FIREWALLS

COMPUTER SECURITY I
DVGC19

3 DECEMBER 2023

AHMAD SHAMMOUT : SHMOOT001@GMAIL.COM
JOHANNA OLSSON : JOHANNAMARIA1@LIVE.SE

1. Firewalls

A firewall is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. It acts as a barrier between a trusted internal network and untrusted external networks, such as the internet, to prevent unauthorized access and protect against potential cyber threats. Firewalls can be implemented in hardware or software, and they play a crucial role in safeguarding computer systems and networks.

2. Performed Tasks

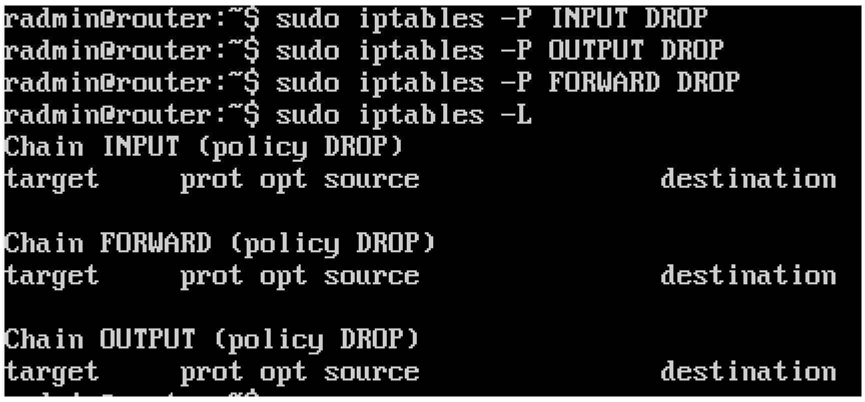
2.1 Default Policy

1. We chose DROP as the default policy for the INPUT, OUTPUT, and FORWARD chains. The reason why we chose DROP is because it is a security-focused approach that follows the "default deny" principle, where incoming traffic automatically gets discarded or dropped unless it is explicitly allowed. This strategy enhances network security by minimizing the attack surface, preventing unauthorized access attempts, and controlling outbound connections.

   To choose the default policy for all firewall chains, we used these commands :

   sudo iptables -P INPUT DROP
   sudo iptables -P OUTPUT DROP
   sudo iptables -P FORWARD DROP

   -P was used to set the policy for the chain to the given target.
   Then we used *sudo iptables -L* to list all the rules in the selected chain.



```
radmin@router:~$ sudo iptables -P INPUT DROP
radmin@router:~$ sudo iptables -P OUTPUT DROP
radmin@router:~$ sudo iptables -P FORWARD DROP
radmin@router:~$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

Figure 1 : Choosing DROP for all firewall chains.

2. a) To allow internal servers to access the outside network, we used two rules to make it happen, the first one :

   sudo iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
   In this configuration, we permitted traffic from the internal interface (*eth1*) of the router to the external interface (*eth0*).

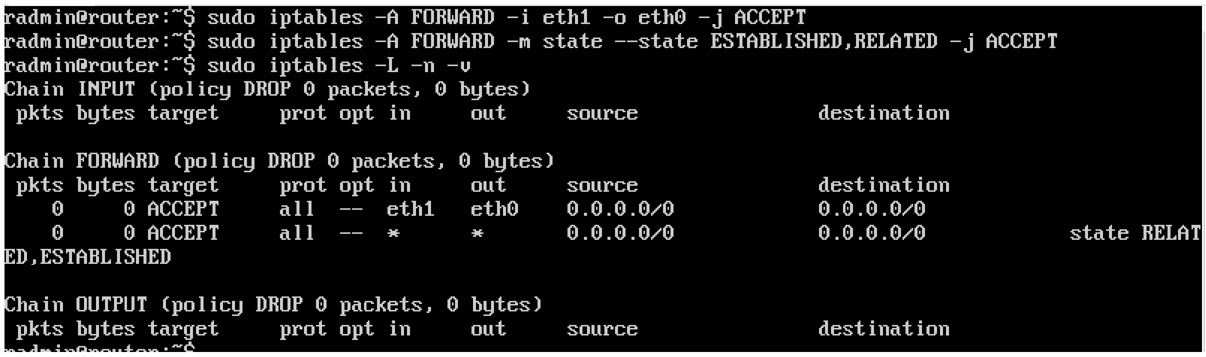
   "-A FORWARD" : appends the rule to the FORWARD chain.
   "-i eth1" : Specifies the input interface as eth1.
   "-o eth0" : Specifies the output interface as eth0.
   "-j ACCEPT" : Specifies the target action if the packet matches the criteria, and in this case, it's to accept the packet.

   And the second rule :

   sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
   This rule allows traffic in the FORWARD chain that is associated with existing or related connections. For example, if your system has initiated a connection (*ESTABLISHED*) or if the traffic is related to an existing connection (*RELATED*), this rule ensures that the packets are accepted and allowed to pass through the FORWARD chain.

   "-m state –state ESTABLISHED, RELATED" : Uses the state module to match packets based on their connection state. In this case, it allows packets that are part of established connections or related to established connections.

   Then we used the command *"sudo iptables -L -v -n"* to display a list of the current iptables rules in a detailed and numeric format.

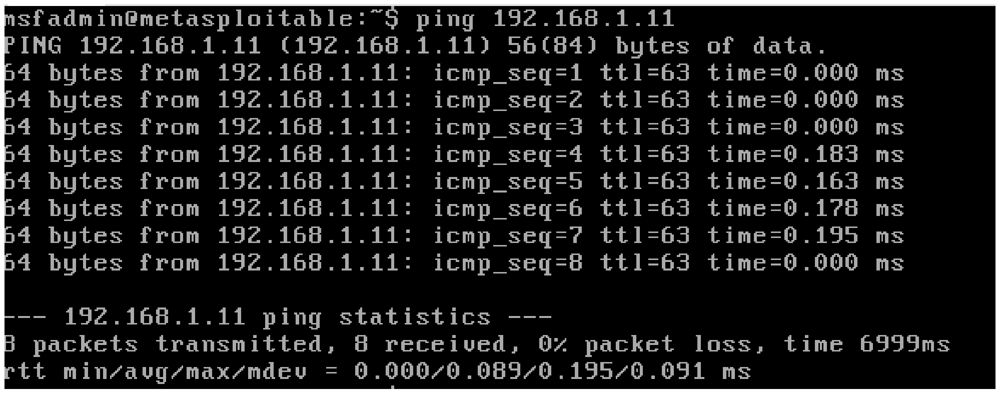


