UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

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

Network Security

Progettazione e Configurazione di una rete protetta da due Firewall e dotata di DMZ

Anno Accademico 2023/2024

Candidati

Alfonso D'avino *matr. M63001493*

Francesca Di Martino *matr. M63001478*

Indice

Capitolo 1

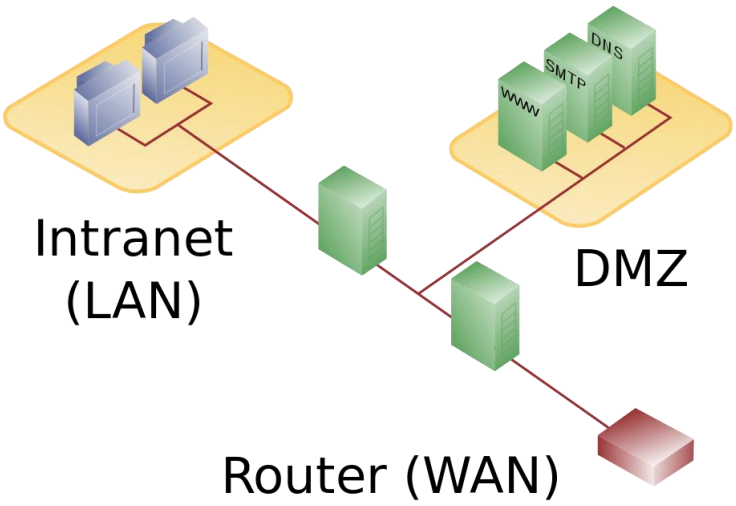
Introduzione

L'obiettivo di questo lavoro è la creazione di un testbed di rete in grado di simulare in modo realistico l'architettura di una rete aziendale, includendo una zona DMZ (DeMilitarized Zone) e una protezione firewall.

Un firewall è un sistema di sicurezza informatica progettato per controllare e filtrare il traffico di rete in ingresso e in uscita, basandosi su un insieme di regole predefinite per autorizzare o bloccare determinate comunicazioni.

La DMZ rappresenta una porzione della rete aziendale accessibile sia dall'esterno che dai dispositivi interni, pur rimanendo isolata dalla rete principale per motivi di sicurezza.

Per l'implementazione del firewall è stato utilizzato un host Ubuntu, sfruttando l'infrastruttura del kernel Linux nota come **Netfilter**, che permette il filtraggio dei pacchetti di rete. Il tool impiegato per gestire l'inserimento e la rimozione delle regole di filtraggio è **iptables**, che consente di configurare il comportamento del firewall in base alle esigenze di sicurezza della rete.



Si è scelta una configurazione con doppio firewall:

- Firewall esterno, il quale filtra i dati in ingresso alla DMZ.
- Firewall interno, il quale principalmente regola il traffico tra rete interna e DMZ.

Si è optato per un'architettura a due livelli anziché per una soluzione con un solo firewall, principalmente per evitare che quest'ultimo rappresenti un **single point of failure**. Inoltre, questa suddivisione delle funzionalità ha reso più semplice la gestione della rete e continuerà a facilitare eventuali interventi futuri.

L'intera simulazione è stata realizzata utilizzando **Docker**, un progetto open-source che ha permesso di riprodurre l'infrastruttura di rete in modo efficiente.

Per monitorare il traffico di rete e verificare il funzionamento delle regole di sicurezza, è stato impiegato lo strumento **ulogd2**, configurato direttamente sui firewall. Grazie a questo sistema di logging, è stato possibile registrare e analizzare tutti i pacchetti filtrati, fornendo così una dimostrazione pratica dell'efficacia delle policy di sicurezza implementate.

Capitolo 2

Descrizione della rete

La rete aziendale è progettata per garantire la separazione e la protezione dei dati tra le diverse aree, permettendo allo stesso tempo l'accesso sicuro ai servizi aziendali. È suddivisa in tre principali zone di rete: la rete esterna, la DMZ (Demilitarized Zone) e la rete interna. Ogni zona ha specifiche funzioni e livelli di sicurezza.

2.1 Rete esterna

La rete esterna è la zona di accesso per i dispositivi provenienti dall'esterno dell'azienda. Consente ai dispositivi remoti (come il client cliente1) di connettersi ai servizi pubblici aziendali ospitati nella DMZ. È protetta da un firewall che gestisce il traffico e limita l'accesso alla rete interna.

Componenti della Rete Esterna

- **Client Esterno (cliente1)**: Dispositivo connesso alla rete esterna, che rappresenta un utente remoto o un sistema esterno che necessita di interagire con i servizi aziendali. cliente1 è configurato con l'indirizzo IP statico 192.1.1.2 nella subnet 192.1.1.0/24. Questo dispositivo è il punto di ingresso per qualsiasi interazione da parte di client esterni.
- **Firewall Esterno (firewall1)**: È il punto di instradamento principale della rete esterna, con IP 192.1.1.5. Il firewall esterno funge da "guardiano" che regola il traffico tra la rete esterna e le zone interne. È configurato per consentire solo traffico specifico e sicuro verso i server pubblici nella DMZ (come il web server, DNS, e FTP server), mentre filtra il traffico sospetto o non autorizzato.
  Il firewall esterno, configurato anche come router, possiede due interfacce:
  - eth0 (192.1.1.5/24) connessa alla rete esterna.
  - eth1 (192.1.2.5/24) connessa alla DMZ.
- **Gateway**: Il firewall agisce anche come gateway predefinito per i dispositivi nella rete esterna. I dispositivi come cliente1 inviano le loro richieste verso il firewall, che poi le inoltra verso la destinazione appropriata.

2.2 DMZ

La **DMZ** è una zona di rete separata e protetta che ospita i server pubblici aziendali, i quali devono essere accessibili dall'esterno, ma devono essere isolati dalla rete interna per

motivi di sicurezza. In questa zona vengono collocati i server che forniscono servizi pubblici, come il **web server**, il **server FTP** e il **server DNS**. Questi server sono essenziali per il funzionamento dell'azienda, ma devono essere protetti da attacchi esterni e non devono avere accesso diretto alle risorse della rete interna, che contengono dati sensibili o sistemi critici.

La DMZ agisce come un "perimetro di sicurezza" tra la rete esterna (internet) e la rete interna, riducendo al minimo i rischi di compromissione. I firewall esterni e interni sono configurati per monitorare e filtrare il traffico in entrata e in uscita da questa zona, garantendo che solo il traffico autorizzato possa entrare nella DMZ e, successivamente, verso la rete interna, se necessario.

