

# Lezione 12

# Elevazione privilegi

Sistemi Operativi (9 CFU), CdL Informatica, A. A. 2022/2023

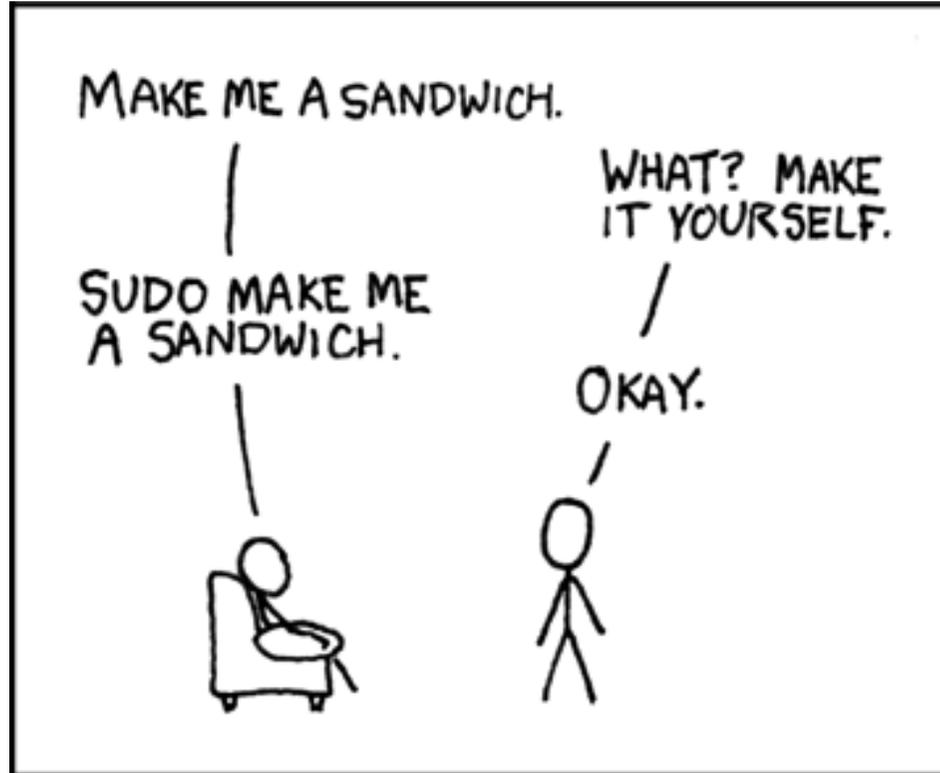
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Università di Modena e Reggio Emilia

<http://weblab.ing.unimo.it/people/andreolini/didattica/sistemi-operativi>

# Quote of the day

(<https://xkcd.com/149/>)



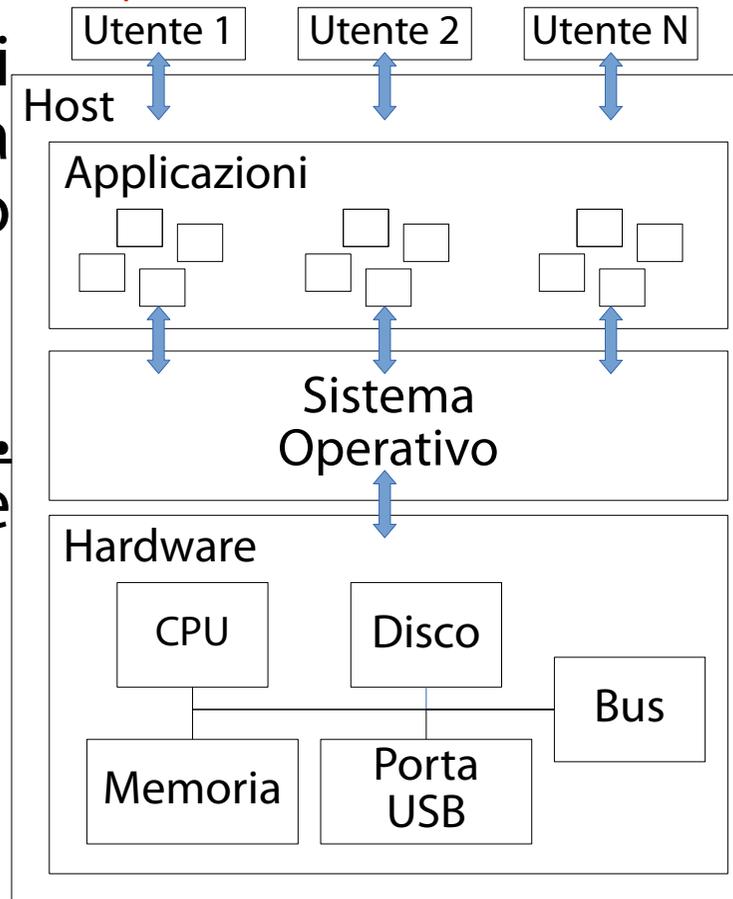
# INTRODUZIONE

# Lo scenario

(Sistema multi-utente)

**Molteplicità di utenti.** Più utenti possono usare i servizi messi a disposizione dall'host nello stesso istante.

**Principio del minimo privilegio.** Ispira le decisioni del SO nelle assegnazioni delle risorse.



# Interrogativi 1/1

(Si possono eseguire operazioni con privilegi specifici?)

Alcune operazioni critiche sono impedito agli utenti normali.

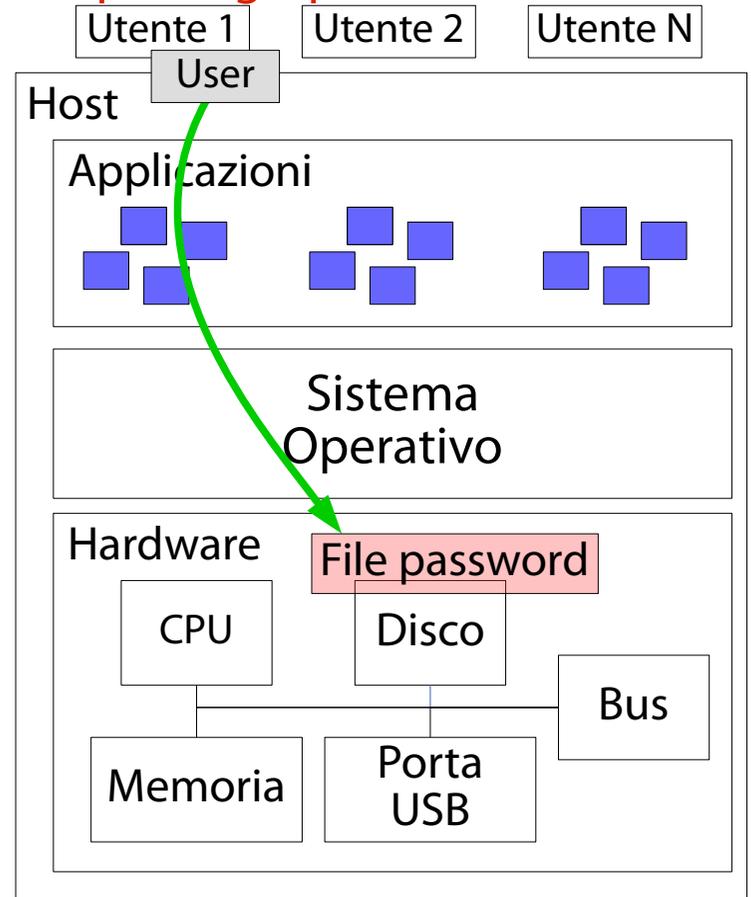
Accensione periferica.

