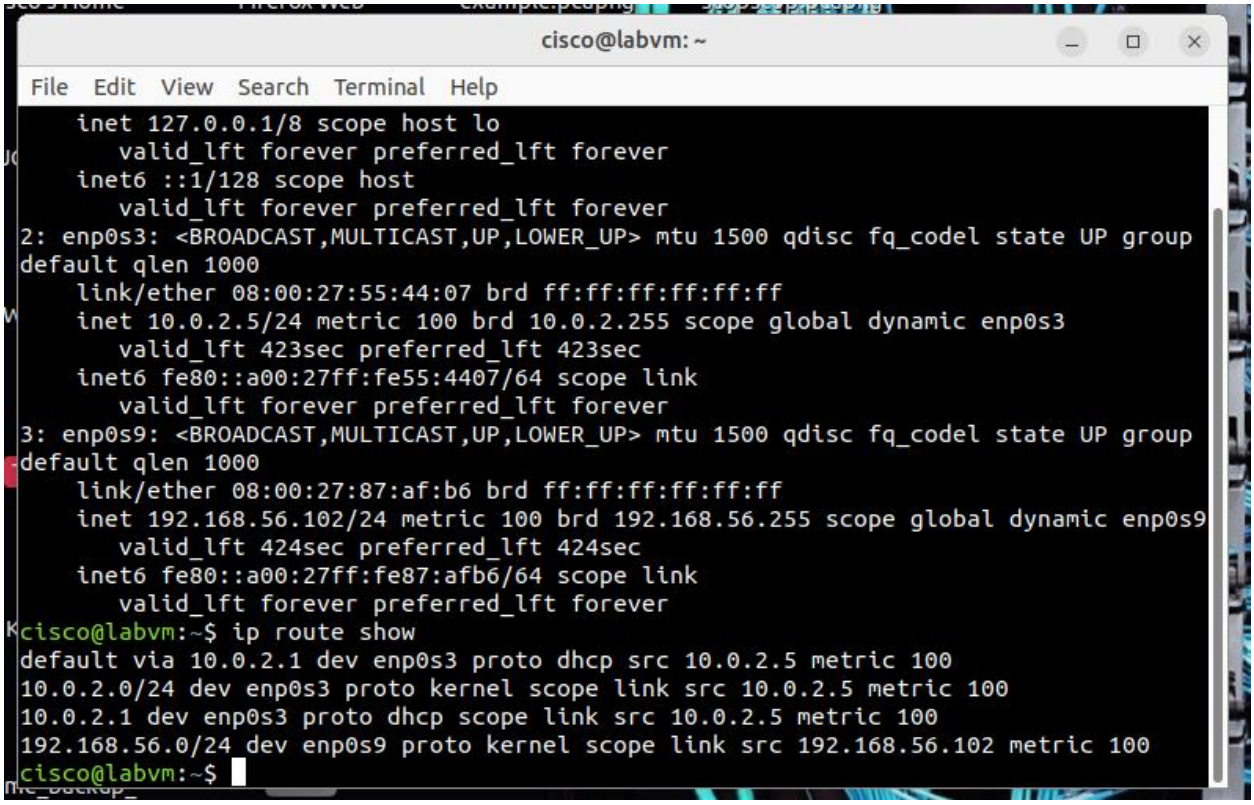
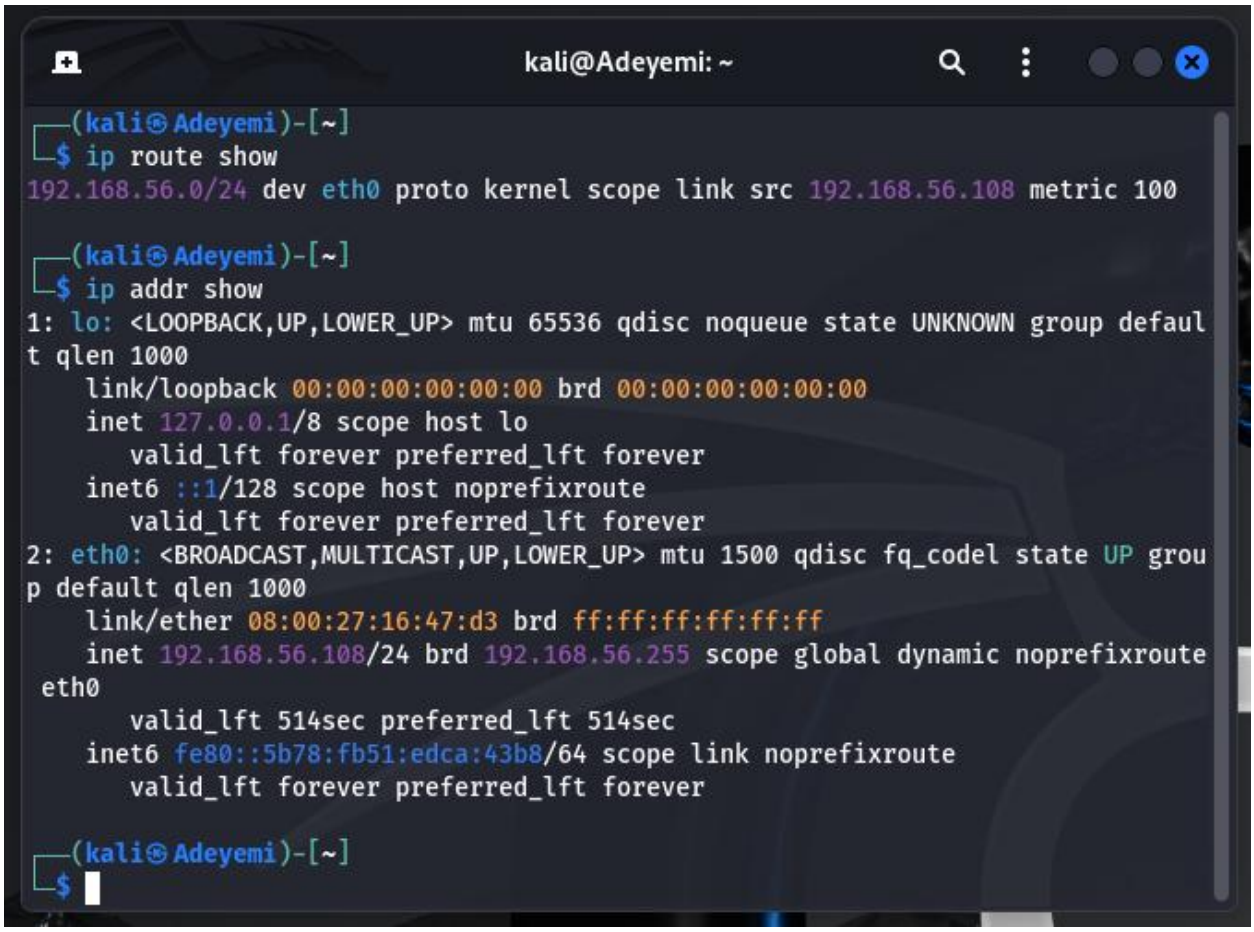
