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 aziendale protetta da due Firewall e dotata di DMZ

Anno Accademico 2021/2022

Professore
Simon Pietro Romano

Candidati
Emma Melluso *matr. M63001176*
Carmine Pio D'Antuono *matr. M63001224*
Pasquale Gaviglia *matr. M63001188*

Indice

1 Introduzione **2**

2 Descrizione della rete **4**

2.1 Progettazione 4

2.2 Dockerfile 5

2.2.1 Dockerfile host 5

2.2.2 Dockerfile server FTP 6

2.2.3 Dockerfile firewall 6

2.3 Configurazione 7

2.4 Docker-Compose 8

3 Firewall con Iptables **11**

3.1 Ulogd2 - Logging delle pacchetti 11

3.2 Regole 13

3.2.1 IP Spoofing 15

3.2.2 SYN Flood Attack 15

3.2.3 Ping of Death 16

3.2.4 State 17

3.2.5 UDP 18

4 Test regole security **19**

4.1 Test IP Spoofing 19

4.2 Test Syn Flood Attack 20

4.3 Test Ping of Death Attack 21

4.4 Test UDP Flood Attack 22

Capitolo 1

Introduzione

L'obiettivo di questo elaborato è la realizzazione (in ambiente controllato) di un testbed di rete il quale emula, in maniera realistica, la configurazione di una rete aziendale protetta da *firewall* e dotata di *DMZ - DeMilitarized Zone*.
Un firewall è un dispositivo per la sicurezza della rete che permette di monitorare il traffico in entrata e in uscita utilizzando una serie predefinita di regole di sicurezza per consentire o bloccare gli eventi.
La DMZ è una zona della rete aziendale "esposta" e che offre funzionalità sia verso l'esterno, ma anche a tutti quei dispositivi interni i quali sono completamente isolati dalla rete esterna.
I firewall sono stati implementati a partire da un semplice host *Ubuntu*, sfruttando l'infrastruttura contenuta nel kernel Linux: *netfilter*, che consente il filtraggio dei pacchetti in ingresso/uscita dalla macchina. In particolare, il tool utilizzato per l'interazione con il kernel, il quale permette l'inserimento e la rimozione di regole dalla sua tabella di filtraggio, è **iptables**.

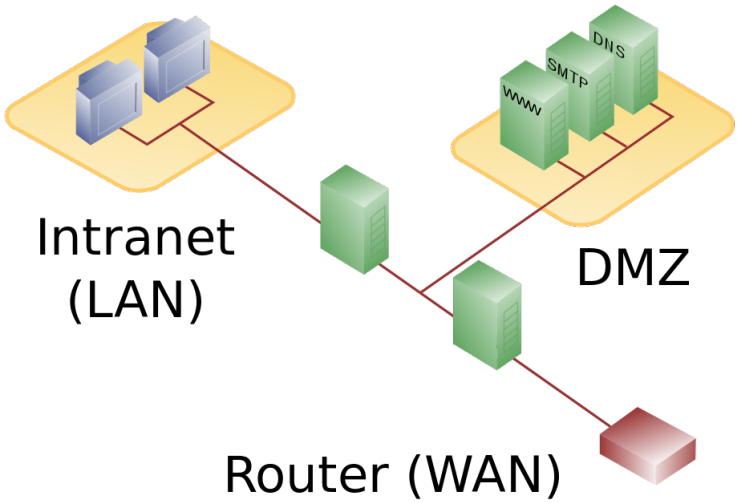

Figura 1.1: Struttura ad alto livello della rete

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 preferita un'architettura su due livelli, rispetto a quella con un unico firewall, innanzitutto per evitare situazioni in cui quest'ultimo possa fungere da **single point of failure**. Inoltre questa "suddivisione dei compiti" ha permesso e consentirà anche in futuro, una gestione più agevole della rete stessa.
Il tutto è stato simulato sfruttando il progetto open-source Docker.
Inoltre, mediante l'impiego dello strumento **ulogd2**, opportunamente configurato nei firewall, è stato possibile inserire delle regole per loggare tutti i pacchetti che successivamente verranno filtrati. In questo modo è stato possibile dimostrare praticamente il corretto funzionamento delle regole, in particolare di quelle associate alla *sicurezza* della rete.

