ChatGPT

# Test di Sicurezza Linux Interattivo

## Introduzione

Benvenuto in questo **laboratorio interattivo** di Linux focalizzato sulla **cybersecurity in ambito aziendale**. Seguirai una serie di scenari pratici simulati nel terminale Linux, con difficoltà tra il principiante e l'intermedio. Ogni scenario rappresenta una situazione reale (dal rilevamento di attività sospette al "hardening" di un server) e ti guiderà **passo-passo** attraverso comandi chiave di Linux relativi alla sicurezza. I comandi trattati includono:

- **Gestione di file e permessi**: `ls`, `chmod`, `chown`, `find`, etc.
- **Monitoraggio e logging**: `journalctl`, `top`, `ps`, `netstat` / `ss`, etc.
- **Gestione utenti**: `useradd`, `passwd`, `sudo`, `groups`, etc.
- **Reti e firewall**: `iptables`, `ufw`, `nmap`, `netcat`, etc.
- **Analisi e sicurezza**: `chkrootkit`, `lynis`, `fail2ban`, `auditd`, etc.

Ogni esercizio fornirà **feedback dettagliato** e spiegazioni, in modo da imparare non solo "cosa" fare ma anche "perché". Inoltre, troverai alcuni **mnemonici e trucchi logici** per ricordare meglio i comandi. Ad esempio, molti comandi di modifica iniziano con `ch` (**ch**ange) come `chmod` (cambia permessi) e `chown` (cambia proprietario). Allo stesso modo, `fail2ban` significa letteralmente "fallire per bannare" (viene bannato un IP dopo troppi **fail**), aiutandoti a ricordare la sua funzione.

> **Nota:** Ogni blocco di codice simula comandi e output in un terminale Linux. Puoi immaginare di essere collegato a un server aziendale come utente amministratore e di seguire le istruzioni. Dopo i comandi, leggi attentamente la **spiegazione** per comprendere il risultato e il legame con la sicurezza.

Iniziamo il nostro percorso!

## Scenario 1: Gestione di File e Permessi

**Situazione:** Sei l'amministratore di un server web aziendale. Una recente verifica di sicurezza ha segnalato che alcuni file potrebbero avere permessi troppo aperti (ad esempio **777**, cioè leggibili, scrivibili ed eseguibili da chiunque). I file con permessi e proprietari errati possono esporre dati sensibili o permettere modifiche non autorizzate. In questo esercizio dovrai **individuare** e **correggere** tali file. Useremo comandi come `ls` (lista file), `find` (ricerca), `chmod` (cambia permessi) e `chown` (cambia proprietario).

1. **Elenco file e permessi:** Iniziamo elencando il contenuto di una directory sospetta (es. la cartella web `/var/www/html`). Il flag `-l` di `ls` mostra dettagli come permessi (colonna di sinistra), proprietario e gruppo:

```
user@server:~$ ls -l /var/www/html
total 16
-rw-rw-r-- 1 alice    alice    4096 Jul  8 14:20 index.html
-rwxrwxrwx 1 www-data www-data 1024 Jul  8 14:00 config.php
```

```
-rw-r--r-- 1 root     root      512 Jul  8 13:50 db.conf
drwxr-xr-x 2 www-data www-data 4096 Jul  8 13:45 uploads/
```

**Spiegazione:** L'output sopra elenca tre file e una directory `uploads/`. Notiamo subito qualcosa di preoccupante: il file `config.php` ha permessi `rwxrwxrwx` (indicati anche come **777** in notazione ottale). Ciò significa che *chiunque* può leggere, modificare o eseguire questo file – un grosso rischio se contiene configurazioni sensibili! Al contrario, `index.html` ha permessi `rw-rw-r--` (664) e `db.conf` ha `rw-r--r--` (644), che sono più restrittivi. Prima di agire, confermiamo se esistono *altri* file con permessi globali troppo aperti. Usiamo `find` per cercare file con permesso 777 all'interno di `/var/www`:

```
user@server:~$ sudo find /var/www -type f -perm 0777
/var/www/html/config.php
/var/www/html/uploads/debug.log
```

**Spiegazione:** Il comando `find` ha trovato due file con permessi **0777**: oltre a `config.php`, c'è un `debug.log` nella sottocartella `uploads`. È buona pratica utilizzare `find` per audit di sicurezza: ad esempio, un amministratore può cercare file con permessi inappropriati e confrontarli con le policy aziendali [1]. In ambienti enterprise, `find` **diventa uno strumento prezioso di sicurezza** perché aiuta a individuare e correggere configurazioni errate prima che possano essere sfruttate [1]. Ora procediamo a **ristringere i permessi** di questi file.

1. **Correzione dei permessi con `chmod`:** Utilizziamo `chmod` per rimuovere i permessi eccessivi. Ad esempio, impostiamo `config.php` a 640 (lettura/scrittura per proprietario, lettura per gruppo, niente per altri) e `debug.log` a 600 (accessibile solo al proprietario):

```
user@server:~$ sudo chmod 640 /var/www/html/config.php
user@server:~$ sudo chmod 600 /var/www/html/uploads/debug.log
user@server:~$ ls -l /var/www/html | grep config.php
-rw-r----- 1 www-data www-data 1024 Jul  8 14:00 config.php
```

**Spiegazione:** Ora `config.php` ha permessi `rw-r-----` (640), cioè leggibile solo dal proprietario e dal gruppo, e non accessibile dagli altri utenti. `debug.log` con 600 è accessibile solo al proprietario. Questo impedisce a utenti non autorizzati di curiosare o manomettere questi file. Ricorda il **mnemonico**: `chmod` = *change mode*, ovvero cambia i permessi (o "mode") di un file. Puoi specificare i permessi in notazione ottale (come 640) o simbolica (es. `u=rw,g=r,o=-`). Una regola pratica per file sensibili è evitare `7` nei permessi per "others" (altri utenti). In generale, **777 va evitato**: è preferibile concedere il minimo accesso necessario (principio del *least privilege*). Il comando `find` + `-perm` visto sopra può essere usato periodicamente proprio per assicurarsi che nessun file abbia permessi 777 non conformi alle policy [1].

1. **Correzione del proprietario con `chown`:** Diamo un'occhiata anche ai proprietari dei file. Dal listing iniziale, `config.php` e `uploads/` erano di proprietà dell'utente e gruppo `www-data` (tipico utente del web server), mentre `db.conf` era di `root`. Supponiamo che `db.conf` debba invece appartenere all'utente `alice` (magari uno sviluppatore web) per permetterle di modificarlo. Utilizziamo `chown` per cambiare proprietario e gruppo:

```
user@server:~$ sudo chown alice:alice /var/www/html/db.conf
user@server:~$ ls -l /var/www/html/db.conf
-rw-r--r-- 1 alice alice 512 Jul  8 13:50 db.conf
```

**Spiegazione:** Ora il file di configurazione del database appartiene ad Alice (utente e gruppo). `chown` = ch**ange** own**er** – un altro comando che inizia con `ch` a indicare un cambiamento. Assicurare i proprietari corretti è importante: se un file critico è di `root` ma dovrebbe essere gestito da un servizio o utente applicativo, potrebbe causare sia problemi di permessi sia rischi di sicurezza (ad esempio, il servizio potrebbe girare come utente privilegiato per scrivere su quel file, esponendo l'intero sistema in caso di compromissione). D'ora in avanti, ricorda questo semplice schema logico per file e permessi: **Lista → Individua → Correggi**. Prima **lista** i file (`ls`), **individua** problemi (`find` o guardando i permessi), quindi **correggi** con `chmod`/`chown`.