Formattazione disco.

Modifica password.

Tuttavia, un utente vorrebbe poter eseguire tali operazioni (pena un'esperienza d'uso fortemente degradata).

Può l'utente eseguire comandi con privilegi specifici, in maniera controllata?



# Cambio di privilegio

(Permette di eseguire comandi con le credenziali di altri utenti)

I SO moderni forniscono astrazioni software per il **cambio di privilegio**.

Casi tipici:

un utente lancia un processo (spesso, **bash**) con i privilegi di un altro utente (spesso, **root**).

un utente lancia un processo che assume automaticamente privilegi superiori.

# Avvio processo con privilegi diversi

(Banale dal punto di vista teorico)

L'avvio di un processo con privilegi diversi è banale dal punto di vista della teoria.

Il SO mette a disposizione programmi di sistema che permettono eseguire un comando cambiando l'utente o il gruppo di lavoro primario.

Posto che abbiate le credenziali per farlo!

# Elevazione dei privilegi di un comando

(Molto più interessante)

Lo scenario di gran lunga più interessante è il secondo: un utente lancia un comando, che assume automaticamente privilegi superiori.

Il tutto, in maniera controllata dal SO!

Come fare?

# L'idea di fondo

(Prevedere due personalità di esecuzione in modo utente)

L'idea di fondo è quella di prevedere, per un processo in esecuzione in modo utente, due distinte “personalità”.

Una personalità normale, corrispondente ad un processo che può fare cose normali (no superpoteri).

Una personalità “eroe”, corrispondente ad un processo che può fare tutte (sì superpoteri).

E se si usassero due coppie distinte di UID/GID per rappresentare tali personalità?

# Utente e gruppo effettivi

(La personalità "eroe")

Il SO fornisce due ulteriori astrazioni software dal nome **identificatore di utente effettivo (effective user id)** e **identificatore di gruppo effettivo (effective group id)**. Queste due astrazioni sono credenziali di processo, associate al processo in esecuzione.

- Non ai file.

- Non all'account.

I relativi campi delle strutture dati di controllo sono memorizzati nel descrittore di processo.

# Utenti e gruppo “reali”

(La personalità “normale”)

Vi ricordate le credenziali di processo introdotte nelle lezioni precedenti?

Identificatore utente.

Identificatore gruppo.

Nella realtà, queste due credenziali prendono il nome di:

**Identificatore reale di utente (real user id).**

**Identificatore reale di gruppo (real group id).**

Il motivo? Per distinguerle dagli identificatori di utente e gruppo effettivi.

# Orrore!

(Danny is scared by all these IDs...)



# Domande

(Indotte dalla confusione)

Quando e come un processo passa dalla personalità "normale" alla personalità "eroe"?

Il controllo degli accessi ai file non avviene esibendo gli identificatori reali? Che senso ha introdurre quelli effettivi?



# Una tragica confessione del docente

(Che sta cercando di indorare la pillola amarissima)

Ricordate le procedure di controllo degli accessi viste poco fa?

Permessi.

Liste di controllo degli accessi.

Le procedure sono state raccontate in maniera semplificata, per non turbarvi troppo.

Sono state omesse due informazioni importanti. È giunta l'ora di dirvele.

# “Com'è umano Lei...”

(Ugo è contentissimo di scoprire che quello che ha studiato non è tutto vero...)



# UID reale = 0 → Accesso garantito

(Subito, senza controllo)

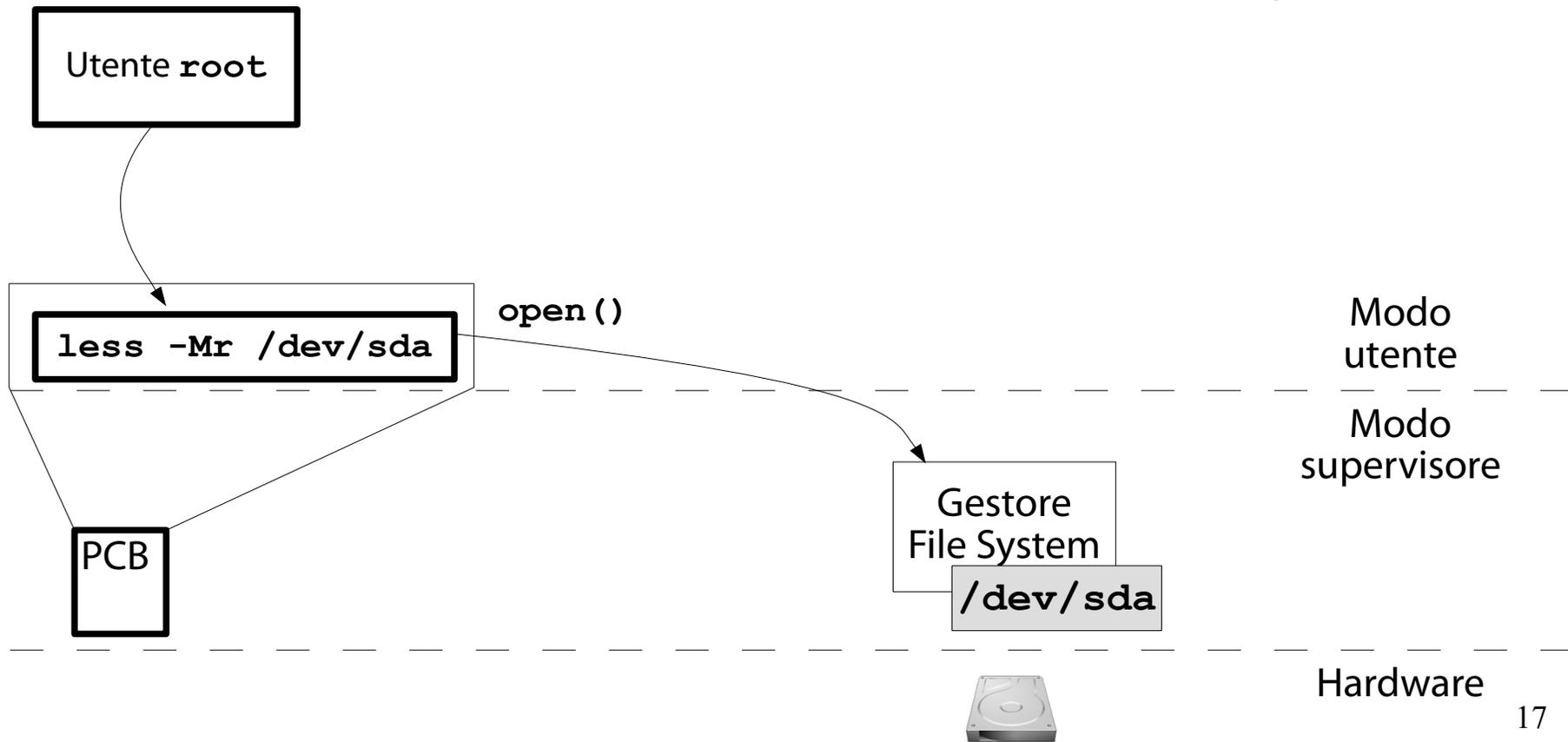
Se il processo si presenta con un identificatore di utente reale uguale a 0

ossia, se è stato lanciato dall'utente **root**  
allora il nucleo non effettua alcun controllo!