Capitolo 2

Descrizione della rete

La rete che abbiamo deciso di realizzare possiede l'aspetto di una tipica infrastruttura aziendale, composta da un firewall interno e da uno esterno. Il firewall esterno ha il compito di filtrare i pacchetti che provengono dalla rete esterna e sono diretti alla DMZ e viceversa. Analogamente il firewall interno filtra i pacchetti provenienti dalla rete interna e diretti alla DMZ e viceversa.

2.1 Progettazione

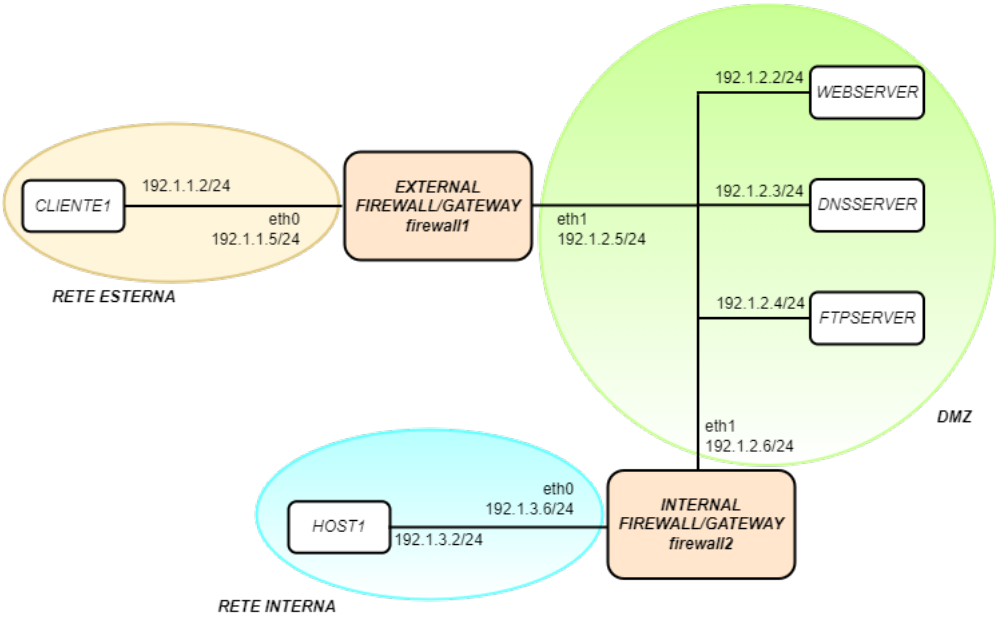

Figura 2.1: Struttura della rete

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.

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 realizzare la rete si è deciso di utilizzare la piattaforma open source **Docker**. Essa permette lo sviluppo e l'esecuzione di applicazioni sfruttando la tecnologia della virtualizzazione dei container. A differenza delle macchine virtuali non fa uso di un hypervisor, bensì gestisce una serie di **container** (processi UNIX).
Docker è composto da un **Docker Engine** che permette di mantenere vivi i container e da un **Docker Client** che permette al client di comunicare con il Docker Engine.
La community di Docker mette a disposizione un numero elevato di immagini, presenti sulla repository **Docker Hub**, dalla quale è possibile effettuare dei pull per sfruttare immagini offerte da altri client oppure è possibile fare delle push per pubblicare le proprie immagini.
Per questo progetto si è deciso di seguire un approccio "ibrido", ovvero sono state recuperate delle immagini messe a disposizione su Docker Hub e allo stesso tempo sono stati creati dei Dockerfile con l'obiettivo di costruire delle immagini in locale.

