



Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Informatica

# Elaborato di Network Security

Anno Accademico 2023/2024

Docente

**Prof. Simon Pietro Romano**

Studente

**Alberto Arola      M63001266**

# Sommario

|       |  |    |
|-------|--|----|
| 1     | Scenario.....                              | 1  |
| 1.1   | Creazione dei container e delle reti ..... | 4  |
| 1.1.1 | Server Web.....                            | 5  |
| 1.1.2 | Firewall .....                             | 5  |
| 1.1.3 | Client kali.....                           | 7  |
| 1.1.4 | Docker compose .....                       | 8  |
| 2     | Testing del firewall.....                  | 10 |
| 2.1   | SYN flood .....                            | 10 |
| 2.1.1 | Mitigazione SYN flood .....                | 12 |
| 2.2   | Slowloris .....                            | 18 |
| 2.2.1 | Mitigazione Slowloris .....                | 20 |
| 3     | Migliorie da poter applicare.....          | 23 |

# 1 Scenario

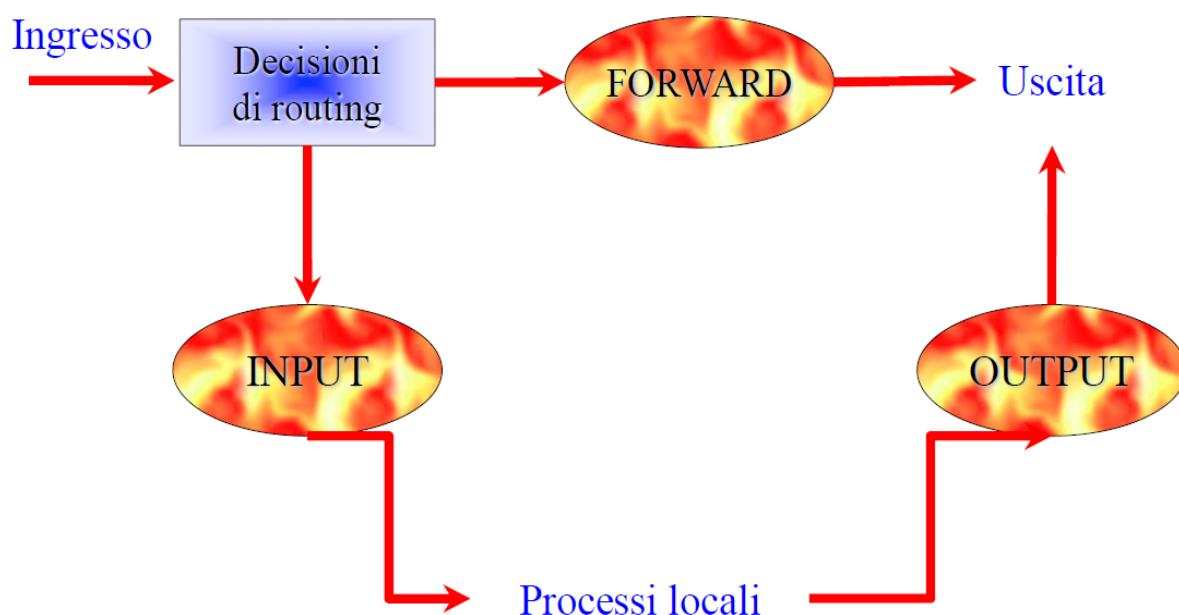
Il progetto è incentrato sulla realizzazione di un firewall linux tramite iptables. Iptables è un tool di configurazione del firewall che interagisce con il kernel Linux tramite le funzionalità di Netfilter. Netfilter è una infrastruttura del kernel Linux che consente il filtraggio dei pacchetti. Il filtraggio avviene analizzando l'header dei pacchetti e in base al contenuto decidere il destino di essi.

Iptables ha una struttura gerarchica per le regole che si creano; in particolare è organizzato in tabelle e catene. Le tabelle si occupano di un aspetto specifico del filtraggio, ad esempio:

- 'filter': è la tabella predefinita e viene utilizzata per il filtraggio del traffico. Le regole di questa tabella determinano se un pacchetto deve essere accettato, rifiutato o inoltrato
- 'nat': è la tabella responsabile della traduzione degli indirizzi di rete (NAT) e delle porte. Viene utilizzata per consentire a più host di condividere una singola connessione Internet e per eseguire il port forwarding
- 'mangle': è la tabella che consente di modificare specifiche caratteristiche dei pacchetti, come il TTL (Time to Live) o il Tipo di Servizio (ToS)
- 'raw': è la tabella utilizzata per configurare il firewall in modo che possa manipolare i pacchetti in una fase molto precoce del processo di gestione del pacchetto. Le regole definite nella tabella 'raw', quindi, vengono applicate prima che qualsiasi altra decisione di routing o filtraggio sia presa
- Tabelle personalizzate: l'utente ha la possibilità di creare tabelle personalizzate per scopi specifici

All'interno di ciascuna tabella, le regole sono organizzate in catene. Le catene sono sequenze di regole che definiscono il comportamento del traffico di rete in base a determinati criteri. Le principali catene sono:

- INPUT: catena che gestisce il traffico in ingresso destinato al sistema stesso. I pacchetti che attraversano questa interfaccia non hanno una interfaccia di output perciò qualsiasi regola contenente che usa '-o' in questa catena non troverà mai una corrispondenza
- OUTPUT: catena che gestisce il traffico in uscita generato dal sistema stesso la quale ha solo una interfaccia di output. Analogamente alla catena INPUT, i pacchetti che attraversano la catena non hanno un'interfaccia di input
- FORWARD: catena che gestisce il traffico inoltrato attraverso il sistema da un'interfaccia di rete all'altra. I pacchetti che l'attraversano hanno sia una interfaccia di input che di output
- Catene personalizzate: l'utente può creare catene personalizzate per organizzare le proprie regole di filtraggio in base alle esigenze specifiche



I pacchetti che arrivano ad una di queste catene vengono esaminati attraverso le regole in maniera sequenziale per controllare se i campi degli header di questi pacchetti soddisfano una certa regola. Se il pacchetto non soddisfa la prima regola della catena passa alla seconda poi alla terza e così via. Appena il pacchetto trova corrispondenza con una delle regole della catena viene applicata la policy scelta (ad esempio: DROP, ACCEPT...). Se, invece, il pacchetto non trova corrispondenza con nessuna delle regole viene applicata la policy di default della catena stessa. Quindi l'ordine con cui vengono inserite le regole è molto importante.

Per poter creare il firewall con le regole e politiche di sicurezza si è deciso grazie ad esso di proteggere un web server con attivi HTTP e HTTPS. Tramite Docker, quindi, si è creato un container con un web server Apache con esposti HTTP/HTTPS all'interno di una sottorete docker, un container con Linux nel quale utilizzare iptables per creare delle regole e messo a protezione della sottorete del web server e agendo anche da gateway di default per la sottorete

stessa e infine un container con Kali per poter testare il firewall messo su di una sottorete differente da quella del server e avente sempre il firewall come gateway di default.



