Network Security

Vincenzo Di Bernardo

February 2023

Configurazione di una rete aziendale protetta da un firewall e dotata di una zona de-militarizzata con simulazione di un attacco Slowloris

Contents

1 Introduzione

L'obiettivo del presente elaborato consiste nella descrizione dettagliata del processo di progettazione e configurazione di un testbed di rete, condotto in un ambiente controllato. Lo scopo del testbed consiste nell'emulare in maniera realistica una tipica configurazione di una rete aziendale protetta da un firewall e dotata di una DMZ (DeMilitarized Zone). Per la realizzazione del firewall, si è fatto uso di iptables, tecnologia sviluppata nell'ambiente Linux. Inoltre, al fine di testare l'affidabilità del sistema, si è simulato un attacco di tipo DoS (Denial-of-Service) noto come Slowloris, proponendo altresì una possibile contromisura.

1.1 Firewall

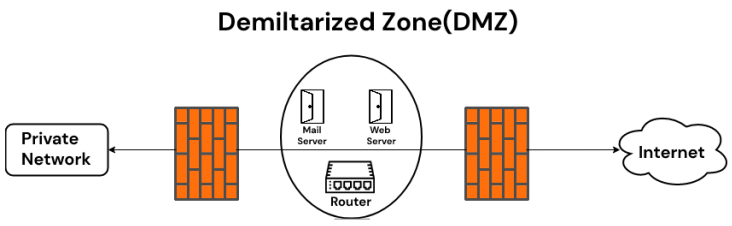
Il firewall rappresenta una struttura hardware e software, avente il compito di separare una rete privata dal resto di Internet, permettendo all'amministratore di controllare e gestire il flusso di traffico tra il mondo esterno e le risorse interne. In particolare, il firewall determina quali pacchetti sono autorizzati a transitare e quali devono essere bloccati (Kurose & Ross, 2008).

Conformemente a quanto affermato da Kurose e Ross (2008), i firewall perseguono tre obiettivi principali:

- Tutti i flussi di traffico tra l'esterno e l'interno passano attraverso il firewall.
- Solo il traffico autorizzato, definito dalla politica di sicurezza locale, è autorizzato a transitare.
- Il firewall stesso risulta immune da eventuali tentativi di intrusione.

1.2 DMZ

Una zona demilitarizzata (DMZ) è un'area di una rete informatica che separa e protegge la rete interna dalla rete esterna. La DMZ contiene dispositivi come server web, server di posta elettronica e altri servizi che devono essere accessibili dall'esterno, ma che possono rappresentare una potenziale minaccia per la sicurezza della rete interna.



La DMZ è situata tra il firewall e la rete interna, e spesso è creata configurando una seconda interfaccia di rete sul firewall. Questa seconda interfaccia è config-

urata con un set di regole specifiche che consentono solo il traffico autorizzato, come quello proveniente da indirizzi IP noti o specifici protocolli di rete.

Gli utenti esterni che accedono ai servizi nella DMZ non hanno accesso diretto alla rete interna e non possono quindi compromettere la sicurezza dei dati sensibili. Allo stesso tempo, i dispositivi nella DMZ non possono accedere direttamente alla rete interna, impedendo la diffusione di eventuali minacce.

La DMZ può essere vista come una sorta di buffer di sicurezza tra le reti interne ed esterne. Essa rappresenta una componente essenziale dell'architettura di sicurezza di molte aziende e organizzazioni, in quanto consente di gestire e proteggere il flusso di informazioni tra le diverse parti della rete.L'implementazione di una DMZ richiede una pianificazione accurata e una configurazione appropriata. È importante definire con precisione i servizi che devono essere accessibili dall'esterno, impostare le regole di sicurezza per il traffico in entrata e in uscita, e mantenere costantemente monitorata la DMZ per rilevare eventuali attacchi o minacce.

2 Progettazione Rete

La rete è stata suddivisa in segmenti al fine di consentire l'accesso ai server situati all'interno della DMZ per tutte le richieste provenienti dalla rete esterna dell'azienda, al fine di impedire la compromissione della rete interna. Inoltre, si è reso necessario prevedere misure di protezione aggiuntive al fine di garantire la sicurezza della rete interna in caso di compromissione della rete DMZ. Per ragioni di semplicità, è stato installato un singolo web server all'interno della DMZ.

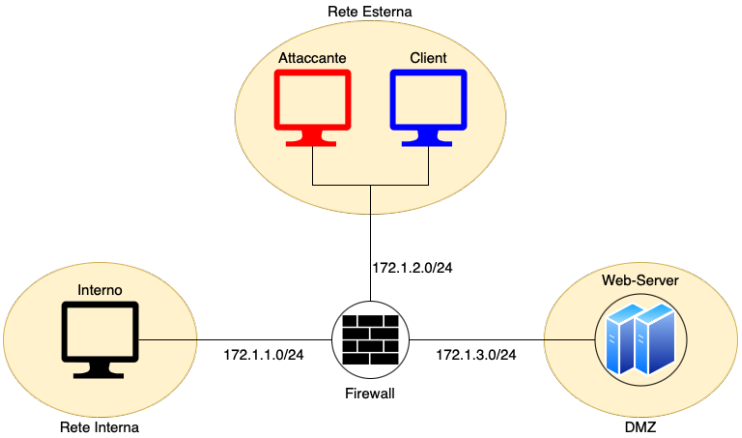


Figure 1: Progettazione Rete

Il firewall, configurato come un router, è un sistema composto da 3 interfacce:

- eth0(172.1.1.2), connesso alla rete interna
- eth1(172.1.2.2), connesso alla rete esterna,
- eth2(172.1.3.2), connesso alla rete DMZ, il cui web-server ha indirizzo 172.1.3.3

2.1 Configurazione Rete

Per realizzare la rete discussa precedentemente, è stato utilizzato Docker, una piattaforma open-source che consente di creare, distribuire e gestire applicazioni in contenitori leggeri e autonomi, garantendo una maggiore efficienza, flessibilità e portabilità rispetto alle tradizionali macchine virtuali. I container Docker isolano l'applicazione e le sue dipendenze dal sistema operativo e dall'ambiente di esecuzione, consentendo di eseguire lo stesso software in modo uniforme su diversi ambienti e piattaforme.

