Report Analisi Filesystem

```
[analyst@secOps ~]$ cd /
[analyst@secOps /]$ ls -l
total 52
lrwxrwxrwx   1 root root     7 May  3 15:26 bin -> usr/bin
drwxr-xr-x   3 root root  4096 Jun 18 19:07 boot
drwxr-xr-x  20 root root  3920 Sep 29 09:15 dev
drwxr-xr-x  73 root root  4096 Jun 19 04:45 etc
drwxr-xr-x   3 root root  4096 Mar 20  2018 home
lrwxrwxrwx   1 root root     7 May  3 15:26 lib -> usr/lib
lrwxrwxrwx   1 root root     7 May  3 15:26 lib64 -> usr/lib
drwx------   2 root root 16384 Mar 20  2018 lost+found
drwxr-xr-x   2 root root  4096 Jan  5  2018 mnt
drwxr-xr-x   3 root root  4096 Jun 17 15:07 opt
dr-xr-xr-x 202 root root     0 Sep 29 09:15 proc
drwxr-x---   8 root root  4096 Jun 18 20:09 root
drwxr-xr-x  22 root root   580 Sep 29 09:15 run
lrwxrwxrwx   1 root root     7 May  3 15:26 sbin -> usr/bin
drwxr-xr-x   6 root root  4096 Mar 24  2018 srv
dr-xr-xr-x  13 root root     0 Sep 29 09:15 sys
drwxrwxrwt  11 root root   260 Sep 29 09:16 tmp
drwxr-xr-x  10 root root  4096 Jun 19 03:15 usr
drwxr-xr-x  12 root root  4096 Jun 19 04:45 var
```

1. **Qual è il significato dell'output?**

   Il comando mount | grep sda1 mostra che la partizione **/dev/sda1** è montata come filesystem root (/), con filesystem **ext4**.
   Il comando ls -l / elenca le directory principali presenti in root del filesystem

---

2. Dove sono fisicamente memorizzati i file elencati?
I file e le directory mostrati in root (/) sono fisicamente memorizzati nella partizione **/dev/sda1** del disco principale, formattata in **ext4**.

```
[analyst@secOps ~]$ mount | grep sda1
/dev/sda1 on / type ext4 (rw,relatime)
```

3. Perché /dev/sdb1 non viene mostrato nell'output sopra?
La partizione **/dev/sdb1** non compare nell'output di mount perché non è montata in alcun punto del filesystem. Di conseguenza, i suoi contenuti non sono accessibili fino a quando non viene eseguita un'operazione di mount manuale o configurata in /etc/fstab.

```
[analyst@secOps ~]$ ls -l second_drive/
total 0
[analyst@secOps ~]$ sudo mount /dev/sdb1 ~/second_drive/
[sudo] password for analyst:
[analyst@secOps ~]$ ls -l second_drive/
total 20
drwx------ 2 root    root    16384 Mar 26  2018 lost+found
-rw-r--r-- 1 analyst analyst   183 Mar 26  2018 myFile.txt
[analyst@secOps ~]$
```

4. Perché la directory non è più vuota?
Prima del comando mount, la directory second_drive/ era semplicemente una cartella vuota del filesystem root. Dopo aver eseguito

sudo mount /dev/sdb1 ~/second_drive/

quella directory è diventata il **punto di mount** della partizione **/dev/sdb1**. Per questo motivo ora mostra i file e le directory contenuti all'interno di sdb1 (in questo caso lost+found e myFile.txt).

---

5. Dove sono fisicamente memorizzati i file elencati?
I file visualizzati dentro second_drive/ (es. myFile.txt) non si trovano più nella partizione principale **/dev/sda1**, bensì sono fisicamente memorizzati nella partizione **/dev/sdb1** (1 GB).

Il comando **mount | grep /dev/sd** mostrava che la partizione **/dev/sdb1** era montata sulla directory /home/analyst/second_drive. Dopo l'esecuzione di

sudo umount /dev/sdb1

la partizione è stata smontata. A quel punto, la directory second_drive/ non rappresenta più il contenuto della partizione, ma torna a essere la semplice cartella vuota che era stata creata inizialmente.

