individuare tutti i file con il bit SETUID accesso è possibile utilizzare il comando *find* con l'opzione *-perm* (filtro in base ai permessi).
  - Esempio: *find / -perm / u+s*
  - Per evitare di visualizzare i messaggi di errore (permission denied): *find / -perm / u+s 2>/dev/null*



Impostare SETGID

Per impostare a 1 il bit SETGID di un file, si utilizza il comando: *chmod g+s file*.

- Per stabilire se il bit è settato a 1 → *ls -l nomefile* → se è presente il permesso "s" tra i permessi del gruppo e se il file è evidenziato di giallo allora SETGID è impostato a 1.



- Per individuare tutti i file con il bit SETGID accesso → *find / -perm / g+s*.

Utente e gruppo reale e effettivo

Nei SO UNIX, il descrittore di un processo memorizza **una prima coppia di credenziali**:

- ***Real User ID (RUID)*** è l'UID di chi ha lanciato il comando;
- ***Real Group ID (RGID)*** è l'GID di chi ha lanciato il comando.

Ed una **seconda coppia di credenziali**:

- ***Effective User ID (EUID)***;
- ***Effective Group ID (EGID)***.

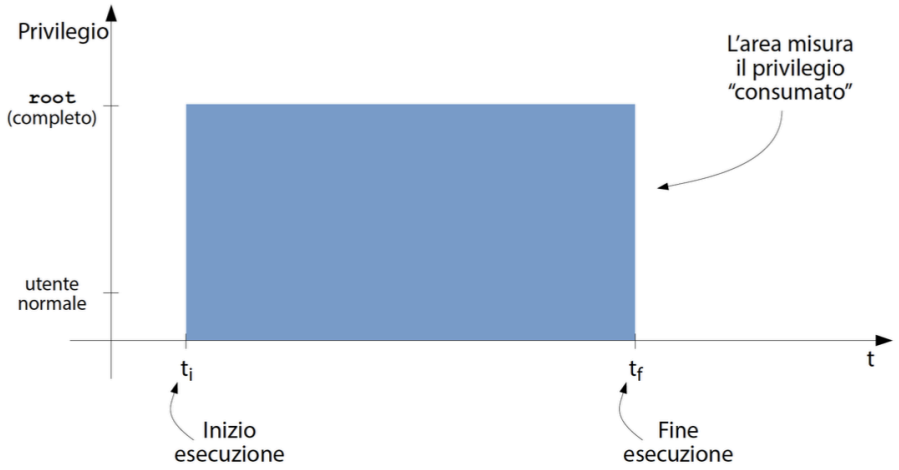
In condizioni normali, quando SETUID e SETGID **disattivati**, queste coppie, *real User/Group ID e effective User/Group ID*, **coincidono**:

- EUID è l'UID *di chi ha lanciato il comando*;
- EGID è il GID di *chi ha lanciato il comando*.

Se invece, i bit SETUID E SETGID sono **attivati**, ha importanza l'effettività poichè:

- EUID è l'UID *di chi possiede il file*;
- EGID è il GID *di chi possiede il file*.

Quando i bit SETUID o SETGID sono attivi su un file eseguibile in un sistema UNIX, chi avvia il comando può temporaneamente assumere i privilegi dell'utente o del gruppo proprietario del file → **un processo con SETUID/SETGID attivi ottiene i pieni poteri dell'utente privilegiato** (se l'utente privilegiato è root, il processo può fare di tutto). Se il bit SETUID è attivo, il processo può ottenere i pieni poteri dell'utente proprietario del file, inclusi i privilegi di root. Questo consente al processo di eseguire operazioni con autorizzazioni elevate, come modificare file di sistema o avviare altri programmi con accesso speciale. Il privilegio ottenuto viene mantenuto per l'intera esecuzione del processo, consentendo al processo di eseguire qualsiasi operazione per cui ha ottenuto privilegi elevati.



