Lindsay Graham

Firewalls

This document will outline a simple Iptables configuration for the Ubuntu and CentOS servers through a series of examples. Please note that **sudo** permissions are needed to make changes to the IPtables. The first section will explain each example's purpose and then show the scripts that were used to accomplish them. The second section will focus on preventing a Distributed Denial of Service (DDoS) attack. The final section will test connectivity on the ports that were added and blocked in the first section.

To ensure the IPtables rules save after a reboot on the Ubuntu server, you must make them persistent (Reynolds, 2016). This can be done by installing the iptables-persistent package with the command **sudo apt install iptables-persistent**. For the CentOS server, I had to install iptables services with the command **sudo yum install iptables-services**. Next, the iptables services need to be enabled and started with the commands **sudo systemctl start iptables** and **sudo systemctl enable iptables**.

Originally, I had only used commands for the INPUT chain. During testing I made a modification to add commands to allow outgoing traffic on the OUTPUT chain.

Example 1. *Deal with web server (open needed ports, and forward port 80 traffic to 8080)*

Ports 80 and 443 are necessary ports for communicating with a web server and allowing web traffic (Peppas, 2019). Port 80 is used for hypertext transfer protocol (HTTP) traffic, and port 443 is used for hypertext transfer protocol secure (HTTPS) traffic.

The command syntax to allow incoming traffic on port 80 and 443 on the Ubuntu server is:

sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

The command syntax to allow incoming on port 80 and 443 on the CentOS server is:

sudo iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT

sudo iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT

The command syntax to allow outgoing traffic on both servers is:

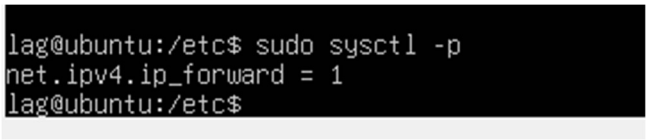
sudo iptables -A OUTPUT -p tcp --dport 8080-j ACCEPT

sudo iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT

The outgoing port is 8080, because that is where port 80 will be sent to, so that will be the port used for replies.

For the Ubuntu server, a rule had to be created to enable port forwarding. To do this, the sysctl.conf file in the etc directory had to be modified (Asghar, 2024). Navigate to the etc directory and open the sysctl.conf file with a text editor. Remember to open the file as sudo so that the changes you make can be saved. Next, add a line that says "net.ipv4.ip_forward=1" and save the file. After saving the changes, they must be applied. To apply the changes use the command **sudo sysctl -p**.

Figure A.1 Image of command to apply configuration changes



The command syntax to forward traffic from port 80 to 8080 on the Ubuntu server is:

sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination 10.0.0.49:8080

The IP address 10.0.0.49 is the Ubuntu server's external IP address.

The command syntax to forward traffic from port 80 to 8080 on the CentOS server is:

sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080

Example 2. *Deal with MySQL service (open needed ports)*

To allow MySQL traffic, port 3306 needed to be added (Oracle, 2024). Applications use this port to establish network connections with applications.

The command syntax to allow incoming traffic on port 3306 on the Ubuntu server is:

sudo iptables -A INPUT -p tcp --dport 3306 -j ACCEPT

The command syntax to allow incoming traffic on port 3306 on the CentOS server is:

sudo iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT

The command syntax to allow outgoing traffic on both servers is:

sudo iptables -A OUTPUT -p tcp --dport 3306 -j ACCEPT

Example 3. *Deal with SSH service (allow incoming and outgoing SSH, second script to deny SSH)*

The port that is used for Secure Shell (SSH) traffic that establishes secure connections to a remote device is port 22 (Peppas, 2019). The commands below show how to allow incoming and outgoing traffic on port 22. I also made a basic script to stop SSH for both servers (see Figures 1.3 and 2.3). The main difference in the command syntax is that instead of ACCEPT, DROP is used.

The command syntax to allow incoming and outgoing traffic on port 22 on the Ubuntu server is:

sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

sudo iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT

The command syntax to allow incoming and outgoing traffic on port 22 on the CentOS server is:

sudo iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT

sudo iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT

Example 4. *Script to allow/block specific hosts, MAC addresses*

There may be times when it can be helpful to block a particular media access control (MAC) address. The mac option can be used with a command to block all incoming traffic from a particular host if the MAC address is known (Gite, 2005b). I made a basic script block MAC address, 01:23:45:67:89:ab, and allow MAC address, 08:00:27:e6:5a:c3 to communicate. The blocked address is made up, and the allowed address is my Kali Linux virtual machine. See Figures 1.2 and 2.2 for the scripts.

Example 5. *Write a script or command to block telnet, and another one to block ping.*

The Telnet protocol is considered a security risk because the packets are sent as plain text, unencrypted, and susceptible to eavesdropping and/or man-in-the-middle among other attacks (Rasmussen Software, 2024). Telnet communicates on port 23. To block ping, there needs to be a rule to block Internet Control Message Protocol (ICMP) traffic (Gite, 2005a).

The command syntax to block Telnet and ping on both the Ubuntu and CentOS servers are:

sudo iptables -A INPUT -p tcp --dport 23 -j DROP

sudo iptables -A INPUT -p icmp -j DROP

Ubuntu Server Scripts

Figure 1.1 Script to configure IPtables

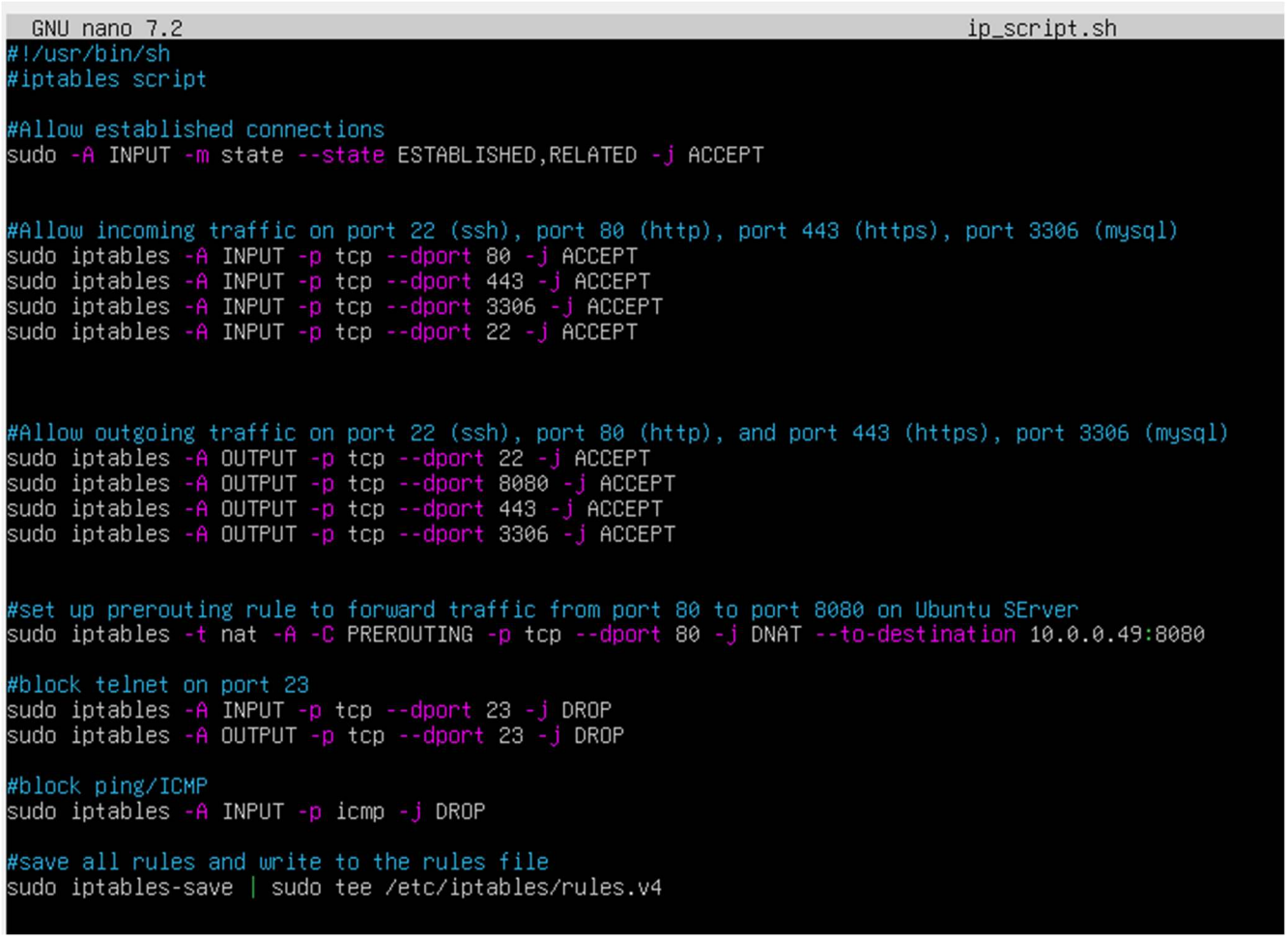


Figure 1.2 Script to allow or block a specific host

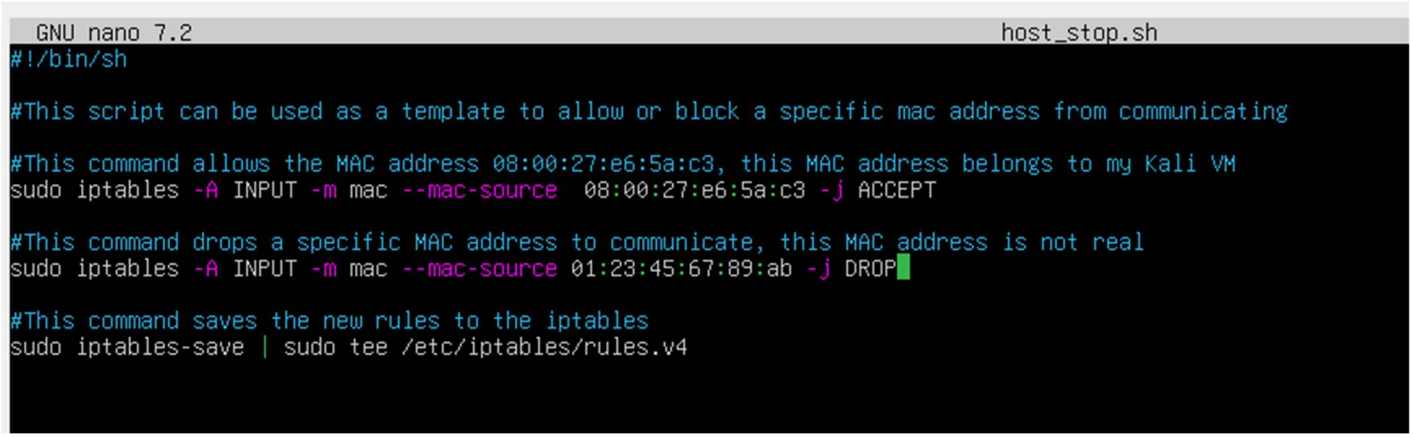


Figure 1.3 Script to stop SSH

```
  GNU nano 7.2                                                    stop_ssh.sh
#!/usr/bin/sh


#script to stop incoming ssh
sudo iptables -A INPUT -p tcp -m tcp --dport 22 -j DROP

#script to stop outgoing ssh
sudo iptables -A OUTPUT -p tcp -m tcp --dport 22 -j DROP

#save rules

sudo iptables-save | sudo tee /etc/iptables/rules.v4
```

Figure 1.4 Original IPtables before rules were added

```
lag@ubuntu:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
lag@ubuntu:~$
```