Quando un filesystem viene montato su una directory, il suo contenuto “sovrascrive” temporaneamente quello della directory. Smontando (umount), quel collegamento viene rimosso e la directory ritorna a mostrare il proprio contenuto originario. Nel caso di second_drive/, essendo stata creata come cartella vuota, dopo lo smontaggio risulta nuovamente vuota.

```
[analyst@secOps ~]$  cd lab.support.files/scripts/
[analyst@secOps scripts]$ ls -l
total 68
-rwxr-xr-x 1 analyst analyst  952 Mar 21  2018 configure_as_dhcp.sh
-rwxr-xr-x 1 analyst analyst 1153 Mar 21  2018 configure_as_static.sh
-rwxr-xr-x 1 analyst analyst 4053 Jun 18 20:09 cyberops_extended_topo_no_fw.py
-rwxr-xr-x 1 analyst analyst 5016 Jun 18 20:07 cyberops_extended_topo.py
-rwxr-xr-x 1 analyst analyst 4189 Jun 18 19:22 cyberops_topo.py
-rw-r--r-- 1 analyst analyst 2871 Mar 21  2018 cyops.mn
-rwxr-xr-x 1 analyst analyst  458 Mar 21  2018 fw_rules
-rwxr-xr-x 1 analyst analyst   70 Mar 21  2018 mal_server_start.sh
drwxr-xr-x 2 analyst analyst 4096 Mar 21  2018 net_configuration_files
-rwxr-xr-x 1 analyst analyst   65 Mar 21  2018 reg_server_start.sh
-rwxr-xr-x 1 analyst analyst  189 Mar 21  2018 start_ELK.sh
-rwxr-xr-x 1 analyst analyst   86 Jun 18 20:27 start_miniedit.sh
-rwxr-xr-x 1 analyst analyst   86 Jun 19 03:16 start_pox.sh
-rwxr-xr-x 1 analyst analyst  117 Jun 19 03:31 start_snort.sh
-rwxr-xr-x 1 analyst analyst   61 Mar 21  2018 start_tftpd.sh
```

6. Chi è il proprietario del file?
Il proprietario è l’utente **analyst**.

---

7. Chi è il gruppo associato al file?
Il gruppo è anch’esso **analyst**.

---

8. I permessi per cyops.mn sono -rw-r--r--. Cosa significa?
La stringa si legge così:

- **-** → è un file regolare (non directory, non link, ecc.).
- **rw-** → analyst ha permesso di **lettura** e **scrittura**.
- **r--** → i membri del gruppo analyst hanno solo permesso di **lettura**.
- **r--** → tutti gli altri utenti hanno solo permesso di **lettura**.
- Mentre per l'esecuzione nessuno ha il permesso.

In sintesi:

- **Proprietario:** può leggere e modificare il file.
- **Gruppo:** può solo leggerlo.
- **Altri:** possono solo leggerlo.

```
[analyst@secOps ~]$ ls -ld /mnt
drwxr-xr-x 2 root root 4096 Jan  5  2018 /mnt
```

9. Perché il file non è stato creato?
Il comando touch /mnt/myNewFile.txt fallisce con "Permission denied" perché la directory **/mnt** appartiene all'utente **root** ed ha permessi **drwxr-xr-x**.

- Il proprietario (root) ha **lettura, scrittura ed esecuzione**.
- Il gruppo (root) ha **lettura ed esecuzione**.
- Gli altri (tutti gli utenti, incluso analyst) hanno solo **lettura ed esecuzione**.

Poiché l'utente analyst non è né proprietario né parte del gruppo root, non ha il permesso di scrivere in /mnt.

---

10. Elenca i permessi, la proprietà e il contenuto della directory /mnt.

- **Permessi:** drwxr-xr-x
- **Proprietario:** root
- **Gruppo:** root
- **Contenuto:** la directory appare vuota.

```
[analyst@secOps ~]$ cd /mnt
[analyst@secOps mnt]$ ls
```

---

11. Con l’opzione -d, elenca i permessi della directory genitore.
L’output di ls -ld /mnt conferma:

drwxr-xr-x 2 root root 4096 Jan 5 2018 /mnt

Questo significa che la directory è scrivibile solo da root.