```
radmin@router:~$ sudo iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
radmin@router:~$ sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
radmin@router:~$ sudo iptables -L -n -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  eth1   eth0    0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            state RELAT
ED,ESTABLISHED

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Figure 2 : List of the current iptables rules in a detailed and numeric format

We also tested the set rules to verify them working as intended by pinging from the internal servers to the outside network, and from the outside network to the inside servers.

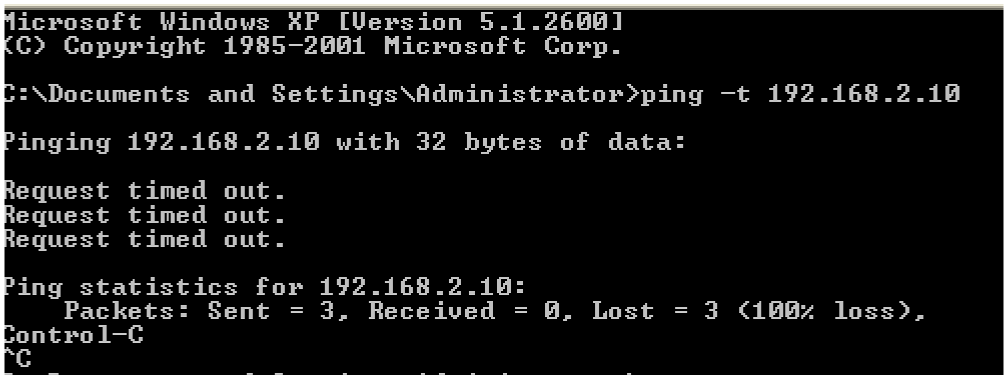
First test of pinging from the Metasploitable server (inside) to Kali Linux (outside) is visualized below, where we can tell that the inside server gets access to the outside network.



```
msfadmin@metasploitable:~$ ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=63 time=0.000 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=63 time=0.000 ms
64 bytes from 192.168.1.11: icmp_seq=3 ttl=63 time=0.000 ms
64 bytes from 192.168.1.11: icmp_seq=4 ttl=63 time=0.183 ms
64 bytes from 192.168.1.11: icmp_seq=5 ttl=63 time=0.163 ms
64 bytes from 192.168.1.11: icmp_seq=6 ttl=63 time=0.178 ms
64 bytes from 192.168.1.11: icmp_seq=7 ttl=63 time=0.195 ms
64 bytes from 192.168.1.11: icmp_seq=8 ttl=63 time=0.000 ms

--- 192.168.1.11 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6999ms
rtt min/avg/max/mdev = 0.000/0.089/0.195/0.091 ms
```

Figure 3 : Ping from the Metasploitable server to Kali Linux, 100% packets received

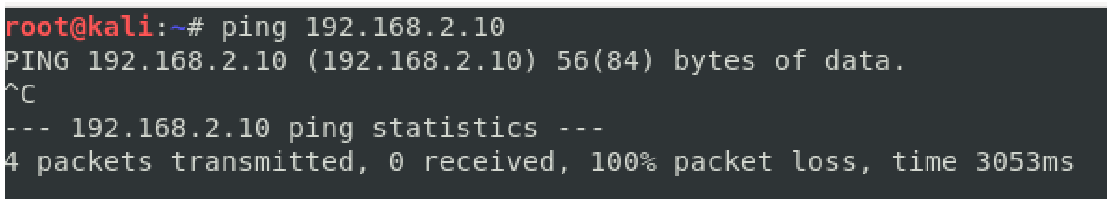
Second test of pinging was from the Windows network to the Metasploitable server, and it is visualized below. Compared to the first test, here we can tell that the outside network does not get access to the inside server, as planned.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ping -t 192.168.2.10

Pinging 192.168.2.10 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.10:
    Packets: Sent = 3, Received = 0, Lost = 3 (100% loss),
Control-C
^C
```

Figure 4 : Ping from Win XP to the Metasploitable server, 0% packets received.

In the final test we again pinged from the other outside network, this time using Kali Linux, to the Metasploitable server. Equal to the second test, the outside network does not get access to the inside server.