> **Riepilogo & Suggerimenti:**
> - Utilizza `ls -l` per **visualizzare permessi e proprietari** rapidamente.
> - Usa `find -perm` per **audit di sicurezza** dei file (es. trovare tutti i 777, SUID/SGID, ecc.) [1].
> - **Mnemonico:** i comandi di modifica permessi/proprietari iniziano per **"ch"** (cambia): `chmod` cambia i **modi** (permessi), `chown` cambia l'**owner** (proprietario).
> - Imposta i file secondo il **principio del minimo privilegio**: dai accesso solo a chi ne ha bisogno. Ad esempio, file con dati sensibili dovrebbero spesso essere 600 o 640, non 777.

Ora che i file sono in ordine, passiamo a monitorare il sistema per attività sospette.

## Scenario 2: Monitoraggio e Logging

**Situazione:** Un collega ti segnala che il server sembra rallentato e teme ci sia un processo sospetto (es. un malware miner o un intruso). Inoltre, ha notato tentativi di accesso SSH falliti. In questo scenario impareremo a **monitorare il sistema** in tempo reale e ad **analizzare i log** per individuare attività anomale. Useremo comandi come `journalctl` (log di sistema), `top` (processi in tempo reale), `ps` (stato dei processi), e `netstat`/`ss` (connessioni di rete).

1. **Verifica dei log di sistema con `journalctl`:** Su sistemi moderni (con systemd), i log di sistema sono centralizzati nel *journal* e consultabili con `journalctl` [2]. Possiamo filtrare i messaggi importanti, ad esempio errori di autenticazione. Proviamo a cercare nel journal gli errori (**priority err**) dall'ultimo riavvio (`-b`) relativi a SSH:

```
user@server:~$ sudo journalctl -p err -b | grep sshd
Jul 08 15:20:01 server sshd[2021]: Failed password for invalid user admin
from 10.0.0.5 port 51812 ssh2
Jul 08 15:20:05 server sshd[2021]: Failed password for invalid user admin
from 10.0.0.5 port 51812 ssh2
Jul 08 15:21:44 server sshd[2050]: Accepted password for alice from 192.
168.0.42 port 60544 ssh2
```

**Spiegazione:** Abbiamo estratto dal journal alcuni eventi di autenticazione: due tentativi *falliti* per un utente inesistente "admin" da un IP esterno (10.0.0.5) e un accesso riuscito per l'utente *alice* da un IP interno (192.168.0.42). I messaggi "Failed password for invalid user" indicano probabilmente un

**tentativo di brute-force** o scansione automatizzata su SSH. In un contesto aziendale, monitorare questi log è cruciale: `journalctl` rende facile filtrare per servizio (`-u sshd`), per priorità (`-p err` per errori) o per intervallo di tempo (`--since/--until`). La centralizzazione dei log con systemd journal consente ricerche potenti come queste [2]. Ad esempio, `journalctl -f` ti permette di **seguire in tempo reale** gli eventi man mano che accadono, molto utile durante un troubleshooting live [2].

> *Mnemonico: Journalctl = diario di bordo del sistema*. Ricorda che i log di sistema sono il primo posto dove cercare segni di attività sospette: **Guarda i LOG prima di tutto**.

1. **Identificazione di processi anomali con `top` e `ps`:** Passiamo ora a esaminare i processi attivi. Eseguiamo `top` per vedere i processi in tempo reale, ordinati per utilizzo CPU (premendo *q* uscirai da top):

```
user@server:~$ top -b -n1 | head -5   # Modalità batch per simulare output
top
PID   USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
1234  www-data  20   0   80564   3460   2900 R  98.7  0.1   3:21.45 kworker/
u8+
987   root      20   0  162648  12456   8320 S   5.3  0.3   0:01.05 apache2
456   mysql     20   0 1677800 107000   9120 S   3.0  2.6   0:30.22 mysqld
... (altri processi) ...
```

**Spiegazione:** Nell'output di `top` sopra (modalità batch), notiamo un processo PID **1234** eseguito da `www-data` che utilizza quasi il 99% di CPU. Il nome del comando è troncato come `kworker/u8+` – questo potrebbe essere un tentativo di mascherarsi come un processo di sistema (i veri `kworker` sono thread del kernel). Il fatto che giri sotto l'utente del web server e consumi tanta CPU è **altamente sospetto**. Quando vedi un processo anomalo in `top`, è utile ottenere maggiori dettagli con `ps`. Poiché i processi malevoli possono scomparire velocemente, *non affidarti solo a top*: conviene usare `ps` per catturare uno snapshot stabile dei processi [3]. Vediamo i dettagli del PID 1234:

```
user@server:~$ ps -p 1234 -o pid,user,cmd,%cpu,%mem,etime
  PID USER     CMD                     %CPU %MEM ELAPSED
 1234 www-data /tmp/kworker -d 4444  98.7  0.1  00:03:22
```

**Spiegazione:** Ora è chiaro: il processo `1234` è un binario eseguito da */tmp/kworker* con opzione `-d 4444`. Questa potrebbe essere una **backdoor**: ad esempio un programma maligno che apre una porta di debug/ascolto 4444. Notiamo alcuni *indicatori di compromissione* classici: il processo gira da `/tmp` (directory temporanea, insolito per processi legittimi) e il nome `kworker` cerca di confondersi con processi di sistema [4]. Questi dettagli rispecchiano consigli pratici di analisti: **cercare processi con nomi strani o camuffati, script shell lanciati da utenti inaspettati (es. una shell Bash sotto www-data), comandi come curl/wget o Python eseguiti dentro processi sospetti, o qualsiasi cosa che giri da /tmp** [4]. Tutti questi sono segnali di un possibile malware attivo.

> **Suggerimento:** in situazioni reali, potresti anche usare `ps aux --sort=-%cpu | head` per elencare i processi più pesanti, oppure cercare con `ps` processi che contengono parole chiave (ad esempio `ps -ef | grep tmp`). Come notato da esperti, `top` *può mancare processi che appaiono e scompaiono*

*velocemente*, quindi è sempre bene usare `ps` per un controllo approfondito [3]. Inoltre, confronta i processi con quelli attesi: se un server web dovrebbe avere solo `apache2` e `mysqld` oltre ai demoni di sistema, un eseguibile sconosciuto è un enorme campanello d'allarme.