Componenti della DMZ

- **Web Server**: Il **web server** (ad esempio, webserver) è un server che ospita il sito web aziendale e altre applicazioni accessibili pubblicamente. Il suo IP è 192.1.2.2 e viene configurato per ascoltare sulla porta 80 (HTTP) .Il traffico in entrata sulla porta 80 viene filtrato dal firewall esterno e instradato verso il server web.
- **FTP Server**: Il **server FTP** (ad esempio, ftpserver) è utilizzato per trasferire file tra l'azienda e i clienti esterni. Il suo IP è 192.1.2.4, e ascolta sulle porta 20. Il traffico FTP è isolato nella DMZ, e solo i client autorizzati possono connettersi.
- **DNS Server**: Il **server DNS** (ad esempio, dnsser) è responsabile della risoluzione dei nomi di dominio per la rete aziendale e per le richieste provenienti dall'esterno. Il suo IP è 192.1.2.3, e gestisce il traffico in entrata sulla porta 53 (UDP). Questo server è fondamentale per la corretta gestione della rete, ma la sua comunicazione è limitata e protetta.

2.3 **Rete Interna**

La **rete interna** è la zona più protetta della rete aziendale, dove sono ospitati dispositivi e sistemi che gestiscono i dati sensibili e le risorse aziendali critiche. La rete interna è separata dalle altre zone della rete, come la **DMZ** e la **rete esterna**, per garantire che i sistemi aziendali vitali siano protetti da accessi non autorizzati. Questa separazione è essenziale per minimizzare il rischio di compromettere dati sensibili, applicazioni aziendali e altre risorse critiche.

In un contesto aziendale, la **rete interna** potrebbe ospitare file server, database, applicazioni aziendali, sistemi di gestione, e altre risorse vitali per il funzionamento quotidiano dell'organizzazione. La protezione di questi sistemi è una priorità assoluta, poiché qualsiasi vulnerabilità o attacco che comprometta la rete interna potrebbe avere gravi implicazioni per la sicurezza dell'intera azienda.

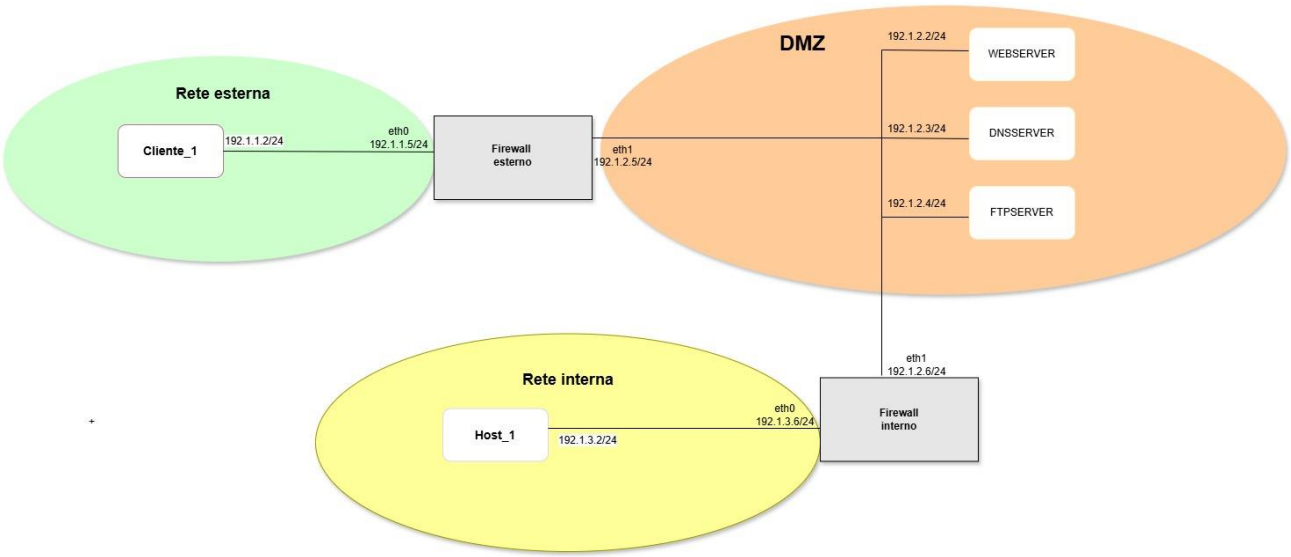
L'accesso alla rete interna è rigorosamente controllato tramite firewall interni che monitorano e filtrano il traffico proveniente dalla **DMZ**. Le comunicazioni tra la rete interna e le altre zone della rete sono limitate, e ogni tentativo di accesso deve essere autorizzato attraverso una configurazione specifica delle regole di firewall.

Componenti della Rete Interna

- **Host Aziendali (host1)**: Dispositivo che rappresenta un computer aziendale o una risorsa interna, configurato con l'indirizzo IP 192.1.3.2 nella subnet 192.1.3.0/24. Questo host può essere un server, un terminale, o una workstation utilizzata per l'elaborazione dei dati aziendali.
- **Firewall Interno (firewall2)**: Il **firewall interno** (firewall2), con IP 192.1.3.6, controlla il traffico che proviene dalla **DMZ** e si dirige verso la rete interna. È configurato per filtrare tutto il traffico e consentire solo quello autorizzato attraverso regole specifiche.

  Il firewall interno, anch'esso configurato come router, possiede due interfacce:

  - eth0 (192.1.3.6/24) connessa alla rete interna.
  - eth1 (192.1.2.6/24) connessa alla DMZ.



2.2 **Dockerfile**

Per la realizzazione della rete è stata scelta la piattaforma open source Docker. Docker consente lo sviluppo e l'esecuzione di applicazioni attraverso la tecnologia della virtualizzazione tramite container. A differenza delle macchine virtuali, Docker non utilizza un hypervisor, ma gestisce direttamente una serie di container che sono fondamentalmente processi UNIX isolati.
Docker è composto da un **Docker Engine**, che gestisce e mantiene attivi i container, e da un **Docker Client**, che consente al cliente di interagire con il Docker Engine.
La community di Docker offre una vasta gamma di immagini, disponibili sulla repository **Docker Hub**, da cui è possibile scaricare (pull) immagini già pronte messe a disposizione da altri utenti. Inoltre, è possibile caricare (push) le proprie immagini per renderle accessibili a chiunque.
Per il nostro progetto è stato adottato un approccio **ibrido**, in cui alcune immagini sono state scaricate direttamente da Docker Hub, mentre altre sono state create tramite Dockerfile, costruendo così delle immagini personalizzate in locale.

I **Dockerfile** sono file di configurazione che contengono una serie di comandi che definiscono l'ambiente di esecuzione di un container Docker. In altre parole, un Dockerfile è un modo per "costruire" un'immagine Docker, che successivamente può essere utilizzata per creare container. Questi file sono strumenti fondamentali per automatizzare la creazione e configurazione di ambienti di sviluppo e produzione in modo ripetibile e coerente.

Nel contesto della nostra rete aziendale, abbiamo creato due **Dockerfile**

- Firewall: Sono state sviluppate due immagini dedicate alla configurazione dei firewall1 e firewall2.
- Host: Sono state create anche due immagini per simulare gli host aziendali, ossia cliente1 e host1.

Questi Dockerfile sono stati **pushati su Docker Hub**, il che consente di scaricare facilmente le immagini associate a questi Dockerfile, semplificando la configurazione della rete aziendale e permettendo di replicare facilmente questi ambienti su altri sistemi.