2.2.1 Dockerfile host

```
FROM ubuntu:latest

LABEL maintainer emme "emmamelluso@gmail.com"

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.

2.2.2 Dockerfile server FTP

```
FROM emalderson/ftplab:ftp_1.0

LABEL maintainer emme "emmamelluso@libero.it"

RUN apt-get update && apt-get install -y \
net-tools

CMD echo "Dockerfile FTP server"
```

In questo Dockerfile è stata semplicemente caricata l'immagine di un server FTP presente su Docker Hub.

2.2.3 Dockerfile firewall

```
FROM ubuntu:latest

LABEL maintainer emme "emmamelluso@libero.it"

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

CMD echo "Dockerfile Firewall. Iptables : " && \
iptables -L
```

Per il firewall il procedimento adottato è leggermente più complesso. Il primo passo è stato quello di creare, mediante il Dockerfile riportato in alto, l'immagine locale di un firewall su cui sono stati installati tutti i tool di interesse, in particolar modo **ulogd2**, il quale effettua il log dei pacchetti passanti per il firewall stesso.
Il tool *ulogd2* deve essere configurato in modo opportuno per poter usufruire dei suoi servizi, tali modifiche, però, vengono perse nel momento in cui il container viene riavviato. Si è pensato allora di effettuare dapprima un commit del container per salvare in modo permanente le modifiche e successivamente pubblicare l'immagine su Docker.

```
docker login -u username                                    //Accedo a Docker Hub
docker commit <container_id> emmame/firewall_ulogd2:latest
docker push emmame/firewall_ulogd2:latest
```

Il *container_id* è associato all'immagine locale del firewall su cui sono state effettuate le modifiche, *emmame* identifica l'username, *firewall_ulogd2* rappresenta il nome della repository, infine, *latest* è il tag dell'immagine.

2.3 Configurazione

In questa sezione si presenta parte dello script utilizzato per creare e configurare automaticamente la rete, e le indicazioni su come bisogna utilizzarlo.
Tutti i container utilizzati saranno eseguiti in modalità privilegiata (ovvero in root). Inoltre, per poter connettere le diverse sottoreti tra di loro, è necessario che i container siano *up*. Ciò non si verifica semplicemente dando l'istruzione di **run**, ma per far in modo che restino *up* anche dopo che sono stati eseguiti è necessario specificare l'opzione **-td** e soprattutto far eseguire il comando **bash**.

```
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 hostubuntu bash
```

In questa parte di script è stata creata una sottorete denominata *rete_interna* a cui è stato assegnato l'indirizzo 192.1.3.0/24. È stato mandato in esecuzione un container (*host1*) a partire dall'immagine costruita del Dockerfile per gli host Ubuntu attribuendogli l'indirizzo IP 192.1.3.2.

```
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 hostubuntu bash
```

Stesso discorso può essere fatto per la sottorete *rete_esterna* a cui è stato assegnato l'indirizzo 192.1.1.0/24. È stato mandato in esecuzione un container (*cliente1*) a partire dall'immagine costruita del Dockerfile per gli host Ubuntu attribuendogli l'indirizzo IP 192.1.1.2.

```
docker network create --driver bridge --subnet=192.1.2.0/24 dmz
docker run --privileged --network=dmz --ip 192.1.2.2 -p80:80 -p443:443 -tdi --name=webserver
    linode/lamp bash
docker exec --privileged -t webserver service apache2 start
docker run --privileged --network=dmz --ip 192.1.2.3 -p53:53/udp -tdi --name=dnsser
    cosmicq/docker-bind:latest bash
docker run --privileged --network=dmz --ip 192.1.2.4 -p20:20 -p21:21 -tdi --name=ftpserver
    ftpser 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.

```
docker run --privileged --sysctl net.ipv4.ip_forward=1 --network=rete_esterna --ip 192.1.1.5
    -td --name=firewall1 emmame/firewall_ulogd2 bash
docker exec --privileged -t firewall1 service ulogd2 restart
docker network connect --ip 192.1.2.5 dmz firewall1
```

In questa parte di script si esegue in modalità privilegiata un container dell'immagine firewall creato precedentemente e pushato su Docker Hub. Si noti che tale modalità è necessaria in questo caso, in quanto nel Dockerfile del firewall sono presenti le iptables: per eseguire una qualsiasi istruzione iptables sono necessari i permessi di root. Altra opzione di fondamentale importanza è *–sysctl net.ipv4.ip_forward=1* che configura il sistema come router, abilitando l'ip forwarding. Infatti per default, la policy IPv4 nei kernel Linux disabilita il supporto per l'inoltro IP, evitando alle macchine Linux di comportarsi come dei router.
Si noti, infine, che con l'istruzione *run* si sta collegando il sistema *firewall1* (firewall esterno) alla rete esterna mediante una delle interfacce avente indirizzo 192.1.1.5. Con l'ultima istruzione colleghiamo il firewall esterno alla sottorete *dmz* sfruttando l'interfaccia con indirizzo IP 192.1.2.5.
La seconda istruzione è importante in modo tale da riavviare il servizio di *ulogd2* una volta avviato il container, altrimenti questo non funzionerebbe.

```
docker run --privileged --sysctl net.ipv4.ip_forward=1 --network=rete_interna --ip 192.1.3.6
    -td --name=firewall2 emmame/firewall_ulogd2 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.

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

