

Luca Lussu

Tutorial e test di nftables per realizzare un firewall

Del corso del
Prof. Mauro Femminella

Perugia, Anno Accademico 2023/2024
Università degli Studi di Perugia
Corso di laurea magistrale in Ingegneria Informatica e Robotica
Curriculum Robotica
Dipartimento di Ingegneria



A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

0. Indice

1	Introduzione Netfilter e nftables	3
1.1	Netfilter e nftables	3
1.1.1	Introduzione filtraggio pacchetti	3
1.1.2	Installazione nftables	4
1.1.3	Sintassi nftables	6
2	Creazione di un ambiente virtuale con VirtualBox	13
2.1	Configurazione Rete - VirtualBox	13
3	Test Firewall con nftables	18
3.1	Test NAT	18
3.2	Test regole di filtraggio	20
3.3	Test Bilanciamento di carico e DDoS	26
3.4	Test nftables, ebtables e iptables - strato due	31
4	Migrazione da iptables a nftables	37
4.1	Migrazione da iptables a nftables	37
4.1.1	Esempio di traduzione di singoli comandi da iptables a nftables con iptables-translate	37
4.1.2	Esempio di come tradurre un file iptables-save in nftables con iptables-restore-translate	39

1. Introduzione Netfilter e nftables

1.1 Netfilter e nftables

Secondo il progetto **Netfilter**, **nftables** è un framework di classificazione dei pacchetti open source gratuito, rilasciato nel 2014 per Linux ed è incorporato nel kernel Linux.

La sua principale funzione è consentire agli amministratori di sistema di definire regole per il filtraggio dei pacchetti, decidendo quali pacchetti vengono accettati, respinti o instradati e la traduzione degli indirizzi di rete (NAT).

Nftables, fondamentalmente, è un sostituto e successore di *iptables* che è un programma di filtraggio dei pacchetti come nftables per Linux per definire le regole per filtrare e registrare l'attività del traffico di rete. Nftables sta guadagnando popolarità in quanto presenta alcuni vantaggi rispetto a iptables, tra cui una migliore e più facile scalabilità e prestazioni che lo rendono una scelta ideale per il filtraggio dei pacchetti nelle nostre reti.

La sua architettura modulare consente una gestione più snella e la sua sintassi uniforme semplifica la creazione e la manutenzione delle regole di filtraggio. Inoltre, Nftables è progettato per garantire prestazioni superiori rispetto ai suoi predecessori, offrendo una soluzione più efficiente ed evoluta per il filtraggio dei pacchetti.

1.1.1 Introduzione filtraggio pacchetti

Per comprendere al meglio quello che andremo a fare successivamente nelle configurazioni di un **Firewall**, è necessario conoscere il funzionamento del filtraggio dei

pacchetti.

In breve, ogni rete locale che si connette ad altre reti esterne richiede una sorveglianza attenta e un'esaminazione scrupolosa mediante l'utilizzo di un filtro dei pacchetti o di un nodo firewall affidabile, in modo da poter controllare il traffico di rete e proteggere la rete locale da attacchi esterni.

L'utilizzo di un firewall consente di sottoporre ogni pacchetto in transito, sia in entrata che in uscita, a un rigoroso controllo e monitoraggio, considerando parametri come porte, protocolli, origine e destinazione, che sono destinati a essere inviati o ricevuti da un nodo della rete.

In caso di rilevamento di discrepanze nei pacchetti di rete causate da attacchi dannosi, il nodo di filtraggio dei pacchetti può prendere decisioni sulla base di regole predefinite, consentendo o rifiutando i pacchetti per prevenire ulteriori attraversamenti della rete.

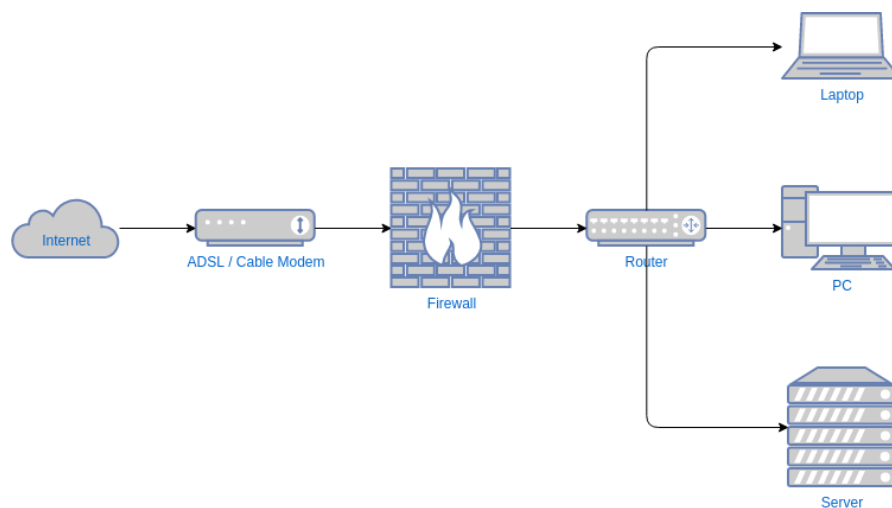


Figura 1.1:

1.1.2 Installazione nftables

Nel mio caso utilizzo una macchina virtuale con sistema operativo **Ubuntu 20.04.3 LTS** e prima di installare nftables è buona pratica aggiornare il sistema operativo con il

comando:

```
1 $ sudo apt update && sudo apt upgrade
```

Una volta aggiornato il sistema operativo, possiamo procedere con l'installazione di nftables con il comando:

```
1 $ sudo apt install nftables
```

Possiamo adesso avviare il service di nftables con il comando:

```
1 $ sudo systemctl start nftables
```

Puoi controllare lo stato di Nftables per assicurarti che sia in esecuzione correttamente:

```
1 $ sudo systemctl status nftables
```

Ed in fine siamo pronti a poter fare la configurazione del nostro firewall con Nftables. Ad esempio, puoi usare un editor di testo come nano:

```
1 $ sudo nano /etc/nftables.conf
```

Prima di addentrarci nella scrittura di regole per la creazione del firewall, è necessario conoscere la sintassi di nftables e le sue funzionalità.

1.1.3 Sintassi nftables

Nftables organizza le configurazioni in moduli gerarchici. I principali moduli sono:

- **Table:** è il modulo di livello superiore che contiene le regole di filtraggio dei pacchetti. Una tabella può contenere più catene.
- **Chain:** è un insieme di regole che vengono eseguite in sequenza. I tipi di catene più comuni sono *input*, *output* e *forward*, *prerouting* e *postrouting*.
- **Rule:** definisce un'azione specifica da applicare al traffico.

Tabelle

Possiamo aggiungere una tabella, una catena o una regola utilizzando il comando *nft add*, oppure possiamo scrivere direttamente le regole nel file di configurazione */etc/nftables.conf*.

```
table inet my_filter {  
    chain input {  
        type filter hook input priority 0; policy drop;  
        ct state established,related accept  
        iif "lo" accept  
        tcp dport {22, 80} accept  
        ip protocol icmp accept  
    }  
}
```

Figura 1.2: Table > Chain > Rule

Catene

Le catene sono sequenze ordinate di regole all'interno di una tabella. Ogni catena è associata a un tipo specifico di traffico o a uno stadio specifico del flusso di pacchetti. Ad esempio, una catena può essere progettata per gestire il traffico in arrivo *input*, il traffico in transito attraverso il sistema *forward*, o il traffico generato localmente *output*.

Le regole all'interno di una catena sono valutate in sequenza, e la prima regola che corrisponde al pacchetto determina l'azione da intraprendere. Queste azioni possono includere l'accettazione (accept), il rifiuto (drop), il rifiuto con un messaggio specifico (reject), o altre azioni personalizzate.

Le catene inoltre sono dotate di un *Type*, *Hook*, *Priority* e *Policy*.