```
root@kali:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
^C
--- 192.168.2.10 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3053ms
```

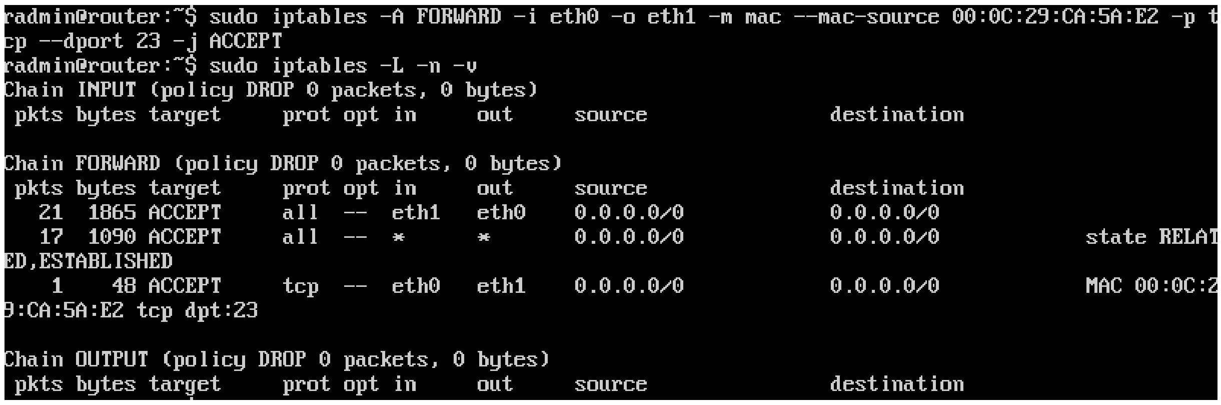
Figure 5: Ping from Kali Linux to the Metasploitable server, 0% packets received.

b)
Enabling the Windows machine to transmit Telnet packets to the Metasploitable server through iptables involves creating a rule in the FORWARD chain that designates the source MAC address of the Windows machine. To accomplish this, we implemented the following rule :

sudo iptables -A FORWARD -i eth0 -o eth1 -m mac --mac-source 00:0C:29:CA:5A:E2 -p tcp --dport 23 -j ACCEPT

This iptables rule is designed to allow the forwarding of TCP traffic with destination port 23 (Telnet) from the *eth0* interface to the *eth1* interface, specifically for packets with a source MAC address of 00:0C:29:CA:5A:E2

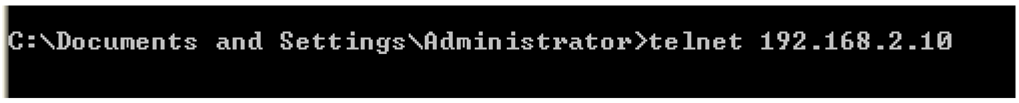
"-i eth0" : Specifies the input interface as eth0.
"-o eth1" : Specifies the output interface as eth1.
"-m mac --mac-source 00:0C:29:CA:5A:E2" : Uses the "mac" module to match packets based on their source MAC address, allowing only packets with the specified MAC address (00:0C:29:CA:5A:E2).
"-p tcp --dport 23" : Specifies that the rule applies to TCP traffic on port 23 (Telnet).



```
radmin@router:~$ sudo iptables -A FORWARD -i eth0 -o eth1 -m mac --mac-source 00:0C:29:CA:5A:E2 -p t
cp --dport 23 -j ACCEPT
radmin@router:~$ sudo iptables -L -n -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   21  1865 ACCEPT     all  --  eth1   eth0    0.0.0.0/0            0.0.0.0/0
   17  1090 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            state RELAT
ED,ESTABLISHED
    1    48 ACCEPT     tcp  --  eth0   eth1    0.0.0.0/0            0.0.0.0/0            MAC 00:0C:2
9:CA:5A:E2 tcp dpt:23

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Figure 6 : List of the current rule in a detailed and numeric format

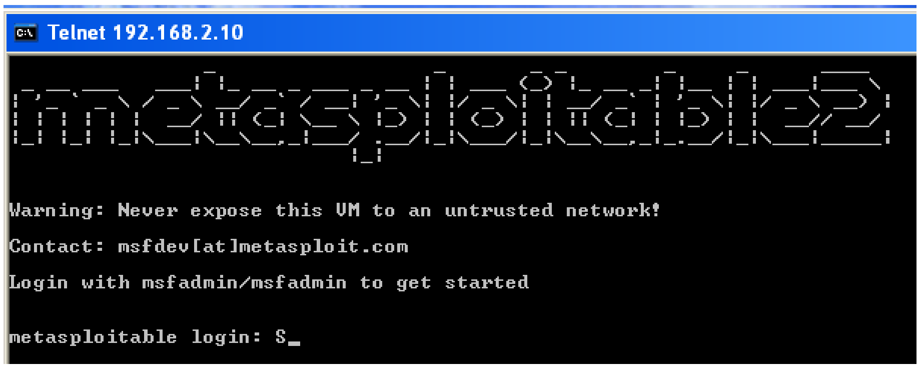
After applying the rule of allowing the Windows machine to send telnet packets to the Metasploitable server, we verified that our rule worked as intended by trying to send a packet.



```
C:\Documents and Settings\Administrator>telnet 192.168.2.10
```

Figure 7 : Command used in the Windows terminal for sending the telnet packet

By sending the packet, we were then able to get remote control over the Metasploitable server, as visualized below.



```
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started

metasploitable login: S_
```

Figure 8 : Remote control over the Metasploitable server by sending the telnet packet.

