Firewall

▼ **Preparation**

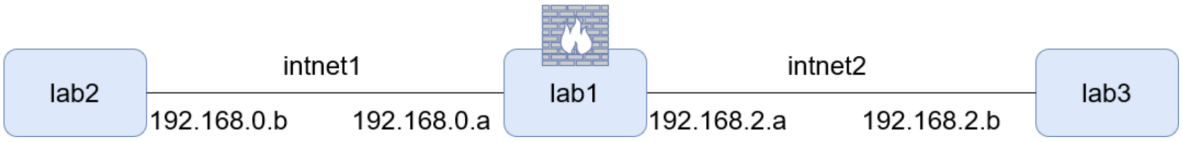
If you are doing both paths, you might want to consider making new virtual machines for this exercise, because the two assignments might cause some conflicts or problems.

You will need all three virtual machines for this exercise. Lab1 functions as a router/firewall between lab2 and lab3, which are in different subnetworks. The enp0s3 interface allows access to the virtual machines. Be careful not to modify it or block access to it. Make sure you are not sending packets through enp0s3 when connecting to other virtual machines, because that way you will bypass the firewall. The communication between VMs should be through the internal networks.

Please remember to take backups of the folders you have modified on the virtual machines.

▼ **Set up the network**

You will configure lab1 to act as a router between lab2 and lab3. The resulting network should look like the following:



vagrant file

All machines:

Update and upgrade:

```
sudo apt-get update
sudo apt-get upgrade -y
```

Lab1:

Enable forwarding and arp proxying on **lab1** for the enp0s8 and enp0s9 interfaces.

```
sudo sysctl -w net.ipv4.conf.enp0s8.forwarding=1
sudo sysctl -w net.ipv4.conf.enp0s9.forwarding=1
sudo sysctl -w net.ipv4.conf.enp0s8.proxy_arp=1
sudo sysctl -w net.ipv4.conf.enp0s9.proxy_arp=1
```

Lab2:

add route to lab3

```
sudo ip route add 192.168.2.0/24 via 192.168.0.10 dev enp0s8
```

Lab3:

add route to lab2

```
sudo ip route add 192.168.0.0/24 via 192.168.2.10 dev enp0s8
```

Lab1

Check that there is no firewall rules at this point (*iptables -L*)

```
sudo iptables -L
```

```
vagrant@lab1:~$ 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
```

Lab2:

```
sudo apt-get install traceroute
```

2.1 List all commands you used to create the router setup, and briefly explain what they do. Show the results of the traceroute as well.

route setup

Lab1:

enable port forwarding and proxy arp

```
sudo sysctl -w net.ipv4.conf.enp0s8.forwarding=1
sudo sysctl -w net.ipv4.conf.enp0s9.forwarding=1
sudo sysctl -w net.ipv4.conf.enp0s8.proxy_arp=1
sudo sysctl -w net.ipv4.conf.enp0s9.proxy_arp=1
```

enable IP packet forwarding

Lab2:

add route to lab3 via lab1

```
sudo ip route add 192.168.2.0/24 via 192.168.0.10 dev enp0s8
```

Lab3:

add route to lab2 via lab1

```
sudo ip route add 192.168.0.0/24 via 192.168.2.10 dev enp0s8
```

Lab2:

traceroute

```
traceroute lab3
```

```
vagrant@lab2:~$ traceroute lab3
traceroute to lab3 (192.168.2.30), 30 hops max, 60 byte packets
 1  lab1 (192.168.0.10)  6.657 ms  7.416 ms  9.101 ms
 2  lab3 (192.168.2.30)  15.292 ms  18.238 ms  18.230 ms
```

path taken by packets from one device to another.

2.2 Explain Tables ,chains, hooks and rules in nftables?

```
vagrant@lab1:~$ 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
```

nftables

1. packet filtering framework
2. used to classify and handle network traffic on Linux systems

In nftables, a packet filtering configuration consists of tables, chains, hooks, and rules.

1. **Tables: (table hold chain)**
    a. Top-level containers
    b. hold the different sets of rules
    c. analogous to chains in iptables.
    d. Can have multiple chains, and each chain can contain multiple rules
    e. e.g filter (for packet filtering), nat (for network address translation)
2. **Chains: (chain hold rule)**
    a. table can have multiple chains.
    b. organize the rules into different categories
    c. e.g input (for incoming packets), output (for outgoing packets), and forward (for forwarded packets).
3. **Hooks:**
    a. entry and exit points of the packet filtering
    b. determine when a rule should be applied to a packet
    c. determine the direction of the packet flow (e.g., incoming or outgoing)
    d. e.g prerouting, input, forward, output, and postrouting.

4. **Rules:**
    a. lowest level in the nftables hierarchy
    b. define the specific conditions that a packet must meet in order to be processed or forwarded
    c. Rules can specify various conditions such as source and destination IP addresses, protocol types, ports, and more.

▼ **Implement packet filtering on the router**

First, scan **lab3** from **lab2** and vice versa with *nmap(1)* to see what services they are running. Try to gather as much information on the machine as feasible, including information about software versions and the operating system.

Set up an *nftables(8)* FORWARD policy to disallow traffic through the router by default. Add rules to allow *ping(8)* from **lab2** on the enp0s8 interface and replies to **lab2**. Change rules only for the FORWARD hook! Once this is working, expand the ruleset to allow SSH connections to and from **lab2**. Also allow browsing the web and transferring files via FTP (both active and passive modes) from **lab2**. Set up a web server (e.g. *httpd(8)*) and an ftp server (e.g. *proftpd(8)*) on **lab3** for testing. Use as restricting ruleset as possible while allowing full functionality. You will probably need the "ip_conntrack_ftp" kernel module for FTP filtering. Load it with *modprobe(8)*.

Finally, rescan **lab3** from **lab2** and vice versa.

Lab2 and lab3:

```
sudo apt-get install nmap -y
```

Lab2:

```
nmap -A lab3
//-A enables OS detection , Version detection of services,
// Script scanning to gather additional information
```

```
vagrant@lab2:~$ nmap -A lab3
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-28 23:00 UTC
Nmap scan report for lab3 (192.168.2.30)
Host is up (0.025s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.32 seconds
```

Lab3:

```
nmap -A lab2
```

```
vagrant@lab3:~$ nmap -A lab2
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-28 23:01 UTC
Nmap scan report for lab2 (192.168.0.20)
Host is up (0.020s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 1.14 seconds
```

Lab1:

```
sudo apt-get install nftables -y
```

<info>

three main chains in each table

INPUT chain: processes incoming packets that are destined for the local system

OUTPUT chain: processes outgoing packets that originate from the local system

FORWARD chain: processes packets that are forwarded from one network interface to another

Set up an *nftables(8)* FORWARD policy to disallow traffic through the router by default.

// by doing this we wont be able to ping lab3 from lab2 even with the routes added

Add the rules to the config file:

```
sudo nano  /etc/nftables.conf
```

contents

1. FORWARD chain to filter table
2. type "filter"
3. hooked to the FORWARD hook - all packets that are forwarded through the router
4. priority 0 - will be processsed first most
5. policy drop - all packets forwarded will be dropped unless it matches a rule

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
        chain input {
                type filter hook input priority 0;
        }
        chain forward {
                type filter hook forward priority 0; policy drop;

        }
        chain output {
                type filter hook output priority 0;
        }
}
```

To check if the file format is correct:

```
sudo nft --check --file /etc/nftables.conf
```

reload rule file

```
sudo nft -f /etc/nftables.conf
```

```
sudo nft list ruleset
```

```
vagrant@lab1:~$ sudo nft list ruleset
table inet filter {
        chain input {
                type filter hook input priority filter; policy accept;
        }

        chain forward {
                type filter hook forward priority filter; policy drop;
        }

        chain output {
                type filter hook output priority filter; policy accept;
        }
}
```

Add rules to allow *ping(8)* from **lab2** on the enp0s8 interface and replies to **lab2**. Change rules only for the FORWARD hook!

Add the rules to the config file:

```
sudo nano  /etc/nftables.conf
```

contents

#ping

iif "enp0s8" ip saddr lab2 icmp type echo-request counter accept
oif "enp0s8" ip daddr lab2 icmp type echo-reply counter accept

1. iif - input interface s8
2. oif - output interface s8
3. ip saddr lab2 - source ip address of lab2
4. ip daddr lab2 - destination ip address of lab2
5. icmp type of echo-request / reply
6. counter - track of the number of packets
7. accept - action to take for the packets matching the rule

```
#!/usr/sbin/nft -f
```

```
flush ruleset

table inet filter {
        chain input {
                type filter hook input priority 0;
        }
        chain forward {
                type filter hook forward priority 0; policy drop;