Chain chain-name type <type> hook <hook> priority <priority> ; policy <policy> ;

- **Type:** definisce il tipo di catena. I tipi di catene più comuni sono *filter*, *nat* e *route*.
- **Hook:** definisce il punto in cui la catena viene collegata al flusso di pacchetti. I punti di aggancio sono associati a eventi chiave nel ciclo di vita dei pacchetti, i più comuni sono *prerouting*, *input*, *forward*, *output*, *postrouting*.
- **Priority:** definisce la priorità della catena.
- **Policy:** definisce l'azione predefinita da intraprendere se nessuna regola corrisponde al pacchetto. Le azioni più comuni sono *accept*, *drop*, *reject*.

I tipi di catena sono i seguenti:

- **filter:** è il tipo di catena predefinito. Viene utilizzato per filtrare il traffico di rete.
- **nat:** viene utilizzato per la traduzione degli indirizzi di rete (NAT).
- **route:** viene utilizzato per il routing di rete.
- **bridge:** viene utilizzato per il bridging di rete.
- **inet:** è un alias per le catene *ipv4* e *arp*.
- **ip:** è un alias per le catene *ipv4*.
- **ip6:** è un alias per le catene *ipv6*.
- **arp:** viene utilizzato per il traffico ARP.
- **ipv4:** viene utilizzato per il traffico IPv4.

Un **Hook** in una catena si riferisce a una fase specifica in cui un pacchetto viene elaborato attraverso un kernel Linux in base a regole definite.

I ganci (hooks) sono collegamenti tra le catene e gli eventi chiave che si verificano durante il percorso di un pacchetto attraverso il sistema, in breve sono fondamentali per determinare quando e dove verranno applicate le regole di filtraggio definite all'interno

delle catene.

Tipi principali di hook sono:

- **prerouting**: viene eseguito prima che il pacchetto venga instradato, viene attivato prima del routing.
- **input**: viene eseguito quando il pacchetto è destinato al sistema locale, Questo hook viene eseguito dopo la decisione di routing.
- **forward**: viene eseguito quando il pacchetto è destinato a un altro sistema. Questo hook si verifica anche dopo la decisione di routing, i pacchetti che non sono indirizzati alla macchina locale vengono elaborati da questo hook (eseguito per i pacchetti in transito attraverso il sistema).
- **output**: viene eseguito quando il pacchetto è generato dal sistema locale.
- **postrouting**: viene eseguito dopo che il pacchetto è stato instradato, questo hook viene utilizzato per i pacchetti che lasciano un sistema locale dopo la decisione di routing e quindi influenza il pacchetto anche dopo che è stato instradato.

Ogni pacchetto che attraversa il sistema (con nftables attivo) che sia in ingresso o in uscita, viene elaborato da una catena in base al suo hook, questo è consentito dal Kernel Linux che permette alle regole associate a questi hook d'interagire con il traffico di rete. Quando un pacchetto arriva alla macchina, prima attraversa l'hook di prerouting, dove potrebbe subire modifiche di routing. Successivamente, passa attraverso l'hook di input, dove vengono applicate le regole specifiche per il traffico in ingresso destinato alla macchina. Se il pacchetto è generato localmente, attraversa l'hook di output, dove potrebbero essere applicate ulteriori regole. Infine, l'hook di postrouting influenzerà il pacchetto dopo che è stato instradato.

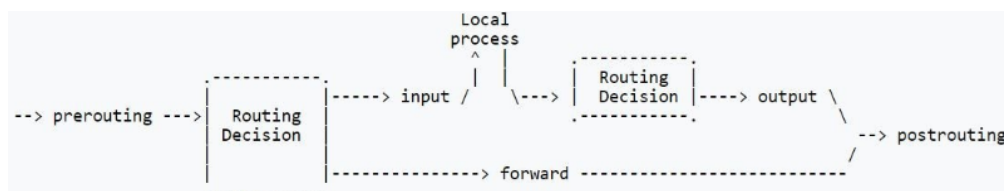


Figura 1.3:

Politiche

Le catene in Nftables devono definire politiche, come "*accept*" o "*drop*", che specificano se i pacchetti, in base alla loro posizione definita dagli hooks, devono essere accettati o rifiutati per impostazione predefinita. La politica di accettazione comporta l'accettazione predefinita di tutti i pacchetti, mentre la politica di eliminazione implica il rifiuto predefinito, e le regole specifiche all'interno della catena determinano il comportamento finale.

Un parametro critico è sicuramente il parametro **priority**, che influisce sull'ordine in cui vengono valutate le catene collegate a un hook specifico.

La priorità è basata sui punti seguenti:

- Le catene con una priorità inferiore vengono valutate prima di quelle con una priorità superiore, Una catena con priorità 0 sarà valutata prima di una con priorità 1 e così via.
- Le catene con la stessa priorità vengono valutate in base all'ordine in cui sono state aggiunte.
- Se la priorità non è esplicitamente specificata, viene assegnata una priorità predefinita. Le catene vengono valutate in base a questa priorità predefinita quando non è specificata una priorità numerica, quindi eseguite sempre dopo quelle definite.
- Le priorità possono anche essere assegnate in modo negativo. Una catena con priorità -1 sarà valutata prima di una con priorità 0.
- Le priorità sono specifiche per le singole tabelle e sono utilizzate per determinare l'ordine di valutazione delle catene collegate agli hook in quelle tabelle.
- Quando si hanno più tabelle, l'ordine di valutazione è determinato dalla priorità delle catene collegate agli hook corrispondenti.
- Tabelle con catene collegate ad un hook con priorità più bassa vengono valutate prima di quelle con priorità più alta.

Per spiegare meglio i concetti, possiamo fare degli esempi pratici.

Supponiamo di avere una tabella `my filter` così composta:

```
1  table inet my_filter {
2    chain input { type filter hook input priority 0; }
3    chain forward { type filter hook forward priority
4      -1; }
5    chain output { type filter hook output priority 5; }
6  }
```

Listing 1.1: la catena `forward` con priorità `-1` verrà valutata prima della catena `input` con priorità `0` e la catena `output` con priorità `5` verrà valutata dopo entrambe.

Supponiamo di avere due tabelle, ciascuna con una catena collegata all'hook di input.

```
1  table inet filter1 {
2    chain input { type filter hook input priority 0; }
3  }
4
5  table inet filter2 {
6    chain input { type filter hook input priority 1; }
7  }
```

Listing 1.2: Quando un pacchetto attraversa il sistema e raggiunge l'hook di input, valuta le catene in base alle priorità. In questo caso `filter1` verrà valutata prima

Se entrambe le tabelle avessero avuto la stessa priorità, l'ordine di valutazione sarebbe stato basato sulla posizione della dichiarazione delle tabelle nel file di configurazione.

Analizziamo la sintassi principale per inserire le regole da terminale, seguono i comandi:

- `sudo nft add table inet filter`: crea una nuova tabella chiamata `filter`, `inet` indica che la tabella è per il protocollo IPv4.

- *sudo nft add chain inet filter input type filter hook input priority 0* : questo comando aggiunge una catena di input alla tabella di filtraggio.
- *sudo nft add rule inet filter input ct state established,related accept:* aggiunge una regola alla catena di input, che accetta tutti i pacchetti che sono già stati accettati in precedenza.
- *sudo nft add rule inet filter input tcp dport 22 accept:* aggiunge una regola alla catena di input, che accetta tutti i pacchetti che hanno come porta di destinazione la porta 22.
- *sudo nft delete rule inet filter input tcp dport 22 accept:* elimina la regola aggiunta in precedenza.
- *sudo nft replace rule inet filter input tcp dport 22 drop:* aggiorna la regola aggiunta in precedenza, con una regola che scarta tutti i pacchetti che hanno come porta di destinazione la porta 22.
- *sudo nft delete chain inet filter input:* elimina la catena di input.
- *sudo nft delete table inet filter:* elimina la tabella di filtraggio.
- *sudo nft rename chain inet filter input newinput:* rinomina la catena di input in newinput.
- *sudo nft delete rule inet filter input position 1:* elimina la regola in posizione 1.
- *sudo nft flush chain inet filter input:* elimina tutte le regole nella catena di input.
- *sudo nft flush ruleset:* elimina tutte le regole.