```
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
```

2.4 Docker-Compose

Compose è un tool per definire ed eseguire applicazioni Docker multi-container, usando il file di tipo YAML *docker-compose.yml*. Uno dei suoi vantaggi è che con un singolo comando, *docker-compose up*, è possibile creare e avviare tutti i servizi specificati nella configurazione.

Configurazione con Docker-Compose

É stata implementata una versione alternativa della configurazione della rete con docker-compose, con lo scopo di riportare il progetto anche in *Docker Security Playground*.
A tal proposito, sono state appositamente caricate su Docker Hub le immagini Docker precedentemente realizzate.

```
version: "3"
services:
    cliente1:
        image: emmame/simpleubuntu
        container_name: c1-esterno
        tty: true
        stdin_open: true
        privileged: true
        command: bash -c "route add default gw f1-esterno && bash"
        networks:
            net-esterna:
                ipv4_address: 192.1.1.2
    firewall1:
        image: emmame/firewall_ulogd2
        container_name: f1-esterno
        tty: true
        stdin_open: true
        privileged: true
        command: bash -c "service ulogd2 restart && bash"
        sysctls:
            - net.ipv4.ip_forward=1
        networks:
            net-esterna:
                ipv4_address: 192.1.1.5
            net-dmz:
                ipv4_address: 192.1.2.5
    webserver:
        image: linode/lamp
        stdin_open: true
        privileged: true
        ports:
            - '80:80/tcp'
            - '443:443/tcp'
        container_name: webserver1
        tty: true
        command: bash -c "service apache2 start && route add 192.1.1.2 gw f1-esterno &&
        route add 192.1.3.2 gw f2-interno && bash"
        networks:
            net-dmz:
                ipv4_address: 192.1.2.2
    firewall2:
             image: emmame/firewall_ulogd2
```

```
            container_name: f2-interno
            tty: true
            stdin_open: true
            privileged: true
            command: bash -c "service ulogd2 restart && bash"
            sysctls:
               - net.ipv4.ip_forward=1
            networks:
               net-interna:
                   ipv4_address: 192.1.3.6
               net-dmz:
                   ipv4_address: 192.1.2.6
    hostint1:
        image: emmame/simpleubuntu
        container_name: h1-interno
        tty: true
        privileged: true
        stdin_open: true
        command: bash -c "route add default gw f2-interno && bash"
        networks:
            net-interna:
                ipv4_address: 192.1.3.2
networks:
    net-esterna:
        name: net-192.1.1.0
        ipam:
            config:
                - subnet: 192.1.1.0/24
    net-dmz:
        name: net-192.1.2.0
        ipam:
            config:
                - subnet: 192.1.2.0/24
    net-interna:
        name: net-192.1.3.0
        ipam:
            config:
                - subnet: 192.1.3.0/24
```

Capitolo 3

Firewall con Iptables

Iptables, già precedentemente mezionato, è il tool che permette, tramite la creazione di regole, di avere una protezione per il filtraggio del traffico. Esso implementa un **Firewall Packet Filtering**.
Ogni regola inserita deve necessariamente far parte di una *catena*, la quale è una lista di regole. Queste ultime a loro volta, vengono inserite in tabelle (di nostro interesse sono la tabella *filter* e quella *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 caso specifico del progetto d'esame, la catena utilizzata sarà quella di FORWARD.
Le catene standard della tabella nat, la quale si occupa principalmente dell'instradamento e del natting dei pacchetti che attraversano il firewall, sono:

- **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

La regola *-j LOG* di iptables risulta disabilitata nel caso in cui il firewall è costruito a partire da un container Docker. Questo per prevenire eventuali attacchi DoS che un container può

eseguire nei confronti della macchina che lo "ospita" (inondandola di messaggi di LOG).
Per questo motivo è stata necessaria la ricerca di un metodo alternativo per realizzare questo logging, trovata nel modulo **ulogd2**, il quale non è altro che un *userspace logging daemon* per il logging effettuato mediante netfilter/iptables.
La sintassi delle nuove regole di log è praticamente identica alla precedente. L'unica differenza sta nella sostituzione di NFLOG al target standard LOG.
Inoltre è risultato necessario modificare il file di configurazione del modulo: *ulogd.conf*. In esso sono stati decommentati gli opportuni plugin, impostata la configurazione ed è stato specificato il file su cui andranno memorizzati i pacchetti sottoposti a logging.

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 decommentati i plugin che verranno caricati prima dell'inizializzazione di ulogd, per permettere il corretto funzionamento del log:

```
##########################################################################
# PLUGIN OPTIONS
##########################################################################
# We have to configure and load all the plugins we want to use
# general rules:
#
# 0. don't specify any plugin for ulogd to load them all
# 1. load the plugins _first_ from the global section
# 2. options for each plugin in seperate section below

plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inppkt_NFLOG.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inppkt_ULOG.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inppkt_UNIXSOCK.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_inpflow_NFCT.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IFINDEX.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2STR.so"
plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2BIN.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2HBIN.so"
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 è possibile notare, esistono diverse tipologie di plugin. Le possiamo suddividere in tre categorie fondamentali:

- **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"
sync=1
```

3.2 Regole

Il primo passo da compiere, per poter definire le regole attraverso delle *catene*, è eliminare tutte le regole preesistenti nella catena iniziale: il comportamento predefinito di *iptables* è

una policy di tipo *ACCEPT*, la quale permette a tutto il traffico di essere accettato e non bloccato da nessun dispositivo nella rete. Dunque, procediamo al reset delle regole sia per la tabella *filter* che per la *nat* e si eliminano le catente non standard vuote, per entrambi il **firewalls**[1]:

```
# Eliminazione catene standard #
docker exec --privileged -t firewall1 iptables -F
docker exec --privileged -t firewall1 iptables -F -t nat

# Eliminazione catene non standard vuote #
docker exec --privileged -t firewall1 iptables -X
```

Difatti, il comando di iptables con l'opzione *-X* permette di eliminare la catena vuota (non standard) e opzionalmente è possibile indicare il nome della catena da eliminare. Analogamente, l'opzione *-F* prevede la cancellazione delle catene standard.
Dopo aver cancellato le regole per le catene preesistenti, vengono impostate le nuove regole per il funzionamento della rete. Le nuove policy effettueranno il *DROP* di tutti i pacchetti in *INPUT, FORWARD e OUTPUT*:

```
# Policy di base #
docker exec --privileged -t firewall1 iptables -P INPUT DROP
docker exec --privileged -t firewall1 iptables -P OUTPUT DROP
docker exec --privileged -t firewall1 iptables -P FORWARD DROP
```

L'opzione *-P* consente di inizializzare le nuove regole per la catena.
Dopo aver proceduto alla standardizzazione delle regole base, si è passati alla rassegna degli eventuali attacchi che potrebbero assoggettare la rete. Una prima limitazione è stata effettuare i *DROP* dei pacchetti frammentati del protocollo HTTP. Tale scelta è dettata dal voler evitare attacchi come il *"Tiny fragment attack"*, il quale prevede di utilizzare piccoli pacchetti frammentati per provocare problemi durante il riassemblaggio causando un attacco DoS. Per evitare tale situazione è stata aggiunta la regola:

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

Un'altra scelta effettuata è stata quella di effettuare il *DROP* dei pacchetti appartenenti al protocollo di trasporto **TCP** non congruenti. Difatti, molte operazioni di scanning di sistemi ed eventuali attacchi partono proprio dall'invio di questo tipo di pacchetti contenenti *flags* settati ad arte per "infastidire" il sistema. Come è noto, il protocollo TCP è utilizzato per il trasporto dei dati creando una connessione tra client e server tramite il **three-way handshake**. Tali pacchetti possono essere definiti come pacchetti *"no-sense"* poichè, effettivamente, non hanno alcuna funzionalità nella comunicazione. Alcuni sono:

[1] Si noti che verranno riportati solo le regole per il *firewall1*. In maniera analoga sono implementati quelli per il *firewall2*, a meno di esplicite differenze