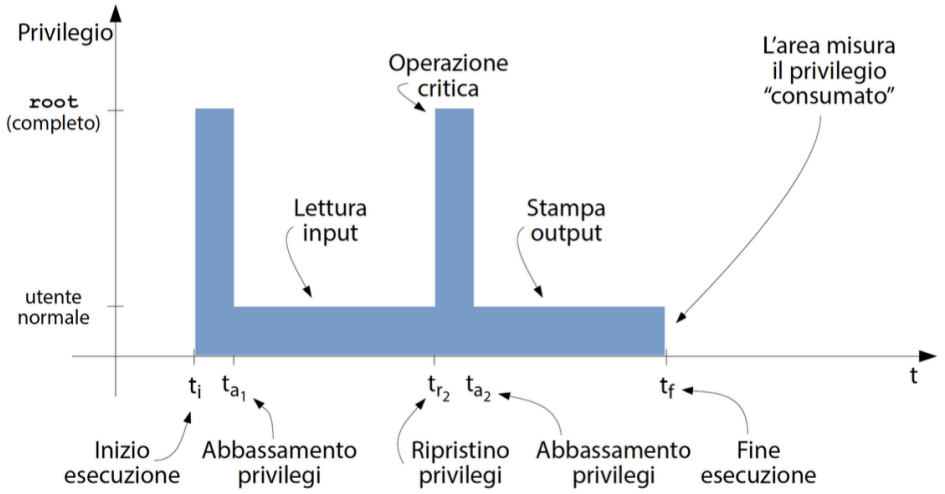
Problemi e debolezze dei privilegi elevati

L'elevazione automatica dei privilegi è una funzionalità interessante offerta da UNIX. Tuttavia, il conferimento dei pieni poteri di *root*, quando non sono strettamenti necessari, è una **debolezza**. Anche il mantenimento dei privilegi per tutta la durata dell'esecuzione è una debolezza.

NON si tratta di una **vulnerabilità**, perché le debolezze non sono sfruttabili da sole! Questo perché, se un programma è scritto in modo corretto, la sola elevazione automatica non può dare alcun vantaggio a un attaccante. Tuttavia, se unita ad altre debolezze, l'elevazione automatica dei privilegi può avere ***conseguenze devastanti*** per la sicurezza di un programma.

Come risolvere? Per risolvere i problemi legati ai privilegi elevati nei sistemi UNIX, si possono adottare le seguenti pratiche:

1. **Abbassamento dei privilegi**: se l'applicazione non svolge operazioni critiche che richiedono privilegi elevati, può decidere di abbassare i propri privilegi a quelli dell'utente che ha avviato il comando. Questo processo, conosciuto come ***privilege drop***, limita l'impatto delle potenziali vulnerabilità e protegge il sistema da possibili attacchi.
2. **Ripristino dei privilegi**: quando l'applicazione deve svolgere operazioni critiche che richiedono privilegi elevati, può temporaneamente ottenere tali privilegi tramite un processo di elevazione automatica. Tuttavia, è importante che l'applicazione ripristini nuovamente i privilegi al loro livello originale una volta completata l'operazione critica. Questo processo è noto come ***privilege restore*** e aiuta a garantire che i privilegi elevati vengano utilizzati solo quando strettamente necessario, riducendo così il rischio di abusi o violazioni di sicurezza.



Privilege drop and restore
Come si implementano l'*abbassamento* ed il *ripristino dei privilegi* di un programma? Ci sono delle **chiamate di sistema** che assolvono al compito che variano per i diversi sistemi: i primissimi sistemi UNIX, i sistemi UNIX System V, i sistemi UNIX BSD, i sistemi POSIX e i sistemi UNIX moderni.

Primissimi sistemi UNIX
Nei primissimi sistemi UNIX sono previste solo due tipologie di user e group ID:

- RUID e RGID;
- EUID e EGID.

I valori RUID e EUID possono essere determinati mediante le chiamate di sistema:

- ***getuid()*** → che restituisce il RUID del processo invocante;
- ***geteuid()*** → che restituisce l'EUID del processo invocante.