                #ping
                iif "enp0s8" ip saddr lab2 icmp type echo-request counter accept
                oif "enp0s8" ip daddr lab2 icmp type echo-reply counter  accept
        }
        chain output {
                type filter hook output priority 0;
        }
}
```

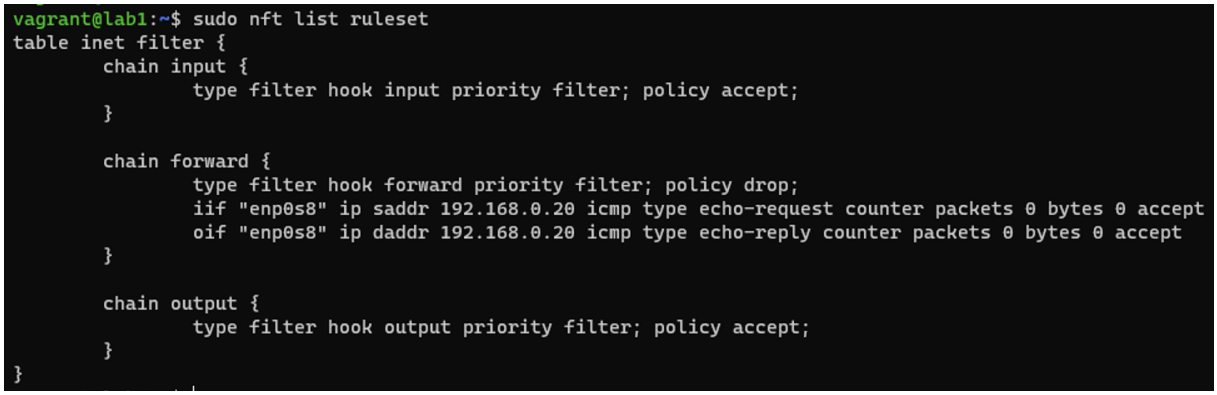
To check if the file format is correct:

```
sudo nft --check --file /etc/nftables.conf
```

reload rule file

```
sudo nft -f /etc/nftables.conf
```

```
sudo nft list ruleset
```



Lab2:

Test lab2 can ping lab3

```
vagrant@lab2:~$ ping -c 3 lab3
PING lab3 (192.168.2.30) 56(84) bytes of data.
64 bytes from lab3 (192.168.2.30): icmp_seq=1 ttl=63 time=2.99 ms
64 bytes from lab3 (192.168.2.30): icmp_seq=2 ttl=63 time=2.84 ms
64 bytes from lab3 (192.168.2.30): icmp_seq=3 ttl=63 time=2.43 ms

--- lab3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 2.432/2.753/2.993/0.236 ms
```

Lab3:

Now lab3's packets are not being forwarded:

```
vagrant@lab3:~$ ping lab2
PING lab2 (192.168.0.20) 56(84) bytes of data.
^C
--- lab2 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5180ms
```

Lab1:

Once this is working, expand the ruleset to allow SSH connections to and from lab2.

from:

Modify the rule to the config file:

```
sudo nano  /etc/nftables.conf
```

contents

1. iif - input interface s8
2. ip saddr lab2 - source ip lab2
3. tcp dport ssh - destination port ssh
4. ct state established, related, new - connection tracking state of established, related, new
5. counter - track of the number of packets
6. accept - action to take for packets that match the rule, accept and allow them through the firewall.

same for other ssh rules

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
        ...
        chain forward {
                ...
                #ssh
                #from
                iif "enp0s8" ip saddr lab2 tcp dport ssh ct state established,related,new counter accept
                oif "enp0s8" ip daddr lab2 tcp sport ssh ct state established,related counter accept
                #to
                iif "enp0s8" ip saddr lab2 tcp sport ssh ct state established,related counter accept
                oif "enp0s8" ip daddr lab2 tcp dport ssh ct state established,related,new counter accept
```

```
                ...
        }
        ...
}
```

To check if the file format is correct:

```
sudo nft --check --file /etc/nftables.conf
```

reload rule file

```
sudo nft -f /etc/nftables.conf
```

To test ssh **FROM** lab2:

Lab3:

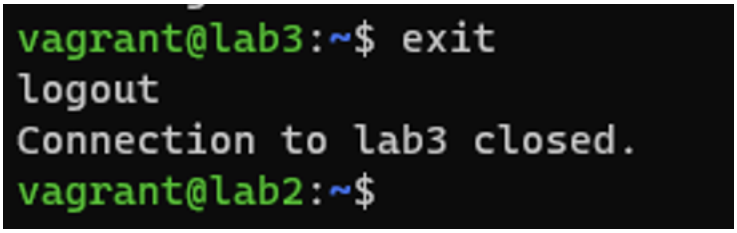
```
sudo nano /etc/ssh/sshd_config
```

//Set Password Authentication to yes

```
sudo systemctl restart sshd
```

Lab2:

```
ssh vagrant@lab3
```



To test ssh **To** lab2:

Lab2:

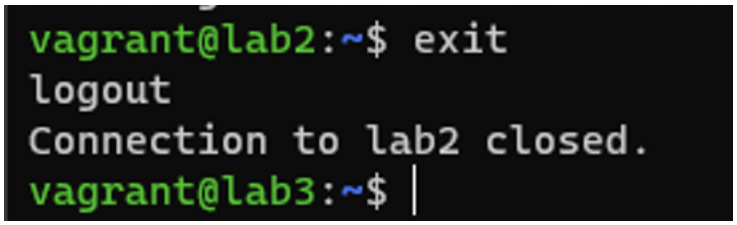
```
sudo nano /etc/ssh/sshd_config
```

//Set Password Authentication to yes

```
sudo systemctl restart sshd
```

Lab3:

```
ssh vagrant@lab2
```



Lab1:

Also allow browsing the web and transferring files via FTP (both active and passive modes) from **lab2.**

Browsing:

Modify the rules in the config file:

```
sudo nano  /etc/nftables.conf
```

contents

these rules allow HTTP and HTTPS traffic for browsing from the lab2

1. iifname - input interface s8
2. ip saddr lab2 - source ip lab2
3. tcp dport {http, https} - destination port http/https
4. ct state established, related, new - connection tracking state of established, related, new
5. counter - track of the number of packets
6. accept - action to take for packets that match the rule, accept and allow them through the firewall.

//same for other rules

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
        ...
        chain forward {
                ...
                #browsing
                iifname "enp0s8" ip saddr lab2 tcp dport { http, https } ct state established,related,new counter accept
                oifname "enp0s8" ip daddr lab2  tcp sport { http, https } ct state established,related counter accept
        }
        ...
}
```

FTP:

contents

These rules allow incoming and outgoing traffic on the enp0s8 interface for the following ports:

80 (HTTP)

443 (HTTPS)

20 (FTP Data)

21 (FTP Control)

1024-65535 (Passive FTP Data)

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
        ...
        chain forward {
                ...
                #ftp
                iifname "enp0s8" ip saddr lab2 tcp dport ftp ct state established,new counter accept
                oifname "enp0s8" ip daddr lab2 tcp sport ftp ct state established counter accept
                #Additional rule for active FTP
                oifname "enp0s8" ip daddr lab2 tcp sport ftp-data ct state established,related counter accept
                iifname "enp0s8" ip saddr lab2 tcp dport ftp-data ct state established counter accept
                #Additional rule for passive FTP
                iifname "enp0s8" ip saddr lab2 tcp dport 1024-65535 ct state established,related counter  accept
                oifname "enp0s8" ip daddr lab2 tcp sport 1024-65535 ct state established counter accept
        }
        ...
}
```

To check if the file format is correct:

```
sudo nft --check --file /etc/nftables.conf
```

reload rule file

```
sudo nft -f /etc/nftables.conf
```

<info>

In active mode, the FTP client initiates a data connection to the FTP server's port 20, while the server initiates a control connection to the client's port 21. This mode is sometimes called "PORT mode."

In passive mode, the client initiates both the control and data connections to the server. The server opens a new port to transfer data, and the client connects to that port to receive the data. This mode is sometimes called "PASV mode."

In summary, active mode requires the FTP server to initiate the data connection, while passive mode requires the FTP client to initiate the data connection.

Lab3:

Set up a web server (e.g. *httpd(8)*) and an ftp server (e.g. *proftpd(8)*) on **lab3** for testing.

install apache2 // server

```
sudo apt-get install apache2 -y
```

start apache

```
sudo systemctl start apache2
```

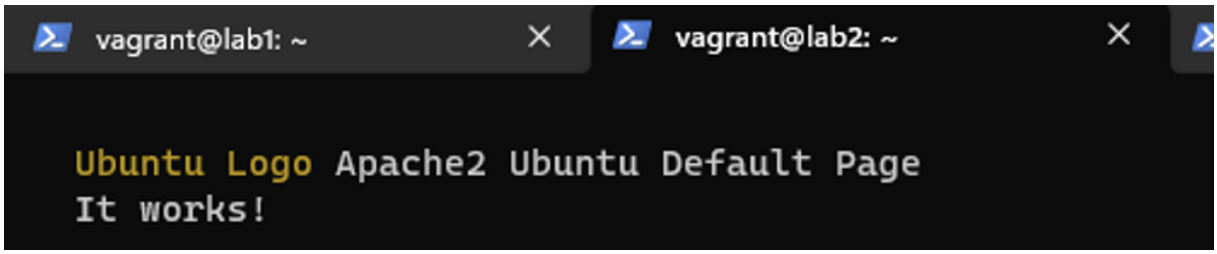
check

```
sudo systemctl status apache2
```

Lab2: test browsing the web

```
sudo apt-get install lynx -y
```

```
lynx http://lab3
```



Lab3:

Install ProFTPD // ftp server

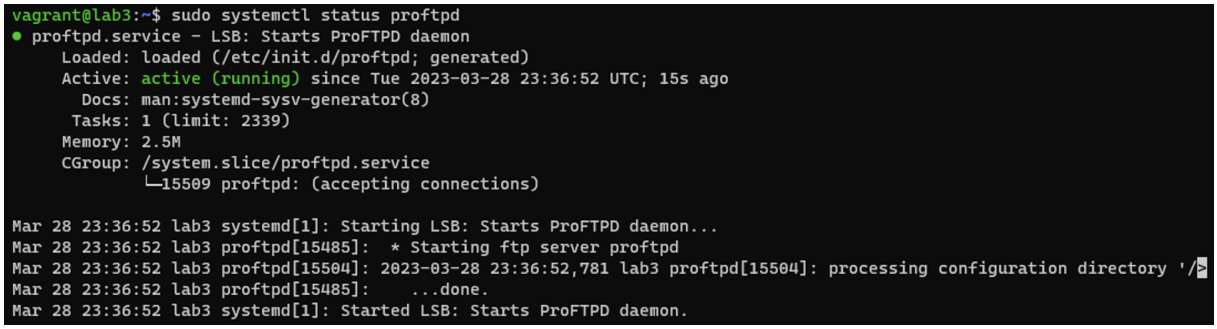
```
sudo apt-get install proftpd -y
```

start

```
sudo systemctl start proftpd
```

check the status of the ProFTPD

```
sudo systemctl status proftpd
```



Lab1:

You will probably need the "ip_conntrack_ftp" kernel module for FTP filtering. Load it with *modprobe(8)*

```
sudo modprobe ip_conntrack_ftp
```

```
sudo sysctl -w net.netfilter.nf_conntrack_helper=1
```

Lab3:

create testfile

```
nano testlab3.txt
```

// This is a file in lab3

Lab2:

```
sudo nano testlab2.txt
```

// This is a file in lab2

From lab2

```
ftp lab3
```

```
ls
```

```
put /home/vagrant/testlab2.txt
```

```
vagrant@lab2:~$ ftp lab3
Connected to lab3.
220 ProFTPD Server (Debian) [::ffff:192.168.2.30]
Name (lab3:vagrant): vagrant
331 Password required for vagrant
Password:
230 User vagrant logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> testlab2.txt
?Invalid command
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
-rw-rw-r--   1 vagrant  vagrant        14 Mar 28 23:38 testlab3.txt
226 Transfer complete
ftp> put /home/vagrant/testlab2.txt
local: /home/vagrant/testlab2.txt remote: /home/vagrant/testlab2.txt
200 PORT command successful
150 Opening BINARY mode data connection for /home/vagrant/testlab2.txt
226 Transfer complete
14 bytes sent in 0.00 secs (12.8737 kB/s)
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
-rw-r--r--   1 vagrant  vagrant        14 Mar 28 23:39 testlab2.txt
-rw-rw-r--   1 vagrant  vagrant        14 Mar 28 23:38 testlab3.txt
226 Transfer complete
ftp> exit
221 Goodbye.
```

ftp active mode

In active mode, the client initiates the connection to the server on a specific port (usually port 20), and the server responds by connecting back to the client on another port (usually a port number greater than 1024) to transfer the data.

In passive mode, the client initiates both the control and data connections to the server. The control connection is established on port 21 as usual, but the data connection is opened on a random high port chosen by the client. This allows the client to avoid any issues with firewalls or NAT devices that might block incoming connections on non-standard ports.

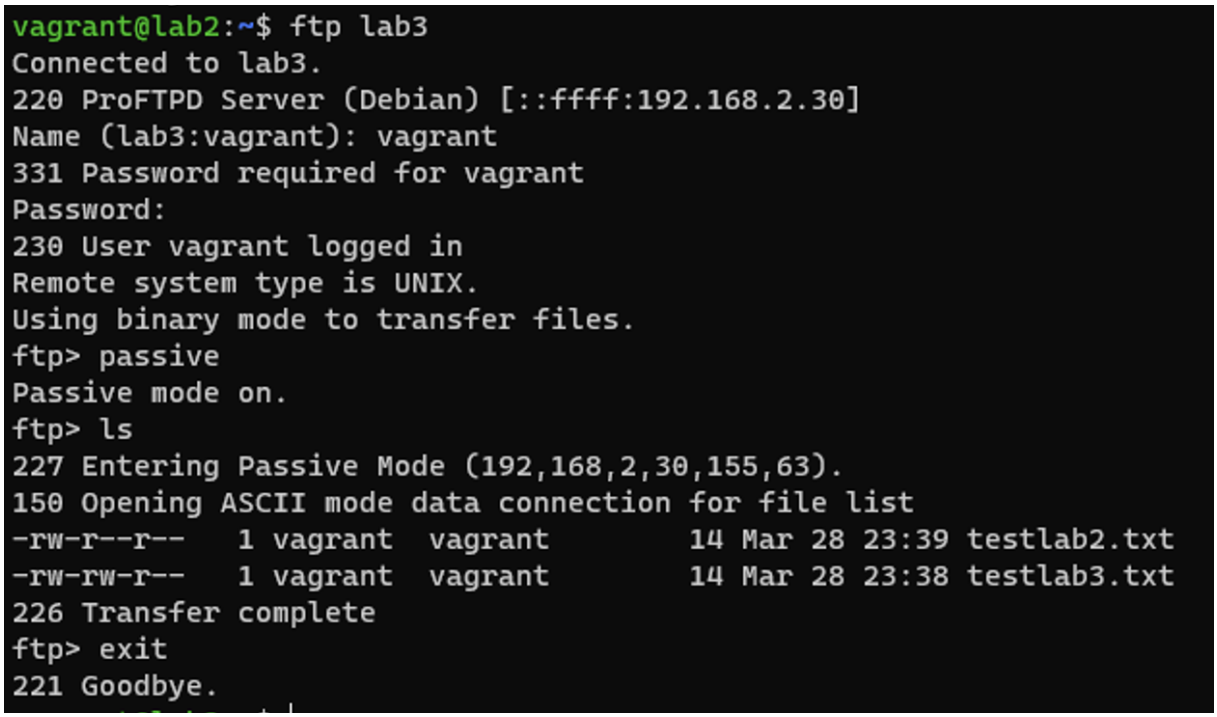
When a client requests a passive mode transfer, the server responds with an IP address and port number to use for the data transfer. The client then connects to that IP address and port to transfer the data.

Overall, passive mode can be more reliable and secure than active mode, especially when transferring files through firewalls or other network devices that may interfere with the data transfer. However, it may require some additional configuration and setup on both the client and server sides.

```
ftp lab3
```

```
passive
```

```
ls
```



Final file (filter table):

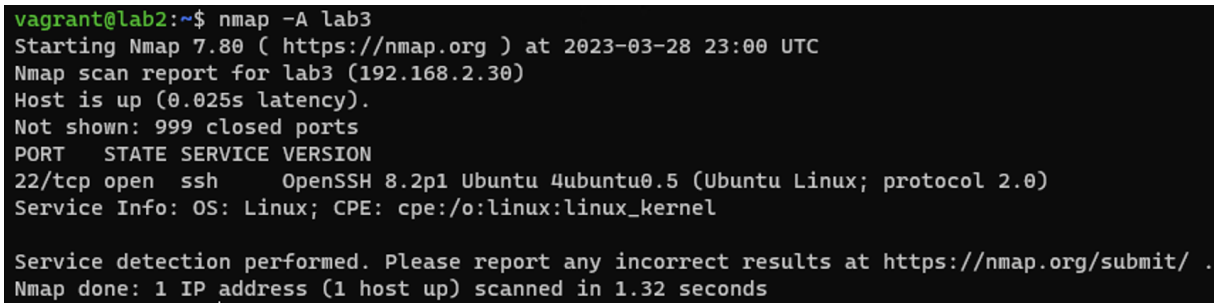
```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
        chain input {
                type filter hook input priority 0;
        }
        chain forward {
                type filter hook forward priority 0; policy drop;

                #ping
                iif "enp0s8" ip saddr lab2 icmp type echo-request counter accept
                oif "enp0s8" ip daddr lab2 icmp type echo-reply counter  accept

                #ssh
                #from
                iif "enp0s8" ip saddr lab2 tcp dport ssh ct state established,related,new counter accept
                oif "enp0s8" ip daddr lab2 tcp sport ssh ct state established,related counter accept
                #to
                iif "enp0s8" ip saddr lab2 tcp sport ssh ct state established,related counter accept
                oif "enp0s8" ip daddr lab2 tcp dport ssh ct state established,related,new counter accept

                #browsing
                iifname "enp0s8" ip saddr lab2 tcp dport { http, https } ct state established,related,new counter accept
                oifname "enp0s8" ip daddr lab2  tcp sport { http, https } ct state established,related counter accept

                #ftp
                iifname "enp0s8" ip saddr lab2 tcp dport ftp ct state established,new counter accept
                oifname "enp0s8" ip daddr lab2 tcp sport ftp ct state established counter accept
                #Additional rule active FTP
                oifname "enp0s8" ip daddr lab2 tcp sport ftp-data ct state established,related counter accept
                iifname "enp0s8" ip saddr lab2 tcp dport ftp-data ct state established counter accept
                #Additional rule passive FTP
                iifname "enp0s8" ip saddr lab2 tcp dport 1024-65535 ct state established,related counter  accept
```

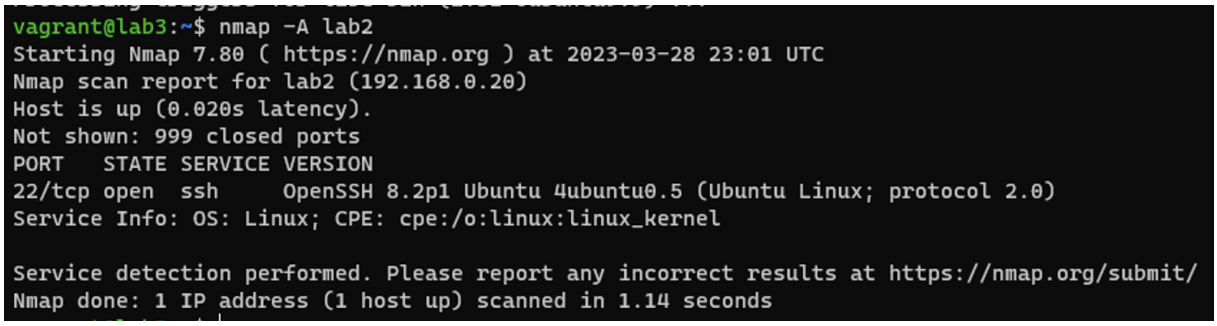
```
                oifname "enp0s8" ip daddr lab2 tcp sport 1024-65535 ct state established counter accept



        }
        chain output {
                type filter hook output priority 0;
        }
}
```

3.1 List the services that were found scanning the machines with and without the firewall active. Explain the differences in how the details of the system were detected.

