

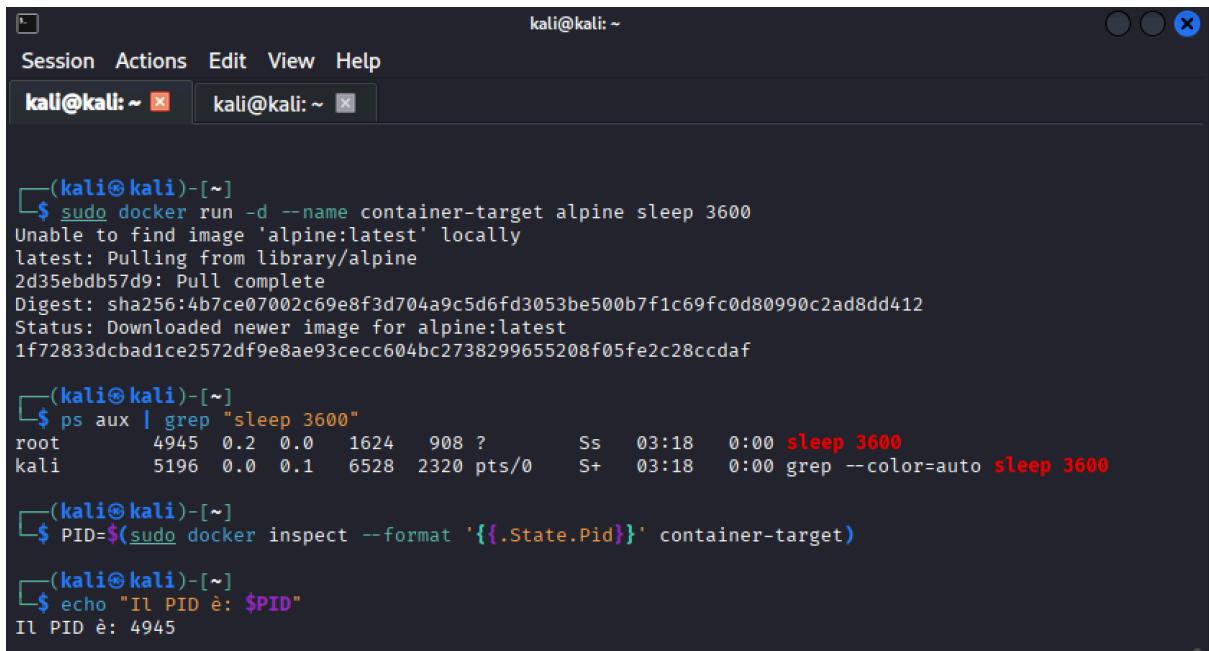
ESERCIZIO CONTAINER/NAMESPACES

Obiettivo: Lanciare un processo esterno condividendo i namespace di un container e verificarne la visibilità eseguendo “ps” al suo interno.

Di seguito riporto i passaggi svolti e l’analisi dei risultati ottenuti.

1. AVVIO CONTAINER E ANALISI PROCESSI SUL HOST

Ho avviato il container alpine con un processo “sleep 3600”. A differenza dell’esecuzione su Mac, eseguendo una “ps aux” direttamente sul terminale della macchina Kali, il processo del container è visibile. Questo conferma che su Linux i container sono semplici processi isolati e non macchine virtuali nascoste.



The screenshot shows a terminal window titled "kali@kali: ~". It has two tabs: "kali@kali: ~" and "kali@kali: ~". The terminal content is as follows:

```
(kali㉿kali)-[~]
$ sudo docker run -d --name container-target alpine sleep 3600
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
2d35ebdb57d9: Pull complete
Digest: sha256:4b7ce07002c69e8f3d704a9c5d6fd3053be500b7f1c69fc0d80990c2ad8dd412
Status: Downloaded newer image for alpine:latest
1f72833dcbadice2572df9e8ae93cecc604bc2738299655208f05fe2c28ccdaf

(kali㉿kali)-[~]
$ ps aux | grep "sleep 3600"
root      4945  0.2  0.0   1624   908 ?        Ss   03:18   0:00 sleep 3600
kali      5196  0.0  0.1   6528  2320 pts/0    S+   03:18   0:00 grep --color=auto sleep 3600

(kali㉿kali)-[~]
$ PID=$(sudo docker inspect --format '{{.State.Pid}}' container-target)

(kali㉿kali)-[~]
$ echo "Il PID è: $PID"
Il PID è: 4945
```

Risultato: il sistema operativo vede il processo PID 4945.

2. CONFERMA PID E INGRESSO NEL NAMESPACE

Ho utilizzato il comando “docker inspect” per confermare che il PID visto dall’host fosse corretto. Successivamente (vedere prossima slide), ho utilizzato “nsenter” direttamente dalla shell di Kali (stavolta senza utilizzare container privilegiati) per “entrare” nel namespace del processo.

3. VERIFICA ISOLAMENTO

Una volta lanciata la shell all'interno dei namespace agganciati (con -a prendo tutti i namespace), ho eseguito ps aux.

```
(kali㉿kali)-[~]
└─$ sudo nsenter -t 4945 -a sh
/ # ps aux
PID   USER      TIME  COMMAND
 1  root      0:00  sleep 3600
 8  root      0:00  sh
 9  root      0:00  ps aux
/ # sleep 1000 &
/ # █
```

Risultato: il processo che fuori era 4945, qui dentro appare come PID 1.

Questo perchè sarebbe il processo di init del container.

Il PID Namespace ha funzionato correttamente, fornendo al processo una vista virtualizzata degli ID.

4. ULTERIORE PROCESSO

Per completare l'esercizio, ho lanciato un processo “sleep 1000” dalla sessione nsenter (come si può vedere sopra). Poi, da un secondo terminale sul Host, ho interrogato il container con “docker exec”.

```
kali@kali: ~  ✘  kali@kali: ~  ✘
└─(kali㉿kali)-[~]
└─$ sudo docker exec container-target ps aux
[sudo] password for kali:
PID   USER      TIME  COMMAND
 1  root      0:00  sleep 3600
 8  root      0:00  sh
10  root      0:00  sleep 1000
11  root      0:00  ps aux
```

Risultato: Il comando “docker exec” mostra correttamente il processo “sleep 1000” con PID 10.

Conclusione: A differenza di quanto osservato su Mac, dove l'architettura richiede uno strato di virtualizzazione intermedio, su Linux è stato possibile osservare come i container siamo a tutti gli effetti processi del sistema operativo a cui vengono applicati dei filtri (Namespaces).

L'uso di nsenter mi ha fatto capire che le barriere di un container non sono impermeabili dato che, conoscendo il PID del processo su Host, è possibile "teletrasportare" un processo all'interno del contesto isolato.