Insieme a Docker è stato utilizzato il tool Docker Compose per semplificare la gestione e la creazione di applicazioni containerizzate.

Mentre Docker permette di eseguire le applicazioni in modo coerente e affidabile su qualsiasi ambiente, eliminando così i problemi di compatibilità e dipendenze che possono sorgere con l'esecuzione di applicazioni su host diversi, con Docker Compose, è possibile descrivere l'intera applicazione, compresi i servizi e le dipendenze tra di essi, in un file di configurazione YAML. In questo modo, grazie al comando

docker compose up

è possibile eseguire l'intera applicazione, semplificando notevolmente il processo di sviluppo, distribuzione e gestione.

2.2 Docker Compose

In questa sezione è riportato il file YAML Docker Compose realizzato per poter implementare tutto il testbed dell'architettura di rete con le diverse reti e servizi che operano su di essa.

```yaml
version: '3'
networks:
  rete_interna:
    name: rete_interna
    ipam:
      config:
        - subnet: 172.1.1.0/24
  rete_esterna:
    name: rete_esterna
    ipam:
      config:
        - subnet: 172.1.2.0/24
  DMZ:
    name: rete_DMZ
    ipam:
      config:
        - subnet: 172.1.3.0/24

services:
  firewall:
    image: ubuntu:latest
    hostname: myfirewall
    container_name: firewall
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
```

```
                                        nano \
                                        nmap \
                                        iptables && \
                    sysctl -w net.ipv4.ip_forward=1 && \
                    update-alternatives --set iptables /usr/sbin/iptables-legacy && \
                    update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy && \
                    bash"
    volumes:
      - ./regole.sh:/firewall/regole.sh
    networks:
      rete_interna:
        ipv4_address: 172.1.1.2
      rete_esterna:
        ipv4_address: 172.1.2.2
      DMZ:
        ipv4_address: 172.1.3.2

  interno1:
    image: ubuntu:latest
    hostname: interno1
    container_name: interno1
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        nmap \
                                        iproute2 && \
                    ip route add 172.1.0.0/16 via 172.1.1.2 dev eth0 && \
                    bash"
    networks:
      rete_interna:
        ipv4_address: 172.1.1.3

  client1:
    image: ubuntu:latest
    hostname: client1
    container_name: client1
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        nmap \
```

```
                                    curl \
                                    iproute2 && \
                    ip route add 172.1.0.0/16 via 172.1.2.2 dev eth0 && \
                    bash"
    networks:
      rete_esterna:
        ipv4_address: 172.1.2.3

  attaccante1:
    image: ubuntu:latest
    hostname: attaccante1
    container_name: attaccante1
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        nmap \
                                        iproute2 \
                                        python3-pip && \
                    ip route add 172.1.0.0/16 via 172.1.2.2 dev eth0 && \
                    bash"
    volumes:
      - ./slowloris:/slowloris
    networks:
      rete_esterna:
        ipv4_address: 172.1.2.4

  server1:
    image: linode/lamp
    hostname: server1
    container_name: server1
    tty: true
    privileged: true
    ports:
      - "80:80"
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        iproute2 && \
                    ip route add 172.1.0.0/16 via 172.1.3.2 dev eth0 && \
                    service apache2 start && \
                    bash"
    networks:
```

```
      DMZ:
        ipv4_address: 172.1.3.3
```

2.2.1 Networks

Il codice definisce un'architettura di rete complessa con diverse subnet e servizi che operano su di esse. In particolare, la parte iniziale del codice definisce tre reti diverse: rete_interna, rete_esterna e DMZ, ciascuna con il proprio intervallo di indirizzi IP.

```
version: '3'
networks:
  rete_interna:
    name: rete_interna
    ipam:
      config:
        - subnet: 172.1.1.0/24
  rete_esterna:
    name: rete_esterna
    ipam:
      config:
        - subnet: 172.1.2.0/24
  DMZ:
    name: rete_DMZ
    ipam:
      config:
        - subnet: 172.1.3.0/24
```

2.2.2 Services

Successivamente, vengono definiti i 5 servizi (firewall, client1, attaccante1, interno1, server1) che simulano i nodi dell'architettura, ognuno dei quali viene eseguito in un contenitore Docker separato. Tutti i servizi vengono implementati sulla più recente immagine Ubuntu, fatta eccezione per il container denominato "server1" che ha come immagine Linode LAMP, ovvero un'immagine di sistema preconfigurata e ottimizzata per eseguire un'installazione di LAMP (Linux, Apache, MySQL e PHP) su un server Linux. All'avvio dei container, grazie all'esecuzione di comandi su shell, installiamo tutta una serie di strumenti di rete.

```
firewall:
    image: ubuntu:latest
    hostname: myfirewall
    container_name: firewall
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
```

```
                apt-get install -y  bridge-utils \
                                    net-tools \
                                    iputils-ping \
                                    nano \
                                    nmap \
                                    iptables && \
                sysctl -w net.ipv4.ip_forward=1 && \
                update-alternatives --set iptables /usr/sbin/iptables-legacy && \
                update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy && \
                bash"
    volumes:
      - ./regole.sh:/firewall/regole.sh
    networks:
      rete_interna:
        ipv4_address: 172.1.1.2
      rete_esterna:
        ipv4_address: 172.1.2.2
      DMZ:
        ipv4_address: 172.1.3.2
```

Il primo servizio che andiamo ad analizzare è il firewall. In questo container, oltre agli strumenti di rete, viene anche installato iptables, un firewall di rete incorporato nel kernel Linux. Con iptables è possibile filtrare il traffico di rete in entrata e in uscita da un sistema, permettendo o bloccando determinati pacchetti sulla base di vari criteri, come ad esempio l'indirizzo IP di origine, la porta di origine o di destinazione, il protocollo utilizzato e tanto altro.