Before activating the firewall

1. ssh
2. os
3. service info





After activating the firewall

Lab2:

```
nmap -A lab3
// default host discovery method
// pings first
// then scans
// -A flag enables aggressive scanning (version detection, OS detection, script scanning, and traceroute)
```

```
nmap -A -Pn lab3
// doesnt ping
// directly scans
```

```
vagrant@lab2:~$ nmap -A lab3
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-28 23:45 UTC
Nmap scan report for lab3 (192.168.2.30)
Host is up (0.015s latency).
Not shown: 996 filtered ports
PORT    STATE  SERVICE VERSION
21/tcp  open   ftp     ProFTPD
22/tcp  open   ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
80/tcp  open   http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
443/tcp closed https
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.22 seconds
```

```
vagrant@lab2:~$ nmap -A -Pn lab3
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-28 23:47 UTC
Nmap scan report for lab3 (192.168.2.30)
Host is up (0.0065s latency).
Not shown: 996 filtered ports
PORT    STATE  SERVICE VERSION
21/tcp  open   ftp     ProFTPD
22/tcp  open   ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
80/tcp  open   http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
443/tcp closed https
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 17.08 seconds
```

Lab3:

```
nmap -A lab2
```

```
nmap -A -Pn lab2
```

```
vagrant@lab3:~$ nmap -A lab2
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-28 23:48 UTC
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 3.52 seconds
vagrant@lab3:~$ nmap -A -Pn lab2
Starting Nmap 7.80 ( https://nmap.org ) at 2023-03-28 23:49 UTC
Nmap scan report for lab2 (192.168.0.20)
Host is up (0.0050s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.01 seconds
```

1. Port 21: FTP service running ProFTPD 1.3.5
2. Port 22: SSH service running OpenSSH 7.6p1 Ubuntu 4ubuntu0.7
3. Port 80: HTTP service running Apache httpd 2.4.29
4. Port 443: HTTPS service is closed

the firewall was able to prevent the detection of some services on both machines

3.2 List the commands used to implement the ruleset with explanations.

```
## Set up an nftables FORWARD policy to disallow traffic through the router by default.
sudo nft add chain inet filter forward { type filter hook forward priority 0\; policy drop\; }

## Add rules to allow ping from lab2 on the enp0s8 interface and replies to lab2. Change rules only for the FORWARD hook!
sudo nft add rule inet filter forward iif enp0s8 ip saddr lab2 icmp type echo-request counter accept
sudo nft add rule inet filter forward oif enp0s8 ip daddr lab2 icmp type echo-reply counter accept

##expand the ruleset to allow SSH connections to and from lab2.
#From lab2
sudo nft add rule inet filter forward iif enp0s8 ip saddr lab2 tcp dport 22 ct state new,established,related counter accept
sudo nft add rule inet filter forward oif enp0s8 ip daddr lab2 tcp sport 22 ct state established,related counter accept

#To lab2
sudo nft add rule inet filter forward iif enp0s8 ip saddr lab2 tcp sport 22 ct state established,related counter accept
sudo nft add rule inet filter forward oif enp0s8 ip daddr lab2 tcp dport 22 ct state new,established,related counter accept

#browsing
sudo nft add rule inet filter forward iifname enp0s8 ip saddr lab2 tcp dport { 80, 443 } ct state new,established,related counter accep
sudo nft add rule inet filter forward oifname enp0s8 ip daddr lab2 tcp sport { 80, 443 } ct state established,related counter accept

#FTP
sudo nft add rule inet filter forward iifname enp0s8 ip saddr lab2 tcp dport 21 ct state new,established counter accept
sudo nft add rule inet filter forward oifname enp0s8 ip daddr lab2 tcp sport 21 ct state established counter accept

#Additional rule active FTP
sudo nft add rule inet filter forward oifname enp0s8 ip daddr lab2 tcp sport 20 ct state established,related counter accept
sudo nft add rule inet filter forward iifname enp0s8 ip saddr lab2 tcp dport 20 ct state established counter accept

#Additional rule passive FTP
sudo nft add rule inet filter forward iifname enp0s8 ip saddr lab2 tcp dport 1024-65535 ct state established,related counter accept
sudo nft add rule inet filter forward oifname enp0s8 ip daddr lab2 tcp sport 1024-65535 ct state established counter accept
```

3.3 Create a few test cases to verify your ruleset. Run the tests and provide minimal, but sufficient snippets of iptables' or tcpdump's logs to support your test results.

- Ping lab3 from lab2 and verify that the response is received

tcpdump

Lab1:

```
sudo tcpdump -i enp0s8 icmp
```

Lab2:

```
ping -c 3 lab3
```

```
vagrant@lab2:~$ ping -c 3 lab3
PING lab3 (192.168.2.30) 56(84) bytes of data.
64 bytes from lab3 (192.168.2.30): icmp_seq=1 ttl=63 time=3.94 ms
64 bytes from lab3 (192.168.2.30): icmp_seq=2 ttl=63 time=3.31 ms
64 bytes from lab3 (192.168.2.30): icmp_seq=3 ttl=63 time=2.98 ms

--- lab3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 2.982/3.410/3.937/0.396 ms
```

```
vagrant@lab1:~$ sudo tcpdump -i enp0s8 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
23:51:22.742308 IP lab2 > lab3: ICMP echo request, id 7, seq 1, length 64
23:51:22.747158 IP lab3 > lab2: ICMP echo reply, id 7, seq 1, length 64
23:51:23.745710 IP lab2 > lab3: ICMP echo request, id 7, seq 2, length 64
23:51:23.747607 IP lab3 > lab2: ICMP echo reply, id 7, seq 2, length 64
23:51:24.748086 IP lab2 > lab3: ICMP echo request, id 7, seq 3, length 64
23:51:24.750266 IP lab3 > lab2: ICMP echo reply, id 7, seq 3, length 64
```

- SSH from lab2 to lab3 and verify that a connection can be established

Lab1: capture the traffic of SSH communication between lab2 and lab3.

```
sudo tcpdump -i enp0s8 tcp port 22
```

Lab2:

```
ssh lab3
```

```
vagrant@lab1:~$ sudo tcpdump -i enp0s8 tcp port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
23:52:08.591291 IP lab2.58960 > lab3.ssh: Flags [S], seq 1569663410, win 64240, options [mss 1460,sackOK,TS val 947567351 ecr 0,nop,wscale 7], length 0
23:52:08.592711 IP lab3.ssh > lab2.58960: Flags [S.], seq 2489417707, ack 1569663411, win 65160, options [mss 1460,sackOK,TS val 3681863915 ecr 947567351,nop,wscale 7], length 0
23:52:08.594067 IP lab2.58960 > lab3.ssh: Flags [.], ack 1, win 502, options [nop,nop,TS val 947567354 ecr 3681863915], length 0
23:52:08.595545 IP lab2.58960 > lab3.ssh: Flags [P.], seq 1:42, ack 1, win 502, options [nop,nop,TS val 947567356 ecr 3681863915], length 41
23:52:08.598644 IP lab3.ssh > lab2.58960: Flags [.], ack 42, win 509, options [nop,nop,TS val 3681863919 ecr 947567356], length 0
23:52:08.614892 IP lab3.ssh > lab2.58960: Flags [P.], seq 1:42, ack 42, win 509, options [nop,nop,TS val 3681863937 ecr 947567356], length 41
23:52:08.616385 IP lab2.58960 > lab3.ssh: Flags [.], ack 42, win 502, options [nop,nop,TS val 947567376 ecr 3681863937], length 0
23:52:08.619888 IP lab2.58960 > lab3.ssh: Flags [P.], seq 42:1554, ack 42, win 502, options [nop,nop,TS val 947567379 ecr 3681863937], length 1512
23:52:08.620432 IP lab3.ssh > lab2.58960: Flags [P.], seq 42:1098, ack 42, win 509, options [nop,nop,TS val 3681863942 ecr 947567376], length 1056
23:52:08.623239 IP lab3.ssh > lab2.58960: Flags [.], ack 1554, win 499, options [nop,nop,TS val 3681863943 ecr 947567379], length 0
23:52:08.623366 IP lab2.58960 > lab3.ssh: Flags [.], ack 1098, win 501, options [nop,nop,TS val 947567382 ecr 3681863942], length 0
23:52:08.625525 IP lab2.58960 > lab3.ssh: Flags [P.], seq 1554:1602, ack 1098, win 501, options [nop,nop,TS val 947567385 ecr 3681863943], length 48
23:52:08.627850 IP lab3.ssh > lab2.58960: Flags [.], ack 1602, win 501, options [nop,nop,TS val 3681863950 ecr 947567385
```