Per questo scenario è stata utilizzata in particolare la tabella ‘filter’ di iptables e poiché il firewall dovrà agire da gateway per le due sottoreti create, è stata utilizzata la catena FORWARD in quanto i pacchetti, una volta analizzati dalle regole, dovranno essere opportunamente inoltrati per poter far comunicare il client kali e il server web.

## 1.1 Creazione dei container e delle reti

Tramite Docker, quindi, sono stati creati i vari container e le sottoreti per poter esporre un server web protetto da un firewall e testando il firewall stesso tramite un client esterno.

### 1.1.1 Server Web

```
2 FROM ubuntu:latest
3
4 # Installa Apache e SSL
5 RUN apt-get update \
6     && apt-get install -y apache2 \
7     && apt-get clean \
8     && rm -rf /var/lib/apt/lists/*
9
10 # Installa strumenti per la gestione di rete
11 RUN apt-get update \
12     && apt-get install -y iproute2 tcpdump iputils-ping \
13     && apt-get clean \
14     && rm -rf /var/lib/apt/lists/*
15
16 # Abilita i moduli Apache necessari
17 RUN a2enmod ssl
18 RUN a2enmod rewrite
19 RUN a2enmod headers
20
21 # Copia i file di configurazione Apache
22 COPY apache-config.conf /etc/apache2/sites-available/000-default.conf
23
24 # Genera una chiave privata e un certificato autofirmato per SSL
25 RUN openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt -subj "/C=US/ST=State/L=City/O=Organization/CN=localhost"
26
27 # Abilita il sito HTTPS
28 RUN a2ensite default-ssl
29
30 # Espone le porte 80 e 443
31 EXPOSE 80
32 EXPOSE 443
33
34 # Avvia Apache in primo piano
35 CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Il dockerfile in figura è utilizzato per creare l'immagine di server web Apache con esposte le porte 80 e 443 rispettivamente per poter permettere connessioni HTTP e HTTPS. Per poter esporre HTTPS è stato necessario creare un certificato auto firmato.

### 1.1.2 Firewall

```
1 FROM ubuntu:latest
2
3 # Installa iptables, tcpdump e iproute2
4 RUN apt-get update \
5     && apt-get install -y iptables tcpdump iproute2 \
6     && apt-get clean \
7     && rm -rf /var/lib/apt/lists/*
8
9 # Copia lo script di configurazione del firewall
10 COPY firewall-config.sh /usr/local/bin/firewall-config.sh
11 RUN chmod +x /usr/local/bin/firewall-config.sh
12
13 # Avvia lo script di configurazione del firewall e lasciare il container in esecuzione
14 CMD ["/bin/bash", "-c", "/usr/local/bin/firewall-config.sh && tail -f /dev/null"]
15
```

Il dockerfile per il firewall permette di utilizzare la versione più recente di una immagine Ubuntu sulla quale installare vari tool di rete tra cui in particolare iptables per la creazione del firewall. Per rendere il tutto più organizzato, le regole del firewall sono state scritte in uno script che all'avvio del container verrà eseguito in modo tale da applicare le regole.

```
2
3 # Pulisce tutte le regole iptables esistenti
4 iptables -F
5 iptables -X
6
7 # Politica predefinita: blocca tutto il traffico in ingresso e permette tutto il traffico in uscita
8 iptables -P INPUT DROP
9 iptables -P FORWARD DROP
10 iptables -P OUTPUT DROP
11
12 # Abilita il forwarding del traffico IPv4
13 echo 1 > /proc/sys/net/ipv4/ip_forward
14
15 # Consentire il forwarding del traffico HTTP e HTTPS tra il server web e il mondo esterno
16 # Inoltre il server potrà solo rispondere e non iniziare una nuova connessione
17 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
18
19 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT
20
21 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
22
23 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 443 -m state --state RELATED,ESTABLISHED -j ACCEPT
24
25
26
```

Lo script iniziale per la creazione delle regole del firewall prevede di eliminare tutte le regole già esistenti, impostare DROP policy di default per tutte le catene in modo tale che sia permesso solo traffico definito dalle regole create nelle catene e scartare tutto il traffico che non trova corrispondenza con alcuna regola. È stato, inoltre, abilitato il forwarding così da permettere al firewall di agire da gateway per le due sottoreti docker create: *server\_network* ed *external\_network*. In particolare le prime due regole della catena FORWARD permettono il traffico HTTP: vengono accettati i pacchetti ricevuti dal firewall sull'interfaccia eth0 (quindi quella della *external\_network*), con protocollo TCP per il livello trasporto, IP destinazione quello del web server sulla porta 80; si permette la risposta del server al client quindi vengono accettati i pacchetti in ingresso al firewall dall'interfaccia eth1 (dalla *server\_network*), con indirizzo IP



sorgente del server web, porta sorgente 80 e protocollo trasporto TCP. In particolare nella regola che permette al server web di rispondere si è specificato che solo il traffico di tipo RELATED (pacchetti che sono correlati a una connessione già esistente) ed ESTABLISHED (pacchetti associati a connessioni che sono già state stabilite e che sono in uno stato di comunicazione attiva) possa essere accettato in modo tale che il server non possa iniziare egli stesso una connessione con il mondo esterno.

Il tutto vale anche per il traffico HTTPS con la differenza che la porta invece di essere quella 80 è la 443.

### 1.1.3 Client kali

```
1 # Immagine di Kali Linux ufficiale
2 FROM kalilinux/kali-rolling
3
4 # Aggiorna i repository e installa gli strumenti
5 RUN apt-get update && \
6     apt-get install -y \
7         nmap \
8         tcpdump \
9         iproute2 \
10        iputils-ping \
11        curl \
12        hping3 \
13        metasploit-framework \
14        && apt-get clean
15
16 # Imposta l'ambiente di lavoro predefinito
17 WORKDIR /root
18
19 # Mantenere il container in esecuzione
20 CMD ["tail", "-f", "/dev/null"]
21|
```

Tramite questo dockerfile si è creata una immagine kali per il container del client con alcuni tool di rete. I tool di rete più importanti sono metasploit che è un framework open-source ampiamente utilizzato per testare la sicurezza dei sistemi informatici il quale fornisce una vasta gamma di funzionalità per eseguire test di penetrazione e analizzare la sicurezza dei sistemi; iproute2 con il quale poter impostare l'interfaccia del firewall sulla external\_network come gateway di default.

### 1.1.4 Docker compose

```
1 version: '3'
2
3 services:
4   server:
5     image: server_image
6     networks:
7       server_network:
8         ipv4_address: 172.20.0.3
9     container_name: server_container
10    stdin_open: true
11    cap_add:
12      - ALL
13
14   kali:
15     image: kali_image
16     networks:
17       external_network:
18         ipv4_address: 172.21.0.3
19     container_name: kali_container
20    stdin_open: true
21    cap_add:
22      - ALL
23
24   firewall:
25     image: firewall_image
26     networks:
27       server_network:
28         ipv4_address: 172.20.0.2
29       external_network:
30         ipv4_address: 172.21.0.2
31     container_name: firewall_container
32    stdin_open: true
33    cap_add:
34      - ALL
```

Tramite il file docker-compose.yml si andranno a creare i container a partire dalle immagini create con i precedenti dockerfile e le due sottoreti. In particolare si assegnano i nomi ai container e gli indirizzi IP ad essi con il firewall avente due indirizzi in quanto dovrà trovarsi su entrambe le sottoreti create.