Capire la sintassi di nftables risulta fondamentale non solo per implementare correttamente le regole di filtraggio, ma la scrittura di uno script per una configurazione con nftables può essere scritta e automatizzata tramite un motore decisionale esterno, come ad esempio un programma scritto in Python.

L'automazione è essenziale quando si desidera consentire a un motore decisionale esterno di prendere dinamicamente decisioni sulla configurazione del firewall,

l'automazione mediante script offre maggiore flessibilità e dinamicità nella gestione delle regole nftables, consentendo al sistema di adattarsi in tempo reale alle mutevoli esigenze di sicurezza e alle condizioni di rete. Inoltre, l'automazione consente di ridurre al minimo il rischio di errori umani, che possono portare a configurazioni errate e vulnerabilità di sicurezza.

Utilizzando script in altri linguaggi di programmazione, è possibile generare dinamicamente le regole in risposta a eventi o condizioni specifiche. Gli script possono essere eseguiti automaticamente in risposta a queste condizioni, ad esempio, ogni volta che una nuova connessione è rilevata o quando si verificano cambiamenti nelle metriche di sistema. L'approccio di aggiungere regole una ad una consente di adattarsi facilmente a cambiamenti dinamici nelle esigenze di sicurezza. Il motore decisionale può modificare le regole in tempo reale senza dover ricaricare l'intera configurazione del firewall.

Note queste informazioni preliminari, possiamo andare a implementare una rete con *VirtualBox* e andare a fare le configurazioni corrette per ogni VMs del caso.

2. Creazione di un ambiente virtuale con VirtualBox

2.1 Configurazione Rete - VirtualBox

Dobbiamo in primis realizzare un router che possa agire da firewall e che possa essere in grado di instradare il traffico tra le reti. Per fare ciò, creiamo una nuova macchina virtuale con VirtualBox, che chiameremo **VM Firewall**, con le seguenti caratteristiche: Quindi dopo aver importato un'immagine Ubuntu 20.04 LTS, andiamo a configurare la rete della macchina virtuale.

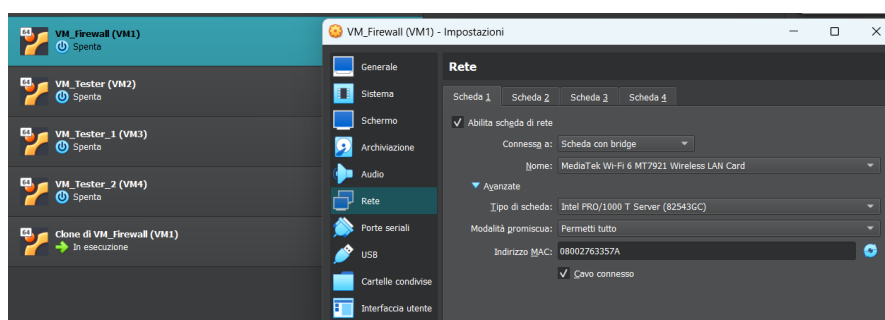


Figura 2.1:

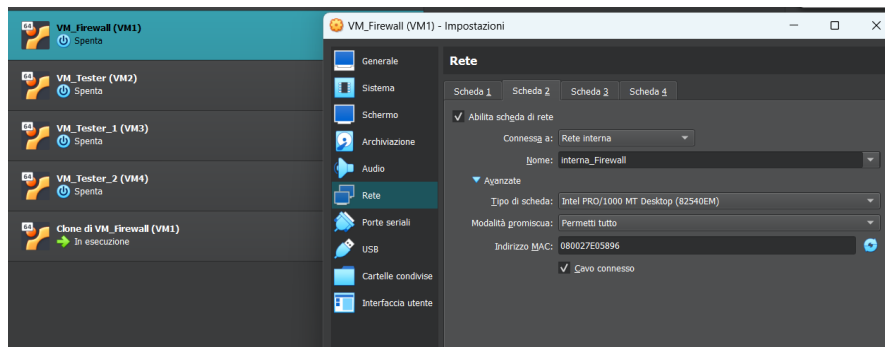


Figura 2.2:

In breve, ho creato due schede di rete per la macchina virtuale, una che sarà connessa alla rete interna (*interna Firewall*) e l'altra che sarà connessa alla rete esterna (*Scheda con bridge*).

Nella scheda di rete *interna Firewall* si andranno a connettere le altre VM per testare il firewall, quindi creare una rete locale con più VM connesse al Firewall.

Poi andiamo a configurare altre 3 macchine virtuali per i test del firewall e le chiameremo *VM Tester*, *VM Tester2* e *VM Tester3*, segue la configurazione della rete nel caso della VM Tester.

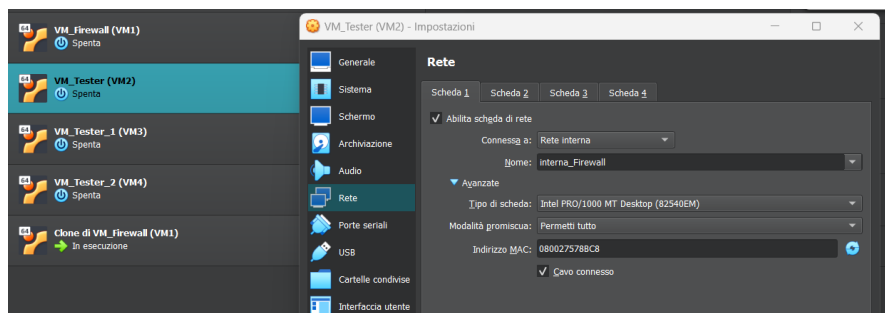


Figura 2.3:

Prossimo step è quello di configurare all'interno della VM Firewall la rete, assegnando l'indirizzo ip manualmente, quindi andiamo a modificare il file `/etc/netplan/00-installer-config.yaml` con il seguente codice:

```
1 network:
2   ethernet:
3     enp0s3:
4       dhcp4: true
5     enp0s8:
6       dhcp4: false
7       addresses: [10.0.0.1/24]
8     version: 2
```

Poi applichiamo le modifiche con il comando:

```
1 sudo netplan apply
```

Andiamo a verificare se le modifiche sono state applicate correttamente con il comando:

```
1 ip a
```

```
valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel sta
te UP group default qlen 1000
    link/ether 08:00:27:af:31:94 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global enp0s8
        valid_lft forever preferred_lft forever
```

Figura 2.4:

Configuriamo la rete di una delle VM Tester, in questo caso la VM Tester, assegnando l'indirizzo ip manualmente, andiamo a modificare il file */etc/netplan/00-installer-config.yaml* con il seguente codice:

```
1 network:
2   ethernet:
3     enp0s3:
```



```
4      dhcp4: false
5      addresses: [10.0.0.2/24]
6      nameservers:
7          addresses: [8.8.8.8]
8      routes:
9          - to: default
10            via: 10.0.0.1
11      version: 2
```

In sintesi questa configurazione Netplan assegna l'indirizzo ip all'interfaccia *enp0s3* della VM Tester, disabilitando DHCP e specificando un server DNS (Google DNS) e un gateway (il Firewall con ip 10.0.0.1).

Per essere sicuri che le modifiche siano state applicate correttamente, oltre a poterlo verificare con il comando "*ip a*" possiamo usare anche il seguente comando che permette di visualizzare la tabella di routing del sistema, in questo caso della VM Tester:

```
1 ip route
```

```
studente@ins:~$ ip route
default via 10.0.0.1 dev enp0s3 proto static
10.0.0.0/24 dev enp0s3 proto kernel scope link src 10.0.0.2
studente@ins:~$ █
```

Figura 2.5: Rotta predefinita della VM Tester verso il Firewall

Quest'ultima configurazione è stata applicata anche alle altre VM Tester, quindi ora abbiamo una rete locale con più VM connesse al Firewall.