```
docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --tcp-flags ALL ACK,RST,SYN,FIN -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --tcp-flags ALL ALL -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
```

Alcuni di questi pacchetti sono noti come tecniche di scan dalla comunità di internet, per esempio: **null scan** in cui sono settati tutti i flag come *null*.

3.2.1 IP Spoofing

L'**IP spoofing** è una tecnica di attacco che prevede l'utilizzo di un pacchetto IP nel quale viene falsificato l'indirizzo IP del mittente.
Tale attacco è possibile semplicemente perchè al fine di effettuare routing, l'unico campo che interessa il protocollo di trasporto è l'indirizzo destinazione, dunque, non viene effettuata alcuna discriminazione in base all'indirizzo sorgente.
E' stata introdotta questa regola per evitare che un cliente esterno possa fingersi un host della rete interna, ed avere i suoi eventuali privilegi. Verranno scartati tutti i pacchetti in arrivo sull'interfaccia ***eth0*** e che hanno come ip sorgente uno qualsiasi della rete interna (***192.1.3.0/24***):

```
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 di tipo DoS basato sul protocollo TCP. Questa tipologia di attacco prevede che l'attaccante mandi una richiesta di connessione verso la vittima utilizzando il flag ***SYN*** alto in modo tale da richiedere la connessione a cui la vittima risponde con i flag ***ACK, SYN*** alti, ma l'attaccante non risponderà mai con un ulteriore ***ACK***, mantenendo la connessione sempre attiva. Tale attacco non si limita a ciò, ma sfrutta anche lo spoofing ip in modo tale da contattare la vittima con numerosi indirizzi IP differenti.
Tale problematica è risolvibile utilizzando una catena che limita il passaggio dopo un certo numero di pacchetti IP con il flag SYN alto. Di seguito è esaminata la creazione di tale catena.
La catena viene eseguita con la denominazione *"SYN_FLOOD"* solo se il pacchetto in ingresso appartiene al protocollo TCP e ha settato come alto il flag di SYN:

```
docker exec --privileged -t firewall1 iptables -N SYN_FLOOD
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:

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

Altrimenti viene effettuato il *DROP* di tale pacchetto:

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

É bene descrivere il comportamento della regola che permette il controllo del pacchetto, in particolare le opzioni **limit**. Tale regola permette di settare il limitare di pacchetti al secondo ricevuti per evitare attacchi DoS: di default tale valore è impostato a un rate massimo di 5 pacchetti, dopodiché verranno scartati tutti i pacchetti istanziati come possibili attacchi per un tempo standard durante il quale verrà ricaricato il carico di "burst" che può ricevere il server. Tale situazione è visionabile attraverso un grafico che permette di visualizzare i confini entro cui non si parla di attacco oltre ai quali è possibile definire un DoS:

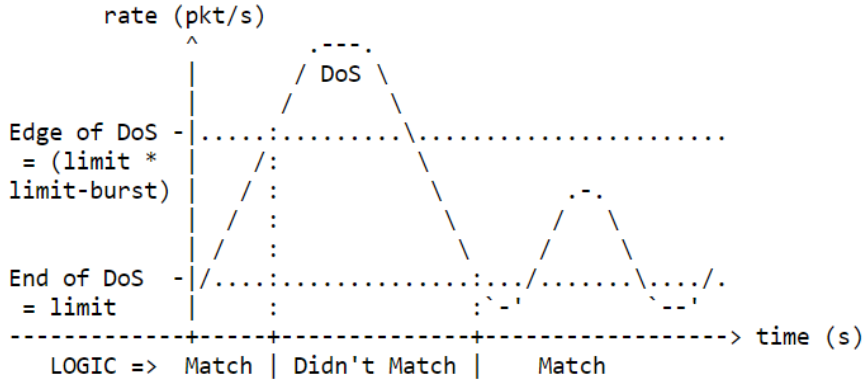

Figura 3.1: Grafico di andamento di limit

3.2.3 Ping of Death

Il **Ping of death** è un attacco di tipo Dos in cui l'attaccante invia un pacchetto ICMP di grandi dimensioni verso un sistema vittima. Tale pacchetto viene frammentato e una volta a destinazione provocherà un **buffer overflow** sul sistema a causa del superamento della dimensione consentita, generando così un denial of service.
Tale attacco oltre ad essere evitabile effettuando il *DROP* dei pacchetti frammentati, può essere eliminato procedendo come fatto nel caso del SYN flood attack tramite la generazione di una catena:

```
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
```

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:

```
docker exec --privileged -t firewall1 iptables -A INPUT -m state --state INVALID -j DROP
docker exec --privileged -t firewall1 iptables -A FORWARD -m state --state INVALID -j DROP
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:

```
docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth0 -o eth2
    -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*):

```
docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth0 -o eth1
    -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
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*):

```
docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth2 -o eth1
    -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
docker exec --privileged -t firewall1 iptables -t filter -A FORWARD -i eth1 -o eth2
    -m state --state ESTABLISHED,RELATED -j ACCEPT
```

3.2.5 UDP

UDP Flood Attack

Un **UDP flood** è un attacco di tipo DoS basato sul protocollo di trasporto UDP che consiste nell'inondare la vittima di datagrammi UDP. Come è noto, a differenza del protocollo TCP, UDP è un protocollo *connectionless*, dunque, non necessita di concordare una connessione per l'invio di datagrammi tra due host. Questo tipo di attacco può essere gestito attraverso l'impostazione di una regola nei firewalls che limita il rate di datagrammi ricevuti dalla vittima:

```
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
```

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:

```
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
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:

```
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**:

```
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: "
```

Figura 4.1: Verifica IP spoofing

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 --rawip -d 120 --spoof 192.1.3.2 192.1.2.2
```

L'attacco prevede l'invio di pacchetti IP (*–rawip*) di dimensione 120 bytes, il cui indirizzo IP 192.1.3.2 viene forgiato ad hoc mediante l'opzione *–spoof* e diretti verso il web server avente indirizzo IP 192.1.2.2. Il risultato è riportato nei log presenti all'interno del firewall esterno:

```
Apr  6 14:50:48 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:48 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:50 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:50 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:52 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:52 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:54 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:54 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:56 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
Apr  6 14:50:56 bc79ac0d5099 FORWARD Log pre-regola :  IN=eth0 OUT=eth1 MAC=02:42:c0:01:01:05:02:42:c0:01:01:02:08:00 SRC=192.1.3.2 DST=192.1.2.2
```

Figura 4.2: Verifica IP spoofing

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: "
# Creo nuova catena SYN_FLOOD
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
# Il pacchetto viene fatto passare se rispetta i limiti prefissati
# Numero massimo di confronti al secondo (in media) = 1
# Numero massimo di confronti iniziali (in media) = 5 default
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: "
```

Figura 4.3: Verifica Syn Flood Attack

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:

```
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*:

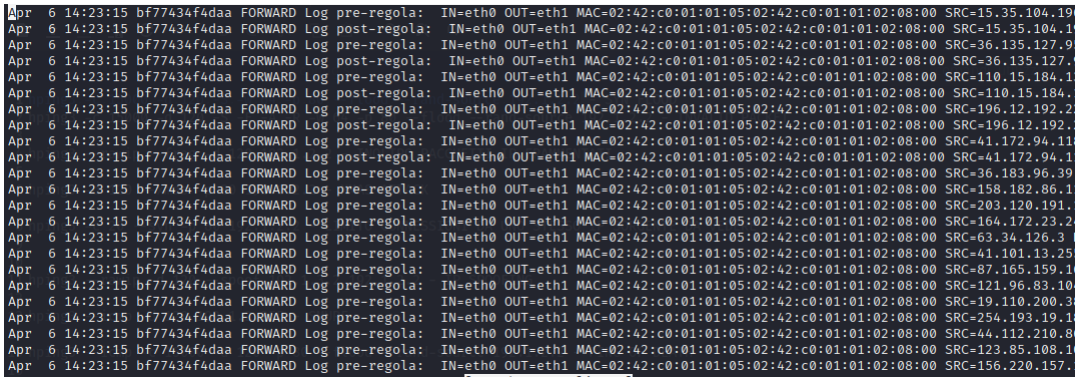

Figura 4.4: Parte del log file - Syn Flood Attack

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.

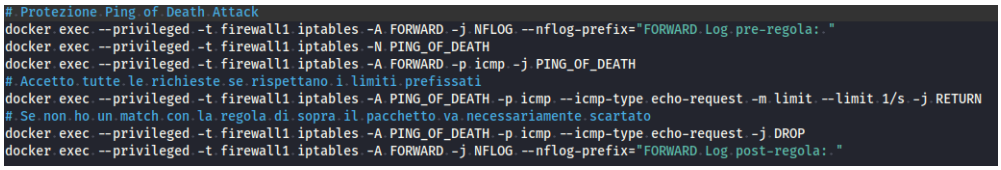

Figura 4.5: Verifica Ping of Death Attack

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