```
35
36 networks:|
37   server_network:
38     driver: bridge
39     ipam:
40       config:
41         - subnet: 172.20.0.0/24
42
43   external_network:
44     driver: bridge
45     ipam:
46       config:
47         - subnet: 172.21.0.0/24
48
```

Dopo aver creato e attivato i container si va ad impostare nel server e nel client come default gateway gli indirizzi IP del firewall che si troveranno sulle loro sottoreti tramite il tool iproute2.

## 2 Testing del firewall

Innanzitutto è stato provato se il firewall permette la comunicazione tra il client kali e il server web tramite HTTP e HTTPS. In particolare sul terminale del container del client si è utilizzato il comando `curl http://172.20.0.3` per provare una connessione HTTP con il server il quale risponde correttamente inviando al client la pagina di default del server Apache. È stato fatto lo stesso con la connessione HTTPS utilizzando in questo caso il comando `curl -k https://172.20.0.3`; l'opzione `-k` dice a curl di effettuare connessioni e trasferimenti SSL non sicuri in quanto i certificati creati per il server web sono self signed e quindi non troverò vedrò la risposta del server ma vedrò l'errore di certificato non attendibile.

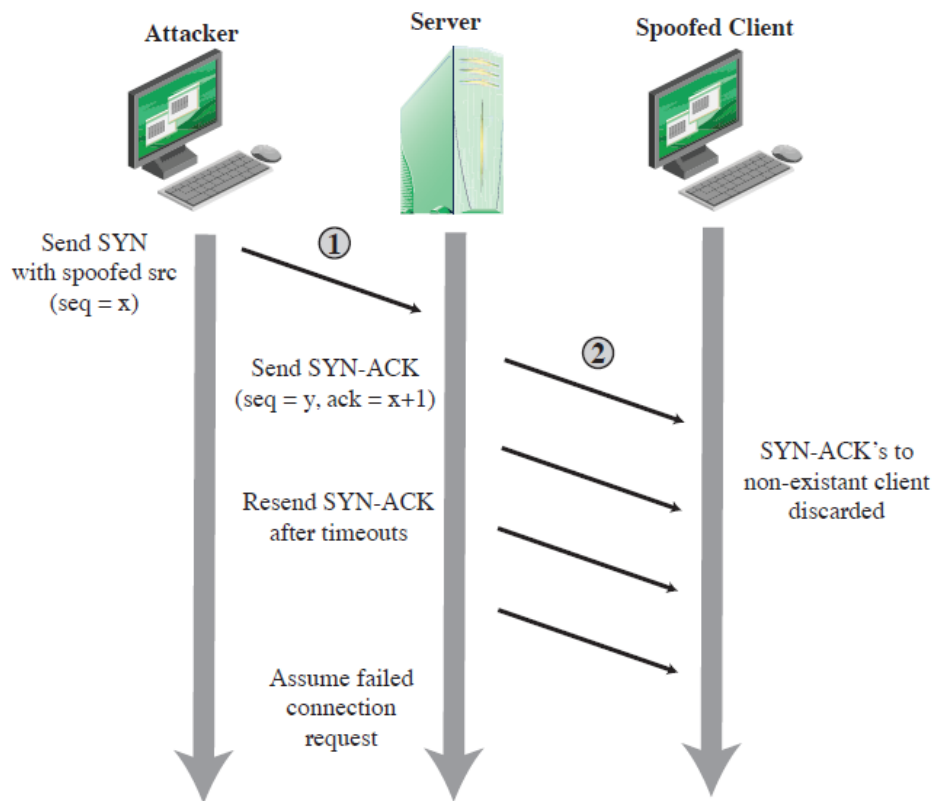
Appurato che il client riesce a connettersi al server e a ricevere in risposta la pagina di default del server Apache sono stati provati gli attacchi Dos SYN flood e Slowloris tramite il client kali verso il server e successivamente sono state create delle regole ad hoc per contrastare gli stessi.

### 2.1 SYN flood

Utilizzando il modulo `auxiliary/dos/tcp/synflood` di metasploit sul container kali si è provato ad inondare il server di pacchetti TCP con il flag SYN alzato e utilizzando come indirizzo IP sorgente un indirizzo facente sempre parte della sottorete del client kali ma differente da quello assegnato.

Il SYN flood cerca di sovraccaricare un server con un alto volume di richieste di connessione. Durante questo attacco, l'attaccante invia un gran numero di pacchetti SYN con indirizzo IP sorgente spoofed al server di destinazione, ma non completa mai la procedura di handshake TCP, lasciando le connessioni in

uno stato di "half-open". Questo consumo di risorse del server può rendere il sistema inaccessibile agli utenti legittimi, impedendo loro di stabilire connessioni valide.



In particolare mentre con il client kali si utilizzava il modulo di metasploit per inondare il server di pacchetti con docker stats si è monitorato le risorse del container del server notando un significativo aumento del consumo della CPU.

```
kali@kali: ~/Desktop/ns_project
File Actions Edit View Help
CONTAINER ID   NAME      CPU %     MEM %     NET I/O    BLOCK I/O  PIDS
d89ac70f2f2e   server_container  19.01%    51.01%    23.9GB / 27.6MB  12.7MB / 4.1kB  58

root@725b9a6f6b16: ~
File Actions Edit View Help
Name      Current Setting  Required  Description
---
INTERFACE  eth0             no        The name of the interface
NUM        no              no        Number of SYNs to send (else unlimited)
RHOSTS     172.20.0.3      yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit.html
RPORT      80              yes       The target port
SHOST      172.21.0.25     no        The spoofable source address (else randomizes)
SNAPLEN    65535           yes       The number of bytes to capture
SPORT      no              no        The source port (else randomizes)
TIMEOUT    500             yes       The number of seconds to wait for new data

View the full module info with the info, or info -d command.