2.2.1 Dockerfile host

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y \
bridge-utils \
net-tools \
iputils-ping \
nmap \
hping3 \
ftp

CMD echo "Ubuntu Host"
```

Tale Dockerfile permette la creazione di un'immagine per poter dar vita a degli host Ubuntu che permettono la simulazione di un host sulla rete interna e di un host sulla rete esterna. I tool nmap e hping3 saranno utilizzati in seguito per poter effettuare i test di funzionamento della configurazione realizzata.

2.2.3 **Dockerfile firewall**

Il Dockerfile crea un'immagine che si basa su Ubuntu e installa vari strumenti utili per la gestione della rete, tra cui iptables (per il filtraggio del traffico) e ulogd2 (per il logging del traffico di rete).

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y \
bridge-utils \
net-tools \
iptables \
ulogd2 \
nano

CMD service ulogd2 start && echo "Dockerfile Firewall. Iptables : " && iptables -L
```

1. **Immagine di base**: Parte da un'immagine di **Ubuntu** (l'ultima versione disponibile).
2. **Installazione di pacchetti**:
   Il comando RUN esegue l'aggiornamento dell'elenco dei pacchetti e installa i seguenti strumenti:
   - **bridge-utils**: Pacchetto che contiene strumenti per creare e gestire bridge di rete. Un bridge collega diverse interfacce di rete, permettendo di interagire tra diverse sottoreti.
   - **net-tools**: Fornisce strumenti di rete di base, come ifconfig, netstat, route che sono utili per la gestione e il monitoraggio delle configurazioni di rete.
   - **iptables**: Uno strumento di filtraggio dei pacchetti di rete. Permette di configurare il firewall per controllare il traffico di rete tra il container e altre macchine o reti.
   - **ulogd2**: Un demone di logging che registra i pacchetti di rete elaborati da iptables. È utile per monitorare e fare il log dei pacchetti che attraversano il firewall.
   - **nano**: Un editor di testo a riga di comando, che permette di modificare rapidamente i file di configurazione all'interno del container.
3. **Comando di avvio**:
   - **Avvia il servizio ulogd2** per registrare i pacchetti di rete.
   - **Visualizza le regole di iptables** per vedere la configurazione del firewall.

2.3 Configurazione

In questa sezione viene descritta la configurazione della rete tramite Docker, suddivisa in più sottoreti e connessa mediante firewall. Verranno illustrati i comandi utilizzati per creare le reti, avviare i container dei vari host e servizi, e configurare il routing per garantire la comunicazione tra le diverse componenti dell'infrastruttura.

Tutti i container saranno eseguiti in **modalità privilegiata**, il che significa che avranno accesso completo alle risorse di sistema, inclusi comandi avanzati e configurazioni di rete, come la gestione delle **iptables** per il firewall. Questo è necessario per configurare correttamente il routing tra le varie sottoreti.
Inoltre, per garantire che i container rimangano attivi dopo l'avvio, non basta eseguire semplicemente il comando docker run. Se non si specificano determinate opzioni, il container potrebbe avviarsi ed uscire immediatamente dopo l'esecuzione del comando iniziale.

Per evitare questo problema, si utilizza l'opzione **-td**, che permette di:

- **-t (pseudo-TTY)**: Assegna un terminale al container, utile per mantenere attiva l'esecuzione.
- **-d (detached mode)**: Esegue il container in background, evitando che si chiuda al termine del comando di avvio.

Inoltre, si esegue il comando bash all'interno del container. Questo assicura che il container resti attivo perché, senza un processo in esecuzione in foreground, Docker lo terminerebbe automaticamente. Eseguendo una shell interattiva (bash), si garantisce che il container rimanga operativo e pronto per ulteriori configurazioni.

Creazione rete interna e connessione host1:

```
docker network create --driver bridge --subnet=192.1.3.0/24 rete_interna
docker run --privileged --network=rete_interna --ip 192.1.3.2 -td --name=host1 alfonso991999/hostimg bash
```

In questa parte dello script è stata creata una sottorete denominata rete_interna, alla quale è stato assegnato l'indirizzo 192.1.3.0/24. Successivamente, è stato avviato un container denominato host1, basato sull'immagine alfonso991999/hostimg, precedentemente caricata su Docker Hub. Al container è stato assegnato l'indirizzo IP 192.1.3.2.

Creazione della rete esterna e connessione di cliente1:

```
docker network create --driver bridge --subnet=192.1.1.0/24 rete_esterna
docker run --privileged --network=rete_esterna --ip 192.1.1.2 -td --name=cliente1 alfonso991999/hostimg bash
```

Allo stesso modo della rete interna, è stata creata una sottorete denominata rete_esterna, con subnet 192.1.1.0/24. Successivamente, è stato avviato un container denominato cliente1, basato sull'immagine alfonso991999/hostimg, e gli è stato assegnato l'indirizzo IP 192.1.1.2.

Creazione della DMZ e connessione dei server:

```
docker network create --driver bridge --subnet=192.1.2.0/24 dmz
docker run --privileged --network=dmz --ip 192.1.2.2 -p80:80 -p 443:443 -td --name=webserver httpd
docker run --privileged --network=dmz --ip 192.1.2.4 -p20:20 -p21:21 -tdi --name=ftpserver stilliard/pure-ftpd bash
docker run  --privileged --network=dmz --ip 192.1.2.3 -p53:53/udp -tdi --name=dnsser cosmicq/docker-bind:latest bash
```

In seguito è stata creata la sottorete *dmz* con indirizzo 192.1.2.0/24, a cui sono stati annessi un server WEB, un server DNS e un server FTP scaricabili da Docker Hub. A questi verranno assegnati rispettivamente gli indirizzi IP 192.1.2.2, 192.1.2.3 e 192.1.2.4.

Creazione e Configurazione del Firewall Esterno

```
docker run  --privileged --sysctl net.ipv4.ip_forward=1 --network=rete_esterna --ip 192.1.1.5 -td --name=firewall1  bash
docker exec --privileged -t firewall1 service ulogd2 restart
docker network connect --ip 192.1.2.5 dmz firewall1
```

In questa parte dello script, viene avviato un container basato sull'immagine **firewall**, precedentemente creata e pushata su Docker Hub, in modalità privilegiata. Questo è necessario perché nel Dockerfile del firewall sono presenti configurazioni delle **iptables**, e per eseguire qualsiasi operazione su di esse sono richiesti permessi di root.
Un'altra configurazione fondamentale nel **firewall1** è l'uso dell'opzione **--sysctl net.ipv4.ip_forward=1**, che abilita l'inoltro IP (IP forwarding) nel container. Il kernel Linux disabilita di default questa funzionalità per evitare che i dispositivi possano agire come router senza autorizzazione. Abilitando questa opzione, il container diventa capace di instradare il traffico tra reti diverse, in particolare tra la rete esterna e la sottorete **DMZ** (zona demilitarizzata).
Inoltre, con l'istruzione **docker run**, il container **firewall1** viene connesso alla rete esterna, utilizzando l'indirizzo IP **192.1.1.5** su una delle sue interfacce. Questo permette al firewall di interagire con la rete esterna.
La seconda istruzione **docker exec --privileged -t firewall1 service ulogd2 restart** viene utilizzata per riavviare il servizio **ulogd2** all'interno del container. Questo servizio è essenziale per il **logging** delle attività del firewall, e senza il riavvio, il servizio non funzionerebbe correttamente una volta avviato il container.