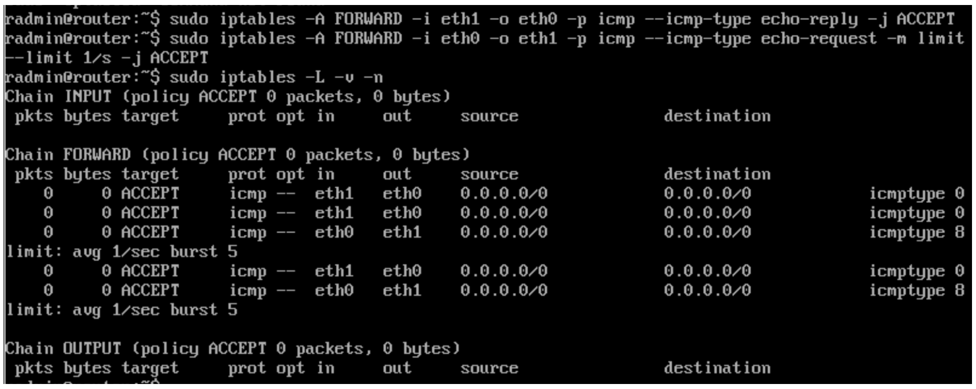
2.2 ICMP Packets Filtering

To allow legitimate ICMP traffic (such as ICMP echo requests commonly used for ping) from the outside network to the inside network using iptables, you can add a rule to permit incoming ICMP packets. However, it's important to restrict the rule to allow only specific ICMP types and limit the rate to avoid abuse. With our network interface being eth0 and the internal network interface being eth1, we added the following rules to allow ICMP echo requests from the outside network to the inside network and also echo replies:

sudo iptables -A FORWARD -i eth1 -o eth0 -p icmp –icmp-type echo-reply -j ACCEPT
sudo iptables -A FORWARD -i eth0 -o eth1 -p icmp --icmp-type echo-request -m limit --limit 1/minute -j ACCEPT

Explanation of the rules:
"-p icmp --icmp-type echo-request" : Matches ICMP echo requests.
"-m limit --limit 1/minute" : Matches packets at a limited rate, a rule using this extension will match until the set limit is reached.
"echo-request" : A message sent by a host to another host, typically to test if the remote host is reachable and responsive.
"echo-reply" : The response to an echo-request message.

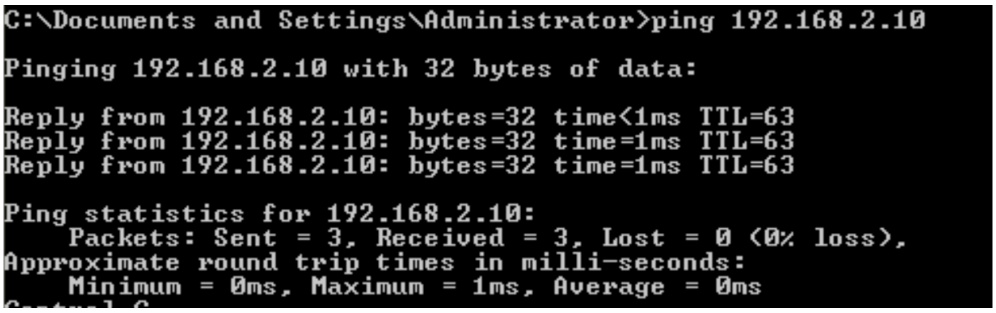
These rules allow legitimate ICMP echo requests and replies to pass through the firewall from the outside network to the inside network.



```
radmin@router:~$ sudo iptables -A FORWARD -i eth1 -o eth0 -p icmp --icmp-type echo-reply -j ACCEPT
radmin@router:~$ sudo iptables -A FORWARD -i eth0 -o eth1 -p icmp --icmp-type echo-request -m limit
--limit 1/s -j ACCEPT
radmin@router:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     icmp --  eth1   eth0    0.0.0.0/0            0.0.0.0/0            icmptype 0
    0     0 ACCEPT     icmp --  eth1   eth0    0.0.0.0/0            0.0.0.0/0            icmptype 0
    0     0 ACCEPT     icmp --  eth0   eth1    0.0.0.0/0            0.0.0.0/0            icmptype 8
limit: avg 1/sec burst 5
    0     0 ACCEPT     icmp --  eth1   eth0    0.0.0.0/0            0.0.0.0/0            icmptype 0
    0     0 ACCEPT     icmp --  eth0   eth1    0.0.0.0/0            0.0.0.0/0            icmptype 8
limit: avg 1/sec burst 5

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

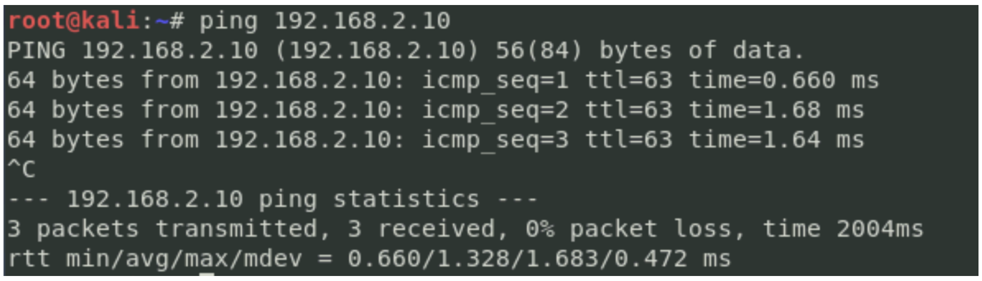
Figure 9 : List of the current rule in a detailed and numeric format

To test the set rules we then pinged from the Kali machine and the Windows machine to the Metasploitable server, wanting the echo-requests to reach the server and also getting the echo-replies back.