The output lists the details of each packet captured by tcpdump, including the source and destination IP addresses, the flags, the sequence and acknowledgement numbers, and the length of the packet. The flags indicate the purpose of the packet, such as whether it is a SYN packet (request to synchronize sequence numbers), an ACK packet (acknowledge receipt of a packet), or a PSH packet (push data to the receiver without waiting for more data)

- Access the web server on lab3 from lab2 and verify that the web page is displayed

Lab1:

```
sudo tcpdump -i enp0s8 tcp port 80 or port 443
```

Lab2: // doesnt work now cuz added proxy in part 4

```
lynx http://lab3
```

```
vagrant@lab1:~$ sudo tcpdump -i enp0s8 tcp port 80 or port 443
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
23:53:13.254635 IP lab2.51570 > lab3.http: Flags [S], seq 3997759237, win 64240, options [mss 1460,sackOK,TS val 9476320
14 ecr 0,nop,wscale 7], length 0
23:53:13.255862 IP lab3.http > lab2.51570: Flags [S.], seq 306854679, ack 3997759238, win 65160, options [mss 1460,sackO
K,TS val 3681928578 ecr 947632014,nop,wscale 7], length 0
23:53:13.258445 IP lab2.51570 > lab3.http: Flags [.], ack 1, win 502, options [nop,nop,TS val 947632018 ecr 3681928578],
 length 0
23:53:13.259979 IP lab2.51570 > lab3.http: Flags [P.], seq 1:227, ack 1, win 502, options [nop,nop,TS val 947632020 ecr
3681928578], length 226: HTTP: GET / HTTP/1.0
23:53:13.261427 IP lab3.http > lab2.51570: Flags [.], ack 227, win 508, options [nop,nop,TS val 3681928583 ecr 947632020
], length 0
23:53:13.266123 IP lab3.http > lab2.51570: Flags [P.], seq 1:3441, ack 227, win 508, options [nop,nop,TS val 3681928586
ecr 947632020], length 3440: HTTP: HTTP/1.1 200 OK
23:53:13.266888 IP lab3.http > lab2.51570: Flags [F.], seq 3441, ack 227, win 508, options [nop,nop,TS val 3681928587 ec
r 947632020], length 0
23:53:13.272630 IP lab2.51570 > lab3.http: Flags [.], ack 3441, win 489, options [nop,nop,TS val 947632028 ecr 368192858
6], length 0
23:53:13.277063 IP lab2.51570 > lab3.http: Flags [F.], seq 227, ack 3442, win 501, options [nop,nop,TS val 947632037 ecr
 3681928587], length 0
23:53:13.279729 IP lab3.http > lab2.51570: Flags [.], ack 228, win 508, options [nop,nop,TS val 3681928600 ecr 947632037
], length 0
```

syn, ack, get, ok, fin

- Upload and download files using FTP from lab2 to lab3 and vice versa

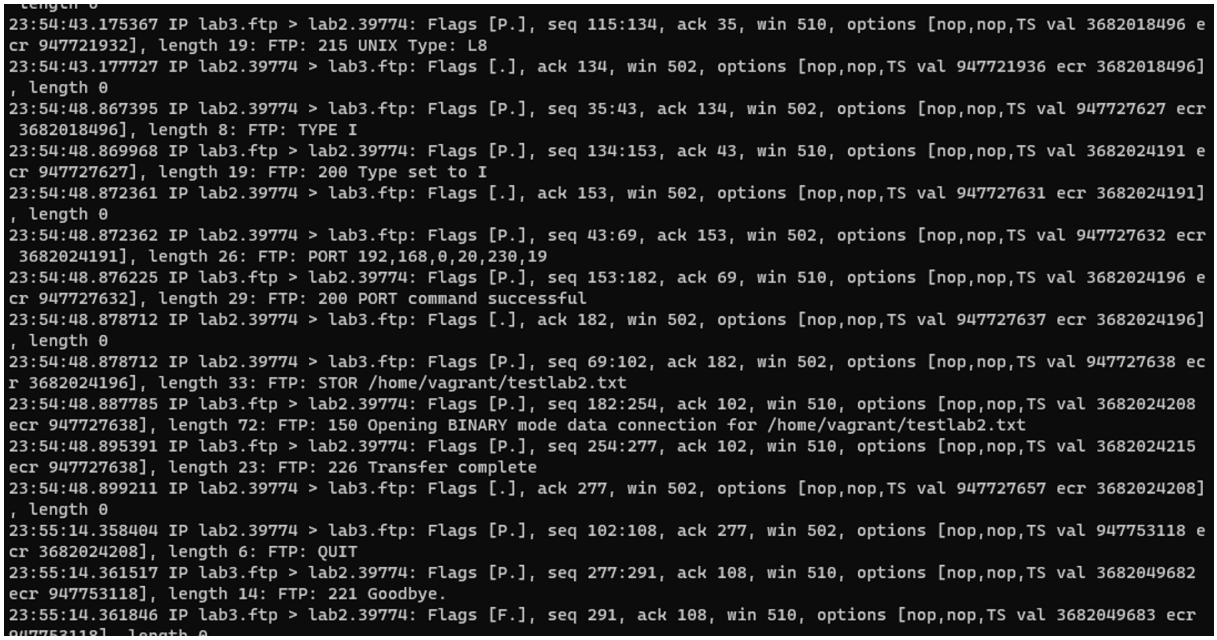
Lab1:

```
sudo tcpdump -i enp0s8 tcp port 21
```

Lab2:

```
ftp lab3
passive
```

```
put /home/vagrant/testlab2.txt
```



3.4 Explain the difference between netfilter DROP and REJECT targets. Test both of them, and explain your findings.

The netfilter DROP and REJECT targets both block traffic (packets), but they differ in how they handle the blocked traffic (response message). The **DROP** target silently drops the traffic without sending any response back to the sender, while the **REJECT** target sends an error message back to the sender

Test

Lab1:

```
sudo nano  /etc/nftables.conf
```

will test by edit the icmp rule which is now accept (so currently lab2 can ping lab3)

but will try change it to “drop”

```
iif "enp0s8" ip saddr lab2 icmp type echo-request counter drop
oif "enp0s8" ip daddr lab2 icmp type echo-reply counter  drop
```

reload table

```
sudo nft -f /etc/nftables.conf
```

Lab2: test ping

```
ping -c 3 lab3
```

```
vagrant@lab2:~$ ping -c 3 lab3
PING lab3 (192.168.2.30) 56(84) bytes of data.

--- lab3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2028ms
```

nothing is show : **DROP** target silently drops the traffic without sending any response back to the sender

Test reject, change the icmp rule to reject

Lab1 :

```
sudo nano  /etc/nftables.conf
```

change the config to reject

```
iif "enp0s8" ip saddr lab2 icmp type echo-request counter reject
oif "enp0s8" ip daddr lab2 icmp type echo-reply counter reject
```

reload table

```
sudo nft -f /etc/nftables.conf
```

Lab2 : Test ping

```
ping -c 3 lab3
```

```
vagrant@lab2:~$ ping -c 3 lab3
PING lab3 (192.168.2.30) 56(84) bytes of data.
From lab1 (192.168.0.10) icmp_seq=1 Destination Port Unreachable
From lab1 (192.168.0.10) icmp_seq=2 Destination Port Unreachable
From lab1 (192.168.0.10) icmp_seq=3 Destination Port Unreachable

--- lab3 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2010ms
```

see the error message : **REJECT** target sends an error message back to the sender

Change it back to **accept**.

▼ **Implement a web proxy**

In addition to packet filtering, a proxy can be used to control traffic. In this step, you will set up a web proxy and force all http traffic to go through the proxy, where more detailed rules can be applied.

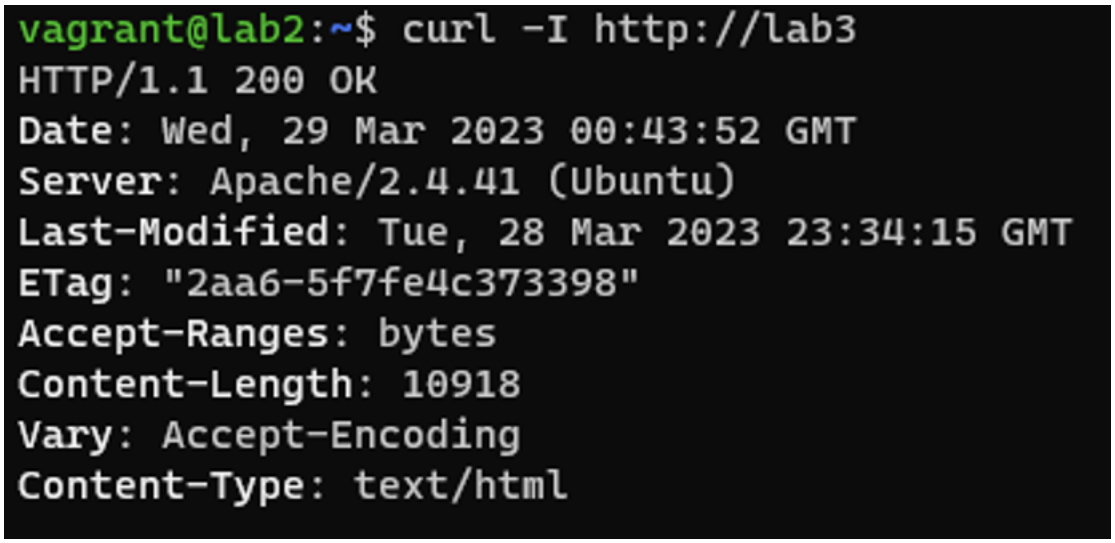
- Connect from **lab2** to the HTTP server running on **lab3** and capture the headers of the response.
- On **lab1**, configure a *squid(8)* web proxy to serve only requests from **lab2** as a transparent proxy.
- Configure the firewall on **lab1** to send all TCP traffic from **lab2** bound to port 80 to the squid proxy.
- Connect to the HTTP server on **lab3** again and capture the headers of the response.