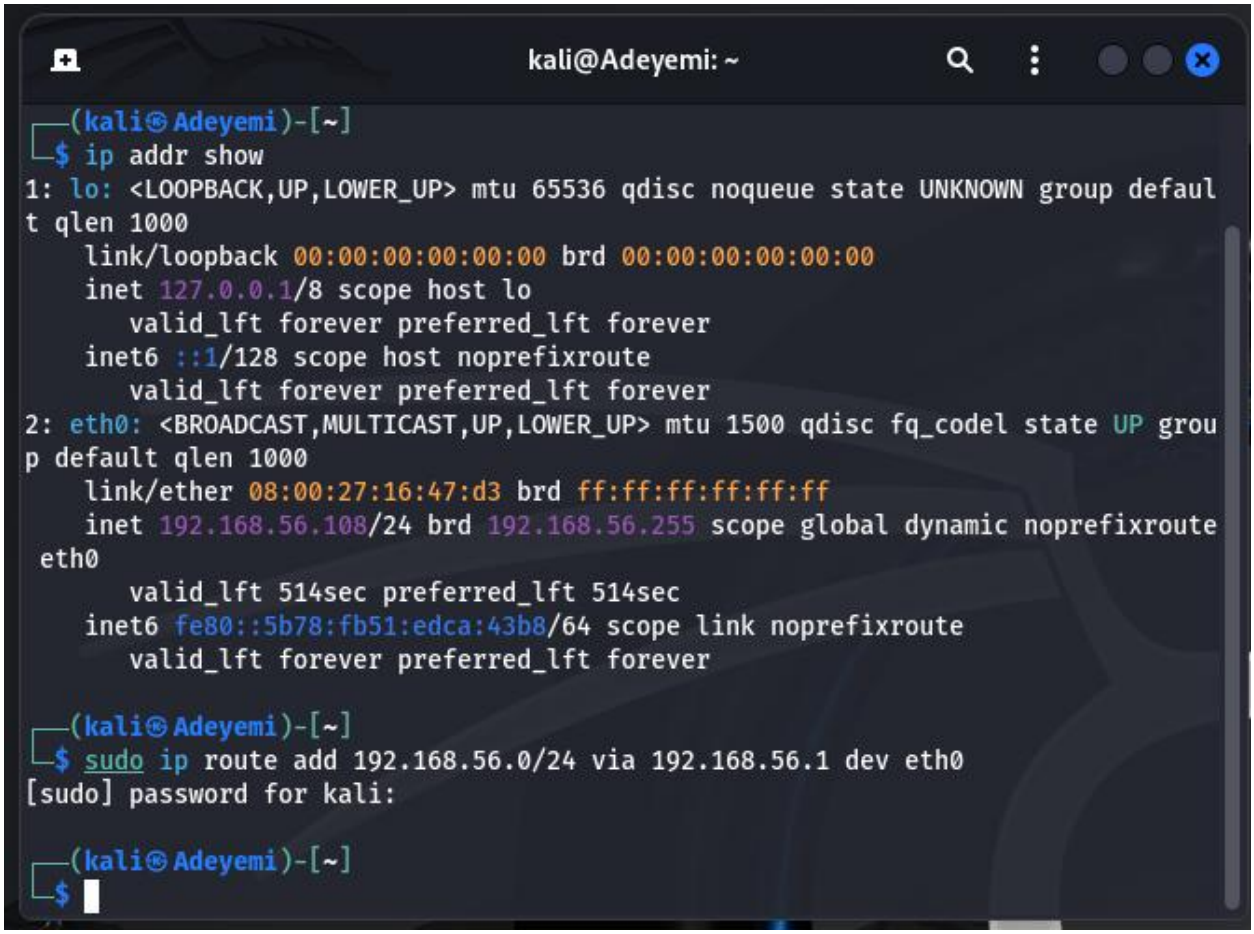
NAME: Adeyemi Akande
REG NO: 2024/INT/3800