```
C:\Documents and Settings\Administrator>ping 192.168.2.10

Pinging 192.168.2.10 with 32 bytes of data:

Reply from 192.168.2.10: bytes=32 time<1ms TTL=63
Reply from 192.168.2.10: bytes=32 time=1ms TTL=63
Reply from 192.168.2.10: bytes=32 time=1ms TTL=63

Ping statistics for 192.168.2.10:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

Figure 10 : Ping from the Windows machine to the Metasploitable server



```
root@kali:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=63 time=0.660 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=63 time=1.68 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=63 time=1.64 ms
^C
--- 192.168.2.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.660/1.328/1.683/0.472 ms
```

Figure 11 : Ping from the Kali machine to the Metasploitable server.

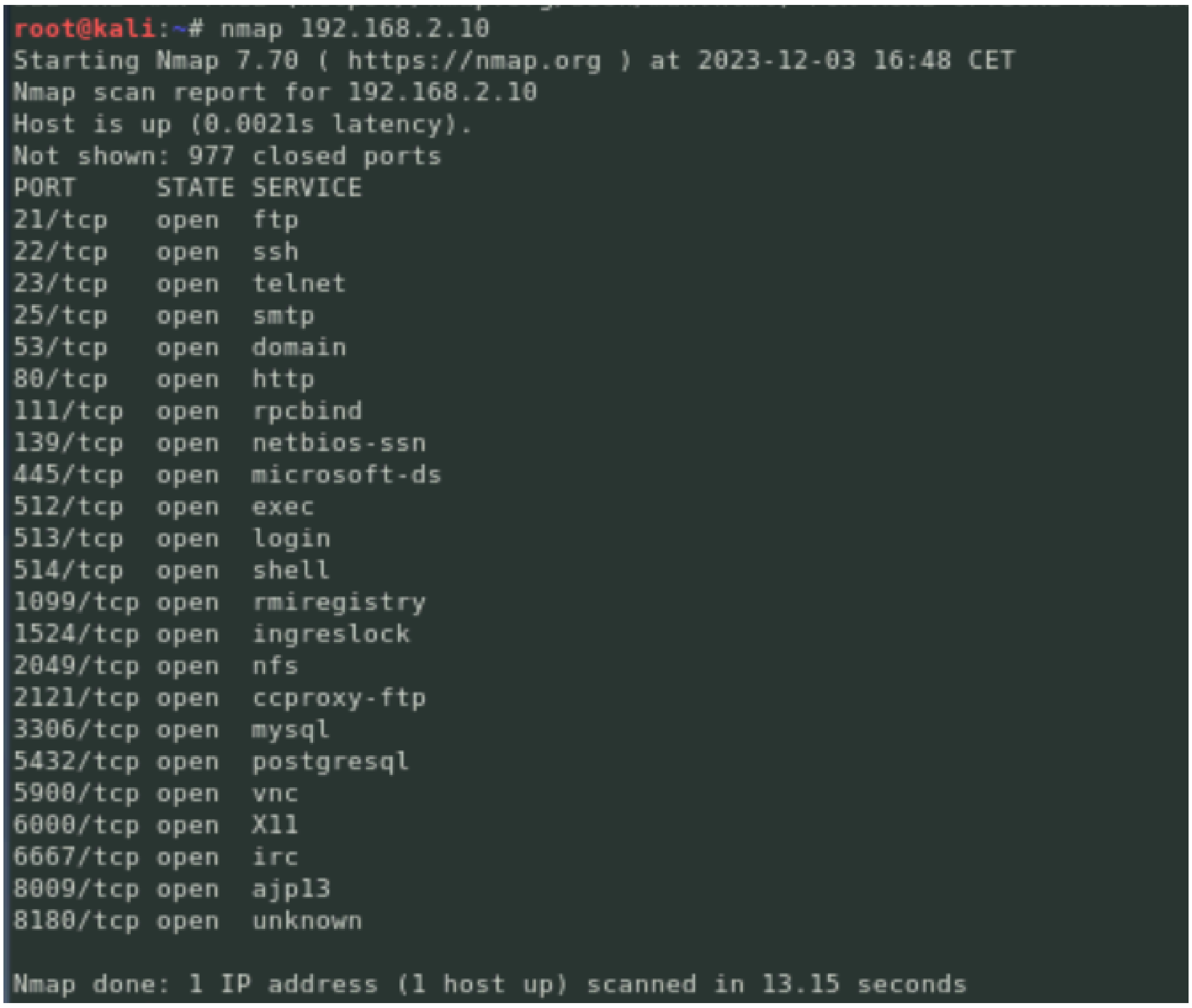
As we can tell by figure 10 and 11, all sent packets are received and replied to by the Metasploitable server, proving that our rules for ICMP traffic works as intended.