Dovendo funzionare anche come un router è stato inoltre configurato il forwarding dei pacchetti (sysctl -w net.ipv4.ip_forward=1) tra le reti. All'interno del container è stato montato un file di regole presente sull'host (regole.sh) nella cartella "firewall" del container omonimo. Infine, il servizio viene collegato alle tre reti definite precedentemente, ognuna con un indirizzo IP assegnato.

```
interno1:
    image: ubuntu:latest
    hostname: interno1
    container_name: interno1
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                apt-get install -y  bridge-utils \
                                    net-tools \
                                    iputils-ping \
                                    nmap \
                                    iproute2 && \
                ip route add 172.1.0.0/16 via 172.1.1.2 dev eth0 && \
```

```
                bash"
    networks:
      rete_interna:
        ipv4_address: 172.1.1.3

  client1:
    image: ubuntu:latest
    hostname: client1
    container_name: client1
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        nmap \
                                        curl \
                                        iproute2 && \
                    ip route add 172.1.0.0/16 via 172.1.2.2 dev eth0 && \
                    bash"
    networks:
      rete_esterna:
        ipv4_address: 172.1.2.3

  attaccante1:
    image: ubuntu:latest
    hostname: attaccante1
    container_name: attaccante1
    tty: true
    privileged: true
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        nmap \
                                        iproute2 \
                                        python3-pip && \
                    ip route add 172.1.0.0/16 via 172.1.2.2 dev eth0 && \
                    bash"
    volumes:
      - ./slowloris:/slowloris
    networks:
      rete_esterna:
        ipv4_address: 172.1.2.4
```

Il servizio interno1 viene assegnato solo alla rete rete_interna, mentre client1 e

attaccante1 vengono entrambi assegnati alla rete rete_esterna. Inoltre, i servizi client1 e attaccante1 vengono eseguiti con comandi differenti, rispettivamente per simulare un client e un attaccante esterni alla rete.

Nel container client1 è stato installato il comando di interfaccia a riga di comando *"curl"* utilizzato per trasferire dati da o verso un server, utilizzando uno dei molti protocolli supportati, come HTTP, HTTPS, FTP, SMTP, POP3 e altri ancora. Nel nostro caso ci servirà per richiedere la pagina web al web server.

Nel container attaccante1, invece è stato installato *"python3-pip"*, ovvero il pacchetto del gestore di pacchetti pip per Python3, necessario per far partire l'attacco DoS Slowloris, montato nella cartella *slowloris* del container.

```
server1:
    image: linode/lamp
    hostname: server1
    container_name: server1
    tty: true
    privileged: true
    ports:
      - "80:80"
    command: sh -c "apt-get update && \
                    apt-get install -y  bridge-utils \
                                        net-tools \
                                        iputils-ping \
                                        iproute2 && \
                    ip route add 172.1.0.0/16 via 172.1.3.2 dev eth0 && \
                    service apache2 start && \
                    bash"
    networks:
      DMZ:
        ipv4_address: 172.1.3.3
```

Il servizio server1 è stato implementato su un immagine Linux linode/lamp, la quale contiene un server web Apache preconfigurato. In questo caso, il servizio viene assegnato solo alla rete DMZ e viene mappata la porta 80 sulla porta 80 del contenitore. All'avvio del container viene eseguito anche il comando per l'avvio del web-server Apache (*service apache2 start*).

Per poter connettere le diverse sottoreti tra di loro, è assolutamente necessario che i container siano e restino in esecuzione dopo la creazione. Per fare in modo che ciò accada è necessario impostare l'opzione ***tty: true*** e soprattutto far eseguire il comando ***bash*** a fine istruzione, in modo che i container restino attivi su quel comando.

Si noti che in tutti i container, fatta eccezione per il container firewall, sono state aggiunte delle rotti (route) statiche nella tabella di routing del sistema operativo Linux, grazie al comando:

ip route add <destinazione> via <gateway> <opzioni>

dove:

- **<destinazione>** è l'indirizzo di rete di destinazione della rotta. Può essere specificato come un indirizzo IP o come un nome di rete.
- **<gateway>** è l'indirizzo IP del gateway di rete da utilizzare per raggiungere la destinazione.
- **<opzioni>**: ulteriori opzioni per la configurazione della rotta, come la specifica dell'interfaccia di rete da utilizzare per inviare il traffico.

3 Firewall IPTables

In questa sezione sono presentate le istruzioni dello script che permettono l'esecuzione delle regole iptables sul sistema firewall. Si vuole sottolineare che questa trattazione non vuole essere una guida su come usare le regole iptables, dunque verranno evidenziati solo gli aspetti necessari a comprendere lo script ai fini dell'illustrazione del progetto.

Un Packet Filtering Firewall può vedere informazioni di tipo indirizzo sorgente, indirizzo destinazione, Transport Level Address (porta destinazione e sorgente), il campo IP Protocol (quindi l'header IP) e l'interfaccia che stiamo utilizzando.