msf6 auxiliary(dos/tcp/synflood) > run
[*] Running module against 172.20.0.3

[*] SYN flooding 172.20.0.3:80 ...
```

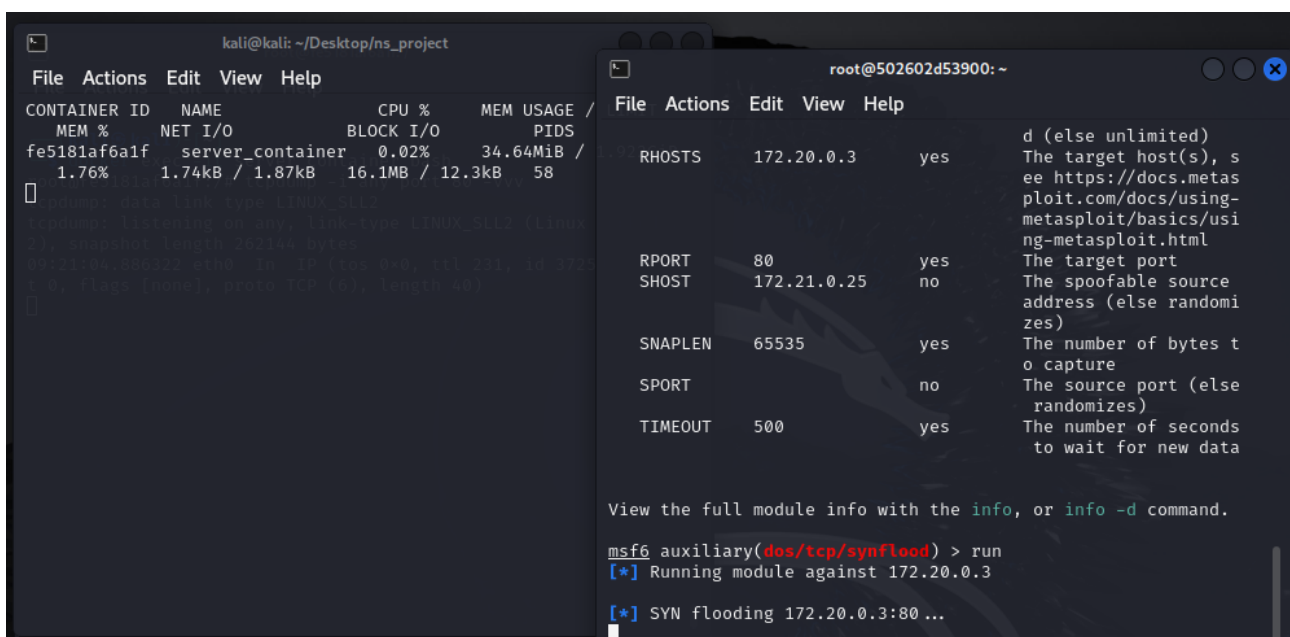
### 2.1.1 Mitigazione SYN flood

Per mitigare questo attacco si è modificata la regola che permette ad un pacchetto proveniente dalla external\_network di poter iniziare una connessione TCP con il server sul porto 80 specificando un rate massimo di 10 pacchetti al secondo. In questo modo se il tasso di arrivo di pacchetti con solo il flag SYN alzato è superiore a 10 pacchetti al secondo, quelli in eccesso verranno scartati.

```
3 # Pulisce tutte le regole iptables esistenti
4 iptables -F
5 iptables -X
6
7 # Politica predefinita: blocca tutto il traffico in ingresso e permette tutto il traffico in uscita
8 iptables -P INPUT DROP
9 iptables -P FORWARD DROP
10 iptables -P OUTPUT DROP
11
12 # Abilita il forwarding del traffico IPv4
13 echo 1 > /proc/sys/net/ipv4/ip_forward
14
15 # Consentire il forwarding del traffico HTTP e HTTPS tra il server web e il mondo esterno
16 # Inoltre il server potrà solo rispondere e non iniziare una nuova connessione
17 iptables -A FORWARD -i eth0 -p --syn -m limit --limit 10/s tcp -d 172.20.0.3 --dport 80 -m state --state NEW -j ACCEPT
18
19 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 80 -m state --state ESTABLISHED -j ACCEPT
20
21 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT
22
23 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
24
25 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 443 -m state --state RELATED,ESTABLISHED -j ACCEPT
26
```

In particolare la prima regola è stata divisa in due in cui la prima come stato della connessione ha solo NEW mentre la seconda solo ESTABLISHED. In questo modo, essendo che con il SYN flood si cerca di iniziare l'handshake TCP senza concluderlo, il filtro sul tasso di arrivo sarà applicato solo ai pacchetti che vorranno iniziare una nuova connessione e che hanno il solo flag SYN alto (utilizzando l'opzione `-syn`).

Dopo aver aggiunto la regole e averla fatta applicare dal firewall si è riprovato ad utilizzare il modulo di metasploit notando che il consumo di CPU da parte del container con il server è stato notevolmente abbattuto.



The image shows two terminal windows. The left window, titled 'kali@kali: ~/Desktop/ns\_project', displays container statistics for 'server\_container' (ID: fe5181af6a1f). The right window, titled 'root@502602d53900: ~', shows the Metasploit (msf6) interface running the 'auxiliary(dos/tcp/synflood)' module against the target 172.20.0.3.

| CONTAINER ID | NAME             | CPU % | MEM USAGE /                        |
|--------------|------------------|-------|------------------------------------|
| fe5181af6a1f | server_container | 0.02% | 34.64MiB /                         |
|              |                  | 1.76% | 1.74kB / 1.87kB 16.1MB / 12.3kB 58 |

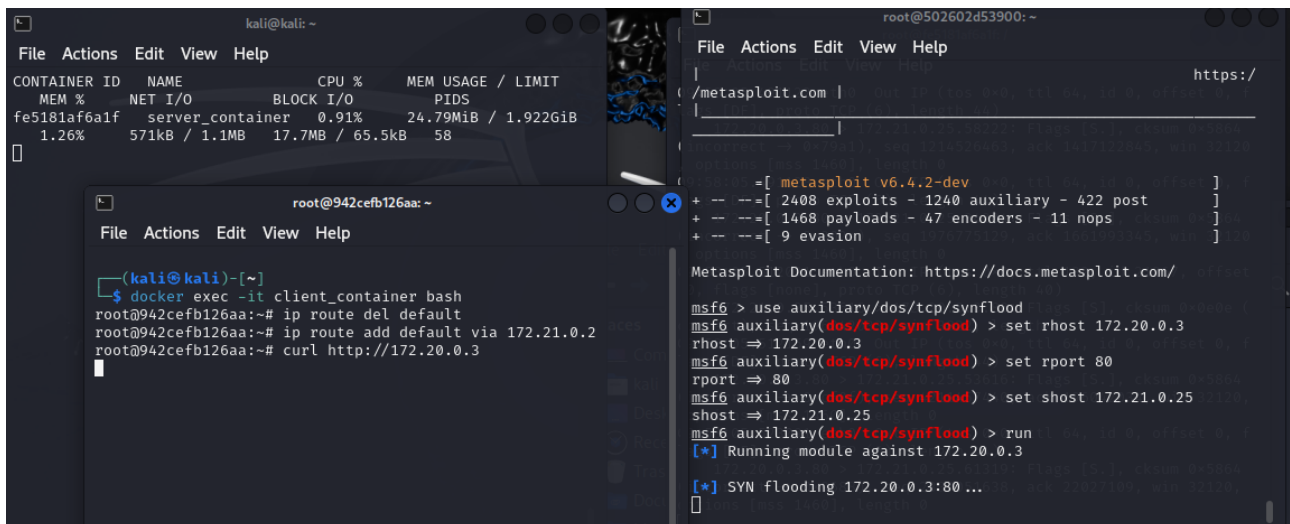
  

| File    | Actions     | Edit | View | Help  |
|---------|-------------|------|------|---|
| RHOSTS  | 172.20.0.3  | yes  |      | d (else unlimited)<br>The target host(s), see <a href="https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html">https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html</a> |
| RPORT   | 80          | yes  |      | The target port   |
| SHOST   | 172.21.0.25 | no   |      | The spoofable source address (else randomizes)  |
| SNAPLEN | 65535       | yes  |      | The number of bytes to capture  |
| SPOOF   |             | no   |      | The source port (else randomizes)   |
| TIMEOUT | 500         | yes  |      | The number of seconds to wait for new data  |

View the full module info with the `info`, or `info -d` command.