1. **Analisi delle connessioni di rete con `netstat` / `ss`:** Abbiamo individuato un potenziale malware che potrebbe aver aperto una porta di rete (la 4444, come suggerisce l'opzione `-d 4444`). Dobbiamo verificare le connessioni e porte aperte. Possiamo usare `netstat -tulnp` (liste TCP/UDP Listening con PID) oppure il più moderno `ss`. `ss` è generalmente preferito perché più veloce e dettagliato rispetto a netstat [5], ma vediamo entrambe le alternative:

```
user@server:~$ sudo netstat -tulnp | grep 4444
tcp        0      0 0.0.0.0:4444      0.0.0.0:*     LISTEN      1234/./
kworker
```

```
user@server:~$ sudo ss -tulnp | grep 4444
LISTEN 0      50         *:4444            *:*    users:
(("kworker",pid=1234,fd=3))
```

**Spiegazione:** Entrambi i comandi confermano che **il processo 1234 sta ascoltando sulla porta TCP 4444** su tutte le interfacce. `netstat` mostra il PID/Programma (`1234/./kworker`), mentre `ss` fornisce info simili in formato leggermente diverso. In scenari di **incident response**, `ss` è uno strumento prezioso: esso può elencare socket e connessioni aperte, filtrare per stato o porta, e perfino mostrare i **processi associati** con l'opzione `-p` [6] [7]. Ad esempio, `ss -tuln` mostra tutte le porte in ascolto (utile per scovare servizi non autorizzati) [7], mentre `ss -tp` includerebbe i nomi dei processi [6].

Dal nostro output vediamo che `kworker` (il malware) è in ascolto. Probabilmente sta aspettando connessioni di comando e controllo (C2) da un aggressore [8]. Questo combacia con la realtà: molti malware e script non autorizzati "fanno **call home**" (si connettono a un server esterno) o **aprono porte** per ricevere comandi [9]. Se non controlli regolarmente le connessioni, potresti non accorgertene finché è troppo tardi [9].

A questo punto, un amministratore intraprenderebbe azioni immediate: terminare il processo maligno (`kill 1234`), investigare come ci è finito (controllando file dropper in /tmp), e mettere in sicurezza il sistema. Prima però, concludiamo la nostra analisi guardando se il malware ha stabilito connessioni esterne attive. Usiamo `ss` per cercare connessioni *established* da quel processo:

```
user@server:~$ sudo ss -tp state established '( sport = :4444 )'
ESTAB      0      0        10.0.0.10:4444      45.67.89.10:52134   users:
(("kworker",pid=1234,fd=4))
```

**Spiegazione:** Questo output (filtrato per porta sorgente 4444 in stato established) indica che il nostro malware su *10.0.0.10:4444* è connesso all'IP esterno *45.67.89.10:52134*. Abbiamo individuato l'IP di un possibile *attaccante remoto*. Informazioni come questa sono vitali per un'indagine. Vale la pena sottolineare: `ss` **supera** `netstat` in velocità e informazioni dettagliate ed è considerato essenziale nelle analisi moderne [5]. In effetti, `netstat` è deprecato su molte distro e bisogna installare

manualmente il pacchetto `net-tools` per usarlo (su Debian/Ubuntu ad esempio) [10]. Sulla nostra macchina era disponibile, ma tieni a mente di usare preferibilmente `ss` su sistemi attuali.

> **Riepilogo & Suggerimenti:**
> - **L-P-N (Log-Processi-Rete):** in caso di sospetti, controlla **Log** di sistema (`journalctl`), **Processi** attivi (`top` / `ps`) e **Network** (`netstat` / `ss`). Questa triade ti aiuta a coprire i principali indizi.
> - `journalctl` centralizza i log di servizi, kernel e sistema, con filtri potenti – sfruttalo per estrarre errori o eventi critici [2].
> - Usa `top` per un colpo d'occhio, ma approfondisci con `ps` per catturare processi lampo e dettagli come percorso di esecuzione e parametri [3].
> - `netstat` e `ss` mostrano porte e connessioni. **Tip:** `ss -tulnp` (TCP/UDP Listen) elenca tutte le porte in ascolto con i relativi processi – un modo rapido per trovare eventuali "backdoor" in ascolto [7].
> - Un processo in esecuzione da `/tmp` o con nome camuffato (es. un falso `sshd` non avviato da root) è segnale di compromissione [4].
> - **Ricorda:** Molti malware puntano a *nascondersi in bella vista* (es. nomi simili a processi di sistema) e a *comunicare verso l'esterno* [8]. La sorveglianza costante di processi e connessioni è una difesa fondamentale.

Con il malware identificato e (si suppone) terminato, passiamo allo scenario successivo: gestione sicura degli utenti sul sistema.

## Scenario 3: Gestione Utenti Sicura

**Situazione:** Devi creare un nuovo account per un amministratore di sistema appena assunto, assicurandoti che abbia i giusti privilegi senza compromettere la sicurezza. In questo scenario impariamo a **creare e gestire utenti** in modo sicuro, utilizzando `useradd` (creazione utente), `passwd` (impostazione password), i gruppi e il comando `sudo` per i privilegi amministrativi. Inoltre, verificheremo che l'account sia configurato correttamente.

1. **Creazione di un nuovo utente con `useradd`:** Creiamo l'utente **alice**. In molti sistemi (come Debian/Ubuntu) esiste anche uno script `adduser` interattivo, ma qui useremo `useradd` con opzioni per creare la home directory (`-m`) e impostare la shell (`-s`). Assicuriamoci di eseguirlo con privilegi elevati (tramite `sudo`):

```
user@server:~$ sudo useradd -m -s /bin/bash alice
```

**Spiegazione:** Il comando sopra crea l'utente *alice*, genera la directory home `/home/alice` con i file iniziali di default (grazie a `-m`), e assegna `/bin/bash` come shell di login. Non abbiamo specificato una password in questo passo (su alcune distro, l'account potrebbe essere creato inizialmente bloccato). È importante *creare sempre la home directory* per un nuovo utente umano, in modo che abbia un ambiente personale con i giusti permessi. Adesso impostiamo una password sicura per Alice.

1. **Impostazione della password con `passwd`:** Usiamo il comando `passwd` per assegnare una password all'utente *alice*:

```
user@server:~$ sudo passwd alice
New password: ********
```

```
Retype new password: ********
passwd: password updated successfully
```

**Spiegazione:** Abbiamo impostato la password (i caratteri inseriti non vengono mostrati per sicurezza). Scegli password robuste e, in ambiente aziendale, considera di applicare policy di scadenza password, complessità e storicità. Ad esempio, puoi configurare età minima/massima della password modificando `/etc/login.defs` (Lynis spesso suggerisce di impostarle) [11]. In questo modo si obbligano gli utenti a cambiarla periodicamente. Ora l'account *alice* è creato e attivo. Il passo successivo è conferirle i permessi amministrativi in modo sicuro.