Infine, l'ultima istruzione **docker network connect --ip 192.1.2.5 dmz firewall1** collega il container **firewall1** alla sottorete **DMZ**. In questo modo, il firewall è configurato per gestire il traffico tra la rete esterna e la DMZ, utilizzando l'indirizzo IP **192.1.2.5** sull'interfaccia connessa alla DMZ.

Creazione e Configurazione del Firewall Interno

```
docker run  --privileged --sysctl net.ipv4.ip_forward=1 --network=rete_interna --ip 192.1.3.6 -td --name=firewall2 alfonso991999/firewallimg bash
docker exec --privileged -t firewall2 service ulogd2 restart
docker network connect --ip 192.1.2.6 dmz firewall2
```

Viene creato il firewall interno (*firewall2*) e lo si connette sia alla *rete_interna* tramite l'interfaccia con indirizzo 192.1.3.6, sia alla *dmz* tramite l'interfaccia con indirizzo 192.1.2.6.

Configurazione delle Rotte nei Dispositivi Client e Server

Con le seguenti istruzioni invece, è stato designato il firewall come gateway di default per tutti gli host presenti nelle sottoreti.
Questo vuol dire che:

- Quando un dispositivo vuole comunicare con una destinazione che non appartiene alla sua rete locale (come internet o un'altra rete), invia il traffico al firewall.
- Il firewall si occupa di instradare il traffico verso la destinazione corretta.

```
docker exec cliente1  route add default   gw firewall1
docker exec host1     route add default   gw firewall2
docker exec webserver route add 192.1.1.2 gw firewall1
docker exec webserver route add 192.1.3.2 gw firewall2
docker exec dnsser    route add 192.1.1.2 gw firewall1
docker exec dnsser    route add 192.1.3.2 gw firewall2
docker exec ftpserver route add 192.1.1.2 gw firewall1
docker exec ftpserver route add 192.1.3.2 gw firewall2
```

In particolare:

- Client1 invia tutto il traffico verso 192.1.1.5 (firewall esterno).
- Host1 invia tutto il traffico verso 192.1.3.6 (firewall interno).
- Webserver, Dnsserver e Ftpserver sono configurati per inviare il traffico destinato a 192.1.1.2 (Client1) attraverso 192.1.1.5 (firewall esterno) e il traffico destinato a 192.1.3.2 (Host1) attraverso 192.1.3.6 (firewall interno).

Firewall con Iptables

Come già menzionato in precedenza, **iptables** è il tool che consente di definire regole per il filtraggio del traffico di rete, implementando un **Firewall Packet Filtering**.

Ogni regola creata deve appartenere a una **catena**, ovvero una sequenza di regole applicate ai pacchetti in transito. Le catene, a loro volta, sono organizzate all'interno di **tabelle**, tra cui le più rilevanti per il nostro caso sono la **filter** e la **nat**.

Nella tabella filter esistono 3 tipi di catene (quelli standard):

- **INPUT**, la quale lavora sui pacchetti in entrata al sistema.
- **OUTPUT**, in cui ci si riferisce ai pacchetti in uscita dal sistema.
- **FORWARD**, per i pacchetti che sono diretti ad un altro host della rete ma che per poterci arrivare devono passare dal nostro sistema.

Nel contesto di questo progetto d'esame, la catena di interesse è **FORWARD**, poiché il firewall deve gestire il traffico tra diverse reti.

Per quanto riguarda la **tabella nat**, che si occupa del **natting** e dell'**instradamento** dei pacchetti, le catene standard includono:

- **OUTPUT**
- **PREROUTING**, lavora sui pacchetti in entrata ma a questi pacchetti vengono già applicate delle regole ben definite prima di essere instradate nel sistema.
- **POSTROUTING**, lavora sui pacchetti in uscita dal sistema ma solamente dopo che è stato deciso il loro instradamento.

Sono state implementate nel progetto regole di **PREROUTING**, per garantire il corretto raggiungimento dei servizi della **DMZ**.

3.1 Ulogd2 - Logging delle pacchetti

Quando il firewall viene implementato all'interno di un **container Docker**, la regola **-j LOG** di **iptables** risulta disabilitata. Questa limitazione serve a prevenire possibili attacchi **DoS**, in cui un container potrebbe sovraccaricare la macchina ospitante generando un numero eccessivo di log.

Per ovviare a questo problema, si è adottato un metodo alternativo basato su **ulogd2**, un daemon in **userspace** che gestisce il logging dei pacchetti attraverso **netfilter/iptables**.

La sintassi delle nuove regole di log rimane praticamente invariata rispetto a quella originale, con l'unica differenza che il target **LOG** viene sostituito da **NFLOG**.

Oltre a questa modifica, è stato necessario intervenire sul file di configurazione **ulogd2.conf**, in cui sono stati decommentati i plugin necessari, configurate le impostazioni adeguate e specificato il file di destinazione per la registrazione dei pacchetti loggati.

Configurazione ulogd

Tutti i parametri di configurazione del modulo ulogd2 si trovano nel file di configurazione in */etc/ulogd.conf*.
Innanzitutto è stato definito il path del main logfile, in cui ulogd riporta errori, warnings e altre condizioni inaspettate in cui si trova il sistema:

```
# logfile for status messages
logfile="/var/log/ulog/ulogd.log"
```

In seguito sono stati de-commentati i plugin che verranno caricati prima dell'inizializzazione di ulogd, per permettere il corretto funzionamento del log:

```
1. ######################################################################
2. # PLUGIN OPTIONS
3. ######################################################################
4.
5. # We have to configure and load all the plugins we want to use
6.
7. # general rules:
8. #
9. # 0. don't specify any plugin for ulogd to load them all
10. # 1. load the plugins _first_ from the global section
11. # 2. options for each plugin in seperate section below
12.
13. plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inppkt_NFLOG.so"
14. #plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inppkt_ULOG.so"
15. #plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inppkt_UNIXSOCK.so"
16. plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inpflow_NFCT.so"
17. plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IFINDEX.so"
18. plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2STR.so"
19. plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2BIN.so"
20. #plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2HBIN.so"
21. plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_PRINTPKT.so"
```

```
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_HWHDR.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_PRINTFLOW.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_MARK.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_LOGEMU.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_SYSLOG.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_XML.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_SQLITE3.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_GPRINT.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_NACCT.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_PCAP.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_PGSQL.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_MYSQL.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_DBI.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_raw2packet_BASE.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inpflow_NFACCT.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_GRAPHITE.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_JSON.so"
```

Come si può osservare, esistono diverse categorie di plugin. Questi possono essere suddivisi in tre principali gruppi:

- **Input Plugins**,
- **Filter Plugins**, i quali realizzano un parsing del pacchetto in ingresso.
- **Output Plugins**, che ricevono l'informazione interpretata dai plugin precedenti e la scrivono sul file di output predefinito.