Banalmente un firewall contiene regole. Ogni regola di iptables utilizzata è strutturata nel seguente modo:

- tabella a cui attribuire la regola:
  - filter: tabella (di default) utilizzata per decidere se accettare o rifiutare i pacchetti
  - nat: tabella usata per alterare e instradare i pacchetti
- catena a cui attribuire la regola
- condizioni che fanno applicare la regola al pacchetto in esame
- azione da intraprendere

A seconda della tabella si hanno diverse catene:

- Tabella filter
  - INPUT, raggruppa tutte le regole riguardanti i pacchetti che arrivati ad una delle interfacce di rete della macchina devono essere indirizzati alla macchina stessa
  - FORWARD, raggruppa tutte le regole riguardanti i pacchetti che arrivati ad una delle interfacce di rete della macchina devono essere instradati direttamente ad un'altra sua interfaccia, poiché non destinati a questa macchina
  - OUTPUT, raggruppa tutte le regole riguardanti i pacchetti generati dalla macchina e destinatiad essere inviati altrove
- Tabella NAT
  - PREROUTING, raggruppa tutte le regole che modificano i pacchetti in arrivo ad una delle interfacce della macchina, prima che raggiungano la tabella filter
  - POSTROUTING, raggruppa tutte le regole che modificano i pacchettiin uscita da una delle interfacce della macchina, dopo che hanno già

oltrepassato la tabella filter, catena output o catena forwarding a seconda della provenienza del pacchetto

- OUTPUT, raggruppa tutte le regole che modificano i pacchetti generati dalla macchina prima di transitare attraverso la catena output della tabella filter

3.1 Regole IPTables per configurazione DMZ

La prima cosa da fare è quella di cancellare tutte le regole precedenti presenti nelle chains delle tabelle: infatti il comportamento predefinito di iptables in Debian è una policy di tipo ACCEPT, ovvero risulta permesso tutto il traffico in ogni direzione a meno l'utente non definisca esplicitamente delle regole che lo blocchino.

```
# Flush ed eliminazione catene vuote
iptables -F
iptables -F -t nat
iptables -X
```

Successivamente sono state impostate le policy di base (policy DROP), utilizzando l'opzione -P:

```
# Policy di base
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Nel nostro caso stiamo imponendo di scartare ogni pacchetto transitante attraverso la catena INPUT, OUTPUT e FORWARD della tabella filter (default).

Con le seguenti istruzioni si eliminano i pacchetti non validi e quelli che possono compromettere la sicurezza del sistema:

```
# Pacchetti non validi
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A OUTPUT -m state --state INVALID -j DROP
iptables -A FORWARD -m state --state INVALID -j DROP
```

Le seguenti regole servono a proteggere la rete dall'accesso non autorizzato e da altri tipi di attività malevole che possono tentare di sfruttare vulnerabilità del protocollo TCP:

```
# Regole di sicurezza
iptables -A FORWARD -f -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL ACK,RST,SYN,FIN -j DROP
```

```
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
```

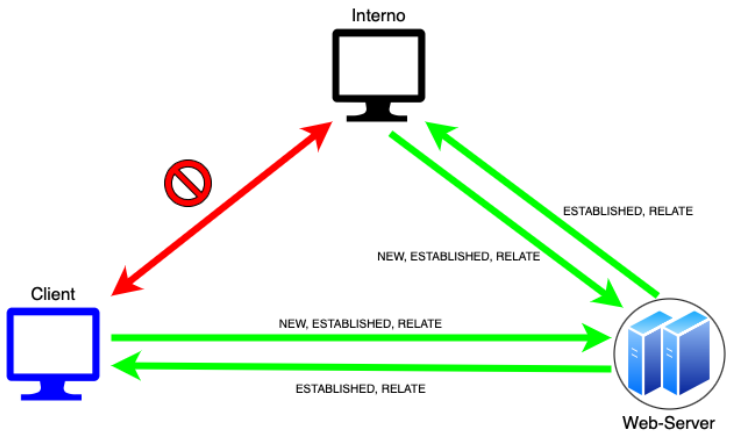
La prima regola serve a bloccare i pacchetti che hanno il flag di frammentazione IP (IP fragmentation flag) impostato. Il flag di frammentazione IP indica che un pacchetto IP è stato frammentato in più parti per essere trasmesso attraverso una rete che ha una MTU (Maximum Transmission Unit) inferiore rispetto alla dimensione del pacchetto originale.In alcune situazioni, i pacchetti frammentati possono essere utilizzati in attacchi di tipo ”Denial of Service” (vedremo di fatti come funziona un attacco del genere con slowloris) o ”Distributed Denial of Service” (DDoS), per cui un attaccante invia un gran numero di pacchetti frammentati per saturare la rete o un server di destinazione.

Di fatti la regola iptables che blocca i pacchetti con il flag di frammentazione (-f) impostato serve a proteggere la rete da questo tipo di attacchi. Quando un pacchetto con il flag di frammentazione arriva alla catena FORWARD del firewall, la regola lo riconosce e fa dropping del pacchetto.

La seconda regola blocca i pacchetti TCP che hanno tutti e quattro i flag ACK, RST, SYN e FIN impostati contemporaneamente. Questo tipo di pacchetto è spesso associato a tentativi di scansione di rete o altre attività sospette.

La terza regola blocca i pacchetti TCP che hanno sia i flag SYN che FIN impostati, ma non altri flag. Questo tipo di pacchetto viene spesso usato per fare scansione di porte.

La quarta regola blocca i pacchetti TCP che hanno sia i flag SYN che RST impostati, ma non altri flag. Questo tipo di pacchetto può essere associato a attacchi di tipo ”SYN flood” o altre forme di attacco DDoS.



Si rende necessario permettere in maniera sicura il passaggio del traffico di rete tra la rete esterna e la rete della zona demilitarizzata , nonché tra la rete interna

e la rete DMZ. A tal fine, è necessario adottare adeguate misure di configurazione del firewall al fine di garantire la protezione della rete informatica da eventuali minacce esterne e di preservare la sicurezza dei dati scambiati attraverso la rete. In particolare, è possibile definire specifiche regole iptables che consentano il traffico solo in base a determinati criteri di sicurezza, come l'accesso solo a determinati servizi o l'impedimento dell'accesso a indirizzi IP non autorizzati.