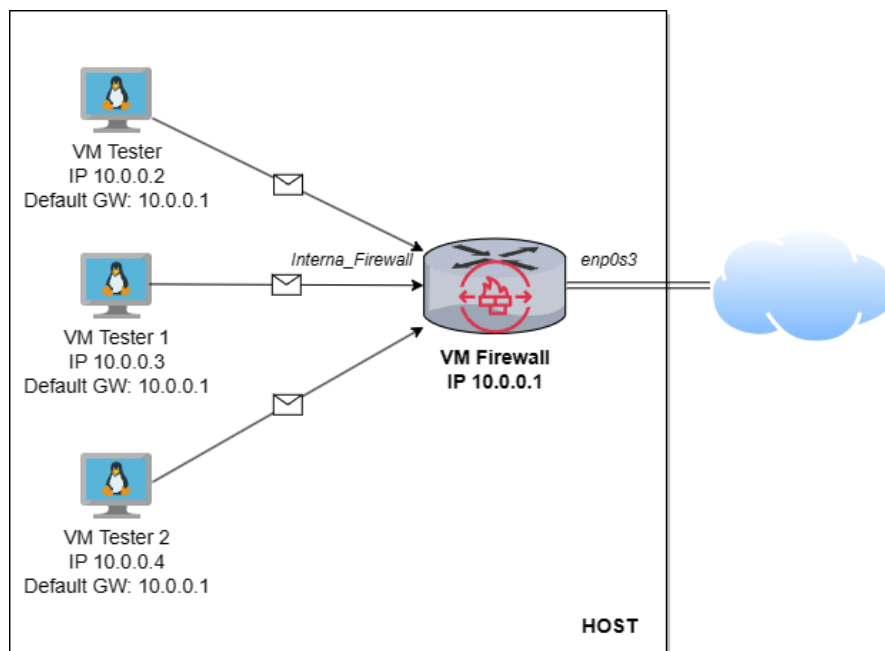


Figura 2.6: Schema Riassuntivo della rete

3. Test Firewall con nftables

3.1 Test NAT

Prima di iniziare con il primo test, servono alcune configurazioni di base per il firewall, come ad esempio abilitare il forwarding dei pacchetti, con il seguente comando:

```
1 sudo sysctl -w net.ipv4.ip_forward=1
```

Quando il forwarding è abilitato, il sistema può inoltrare il traffico di rete proveniente da una interfaccia di rete a un'altra. Questo passaggio è cruciale, in quanto essenzialmente stiamo dicendo al dispositivo di agire come un router, consentendo al dispositivo di inoltrare il traffico di rete locale a una rete esterna o viceversa.

Una volta abilitato il forwarding, possiamo iniziare a creare le regole di base per il firewall, come ad esempio la regola per il NAT (Network Address Translation), che consente a più dispositivi all'interno di una rete di condividere una singola connessione Internet. Questo viene fatto traducendo gli indirizzi IP interni dei dispositivi in un unico indirizzo IP pubblico sul router che si connette a Internet. Essa fornisce anche un livello di sicurezza aggiuntivo nascondendo la struttura interna della rete.

Andiamo a creare nella *VM Firewall* la regola per il NAT, con il seguente comando:

```
1 sudo nano /etc/nftables.conf
```

```
1 #!/usr/sbin/nft -f
2 flush ruleset
3
4 table inet nat {
5     chain postrouting {
6         type nat hook postrouting priority 0;
7         oifname "enp0s3" masquerade
8     }
9 }
```

Listing 3.1: Regola NAT con nftables

- `#!/usr/sbin/nft -f`, indica che il file è un file di configurazione nftables
- `flush ruleset`, cancella tutte le regole esistenti
- `table inet nat`, crea una nuova tabella chiamata `nat`, `inet` indica che la tabella è per il protocollo IPv4.
- `chain postrouting`, crea una nuova chain chiamata `postrouting`, che verrà eseguita dopo che il pacchetto è stato instradato.
- `type nat hook postrouting priority 0`, indica che la chain `postrouting` è una chain di tipo `nat`, la priorità 0 indica che questa catena ha la massima priorità.
- `oifname "enp0s3" masquerade`, indica che tutti i pacchetti in uscita dall'interfaccia `enp0s3` verranno mascherati.

Quindi essenzialmente, la catena `postrouting` è configurata per eseguire il mascheramento degli indirizzi IP in uscita sull'interfaccia di rete `enp0s3`.

Facciamo un test con le VM Tester, andando a pingare un indirizzo ip esterno, in questo caso *google.com*, e verifichiamo che il ping funzioni correttamente.

```
studente@ins:~$ ping google.com
PING google.com (142.250.180.142) 56(84) bytes of data.
64 bytes from mil04s43-in-f14.1e100.net (142.250.180.142): icmp_seq=1 ttl=116 time=21.8 ms
64 bytes from mil04s43-in-f14.1e100.net (142.250.180.142): icmp_seq=2 ttl=116 time=21.7 ms
```

Figura 3.1: Ping da VM Tester a google.com, ha avuto successo questo risultato indica una connessione funzionante

Il medesimo risultato anche per le altre VM Tester, quindi il NAT funziona correttamente.

3.2 Test regole di filtraggio

Lo script che andremo a vedere in questa sezione è stato realizzato per testare le regole di filtraggio del firewall, in particolare riguarda la gestione di connessione già stabilite, il filtraggio per indirizzi IP, gestione delle porte e controllo del traffico ICMP.

```
1      #!/usr/sbin/nft -f
2
3      flush ruleset
4      table inet filter {
5          chain input {
6              type filter hook input priority filter;
policy drop;
7              ct state established,related accept
8              ct state invalid drop
9              iif "lo" accept
10             ip saddr 10.0.0.3 drop
11             ip saddr 10.0.0.2 accept
12             tcp dport { 22, 80 } accept
13             ip protocol icmp accept
14             reject
15         }
```

```
16
17     chain forward {
18         type filter hook forward priority filter;
19     policy accept;
19     }
20
21     chain output {
22         type filter hook output priority filter;
23     policy accept;
23     }
24 }
```

Listing 3.2: Script per test regole di filtraggio

- flush ruleset, cancella tutte le regole esistenti
- table inet filter, crea una nuova tabella chiamata filter, inet indica che la tabella è per il protocollo IPv4.
- chain input, crea una nuova chain chiamata input, che verrà eseguita dopo che il pacchetto è stato ricevuto.
- type filter hook input priority filter; policy drop,, indica che la chain input è una chain di tipo filter, la priorità filter indica che questa catena ha la massima priorità, policy drop indica che tutti i pacchetti che non corrispondono a nessuna regola verranno scartati.
- ct state established, related accept, accetta tutti i pacchetti che sono già stati accettati in precedenza.
- ct state invalid drop, scarta tutti i pacchetti che non sono stati accettati in precedenza.
- iif "lo" accept, accetta tutti i pacchetti che arrivano dall'interfaccia di rete *lo*, che è l'interfaccia di rete di loopback.
- ip saddr 10.0.0.3 drop, scarta tutti i pacchetti che hanno come indirizzo sorgente.

- `ip saddr 10.0.0.2 accept`, accetta tutti i pacchetti che hanno come indirizzo sorgente.
- `tcp dport 22, 80 accept`, accetta tutti i pacchetti che hanno come porta di destinazione la porta 22 e 80, che sono rispettivamente le porte per il protocollo SSH e HTTP.
- `ip protocol icmp accept`, accetta tutti i pacchetti che hanno come protocollo ICMP.
- `reject`, scarta tutti i pacchetti che non corrispondono a nessuna regola.
- `chain forward`, crea una nuova chain chiamata `forward`, che verrà eseguita dopo che il pacchetto è stato instradato.
- `type filter hook forward priority filter; policy accept`;, indica che la chain `forward` è una chain di tipo `filter`, `policy accept` indica che tutti i pacchetti che non corrispondono a nessuna regola verranno accettati.
- `chain output`, crea una nuova chain chiamata `output`, che verrà eseguita dopo che il pacchetto è stato inviato.
- `type filter hook output priority filter; policy accept`;, indica che la chain `output` è una chain di tipo `filter`, `policy accept` indica che tutti i pacchetti che non corrispondono a nessuna regola verranno accettati.