Niente controllo di permessi!

# L'utente root è accontentato sempre

(Controllo accessi /dev/sda: non c'è se è root a lanciare il processo)



# UID reale $\neq$ 0 $\rightarrow$ Controllo

(Le credenziali del processo usate nel controllo sono quelle effettive)

Se il processo si presenta con un identificatore di utente reale diverso da 0

ossia, se è stato lanciato da un utente diverso da **root**

allora il nucleo effettua il controllo di accesso.

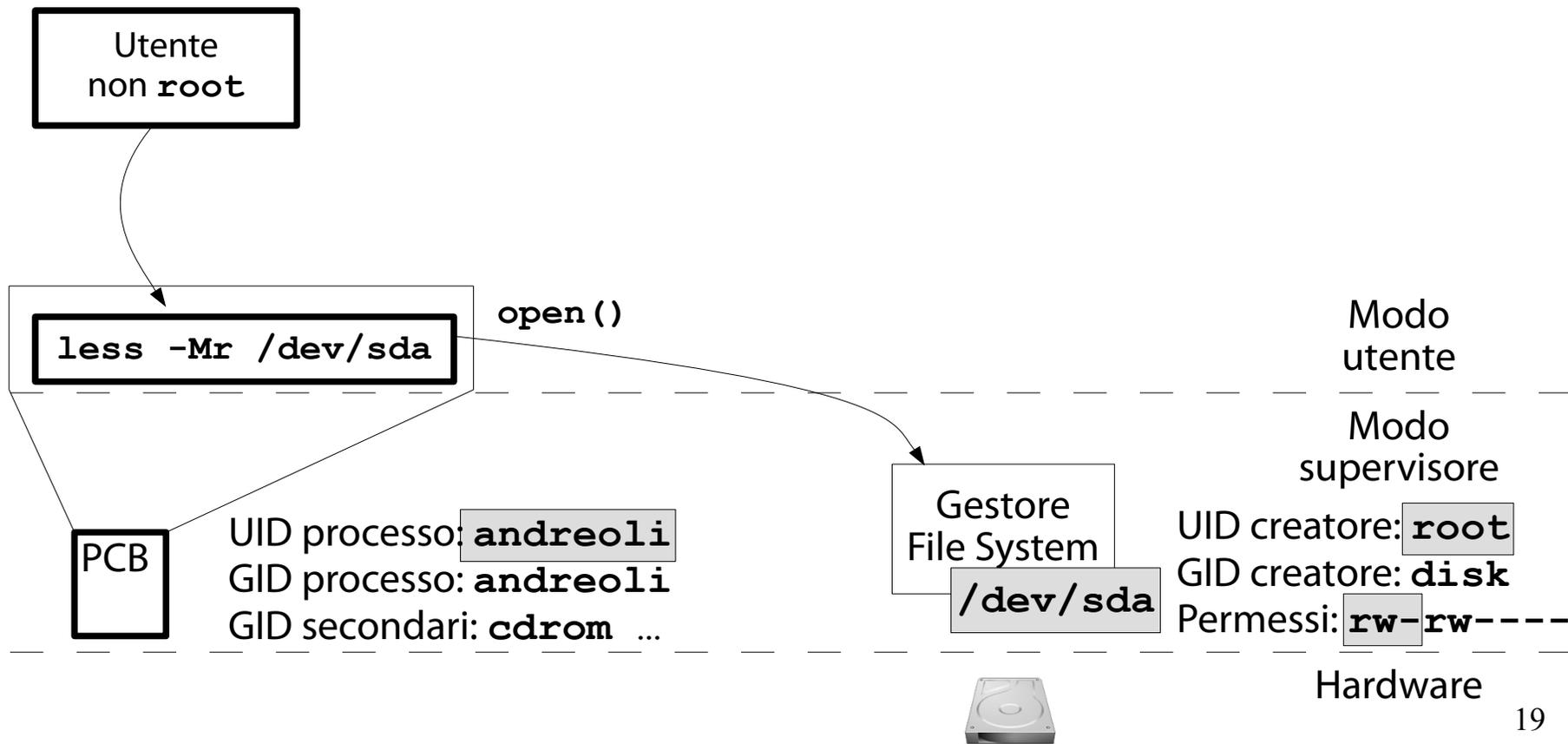
Sui permessi.

Le credenziali di processo usate per effettuare il controllo non sono quelle reali.