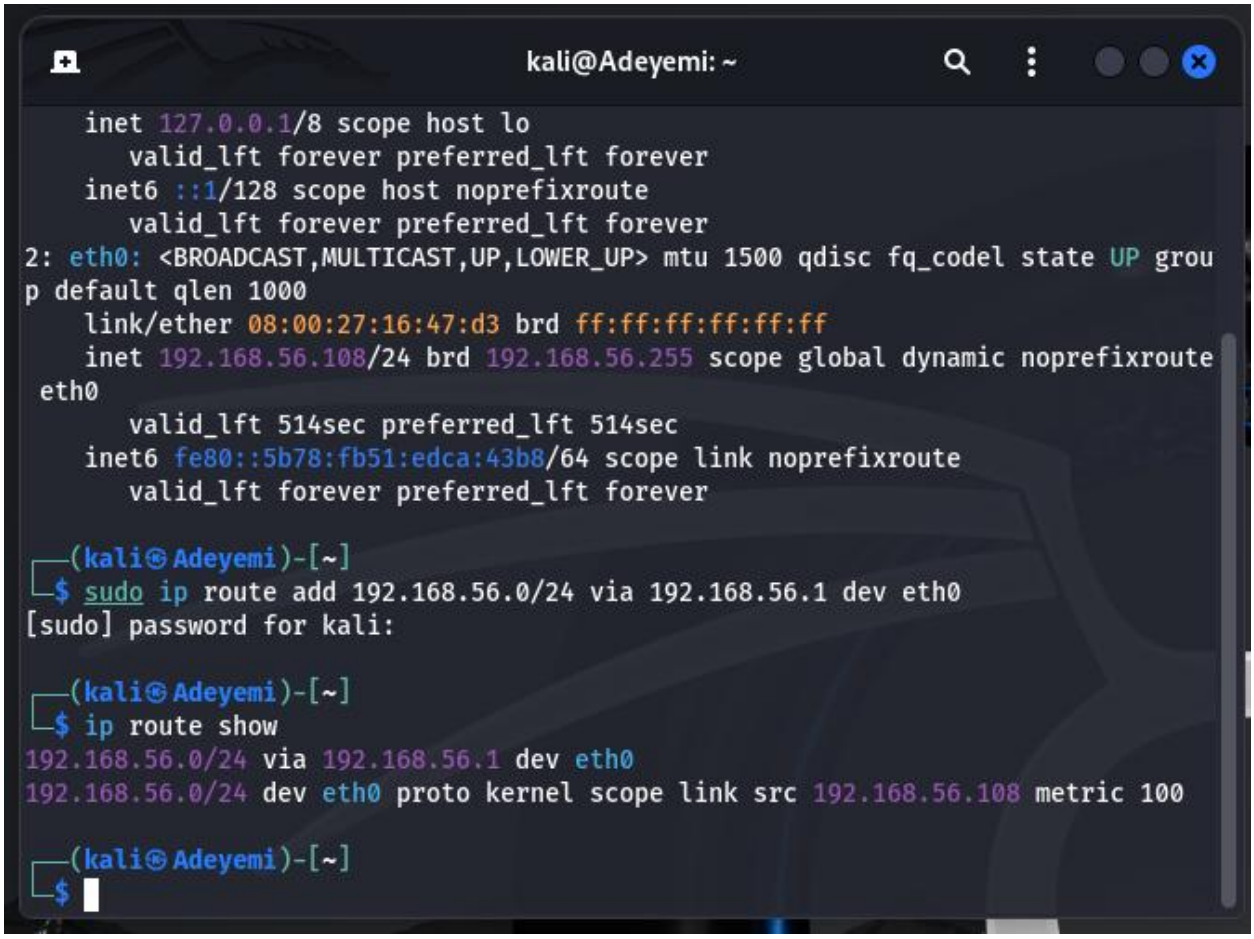
INT303: Networking Fundamentals – Lab 4
Lab 4: Simulating Network Routing and VLAN Configuration in Linux