```
msf6 auxiliary(dos/tcp/synflood) > run
[*] Running module against 172.20.0.3
[*] SYN flooding 172.20.0.3:80 ...
```

Dopo aver modificato le regole e aver appurato che esse funzionino alla rete `external_network` si è aggiunto un nuovo client in modo da poter verificare che il firewall permetta la comunicazione dello stesso con il server mentre il client kali performa il SYN flood.



Come si nota in figura mentre il client malevolo performa l'attacco, quello legittimo non riesce a connettersi, questo perché quello malevolo supera il tasso imposto e qualsiasi altro utente sulla external\_network non potrà iniziare una comunicazione con il server.

Sono state quindi modificate di nuovo le regole per poter discriminare quale host supera il tasso imposto in modo tale che un altro host che non superi lo stesso possa comunque comunicare con il server.

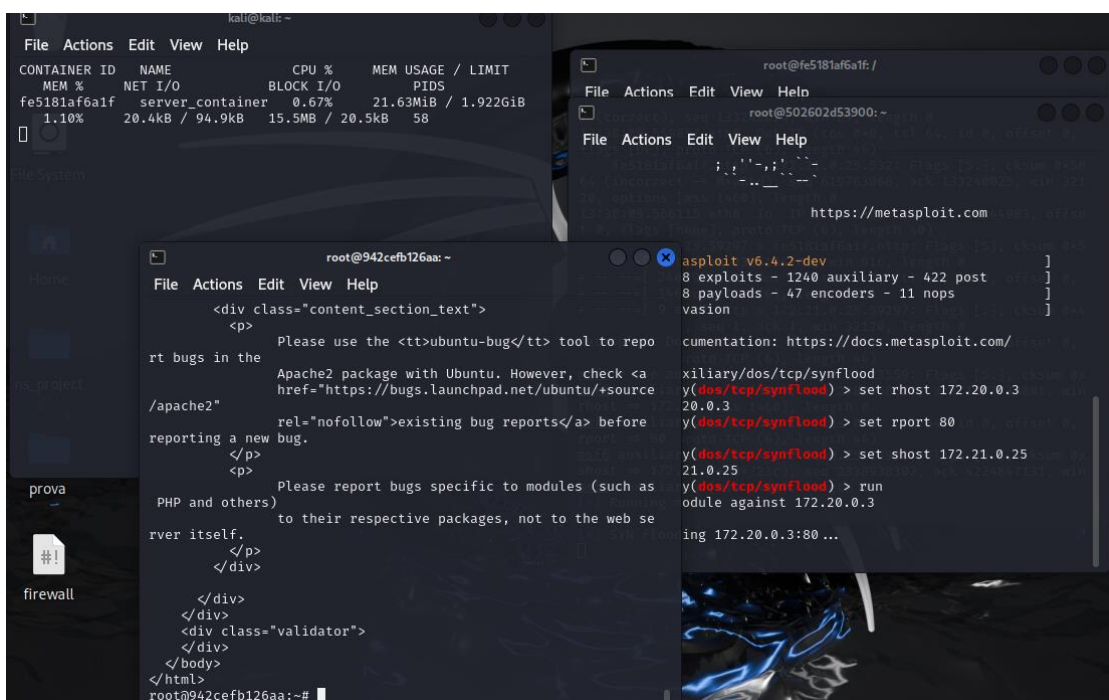
```
3 # Pulisce tutte le regole iptables esistenti
4 iptables -F
5 iptables -X
6
7 # Politica predefinita: blocca tutto il traffico in ingresso e permette tutto il traffico in uscita
8 iptables -P INPUT DROP
9 iptables -P FORWARD DROP
10 iptables -P OUTPUT DROP
11
12 # Abilita il forwarding del traffico IPv4
13 echo 1 > /proc/sys/net/ipv4/ip_forward
14
15 # Crea una nuova catena chiamata "RATE_LIMIT" per gestire il rate limiting
16 iptables -N RATE_LIMIT
17
18 # Imposta un limite di 10 pacchetti al secondo per il traffico in ingresso
19 iptables -A RATE_LIMIT -m hashlimit --hashlimit 10/s --hashlimit-mode srcip --hashlimit-name conn_rate_limit -j ACCEPT
20
21 # Se superato il limite, i pacchetti vengono droppati
22 iptables -A RATE_LIMIT -j DROP
23
24 # Consentire il forwarding del traffico HTTP e HTTPS tra il server web e il mondo esterno
25 # Inoltre il server potrà solo rispondere e non iniziare una nuova connessione
26
27 iptables -A FORWARD -i eth0 -p tcp --syn -d 172.20.0.3 --dport 80 -m state --state NEW -j RATE_LIMIT
28
29 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 80 -m state --state ESTABLISHED -j ACCEPT
30
31 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT
32
33 iptables -A FORWARD -i eth0 -p tcp --syn -d 172.20.0.3 --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
34
35 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 443 -m state --state RELATED,ESTABLISHED -j ACCEPT
36
```



Per fare ciò è stata creata una nuova catena denominata RATE\_LIMIT in modo tale che quando arriva dalla external\_network un pacchetto che vuole iniziare una connessione TCP con il flag SYN alto questo passi per le regole specificate nella nuova catena creata. La seconda banalmente scarta i pacchetti che non trovano corrispondenza con la prima ovvero i pacchetti che sono in eccesso verranno scartati, mentre la prima è più articolata:

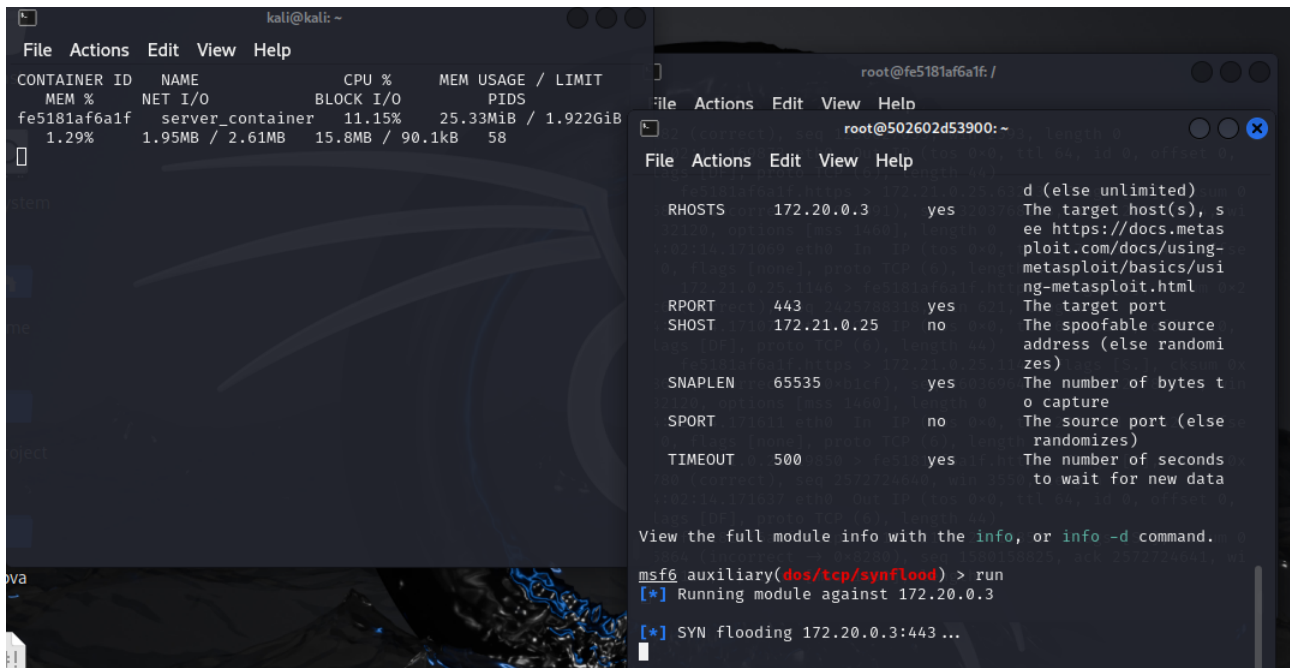
- *-m hashlimit*: specifica l'uso del modulo hashlimit, che consente di limitare il tasso di pacchetti in base ad un hash
- *--hashlimit 10/s*: imposta il limite massimo di pacchetti a 10 al secondo
- *--hashlimit-mode srcip*: specifica la modalità di hashing basata sull'indirizzo IP sorgente, il che significa che il limite si applica a ciascun indirizzo IP sorgente separatamente
- *--hashlimit-name conn\_rate\_limit*: assegna un nome al limite impostato. È utile per riferirsi a questo limite in altre parti delle regole iptables

Con questa modifica verrà attivato il limite del rate di pacchetti host per host sulla base del loro indirizzo IP in modo tale che se uno supera il rate imposto un host diverso possa comunque iniziare la comunicazione col server.



Dall'immagine si nota come a differenza di prima il client riceva la risposta dal server mentre il client kali sta perpetrando l'attacco.

Ora si è testato il SYN flood anche per le connessioni HTTPS cambiando semplicemente la porta destinazione del modulo metasploit da 80 a 443.



The screenshot displays two terminal windows. The left window, titled 'kali@kali: ~', shows the output of the 'docker ps' command, listing a container named 'server\_container' with ID 'fe5181af6a1f'. The right window, titled 'root@fe5181af6a1f: /', shows a Metasploit session where the 'auxiliary(dos/tcp/synflood)' module is being executed against the target IP 172.20.0.3 on port 443. The output of the module execution is visible, showing the target host, port, and the status of the SYN flood attack.