Figure1.5 IPtables after rules were added

```
lag@ubuntu:~/scripts$ sudo iptables -L -n
[sudo] password for lag:
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:80
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:443
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:3306
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
DROP       1    --  0.0.0.0/0            0.0.0.0/0
ACCEPT     0    --  0.0.0.0/0            0.0.0.0/0           MAC 08:00:27:e6:5a:c3
DROP       0    --  0.0.0.0/0            0.0.0.0/0           MAC 01:23:45:67:89:ab

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:8080
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:443
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:3306
DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
lag@ubuntu:~/scripts$ _
```

CentOS Server Scripts

Figure 2.1 Script to configure IPtables

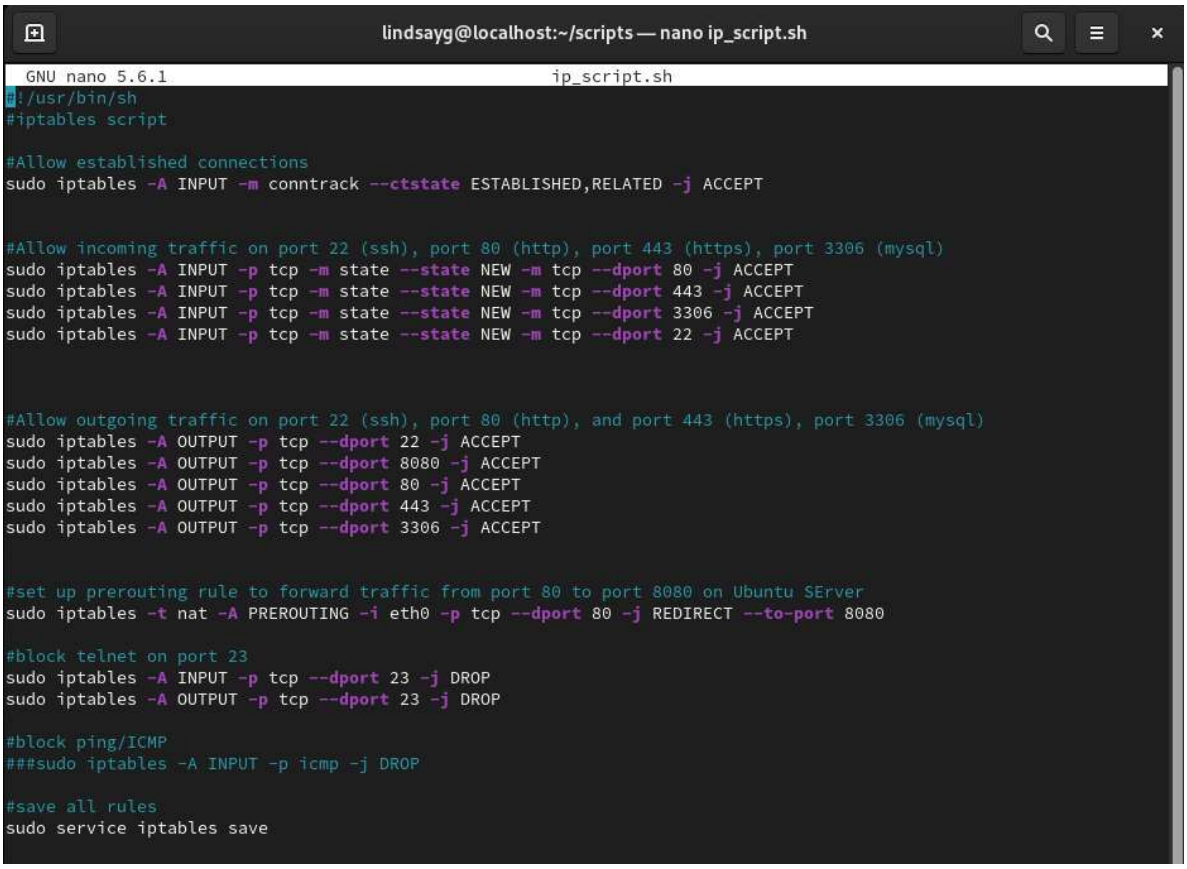


Figure 2.2 Script to allow or block a specific host

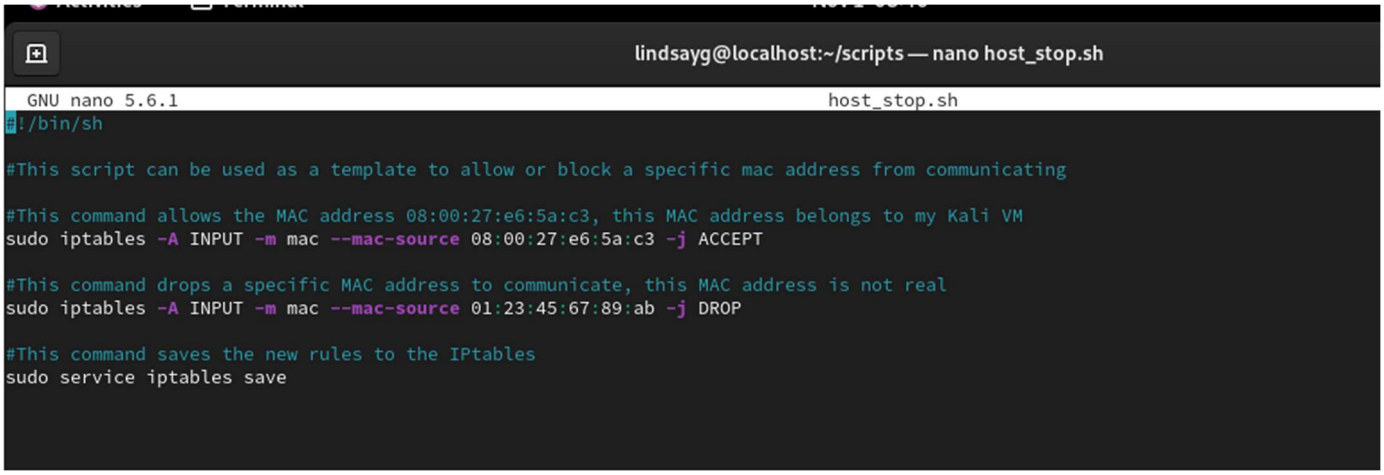


Figure 2.3 Image showing script that can be run to stop SSH on port 22

```
  GNU nano 7.2                                                    stop_ssh.sh
#!/usr/bin/sh


#script to stop incoming ssh
sudo iptables -A INPUT -p tcp -m tcp --dport 22 -j DROP

#script to stop outgoing ssh
sudo iptables -A OUTPUT -p tcp -m tcp --dport 22 -j DROP

#save rules

sudo iptables-save | sudo tee /etc/iptables/rules.v4
```