**SONO QUELLE EFFETTIVE!**

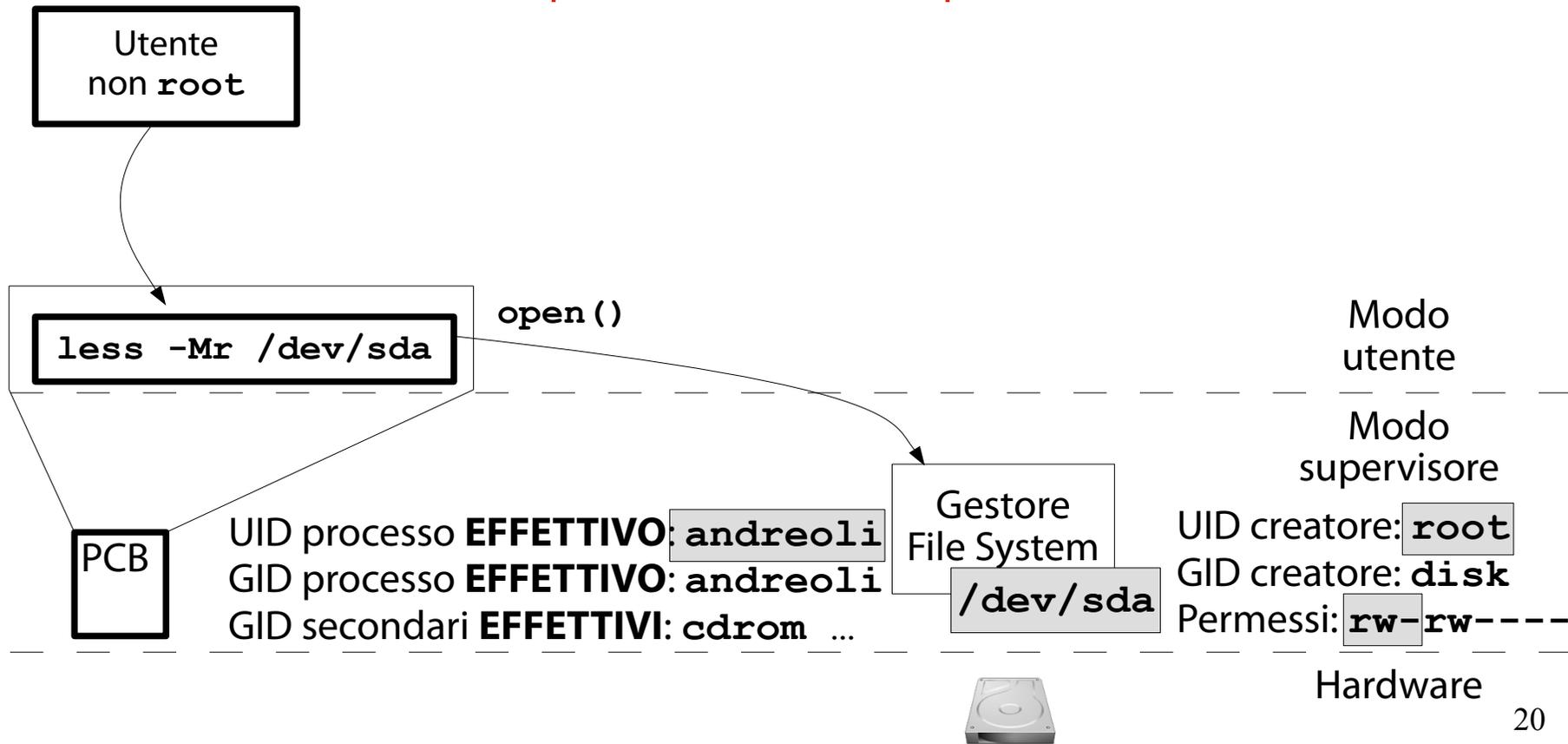
# L'utente normale subisce i controlli

(Controllo accessi /dev/sda)



# Le credenziali di processo sono diverse

(Sono quelle effettive, non quelle reali)



# La differenza tra "reale" ed "effettivo"

(In condizioni normali)

Quando un processo è creato, di solito:

l'identificatore utente effettivo è posto uguale all'identificatore utente reale.

l'identificatore di gruppo effettivo è posto uguale all'identificatore di gruppo reale.

Di solito...

# La differenza tra "reale" ed "effettivo"

(In condizioni non normali)

In una situazione ben specifica, invece:

l'identificatore utente effettivo è posto uguale all'identificatore utente del creatore del file.

l'identificatore di gruppo effettivo è posto uguale all'identificatore di gruppo del creatore del file.

# La conseguenza

(Guardate un po' i comandi nella directory `/usr/bin...`)

Elencando i vari attributi dei file eseguibili nella directory `/usr/bin` con `ls -l`:

```
ls -l /usr/bin
```

si scopre che utente creatore del file e il gruppo del file sono `root` e `root`.

→ In condizioni non normali, l'utente potrebbe acquisire i privilegi di `root` quando esegue questi comandi.

Interessante...

# Il bit SETUID dei permessi

(Il bit che abilita la condizione non normale)

Le azioni disponibili su un file non sono solo:

**r**: lettura

**w**: scrittura

**x**: esecuzione

Ve ne è un'altra, sostituta di **x**, applicabile alla prima terna nel caso di file.

**s**: esecuzione con i privilegi dell'utente creatore del file.

Il bit **s** prende il nome di **set user ID (SETUID)**.

# Il bit SETUID all'opera

(Analisi dei permessi del comando `/usr/bin/passwd`)

Si osservino gli attributi del file eseguibile

`/usr/bin/passwd:`

```
ls -l /usr/bin/passwd
```

Utente creatore: **root**.

Gruppo primario del file: **root**.

Permessi: **rwsr-xr-x**.

Si noti la **s** applicata  
alla prima terna di permessi



# Il bit SETUID all'opera

(Esecuzione del comando `passwd`)

Quando l'utente lancia il comando `passwd`, accade questo:

- viene creato un nuovo processo;

- utente e gruppo reali rimangono quelli dell'utente di partenza (ad esempio, **studente**);

- l'utente effettivo diventa quello dell'utente creatore del file (**root**);

- viene caricata in memoria l'immagine dell'eseguibile `/usr/bin/passwd`.

→ `passwd` esegue come se fosse lanciato dall'utente **root**.

# Pro e contro del bit SETUID

(Una comodità che può costare molto cara)

**Vantaggi:** il processo esegue con le credenziali di `root` e può eseguire operazioni delicate.

Senza che l'utente debba tramutarsi prima in un amministratore.

→ Comodità d'uso.

**Svantaggi:** il processo esegue con i massimi privilegi possibili.

Se il programma è scritto male, è possibile fargli compiere azioni spregevoli, non ultima l'esecuzione di codice arbitrario!

# Il bit SETGID dei permessi

(Analogo al bit SETUID, per i gruppi)

Le azioni disponibili su un file non sono solo:

**r**: lettura

**w**: scrittura

**x**: esecuzione

Ve ne è un'altra, sostituita di **x**, applicabile alla seconda terna nel caso di file.

**s**: esecuzione con i privilegi gruppo del file.

Il bit **s** prende il nome di **set group ID (SETGID)**.

Valgono tutte le considerazioni svolte per il bit SETUID. 28

# Il problema dell'elevazione dei privilegi

(O non si può fare nulla, o si può fare tutto; non c'è una via di mezzo)

Il vero problema dello schema di permessi visto finora è il seguente.

Un processo non ha alcun privilegio particolare oppure, quando diventa amministratore, acquisisce tutti i privilegi possibili per l'intera esecuzione!

Non esiste una via di mezzo.

# Una domanda (im)pertinente

(Di quelle che fanno guadagnare punti bonus)

È oramai chiaro che il modello di permessi visto finora è di tipo “o tutto o niente”.

Ciò non va bene per diversi motivi.

Il processo mantiene i pieni privilegi anche dopo aver svolto le operazioni critiche.

Il processo ottiene i pieni privilegi laddove ne servivano di meno.

In entrambi i casi si viola il principio del minimo privilegio!

Si può raffinare lo schema dei permessi in tal senso?

# La risposta

(Dovrebbe essere sufficientemente chiara)

**SÌ!\***

\*Si possono addirittura risolvere entrambi i problemi proposti!

# Identificatore SETUID/SETGID salvato

(Risolve il problema del mantenimento dei privilegi per l'intera esecuzione)

L'**identificatore SETUID salvato (saved SETUID identifier)** di un processo è una copia del valore dell'identificatore di utente effettivo all'inizio dell'esecuzione.

Tale campo è presente nel descrittore di processo.

L'**identificatore SETGID salvato (saved SETGID identifier)** di un processo è una copia del valore dell'identificatore di gruppo effettivo all'inizio dell'esecuzione.

Tale campo è presente nel descrittore di processo.

# Rilascio temporaneo dei privilegi

(Il processo riduce la sua “potenza” da utente effettivo a utente reale)

Per un processo SETUID/SETGID in esecuzione, il SO permette (tramite una funzione specifica) il **rilascio temporaneo dei privilegi effettivi (privilege drop)**.

Il processo invoca la chiamata di sistema.

Il nucleo scrive negli identificatori effettivi i valori degli identificatori reali.

Poiché gli utenti reali sono meno privilegiati degli utenti effettivi, si ottiene un rilascio di privilegio.

# Motivazione

(Del rilascio temporaneo)

Perché un processo dovrebbe effettuare il rilascio temporaneo dei privilegi?

La risposta è semplice e diretta.

Le porzioni critiche del programma sono eseguite con i privilegi più elevati possibile (gli effettivi).

Porzione critica → aggiornamento file password, accesso ad una periferica, ...

Le porzioni non critiche del programma sono eseguite con i privilegi originari (i reali).

Porzione non critica → Stampa menu help, ...