```
kali@kali: ~  
File Actions Edit View Help  
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O  PIDS  
fe5181af6a1f   server_container  11.15%    25.33MiB / 1.922GiB  1.29%     1.95MB / 2.61MB  15.8MB / 90.1kB  58
```

```
root@fe5181af6a1f: /  
File Actions Edit View Help  
RHOSTS        172.20.0.3      yes       d (else unlimited)  
The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html  
RPORT         443             yes       The target port  
SHOST         172.21.0.25     no        The spoofable source address (else randomizes)  
SNAPLEN       65535           yes       The number of bytes to capture  
SPORT         0               no        The source port (else randomizes)  
TIMEOUT       500             yes       The number of seconds to wait for new data  
View the full module info with the info, or info -d command.  
msf6 auxiliary(dos/tcp/synflood) > run  
[*] Running module against 172.20.0.3  
[*] SYN flooding 172.20.0.3:443 ...
```

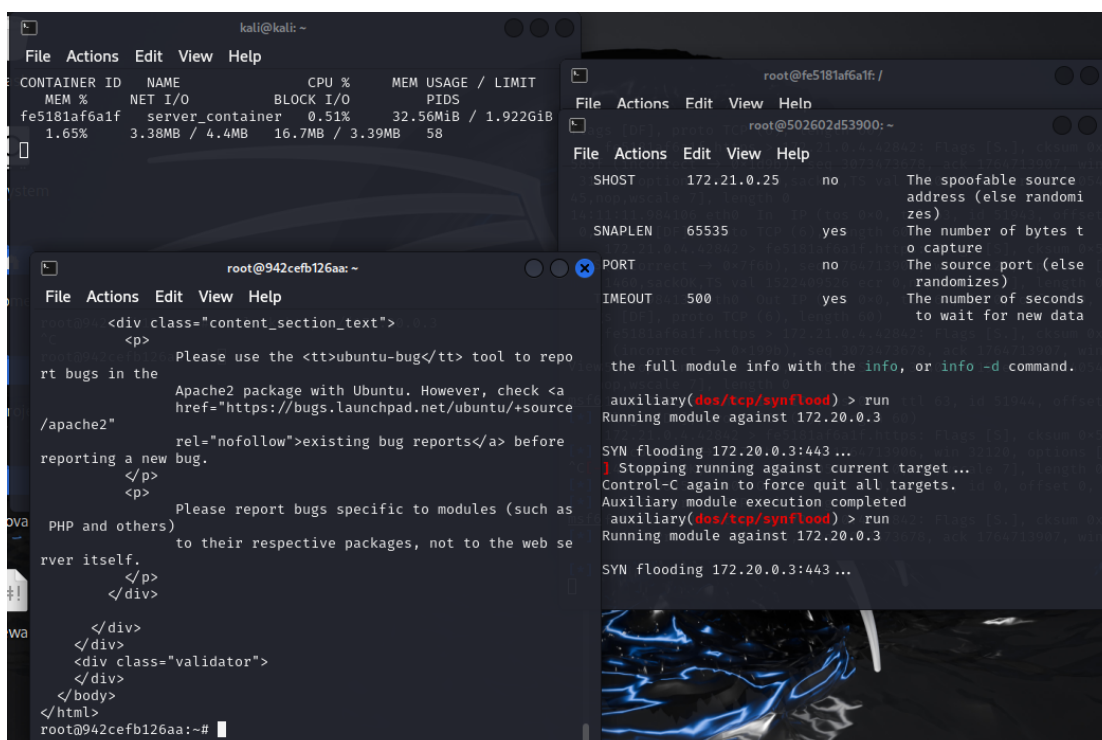
Per mitigare l'attacco sono state applicate le stesse modifiche alle regole che permettono il traffico HTTPS.