Figure 2.4 Original IPtables before rules were added

```
[lindsayg@localhost ~]$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[lindsayg@localhost ~]$
```

Figure 2.5 IPtables after rules were added

```
[lindsayg@localhost scripts]$ sudo iptables -L -n
[sudo] password for lindsayg:
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     0    --  0.0.0.0/0            0.0.0.0/0           ctstate RELATED,ESTABLISHED
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           state NEW tcp dpt:80
ACCEPT     6    --  9.9.9.9              0.0.0.0/0           state NEW tcp dpt:443
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           state NEW tcp dpt:3306
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22 state NEW,ESTABLISHED
DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
DROP       1    --  0.0.0.0/0            0.0.0.0/0
ACCEPT     0    --  0.0.0.0/0            0.0.0.0/0           MAC 08:00:27:e6:5a:c3
DROP       0    --  0.0.0.0/0            0.0.0.0/0           MAC 01:23:45:67:89:ab

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp spt:22 state ESTABLISHED
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:8080
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:443
ACCEPT     6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:3306
DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
[lindsayg@localhost scripts]$
```

Distributed Denial of Service (DDoS)

A Distributed Denial of Service (DDoS) attack is when an attacker floods a server with traffic from multiple sources with a goal of paralyzing the server so that is inaccessible (First2Host, 2024). Recently, “botnet-for-hire” programs have become popular with ransom-seeking hackers looking for an inexpensive and easy way to attack a target (Day, 2023). Botnets represent a mass of compromised hosts who the hacker can control and direct so that they can send the traffic simultaneously from different geographic locations. DDoS attacks, even those without the ransom element, can result in lost revenue and productivity that can have a significant negative impact on an organization. IPtables can be used as a tool to help block some of the traffic types that are often associated with DDoS attacks. The next few paragraphs will detail a basic script that contains commands to block traffic that is often associated DDoS attacks.

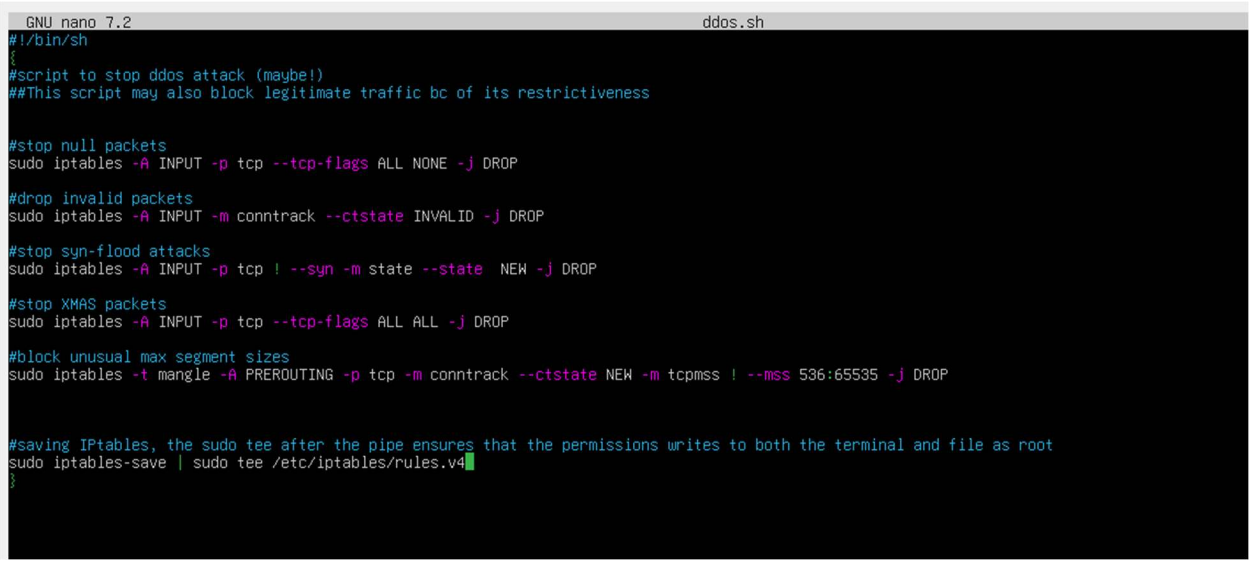
The first line in the script focuses on null packets. In DDoS attacks, bots are often used to send null packets in an attempt to identify firewall vulnerabilities (First2Host, 2024). A flag-less packet would be considered a null packet, so the command sudo iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP adds a rule that will block all null TCP packets. The second line focuses on dropping invalid, or non-standard packets which could signify an attack (Todorov, 2016). The command to drop invalid packets is sudo iptables -A INPUT -m conntrack --ctstate INVALID -j DROP.

The third line focuses on stopping SYN-Flood attacks. SYN-Flood attacks are a type of attack where a hacker connects to a server without sending or receiving any information so that they can consume resources that affect system availability and performance (First2Host, 2024). The IPtables command to stop this type of attack is sudo iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP. The fourth line focuses on stopping XMAS packets. XMAS packets are not beautifully wrapped gifts, but instead malformed data packets that are often present in DDoS attacks. The command to stop XMAS packets is sudo iptables -A INPUT -p tcp –-tcp-flags ALL ALL -j DROP. The last command blocks packets that have unusual max segment sizes through the command sudo iptables -t mangle -A -C PREROUTING -p tcp -m conntrack --ctstate NEW -m tcpmss ! --mss 536:65535 -j DROP.