1. **Aggiungere l'utente al gruppo sudo (privilegi amministrativi):** Invece di consentire a *alice* di loggarsi direttamente come root (pratica sconsigliata), seguiremo la strada standard: aggiungerla al gruppo **sudo** (su distribuzioni Debian/Ubuntu) o **wheel** (su CentOS/RHEL). I membri di questi gruppi hanno il diritto di usare `sudo` per eseguire comandi come amministratori [12]. Aggiungiamo *alice* al gruppo sudo:

```
user@server:~$ sudo usermod -aG sudo alice
```

**Spiegazione:** Abbiamo usato `usermod -aG` (append to group) per aggiungere Alice al gruppo `sudo`. (Alternativa: su Ubuntu avremmo potuto fare `sudo adduser alice sudo` equivalente). **Best practice:** non modifichiamo manualmente il file `/etc/sudoers` per ogni utente; la pratica predefinita e consigliata è appunto usare i gruppi standard [13]. In particolare, su Debian/Ubuntu tutti gli utenti nel gruppo "sudo" hanno privilegi sudo completi di default [14]. Allo stesso modo su CentOS/Fedora il gruppo "wheel" è spesso usato. Inserendo gli amministratori in questi gruppi eviti di dover elencare singolarmente ogni utente in sudoers (riducendo errori e semplificando la gestione) [13].

```
Per motivi di sicurezza, verifica ora i gruppi di appartenenza dell'utente e
prova i permessi sudo:
```

```
user@server:~$ groups alice
alice : alice sudo

user@server:~$ sudo -l -U alice
User alice may run the following commands on server:
    (ALL : ALL) ALL
```

**Spiegazione:** Il primo comando conferma che *alice* appartiene ai gruppi `alice` (privato) e `sudo`. Il secondo comando, `sudo -l -U alice`, mostra le regole sudo applicate ad Alice: in questo caso può eseguire **qualsiasi comando** come qualsiasi utente (`ALL : ALL`) usando sudo. Questo è l'impostazione standard quando aggiungi qualcuno al gruppo sudo.

> **Nota di sicurezza:** Garantire che solo personale autorizzato sia in questo gruppo, poiché equivale a dare poteri di root. In un contesto aziendale, mantenere un elenco aggiornato

degli utenti amministratori è fondamentale, e rimuovere subito chi non deve più averli (principio dei privilegi minimi e revoca tempestiva).

1. **Accesso con il nuovo utente (test):** Come verifica finale, effettuiamo il login con Alice (qui simuliamo con `su - alice`) e proviamo a eseguire un comando amministrativo con sudo, ad esempio elencare la cartella root (normalmente inaccessibile ai non-root):

```
user@server:~$ su - alice
alice@server:~$ sudo ls -la /root
[sudo] password for alice: ********
total 4
drwx------  2 root root   4096 Jul  8 16:00 .
drwxr-xr-x 18 root root   4096 Jul  8 12:30 ..
-rw-------  1 root root      0 Jul  8 16:00 secret.txt
```

**Spiegazione:** Come mostrato, ad Alice viene chiesta la sua password (non quella di root) al primo uso di sudo. Dopo l'autenticazione, il comando `ls /root` viene eseguito e listato con successo, segno che i privilegi sudo funzionano. Questo conferma che l'account è configurato correttamente: Alice può svolgere attività amministrative usando `sudo` (che logga ogni comando eseguito per audit) invece di accedere direttamente come root. Questo approccio è più sicuro e tracciabile: gli amministratori usano i propri account nominali con sudo, così ogni azione è riconducibile a una persona (grazie ai log di `/var/log/auth.log` o `journalctl`) e si evita di usare root direttamente se non necessario [15] [16].

**Riepilogo & Suggerimenti:**
- Usa `useradd` / `adduser` con attenzione: crea sempre la **home directory** (`-m`) e assegna una **shell valida** (`-s /bin/bash` o altra) per utenti reali.
- Imposta subito una **password robusta** con `passwd` e considera politiche di scadenza e complessità. Esempio: parametri *PASS_MAX_DAYS* e *PASS_MIN_DAYS* in `/etc/login.defs` (Lynis segnala spesso di configurarli) [17].
- **Sudo vs root:** Non dare in giro la password di root. Aggiungi gli utenti al gruppo *sudo* (o *wheel*) per i privilegi amministrativi [13]. Questo è lo standard su molte distro (Ubuntu incluse).
- Puoi verificare i permessi sudo di un utente con `sudo -l -U nomeutente`.
- Ogni tanto, **rivedi i membri** del gruppo sudo (vedi `/etc/group` o usa `getent group sudo`) per assicurarti che solo chi deve abbia accesso amministrativo.
- *Mnemonico:* **U-P-G** – Utente, Password, Gruppo: crea l'utente, imposta la password, aggiungilo al gruppo giusto (es. sudo) se deve avere privilegi. Non modificare direttamente sudoers a meno di esigenze speciali (ad es. concedere solo comandi specifici).

Il nuovo amministratore Alice ora ha un account sicuro e tracciabile. Passiamo alla sicurezza di rete e firewall del sistema.

## Scenario 4: Reti e Firewall

**Situazione:** Per proteggere il server, dobbiamo assicurarci che solo le porte necessarie siano aperte e che il firewall sia configurato correttamente. Inoltre, vogliamo verificare dall'esterno quali servizi risultano accessibili. In questo scenario combineremo strumenti di **ricognizione di rete** come `nmap` e

`netcat` con la configurazione di firewall usando `ufw` (Uncomplicated Firewall) e uno sguardo a `iptables`.

1. **Scansione delle porte aperte con `nmap`:** Prima di configurare il firewall, facciamo un audit delle porte attualmente aperte sul server. Usiamo `nmap` (Network Mapper) dal punto di vista di un client esterno. Supponiamo che il server abbia IP 10.0.0.10. Scansioniamo le porte comuni (1-10000) per vedere cosa è aperto:

```
# Eseguito da una macchina di scansione nella stessa rete
scanner$ nmap -p 1-10000 10.0.0.10

Starting Nmap 7.94 ( https://nmap.org ) at 2025-07-08 16:30 CEST
Nmap scan report for 10.0.0.10
Not shown: 9998 closed ports
PORT     STATE    SERVICE
22/tcp   open     ssh
80/tcp   open     http
4444/tcp open     krb524?
```

**Spiegazione:** L'output di nmap indica che sul server sono aperte la porta 22 (SSH), 80 (HTTP) e la famigerata 4444/tcp. Nmap addirittura ipotizza il servizio *krb524* su 4444 (potrebbe non riconoscere il servizio personalizzato, e tenta un'indovinazione). Questo conferma quanto visto nello scenario precedente: c'è una porta 4444 aperta (che sappiamo essere il malware). Nmap è uno strumento potentissimo per scansionare porte, scoprire host e servizi [18]. Viene usato sia dai difensori che dai pentester per **audit di sicurezza** delle reti [19]. Ad esempio, con opportuni flag può anche tentare di identificare versioni dei servizi, il sistema operativo della macchina, e persino fare scansioni di vulnerabilità basilari. Il suo nome stesso ("Network Mapper") ci ricorda lo scopo: *mappare la rete*. In ambito enterprise, è buona prassi eseguire scansioni periodiche sui propri server per assicurarsi che non ci siano servizi inaspettati aperti. Nmap infatti è utilizzato regolarmente per **auditing di server e rilevamento di porte aperte non autorizzate** [20]. Ora che conosciamo lo stato iniziale, procediamo a mettere in sicurezza le porte.