```
# Interna (eth2) <--> (eth0) DMZ
iptables -t filter -A FORWARD -i eth2 -o eth0 -m state --state
    NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A FORWARD -i eth0 -o eth2 -m state --state
    ESTABLISHED,RELATED -j ACCEPT
```

Queste due regole iptables servono a consentire il traffico inoltrato attraverso il firewall da e verso due interfacce di rete specifiche (eth2 e eth0) in base allo stato della connessione TCP.

La prima regola consente il traffico TCP in uscita dalla rete locale o rete_interna (eth2) verso la rete_DMZ (eth0) per le connessioni che sono in uno dei tre stati: NEW, ESTABLISHED o RELATED. Lo stato NEW si riferisce alle connessioni appena avviate, mentre gli stati ESTABLISHED e RELATED si riferiscono a connessioni in corso o a connessioni correlate.

La seconda regola consente il traffico TCP in ingresso dalla rete_DMZ (eth0) dalla rete locale (eth2) per le connessioni che sono nello stato ESTABLISHED o RELATED. Questa regola è necessaria poiché il traffico in ingresso è stato già autorizzato dalla prima regola e quindi ha stabilito una connessione, pertanto non ha più bisogno del controllo dello stato NEW. In questo modo, il firewall può proteggere la rete da connessioni non autorizzate o potenzialmente pericolose.

```
# Esterna (eth1) <--> (eth0) DMZ
iptables -t filter -A FORWARD -i eth1 -o eth0 -m state --state
    NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state
    ESTABLISHED,RELATED -j ACCEPT
```

Per garantire una maggiore sicurezza nella gestione del traffico tra la rete esterna e la DMZ, è necessario adottare misure specifiche in termini di configurazione del firewall. Analogamente a quanto fatto per il traffico tra la rete interna e la DMZ, è possibile definire regole iptables che permettano il passaggio del traffico dalla rete_esterna alla DMZ e viceversa.

4 Implementazione e testing

Il presente studio ha lo scopo di dimostrare il funzionamento della rete e di fornire indicazioni su come eseguire autonomamente dei test su di essa. In particolare, la sezione corrente illustrerà la procedura da seguire per effettuare dei test sulla rete utilizzando una macchina su cui è stato installato Docker.

Come evidenziato nelle sezioni precedenti, è stato possibile implementare l'intero lavoro all'interno di un singolo file Docker Compose. Grazie a questa soluzione, è possibile avviare l'intera infrastruttura di rete con un unico comando, il quale provvederà al download delle immagini necessarie e alla creazione dei relativi container associati ai servizi richiesti. In aggiunta, il comando in questione creerà anche le reti definite nel file YAML tra i vari container. Avviati i container si avrà la possibilità di accedere ai singoli container con privilegi di root col fine di eseguire dei test sulla rete utilizzando gli appositi tool installati.

È di fondamentale importanza tenere a mente che, una volta completati gli esperimenti all'interno di uno dei container, non bisogna disconnettersi dal container stesso utilizzando il comando "exit", ma è necessario premere contemporaneamente i tasti CTRL+P e CTRL+Q. In questo modo, lo script passerà ad eseguire il terminale in modalità "root" del successivo container, senza però interrompere l'esecuzione del container precedente. Questa procedura è estremamente importante, poiché se uno qualsiasi dei container non è in esecuzione, non sarà possibile testare correttamente la rete.

4.1 Ispezione delle reti

Una volta che abbiamo mandato in esecuzione il comando:

docker compose up

verranno create le reti e i container associati ai servizi definiti nel docker compose. Docker mette a disposizione degli amministratori di sistema due comandi essenziali per gestire e monitorare le reti all'interno di un ambiente

docker network ls
docker inspect <nome_rete>

Il primo comando consente di ottenere un elenco completo di tutte le reti presenti nel sistema Docker, accompagnate dalle rispettive informazioni, quali nome, ID, driver di rete, stato e numero di container ad esse collegati.

```
vincenzodibernardo@MBP-di-Vincenzo ~ % docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
251c87114bb9   bridge         bridge    local
88823e03a841   host           host      local
d1fa8fdcaef7   none           null      local
895ea6162735   rete_DMZ       bridge    local
97a05b4b31d1   rete_esterna   bridge    local
94344eec5367   rete_interna   bridge    local
```

Il secondo comando, invece, fornisce informazioni dettagliate sulla rete specificata, tra cui i dettagli di configurazione e l'elenco dei container associati ad essa. Grazie a questi due comandi, gli amministratori di sistema sono in grado di monitorare e gestire le reti in modo efficace e preciso.

Per dimostrare la corretta connessione della rete, utilizziamo questo comando per visualizzare i dettagli della rete, compresi i container associati. Grazie a questi comandi, possiamo confermare che la rete è stata creata e configurata correttamente, garantendo una comunicazione sicura e protetta tra i servizi associati alla rete.

```
vincenzodibernardo@MBP-di-Vincenzo ~ % docker inspect rete_interna
[
    {
        "Name": "rete_interna",
        "Id": "94344eec53672b3eebf284c723122d10af01c7d3c694beb74c6897f4a06caa77",
        "Created": "2023-02-17T18:59:25.374637388Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.1.1.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "798084a6bbee513b2a24022911143a3e3d987897a5e7c21fd16f16bab4e2ce7d": {
                "Name": "interno1",
                "EndpointID": "063dffd94519988719ee94cc60af4a53dd7abb593449ecfa603d066b96b4d9b3",
                "MacAddress": "02:42:ac:01:01:03",
                "IPv4Address": "172.1.1.3/24",
                "IPv6Address": ""
            },
            "f42ad2edd7b094dc4a6069de2930977446f68e1c6396e989cf809bddd985c7df": {
                "Name": "firewall",
                "EndpointID": "fa78334b550bed7de46dcd116774774e505e317d2221c931a0e2e3592a73b8d4",
                "MacAddress": "02:42:ac:01:01:02",
                "IPv4Address": "172.1.1.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "rete_interna",
            "com.docker.compose.project": "docker-compose",
            "com.docker.compose.version": "2.15.1"
        }
    }
]
```