The script also has a command to save the iptables and write them to the configuration file, rules.v4. I used variables to store each of the commands in the script for readability, and simplicity should this script get more advanced.

Ubuntu Server

Figure 1.6 DDoS script for Ubuntu



```
  GNU nano 7.2                                                    ddos.sh
#!/bin/sh
{
#script to stop ddos attack (maybe!)
##This script may also block legitimate traffic bc of its restrictiveness


#stop null packets
sudo iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP

#drop invalid packets
sudo iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

#stop syn-flood attacks
sudo iptables -A INPUT -p tcp ! --syn -m state --state  NEW -j DROP

#stop XMAS packets
sudo iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP

#block unusual max segment sizes
sudo iptables -t mangle -A PREROUTING -p tcp -m conntrack --ctstate NEW -m tcpmss ! --mss 536:65535 -j DROP



#saving IPtables, the sudo tee after the pipe ensures that the permissions writes to both the terminal and file as root
sudo iptables-save | sudo tee /etc/iptables/rules.v4
}
```

Figure 1.7 IPtables after script is run

```
lag@ubuntu:~/scripts$ ./ddos.sh
# Generated by iptables-save v1.8.10 (nf_tables) on Fri Nov  1 18:41:22 2024
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A PREROUTING -p tcp -m conntrack --ctstate NEW -m tcpmss ! --mss 536:65535 -j DROP
COMMIT
# Completed on Fri Nov  1 18:41:22 2024
# Generated by iptables-save v1.8.10 (nf_tables) on Fri Nov  1 18:41:22 2024
*filter
:INPUT ACCEPT [2125:1919442]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [213:16800]
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 3306 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 23 -j DROP
-A INPUT -p icmp -j DROP
-A INPUT -m mac --mac-source 08:00:27:e6:5a:c3 -j ACCEPT
-A INPUT -m mac --mac-source 01:23:45:67:89:ab -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -j DROP
-A INPUT -m conntrack --ctstate INVALID -j DROP
-A INPUT -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,PSH,ACK,URG -j DROP
-A OUTPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 8080 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 3306 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 23 -j DROP
COMMIT
# Completed on Fri Nov  1 18:41:22 2024
# Generated by iptables-save v1.8.10 (nf_tables) on Fri Nov  1 18:41:22 2024
*nat
:PREROUTING ACCEPT [664:62155]
:INPUT ACCEPT [659:61042]
:OUTPUT ACCEPT [65:4659]
:POSTROUTING ACCEPT [65:4659]
-A PREROUTING -p tcp -m tcp --dport 80 -j DNAT --to-destination 10.0.0.49:8080
COMMIT
# Completed on Fri Nov  1 18:41:22 2024
```

CentOS Server

Figure 2.6 DDoS script for CentOS

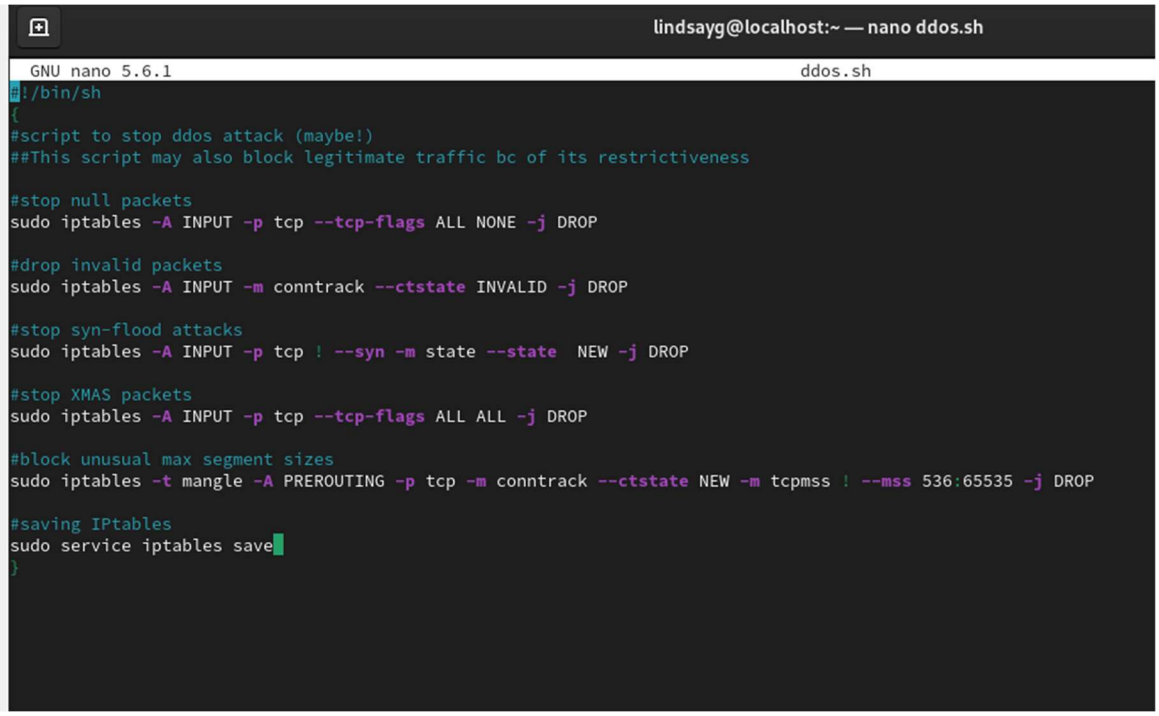
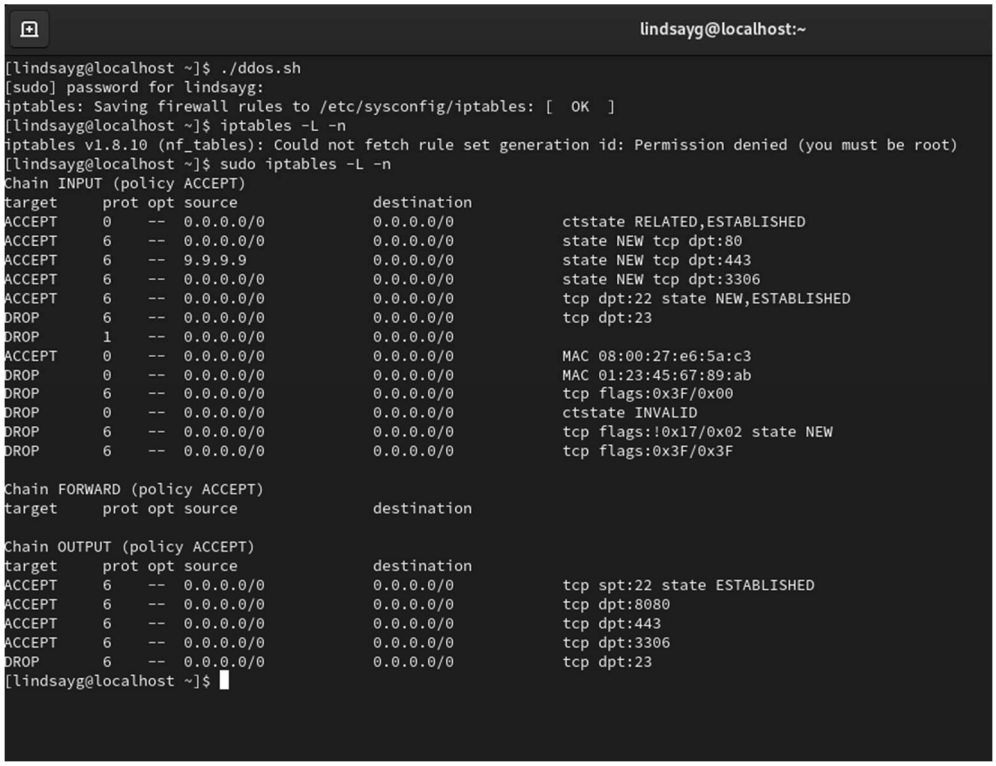