---

12. Cosa si può fare affinché il comando touch /mnt/myNewFile.txt abbia successo?
Ci sono diverse soluzioni:

1. **Eseguire il comando come root** (ad esempio con sudo touch /mnt/myNewFile.txt).
2. **Cambiare la proprietà della directory** ad analyst o aggiungere analyst al gruppo con permessi di scrittura, ad esempio:

   sudo chown analyst:analyst /mnt

3. **Modificare i permessi della directory /mnt** per consentire la scrittura anche ad altri utenti, ad esempio:

   sudo chmod 777 /mnt

13. Quali sono i permessi del file myFile.txt inizialmente?
All’inizio il file aveva i permessi:

-rw-r--r--

- **Owner (analyst):** lettura e scrittura.
- **Group (analyst):** sola lettura.
- **Others:** sola lettura.

---

14. I permessi sono cambiati dopo il comando chmod 665 myFile.txt?
Sì, sono cambiati. Dopo l’esecuzione di chmod 665, i permessi risultano:

-rw-rw-r-x

---

15. Quali sono ora i permessi di myFile.txt?

- **Analyst:** lettura e scrittura.
- **Analyst:** lettura e scrittura.
- **Others:** lettura ed esecuzione.

16. Quale comando cambierebbe i permessi di myFile.txt a rwxrwxrwx?

Il comando è: chmod 777 myFile.txt

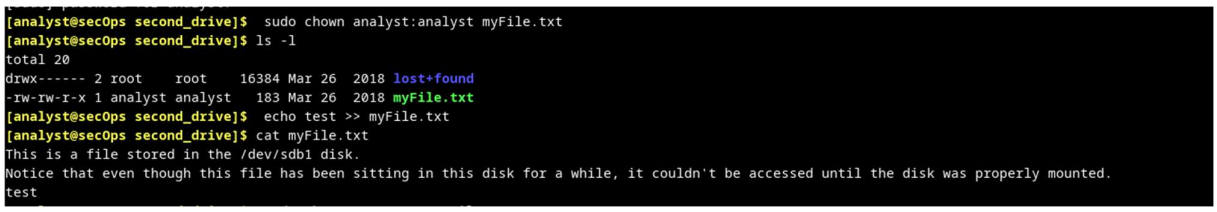
In ottale, **7** corrisponde a **111** in binario → rwx (lettura, scrittura ed esecuzione).

Con 777, i permessi vengono applicati a:

- **Proprietario:** rwx
- **Gruppo:** rwx
- **Altri:** rwx

- **Nota:** il valore 777 assegna pieno accesso (lettura, scrittura ed esecuzione) a proprietario, gruppo e tutti gli altri utenti. Questa configurazione è molto permissiva ed è **sconsigliata in ambienti di produzione** perché elimina ogni controllo sugli accessi al file.

17. L'operazione è riuscita? Spiega.



Il comando sudo chown analyst:analyst myFile.txt è stato eseguito con successo, ma non ha prodotto un cambiamento reale: il file era già di proprietà dell'utente analyst e del gruppo analyst. L'output di ls -l conferma infatti che la proprietà è rimasta invariata.
Successivamente, l'utente analyst ha potuto modificare il file (echo test >> myFile.txt) perché, in quanto proprietario, aveva i permessi di lettura e

scrittura. Il comando cat ha mostrato sia il contenuto originale sia la nuova riga aggiunta.

Per curiosità è stato eseguito anche il comando:

```
[analyst@secOps second_drive]$  sudo chown root:root myFile.txt
[analyst@secOps second_drive]$ ls -l
total 20
drwx------ 2 root root 16384 Mar 26  2018 lost+found
-rw-rw-r-x 1 root root   188 Sep 29 10:37 myFile.txt
[analyst@secOps second_drive]$  echo root >> myFile.txt
bash: myFile.txt: Permission denied
[analyst@secOps second_drive]$  echo test >> myFile.txt
bash: myFile.txt: Permission denied
```

sudo chown root:root myFile.txt

In questo modo la proprietà del file è passata a root:root.