Tra di essi vale la pena approfondire i seguenti:

- ***ulogd_raw2packet_BASE.so***, forse quello più importante, il quale permette di interpretare gli header di svariate tipologie di pacchetto.
- ***ulogd_output_LOGEMU.so***, plugin di output il quale emula il target standard LOG e permette il salvataggio dei pacchetti in un file.

E' stata lasciata la configurazione di default:

```
# this is a stack for logging packet send by system via LOGEMU
stack=log1:NFLOG,base1:BASE,ifi1:IFINDEX,ip2str1:IP2STR,print1:PRINTPKT,emu1:LOGEMU

```

Infine è stato definito il file su cui verranno poi loggati i pacchetti, ed è stata settata la variabile sync = 1, in modo da trascrivere i log in maniera sincrona:

```
[emu1]
file="/var/log/ulog/syslogemu.log"
```

```
3. sync=1
4.
```

3.2 Regole

Per configurare correttamente le regole del firewall, è necessario partire da una base pulita, rimuovendo tutte le regole e le catene preesistenti. Questo processo garantisce che il firewall parta con una configurazione vuota, senza interferenze da regole passate che potrebbero influire sulla sicurezza. Inizialmente, iptables ha una policy di tipo ACCEPT, il che significa che tutto il traffico è consentito di default, senza essere bloccato. Dopo aver eseguito questo reset, si possono impostare nuove policy che definiscono come il traffico di rete deve essere gestito, stabilendo quali pacchetti devono essere accettati o rifiutati.

Le operazioni che seguono riguardano:

Eliminazione delle Regole Preesistenti:

- Vengono **eliminate tutte le regole** preesistenti dalle catene di **iptables** (filter e nat), e vengono rimosse anche le catene non standard vuote. Questo passo è necessario per ripartire da una configurazione pulita.

```
1.
2. #               Cancellazione delle regole presenti nelle chains                    #
3. docker exec --privileged -t firewall1 iptables -F
4. docker exec --privileged -t firewall2 iptables -F
5. docker exec --privileged -t firewall1 iptables -F -t nat
6. docker exec --privileged -t firewall2 iptables -F -t nat
7.
8. #               Eliminazione delle chains non standard vuote                        #
9. docker exec --privileged -t firewall1 iptables -X
10. docker exec --privileged -t firewall2 iptables -X
11.
```

Infatti, il comando di **iptables** con l'opzione **-X** consente di rimuovere una catena vuota (non standard), e, se necessario, è possibile specificare il nome della catena da eliminare. Allo stesso modo, l'opzione **-F** permette di cancellare le catene standard.

Impostazione delle Nuove Policy:

- Dopo aver cancellato le vecchie regole, vengono impostate nuove **policy di default**. Queste policy fanno sì che il firewall **rifiuti** (DROP) tutto il traffico in

ingresso (INPUT), in uscita (OUTPUT) e in transito tra le interfacce (FORWARD), a meno che non vengano definite regole specifiche per consentirlo.

```
1. docker exec --privileged -t firewall1 iptables -P INPUT DROP
2. docker exec --privileged -t firewall1 iptables -P OUTPUT DROP
3. docker exec --privileged -t firewall1 iptables -P FORWARD DROP
4. #stessa cosa per firewall2
5. docker exec --privileged -t firewall2 iptables -P INPUT DROP
6. docker exec --privileged -t firewall2 iptables -P OUTPUT DROP
7. docker exec --privileged -t firewall2 iptables -P FORWARD DROP
```

L'opzione *-P* consente di inizializzare le nuove regole per la catena.
Dopo aver standardizzato le regole di base, si è proceduto ad esaminare i possibili attacchi che potrebbero compromettere la rete. Una prima misura di protezione è stata quella di eseguire il **DROP** sui pacchetti frammentati del protocollo **HTTP**. Questa decisione è stata presa per prevenire attacchi come il **"Tiny Fragment Attack"**, che sfrutta pacchetti frammentati di dimensioni ridotte per causare problemi durante il processo di riassemblaggio, con conseguenti attacchi **DoS**. Per mitigare tale rischio, è stata implementata una regola specifica:

```
1. docker exec --privileged -t firewall1 iptables -A FORWARD -p ip -f -j DROP
```

3.2.1 IP Spoofing

L'**IP spoofing** è una tecnica di attacco in cui viene falsificato l'indirizzo IP del mittente di un pacchetto. Questo tipo di attacco è possibile perché, durante il processo di routing, il protocollo di trasporto si basa esclusivamente sull'indirizzo di destinazione, senza effettuare alcun controllo sull'indirizzo di origine.
Per prevenire che un attaccante esterno possa mascherarsi da host della rete interna e acquisire privilegi non autorizzati, è stata introdotta una regola specifica. Tale regola scarta tutti i pacchetti in arrivo sull'interfaccia **eth0** con un indirizzo IP di origine appartenente alla rete interna **192.1.3.0/24**.:

```
1. docker exec --privileged -t firewall1 iptables -A FORWARD -s 192.1.3.0/24 -i eth0 -j DROP
```

É possibile notare che questa regola, a differenza delle altre, interessa solo il Firewall 1.

3.2.2 SYN Flood Attack

Il **SYN Flood** è un attacco **DoS** che sfrutta il protocollo **TCP**. In questo tipo di attacco, l'aggressore invia una richiesta di connessione alla vittima con il flag **SYN** attivato, richiedendo l'apertura di una connessione. La vittima risponde con i flag **ACK** e **SYN** alti, ma l'attaccante non invia mai il pacchetto di conferma con il flag **ACK**, mantenendo la connessione in stato pendente e consumando risorse della vittima. Inoltre, l'attacco può includere anche **IP spoofing**, facendo sembrare che le richieste provengano da numerosi indirizzi IP differenti.