Figure 2.7 Iptables after script has run



Testing

Ubuntu Server

Tshark, the command line version of Wireshark was used to test the newly added rules. The command to add tshark was **sudo apt install tshark**. I used a Kali Linux virtual machine to test the rules.

All ICMP traffic should be blocked, so this is a test to see if the ping/ICMP will be successful. As the output in Figure 7.2 shows, there was 100% packet loss illustrating the ping was unsuccessful. ICMP uses port 23.

Ping - Unsuccessful

Figure 1.8 Results of the ping request from the Kali system

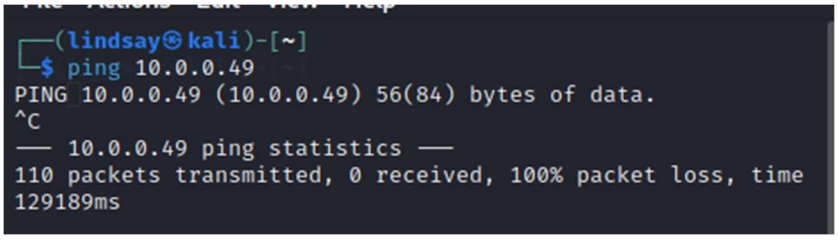
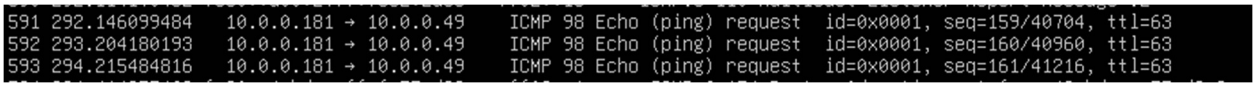


Figure 1.9 Tshark output from the Ubuntu server showing the unsuccessful ping request



Next, I used Netcat from the Kali system to test ports 443, 3306, 80, 8080, and 22. I had to set up a Netcat listener on the Ubuntu Server. The command to set up a listener for a specific port is **netcat 10.0.0.49 "port number**" (thandel, 2019). I was unable to run Tshark in the same terminal as the listener, so I used Tmux with Tshark in one terminal with the listener in the main terminal. I knew that Tmux had that ability, but it was my first time using it. To start a Tmux session you can just type tmux. To detach from a session you select Ctrl+B then D. To reattach, you use the command tmux attach -d -t "session number" (StackOverflow, 2014).

Port 443 – Successful Connection

Figure 1.10 Image of Netcat command on Kali Linux system

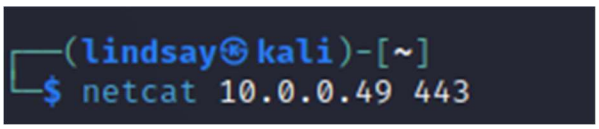


Figure 1.11 Tshark capture showing the successful connection to port 443

```
  1 0.000000000    10.0.0.181 → 10.0.0.49    TCP 68 57325 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2 0.000040083     10.0.0.49 → 10.0.0.181   TCP 68 443 → 57325 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
3 0.000608083    10.0.0.181 → 10.0.0.49    TCP 62 57325 → 443 [ACK] Seq=1 Ack=1 Win=131328 Len=0
4 7.061136149     10.0.0.49 → 10.0.0.181   TCP 56 443 → 57325 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
5 7.061761173    10.0.0.181 → 10.0.0.49    TCP 62 57325 → 443 [ACK] Seq=1 Ack=2 Win=131328 Len=0
6 7.063570433    10.0.0.181 → 10.0.0.49    TCP 62 57325 → 443 [FIN, ACK] Seq=1 Ack=2 Win=131328 Len=0
7 7.063599259     10.0.0.49 → 10.0.0.181   TCP 56 443 → 57325 [ACK] Seq=2 Ack=2 Win=64256 Len=0
```