# Recupero dei pieni privilegi

(Il processo ripristina la sua “potenza” da utente reale a utente effettivo)

Per un processo SETUID/SETGID in esecuzione, il SO permette (tramite una chiamata di sistema opportuna) **il ripristino dei privilegi effettivi (privilege restore)**.

Il processo invoca la chiamata di sistema.

Il nucleo scrive negli identificatori effettivi i valori degli identificatori salvati.

Ecco a cosa servivano gli identificatori salvati!

# Capability

(Risolvono il problema dei troppi privilegi associati all'account `root`)

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.

# Quali capability esistono?

(**man 7 capabilities** ve lo dice)

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 se il processo carica in memoria una nuova immagine con la chiamata di sistema opportuna.

# Capability di file

(La naturale estensione del bit setuid)

Un file può contenere (oppure no) informazioni riguardanti le capability.

Se le contiene, queste informazioni sono usate dal nucleo 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 nucleo 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 (inizialmente contenente tutte le capability) da cui il processo può depennare quelle che non vuole far ereditare ai suoi figli

# Calcolo dell'insieme P'(permitted)

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

Capability permesse del processo dopo l'esecuzione dell'immagine

Capability ereditabili del processo prima dell'esecuzione dell'immagine

AND bit a bit

Capability ereditabili del file

Capability richieste dal processo e presenti nel file

$P'(\text{permitted}) = (P(\text{inheritable}) \& F(\text{inheritable}))$

OR bit a bit

$(F(\text{permitted}) \& M)$

Capability permesse del file

AND bit a bit

Maschera di capability (inizialmente completa)

Capability permesse del file e non filtrate dalla maschera

# Calcolo dell'insieme P'(effective)

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

Capability permesse del  
processo dopo l'esecuzione  
dell'immagine

Bit presente  
nel file

Vale 1?

Sì → le capability  
effettive del processo  
sono tutte quelle  
permesse

No → nessuna  
capability  
effettiva

$$P'(\text{effective}) = F(\text{effective}) ? P'(\text{permitted}) : 0$$

# 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



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

# Una domanda (im)pertinente

(Di quelle che fanno guadagnare punti bonus)

Sono stati discussi due sistemi di elevazione dei privilegi:

Bit SETUID/SETGID;

Capability.

Si supponga che siano entrambi attivi.

Sono usati tutti e due i sistemi?

Se sì, in che ordine?

Oppure ne viene usato solo uno?

Se sì, quale?

# La risposta

(Dovrebbe essere sufficientemente chiara)

Sono usati entrambi i sistemi di elevazione dei privilegi.

Prima i bit SETUID/SETGID (che già danno l'intero insieme dei privilegi).  
Poi le capability.

**ELEVAZIONE CON SU**

# Scenario e interrogativi

(Come sono gestiti i gruppi? È possibile visionare le proprietà di un gruppo?)

**Scenario:** in un SO moderno (time sharing, multi-utente), un utente vuole eseguire comandi con privilegi modificati.

## **Interrogativi:**

Esistono strumenti per eseguire comandi con privilegi modificati?

Esistono strumenti per prendere visione dei privilegi di un utente?

# Visione degli identificatori

(Si usa il comando `id`)

Il comando `id` stampa gli identificatori utente, di gruppo primario e dei gruppi secondari per uno specifico utente. L'uso è molto semplice.

```
id [OPZIONI] . . . [UTENTE]
```

Se usato senza argomenti, `id` considera l'utente attivo sul terminale. Altrimenti, si considera l'utente specificato come argomento.

# Esercizio 1 (1 min.)

Stampate tutti gli identificatori:  
dell'utente attivo sul vostro terminale;  
dell'utente **root**.

# Esecuzione di una shell

(Come utente **root**)

Il comando **su** esegue comandi generici con le credenziali di un altro utente.

Nel suo uso più semplice, **su** non richiede né opzioni, né argomenti:

**su**

In tale scenario, il SO assume che:

l'utente in cui ci si voglia trasformare sia **root**;

il comando che si voglia eseguire sia l'interprete di default (solitamente, **/bin/bash**).

# BIG FAT WARNING

(You have been warned)

Se non si specifica altro, il comando **su** non carica le variabili di ambiente del nuovo utente.

Le applicazioni che leggono queste variabili potrebbero confondersi.



## Esercizio 2 (1 min.)

Diventate l'utente **root**. Stampate le variabili di ambiente con il comando interno **export**.

Notate qualcosa di strano?

# Caricamento del nuovo ambiente

(Si usa l'opzione `-login` del comando `su`)

Per caricare l'ambiente è necessario usare l'opzione `--login` del comando `su`.

I seguenti comandi sono equivalenti:

```
su --login
```

```
su -l
```

```
su -
```

Ad esempio, per diventare `root` e caricare il suo ambiente, dovete digitare il comando seguente:

```
su -
```

## Esercizio 3 (2 min.)

Diventate **root**, caricando in memoria il suo ambiente. Stampate le variabili di ambiente con il comando **export**. Confrontate l'output attuale con quello dell'esercizio precedente.

Notate qualcosa di strano?

# Esecuzione di una shell

(Come un altro utente specifico)

È possibile diventare un altro utente specifico, passando il suo nome di login come argomento di **su**.

Ad esempio, per lanciare una shell come utente **prova** (e caricare il suo ambiente):

```
su - prova
```

## Esercizio 4 (1 min.)

Create un utente **docente**, se già non esiste.  
Lanciate una shell da utente **docente**.

# Esecuzione di un comando generico

(Si usa l'opzione **-c** di **su**)

È possibile lanciare non solo una shell, bensì un comando specifico. Il comando è specificato come argomento dell'opzione **-c** di **su**.

Ad esempio, per lanciare il comando `top` come utente `docente`, potete scrivere:

```
su -c top - docente
```

# Esecuzione di un comando generico

(Come un utente specifico)

Se il comando è complesso (ossia, contiene a sua volta opzioni e argomenti) va scritto fra virgolette.

Ad esempio, per eseguire il comando `ls -a -l`, bisogna eseguire il comando nel modo seguente:

```
su -c "ls -a -l" - docente
```

## Esercizio 5 (1 min.)

Lanciate da utente **prova** un comando che mostri solo file e directory nascosti nella sua home directory.

# ELEVAZIONE CON SUDO

# Scenario e interrogativi

(È possibile eseguire comandi come altri utenti dando la propria password?)

**Scenario:** in un SO moderno (time sharing, multi-utente), un utente può eseguire comandi come un altro utente. Tuttavia, è necessario conoscere le credenziali dell'altro utente.

## Interrogativi:

È possibile eseguire comandi come un altro utente senza dover conoscere le credenziali dell'altro utente?

Magari, presentando le proprie credenziali?

Oppure, non presentandole affatto?

# La risposta

(Dovrebbe essere sufficientemente chiara)

**SÌ!\***

\*Presentando le proprie credenziali oppure, addirittura, senza fornire credenziali!

# Il comando **sudo**

(Permette di eseguire comandi come un altro utente e/o gruppo)

Il comando esterno **sudo** esegue un comando come un altro utente e/o gruppo.

Proprio come il comando **su**.

A differenza di **su**, di default **sudo** chiede la password dell'utente di partenza, non di quello di arrivo.

# Il file `/etc/sudoers`

(Configurazione di `sudo`)

Prima di poter usare `sudo`, si rende necessaria la sua configurazione. La configurazione di `sudo` è contenuta nel file `/etc/sudoers`.