Per mitigare questo tipo di attacco, è possibile implementare una catena che limiti il traffico dopo un determinato numero di pacchetti con il flag **SYN** attivato. Di seguito viene descritta la creazione di questa catena.
La catena è denominata **SYN_FLOOD** e viene eseguita solo se il pacchetto in ingresso appartiene al protocollo **TCP** e ha il flag **SYN** attivo:

```
1. docker exec --privileged -t firewall1 iptables -N SYN_FLOOD
2. docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --syn -j SYN_FLOOD
```

Dopodiché, il pacchetto viene controllato e fatto "passare" e se conforme viene eseguita la regola:

```
1. docker exec --privileged -t firewall1 iptables -A SYN_FLOOD -m limit --limit 1/s -j RETURN
```

Altrimenti viene effettuato il *DROP* di tale pacchetto:

```
1. docker exec --privileged -t firewall1 iptables -A SYN_FLOOD -j DROP
```

La regola che utilizza l'opzione **limit** di iptables serve a limitare il numero di pacchetti che possono attraversare una certa catena nel firewall, per proteggere contro eventuali attacchi DoS (Denial of Service). La regola che è stata configurata impone un limite di **1 pacchetto al secondo** utilizzando l'opzione **--limit 1/s**.
Tuttavia, in modo implicito, iptables applica anche una logica di **"burst"** (picco), che consente un numero maggiore di pacchetti in ingresso prima che il limite venga attuato. Di default, iptables permette fino a **5 pacchetti** in un breve periodo di tempo (questo è il valore predefinito del burst) prima che inizi a limitare il flusso a **1 pacchetto per secondo**.
In altre parole, anche se la regola impone un limite di **1 pacchetto al secondo**, i primi **5 pacchetti** possono essere accettati senza problemi (grazie al burst), consentendo un piccolo picco di traffico senza che venga immediatamente applicato il limite. Dopo questi

primi 5 pacchetti, iptables inizia a applicare il limite, accettando solo **1 pacchetto al secondo** fino a quando non viene "ricaricato" il burst.

3.2.3 Ping of Death

Il **Ping of Death** è un tipo di attacco DoS in cui l'attaccante invia un pacchetto ICMP con una dimensione eccessiva verso la macchina di destinazione. Questo pacchetto viene frammentato lungo il percorso, ma quando giunge al sistema destinatario, il suo reassemblaggio porta a un overflow del buffer, in quanto la dimensione del pacchetto supera quella che il sistema è in grado di gestire. Questo overflow può compromettere il sistema, causando un **Denial of Service (DoS)**.

Per proteggersi da questo tipo di attacco, una soluzione semplice ed efficace consiste nel **DROP** dei pacchetti frammentati. Tuttavia, oltre a questa misura preventiva, l'attacco può essere contrastato con una strategia simile a quella impiegata per il **SYN Flood**. In particolare, è possibile creare una catena di filtraggio che blocchi i pacchetti ICMP sospetti o frammentati, impedendo così che il sistema possa essere vulnerabile a questo tipo di attacco:

```
1. docker exec --privileged -t firewall1 iptables -N PING_OF_DEATH
2. docker exec --privileged -t firewall1 iptables -A FORWARD -p icmp -j PING_OF_DEATH
3. # Accetto tutte le richieste se rispettano i limiti prefissati
4. docker exec --privileged -t firewall1 iptables -A PING_OF_DEATH -p icmp --icmp-type echo-request -m
limit --limit 1/s -j RETURN
5. # Se non ho un match con la regola di sopra il pacchetto va necessariamente scartato
6. docker exec --privileged -t firewall1 iptables -A PING_OF_DEATH -p icmp --icmp-type echo-request -j
DROP
```

Se non avviene il match con la regola, dunque i limiti non sono rispettati, il pacchetto viene droppato.

3.2.4 State

Un ulteriore criterio di valutazione dei pacchetti può essere riscontrato tramite l'utilizzo dell'estensione *state*, il quale permette di discriminare tra diversi stati in cui si trova la connessione:

- **NEW** in cui il pacchetto crea una nuova connessione;
- **ESTABLISHED** in cui il pacchetto appartiene a una connessione già esistente;
- **RELATED** in cui il pacchetto è collegato a una connessione esistente, ma non fa parte di essa (errori di pacchetti ICMP per esempio);

- **INVALID** in cui il pacchetto non può essere identificato per una qualunque ragione e dev'essere droppato.

Le regole impostate per i firewall prevedono il *DROP* dei pacchetti che risultano invalidi in input, forward e output rispettivamente:

```
1. docker exec --privileged -t firewall1 iptables -A INPUT -m state --state INVALID -j DROP
2. docker exec --privileged -t firewall1 iptables -A FORWARD -m state --state INVALID -j DROP
3. docker exec --privileged -t firewall1 iptables -A OUTPUT -m state --state INVALID -j DROP
```

Viene effettuato il *DROP* di tutti i pacchetti provenienti dalla rete esterna con destinazione rete interna per evitare che un cliente possa comunicare con un utente della rete:

```
1. docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state
NEW,ESTABLISHED,RELATED -j DROP
```

Viceversa sono sempre accettati i pacchetti che provengono dalla rete esterna (*eth0*) e diretti verso la DMZ (*eth1*):

```
1. docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state
NEW,ESTABLISHED,RELATED -j ACCEPT
2. docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth1 -o eth0 -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

E i pacchetti provenienti dalla rete interna (*eth2*) e diretti verso la DMZ (*eth1*):

```
1. docker exec --privileged -t firewall2 iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state
NEW,ESTABLISHED,RELATED -j ACCEPT
2. docker exec --privileged -t firewall2 iptables -t filter -A FORWARD -i eth1 -o eth0 -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

3.2.5 **UDP**

UDP Flood Attack

Un **UDP flood** è una tipologia di attacco **DoS (Denial of Service)** che sfrutta il protocollo **UDP** per inondare il sistema vittima con una grande quantità di pacchetti UDP. A differenza di **TCP**, che è un protocollo **connection-oriented** e richiede un processo di negoziazione per stabilire una connessione, **UDP** è un protocollo **connectionless**. Questo significa che i datagrammi possono essere inviati senza stabilire prima una connessione, rendendolo vulnerabile a questi attacchi.

Per contrastare un **UDP flood**, è possibile configurare il firewall per limitare la quantità di pacchetti UDP che la vittima può ricevere in un determinato intervallo di tempo, impedendo così che l'attacco sovraccarichi la rete o il sistema:

```
1. docker exec --privileged -t firewall1 iptables -N UDP_FLOOD
2. docker exec --privileged -t firewall1 iptables -A FORWARD -p udp -j UDP_FLOOD
3. docker exec --privileged -t firewall1 iptables -A UDP_FLOOD -p udp -m limit --limit 1/s -j RETURN docker
exec --privileged -t firewall1 iptables -A UDP_FLOOD -j DROP
```

Tale catena è paragonabile alla catena che permette di limitare l'attacco SYN flood visto in precedenza.

Traffico UDP

Per l'utilizzo del server DNS è abilitato il traffico verso la porta 53 dello già citato server con ip *192.1.2.3*, con annessa regola che permette la risposta da parte del server:

```
1. docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth0 -o eth1 -p udp -d 192.1.2.3
--dport 53 -j ACCEPT
2. docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth1 -o eth0 -p udp -j ACCEPT
```

Viceversa, tutto il traffico UDP verso le altre destinazioni che non siano il server DNS viene bloccato:

```
1. docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth0 -o eth1 -p udp -j DROP
```

Capitolo 4

Test regole security

Le regole e rispettive catene trattate nel precedente capitolo, saranno ora testate con il fine di verificarne il corretto comportamento durante la simulazione della rete.
Il daemon *ulogd2* permette di migliorare la comprensione dell'output in seguito al verificarsi delle regole. Infatti, ogni regola sarà interposta tra due operazioni di logging, le quali permetteranno di comprendere se il matching con tale regola è avvenuto o meno.
Negli screenshots relativi alle seguenti simulazioni non si è riusciti (per motivi di spazio) a riportare la totalità delle informazioni che è possibile ricavare dal log. Oltre ad ip sorgente e destinazione sono presenti la gran parte dei flag e dei campi, con il rispettivo valore, che si trovano negli header dei protocolli specificati.