- Finally, configure the proxy not to serve pages from **lab3** and attempt to retrieve the front page.

Connect from **lab2** to the HTTP server running on **lab3** and capture the headers of the response.

Lab2:

```
curl -I http://lab3
// curl sends http head request to url  - get only headers
// -I for head
```



Lab1:

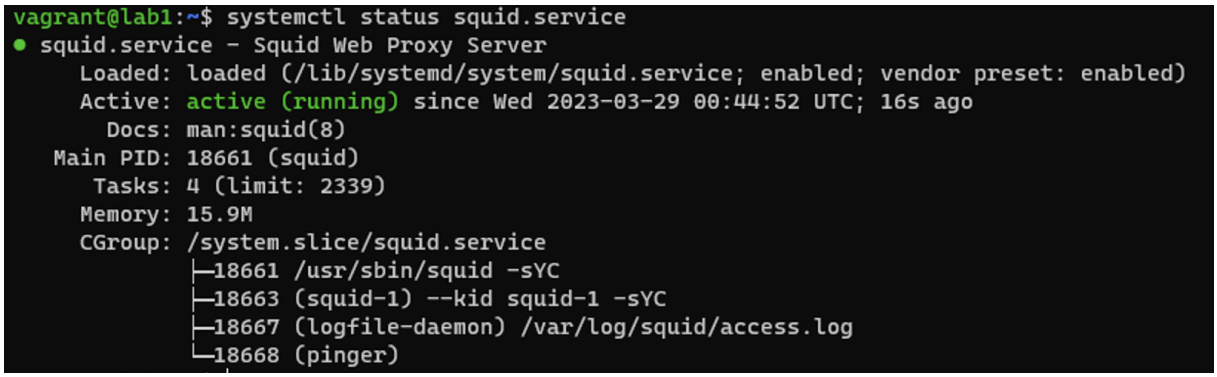
install squid

```
sudo apt-get install squid -y
```

```
sudo systemctl start squid
```

check

```
systemctl status squid.service
```



On **lab1**, configure a *squid(8)* web proxy to serve only requests from **lab2** as a transparent proxy.

edit config

```
sudo nano /etc/squid/squid.conf
```

Add the following in start of the file

```
acl squLab2 src 192.168.0.20
http_access allow squLab2
http_port 3128 transparent
```

`3128`. This is a common default port used by Squid for HTTP traffic.

reload

```
sudo systemctl reload squid
```

Configure the firewall on **lab1** to send all TCP traffic from **lab2** bound to port 80 to the squid proxy.

Add the following nat table in nftables.conf file:

```
sudo nano  /etc/nftables.conf
```

```
table ip nat {
        chain prerouting {
                type nat hook prerouting priority dstnat; policy accept;
                iifname "enp0s8" ip saddr 192.168.0.20 tcp dport 80 redirect to :3128
        }
}
```

```
sudo nft --check --file /etc/nftables.conf
```

```
sudo nft -f /etc/nftables.conf
```

check to see change

```
sudo nft list ruleset
```

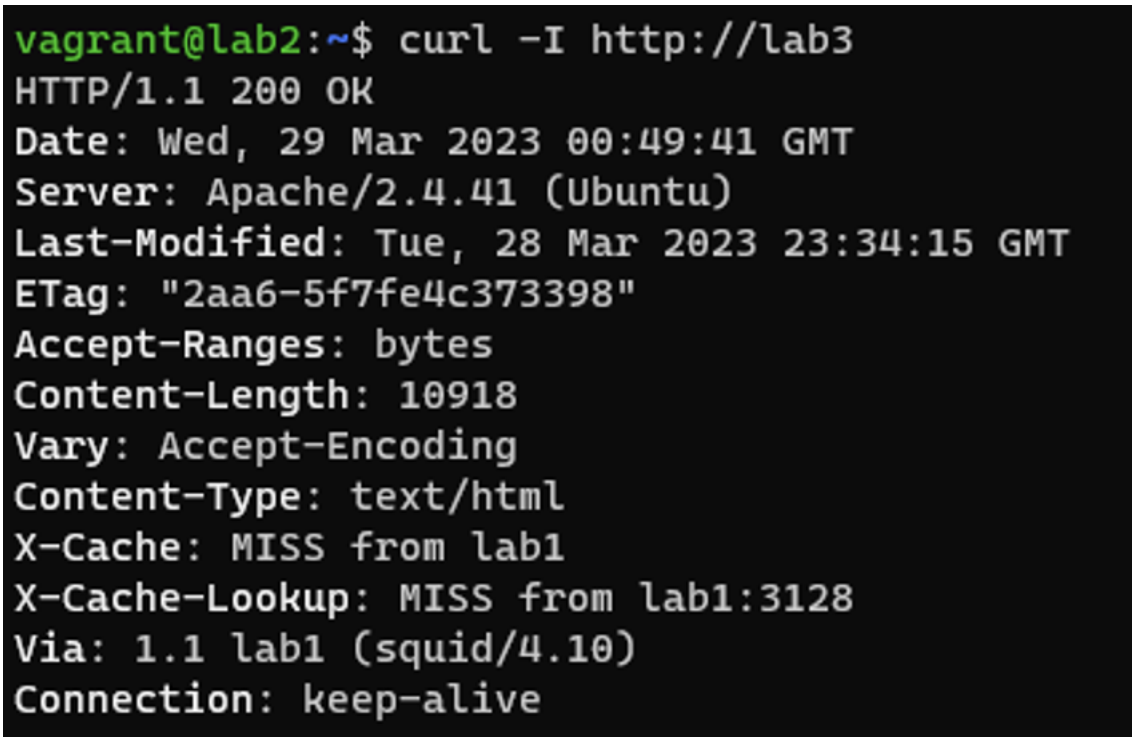
Connect to the HTTP server on **lab3** again and capture the headers of the response.

Lab1:

```
sudo systemctl reload squid
```

Lab2:

```
curl -I http://lab3
```



Finally, configure the proxy not to serve pages from **lab3** and attempt to retrieve the front page.

Modify the squid.conf file, add the following lines below existing config:

```
sudo nano /etc/squid/squid.conf
```

```
acl lab3
never_direct allow lab3
```

```
sudo systemctl reload squid
```

Lab2:

```
curl -I http://lab3
```

```
vagrant@lab2:~$ curl -I http://lab3
curl: (7) Failed to connect to lab3 port 80: Connection refused
```

4.1 List the commands you used to send the traffic to the proxy with explanations.

```
table ip nat {
        chain prerouting {
                type nat hook prerouting priority dstnat; policy accept;
                iifname "enp0s8" ip saddr 192.168.0.20 tcp dport 80 redirect to :3128
        }
}
```

```
//add table nat - for NAT related rules
sudo nft add table nat

// create new prerouting chain -rules applied before packets is routed to dest
sudo nft -- add chain nat prerouting '{ type nat hook prerouting priority -100; };'

// add rule
iifname enp0s8 ip saddr lab2 tcp dport 80 redirect to :3128
```

4.2 Show and explain the changes you made to the squid.conf.

```
acl squLab2 src 192.168.0.20
http_access allow squLab2
http_port 3128 transparent
acl lab3
never_direct allow lab3
```

1. acl squLab2 src 192.168.0.20 - access control list (ACL) named squLab2 that allows traffic from the IP address of lab2
2. http access allow squLab2 - allow traffic from said ACL
3. http port 3128 transparent - Squid should listen on port 3128 for HTTP and act as trasparent proxy - clients do not need to configure any proxy settings in their browsers.
4. acl lab3 - new acl
5. never direct - not serve any requests for pages from lab3

4.3 What is a transparent proxy?

proxy that intercepts and redirects all client requests to the proxy without the client being aware of it