È anche possibile *cambiare* il valore dell'EUID di un processo. La chiamata di sistema *setuid(uid)* imposta l'EUID del per processo al valore *uid* in input. Ad esempio, *setuid(0)*, imposta EUID a 0 (*root*). In caso di errore, l'output della chiamata è -1.
Per effettuare un *abbassamento dei privilegi* basta eseguire il comando *setuid(getuid)*. In tal modo infatti, l'EUID del processo viene settato uguale al RUID.

Nei primissimi sistemi UNIX, l'abbassamento dei privilegi tramite lo statement *setuid(getuid())* (il processo sta cercando di abbassare i suoi privilegi a quelli dell'utente effettivo che lo ha avviato) non permette più il ripristino del privilegio elevato che si aveva in precedenza. Si tratta quindi di un ***abbassamento permanente dei privilegi***.

- Quindi, se un processo utilizzava setuid(getuid()) per abbassare i suoi privilegi, non era in grado di tornare al suo stato di privilegio elevato originale. Questo rendeva tale abbassamento dei privilegi permanente e irreversibile per quel processo.

Supponiamo che un processo
- Parta SETUID `root`
- Abbassi i suoi privilegi con `setuid(getuid())`
- Invochi `setuid(0)` per ripristinare i privilegi di `root`
- Viene generato un errore

Sistemi UNIX SystemV
La possibilità di un ***abbassamento temporaneo dei privilegi*** viene gestita nei sistemi UNIX System V → viene introdotta una nuova chiamata di sistema: *seteuid(uid)*.
Analogamente alla vecchia *setuid(uid)* anche la nuova chiamata di sistema *seteuid(uid)* imposta l'EUID del processo al valore *uid* → per effettuare un abbassamento dei privilegi basta eseguire il comando *seteuid(getuid())*.

Supponiamo che un processo
- Parta SETUID `root`
- Abbassi i suoi privilegi con `seteuid(getuid())`
- Invochi `seteuid(0)`

Allora `seteuid(0)`
- Termina correttamente, perchè `root` è l'utente effettivo
- Imposta l'EUID al valore 0 (`root`)

E' un ripristino di privilegi!

Ci sono quindi *due differenti modalità* di abbassamento dei privilegi:

- **permanente** → *setuid(getuid());*
- **temporaneo** → *seteuid(getuid()).*

Sistemi UNIX BSD

Nei sistemi BSD viene introdotta una chiamata di sistema per *impostare contemporaneamente RUID e EUID*: *setreuid(uid, euid)*. Per lasciare inalterato uno dei due valori, basta fornire in input -1.

L'*abbassamento permanente* dei privilegi si ottiene impostando entrambi i parametri ad un valore non privilegiato: *setreuid(uid, uid)*. In seguito, i parametri RUID e EUID non potranno che assumere il valore *uid* → il ripristino a *root* è impossibile.

L'*abbassamento temporaneo* dei privilegi si ottiene con la chiamata: *setreuid(geteuid(), getuid())* → Nota che RUID e EUID sono scambiati e in questo modo: RUID è da *utente privilegiato* e EUID è da *utente non privilegiato*.
Il *ripristino dei privilegi* si ottiene ancora con la chiamata: *setreuid(geteuid(), getuid())* → dopo lo scambio RUID è da *utente non privilegiato* e EUID è da *utente privilegiato*.

Sistemi POSIX

POSIX (Portable Operating System Interface) è una famiglia di standard specificati dalla IEEE Computer Society per mantenere la compatibilità tra diversi SO. È stato rilasciato nel 1988 con il nome IEEE 103 o ISO/IEC 9945, lo standard definisce: (i) API di sistema, (ii) interfaccia da linea di comando e (iii) applicazioni di base.

Nei sistemi POSIX la chiamata di sistema ~~*setreuid(uid, uid)*~~ è stata deprecata e viene sostituita da *setuid(uid)*. Anche nei sistemi POSIX si hanno due differenti modalità di abbassamento dei privilegi:

- Permanente: *setuid(getuid())*
- Temporaneo: *seteuid(getuid())*.