1. **Configurazione del firewall con `ufw` (frontend di iptables):** Il nostro server gira Ubuntu, che mette a disposizione **ufw (Uncomplicated Firewall)** come interfaccia semplificata per iptables. `ufw` è sostanzialmente un **frontend** per gestire regole netfilter/iptables in maniera più user-friendly [21]. Verifichiamo se il firewall è attivo:

```
user@server:~$ sudo ufw status
Status: inactive
```

Come pensavamo, al momento il firewall è disattivato (Status: inactive). Attiviamolo e consentiamo solo i servizi indispensabili, ad esempio SSH e HTTP:

```
user@server:~$ sudo ufw enable
Firewall is active and enabled on system startup

user@server:~$ sudo ufw allow OpenSSH
Rule added
```

```
Rule added (v6)

user@server:~$ sudo ufw allow 80/tcp
Rule added
Rule added (v6)
```

**Spiegazione:** Abbiamo abilitato `ufw` (che per default imposta una policy di deny all in ingresso, consentendo solo ciò che si aggiunge) e aggiunto regole per permettere SSH (porta 22) e HTTP (porta 80) sia su IPv4 che IPv6. `OpenSSH` è un *profilo applicativo* riconosciuto da ufw che corrisponde alla porta 22/tcp. Se ora ricontrolliamo lo status:

```
user@server:~$ sudo ufw status numbered
Status: active

     To                         Action      From
[ 1] OpenSSH                    ALLOW       Anywhere
[ 2] 80/tcp                     ALLOW       Anywhere
[ 3] OpenSSH (v6)               ALLOW       Anywhere (v6)
[ 4] 80/tcp (v6)                ALLOW       Anywhere (v6)
```

Vediamo le regole in vigore: tutto il resto è bloccato. In particolare, **la porta 4444 ora non è più accessibile** dall'esterno, perché non l'abbiamo permessa esplicitamente. In caso di malware ancora attivo, questo lo isolerebbe da eventuali controller remoti. Da amministratori, è importante avere un firewall attivo su server di produzione: anche se un servizio indesiderato parte, il firewall può mitigare i danni bloccando connessioni non autorizzate. `ufw` semplifica questo compito (accetta regole ad alto livello come "Allow 80/tcp" invece di scrivere comandi iptables complessi). Sotto al cofano, però, `ufw` *scrive regole iptables* equivalenti [21] . Infatti, se eseguissimo `sudo iptables -L -n -v`, vedremmo le catene di netfilter con le regole corrispondenti (inclusi i drop per le altre porte).

```
Vale la pena ricordare che iptables è uno strumento estremamente potente e
flessibile per gestire il traffico di rete in Linux【17†L92-L100】. Con
iptables puoi filtrare pacchetti, fare NAT, port forwarding, ecc. – il
rovescio della medaglia è che la sintassi può essere complessa e soggetta a
errori. Ufw e altri frontend (come Firewalld su CentOS/RHEL) esistono
proprio per rendere più **immediata** la gestione delle regole, specie per
configurazioni comuni. **Ufw** è quindi pensato per *semplificare* la vita
mantenendo la potenza di iptables sullo sfondo【19†L149-L157】. Laddove
servisse una configurazione più fine-grained, si può sempre ricorrere
direttamente a iptables o ai file di configurazione avanzati di ufw.
```

1. **Verifica del firewall con `netcat` (connessione di prova):** Ora simuliamo rapidamente un test di connettività. Usiamo `netcat` dalla macchina di scansione per verificare che la porta 4444 non sia più raggiungibile mentre SSH e HTTP sì:

```
scanner$ nc -zv 10.0.0.10 22
Connection to 10.0.0.10 22 port [tcp/ssh] succeeded!
```

```
scanner$ nc -zv 10.0.0.10 4444
nc: connect to 10.0.0.10 port 4444 (tcp) failed: Connection refused
```

**Spiegazione:** Il primo comando (`nc -zv`, zero I/O mode con output verboso) ci dice che la porta 22 su 10.0.0.10 risponde (successo, come atteso perché l'abbiamo aperta). Il secondo mostra "Connection refused" su 4444: segno che il firewall la sta bloccando attivamente (o che il servizio non è più in ascolto, in ogni caso il risultato è che la porta non è aperta esternamente). **Netcat (`nc`)** è noto come il "coltellino svizzero" della rete: qui lo abbiamo usato in modalità scanner per testare porte specifiche, ma può fare molto altro. Ad esempio, `nc -lvp 1234` su un host apre in ascolto la porta 1234 (utile per testare trasmissioni o simulare un server). Oppure, `nc host port` può fungere da client grezzo per inviare/ricevere dati. Saper usare netcat è utile sia per amministratori (per fare debug di servizi, trasferire file velocemente, ecc.) sia per capire cosa possono fare gli attacker (netcat spesso è usato per creare backdoor e tunnel). Nel nostro caso di test firewall, netcat ci conferma che **solo le porte autorizzate sono accessibili**.

1. **Cenno ad iptables e sue capacità:** Anche se `ufw` è sufficiente per la maggior parte delle esigenze, diamo uno sguardo minimale a iptables giusto per capire la correlazione. Listiamo le regole iptables attualmente in vigore:

```
user@server:~$ sudo iptables -L INPUT -n -v --line-numbers
Chain INPUT (policy DROP 0 packets, 0 bytes)
num   pkts bytes target     prot opt in   out  source      destination
1       15  1240 ACCEPT     tcp  --   *    *   0.0.0.0/0    0.0.0.0/0   tcp
dpt:22
2        8   480 ACCEPT     tcp  --   *    *   0.0.0.0/0    0.0.0.0/0   tcp
dpt:80
3        0     0 ACCEPT     all  --   lo   *   0.0.0.0/0    0.0.0.0/0
4        0     0 DROP       all  --   *    *   0.0.0.0/0    0.0.0.0/0   /*
ufw reject all */
```

**Spiegazione:** Questa è una possibile rappresentazione (semplificata) delle regole sulla chain INPUT: la policy di default è DROP (grazie a ufw), poi abbiamo regola 1 e 2 che **ACCEPT** traffico TCP alle porte 22 e 80 da ovunque, regola 3 che accetta tutto sul loopback (lo) e infine una regola di drop generale. I numeri di pacchetti/byte indicano anche quante connessioni hanno colpito ciascuna (ad esempio 15 pacchetti su regola1, forse tentativi SSH). **iptables** lavora con catene di regole, ognuna con condizioni (es. porta, protocollo, interfaccia) e azioni (target) [23]. I target possono essere ACCEPT, DROP, REJECT, LOG, ecc. In Linux, iptables è il nucleo del firewall: **un pilastro della sicurezza** perché permette di definire politiche su misura per filtrare o modificare il traffico [22]. Tuttavia, come visto, configurarlo manualmente richiede precisione. Frontend come ufw semplicemente generano queste regole per noi.