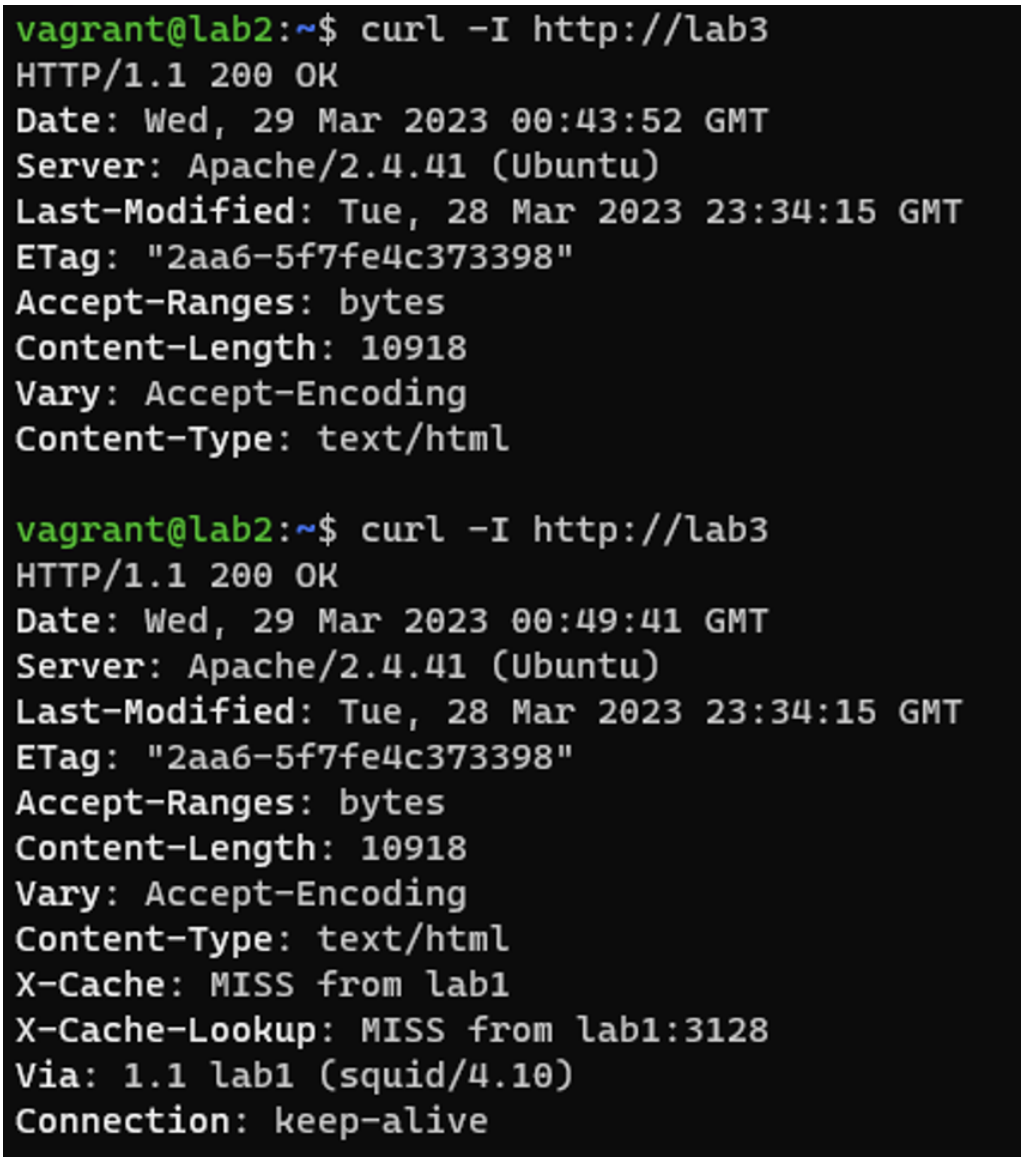
transparent because the client does not need to make any configuration changes to use the proxy, and the proxy operates transparently in the background.

4.4 List the differences in HTTP headers after setting up the proxy. What has changed?

Via header - the request has been forwarded by a proxy server

X cache : miss lab1 - request was not found in the proxy cache and was instead fetched from the origin server (lab3).

x-cache lookup - more info about the cache lookup process, indicating that the cache key was not found in the cache and was instead looked up using the server at lab1



▼ **Extras:**

Use of transparent proxy:

A transparent proxy is a type of proxy server that intercepts network traffic at the network level without requiring any special configuration on the client side. Some of the uses of a transparent proxy are:

1. Content filtering: Transparent proxies can be used to restrict access to certain types of websites or web content, such as adult content, social media, or online gaming sites.
2. Caching: Transparent proxies can cache frequently accessed web pages, which can improve the performance of the network by reducing the amount of bandwidth required to retrieve web content.
3. Load balancing: Transparent proxies can be used to distribute incoming network traffic across multiple servers, which can improve the availability and reliability of web services.
4. Security: Transparent proxies can be used to scan incoming web traffic for malware, viruses, and other security threats.

5. Logging: Transparent proxies can log incoming and outgoing network traffic, which can be useful for monitoring network activity and troubleshooting network problems.

Difference between transparent and non-transparent proxy:

A proxy is a server that sits between a client and a server and acts as an intermediary for requests from clients seeking resources from servers.

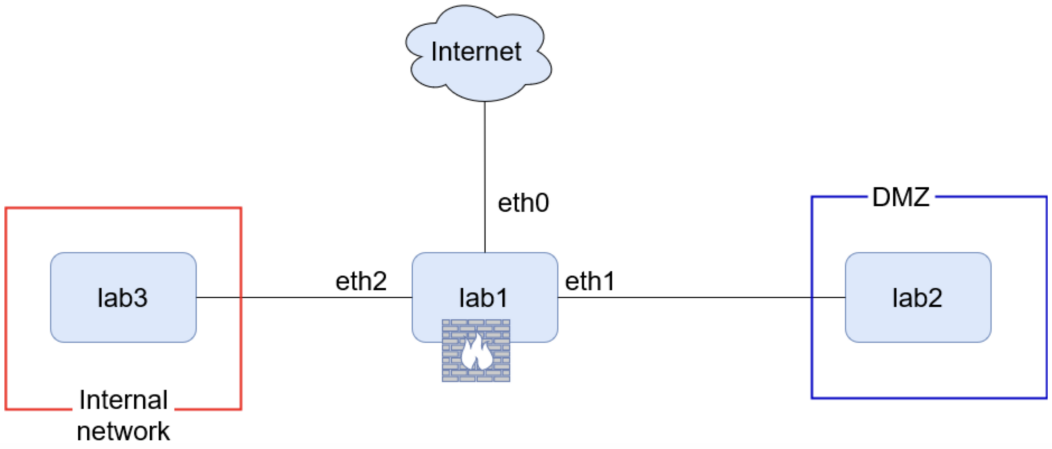
A transparent proxy intercepts traffic without modifying it and routes it to the requested server. The client is unaware that the traffic has been intercepted and the proxy is transparent to the client. Transparent proxies are often used in corporate or educational environments to enforce network policies or to speed up web browsing by caching frequently accessed content.

In contrast, a non-transparent proxy requires the client to configure its browser to use the proxy server explicitly. The client must be aware of the proxy server and must configure its requests to be sent through the proxy server. Non-transparent proxies are often used in situations where more control over the traffic is required, such as in security-conscious environments.

In summary, the main difference between transparent and non-transparent proxies is whether or not the client is aware of the presence of the proxy server.

▼ **Implement a DMZ**

A DMZ (**demilitarized zone**) network is a physical or logical subnet that separates an internal local area network (LAN) from other untrusted networks (usually the Internet). The purpose of a DMZ is to add an extra layer of security to an organization's LAN. In this way, each external network node can access only what is provided through the DMZ, and the rest of the organization's network remains behind the firewall. In this task we design a DMZ network with a firewall. Assume your organizations outward facing webserver running on lab2 is in a DMZ and your lab3 is in Internal Network, while lab1 is the firewall host executing the firewall rules as shown below.



You can use a destination network address translation (DNAT) rule to forward incoming packets on a lab1 port to a port on lab2.

1. On lab1 set up the nftables firewall with 3 network cards. You should forward port 8080 from your host to lab1

- eth0 is attached to NAT
- eth1 is attached to DMZ (lab2)
- eth3 is attached to Internal network (lab3)

2. Add a rule to the prerouting chain that redirects incoming packets on port 8080 to the port 80 on lab2. It means the traffic coming from eth0 will be redirected to eth1.

3. Add a rule to the postrouting chain to masquerade outgoing traffic.

4. The traffic coming from eth2 to eth 1 would be passed without any problem.
5. eth1 just allows to pass traffic in response to requests that have been made to lab2 in DMZ.

Lab1:

```
sudo sysctl -w  net.ipv4.ip_forward=1
```

```
sudo nano /etc/nftables.conf
```

```
table inet filter{
  ...

        chain forward{
          type filter hook forward priority filter; policy drop;

          ## Allow traffic from enp0s9 to enp0s8
          iifname enp0s9 oifname enp0s8 ct state established,new accept
          ## Only allow traffic from enp0s8 in response to requests that have been made to lab2
          iifname enp0s8 ip saddr lab2 ct state established accept
          ## add for host access
          iifname enp0s3 oifname enp0s8 ct state established,new accept
  }
  ...
}

table ip nat {
        chain prerouting {
                type nat hook prerouting priority dstnat; policy accept;
                iifname "enp0s8" ip saddr 192.168.0.20 tcp dport 80 redirect to :3128
                tcp dport 8080 dnat to lab2:80
        }

        chain postrouting {
                type nat hook postrouting priority srcnat; policy accept;

                ## Masquerade traffic from enp0s8
                oifname "enp0s3" iifname "enp0s8" masquerade
                ## Masquerade traffic from enp0s9
                oifname "enp0s3" iifname "enp0s9" masquerade
                ## add for host access
                oifname "enp0s8" ip saddr 10.0.2.2 masquerade
 }
}
```

```
sudo nft --check --file /etc/nftables.conf
```

```
sudo nft -f /etc/nftables.conf
```

Check:

```
sudo nft list ruleset
```

1. The first chain,
   prerouting, is a nat chain that hooks to the prerouting point with the priority of dstnat. It is used to redirect incoming traffic on port 8080 to port 80 on lab2. This rule ensures that traffic coming from eth0 is redirected to eth1, which is the DMZ network. This is accomplished using the "dnat to lab2:80" rule.
2. forward chain:
   first rule accepts packets that are being forwarded from interface enp0s9to interface enp0s8and have a connection tracking

state of either establishedor new.

The second rule accepts packets that are being forwarded from interface enp0s8 and have a source IP address of 192.168.0.20and a connection tracking state of established.