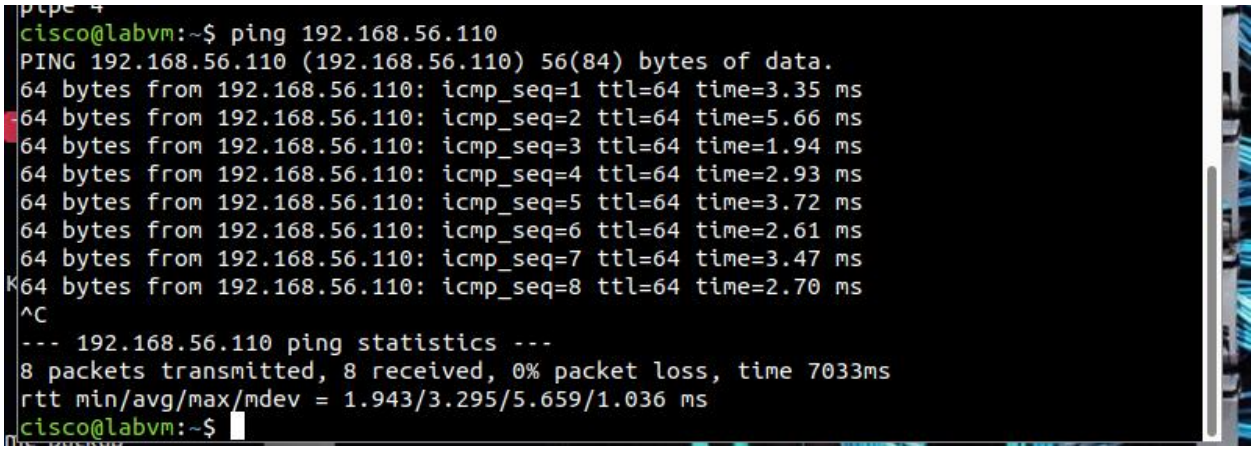
Exercise 1: Setting Up Static Routing in Linux