Adesso possiamo applicare la seguente regola di filtraggio con il comando:

```
1 sudo nft -f /etc/nftables_filter.conf
```

Per verificare che la regola sia stata applicata correttamente, possiamo usare il comando:

```
1 sudo nft list ruleset
```

Verificato questo, passiamo ai seguenti test per la verifica delle regole di filtraggio.

In primis possiamo verificare che la porta SSH sia aperta, andando a connetterci con la VM `Tester2` alla VM `Firewall`, con il seguente comando:

```
ssh 10.0.0.1 -p 22
```

Listing 3.3: Connessione SSH da VM Tester2 a VM Firewall, connessione remota con l'indirizzo del VM Firewall sulla porta 22

```
studente@ins:~$ ssh 10.0.0.1 -p 22
studente@10.0.0.1's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-89-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Dec  1 06:14:27 PM UTC 2023

System load:  0.0615234375      Processes:            190
```

Figura 3.2: Connessione remota con l'indirizzo del VM Firewall sulla porta 22 avvenuta con successo

Ulteriore test SSH può essere fatta dalla VM Tester3, per verificare che la sua richiesta dev'essere rifiutata.

Per testare la porta HTTP, installiamo il servizio Apache2 sulla VM Firewall, e verifichiamo che il servizio sia attivo con il comando:

```
sudo systemctl status apache2
```

Dalla VM Tester testiamo con il comando:

```
telnet 10.0.0.1 80
```

Listing 3.4: Connessione HTTP sulla porta 80

Anche in questo caso la richiesta ha successo, quindi la porta HTTP è aperta per questo terminale.

Ulteriore conferma la possiamo ottenere anche con il comando:

```
curl http://10.0.0.1
```



```
studente@ins:~$ curl http://10.0.0.1
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <!--
    Modified from the Debian original for Ubuntu
    Last updated: 2022-03-22
    See: https://launchpad.net/bugs/1966004
  -->
</head>
```

Figura 3.3: Connessione HTTP sulla porta 80 avvenuta con successo

Ulteriore test può essere quello di fare *port scanning*, per verificare che le porte 22 e 80 siano aperte, per fare ciò possiamo usare il comando dalla VM Tester:

```
nmap 10.0.0.1
```

Questo comando ci permette di eseguire una scansione di porte sulla VM Firewall (indirizzo IP 10.0.0.1). Nmap è uno strumento di scansione di rete che può essere utilizzato per esplorare e rilevare le porte aperte su un altro sistema, eseguendo questo comando si otterrà un output che indica lo stato delle porte sulla VM Firewall.

```
studente@ins:~$ nmap 10.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-12-03 09:42 UTC
Nmap scan report for 10.0.0.1
Host is up (0.00072s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 10.21 seconds
```

Figura 3.4: Port scanning sulla VM Firewall, le porte 22 e 80 sono aperte

Grazie al port scanning possiamo verificare che le porte 22 e 80 sono aperte, gli amministratori di sistema impiegano spesso lo scanning delle porte per garantire una configurazione di sicurezza adeguata, individuare possibili punti deboli e proteggere il sistema da minacce esterne.

Proseguiamo con il test per il filtraggio degli indirizzi IP, in particolare andiamo a verificare che l'indirizzo IP 10.0.0.2 funzioni e blocchi l'indirizzo IP 10.0.0.3, per verificare

possiamo usare il comando ping dalla VM Tester2 e verificare nel coltempo che il firewall accetti protocolli ICMP, con il comando:

```
ping 10.0.0.2
```

Listing 3.5: ping da VM Tester a VM Firewall

Possiamo verificare cosa arriva al firewall con il comando:

```
sudo tcpdump -i enp0s8 icmp
```

Listing 3.6: Verifica pacchetti ICMP che arrivano al firewall

Tcpdump è uno strumento di analisi di rete utilizzato per catturare e analizzare il traffico di pacchetti in una rete, in questo caso stiamo catturando i pacchetti ICMP che arrivano sull'interfaccia di rete enp0s8, confermando il corretto funzionamento della richiesta fatta dalla VM Tester.

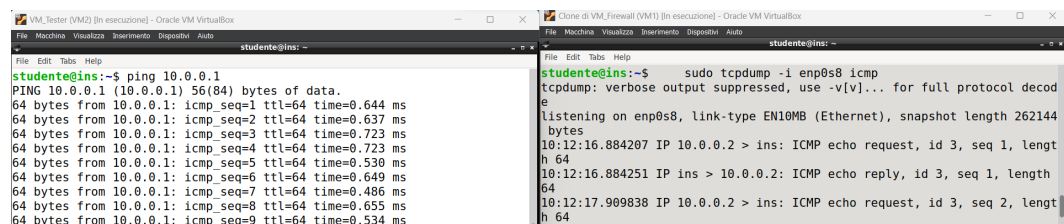


Figura 3.5: ping da VM Tester a VM Firewall

Eseguo i medesimi comandi però da VM Tester1, e verifico che il ping non funziona, in quanto l'indirizzo IP 10.0.0.3 risulta bloccato dal firewall, *ip saddr 10.0.0.3*. Ulteriore conferma la leggiamo dal comando tcpdump, che rileva le richieste ICMP ma non le accetta.

```
studente@ins:~$ sudo tcpdump -i enp0s8 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), snapshot length 262144
bytes
10:24:38.612175 IP 10.0.0.3 > ins: ICMP echo request, id 1, seq 1, length 64
10:24:39.627143 IP 10.0.0.3 > ins: ICMP echo request, id 1, seq 2, length 64
10:24:40.642441 IP 10.0.0.3 > ins: ICMP echo request, id 1, seq 3, length 64
10:24:41.666754 IP 10.0.0.3 > ins: ICMP echo request, id 1, seq 4, length 64
```

Figura 3.6:

3.3 Test Bilanciamento di carico e DDoS

Nella configurazione sottostante, configuro il firewall per fare *Load Balancing* base, in particolare il firewall deve dividere il traffico in modo che il traffico sia distribuito in modo uniforme tra le VM Tester.

La configurazione è la seguente:

```
1 #!/usr/sbin/nft -f
2
3 flush ruleset
4 table ip lb {
5     chain prerouting {
6         type nat hook prerouting priority filter;
7         policy accept;
8         dnat to numgen inc mod 3 map {
9             0 : 10.0.0.2,
10            1 : 10.0.0.3,
11            2 : 10.0.0.4
12        }
13    }
14 table inet my_filter {
15     chain input {
```

```
16      type filter hook input priority filter; policy
    accept;
17      ct state established,related accept
18      iif "lo" accept
19      ip protocol icmp accept
20      tcp flags & syn > 0x0 ct state new limit rate
    30/second accept
21  }
22
23  chain forward {
24      type filter hook forward priority filter;
    policy accept;
25  }
26
27  chain output {
28      type filter hook output priority filter; policy
    accept;
29  }
30  }
```

Listing 3.7: Load Balancing base

Poiché tutte le catene hanno la stessa priorità (filter), l'ordine di valutazione si basa sulla posizione in cui sono dichiarate nel file di configurazione. Quindi la catena *prerouting* nella tabella "ip lb" verrà valutata prima delle catene *input*, *forward*, e *output* nella tabella "my filter".

In questa configurazione viene creata una tabella *lb* per il bilanciamento di carico, e una tabella *my_filter* per il filtraggio, nella tabella *lb* viene definita una catena per il *prerouting*, che verrà eseguita prima che il pacchetto venga instradato. DNAT (Destination Network Address Translation) è un tipo di NAT che traduce l'indirizzo IP di destinazione di un pacchetto IP in un altro indirizzo IP, in questo caso il firewall traduce l'indirizzo IP di destinazione in un indirizzo IP di destinazione diverso, in modo che il pacchetto venga instradato verso un altro indirizzo IP.

- *dnat*: indica che il pacchetto deve essere tradotto
- *numgen*: indica che il pacchetto deve essere tradotto in base al numero generato, in questo caso il numero generato è un numero incrementale, che viene incrementato di 1 ogni volta che un pacchetto viene instradato.
- *mod 3*: indica che il numero generato deve essere diviso per 3.
- *map*: indica che il numero generato deve essere mappato, in questo caso il numero generato viene mappato con un indirizzo IP.
- *0 : 10.0.0.2* : indica che il numero generato 0 deve essere mappato con l'indirizzo IP 10.0.0.2.