Legitimate ICMP traffic is traffic that is not harmful to the host. This includes echo requests and replies, time exceeded and destination unreachable messages, and redirect messages. The “ping” command is a well-known tool that sends echo requests and expects echo replies to determine the reachability of a host. Time exceeded and destination unreachable messages provide information about network issues, and redirect messages are used by routers to inform the host that a better route is available for a specific destination.

Illegitimate ICMP traffic is traffic that could be potentially harmful to the host. This includes ICMP flood attacks, ping of death, smurf attack, ICMP redirect spoofing, and ICMP-based scanning. ICMP flood attacks are used to overwhelm a target with a large volume of ICMP traffic, causing a denial-of-service situation. Ping of death is an attack where oversized ICMP packets are sent to a target, exploiting vulnerabilities in the target’s network stack causing it to crash or make the system unstable. Smurf attacks are when a large number of ICMP echo requests are sent to an IP broadcast address, causing multiple hosts to reply to the victim with echo replies which overwhelms its resources. ICMP redirect spoofing is when an attacker sends fake ICMP redirect messages to manipulate a host’s routing table, potentially leading to traffic interception or redirection. ICMP-based scanning is using ICMP messages for network reconnaissance, trying to discover live hosts or map network topology.

2.3 Defense Against Port Scanning

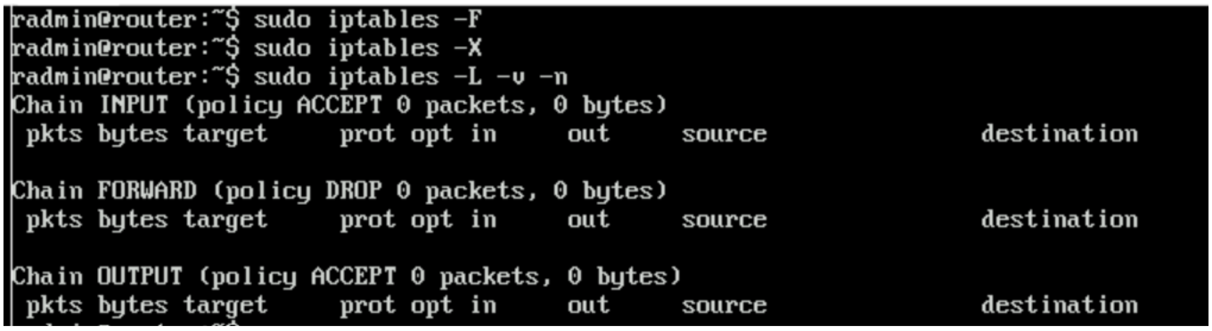
In the initial phase, we employed the Nmap tool on Kali Linux to scan the ports of the Metasploitable machine. This was done to establish a baseline for port status before implementing router rules, facilitating a comparison of results later. As depicted in Figure 12, the port scan of Metasploitable successfully identified all open ports. However, it will be highlighted in the upcoming sections of the report that not all open ports remained visible after the application of router rules.



```
root@kali:~# nmap 192.168.2.10
Starting Nmap 7.70 ( https://nmap.org ) at 2023-12-03 16:48 CET
Nmap scan report for 192.168.2.10
Host is up (0.0021s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
8009/tcp  open  ajp13
8180/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 13.15 seconds
```

Figure 12 : Port scanning Metasploitable by using Nmap on Kali.

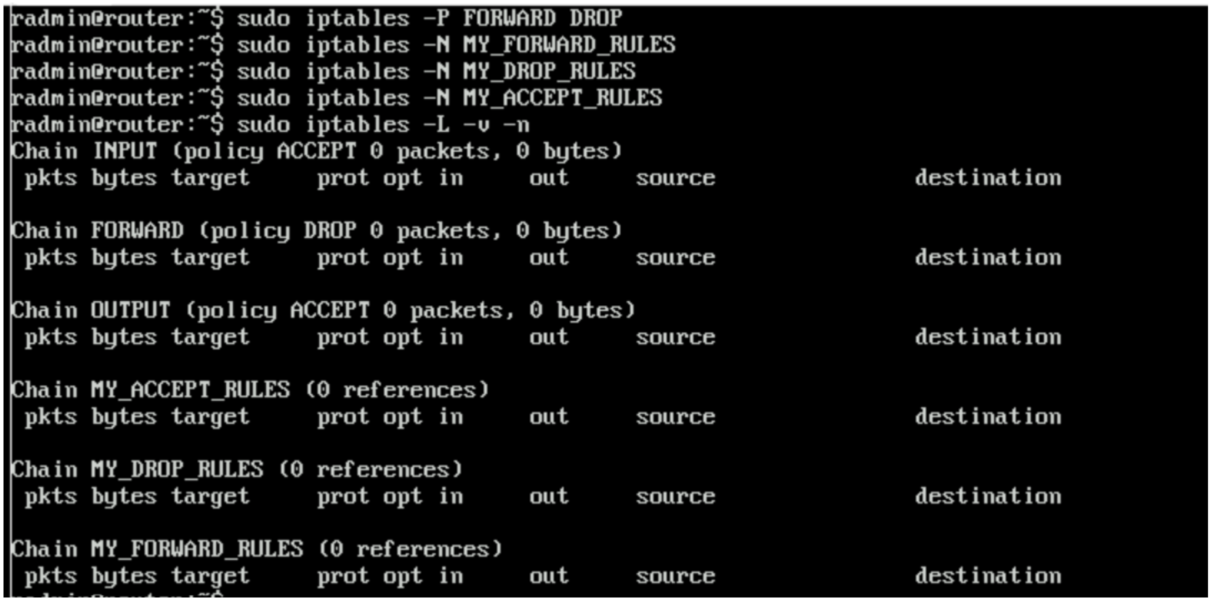
We initiated the process by deleting all pre-existing rules and chains on the router.



```
radmin@router:~$ sudo iptables -F
radmin@router:~$ sudo iptables -X
radmin@router:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

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

Figure 13 : Deleting all existing rules and chains on the router.

Following that, we established the default policy for the *FORWARD* chains as *DROP* and generated three new chains: *MY_FORWARD_RULES, MY_DROP_RULES*, and *MY_ACCEPT_RULES*.