```

3 # Pulisce tutte le regole iptables esistenti
4 iptables -F
5 iptables -X
6
7 # Politica predefinita: blocca tutto il traffico in ingresso e permette tutto il traffico in uscita
8 iptables -P INPUT DROP
9 iptables -P FORWARD DROP
10 iptables -P OUTPUT DROP
11
12 # Abilita il forwarding del traffico IPv4
13 echo 1 > /proc/sys/net/ipv4/ip_forward
14
15 # Crea una nuova catena chiamata "RATE_LIMIT" per gestire il rate limiting
16 iptables -N RATE_LIMIT
17
18 # Imposta un limite di 10 pacchetti al secondo per il traffico in ingresso
19 iptables -A RATE_LIMIT -m hashlimit --hashlimit 10/s --hashlimit-mode srcip --hashlimit-name conn_rate_limit -j ACCEPT
20
21 # Se superato il limite, i pacchetti vengono droppati
22 iptables -A RATE_LIMIT -j DROP
23
24 # Consentire il forwarding del traffico HTTP e HTTPS tra il server web e il mondo esterno
25 # Inoltre il server potrà solo rispondere e non iniziare una nuova connessione
26
27 iptables -A FORWARD -i eth0 -p tcp --syn -d 172.20.0.3 --dport 80 -m state --state NEW -j RATE_LIMIT
28
29 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 80 -m state --state ESTABLISHED -j ACCEPT
30
31 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT
32
33 iptables -A FORWARD -i eth0 -p tcp --syn -d 172.20.0.3 --dport 443 -m state --state NEW -j RATE_LIMIT
34
35 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 443 -m state --state ESTABLISHED -j ACCEPT
36
37 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 443 -m state --state RELATED,ESTABLISHED -j ACCEPT
38

```

Infine è stato testato se il client riesce a connettersi al server mentre il client malevolo effettua l'attacco, notando che anche per le connessioni HTTPS le regole permettono la comunicazione per il primo.



The screenshot shows a Kali Linux terminal with three windows. The top window displays the output of the 'docker ps' command, showing a container named 'server\_container' with ID 'fe5181af6a1f'. The middle window shows the output of the 'docker exec' command, displaying the output of the 'auxiliary(dos/tcp/synflood) > run' command, indicating a SYN flood attack on 172.20.0.3:443. The bottom window shows the output of the 'auxiliary(dos/tcp/synflood) > run' command, indicating a SYN flood attack on 172.20.0.3:443.

```

CONTAINER ID   NAME          CPU %   MEM USAGE / LIMIT   PIDS
fe5181af6a1f   server_container  0.51%   32.56MiB / 1.92GiB   58
1.65%         3.38MB / 4.4MB      16.7MB / 3.39MB

root@fe5181af6a1f: /
root@502602d53900: ~
File Actions Edit View Help
SHOST      172.21.0.25    no      The spoofable source address (else randomizes)
SNAPLEN    65535          yes     The number of bytes to capture
PORT       443            no      The source port (else randomizes)
IMEOUT     500            yes     The number of seconds to wait for new data

the full module info with the info, or info -d command.

auxiliary(dos/tcp/synflood) > run
Running module against 172.20.0.3

SYN flooding 172.20.0.3:443 ...
] Stopping running against current target...
Control-C again to force quit all targets.
Auxiliary module execution completed
auxiliary(dos/tcp/synflood) > run
Running module against 172.20.0.3

SYN flooding 172.20.0.3:443 ...

```

## 2.2 Slowloris

Lo Slowloris è un attacco sempre di tipo DoS che mira ad esaurire la capacità di un web server nell'accettare nuove connessioni. La particolarità di questo attacco, a differenza di altri sempre di tipo DoS, sta nel fatto che genera richieste HTTP a basso volume e velocità ridotta.

Per comprendere meglio l'attacco bisogna ricordare il protocollo HTTP gestisce le richieste e le risposte:

- La richiesta è composta da un header, una riga vuota (doppio carriage return e line feed) e un eventuale payload
- Il server dopo aver ricevuto l'header di una richiesta mantiene la connessione attiva e solo dopo aver ricevuto la riga vuota e l'eventuale payload potrà rispondere e chiudere successivamente la connessione

L'attaccante, quindi, invia periodicamente richieste HTTP incomplete nelle quali dopo l'header non vi è una riga vuota. Dopo un certo timeout TCP potrebbe chiudere la connessione ma l'attaccante, conoscendo questa informazione che è disponibile all'atto della richiesta, invia nuovamente poco prima dello scadere dello stesso di nuovo l'header senza la riga vuota. Così facendo il server manterrà la connessione attiva in attesa della riga vuota (e di un eventuale payload). Quindi vengono aperte verso il server contemporaneamente più connessioni di questo tipo per saturarne la capacità di aprire nuove connessioni e impedendo di fatto a utenti legittimi di comunicare con il web server.

Questo attacco è subdolo in quanto non genera una grossa mole di dati e inoltre fa richieste HTTP legittime.



una considerevole quantità di CPU. Nello specifico l'attacco poco prima dello scadere del timeout di TCP invia per tutte le connessioni aperte un header di tipo keep-alive per tenere le stesse continuamente attive.

## 2.2.1 Mitigazione Slowloris

Per poter mitigare parzialmente questo tipo di attacco (dato che potrebbero servire altri tipi di contromisure per contrastare l'attacco) con il firewall implementato è stata aggiunta una nuova regola in cima alla catena FORWARD nella quale si specifica un numero massimo di connessioni aperte pari a 10.