Come si edita la configurazione?

Si può aprire direttamente il file `/etc/sudoers` con un editor di testo.

È preferibile usare il comando `visudo` che apre un editor, blocca l'accesso al file ad altri utenti e controlla gli errori di configurazione.

# Apertura del file `/etc/sudoers`

(Il comando `visudo`)

Si apra il file `/etc/sudoers`, lanciando `visudo` da utente `root`:

```
su -  
visudo
```

Il comando `visudo` usa la variabile di ambiente `EDITOR` per capire quale editor eseguire.

Se si vuole usare un editor specifico (ad es., `vim`), si può eseguire il comando seguente:

```
EDITOR=vim visudo
```

# Il formato del file `/etc/sudoers`

(Una riga → Un commento o una direttiva di configurazione)

Ogni riga del file `/etc/sudoers` è:

un commento (primo carattere = #);

una direttiva di configurazione (primo carattere ≠ #).

Una direttiva di configurazione specifica:

il caricamento di altri file di configurazione.

la definizione di proprietà e di alias.

specifiche di permessi su utenti/gruppi di utenti.

# Una osservazione

(Non vi si può spiegare tutto in sole 72 ore)

Il formato completo del file `/etc/sudoers` è documentato in una pagina di manuale apposita.

`man 5 sudoers`

Sezione **SUDOERS FILE FORMAT**.

Sono presenti numerosi esempi di configurazione (impossibili da vedere tutti in un corso introduttivo).

`man 5 sudoers`

Sezione **EXAMPLES**.

# Il file `/etc/sudoers` in Debian

(Parte I: configurazioni di default)

```
#  
# This file MUST be edited with the 'visudo' command as root.  
#  
# Please consider adding local content in /etc/sudoers.d/  
instead of  
# directly modifying this file.  
#  
# See the man page for details on how to write a sudoers  
file.  
#  
Defaults                env_reset  
Defaults                mail_badpass  
Defaults  
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bi  
n:/sbin:/bin"
```

# Il file `/etc/sudoers` in Debian

(Parte 0: commenti)

```
#  
# This file MUST be edited with the 'visudo' command as root.  
#  
# Please consider adding local content in /etc/sudoers.d/  
instead of  
# directly modifying this file.  
#  
# See the man page for details on how to write a sudoers  
file.  
#
```

Questi sono commenti. Si possono bellamente trascurare.

# Il file `/etc/sudoers` in Debian

(Parte 1: configurazioni di default)

Per tutti gli utenti è abilitata la proprietà `env_reset`. I comandi lanciati con `sudo` eseguono in un ambiente ristretto comprendente poche e selezionate variabili di ambiente.

```
Defaults          env_reset
Defaults          mail_badpass
Defaults
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

# Il file `/etc/sudoers` in Debian

(Parte 1: configurazioni di default)

Per tutti gli utenti è abilitata la proprietà `mail_badpass`. Se un utente sbaglia la password, viene inviata una e-mail all'indirizzo di posta elettronica dell'amministratore dell'host.

```
"username is not in the sudoers file.  
This incident will be reported."
```

```
Defaults          env_reset  
Defaults          mail_badpass
```

```
Defaults  
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

# Il file `/etc/sudoers` in Debian

(Parte 1: configurazioni di default)

Per tutti gli utenti è abilitata la proprietà `secure_path`. I comandi lanciati con `sudo` ricevono una variabile di ambiente `PATH` con il valore specificato nell'argomento.

```
Defaults          env_reset
Defaults          mail_badpass
```

```
Defaults
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

# Il file `/etc/sudoers` in Debian

(Parte 2: definizione degli alias)

```
# Host alias specification
```

```
# User alias specification
```

```
# Cmnd alias specification
```

# Il file `/etc/sudoers` in Debian

(Parte 2: definizione degli alias)

```
# Host alias specification  
  
# User alias specification  
  
# Cmnd alias specification
```

Qui si possono inserire degli **alias**, ovvero stringhe di testo mirate a semplificare la costruzione dei comandi (un po' come gli alias di shell).

Attualmente non sono definiti alias.

# Il file `/etc/sudoers` in Debian

(Parte 2: definizione degli alias)

```
# Host alias specification  
  
# User alias specification  
  
# Cmnd alias specification
```

Esempio di alias: si vuole usare la stringa **WEBMASTERS** per identificare tutti gli utenti responsabili di un sito Web.

Si scrive:

```
User_Alias WEBMASTERS = will, wendy, wim
```

Si usa **WEBMASTERS** al posto di **will**, **wendy**, **wim**.

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification
```

```
root    ALL=(ALL:ALL) ALL
```

```
# Allow members of group sudo to execute any command
```

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

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification
root    ALL=(ALL:ALL) ALL

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

Queste sono specifiche di privilegi per utenti e gruppi.  
La specifica ha il seguente formato (semplificato):

`<who> <where> = <as whom> <what>`

Who:	chi ha il permesso di eseguire
Where:	da quali host ha il permesso di eseguire
As whom:	chi può diventare prima di eseguire
What:	cosa può eseguire

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification  
root    ALL=(ALL:ALL) ALL
```

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

Ci si concentra sulla prima specifica di permessi.

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification
```

```
root    ALL=(ALL:ALL) ALL
```

```
# Allow members of group sudo to execute any command
```

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

`<who>` `<where>` = `<as whom>` `<what>`

La specifica riguarda l'utente `root`.

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification
```

```
root ALL=(ALL:ALL) ALL
```

```
# Allow members of group sudo to execute any command
```

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

`<who>` `<where>` = `<as whom>` `<what>`

La specifica è valida su tutti i possibili host di Internet (**ALL**).

È possibile restringere l'esecuzione a specifici indirizzi IP. Nella pratica, qui si lascia sempre **ALL**.

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification
```

```
root    ALL=(ALL:ALL) ALL
```

```
# Allow members of group sudo to execute any command
```

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

`<who> <where> = <as whom> <what>`

L'utente `root` può trasformarsi in un qualunque altro utente (primo **ALL**).

Il gruppo primario dell'utente `root` può trasformarsi in un qualunque altro gruppo (secondo **ALL**).

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification
```

```
root    ALL=(ALL:ALL) ALL
```

```
# Allow members of group sudo to execute any command
```

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

`<who>` `<where>` = `<as whom>` `<what>`

L'utente `root` può eseguire tutti i comandi possibili.

# Il file `/etc/sudoers` in Debian

(Parte 3: specifica dei privilegi utente)

```
# User privilege specification  
root    ALL=(ALL:ALL) ALL
```

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

La seconda specifica è quasi identica alla prima.

L'unica differenza è la presenza del carattere `%` prima del primo campo.

Il carattere `%` permette a **sudo** di interpretare la stringa seguente come il nome di un gruppo.

→ I membri del gruppo **sudo** hanno le stesse facoltà di lancio privilegiato concesse all'utente **root**.

# Il file `/etc/sudoers` in Debian

(Parte 4: caricamento di ulteriori configurazioni)

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

```
#includedir /etc/sudoers.d
```

# Il file `/etc/sudoers` in Debian

(Parte 4: caricamento di ulteriori configurazioni)

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

#includedir /etc/sudoers.d
```

La direttiva **`includedir`** carica ed interpreta tutti i file di configurazione contenuti nella directory passata come argomento (`/etc/sudoers.d`).