```
radmin@router:~$ sudo iptables -P FORWARD DROP
radmin@router:~$ sudo iptables -N MY_FORWARD_RULES
radmin@router:~$ sudo iptables -N MY_DROP_RULES
radmin@router:~$ sudo iptables -N MY_ACCEPT_RULES
radmin@router:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

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

Chain MY_ACCEPT_RULES (0 references)
 pkts bytes target     prot opt in     out     source               destination

Chain MY_DROP_RULES (0 references)
 pkts bytes target     prot opt in     out     source               destination

Chain MY_FORWARD_RULES (0 references)
 pkts bytes target     prot opt in     out     source               destination
```

Figure 14 : Setting the default policy for *FORWARD* chain as *DROP*, and creating 3 new chains

Afterwards, we began the process of applying the rules.

First rule :
"sudo iptables -A FORWARD -m state –state NEW -j MY_FORWARD_RULES"

This rule channels incoming packets in the *NEW* state to *MY_FORWARD_RULES*, facilitating customized security measures and thorough inspection of new connections.

Second rule :
"sudo iptables -A MY_FORWARD_RULES -m recent –name DROP_MASK –set -j MY_DROP_RULES"

This rule functions by marking the source IP address of incoming packets within the *MY_FORWARD_RULES* chain. The mark, identified as *DROP_MARK*, is set using *--set*. Subsequently, the rule directs the packet flow to *MY_DROP_RULES* chain using *-j*.

This rule establishes a tracking mechanism within the *MY_FORWARD_RULES* chain by marking source IP addresses of traversing packets. The marked information is then crucial for subsequent rules in the *MY_DROP_RULES* chain, enabling conditional and targeted actions based on the marked source IP addresses. Overall, this rule sets the foundation for a nuanced and responsive packet handling approach, allowing the *MY_DROP_RULES* chain to execute specific actions based on the marked IP addresses.

Third rule :
"sudo iptables -A MY_DROP_RULES -m recent –name DROP_MASK –update –seconds 15 –reap –hitcount 3 -j DROP"

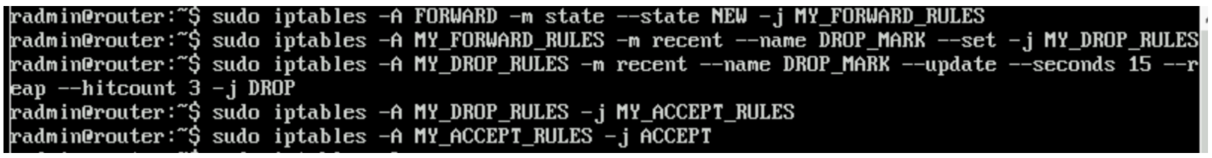
In this rule, packets traversing the *MY_DROP_RULES* chain are evaluated based on the source IP's activity. If a source IP generates more than 3 hits within the last 15 seconds, indicative of potentially malicious behavior, the rule enforces a proactive security measure by dropping the respective packet. This strategic use of the "*recent*" module within iptables contributes to the prevention of network abuse and enhances the overall security posture.

Fourth rule :
“sudo iptables -A MY_DROP_RULES -j MY_ACCEPT_RULES”

This rule defines the post-processing behavior within the *MY_DROP_RULES* chain. After evaluating and potentially dropping packets based on specific conditions, the rule instructs the firewall to jump to *MY_ACCEPT_RULES*. This step allows for the subsequent acceptance of packets that have undergone processing in the *MY_DROP_RULES* chain, contributing to a structured and responsive approach to network traffic management.

Fifth rule : Accepting Post-Processing
“sudo iptables -A MY_ACCEPT_RULES -j ACCEPT”

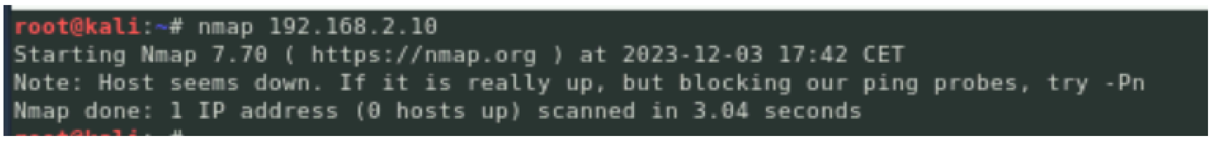
This rule finalizes packet processing in the *MY_FORWARD_RULES* chain. Following evaluations, this rule permits packets that have successfully navigated processing, ensuring controlled acceptance of network traffic.