> **Riepilogo & Suggerimenti:**
> - **Nmap** (Network Mapper) è il tuo amico per scansionare e **mappare porte aperte** e servizi. Usalo per vedere il tuo sistema con gli occhi di un attaccante e scoprire eventuali falle [20] [18].
> - Tieni attivo un **firewall** sul server. Su Ubuntu, abilita `ufw` con poche regole per i servizi necessari. Ricorda che **ufw è un front-end**: crea regole iptables equivalenti dietro le quinte [21].
> - Lascia **chiuse** (o filtrate) tutte le porte non utilizzate. Se un servizio deve essere accessibile solo all'interno della LAN, considera regole più restrittive (ufw consente di specificare l'origine, es: `ufw allow from 192.168.0.0/24 to any port 3306` per

MySQL interno).
- **Netcat** è utile per test veloci di connettività (come `nc -zv` per scan di porte specifiche) e per debug. Se un servizio non risponde, prova con netcat a connetterti alla porta per vedere se è raggiungibile o se magari il firewall scarta i pacchetti.
- *Mnemonico:* pensa **"Lockdown"** – dopo aver impostato i servizi necessari, esegui il *lockdown* del resto col firewall. Inoltre, Nmap ti aiuta a *vedere* se il lockdown è effettivamente in vigore dall'esterno.

Il server ora ha un firewall attivo e lascia passare solo ciò che serve. Abbiamo ridotto la superficie d'attacco di rete. Infine, passiamo agli strumenti di **analisi e hardening** per completare la messa in sicurezza e verifica del sistema.

## Scenario 5: Analisi e Hardening del Sistema

**Situazione:** Dopo aver gestito la minaccia immediata, vuoi assicurarti che il sistema sia pulito e rafforzato contro futuri attacchi. Ciò include **scansioni anti-rootkit**, un **audit di sicurezza generale**, l'implementazione di un sistema di **difesa proattiva (fail2ban)** e la verifica dei **log di audit** per tracciare attività critiche. In questo scenario useremo `chkrootkit`, `lynis`, `fail2ban` e `auditd` per questi scopi.

1. **Scansione anti-rootkit con `chkrootkit`:** *Chkrootkit* (Check Rootkit) è uno strumento che verifica la presenza di rootkit noti e altri indicatori di compromissione nel sistema. Eseguiamolo:

```
user@server:~$ sudo chkrootkit
Initializing chkrootkit...
Checking `passwd'... OK
Checking `sshd'... OK
Checking `suspscan'... not infected
...
```

**Spiegazione:** L'output abbreviato sopra mostra alcuni controlli effettuati da chkrootkit (controlla l'integrità di file di sistema come `passwd`, verifica processi nascosti, etc.). Fortunatamente risulta **"not infected"** nei vari test. Chkrootkit è come una "guardia del corpo digitale" che cerca segni di rootkit e malware nascosti [24] [25]. È particolarmente utile dopo un sospetto incidente: può rilevare se qualche backdoor nota è presente, se comandi di sistema sono stati alterati, o se ci sono moduli del kernel malevoli in memoria. Tieni presente che *nessun scanner è infallibile*, e che a volte possono comparire **falsi positivi** (es. chkrootkit segnala "suspicious files" che però sono legittimi). Infatti, la documentazione di chkrootkit nota che alcuni output vanno interpretati con cautela e talvolta possono essere dovuti a normali attività di sistema [26]. In ogni caso, **eseguirlo regolarmente** (ad esempio schedulandolo via cron) fa parte di una buona routine di monitoraggio [27]. Nel nostro scenario, usando chkrootkit abbiamo ottenuto un parere in più sul fatto che il sistema sia pulito.

1. **Audit di sicurezza completo con `lynis`:** Ora passiamo a *Lynis*, un potente strumento di **auditing e hardening** per sistemi Unix/Linux. Lynis effettua decine di controlli sul sistema (configurazioni, permessi, parametri di sicurezza) e produce un report con **avvisi e suggerimenti** [28] [29]. Lancialo in modalità audit di sistema:

```
user@server:~$ sudo lynis audit system
[+] Initializing Lynis - Security Audit Tool
```

```
... (output omesso per brevità, Lynis esegue molti test) ...

[+] Lynis security scan complete.

================================
  Lynis Audit Report
================================
  Version          : 3.0.8
  Hostname         : server
  Score            : 70 [Hardening Index]
--------------------------------
  Tests performed  : 212
  Warnings found   : 2
  Suggestions      : 4
--------------------------------

[+] Suggestions (output abbreviated):
--------------------------------
[SSH-7408] SSH: Consider hardening SSH configuration (e.g. Disable root
login)
[AUTH-9286] Auth: Configure minimum and maximum password age in /etc/
login.defs [17]
[NETW-3032] Networking: Consider running ARP monitoring software (arpwatch)
[30]
[FIRE-4513] Firewall: Check iptables rules for unused ones [31]
--------------------------------
```

**Spiegazione:** Il rapporto di Lynis indica un punteggio di hardening del 70 (su 100). Ha trovato 2 *Warning* (problemi più seri) e 4 *Suggestions* (miglioramenti consigliati). Nel riepilogo sopra vediamo ad esempio: Lynis suggerisce di **disabilitare il login root via SSH**, impostare politiche di **scadenza password** [17], attivare un monitor ARP e ricontrollare le regole firewall. Queste raccomandazioni corrispondono a best practice note: ad esempio *PermitRootLogin no* in sshd_config aumenta la sicurezza SSH, e parametri come PASS_MAX_DAYS implementano la rotazione password obbligatoria. Lynis effettua centinaia di controlli (dai permessi di file critici, alle configurazioni di rete, ai servizi in esecuzione) [28] e fornisce sia un log dettagliato (`/var/log/lynis.log`) sia un file di report strutturato con tutti i risultati e ID dei test. In produzione, dovresti rivedere uno per uno questi suggerimenti e applicare quelli pertinenti, poi rieseguire Lynis per vedere il punteggio salire. Lynis è ampiamente usato da sysadmin e auditor per **valutare lo stato di sicurezza** di sistemi e garantire la conformità a standard (CIS, PCI-DSS, etc.) [32].

> **Nota:** Lynis assegna un Hardening Index (70 nel nostro caso). In generale, punteggi più alti indicano un sistema più "robusto", ma non bisogna fissarsi solo sul numero: l'importante è capire e implementare le raccomandazioni pertinenti. Lynis è **modulare e flessibile** [33]; per esempio, se in un contesto la regola X non è applicabile, puoi contrassegnarla come giustificata. Un approccio iterativo (scansione -> correggi -> scansione) porta a un sistema notevolmente più sicuro.