L'ordine di caricamento è quello alfanumerico.

Attualmente non sono caricati ulteriori file (la direttiva è commentata).

# Lettura dei propri privilegi

(AKA "Che cosa mi fa fare **sudo**?")

L'opzione **-l** di **sudo** specifica cosa può fare l'utente attuale.

```
sudo -l
```

Se specificata due volte, l'opzione **-ll** produce un output più prolisso.

```
sudo -ll
```

L'opzione **-U** specifica il nome di login dell'utente di cui si vogliono ottenere i privilegi di lancio.

```
sudo -llU docente
```

# L'output del comando

(Deprimente per alcuni, euforico per altri)

Chi ha appena installato **sudo** si vedrà comparire con ogni probabilità un output simile.

```
L'utente nome_login non può eseguire sudo su host_name.
```

Chi ha già installato e configurato **sudo** si vedrà comparire con ogni probabilità un output simile.

```
L'utente nome_login può eseguire i seguenti comandi su nome_host:
```

```
Voce sudoers:
```

```
RunAsUsers: ALL
```

```
Comandi:
```

```
ALL
```

# Una amara constatazione

(**sudo** non fa eseguire nulla in maniera privilegiata)

Il comando **sudo** è configurato per non far eseguire nulla in maniera privilegiata a chi non soddisfa i requisiti del file **/etc/sudoers**.

Per poter usare **sudo** bisognerebbe:

essere **root**

OPPURE

far parte del gruppo **sudo**.

# Aggiunta al gruppo **sudo**

(Rende possibile l'esecuzione di qualunque comando con privilegi arbitrari)

Ci si aggiunga al gruppo **sudo**:

```
gpasswd -a studente sudo
```

Si riavvii il computer e ci si autentichi nuovamente.

# Esecuzione di un comando

(Con privilegi di `root`)

Nella sua forma più semplice, il comando `sudo` riceve un solo argomento: il comando da eseguire.

Il comando è eseguito usando l'utente `root` ed il gruppo `root`.

Provate ad eseguire:

```
sudo cat /dev/sda
```

# L'output del comando

(Il contenuto del disco `/dev/sda`)

L'output del comando è il contenuto del primo disco rigido SATA, rappresentato in ASCII.

Il comando è stato eseguito come utente **root**.

È stata chiesta una password, che non è quella dell'utente finale (**root**), bensì quella dell'utente di partenza (**studente**)!

# Come verificare che siamo **root**?

(Malfidati...)

Si possono verificare le credenziali di **root** con il comando **id**.

Per ottenere gli identificatori reali dell'utente, del gruppo e dei gruppi secondari, si eseguano i comandi seguenti:

```
sudo id -r -u
```

```
sudo id -r -g
```

```
sudo id -r -G
```

→ Si è realmente **root**; si appartiene all'unico gruppo **root**.

# L'ambiente di esecuzione di **sudo**

(Ridotto rispetto a quello standard)

Il comando **sudo**:

- genera un processo figlio;
- seleziona alcune specifiche variabili di ambiente;
- esegue l'immagine del comando con gli argomenti e le variabili di ambiente scelte.

# Esecuzione di un comando

(Con i privilegi di un utente specifico)

L'opzione **-u** di **sudo** permette di specificare il nome di login dell'utente che eseguirà il comando.

Ad esempio, per eseguire il comando **id** come utente

**prova:**

```
sudo -u prova id
```

## Esercizio 6 (2 min.)

Aprire un terminale e lanciate il comando **env**.

Aprire un altro terminale e lanciate il comando **env** come il vostro utente.

Notate delle differenze nell'output dei due comandi?

# Esecuzione di un comando

(Con i privilegi di un gruppo specifico)

L'opzione **-g** di **sudo** permette di specificare il nome del gruppo primario sotto cui si eseguirà il comando.

L'utente può far parte o no del gruppo.

Ad esempio, per eseguire il comando **id** come utente **docente** e gruppo **root**:

```
sudo -u docente -g root id
```

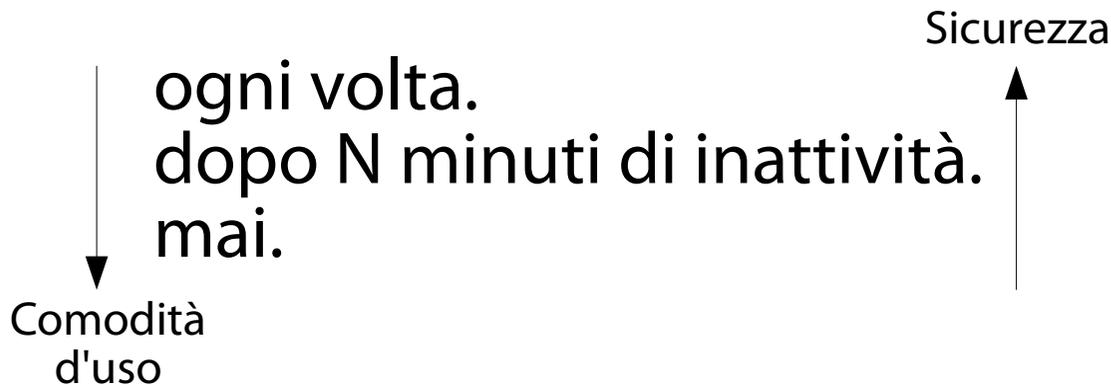
## Esercizio 7 (2 min.)

Eseguite come utente **prova** e gruppo primario **disk** il comando che legge il primo disco rigido SATA.

# Gestione delle credenziali

(Ogni quanto **sudo** chiede la password?)

Quando si lancia un comando con **sudo**, quest'ultimo può chiedere la password:



# Timeout di richiesta password

(Più piccolo è, meglio è)

La proprietà `timestamp_timeout` permette di impostare l'intervallo massimo di validità dei privilegi acquisiti. Scaduto l'intervallo, al primo comando immesso tramite `sudo` viene chiesta nuovamente la password.

Ad esempio, per impostare un intervallo di validità di 5 minuti, si può scrivere in `/etc/sudoers`:

```
Defaults                timestamp_timeout=5
```

# In un mondo ideale...

(...`timestamp_timeout` dovrebbe essere impostato a 0!)

Il valore ottimale di `timestamp_timeout` dal punto di vista della sicurezza è 0.

Se `timestamp_timeout`  $\neq$  0, è possibile per un estraneo dare comandi con `sudo` senza la password.

È sufficiente lasciare incustodito il vostro portatile, con un terminale aperto su cui si è già dato il comando `sudo` (ad esempio, per un aggiornamento software).

# Nel mondo reale...

(...`timestamp_timeout` ha un valore  $\gg 0$ !)

Nelle distribuzioni GNU/Linux, il valore di default per `timestamp_timeout` è  $\gg 0$ .  
Quanto è grande?

# 15 minuti!

# Esecuzione globale senza password

(NON FATELO SULLE VOSTRE MACCHINE)

È possibile disabilitare in maniera permanente la richiesta di una password.

Disabilitate la proprietà **authenticate** nel file **/etc/sudoers**.

**Defaults**                   **!authenticate**

**Occhio!** Il comando **sudo** non chiede più la password a nessuno!

# Esecuzione globale per utente/gruppo

(Senza password; non fatelo sulle vostre macchine, per favore)

Il comando **sudo** permette di etichettare (**tag**) i singoli comandi.