The third rule accepts packets that are being forwarded from interface enp0s3to interface enp0s8and have a connection tracking state of either establishedor new.

The default policy of the chain is set to drop any packets that do not match any of the rules.

3. postroutingChain
   hooks to the postrouting point with a priority of 100
   masquerade outgoing traffic from eth0 and eth1 so that the traffic appears to come from the firewall's public IP address rather than the private IP addresses of the devices in the network.
   This is accomplished using the "masquerade" rule.

On lab2 install Apache web server.

Lab2:

```
sudo apt-get install apache2 -y
```

host:

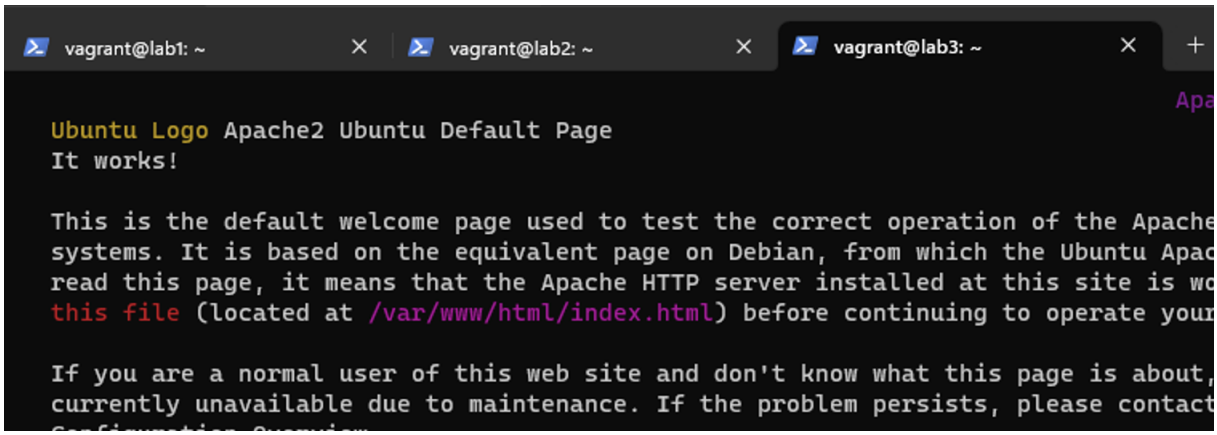
add content to vagrant file :: add to lab1 to do the port forwarding, and reload the vagrant

```
lab1.vm.network "forwarded_port", id: "http", guest: 8080, host: 8080, host_ip:"127.0.0.1"
```

5.1 Demonstrate you can browse the Apache webserver from your host and lab3. Demonstrate you cannot ping from lab2 to lab3

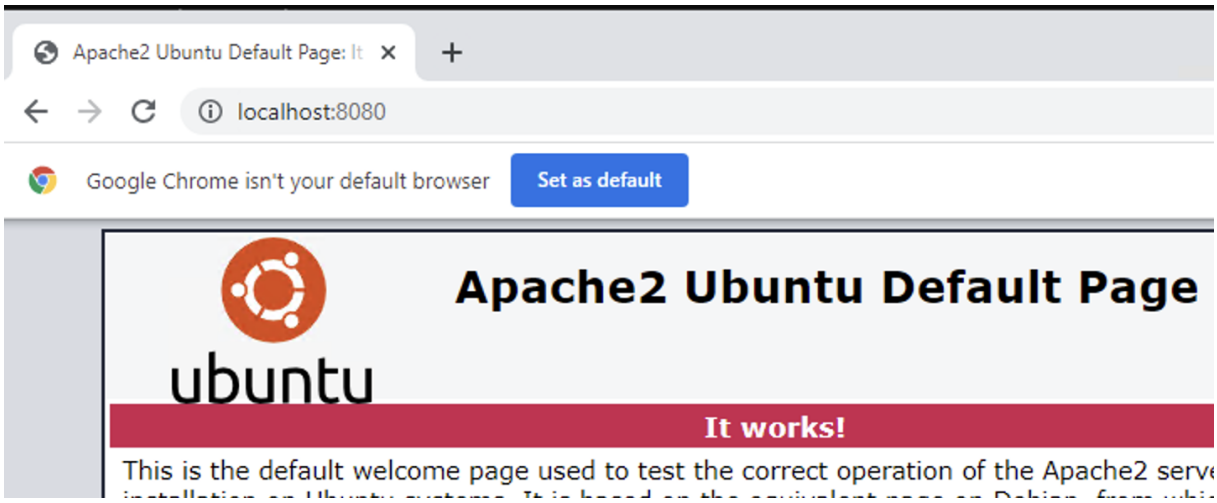
Lab3:

```
lynx http://lab2
```



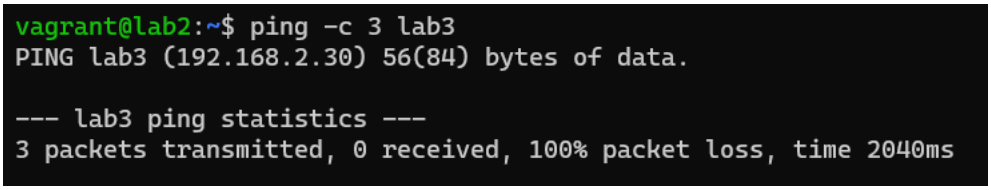
host:

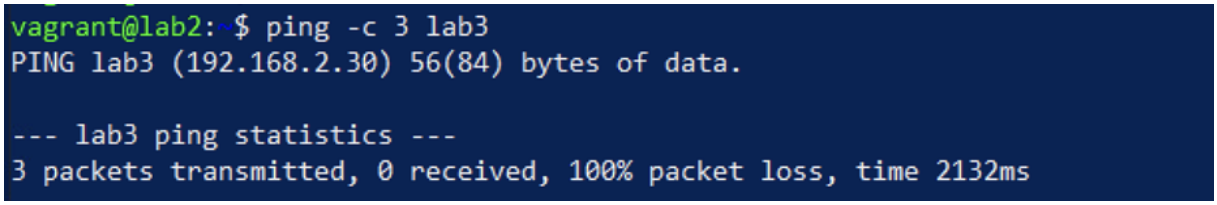
```
http://localhost:8080/
```



Lab2:

```
ping -c 3 lab3
```





Demonstrate that you can't ping from lab2 to lab3:

5.2 List the commands you used to set up the DMZ in nftables. You must show the prerouting, postrouting , forward, input and output chains.

^

▼ **Vagrant file**

```
#Define Vagrant provider and vagrant Box. Number 2 in configure indicates configuration version. Check vagrant documentation for more d
```

```
Vagrant.configure("2") do |config|

  config.vm.define "lab1" do |lab1|

                lab1.vm.box = "ubuntu/focal64"

                lab1.vm.hostname="lab1"

     lab1.vm.network "private_network", ip: "192.168.0.10",virtualbox__intnet: true,virtualbox__intnet:"intnet1"

                lab1.vm.network "private_network", ip: "192.168.2.10",virtualbox__intnet: true,virtualbox__intnet:"intnet2"

                lab1.vm.network "forwarded_port", id: "http", guest: 8080, host: 8080, host_ip:"127.0.0.1"

                lab1.vm.provider :virtualbox do |vb|

                                       # Custom CPU & Memory

                                        vb.customize ["modifyvm", :id, "--memory", "2048"]

                                        vb.customize ["modifyvm", :id, "--cpus", "2"]

             end

             #add provisioning scripts to Vagrantfile

               lab1.vm.provision "shell", inline: <<-SHELL

                         sudo echo "192.168.0.20 lab2" | sudo tee -a /etc/hosts

        sudo echo "192.168.2.30 lab3" | sudo tee -a /etc/hosts

                         sudo apt install net-tools


                   SHELL

     end

  config.vm.define "lab2" do |lab2|

                lab2.vm.box = "ubuntu/focal64"

                lab2.vm.hostname="lab2"

     lab2.vm.network "private_network", ip: "192.168.0.20",virtualbox__intnet: true,virtualbox__intnet:"intnet1"

                lab2.vm.provider :virtualbox do |vb|

                                       # Custom CPU & Memory

                                        vb.customize ["modifyvm", :id, "--memory", "2048"]

                                        vb.customize ["modifyvm", :id, "--cpus", "2"]

             end

             #add provisioning scripts to Vagrantfile

             lab2.vm.provision "shell", inline: <<-SHELL

                         sudo echo "192.168.0.10 lab1" | sudo tee -a /etc/hosts

        sudo echo "192.168.2.30 lab3" | sudo tee -a /etc/hosts

             sudo apt install net-tools


                   SHELL

     end

  config.vm.define "lab3" do |lab3|

                lab3.vm.box = "ubuntu/focal64"

                lab3.vm.hostname="lab3"

     lab3.vm.network "private_network", ip: "192.168.2.30",virtualbox__intnet: true,virtualbox__intnet:"intnet2"

                lab3.vm.provider :virtualbox do |vb|
```

```
                                    # Custom CPU & Memory
                                     vb.customize ["modifyvm", :id, "--memory", "2048"]
                                     vb.customize ["modifyvm", :id, "--cpus", "2"]
            end
            #add provisioning scripts to Vagrantfile
              lab3.vm.provision "shell", inline: <<-SHELL
                         sudo echo "192.168.0.20 lab2" | sudo tee -a /etc/hosts
                         sudo echo "192.168.2.10 lab1" | sudo tee -a /etc/hosts
                         sudo apt install net-tools

                   SHELL
    end
end
```