HPING3

Per poter simulare gli attacchi all'interno della rete è stato utilizzato il tool hping3. Esso è un generatore e analizzatore di pacchetti per il protocollo TCP/IP. Il software si basa sullo stesso concetto del comando Unix *ping*, ma fa uso anche di protocolli differenti dall'ICMP (permette l'invio di segmenti TCP e datagrammi UDP), e permette di gestire la costruzione a piacere dei pacchetti IP. Tale tool è uno degli strumenti più utilizzati per le verifiche di sicurezza e il testing di firewalls e reti.

4.1 Test IP Spoofing

Si testi il meccanismo di protezione contro l'attacco IP spoofing:

```
# Protezione Ip Spoofing
# Tutti i pacchetti che provengono dall'esterno e hanno source address interno vengono scartati
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log pre-regola:"
docker exec --privileged -t firewall1 iptables -A FORWARD -s 192.1.3.0/24  -i eth0 -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log post-regola:"
```

Secondo tale regola, nel momento in cui un client esterno cerca di inviare un pacchetto verso la DMZ, il cui indirizzo IP sorgente appartiene alla rete interna, questo viene scartato.

```
hping3 -S -p 80 --spoof 192.1.3.2 192.1.2.2
```

L'attacco prevede l'invio di pacchetti IP con queste caratteristiche:

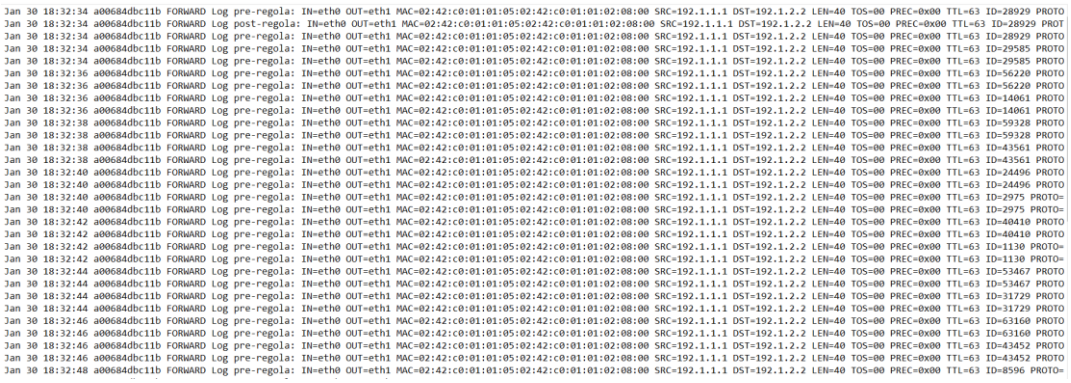
-S: Imposta il flag **SYN** nel pacchetto. Questo indica che il pacchetto è parte di una richiesta di connessione, come nel caso del **Three-Way Handshake** di TCP.

-p 80: Specifica che il pacchetto viene inviato alla porta **80**, che è la porta predefinita per il traffico HTTP.

--spoof 192.1.3.2: Falsifica l'indirizzo IP sorgente, impostando il mittente come **192.1.3.2** anziché l'indirizzo IP reale da cui proviene l'attacco.

192.1.2.2: Indica l'indirizzo IP di destinazione (web server), ovvero la vittima che riceverà il pacchetto.

Il risultato è riportato nei log presenti all'interno del firewall esterno:



Com'è possibile notare dalla figura, i pacchetti matchano la regola definita in precedenza: i log presenti nel file sono solo quelli che antecedono la regola a dimostrazione che il drop di tali pacchetti è avvenuto con successo.

4.2 Test Syn Flood Attack

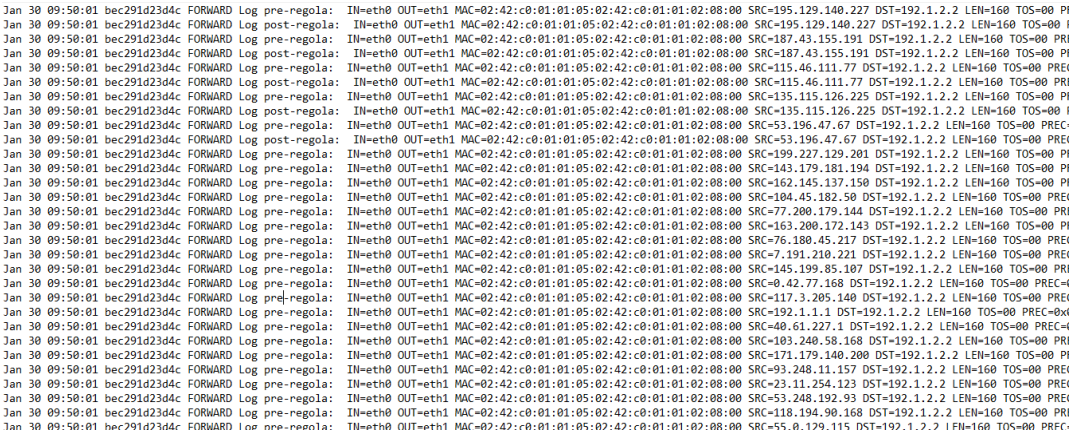
Seguendo il procedimento descritto nell'introduzione di questo capitolo è stato verificato il funzionamento delle regole implementate per evitare eventuali *Syn Flood Attack* nella rete.

```
# Protezione Syn Flood Attack
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log pre-regola:"
docker exec --privileged -t firewall1 iptables -N SYN_FLOOD
# Eseguo le regole della catena SYN_FLOOD se il pacchetto in ingresso è tcp e ha il flag syn = 1
docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --syn -j SYN_FLOOD
docker exec --privileged -t firewall1 iptables -A SYN_FLOOD -m limit --limit 1/s -j RETURN
# Se non ha un match con la regola precedente il pacchetto viene scartato
docker exec --privileged -t firewall1 iptables -A SYN_FLOOD -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log post-regola:"
```

Secondo la logica di tali regole, nel momento in cui il rate di pacchetti TCP in ingresso supera quello *limite* da esse previsto, i pacchetti in questione saranno droppati dal firewall. Partendo da questi presupposti, è stato avviato l'attacco dal container rappresentante il cliente della rete esterna, nei confronti del webserver, attraverso l'esecuzione del seguente comando:

```
1. hping3 -c 15000 -d 120 -S -w 64 -p 80 --flood --rand-source 192.1.2.2
```

In tal caso hping3 è stato utilizzato in modalità default (TCP) e tramite il tag *–flood* il rate di pacchetti inoltrati è impostato al massimo che la macchina riesce a raggiungere. La dimensione di ogni pacchetto è di 120 bytes e la finestra di trasmissione definita è pari a 64 bytes. Inoltre con il tag *-p 80* è stata specificata la porta destinazione mentre con *-S* è stato settato il flag SYN ad 1. Infine *–rand-source* ha permesso di attaccare il webserver da sorgenti random, "mascherando" il reale ip dell'attaccante.
Per verificare il corretto funzionamento delle regole, è stato controllato l'output di ulogd2 nel file *syslogemu.log*:



Come ci si aspettava, i pacchetti iniziali vengono fatti passare (viene stampato il log post-regola, il che significa che la regola di drop non matcha) dato che il rate con cui vengono inoltrati è nei limiti. Dopodichè iniziamo a notare la presenza dei soli *log pre-regola*, indice del dropping dei pacchetti, nel momento in cui il rate con cui arrivano è superiore rispetto a quello stabilito.