Quindi, in base al risultato del calcolo del modulo aritmetico, i pacchetti verranno indirizzati a uno dei tre indirizzi IP di destinazione specificati nella mappatura in modo equo. Questo meccanismo implementa un tipo di load balancing, distribuendo equamente i pacchetti tra le destinazioni disponibili secondo il modulo specificato. Questa tecnica è spesso utilizzata per distribuire il traffico in modo equo tra più server, impedendo a un singolo server di essere sovraccaricato.

Nella tabella *my filter*, viene definita una catena per il filtraggio, con l'aggiunta di una regola che mi consenta di limitare il traffico in ingresso, in particolare la regola *tcp flags & syn > 0x0 ct state new limit rate 30/second accept* limita il traffico TCP in ingresso a 30 pacchetti al secondo.

Può essere utilizzata come misura di difesa contro attacchi di tipo DDoS, in particolare contro attacchi SYN flood, dove un aggressore invia un gran numero di richieste di connessione TCP (pacchetti SYN) al server al fine di saturarne le risorse e impedire la gestione di nuove connessioni legittime.

- *tcp flags & syn > 0x0*: indica che il pacchetto deve essere un pacchetto TCP con il flag SYN impostato.
- *ct state new*: indica che il pacchetto deve essere un nuovo pacchetto, assicura che la regola si applichi solo a pacchetti relativi a nuove connessioni, riducendo così l'impatto di un attacco SYN flood.

- *limit rate 30/second*: indica che il pacchetto deve essere limitato a 30 pacchetti al secondo.

Se il limite viene superato, i pacchetti eccedenti vengono scartati anziché accettati. In questo modo, la regola contribuisce a proteggere il sistema da un carico eccessivo di nuove connessioni SYN, riducendo l'efficacia di un attacco SYN flood.

Configurato la seguente regola, passiamo alla fase di test per verificare il corretto funzionamento del bilanciamento di carico e del filtraggio.

Per prima cosa, verifichiamo che il bilanciamento di carico funzioni correttamente, in particolare andiamo a verificare che il traffico venga distribuito in modo equo tra le VM Tester. Effettuo dalla varie VM Tester un ping verso l'indirizzo IP del Firewall, e nel firewall posso monitorare le richieste con il seguente comando:

```
sudo tcpdump -i enp0s8 dst host 10.0.0.2 or dst
host 10.0.0.3 or dst host 10.0.0.4
```

Listing 3.8: Verifica richieste ICMP che arrivano al firewall

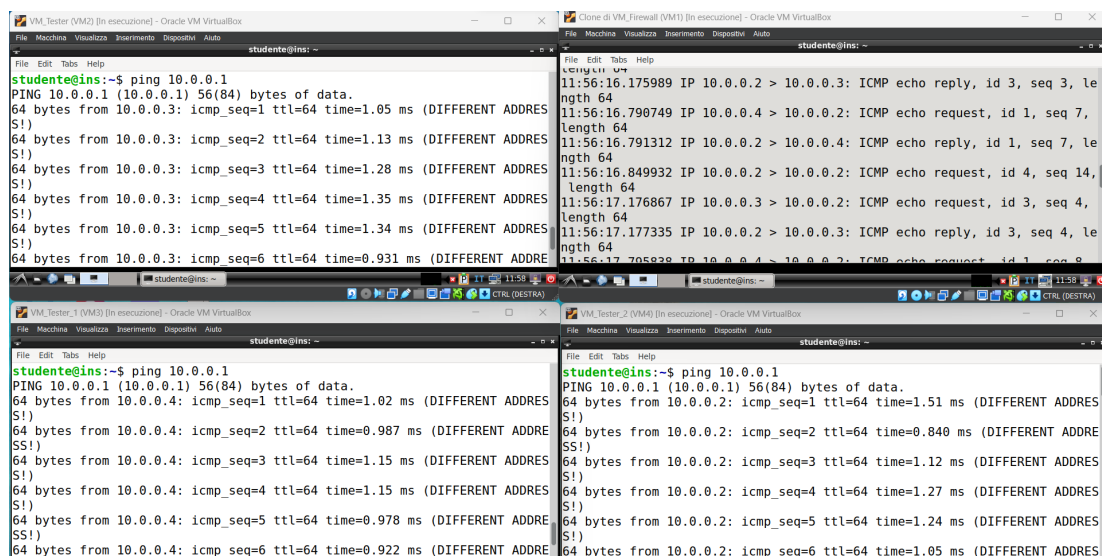


Figura 3.7: ping da VM Testers a VM Firewall per verificare il bilanciamento di carico

L'output indica che il funzionamento del traffico funziona come previsto, le richieste ICMP sono distribuite in modo equo tra le VM Testers, in base alle regole dn timer definite

nel firewall.

Quindi, quando una VM invia una richiesta ICMP a VM Firewall, il firewall decide a quale VM di destinazione instradare la richiesta in base alla sequenza numerica generata. Questo processo si ripete per ogni richiesta ICMP in arrivo, distribuendo così il traffico tra le VM Tester 1, VM Tester 2 e VM Tester 3 in modo equo, seguendo le regole definite nella configurazione di `nftables`.

Proseguiamo con il test per il filtraggio, in particolare andiamo a verificare che il firewall scarti i pacchetti che superano il limite di 30 pacchetti al secondo, mitigando di fatto un possibile attacco DDoS.

Per simulare un attacco DDoS possiamo usare il seguente comando dalla VM Tester:

```
1 sudo hping3 --fast --rand-source --flood 10.0.0.1
```

Listing 3.9: Simulazione attacco DDoS

Monitoro il Firewall con il comando:

```
1 sudo tcpdump -i enp0s8 -n -c 1000
```

Il comando `sudo hping3 -fast -rand-source -flood 10.0.0.1` simula un attacco DDoS, in particolare il comando invia un gran numero di pacchetti ICMP al firewall, utilizza indirizzi IP sorgente casuali per mascherare l'origine degli attacchi. Impiega il *flood mode*, il che significa che verranno inviati pacchetti il più velocemente possibile, senza attendere risposte, con l'obiettivo di saturare le risorse del destinatario.

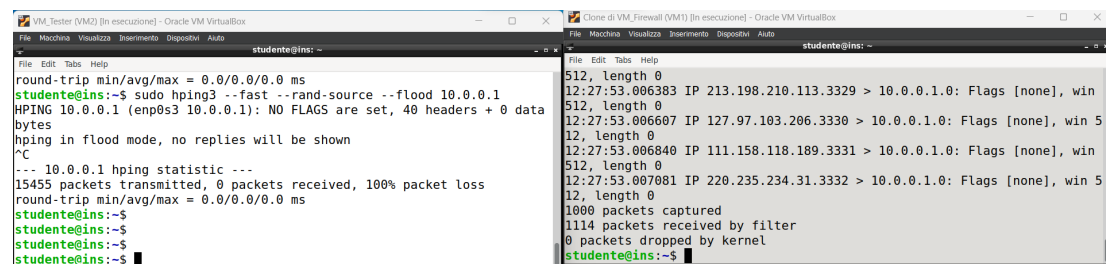


Figura 3.8: Simulazione attacco DDoS

Dal risultato ottenuto, possiamo verificare che il firewall ha scartato i pacchetti che superano il limite di 30 pacchetti al secondo, abbiamo 100% di pacchetti scartati (senza risposta), questo significa che il firewall ha mitigato l'attacco DDoS.

Il medesimo test è stato eseguito anche con le altre VM ottenendo il medesimo risultato.

3.4 Test nftables, ebtables e iptables - strato due

Nei casi precedenti abbiamo inserito le regole direttamente da File **.conf**, in questo caso andiamo a inserire le regole una a una da terminale e verifichiamo che il firewall funzioni correttamente.