```
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.

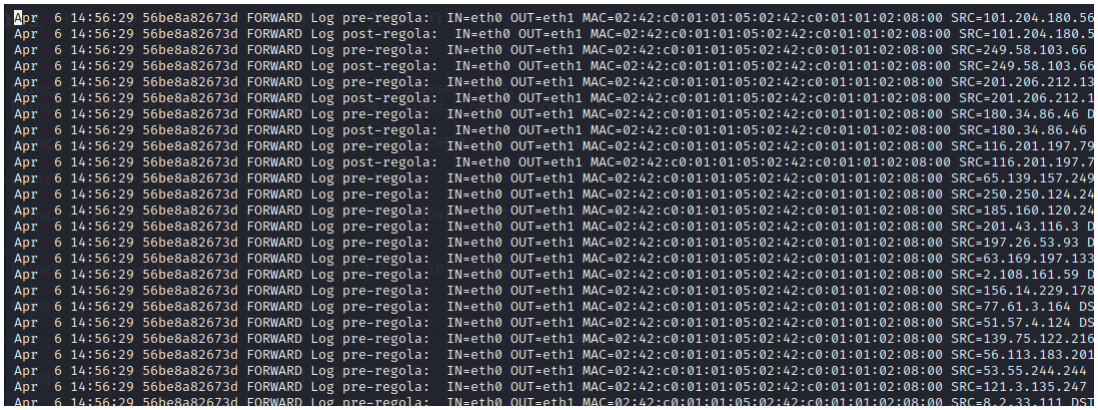

Figura 4.6: Verifica Ping of Death Attack

4.4 Test UDP Flood Attack

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

```
# Evito UDP-flood Attacks
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: "
```

Figura 4.7: Verifica UDP Flood Attack

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.

```
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.

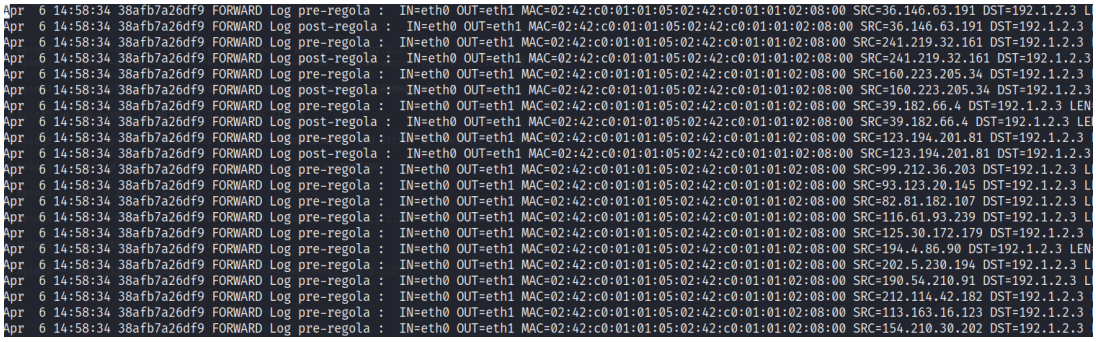

Figura 4.8: Verifica UDP Flood Attack

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"*.