Port 3306 – Successful Connection

Figure 1.12 Image of Netcat command on Kali Linux system

```
┌──(lindsay㉿kali)-[~]
└─$ netcat 10.0.0.49 3306
```

Figure 1.13 Tshark capture showing the successful connection to port 3306

```
lag@ubuntu:~/scripts$ sudo tshark -i any -f "port 3306"
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
    1 0.000000000    10.0.0.181 → 10.0.0.49    TCP 68 57371 → 3306 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    2 0.000057325     10.0.0.49 → 10.0.0.181   TCP 68 3306 → 57371 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
    3 0.000619878    10.0.0.181 → 10.0.0.49    TCP 62 57371 → 3306 [ACK] Seq=1 Ack=1 Win=131328 Len=0
    4 3.366905007     10.0.0.49 → 10.0.0.181   TCP 56 3306 → 57371 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
    5 3.367142442    10.0.0.181 → 10.0.0.49    TCP 62 57371 → 3306 [ACK] Seq=1 Ack=2 Win=131328 Len=0
    6 3.367853315    10.0.0.181 → 10.0.0.49    TCP 62 57371 → 3306 [FIN, ACK] Seq=1 Ack=2 Win=131328 Len=0
    7 3.367867262     10.0.0.49 → 10.0.0.181   TCP 56 3306 → 57371 [ACK] Seq=2 Ack=2 Win=64256 Len=0
^C7 packets captured
```

Port 80 – Connection Refused

Figure 1.14 Image of Netcat command on Kali Linux system

```
┌──(lindsay㉿kali)-[~]
└─$ netcat 10.0.0.49 80
(UNKNOWN) [10.0.0.49] 80 (http) : Connection refused

┌──(lindsay㉿kali)-[~]
```

Figure 1.15 Tshark capture showing the unsuccessful connection to port 80

```
lag@ubuntu:~/scripts$ sudo tshark -i any -f "port 80"
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
    1 0.000000000    10.0.0.181 → 10.0.0.49    TCP 68 57398 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    2 0.000064834     10.0.0.49 → 10.0.0.181   TCP 56 80 → 57398 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
    3 0.501735701    10.0.0.181 → 10.0.0.49    TCP 68 [TCP Port numbers reused] 57398 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    4 0.501779235     10.0.0.49 → 10.0.0.181   TCP 56 80 → 57398 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
    5 1.002986763    10.0.0.181 → 10.0.0.49    TCP 68 [TCP Port numbers reused] 57398 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    6 1.003031892     10.0.0.49 → 10.0.0.181   TCP 56 80 → 57398 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
    7 1.504674291    10.0.0.181 → 10.0.0.49    TCP 68 [TCP Port numbers reused] 57398 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    8 1.504738342     10.0.0.49 → 10.0.0.181   TCP 56 80 → 57398 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
    9 2.005793935    10.0.0.181 → 10.0.0.49    TCP 68 [TCP Port numbers reused] 57398 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
   10 2.005834736     10.0.0.49 → 10.0.0.181   TCP 56 80 → 57398 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
^C10 packets captured
```

Note – I think connection to port 80 was unsuccessful because of the PREROUTING rule that directs all traffic for port 80 to port 8080.

Port 8080 – Successful Connection

Figure 1.16 Example of setting up the Netcat listener for port 8080

```
lag@ubuntu:~/scripts$ sudo nc -lvp 8080
Listening on 0.0.0.0 8080
Connection received on 10.0.0.181 57406
^C
lag@ubuntu:~/scripts$
```

Figure 1.17 Image of Netcat command on Kali Linux system

```
┌──(lindsay㉿kali)-[~]
└─$ netcat 10.0.0.49 8080
```

Figure 1.18 Tshark capture showing the successful connection to port 8080

```
lag@ubuntu:~/scripts$ sudo tshark -i any -f "port 8080"
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
    1 0.000000000   10.0.0.181 → 10.0.0.49    TCP 68 57435 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    2 0.000047386    10.0.0.49 → 10.0.0.181   TCP 68 8080 → 57435 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
    3 0.000324978   10.0.0.181 → 10.0.0.49    TCP 62 57435 → 8080 [ACK] Seq=1 Ack=1 Win=131328 Len=0
    4 2.802502153    10.0.0.49 → 10.0.0.181   TCP 56 8080 → 57435 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
    5 2.803008019   10.0.0.181 → 10.0.0.49    TCP 62 57435 → 8080 [ACK] Seq=1 Ack=2 Win=131328 Len=0
    6 2.803671153   10.0.0.181 → 10.0.0.49    TCP 62 57435 → 8080 [FIN, ACK] Seq=1 Ack=2 Win=131328 Len=0
    7 2.803700859    10.0.0.49 → 10.0.0.181   TCP 56 8080 → 57435 [ACK] Seq=2 Ack=2 Win=64256 Len=0
```

Port 22 – Successful Connection

Figure 1.19 Image of Netcat command on Kali Linux system

```
┌──(lindsay㉿kali)-[~]
└─$ netcat 10.0.0.49 22
SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5
```

Figure 1.20 Tshark capture showing the successful connection to port 22

```
lag@ubuntu:~/scripts$ sudo tshark -i any -f "port 22"
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
    1 0.000000000   10.0.0.181 → 10.0.0.49    TCP 68 57459 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    2 0.000069229    10.0.0.49 → 10.0.0.181   TCP 68 22 → 57459 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
    3 0.000367511   10.0.0.181 → 10.0.0.49    TCP 62 57459 → 22 [ACK] Seq=1 Ack=1 Win=131328 Len=0
    4 0.051968477    10.0.0.49 → 10.0.0.181   SSH 98 Server: Protocol (SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5)
    5 0.093852575   10.0.0.181 → 10.0.0.49    TCP 62 57459 → 22 [ACK] Seq=1 Ack=43 Win=131328 Len=0
^C5 packets captured
lag@ubuntu:~/scripts$
```

CentOS Server

For the CentOS server, I used the Wireshark GUI for testing. First, I had to install it using the command **sudo yum install wireshark**. I used a Kali Linux virtual machine to test the rules.