```
vincenzodibernardo@MBP-di-Vincenzo ~ % docker inspect rete_esterna
[
    {
        "Name": "rete_esterna",
        "Id": "97a05b4b31d154c09a41f3f2520c436c759e413929a63240c951592ee65b09c4",
        "Created": "2023-02-17T18:59:25.220108222Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.1.2.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "110d2a3b1c1ac576dda41cb954224f9179b38115e9e5c1264ccdccc4be55cb65": {
                "Name": "client1",
                "EndpointID": "4c52735530f35c869f7c22e68227ca5ecb1b430686b5e386ad3d8bd2ea0159c0",
                "MacAddress": "02:42:ac:01:02:03",
                "IPv4Address": "172.1.2.3/24",
                "IPv6Address": ""
            },
            "176f5f7b5ded4750e7d7e551987bf96b8e95b1e527db222361cdd08d2ef99270": {
                "Name": "attaccante1",
                "EndpointID": "fa17a99050fd2f6cf9cf3276868252ff4406461156cb0769c750e05097ebe96b",
                "MacAddress": "02:42:ac:01:02:04",
                "IPv4Address": "172.1.2.4/24",
                "IPv6Address": ""
            },
            "f42ad2edd7b094dc4a6069de2930977446f68e1c6396e989cf809bddd985c7df": {
                "Name": "firewall",
                "EndpointID": "e590c5b8305bc783c05ab3519994f92730af5b82bf3b1c59d049b41b844f6a8d",
                "MacAddress": "02:42:ac:01:02:02",
                "IPv4Address": "172.1.2.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "rete_esterna",
            "com.docker.compose.project": "docker-compose",
            "com.docker.compose.version": "2.15.1"
        }
    }
```

4.2 Container

Dopo aver constatato che la rete è stata creata e connessa correttamente in tutte le sue parti, tramite il comando:

docker ps

è possibile visualizzare l'elenco dei container in esecuzione nel sistema Docker. In particolare, il comando elenca l'ID del container, il nome del container, l'immagine utilizzata, lo stato del container, la porta in ascolto e il tempo di esecuzione del container. Dopo aver trovato il container desiderato, è possibile eseguire il comando:

docker exec -it -t <nome_container> bash

per avviare una sessione interattiva all'interno di un container Docker specifico.

La specifica opzione "-it" consente di eseguire il container in modalità interattiva e di associare il terminale corrente del sistema host a quello del container. In questo modo, è possibile eseguire comandi all'interno del container come se ci si trovasse direttamente all'interno di esso.

4.2.1 Regole.sh

Utilizzando il comando precedentemente descritto, è stato possibile accedere al container *firewall*. All'interno della cartella *firewall* di tale container, è stato rinvenuto il file regole.sh, il quale racchiude tutte le regole che sono state illustrate nella sezione 3.1 del documento. Possiamo notare che prima dell'esecuzione del file ***regole.sh***, iptables non ha alcuna regola nelle chains e le policy di default sono impostate per accettare tutti i pacchetti.

```
root@myfirewall:/firewall# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@myfirewall:/firewall#
```

Tuttavia, una volta avviato il file contenente comandi per iptables, queste chains inizieranno a popolarsi di regole (particolar modo per la chains di forward) e le policy di default verranno impostate su DROP.

```
root@myfirewall:/firewall# iptables -L -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  any    any     anywhere             anywhere             state INVALID

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  any    any     anywhere             anywhere             state INVALID
    0     0 DROP       all  -f  any    any     anywhere             anywhere
    0     0 DROP       tcp  --  any    any     anywhere             anywhere             tcp flags:FIN,SYN,RST,PSH,ACK,URG/FIN,SYN,RST,ACK
    0     0 DROP       tcp  --  any    any     anywhere             anywhere             tcp flags:FIN,SYN/FIN,SYN
    0     0 DROP       tcp  --  any    any     anywhere             anywhere             tcp flags:SYN,RST/SYN,RST
    0     0 ACCEPT     all  --  eth2   eth0    anywhere             anywhere             state NEW,RELATED,ESTABLISHED
    0     0 ACCEPT     all  --  eth0   eth2    anywhere             anywhere             state RELATED,ESTABLISHED
    0     0 ACCEPT     all  --  eth1   eth0    anywhere             anywhere             state NEW,RELATED,ESTABLISHED
    0     0 ACCEPT     all  --  eth0   eth1    anywhere             anywhere             state RELATED,ESTABLISHED

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  any    any     anywhere             anywhere             state INVALID
```

Per avere un maggior grado di comprensione dell'architettura, può risultare utile visualizzare le informazioni relative all'interfaccia di rete mediante l'utilizzo del comando

ifconfig

eseguito all'interno del container firewall. Esso mostra informazioni dettagliate sull'interfaccia di rete come l'indirizzo IP, il MAC address, la subnet mask e lo stato dell'interfaccia.

```
root@myfirewall:/firewall# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.1.3.2  netmask 255.255.255.0  broadcast 172.1.3.255
        ether 02:42:ac:01:03:02  txqueuelen 0  (Ethernet)
        RX packets 4326  bytes 32122622 (32.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4104  bytes 282957 (282.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.1.2.2  netmask 255.255.255.0  broadcast 172.1.2.255
        ether 02:42:ac:01:02:02  txqueuelen 0  (Ethernet)
        RX packets 19  bytes 1478 (1.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.1.1.2  netmask 255.255.255.0  broadcast 172.1.1.255
        ether 02:42:ac:01:01:02  txqueuelen 0  (Ethernet)
        RX packets 19  bytes 1546 (1.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10  bytes 780 (780.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 780 (780.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

4.3 Testing

Il testing della sicurezza di una rete è un aspetto fondamentale per garantire l'affidabilità e la protezione dei sistemi informatici. In particolare, quando si parla di una rete con DMZ (Demilitarized Zone), la verifica della corretta configurazione delle regole del firewall e della connettività tra i vari dispositivi assume un'importanza cruciale. Tra gli strumenti utilizzati per effettuare il testing, il ping è uno dei più semplici ed efficaci. Tramite questo comando, è possibile verificare la connessione tra i diversi nodi di rete e individuare eventuali problemi o malfunzionamenti. In questa sezione, verrà descritto il processo di testing tramite ping e saranno presentati i risultati ottenuti, al fine di valutare la sicurezza e l'affidabilità della rete con DMZ.