Risultato:

- L'utente analyst non ha più potuto scrivere sul file, poiché rientra nella categoria "others" che ha solo permessi di lettura ed esecuzione.
- I tentativi di scrittura (echo >>) sono quindi falliti con errore **Permission denied**.

18. Qual è la differenza tra la parte iniziale della riga di malware e quella di mininet_services?

```
[analyst@secOps ~]$  cd ~/lab.support.files/
[analyst@secOps lab.support.files]$ ls -l
total 580
-rw-r--r-- 1 analyst analyst    649 Mar 21  2018 apache_in_epoch.log
-rw-r--r-- 1 analyst analyst    126 Mar 21  2018 applicationX_in_epoch.log
drwxr-xr-x 4 analyst analyst   4096 Mar 21  2018 attack_scripts
-rw-r--r-- 1 analyst analyst    102 Mar 21  2018 confidential.txt
-rw-r--r-- 1 analyst analyst   2871 Mar 21  2018 cyops.mn
-rw-r--r-- 1 analyst analyst     75 Mar 21  2018 elk_services
-rw-r--r-- 1 analyst analyst    373 Mar 21  2018 h2_dropbear.banner
drwxr-xr-x 2 analyst analyst   4096 Apr  2  2018 instructor
-rw-r--r-- 1 analyst analyst    255 Mar 21  2018 letter_to_grandma.txt
-rw-r--r-- 1 analyst analyst  24464 Mar 21  2018 logstash-tutorial.log
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 malware
-rwxr-xr-x 1 analyst analyst    172 Mar 21  2018 mininet_services
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 openssl_lab
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 pcaps
drwxr-xr-x 6 analyst analyst   4096 Aug 15  2022 pox
-rw-r--r-- 1 analyst analyst 473363 Mar 21  2018 sample.img
-rw-r--r-- 1 analyst analyst     65 Mar 21  2018 sample.img_SHA256.sig
drwxr-xr-x 3 analyst analyst   4096 Jun 18 20:07 scripts
-rw-r--r-- 1 analyst analyst  25553 Mar 21  2018 SQL_Lab.pcap
```

Confronto tra malware e mininet_services

Analizziamo le differenze tra la parte iniziale della riga di malware e quella di mininet_services:

malware

drwxr-xr-x 2 analyst analyst ...

La riga inizia con d, indicando che malware è una directory.

I permessi sono rwxr-xr-x:

- Owner: lettura, scrittura, esecuzione
- Gruppo: lettura, esecuzione
- Others: lettura, esecuzione

mininet_services

-rwxr-xr-x 1 analyst analyst

La riga inizia con -, indicando che mininet_services è un file regolare.

I permessi sono rwxr-xr-x:

- Owner: lettura, scrittura, esecuzione
- Gruppo: lettura, esecuzione
- Others: lettura, esecuzione

Differenza principale

La differenza fondamentale riguarda il tipo di oggetto: malware è una directory, mentre mininet_services è un file eseguibile. Questo è identificato dal primo carattere della riga.

I permessi assegnati sono identici per entrambi, ma il tipo di oggetto determina le operazioni possibili su ciascuno.

19. Cosa succederebbe a file2hard se aprissi un editor di testo e cambiassi il contenuto di file2new.txt?
file2hard è un **hard link** a file2.txt. Dopo la rinomina in file2new.txt, l'hard link resta valido perché punta allo stesso inode. Di conseguenza, modificando file2new.txt, anche file2hard mostrerà le stesse modifiche: entrambi condividono lo stesso contenuto fisico sul disco.

Riflessione:
I permessi e la proprietà dei file sono due degli aspetti più importanti in Linux. Sono una causa comune di problemi:

- Se i permessi sono troppo restrittivi, gli utenti o i programmi che dovrebbero accedere al file non riusciranno a farlo.
- Se i permessi sono troppo permissivi (ad esempio 777), il sistema diventa vulnerabile a modifiche non autorizzate.
- Errori nella proprietà o nei permessi possono impedire l'esecuzione di applicazioni o l'accesso a dati critici, causando interruzioni e malfunzionamenti.

Questo scenario con i link simbolici e hard, dimostra inoltre che non basta guardare al nome del file: ciò che conta è **dove punta il collegamento** o **quale inode viene condiviso**.