All ICMP traffic should be blocked, so this is a test to see if the ping/ICMP will be successful. As the output in Figure 7.2 shows, there was 100% packet loss illustrating the ping was unsuccessful. ICMP uses port 23.

Figure 2.8 Results of the ping request from the Kali system

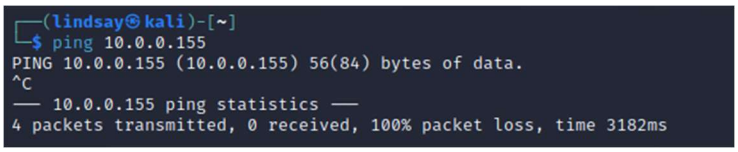


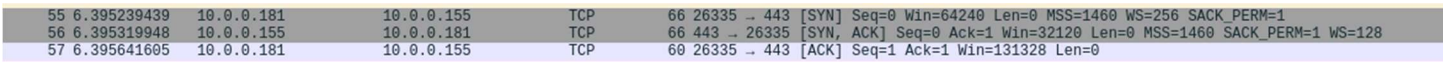
Figure 2.9 Wireshark output showing the unsuccessful ping request

Next, I used Netcat from the Kali system to test ports 443, 3306, 80, 8080, and 22. I had to set up a Netcat listener on the CentOS Server. The command to set up a listener for a specific port is **nc -lvp port number** (thandel, 2019).

Port 443 – Successful Connection

Figure 2.10 Image of Netcat command on Kali Linux system

Figure 2.11 Wireshark capture showing the successful connection to port 443



Port 3306 – Successful Connection

Figure 2.12 Image of Netcat command on Kali Linux system

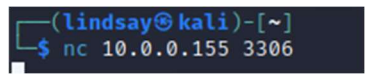
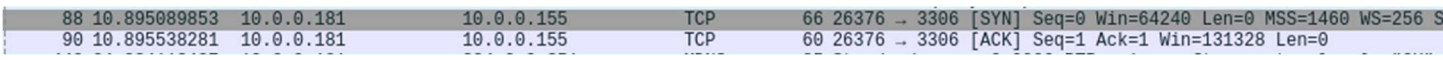


Figure 2.13 Wireshark capture showing the successful connection to port 3306



Port 80 – Successful Connection

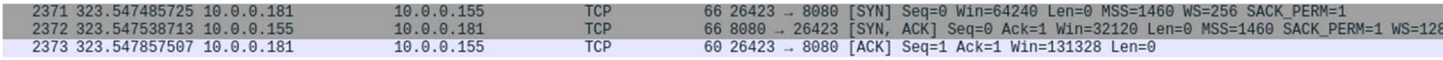
Figure 2.14 Image of Netcat command on Kali Linux system

Figu re 2.15 Wireshark capture showing the successful connection to port 80

Port 8080 – Successful Connection

Figure 2.17 Image of Netcat command on Kali Linux system

Figure 2.18 Wireshark capture showing the successful connection to port 8080



Port 22 – Successful Connection

Figure 2.19 Image of Netcat command on Kali Linux system

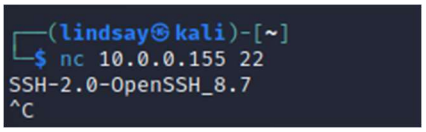
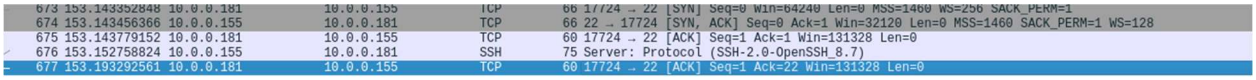


Figure 2.20 Wireshark capture showing the successful connection to port 22



References

Asghar, A. (2024, September 16). *How to Forward Ports With Iptables in Linux*. Ultahost Knowledge Base. https://ultahost.com/knowledge-base/forward-ports-iptables-linux/

Day, B. (2023, August 14). *Preventing Linux DDoS Attacks with Minimal Cybersecurity Knowledge ...* Linux Security. https://linuxsecurity.com/features/preventing-linux-ddos-attacks

First2Host. (2024, February 6). *How to use IPtables to stop DDOS attacks*. First2Host. https://first2host.co.uk/blog/iptables-stop-common-ddos-attacks/

Gite, V. (2005a, June 28). *Linux Iptables allow or block ICMP ping request*. NixCraft. https://www.cyberciti.biz/tips/linux-iptables-9-allow-icmp-ping.html

Gite, V. (2005b, December 27). *Iptables MAC Address Filtering*. NixCraft. https://www.cyberciti.biz/tips/iptables-mac-address-filtering.html

Oracle. (2024). *MySQL :: MySQL Port Reference :: 3 MySQL Port Reference Tables*. Dev.mysql.com. https://dev.mysql.com/doc/mysql-port-reference/en/mysql-port-reference-tables.html

Peppas, N. (2019, May 7). *Liquid Web*. Liquid Web. https://www.liquidweb.com/blog/set-firewall-using-iptables-ubuntu-16-04/

Rasmussen Software. (2024). *A Guide to Telnet*. Anzio.com. https://www.anzio.com/resources/guide-telnet

Reynolds, L. (2016, August 28). *How to make iptables persistent after reboot on Linux*. LinuxConfig. https://linuxconfig.org/how-to-make-iptables-rules-persistent-after-reboot-on-linux

StackOverflow. (2014, May 1). *How can I reattach to tmux process*. Stack Overflow. https://stackoverflow.com/questions/23403368/how-can-i-reattach-to-tmux-process

thandel, lucky. (2019, May 3). *How To Use NetCat Tool In Linux - Technical Navigator*. Technical Navigator. https://technicalnavigator.in/how-to-use-netcat-tool-in-linux/

Todorov, M. (2016, March 1). *25 Useful IPtable Firewall Rules Every Linux Administrator Should Know*. Www.tecmint.com. https://www.tecmint.com/linux-iptables-firewall-rules-examples-commands/