4.3.1 Client & Attaccante

```
root@client1:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.1.2.3  netmask 255.255.255.0  broadcast 172.1.2.255
        ether 02:42:ac:01:02:03  txqueuelen 0  (Ethernet)
        RX packets 3593  bytes 2858124 (2.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3598  bytes 231405 (231.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8  bytes 729 (729.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 729 (729.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@client1:/# ping 172.1.3.3
PING 172.1.3.3 (172.1.3.3) 56(84) bytes of data.
64 bytes from 172.1.3.3: icmp_seq=1 ttl=63 time=0.210 ms
64 bytes from 172.1.3.3: icmp_seq=2 ttl=63 time=0.180 ms
64 bytes from 172.1.3.3: icmp_seq=3 ttl=63 time=0.387 ms
^C
--- 172.1.3.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.180/0.259/0.387/0.091 ms
root@client1:/# ping 172.1.1.3
PING 172.1.1.3 (172.1.1.3) 56(84) bytes of data.
^C
--- 172.1.1.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2079ms

root@client1:/# nmap 172.1.3.3 -sT
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-22 04:52 UTC
Nmap scan report for 172-1-3-3.lightspeed.hstntx.sbcglobal.net (172.1.3.3)
Host is up (0.00018s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds
root@client1:/# nmap 172.1.1.3 -sT
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-22 04:52 UTC
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.07 seconds
```

Una volta all'interno del container *client1*, è possibile utilizzare lo strumento ping al fine di verificare la connettività con il web-server *server1*, avente indirizzo IP 172.1.3.3, e con l'host *interno1*, con indirizzo IP 172.1.1.3. Tale operazione consiste nell'invio di pacchetti di dati verso i dispositivi di destinazione, in attesa di ricevere una risposta. Si può quindi notare che dal container *client1* è possibile contattare il web-server e ottenere una risposta, mentre non è possibile contattare l'host *interno1* poiché i pacchetti inviati non saranno fatti passare dal firewall.

I medesimi risultati sono osservabili eseguendo il testing da parte del container *attaccante1* anziché da quello denominato *client1*

4.3.2 Web-server

```
root@server1:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:01:03:03
          inet addr:172.1.3.3  Bcast:172.1.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4960 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4889 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5655679 (5.6 MB)  TX bytes:279662 (279.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:384 (384.0 B)  TX bytes:384 (384.0 B)

root@server1:/# ping 172.1.1.3
PING 172.1.1.3 (172.1.1.3) 56(84) bytes of data.
^C
--- 172.1.1.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1043ms

root@server1:/# ping 172.1.2.3
PING 172.1.2.3 (172.1.2.3) 56(84) bytes of data.
^C
--- 172.1.2.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2042ms
```

In accordo alle politiche di sicurezza definite, il web-server è configurato in modo tale da non consentire l'inizio di connessione con alcun host, sia della rete interna che della rete esterna. Tale configurazione è stata verificata tramite l'invio di pacchetti di tipo ping, i quali hanno evidenziato la mancata ricezione di alcun pacchetto di risposta in seguito ai tentativi di contatto degli host *interno1* (172.1.1.3) e *client1* (172.1.2.3).

4.3.3 Interno

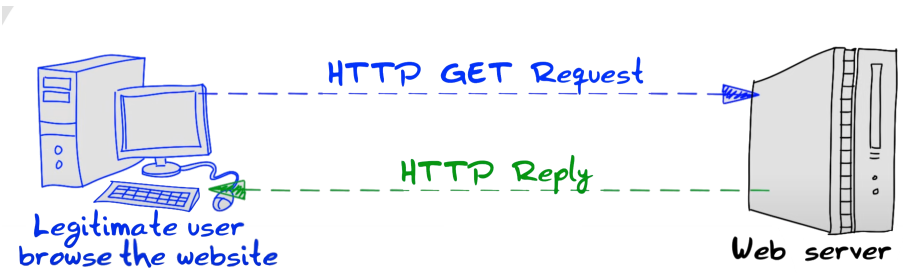
```
[root@interno1:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.1.1.3  netmask 255.255.255.0  broadcast 172.1.1.255
        ether 02:42:ac:01:01:03  txqueuelen 0  (Ethernet)
        RX packets 388  bytes 2682399 (2.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 352  bytes 25137 (25.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 156 (156.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 156 (156.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@interno1:/# ping 172.1.3.3
PING 172.1.3.3 (172.1.3.3) 56(84) bytes of data.
64 bytes from 172.1.3.3: icmp_seq=1 ttl=63 time=0.772 ms
64 bytes from 172.1.3.3: icmp_seq=2 ttl=63 time=0.387 ms
64 bytes from 172.1.3.3: icmp_seq=3 ttl=63 time=0.213 ms
^C
--- 172.1.3.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2032ms
rtt min/avg/max/mdev = 0.213/0.457/0.772/0.233 ms
[root@interno1:/# ping 172.1.2.3
PING 172.1.2.3 (172.1.2.3) 56(84) bytes of data.
^C
--- 172.1.2.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2082ms
```

Nel corso del testing dell'host *interno1*, in maniera speculare al testing del *client1*, emerge la possibilità di contattare il web-server e di ricevere risposta da quest'ultimo, mentre non si riscontra la possibilità di contattare l'host *client1*. Tale circostanza si spiega in ragione del fatto che i pacchetti inviati dal dispositivo client1 non saranno in grado di superare il firewall, che prevede apposite politiche di filtraggio.