1. **Protezione dai brute-force con `fail2ban`:** Adesso implementiamo una misura difensiva attiva. *Fail2ban* è uno strumento che **monitora i log** alla ricerca di attività malevole (tipicamente tentativi di login falliti ripetuti) e **banna automaticamente** gli IP sorgenti bloccandoli sul firewall per un certo tempo [34]. Nel nostro scenario, abbiamo visto tentativi di login SSH falliti da

10.0.0.5. Fail2ban può gestire questo tipo di attacco. Supponiamo di aver installato e già avviato fail2ban con una jail per SSH. Controlliamo lo stato:

```
user@server:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  `- Total failed: 5
`- Actions
   |- Currently banned: 1
   `- Total banned: 1
```

**Spiegazione:** Il comando `fail2ban-client status sshd` ci mostra che per la *jail* SSH (ovvero la sezione di configurazione che protegge il servizio SSH) finora ci sono stati 5 tentativi di accesso falliti, e attualmente 1 IP è in stato di **bannato**. È probabile che l'IP bannato sia proprio *10.0.0.5* che abbiamo visto prima. Infatti, per default fail2ban con jail sshd banna un IP dopo, ad esempio, 5 tentativi errati in 10 minuti per 10 minuti (configurazione tipica: maxretry=5, findtime=600, bantime=600). Fail2ban funziona monitorando continuamente i file di log (come `/var/log/auth.log`) con espressioni regolari: quando vede X tentativi di login falliti, esegue un'azione, tipicamente aggiungendo una regola firewall (iptables) per droppare il traffico da quell'IP [35] [36]. Questo meccanismo aiuta moltissimo a ridurre i rischi di brute-force e scanning automatici, perché blocca sul nascere gli IP malevoli per un periodo. Naturalmente, fail2ban va configurato con giudizio (ad esempio evitando di bannare IP interni o IP legittimi che magari sbagliano password). I *jails* disponibili coprono molti servizi: SSH, FTP, web server, posta, ecc. [37]. Fail2ban è in pratica un **IDS/IPS leggero** a livello host: identifica comportamenti sospetti (intrusion detection) e reagisce bloccandoli (intrusion prevention) [34].

Nel nostro scenario, grazie a fail2ban l'IP dell'attaccante che provava password su SSH è stato automaticamente bloccato dopo 5 tentativi. Possiamo anche controllare il file di log di fail2ban (`/var/log/fail2ban.log`) per vedere quando è scattato il ban.

> **Suggerimenti su fail2ban:**
> - Configura le *jail* pertinenti ai servizi esposti: es. `sshd` per SSH, `nginx-http-auth` per proteggere aree web con autenticazione, etc.
> - Puoi personalizzare **bantime** (durata ban) e **maxretry** (tentativi) in `/etc/fail2ban/jail.local` per adattarli alle tue esigenze.
> - Ricorda che fail2ban utilizza il firewall sottostante (iptables o nftables) per applicare i ban [38]. Quindi assicurati che non confligga con altre regole firewall.
> - Monitorare i log di fail2ban ti fa capire se stai subendo attacchi frequenti. Ad esempio, molti ban su SSH indicano tentativi di brute-force in corso (abbastanza comuni su server esposti).
> - **Mnemonico:** *fail2ban – "fallire per essere bannati"*: se un IP fallisce troppi login, viene bannato automaticamente. Questo ti aiuta a ricordare immediatamente cosa fa lo strumento.

1. **Tracciabilità con i log di audit (`auditd`):** Infine, diamo importanza alla **tracciabilità** delle azioni sul sistema. `auditd` è il demone del Linux Audit System, che registra eventi di sicurezza a basso livello (sistemi di autenticazione, accesso a file critici, chiamate di sistema sensibili, ecc.) con grande dettaglio [39]. In ambienti corporate, auditd è spesso abilitato per motivi di compliance (es. PCI-DSS, SOX) e investigazione forense. Verifichiamo che auditd sia in esecuzione e diamo un'occhiata ad un esempio di log audit:

```
user@server:~$ sudo systemctl status auditd | grep Active
Active: active (running) since Tue 2025-07-08 15:00:00 CEST; 1h 30min ago

user@server:~$ sudo tail -n 2 /var/log/audit/audit.log
type=USER_ADD msg=audit: <...> user pid=2000 uid=0 auid=0 subj==unconfined
res=success msg='uid=1001 exe=\"/usr/sbin/useradd\" hostname=server addr=?
terminal=pts/1 res=success'
type=ADD_GROUP msg=audit: <...> user pid=2001 uid=0 auid=0 subj==unconfined
res=success msg='op=adding user to group auid=1001 uid=0 gid=27 exe=\"/usr/
bin/gpasswd\" hostname=server addr=? terminal=pts/1 res=success'
```

**Spiegazione:** Il servizio auditd risulta attivo (come previsto su un sistema ben configurato). Le ultime due righe di log mostrano eventi generati quando abbiamo creato l'utente *alice* e l'abbiamo aggiunta al gruppo sudo. In particolare, vediamo un evento `USER_ADD` con esito success (`res=success`) tramite il comando `/usr/sbin/useradd`, e un evento `ADD_GROUP` per l'operazione di aggiunta al gruppo (gpasswd eseguito per aggiungerla a sudo). Notare i dettagli: ogni evento audit riporta l'UID reale ed effective (uid=0 indica che l'operazione è stata fatta da root, presumibilmente tramite il nostro account sudo), l'eseguibile coinvolto, il terminale e altro. **Auditd fornisce un livello di dettaglio approfondito** di ciò che accade nel sistema, permettendo di ricostruire il chi-cosa-quando delle azioni sensibili [40] [41]. Ad esempio, può loggare: accessi ai file (chi ha letto/modificato un file specifico), comandi `sudo` eseguiti, modifiche ai permessi, avvii di processi, e molto altro [42] [43].

Nel nostro scenario di esempio, grazie ai log di audit possiamo dimostrare la creazione dell'account Alice e la sua aggiunta ai sudoers, il che è utile per verifiche di compliance (es. un auditor potrebbe chiedere: "mostrami l'evidenza che l'account amministrativo X è stato creato il giorno Y da un utente autorizzato"). Auditd risponde a questo tipo di domande. Naturalmente, per evitare un diluvio di dati, si definiscono **regole di audit** mirate (in `/etc/audit/rules.d/` o usando `auditctl`). Ad esempio, se volessimo monitorare ogni modifica al file `/etc/passwd`, aggiungeremmo una regola: `-w /etc/passwd -p wa -k passwd_changes`. Così avremmo un evento audit ogni volta che quel file viene scritto o modificato (write/attribute change), con tag `passwd_changes` per trovarlo facilmente nei log.

> **Valore di auditd:** I log di auditd offrono una **visibilità totale** sul sistema, fondamentale per *sicurezza, compliance e troubleshooting* [39]. In caso di breach, poter consultare i log audit può aiutare a capire la cronologia delle azioni dell'attaccante (es. quali comandi ha eseguito, quali file ha toccato). Per la compliance normativa, molti standard richiedono l'audit trail degli accessi e delle modifiche ai dati sensibili [44] [45] – auditd risponde a questi requisiti catturando *chi ha fatto cosa*.
>
> **Suggerimenti finali:**
> - Mantieni auditd attivo sui server critici. Utilizza regole focalizzate su eventi chiave (es. modifica utenti, accesso a file critici, riavvio dei servizi) [40], così da non essere sommerso da dati ma avere ciò che serve.
> - Esegui periodicamente strumenti come **chkrootkit** e **rkhunter** per controlli anti-malware, specie dopo incidenti.
> - Integra strumenti di audit/hardening come **Lynis** nei processi di hardening iniziale e nelle verifiche periodiche (ad es. esegui Lynis ogni mese e segui le sue raccomandazioni) [32] [29].
> - Implementa difese attive come **fail2ban** sulle porte esposte per ridurre il rumore di attacco e prevenire brute-force [34].
> - **Difesa in profondità:** Nessun singolo strumento basta. La sicurezza efficace nasce dalla

combinazione di misure: firewall, aggiornamenti costanti, monitoraggio log (manuale o con SIEM), strumenti anti-intrusione, backup, ecc. Gli strumenti visti (chkrootkit, Lynis, fail2ban, auditd) sono tasselli di un mosaico più grande che, usati insieme, alzano di molto il livello di sicurezza.