4.3 Test Ping of Death Attack

Anche questo attacco è stato controllato limitando il rate dei possibili pacchetti *icmp* in ingresso.

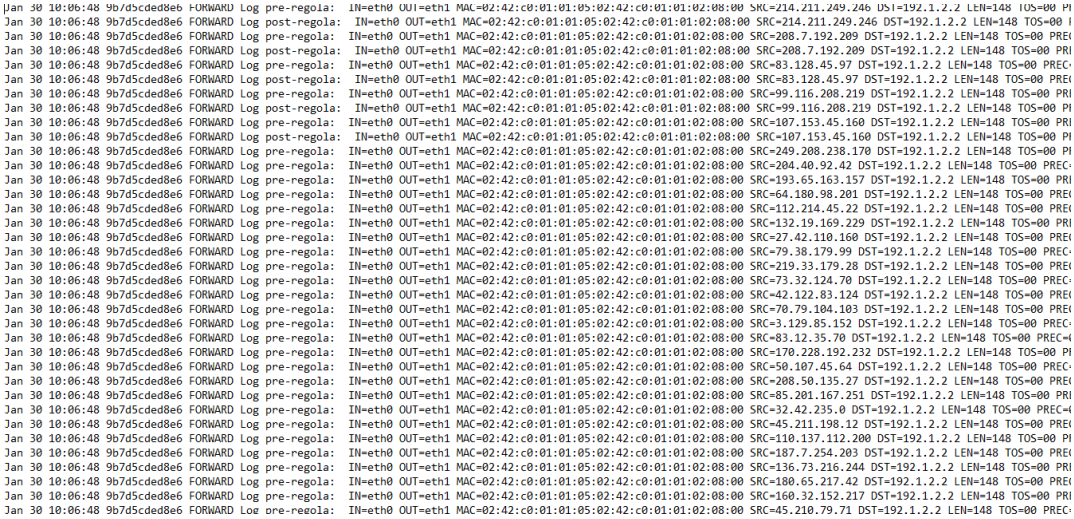
```
# Protezione Ping of Death Attack
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log pre-regola: "
docker exec --privileged -t firewall1 iptables -N PING_OF_DEATH
docker exec --privileged -t firewall1 iptables -A FORWARD -p icmp -j PING_OF_DEATH
docker exec --privileged -t firewall1 iptables -A PING_OF_DEATH -p icmp --icmp-type echo-request -m limit --limit 1/s -j RETURN
docker exec --privileged -t firewall1 iptables -A PING_OF_DEATH -p icmp --icmp-type echo-request -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log post-regola: "
```

Dal container del cliente esterno è stato avviato il seguente comando verso il webserver (192.1.2.2):

```
1. hping3 --icmp -c 15000 -d 120 -p 80 --flood --rand-source 192.1.2.2
```

La sintassi è molto simile a quella per l'attacco Syn Flood. In questo caso è stato necessario specificare il tag *–icmp* per l'inoltro della corretta tipologia di pacchetti.

Anche l'output nel file di log è molto simile a quello del precedente attacco, a dimostrazione del corretto funzionamento delle regole.



4.4 Test UDP Flood Attack

L'attacco *UDP Flood* è analogo a quello *SYN Flood*, differiscono solo per la tipologia di pacchetti inviati.

```
# Protezione UDP-flood
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log pre-regola:"
docker exec --privileged -t firewall1 iptables -N UDP_FLOOD
docker exec --privileged -t firewall1 iptables -A FORWARD -p udp -j UDP_FLOOD
docker exec --privileged -t firewall1 iptables -A UDP_FLOOD -p udp -m limit --limit 1/s -j RETURN
docker exec --privileged -t firewall1 iptables -A UDP_FLOOD -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -j NFLOG --nflog-prefix="FORWARD Log post-regola:"
```

Tale regola prevede che i datagrammi UDP vengano droppati al superamento di una determinata soglia impostata dal valore dell'opzione *–limit* per evitare un attacco DoS.

```
1. hping3 --udp -c 15000 -d 120 -p 53 --flood --rand-source 192.1.2.2
```

L'attacco prevede l'invio di pacchetti di dimensione 120 bytes, presso la porta 53 (*-p*) del web server all'indirizzo IP 192.1.2.2. Inoltre tali pacchetti saranno inviati in flooding (*–flood*) verso la destinazione con indirizzo IP sorgente generato randomicamente (*rand-source*), in tal modo non sarà possibile risalire alla fonte dell'attacco.

```
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=247.161.49.212 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log post-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=247.161.49.212 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=66.110.63.40 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log post-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=66.110.63.40 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=214.249.81.6 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log post-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=214.249.81.6 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=176.84.23.78 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log post-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=176.84.23.78 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=33.90.30.247 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log post-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=33.90.30.247 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=30.135.247.169 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=33.83.149.76 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=74.95.128.15 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=6.197.189.35 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=214.198.189.99 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=193.5.101.215 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=66.225.177.32 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=177.191.162.146 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=143.212.135.217 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=81.40.166.23 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=17.204.189.211 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=109.95.165.72 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=134.85.32.55 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=165.247.164.145 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=111.236.28.249 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=183.193.128.248 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=51.205.202.214 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=59.215.247.133 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=222.243.140.9 DST=192.1.2.3 LEN=148 TOS=00 PRE
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=132.204.252.162 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=131.51.21.23 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=90.255.149.224 DST=192.1.2.3 LEN=148 TOS=00 PRI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=212.6.30.132 DST=192.1.2.3 LEN=148 TOS=00 PREC
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=146.204.191.204 DST=192.1.2.3 LEN=148 TOS=00 PI
Jan 30 10:00:19 87ede4f671e5 FORWARD Log pre-regola:  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=205.38.243.115 DST=192.1.2.3 LEN=148 TOS=00 PRI
```

Com'è verificabile dall'immagine, inizialmente i pacchetti non verranno scartati dalla regola poiché il rate con cui sono inoltrati è inferiore rispetto a quello stabilito dall'opzione di *limit*. In seguito al superamento del limite, i pacchetti verranno droppati per far si che la destinazione possa recuperare dalla situazione di *burst* verificatasi. Tale situazione è riscontrabile dalla presenza dei soli log definiti dalle *"pre-regole"*.