Sistemi UNIX moderni

I sistemi operativi GNU/Linux implementano la **LSB (Linux Standards Base)**, che rappresenta un'evoluzione della SUS (Single UNIX Specification), la quale a sua volta costituisce lo standard POSIX. La LSB sostituisce alcune funzionalità considerate errate o insicure presenti in POSIX/SUS con meccanismi propri, garantendo così una maggiore coerenza e affidabilità nel sistema operativo GNU/Linux.

A partire dai SO UNIX System V è presente una terza coppia di credenziali:

- ***Saved User ID (SUID)***
- ***Saved Group ID (SGID)***
- Quando un processo parte **SUID=EUID** (chi ha lanciato il file).

Non vengono però fornite chiamate di sistema per gestire SUID e SGID → tale limite viene superato nei sistemi UNIX moderni.

La chiamata *getresuid(uid, euid, suid)* consente di recuperare tutti gli user ID del processo invocante. La chiamata di sistema *setresuid(uid, euid, suid)* consente di impostare tutti gli user ID del processo invocante.

L'***abbassamento permanente*** dei privilegi si ottiene impostando tutti i parametri ad un valore non privilegiato: *setresuid(uid, uid, uid);* → in seguito, i parametri RUID, EUID e SUID non potranno che assumere il valore *uid* → il ripristino a *root* è impossibile.

L'**abbassamento temporaneo** dei privilegi si ottiene impostando EUID ad un valore non privilegiati: *setresuid(-1, geteuid(), -1);* → in questo modo si preserva lo user ID salvato e si può effettuare il ripristino in seguito → NB: *setresuid(-1, uid, -1) = seteuid(uid)*.

- Utilizzando la chiamata *setresuid(-1, geteuid(), -1)*, stiamo impostando l'Effective User ID (EUID) del processo a un valore non privilegiato mentre manteniamo il valore salvato del User ID. Questo ci permette di effettuare un abbassamento temporaneo dei privilegi, poiché il processo ora esegue con autorizzazioni ridotte.
- Inoltre, è importante notare che *setresuid(-1, uid, -1)* è equivalente a *seteuid(uid)*. Questo significa che possiamo utilizzare *seteuid()* per impostare l'Effective User ID (EUID) a un valore specifico, consentendo un controllo più preciso sui privilegi del processo.

Il **ripristino temporaneo dei privilegi** si ottiene impostando EUID ad un valore privilegiato: *setresuid(-1, privileged_ID, -1);* → dove *privileged_ID* è lo user ID dell'utente privilegiato ottenuto in partenza (tipicamente *root*).

Elevazione dei privilegi via SETGID

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

`getgid()` restituisce l'RGID del processo
`getegid()` restituisce l'EGID del processo
`setgid(gid)` imposta l'EGID del processo
`setegid(egid)` imposta l'EGID del processo
`getresgid(gid,egid,sgid)` restituisce RGID, EGID e SGID del processo
`setresgid(gid,egid,sgid)` imposta RGID, EGID e SGID del processo

Inibire SETUID e SETGID

Il meccanismo dei privilegi basati su SETUID e SETGID **forniscono i pieni poteri di *root*** → per tale motivo si preferisce inibirlo il più possibile. L'obiettivo è quello di impedire l'esecuzione di shell con privilegi di *root*.

L'opzione *nosuid* inibisce l'elevazione dei privilegi tramite SETUID e SETGID. Quando questa opzione è attiva, i bit setuid e setgid non avranno effetto sui file presenti nel file system. Di conseguenza, i processi eseguiti da questi file non saranno in grado di ottenere temporaneamente privilegi elevati → tutti i *file system virtuali e temporanei* sono montati con tale opzione.

- Quindi, l'"opzione "nosuid" aiuta a mitigare potenziali minacce alla sicurezza limitando la capacità dei processi di ottenere privilegi elevati tramite setuid e setgid su determinati file system.