## Conclusioni

In questo percorso interattivo abbiamo affrontato scenari realistici di sicurezza su Linux in ambito aziendale, coprendo un ampio ventaglio di comandi e strumenti. Hai imparato a:

- **Gestire file e permessi** in modo sicuro, individuando e correggendo configurazioni errate (`ls`, `find`, `chmod`, `chown`).
- **Monitorare il sistema e analizzare i log** per scoprire attività sospette (`journalctl`, `top`, `ps`, `netstat` / `ss`), identificando processi malevoli e connessioni non autorizzate.
- **Gestire gli utenti** applicando il principio dei privilegi minimi, creando account amministrativi sicuri e usando `sudo` al posto di root (`useradd`, `passwd`, gruppi sudo).
- **Proteggere la rete e i servizi** con firewall, chiudendo porte inutili e testandone dall'esterno l'accessibilità (`iptables` / `ufw`, `nmap`, `netcat`).
- **Analizzare e hardenizzare il sistema** utilizzando scanner di sicurezza e misure preventive (`chkrootkit`, `lynis` per audit; `fail2ban` per blocco automatico; `auditd` per tracciare eventi di sicurezza).

Questa esperienza ha messo in luce l'importanza di un approccio **olistico** alla cybersecurity: non basta un solo comando o tool, ma serve conoscere e combinare più strumenti. Ricorda di fare tesoro dei **mnemonici** e delle logiche apprese (ad es. *CH* per i comandi di *CHange*, *L-P-N* per Log-Processi-Rete, *fail2ban* come concetto di "troppi fail = ban", etc.) per richiamare rapidamente i comandi giusti al momento giusto.

Se hai completato gli esercizi mentalmente seguendo il terminale simulato, avrai rafforzato le tue competenze pratiche. Puoi ora applicare questi comandi sul campo, in un ambiente reale o di laboratorio, per ulteriore pratica. La sicurezza informatica è un processo continuo: continua a esplorare, aggiornarti e affinare le tue capacità. Buon hardening del tuo sistema Linux!

**Fonti utilizzate e consigliate:**


- Red Hat Blog – *Audit dei permessi con find*: l'uso di find per individuare file con permessi specifici e conformità alle policy [1].
- Last9 Blog – *Journalctl Commands Cheatsheet*: importanza del journal systemd per centralizzare e filtrare log [2].
- Medium (Faruk Ahmed) – *Identificare processi sospetti*: perché usare ps oltre a top, e cosa cercare nei processi (shell anomale, eseguibili in /tmp) [3] [4].
- SANS Blog – *Uso di ss per analisi rete*: ss come rimpiazzo moderno di netstat, più veloce e informativo, utile per incident response [5] [7].
- Medium (Faruk Ahmed) – *Netstat/ss e connessioni sospette*: importanza di controllare porte aperte e connessioni verso l'esterno, malware che "call home" [8] [9].
- GeeksforGeeks – *iptables in Linux*: panoramica sull'uso di iptables per filtrare traffico e importanza per la sicurezza dei server [22].
- StackExchange Secuity – *UFW vs iptables*: UFW è un frontend di iptables per facilitarne l'uso, iptables offre più flessibilità a costo di complessità [21].

- RecordedFuture – *Nmap Commands Cheat Sheet*: Nmap come strumento per mappare reti, scoprire host e porte aperte, audit di sicurezza su sistemi locali e remoti [19] [18].
- Liquid Web – *What is chkrootkit?*: spiegazione di chkrootkit e perché usarlo per scovare rootkit e anomalie, utile per verificare l'integrità del sistema [25].
- HackTheForum – *Fail2Ban*: panoramica di fail2ban come software di prevenzione intrusioni che monitora log e banna IP per proteggere da brute-force su SSH e altri servizi [34].
- Last9 Blog – *Auditd logs*: valore dei log di auditd per avere visibilità approfondita sul sistema, fondamentale per sicurezza e compliance [39].
- StackExchange – *Best practice sudo*: consenso sulla best practice di usare il gruppo sudo/wheel per privilegi amministrativi invece di inserire utenti singolarmente in sudoers [13].

---


[1] How to audit permissions with the find command
https://www.redhat.com/en/blog/audit-permissions-find

[2] journalctl Commands Cheatsheet for Troubleshooting | Last9
https://last9.io/blog/journalctl-commands-cheatsheet/

[3] [4] How I Spot a Suspicious Process on My Linux Server (Before It Does Damage) | by Faruk Ahmed | Jun, 2025 | Medium
https://medium.com/@bornaly/how-i-spot-a-suspicious-process-on-my-linux-server-before-it-does-damage-b9092ce88f6f

[5] [6] [7] Linux Incident Response - Using ss for Network Analysis | SANS
https://www.sans.org/blog/linux-incident-response-using-ss-for-network-analysis/

[8] [9] [10] How I Use netstat and ss to Catch Suspicious Connections on Linux | by Faruk Ahmed | Jul, 2025 | Medium
https://medium.com/@bornaly/how-i-use-netstat-and-ss-to-catch-suspicious-connections-on-linux-ee69f93a57c2

[11] [17] [30] [31] Install Lynis and Fix Some Suggestions | Karim's Blog
https://elatov.github.io/2017/06/install-lynis-and-fix-some-suggestions/

[12] [14] How To Create a New Sudo-Enabled User on Ubuntu | DigitalOcean
https://www.digitalocean.com/community/tutorials/how-to-create-a-new-sudo-enabled-user-on-ubuntu

[13] [15] [16] sudo - From a security standpoint should I add my user to the sudoers file or not? - Unix & Linux Stack Exchange
https://unix.stackexchange.com/questions/382955/from-a-security-standpoint-should-i-add-my-user-to-the-sudoers-file-or-not

[18] [19] [20] Top 16 Nmap Commands: Nmap Port Scan Cheat Sheet
https://www.recordedfuture.com/threat-intelligence-101/tools-and-techniques/nmap-commands

[21] firewalls - UFW vs IpTables for web application security - Information Security Stack Exchange
https://security.stackexchange.com/questions/260828/ufw-vs-iptables-for-web-application-security

[22] [23] iptables command in Linux with Examples - GeeksforGeeks
https://www.geeksforgeeks.org/linux-unix/iptables-command-in-linux-with-examples/

[24] [25] [26] [27] What is chkrootkit? | Liquid Web
https://www.liquidweb.com/help-docs/what-is-chkrootkit/

[28] [32] [33] Lynis - Security auditing tool for Linux, macOS, and Unix-based systems - CISOfy
https://cisofy.com/lynis/

[29] How to deal with Lynis suggestions? - Linux Audit
https://linux-audit.com/lynis/how-to-deal-with-lynis-suggestions/

34 35 36 37 38 Fail2Ban – Cyber Security – Hack The Forum
https://www.hacktheforum.com/cyber-security/fail2ban/

39 40 41 42 43 44 45 How Auditd Logs Help Secure Linux Environments | Last9
https://last9.io/blog/auditd-logs/