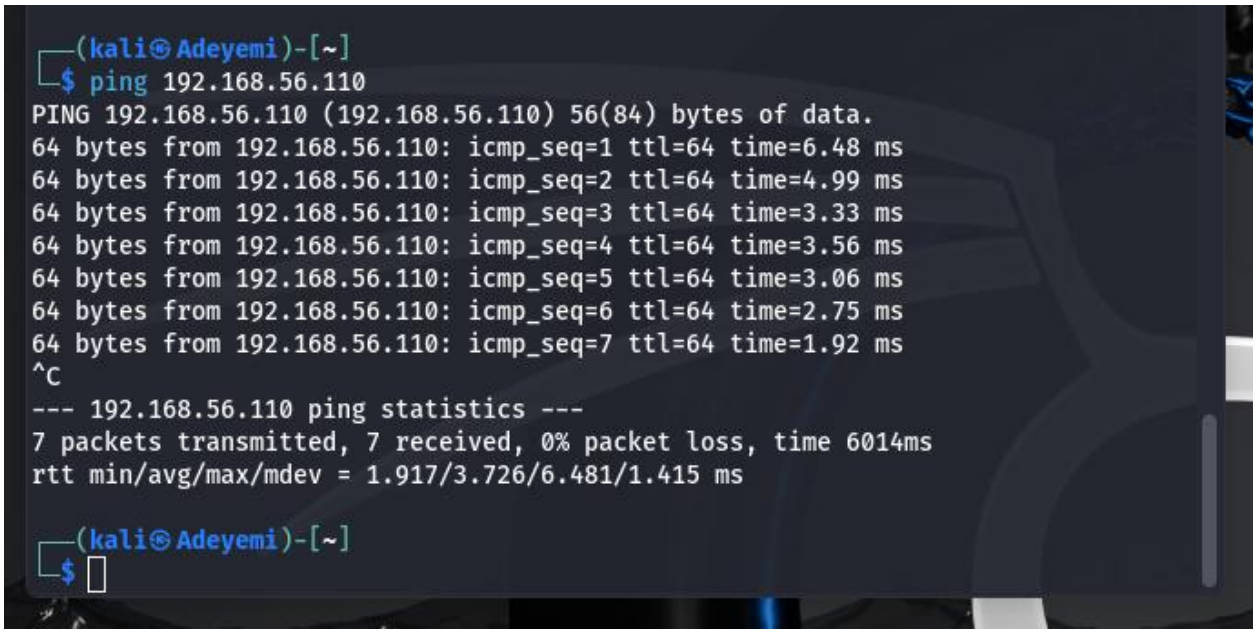



kali@Adeyemi: ~
┌──(kali㉿Adeyemi)-[~]
└─$ ip route show
192.168.56.0/24 dev eth0 proto kernel scope link src 192.168.56.108 metric 100
┌──(kali㉿Adeyemi)-[~]
└─$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host noprefixroute
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
link/ether 08:00:27:16:47:d3 brd ff:ff:ff:ff:ff:ff
inet 192.168.56.108/24 brd 192.168.56.255 scope global dynamic noprefixroute
eth0
valid_lft 514sec preferred_lft 514sec
inet6 fe80::5b78:fb51:edca:43b8/64 scope link noprefixroute
valid_lft forever preferred_lft forever
┌──(kali㉿Adeyemi)-[~]
└─$


kali@Adeyemi: ~
┌──(kali㉿Adeyemi)-[~]
└─$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host noprefixroute
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
link/ether 08:00:27:16:47:d3 brd ff:ff:ff:ff:ff:ff
inet 192.168.56.108/24 brd 192.168.56.255 scope global dynamic noprefixroute
eth0
valid_lft 514sec preferred_lft 514sec
inet6 fe80::5b78:fb51:edca:43b8/64 scope link noprefixroute
valid_lft forever preferred_lft forever
┌──(kali㉿Adeyemi)-[~]
└─$ sudo ip route add 192.168.56.0/24 via 192.168.56.1 dev eth0
[sudo] password for kali:
┌──(kali㉿Adeyemi)-[~]
└─$


kali@Adeyemi: ~
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host noprefixroute
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
link/ether 08:00:27:16:47:d3 brd ff:ff:ff:ff:ff:ff
inet 192.168.56.108/24 brd 192.168.56.255 scope global dynamic noprefixroute
eth0
valid_lft 514sec preferred_lft 514sec
inet6 fe80::5b78:fb51:edca:43b8/64 scope link noprefixroute
valid_lft forever preferred_lft forever
(kali㉿Adeyemi)-[~]
$ sudo ip route add 192.168.56.0/24 via 192.168.56.1 dev eth0
[sudo] password for kali:
(kali㉿Adeyemi)-[~]
$ ip route show
192.168.56.0/24 via 192.168.56.1 dev eth0
192.168.56.0/24 dev eth0 proto kernel scope link src 192.168.56.108 metric 100
(kali㉿Adeyemi)-[~]
$


cisco@labvm:~$ ping 192.168.56.110
PING 192.168.56.110 (192.168.56.110) 56(84) bytes of data.
64 bytes from 192.168.56.110: icmp_seq=1 ttl=64 time=3.35 ms
64 bytes from 192.168.56.110: icmp_seq=2 ttl=64 time=5.66 ms
64 bytes from 192.168.56.110: icmp_seq=3 ttl=64 time=1.94 ms
64 bytes from 192.168.56.110: icmp_seq=4 ttl=64 time=2.93 ms
64 bytes from 192.168.56.110: icmp_seq=5 ttl=64 time=3.72 ms
64 bytes from 192.168.56.110: icmp_seq=6 ttl=64 time=2.61 ms
64 bytes from 192.168.56.110: icmp_seq=7 ttl=64 time=3.47 ms
64 bytes from 192.168.56.110: icmp_seq=8 ttl=64 time=2.70 ms
^C
--- 192.168.56.110 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7033ms
rtt min/avg/max/mdev = 1.943/3.295/5.659/1.036 ms
cisco@labvm:~$