**man 5 sudoers**

Per ottenere un elenco dei tag, cercate **Tag\_Spec**.

L'etichetta **NOPASSWD** : disabilita la richiesta di password per un dato comando.

Se il comando è **ALL** → si disabilitano tutti i comandi!

# Esecuzione globale per utente/gruppo

(Senza password; non fatelo sulle vostre macchine, per favore)

Si modifichi come segue la riga relativa al gruppo **sudo** nel file `/etc/sudoers`.

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

Etichetta    Comando

# Esecuzione comandi specifici

(Senza password; se proprio dovete farlo, fatelo così)

Nell'ultima posizione di una specifica di privilegi potete scrivere, al posto della stringa **ALL**, un elenco di comandi specifici che volete siano eseguiti con le caratteristiche volute.

Scriverete in **/etc/sudoers**:

```
%sudo                ALL=(ALL:ALL)  
/path/to/cmd1 , /path/to/cmd2 , . . .
```

**ATTENZIONE!** Va scritto il percorso assoluto del comando.

# Esecuzione comandi specifici

(Senza password; se proprio dovete farlo, fatelo così)

Esempio: il comando **slabtop** mostra le allocazioni di memoria del kernel ai vari sottosistemi.

Per motivi di sicurezza, l'esecuzione di **slabtop** è permessa al solo utente **root**. Se volete eseguire **slabtop** senza password, scriverete in **/etc/sudoers**:

```
%sudo    ALL=(ALL:ALL) NOPASSWD: /usr/bin/slabtop
```

## Esercizio 8 (2 min.)

Modificate il file `/etc/sudoers` in modo tale da permettere al vostro utente l'esecuzione come `root` e senza password dei seguenti comandi:

```
slabtop
```

```
cat /dev/mem
```

# Ricapitolando

(Repetita iuvant)

Si abusi di **sudo** il meno possibile. Se proprio lo si deve usare, si cerchi almeno di seguire quest'ordine (i primi suggerimenti sono preferibili):

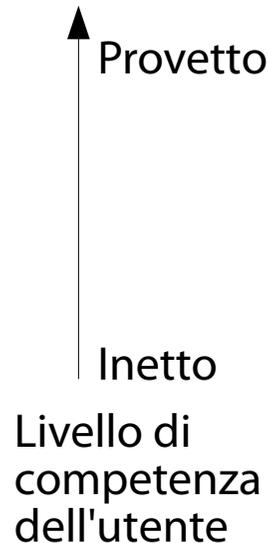
abilite singoli comandi con password.

abilite tutti i comandi con password.

abilite singoli comandi senza password.

abilite tutti comandi senza password.

disabilitate la richiesta di password per tutti.



# ELEVAZIONE CON SETUID/SETGID

# Scenario e interrogativi

(Come si impostano i bit SETUID/SETGID di un file?)

**Scenario:** L'utente vuole eseguire comandi di amministrazione che richiedono privilegi elevati senza dover ogni volta elevare i privilegi. A tal fine, vuole imparare ad impostare i bit SETUID/SETGID di un file.

## **Interrogativi:**

Come si impostano i bit SETUID/SETGID di un file?

# Bit SETUID/SETGID

(Esecuzione automatica con privilegi elevati)

GNU/Linux offre un ulteriore supporto per l'esecuzione con privilegi elevati: i bit di permessi SETUID e SETGID.

Una volta impostati tali permessi con il comando **chmod**, il processo esegue con i seguenti privilegi:

identificatore utente    eff. → identificatore creatore file

identificatore gruppo    eff. → identificatore gruppo file

Non sono necessari altri comandi.

# Impostazione del bit SETUID/SETGID

(Si usa il comando `chmod`)

Il comando `chmod` imposta anche i permessi SETUID e SETGID.

`chmod u+s`: impostazione bit SETUID.

`chmod g+s`: impostazione bit SETGID.

`chmod u-s`: rimozione bit SETUID.

`chmod g-s`: rimozione bit SETGID.

## Esercizio 9 (5 min.)

Il binario eseguibile **top** permette di monitorare i processi in maniera interattiva.

Copiate il file dell'eseguibile **top** nella vostra home directory, dandogli il nome **newtop**.

Cambiate utente creatore del file nel modo seguente:

utente creatore → **root**.

Impostate il bit SETUID e eseguite **newtop**.

Con quali diritti esegue **newtop**?

# **ELEVAZIONE CON CAPABILITY**

# Scenario e interrogativi

(Come si impostano le capability di un file?)

**Scenario:** l'utente vuole eseguire comandi di amministrazione che richiedono privilegi elevati senza dover impostare l'intero privilegio di root. A tal fine, vorrebbe impostare specifiche capability.

## **Interrogativi:**

Come si impostano le capability di un file?

# Capability

(I privilegi di `root` sono spezzati in tanti piccoli privilegi)

GNU/Linux offre un ulteriore supporto per l'esecuzione con privilegi elevati: le capability.

I due comandi usati per leggere ed impostare le capability sono `getcap` e `setcap`.

# Lettura di capability

(Si usa il comando `getcap`)

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

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

Ad esempio:

```
getcap /bin/ls
```

## Esercizio 10 (1 min.)

Per quale motivo non siete riusciti ad eseguire il comando **getcap**?

Che cosa dovete fare per eseguire il comando **getcap**?

# Come trovare comandi con capability

(Si usa la wildcard \* come argomento del comando `getcap`)

Per scoprire i comandi con capability potete eseguire `getcap` con argomenti "wildcard" che rappresentano gli eseguibili in questione:

```
getcap /bin/* /sbin/* /usr/bin/* /usr/sbin/*
```

Scoprirete l'esistenza di alcuni comandi che, per essere eseguiti da utente normale, devono acquisire particolari privilegi.

# Esempio: comando ping

(telnet towel.blinkenlights.nl)

Il comando **ping** verifica se un host remoto è raggiungibile.

```
ping towel.blinkenlights.nl
```

Per funzionare, **ping** ha bisogno di costruire “a mano” pacchetti di controllo ed inviarli.

Ciò non è concesso a tutti.

Ciò è concesso a **root**.

Ciò è concesso a chi esegue un comando con la capability **cap\_net\_raw** attiva sull'eseguibile.

# Un'occhiata alle capability di ping

(What's that "+ep" about?)

L'output del comando:

```
/sbin/getcap /bin/ping
```

è la stringa seguente:

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

Il comando  
in questione

La capability  
assegnata

La capability aggiunta (+)  
è effettiva (**e**)  
e permessa (**p**)

# 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 = clausola1,clausola2,...,clausolaN
```

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 = <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 si può usare 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)

# 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)

# Impostazione di capability

(Si usa il comando `setcap`)

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

```
setcap [OPZIONI] CAPABILITY FILE
```

Ad esempio:

```
setcap cap_net_raw+ep /path/to/ping
```

## Esercizio 11 (5 min.)

Usando il comando `cp`, copiate il comando `ping` nella vostra home directory, dandogli il nome `newping`.

Annulate le capability del file `newping`.

Eseguite `newping 8.8.8.8`. Funziona?

Assegnate le capability `cap_net_raw` agli insiemi `permitted` ed `effective` sul file `newping`.

Eseguite `newping 8.8.8.8`. Funziona?