```
radmin@router:~$ sudo iptables -A FORWARD -m state --state NEW -j MY_FORWARD_RULES
radmin@router:~$ sudo iptables -A MY_FORWARD_RULES -m recent --name DROP_MARK --set -j MY_DROP_RULES
radmin@router:~$ sudo iptables -A MY_DROP_RULES -m recent --name DROP_MARK --update --seconds 15 --r
eap --hitcount 3 -j DROP
radmin@router:~$ sudo iptables -A MY_DROP_RULES -j MY_ACCEPT_RULES
radmin@router:~$ sudo iptables -A MY_ACCEPT_RULES -j ACCEPT
```

Figure 15 : Applying the rules on the router.

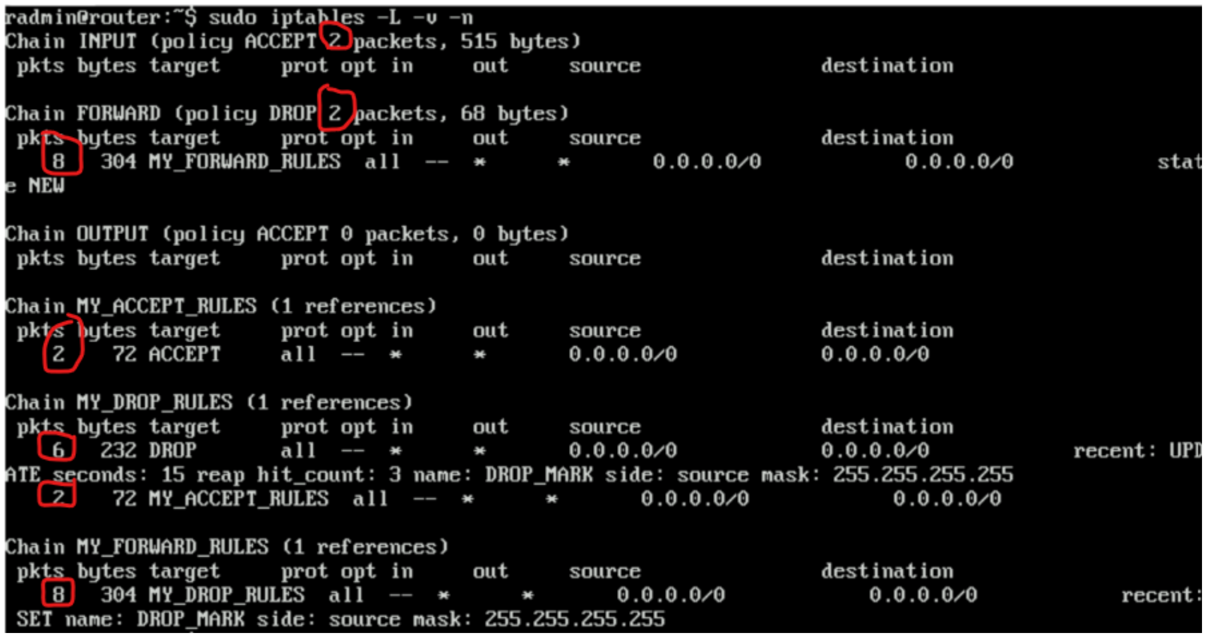
Subsequently, we conducted another port scan on Metasploitable using Nmap, and the results are illustrated in Figure 16. The findings indicated that Metasploitable had successfully blocked Kali's ping, signifying that the machine could no longer be port scanned after implementing all the rules on the router.



```
root@kali:~# nmap 192.168.2.10
Starting Nmap 7.70 ( https://nmap.org ) at 2023-12-03 17:42 CET
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.04 seconds
```

Figure 16 : Port scan the Metasploitable machine by using Nmap on Kali Linux.

Next, we employed the standard command "*sudo iptables -L -v -n*" to display the current iptables rules. Following our attempt to port scan the Metasploitable machine, the results showed that our rules processed some packets—some were dropped, while others were accepted.



```
radmin@router:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 2 packets, 515 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 2 packets, 68 bytes)
 pkts bytes target     prot opt in     out     source               destination
    8   304 MY_FORWARD_RULES  all  --  *      *       0.0.0.0/0            0.0.0.0/0            stat
e NEW

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

Chain MY_ACCEPT_RULES (1 references)
 pkts bytes target     prot opt in     out     source               destination
    2    72 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain MY_DROP_RULES (1 references)
 pkts bytes target     prot opt in     out     source               destination
    6   232 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            recent: UPD
ATE seconds: 15 reap hit_count: 3 name: DROP_MARK side: source mask: 255.255.255.255
    2    72 MY_ACCEPT_RULES  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain MY_FORWARD_RULES (1 references)
 pkts bytes target     prot opt in     out     source               destination
    8   304 MY_DROP_RULES  all  --  *      *       0.0.0.0/0            0.0.0.0/0            recent:
 SET name: DROP_MARK side: source mask: 255.255.255.255
```

Figure 17 : Listing iptables rules.

Limitation :

False positives and negatives may occur, leading to the incorrect identification of legitimate traffic as malicious or the failure to detect certain malicious activities. Dynamic IP addresses could pose a challenge, especially in environments where clients frequently change IP addresses.

The implemented rules focus primarily on IP-based filtering and may not address vulnerabilities associated with specific protocols or application-layer attacks. If the router is a single point of failure, an attacker could potentially compromise it, bypassing the implemented rules.