5 Slowloris

Lo Slowloris è un tipo di attacco DDoS (Distributed Denial of Service) che sfrutta una vulnerabilità del protocollo HTTP. L'obiettivo di questo attacco è impedire l'accesso al sito web o al server web da parte degli utenti legittimi.

Questo attacco sfrutta una debolezza del protocollo HTTP, il quale richiede che ogni richiesta HTTP venga terminata da una sequenza di caratteri di nuova riga. Una richiesta HTTP legittima è normalmente contenuta in un pacchetto e viene terminata rapidamente dalla sequenza di nuove righe alla fine del messaggio.



Tuttavia, quando un utente malintenzionato utilizza lo strumento Slowloris, la richiesta HTTP viene inviata senza la sequenza di terminazione, causando al server web di lasciare la connessione aperta e allocare le risorse in attesa della sequenza di terminazione.

Il tipico web server alloca risorse limitate per la gestione di risorse delle connessioni aperte, poiché si aspetta che le connessioni siano brevi. Slowloris sfrutta questo comportamento e genera migliaia di richieste in diversi minuti, in cui nessuna richiesta viene terminata. Questo consuma le risorse disponibili per le connessioni aperte dal web server, causando l'interruzione della gestione di nuove richieste e l'impossibilità del servizio da parte di utenti legittimi, ottenendo così un Denial of Service.

L'attacco Slowloris può essere particolarmente efficace perché richiede solo una piccola quantità di risorse da parte dell'attaccante per aprire e mantenere le connessioni. Inoltre, è difficile da individuare, poiché le connessioni sono aperte in modo graduale e sembrano comportarsi come connessioni legittime.

5.1 Simulazione attacco

In un contesto usuale, il client effettua una richiesta GET per ottenere una pagina dal server web, la quale viene restituita in risposta. Per raggiungere tale obiettivo, è possibile utilizzare un programma a riga di comando chiamato *curl*, il quale è disponibile sui principali sistemi operativi, inclusi Linux, macOS e Windows. Curl consente di scaricare o inviare dati a server remoti, verificare le connessioni di rete, inviare richieste HTTP personalizzate e svolgere molte altre funzionalità.

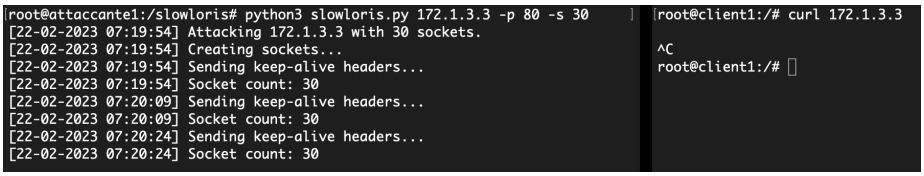
```
root@client1:/# curl 172.1.3.3
<!DOCTYPE html>
  <html>
  <body>

  <br>
  <br>

  <center>
  <p>
  The Docker LAMP stack is working.
  </p>
  </center>

  <center>
  <p>
  The configuration information can be found <a href="https://registry.hub.docker.com/
u/linode/lamp/">here</a> or <a href="https://www.linode.com/docs/websites/hosting-a-we
bsite">here</a>
  </p>
  </center>

  <center>
  <p>
  This index.html file is located in the "/var/www/example.com/public_html" directory.
  </p>
  </center>

  </body>
  </html>
```

Dal punto di vista dell'attaccante, all'interno del container *attaccante1* è stato montato un volume condiviso denominato *slowloris*, contenente le istruzioni per l'utilizzo dell'attacco Slowloris e uno script Python dedicato all'esecuzione dell'attacco. Attraverso il comando:

python3 slowloris.py 172.1.3.3 -p 80 -s 100

l'attaccante esegue lo script Python Slowloris, specificando come argomenti

l'indirizzo IP della macchina bersaglio (172.1.3.3), la porta del server web (80) e il numero di socket da utilizzare per l'attacco (30).



Nel presente scenario, il client è impossibilitato ad ottenere la pagina web richiesta mediante l'utilizzo del programma a riga di comando curl, poiché il server web non è in grado di gestire le richieste concorrenti, costringendo il client a rinunciare.

5.2 Mitigazione

Per difendersi da un attacco Slowloris possiamo utilizzare alcune regole iptables per la mitigazione degli attacchi al protocollo HTTP sulla porta 80.

```
iptables -I FORWARD 1 -p tcp --dport 80 -m connlimit --connlimit-
    above 20 -j DROP
iptables -I FORWARD 2 -p tcp --dport 80 -m limit --limit
    25/minute --limit-burst 100 -j ACCEPT
iptables -I FORWARD 3 -p tcp --dport 80 -f -j DROP
iptables -I FORWARD 4 -p tcp --dport 80 -m length --length
    0:500 -j ACCEPT
iptables -I FORWARD 5 -p tcp --dport 80 -m length --length
    501: -j DROP
```

La prima regola (FORWARD 1) limita il numero di connessioni simultanee sulla porta 80 a un massimo di 20 connessioni e respinge tutte le altre connessioni in entrata sulla stessa porta.

La seconda regola (FORWARD 2) consente un massimo di 25 richieste al minuto sulla porta 80, con un limite di raffica di 100 richieste. Le richieste che superano questa soglia vengono rifiutate.

La terza regola (FORWARD 3) rifiuta tutti i pacchetti che hanno il flag di frammentazione impostato. Avremmo potuto utilizzare anche solo questa regola per mitigare questo tipo di attacco ma in tal caso sarebbero stati scartati anche i pacchetti frammentati da parte di utenti non malevoli.

La quarta e la quinta regola (FORWARD 4 e FORWARD 5) consentono il forward dei pacchetti di lunghezza compresa tra 0 e 500 byte e fanno dropping dei pacchetti che superano tale limite sulla porta 80. Questo perché potrebbe essere aiutare a prevenire attacchi he tentano di inviare grandi quantità di dati al server.