```
2
3 # Pulisce tutte le regole iptables esistenti
4 iptables -F
5 iptables -X
6
7 # Politica predefinita: blocca tutto il traffico in ingresso e permette tutto il traffico in uscita
8 iptables -P INPUT DROP
9 iptables -P FORWARD DROP
10 iptables -P OUTPUT DROP
11
12 # Abilita il forwarding del traffico IPv4
13 echo 1 > /proc/sys/net/ipv4/ip_forward
14
15 # Crea una nuova catena chiamata "RATE_LIMIT" per gestire il rate limiting
16 iptables -N RATE_LIMIT
17
18 # Imposta un limite di 10 pacchetti al secondo per il traffico in ingresso
19 iptables -A RATE_LIMIT -m hashlimit --hashlimit 10/s --hashlimit-mode srcip --hashlimit-name conn_rate_limit -j ACCEPT
20
21 # Se superato il limite, i pacchetti vengono droppati
22 iptables -A RATE_LIMIT -j DROP
23
24 # Imposta un limite di 10 connessioni TCP contemporanee per ogni host che tenta di connettersi al server
25 iptables -A FORWARD -p tcp -d 172.20.0.3 --dport 80 -m conntrack --ctstate NEW -m connlimit --connlimit-above 10 --connlimit-mask 32 -j DROP
26
27 # Consentire il forwarding del traffico HTTP e HTTPS tra il server web e il mondo esterno
28 # Inoltre il server potrà solo rispondere e non iniziare una nuova connessione
29
30 iptables -A FORWARD -i eth0 -p tcp --syn -d 172.20.0.3 --dport 80 -m state --state NEW -j RATE_LIMIT
31
32 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 80 -m state --state ESTABLISHED -j ACCEPT
33
34 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT
35
36 iptables -A FORWARD -i eth0 -p tcp --syn -d 172.20.0.3 --dport 443 -m state --state NEW -j RATE_LIMIT
37
38 iptables -A FORWARD -i eth0 -p tcp -d 172.20.0.3 --dport 443 -m state --state ESTABLISHED -j ACCEPT
39
40 iptables -A FORWARD -i eth1 -p tcp -s 172.20.0.3 --sport 443 -m state --state RELATED,ESTABLISHED -j ACCEPT
41
```

In particolare nella regola sono state specificate le seguenti caratteristiche dei pacchetti:

- `-m conntrack --ctstate NEW`: vengono presi come riferimento solo le nuove connessioni che vengono aperte verso il server web



- -m connlimit --connlimit-above 10: specifica che l'azione (DROP) dovrebbe essere applicata quando il numero di connessioni attive da un singolo indirizzo IP supera il valore di 10.
- --connlimit-mask 32: è un parametro aggiuntivo del modulo connlimit che specifica la maschera di bit da applicare quando si valuta l'indirizzo IP per il controllo del limite di connessione. In particolare indica che il controllo del limite di connessione deve essere effettuato considerando l'intero indirizzo IP, ovvero con una maschera di bit di 32 in modo tale che se un host supera le 10 connessioni attive contemporaneamente la regola non impedisca a un host diverso di potersi connettere al server.

```

kali@kali: ~
File Actions Edit View Help
CONTAINER ID   NAME          CPU %    MEM USAGE / LIMIT   MEM %    NET I/O    BLOCK I/O    PIDS
fe5181af6a1f   server_container  0.15%    23.51MiB / 1.922GiB  1.19%    89.5kB / 438kB  15.8MB / 81.9kB  58

root@fe5181af6a1f: /
File Actions Edit View Help

root@502602d53900: ~
File Actions Edit View Help

root@942cefb126aa: ~
File Actions Edit View Help
<div class="content_section_text">
  <p>
    Please use the <tt>ubuntu-bug</tt> tool to report bugs in the Apache2 package with Ubuntu. However, check <a href="https://bugs.launchpad.net/ubuntu/+source/apache2" rel="nofollow">existing bug reports</a> before reporting a new bug.
  </p>
  <p>
    Please report bugs specific to modules (such as PHP and others) to their respective packages, not to the web server itself.
  </p>
</div>
</div>
<div class="validator">
</div>
</body>
</html>
root@942cefb126aa:~#

host      172.20.0.3   yes
port      80           yes
sockets   150          yes

ssl       false        yes
  sending keep-alive headers
  Randomizes user-agent with each request
  The target address
  The target port
  The number of sockets to use in the attack
  Negotiate SSL/TLS for outgoing connections

the full module info with the info, or info -d command.

auxiliary(dos/http/slowloris) > run

Starting server ...
Attacking 172.20.0.3 with 150 sockets
Creating sockets ...
Sending keep-alive headers ... Socket count: 10
Sending keep-alive headers ... Socket count: 10
  
```

In figura si nota che nonostante nel modulo metasploit sia stato specificato un numero di socket pari a 150, solo 10 riescono a connettersi con il server mentre le restanti 140 vengono scartate. Il client legittimo, inoltre, riesce comunque a

stabilire una connessione verso il server nonostante siano state raggiunte dall'attaccante il numero massimo di connessioni aperte verso di esso.



### 3 Migliorie da poter applicare

Essendo che l'elaborato era incentrato alla creazione di un firewall in linux tramite iptables, le contromisure messe in atto per contrastare gli attacchi SYN flood e Slowloris sono solo parziali e non tengono conto del traffico reale su di un web server, dato che comunque non è stata deployata alcuna web app su di esso.

In particolare per quanto riguarda il SYN flood bisognerebbe innanzitutto monitorare il traffico legittimo verso un web server con una web app per poter tarare opportunamente il rate giusto di connessioni nuove da accettare. Inoltre potrebbe essere necessario replicare la web app su più web server in modo tale da essere più resiliente nel caso in cui avvenga un attacco SYN flood da più di una sorgente (attacco DDoS) ed eventualmente utilizzare un load balancer per poter distribuire opportunamente il traffico.

Per quanto riguarda lo Slowloris, oltre a monitorare un traffico legittimo per poter tarare un numero adeguato di connessioni contemporaneamente attive per un host, una contromisura specifica per questo attacco è di utilizzare la tecnica del **Delayed Binding**, ovvero demandare ad un load balancer la prima gestione delle richieste in arrivo e inoltrare al server solo quelle che comprendono la riga vuota. Il fatto che comunque al web server deve essere concesso comunque un adeguato numero di connessioni contemporaneamente attive con uno stesso host risiede nel fatto che talvolta, le applicazioni client (come i browser web) possono mantenere connessioni persistenti con il server per inviare più richieste senza dover aprire e chiudere una connessione per ogni singola richiesta. Questo è noto come "keep-alive connection". In questo scenario, il server può mantenere aperte più connessioni

con lo stesso host per gestire le richieste successive senza dover ristabilire la connessione ogni volta oppure alcuni servizi web o applicazioni possono richiedere comunicazioni complesse o persistenti tra server e client. In queste situazioni, possono essere necessarie più connessioni per supportare varie operazioni o scambi di dati tra le parti.

Quindi come detto in precedenza il numero di connessioni contemporaneamente attive verso un host va tarato opportunamente e in base alle esigenze di una eventuale web app deployata sul server.