Nel prossimo test di configurazione che seguirà andremo a utilizzare uno dei vantaggi principali di nftables, che unisce insieme le regole relative a IPv4 (iptables), IPv6 (ip6tables) e bridging (ebtables). Questo significa che con nftables, non è più necessario gestire set di regole separati per IPv4, IPv6 e bridging. Invece, è possibile gestire tutte queste regole in un unico posto, il che semplifica notevolmente la configurazione e la gestione del firewall.

Nello script seguente configureremo un firewall utilizzando nftables, ebtables, e iptables. Le regole specifiche consentono il traffico in ingresso solo dalla VM2 Tester con indirizzi IPv4 specifici, filtrano il traffico sulla catena del bridge basandosi sull'indirizzo MAC della VM Tester, e permettono il traffico SSH.


```
1 # Creazione della tabella nftables
2 nft add table inet filter
3 nft add chain inet filter input { type filter hook
   input priority 0 \; }
4
5 nft add rule inet filter input ip saddr 10.0.0.2 accept
6
7 # Creazione di una catena ebttables per il bridge
8 ebttables -t filter -N mybridge
9
10
11 ebttables -t filter -A INPUT -i enp0s3 -j mybridge
12 ebttables -t filter -A FORWARD -i enp0s3 -j mybridge
13
14 # Regola ebttables per permettere il traffico solo da
   VM2 (08:00:27:57:8b:c8)
15 ebttables -t filter -A mybridge -s 08:00:27:57:8b:c8 -j
   ACCEPT
16 ebttables -t filter -A mybridge -p ARP -j ACCEPT
17 ebttables -t filter -A mybridge -j DROP
18
19 # Regole iptables per il traffico IPv4
20 # Consenti SSH
21 iptables -A INPUT -i enp0s3 -p tcp --dport 22 -j ACCEPT
```

- *ebttables -t filter -N mybridge*: crea una nuova catena chiamata mybridge, ebttables è un firewall di livello 2, che opera a livello di bridge.
- *ebttables -t filter -A INPUT -i enp0s3 -j mybridge*: aggiunge una regola alla catena di input, che instrada i pacchetti in ingresso dall'interfaccia di rete enp0s3 alla catena mybridge.

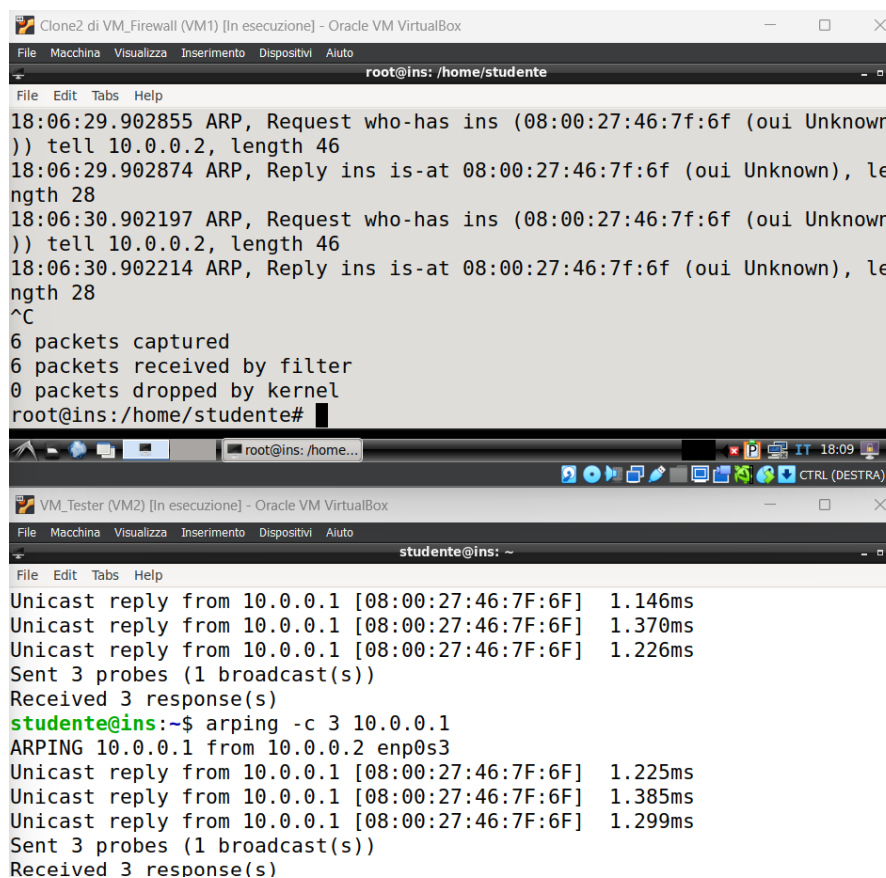
- *ebtables -t filter -A mybridge -s 08:00:27:57:8b:c8 -j ACCEPT*: aggiunge una regola alla catena mybridge, che accetta tutti i pacchetti che hanno come indirizzo MAC sorgente 08:00:27:57:8b:c8.
- *ebtables -t filter -A mybridge -p ARP -j ACCEPT*: aggiunge una regola alla catena mybridge, che accetta tutti i pacchetti che hanno come protocollo ARP.

Ecco il risultato ottenuto, dall'aggiunta di queste regole:

```
1      # nft list ruleset
2      table inet filter {
3  chain input {
4      type filter hook input priority filter; policy
      accept;
5      ip saddr 10.0.0.2 accept
6  }
7  }
8  table bridge filter {
9  chain mybridge {
10     ether saddr 08:00:27:57:8b:c8 counter packets 0
      bytes 0 accept
11     ether type arp counter packets 0 bytes 0 accept
12     counter packets 0 bytes 0 drop
13     counter packets 0 bytes 0 accept
14 }
15 chain INPUT {
16     type filter hook input priority filter; policy
      accept;
17     iifname "enp0s3" counter packets 0 bytes 0 jump
      mybridge
18 }
19 chain FORWARD {
20     type filter hook forward priority filter; policy
      accept;
21     iifname "enp0s3" counter packets 0 bytes 0 jump
      mybridge
22 }
23 }
```

```
26 table ip filter {
27     chain INPUT {
28         type filter hook input priority filter; policy
           accept;
29         iifname "enp0s3" meta l4proto tcp tcp dport 22
           counter packets 0 bytes 0 accept
30     }
31 }
```

Dai test che possiamo effettuare dalla VM tester con il comando *ssh 10.0.0.1* funziona correttamente. Passiamo al test ARP con il comando *arping -c 3 10.0.0.1* che ci permette di verificare che il firewall accetti il traffico ARP, in particolare il comando invia 3 pacchetti ARP alla VM Firewall, questo test serve a verificare la risoluzione degli indirizzi MAC in una rete locale.



```

root@ins: /home/studente
18:06:29.902855 ARP, Request who-has ins (08:00:27:46:7f:6f (oui Unknown)) tell 10.0.0.2, length 46
18:06:29.902874 ARP, Reply ins is-at 08:00:27:46:7f:6f (oui Unknown), length 28
18:06:30.902197 ARP, Request who-has ins (08:00:27:46:7f:6f (oui Unknown)) tell 10.0.0.2, length 46
18:06:30.902214 ARP, Reply ins is-at 08:00:27:46:7f:6f (oui Unknown), length 28
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@ins:/home/studente#

studente@ins: ~
Unicast reply from 10.0.0.1 [08:00:27:46:7F:6F] 1.146ms
Unicast reply from 10.0.0.1 [08:00:27:46:7F:6F] 1.370ms
Unicast reply from 10.0.0.1 [08:00:27:46:7F:6F] 1.226ms
Sent 3 probes (1 broadcast(s))
Received 3 response(s)
studente@ins:~$ arping -c 3 10.0.0.1
ARPING 10.0.0.1 from 10.0.0.2 enp0s3
Unicast reply from 10.0.0.1 [08:00:27:46:7F:6F] 1.225ms
Unicast reply from 10.0.0.1 [08:00:27:46:7F:6F] 1.385ms
Unicast reply from 10.0.0.1 [08:00:27:46:7F:6F] 1.299ms
Sent 3 probes (1 broadcast(s))
Received 3 response(s)

```

Figura 3.9: Test ARP funziona correttamente

```

studente@ins:~$ ping -I enp0s3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) from 10.0.0.2 enp0s3: 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.655 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.887 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.789 ms

```

Figura 3.10: Test traffico dall'indirizzo MAC 08:00:27:57:8b:c8 (VM Tester) funziona correttamente

4. Migrazione da iptables a nftables

4.1 Migrazione da iptables a nftables

Il team di Netfilter per facilitare la migrazione da **iptables** a **nftables** ha incluso nel pacchetto sorgente di iptables, un tool chiamato **iptables-translate** che permette di convertire le regole iptables in regole nftables.