Di default, molte shell moderne inibiscono l'elevazione dei privilegi tramite SETUID e SETGID. In tal modo si evitano attacchi in grado di provocare *esecuzioni di codice arbitrario*.

- Ad esempio, quando BASH esegue uno script con privilegi elevati ne *abbassa i privilegi* a quelli dell'utente che ha invocato la shell.

Ad esempio, si copi `/bin/bash` in una directory non montata `suid`

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

La si renda SETUID `root`

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

La si esegua e si stampino gli ID reale ed effettivo

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

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

- `bash` stampa RUID, RGID, EUID, EGID dell'utente (normale) che lo ha lanciato
- Non vi è traccia di ID privilegiati
- Ciò implica l'abbassamento dei privilegi da parte di `bash`

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

```
#!/bin/bash
      id –u ← stampa EUID
      id –g ← stampa EGID
```

Lo si renda SETUID `root`

```
sudo chown root ./scr.sh
sudo chmod u+s ./scr.sh
```

Lo si esegua

```
./scr.sh
```

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

- Lo script `scr.sh` stampa EUID, EGID 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`

E' possibile inibire l'abbassamento dei privilegi con l'opzione `–p` di BASH

Proviamo

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

BASH esegue con i privilegi di `root`!

- Infatti mentre prima EUID era 1000, ora è 0

Nota: 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 delle credenziali di `root`

Solo se questa condizione è verificata è possibile l'elevazione dei privilegi tramite

```
bash –p
```

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

Se così non fosse si potrebbe copiare un binario SETUID in una cartella locale e modificarlo a piacimento

- Magari permettendo l'esecuzione di una shell

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

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

Se ne esaminino i permessi

```
ls –l passwd
```

Si noti che, in corrispondenza dei bit SETUID e SETGID, le `s` sono sostituite dalle `x`

```
-rwxr-xr-x 1 masucci masucci 54256 mar 10 18:29 passwd
```

Gestione dei privilegi negli altri linguaggi

Tutti i meccanismi visti finora sono stati concepiti per l'uso in **programmi scritti in linguaggio C**. NON è possibile abbassare e ripristinare i privilegi anche in altri ambienti di programmazione… Nei linguaggi di scripting dinamici, tipo Python, i bit SETUID/SETGID ***sono ignorati negli script di ogni genere***.

Nello script di seguito *drop_rest.py*, si prova ad effettuare abbassamento e ripristino dei privilegi mediante funzioni di libreria.

drop_rest.py

```
#!/usr/bin/python3

import os

(uid, euid, suid) = os.getresuid()
print("Prima dell'abbassamento privilegi:
        (uid, euid, suid) = ", (uid, euid, suid))

os.setresuid(-1, uid, -1)
(uid, euid, suid) = os.getresuid()
print("Dopo l'abbassamento privilegi:
        (uid, euid, suid) = ", (uid, euid, suid))

os.setresuid(-1, suid, -1)
(uid, euid, suid) = os.getresuid()
print("Dopo il ripristino privilegi:
        (uid, euid, suid) = ", (uid, euid, suid))
```

Si consideri lo scritpt `drop_rest.py` illustrato nella slide precedente

Lo si renda SETUID `root`

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

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

```
python3 drop_rest.py
```

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

- Lo script `drop_rest.py` stampa RUID, EUID, SUID dell'utente (normale) che lo ha lanciato
- Non vi è traccia di ID privilegiati
- Quindi il bit SETUID è stato ignorato

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

- Bisogna essere root per farlo.
- Avendo a disposizione un interprete SETUID root è possibile far gestire il privilegio all'interno dello script, tramite le funzioni di libreria fornite con l'interprete.

Ad esempio, si copi l'interprete Python in una directory locale

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

Si renda SETUID `root` l'interprete

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

Si esegua lo script `drop_rest.py`

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

Adesso l'abbassamento dei privilegi con lo script `drop_rest.py` funziona

Ciò è dovuto al fatto che l'interprete Python è SETUID `root`

Infine, l'ambiente di esecuzione di Java non supporta la gestione dei privilegi via SETUID/SETGID