Questo strumento semplifica il processo di migrazione tra i due framework di filtraggio dei pacchetti su Linux, semplificando il lavoro dell'amministratore di sistema.

I set di tool iptables-translate principali sono i seguenti:

- *iptables-translate*: converte le regole iptables in regole nftables
- *ip6tables-translate*: converte le regole ip6tables in regole nftables
- *iptables-restore-translate*: legge e converte in nftables le regole iptables da un file (iptables-save)
- *ip6tables-restore-translate*: legge e converte in nftables le regole ip6tables da un file (ip6tables-save)

4.1.1 Esempio di traduzione di singoli comandi da iptables a nftables con iptables-translate

Possiamo utilizzare lo strumento *iptables-translate* per convertire i comandi iptables in comandi nftables e tradurre la singola regola. Seguiranno alcuni esempi di sintassi per la traduzione di singoli comandi da iptables a nftables:

```
1      sudo iptables -translate -A INPUT -p tcp --dport 22
      -j ACCEPT
2      # nft add rule ip filter INPUT tcp dport 22 counter
      accept
```

Possiamo tradurre tutte le regole di una catena specifica utilizzando il seguente comando:

```
1      iptables -translate -L INPUT
```

Tradurre tutte le regole di una tabella specifica utilizzando il seguente comando:

```
1      iptables -translate -t filter -L
2      #dove filter e' il nome della tabella
```

```
1      iptables -translate -t filter -A INPUT -p tcp
      --dport 80 -j ACCEPT
2      # nft add rule ip filter INPUT tcp dport 80 counter
      accept
```

Essenzialmente il funzionamento è il medesimo, *iptables-translate* prende in input la sintassi originale di iptables e restituisce la sintassi nativa di nftables.

Tuttavia è importante notare che iptables-translate non è in grado di tradurre tutte le regole iptables in regole nftables, in particolare non è in grado di tradurre le regole iptables che utilizzano i moduli iptables non supportati da nftables. Alcune estensioni potrebbero non essere supportate o completamente supportate per vari motivi, ad esempio, potrebbero essere considerate obsolete o potrebbe non essere stato possibile lavorarci. Altri esempi:

```
1 iptables-translate -A INPUT -p icmp --icmp-type  
echo-request -j ACCEPT  
2 # nft add rule ip filter INPUT icmp type  
echo-request counter accept
```

```
1 iptables-translate -t nat -A POSTROUTING -s  
192.168.1.0/24 -o eth0 -j MASQUERADE  
2 # nft add rule ip nat POSTROUTING oifname "eth0" ip  
saddr 192.168.1.0/24 counter masquerade
```

```
1 iptables-translate -t nat -A PREROUTING -p udp  
--dport 500 -j DNAT --to-destination 192.168.1.4:4500  
2 # nft add rule ip nat PREROUTING udp dport 500  
counter dnat to 192.168.1.4:4500
```

4.1.2 Esempio di come tradurre un file iptables-save in nftables con iptables-restore-translate

Abbiamo a disposizione un altro tool chiamato *iptables-restore-translate* che ci permette di convertire un file iptables-save in regole nftables, ovvero ci permette di risparmiare parecchio tempo nel caso avessimo un set di regole già salvate in una configurazione, possiamo convertire un intero set di regole con pochi passaggi.

Per fare questo test, andiamo a scrivere una serie di regole per la nostra configurazione del firewall con iptables:


```
1
2 iptables -F
3 iptables -P INPUT DROP
4 iptables -P FORWARD DROP
5 iptables -P OUTPUT ACCEPT
6
7 # Consenti la connessione SSH solo dalla VM con IP
8   10.0.0.2
9 iptables -A INPUT -p tcp --dport 22 -s 10.0.0.2 -j
10   ACCEPT
11
12 iptables -A INPUT -p icmp -j ACCEPT
13
14 iptables -A INPUT -p tcp --dport 80 -j ACCEPT
15
16 # Consenti il forwarding di pacchetti
17 iptables -A FORWARD -j ACCEPT
18
19 # Blocca tutto il resto in ingresso
20 iptables -A INPUT -j DROP
21
22 # Abilita il mascheramento (NAT) per l'invio del
23   traffico sulla rete esterna
24 iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Il prossimo step è salvare le regole seguenti in file con il comando **iptables-save**:

```
1 iptables -save > filtro_iptables
```

Ora che abbiamo salvato le regole in un file, possiamo importarle e tradurle con il seguente comando **iptables-restore-translate**:

```
1 iptables-restore-translate -f filtro_iptables >
   filtro_nftables
```

con il comando nano possiamo leggere il file tradotto correttamente:

```
1 # Translated by iptables-restore-translate v1.8.7 on
   Tue Dec 5 12:06:0>
2 add table ip filter
3 add chain ip filter INPUT { type filter hook input
   priority 0; policy drop ;}
4 add chain ip filter FORWARD { type filter hook forward
   priority 0; policy drop ;}
5 add chain ip filter OUTPUT { type filter hook output
   priority 0; policy accept ;}
6 add rule ip filter INPUT ip saddr 10.0.0.2 tcp dport 22
   counter accept
7 add rule ip filter INPUT ip protocol icmp counter accept
8 add rule ip filter INPUT tcp dport 80 counter accept
9 add rule ip filter INPUT counter drop
10 add rule ip filter FORWARD counter accept
11 add table ip nat
12 add chain ip nat PREROUTING { type nat hook prerouting
   priority -100; policy accept ;}
13 add chain ip nat INPUT { type nat hook input priority
   100; policy accept ;}
14 add chain ip nat OUTPUT { type nat hook output priority
   -100; policy accept ;}
15 add chain ip nat POSTROUTING { type nat hook
   postrouting priority 100; policy accept ;}
16 add rule ip nat POSTROUTING oifname "enp0s3" counter
   masquerade
```

In questo caso specifico le regole corrispondono "quasi" come se avessimo scritto la sintassi manualmente in `nftables`, con l'eccezione dell'aggiunta di `nft` prima di `add`, in quanto in caso di scrittura di uno script `nftables` non è necessario esplicitare `nft add` prima di ogni regola.

Tuttavia, nel caso di comandi eseguiti direttamente da riga di comando, potrebbe essere necessario specificare `nft add` prima di ogni regola.

Notiamo anche l'aggiunta di `counter` che può essere omessa, scelta dell'amministratore in tal caso di ometterla o lasciarla, è utilizzata per attivare il conteggio del numero di pacchetti che corrispondono a una regola. Questo può essere utile per monitorare l'attività del firewall e valutare la quantità di traffico che attraversa determinate regole.

Ci sono ulteriori casi in cui la traduzione può diferire tra `iptables` e la scritta manuale di `nftables`:

Provando a tradurre i seguenti comandi notiamo delle piccole differenze:

```
1 iptables -A FORWARD -m conntrack --ctstate
  ESTABLISHED,RELATED -j ACCEPT
2 # add rule ip filter FORWARD ct state
  related,established counter accept
```

Listing 4.1: noto che l'ordine di "related" e "established" è stato invertito nella traduzione, gli ordine degli stati non influisce utilizzando `nftables`

In tutti i casi è sempre consigliabile conoscere entrambe le sintassi dei due framework, e delle funzionalità specifiche, adattare di conseguenza la scrittura delle regole scritte manualmente se necessario, dato che ci sono piccole differenze nella traduzione specialmente per estensioni più complesse.