How Static Routing Works on a Linux System

Static routing in Linux involves manually configuring the routing table, which dictates how packets are forwarded to their destination. Here's an overview of the process:

1. **Routing Table**:
- Linux maintains a routing table that maps destination networks to specific gateways (or interfaces).
- Use ip route show to view the current table.
2. **Packet Forwarding**:
- When a packet is sent, the kernel checks the routing table to determine the best route.
- The route specifies:
- **Destination network/subnet**: Where the packet is going.
- **Gateway**: The next hop (router) for the packet.
- **Network interface**: The interface to use for forwarding the packet.
3. **Static Route Commands**:
- Use the ip route add command to specify static routes manually.
- Static routes persist only for the current session unless saved in configuration files.
4. **IP Forwarding (If Acting as a Router)**:

- To route packets between interfaces, IP forwarding must be enabled (net.ipv4.ip_forward).

Challenges When Setting Up Static Routes Manually

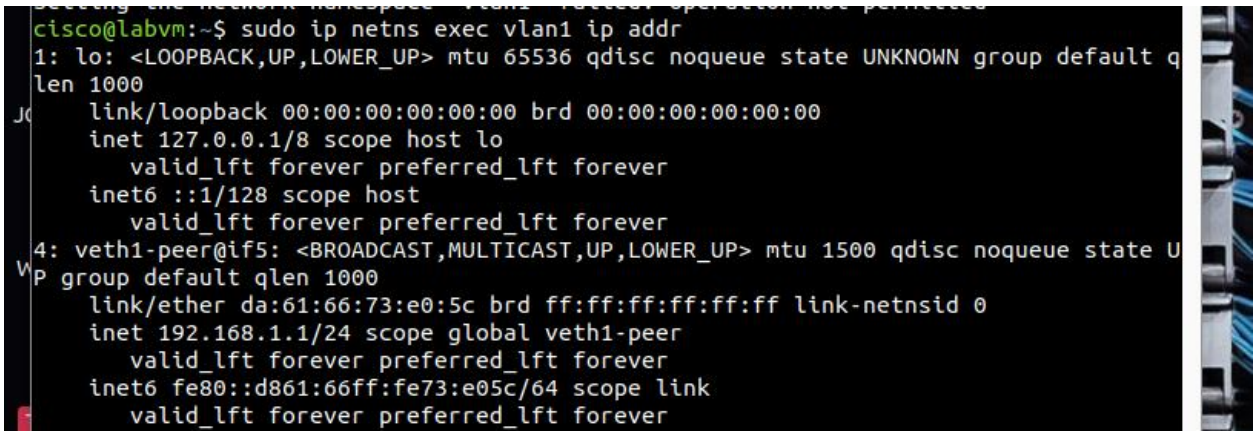
1. **Human Error**:
- **Incorrect Configuration**: Typing the wrong destination, gateway, or interface can lead to misrouted or dropped packets.
- **Overlapping Routes**: Adding conflicting routes can cause unpredictable behavior.
2. **Network Changes**:
- Static routes don't adapt to network topology changes, such as a gateway going offline or a new subnet being added.
3. **Route Persistence**:
- Routes added manually are not persistent across reboots unless explicitly saved, requiring additional configuration.
4. **Scalability**:
- Managing multiple routes across large networks is cumbersome and error-prone. Dynamic routing protocols (e.g., OSPF, BGP) are better suited for such environments.
5. **Debugging Complexity**:
- If packets are not reaching their destination, identifying whether the issue is with the route, gateway, or another part of the network can be challenging.
6. **Gateway or Interface Dependency**:
- Static routes rely on specific gateways or interfaces. If these go down or change IPs, the route breaks unless updated manually.
7. **Overhead for Redundancy**:
- Static routing doesn't support automatic failover. If a gateway becomes unavailable, traffic won't be rerouted unless a new static route is configured.
8. **Administrative Burden**:

- Each system in the network must be individually configured with appropriate routes, which is time-consuming.

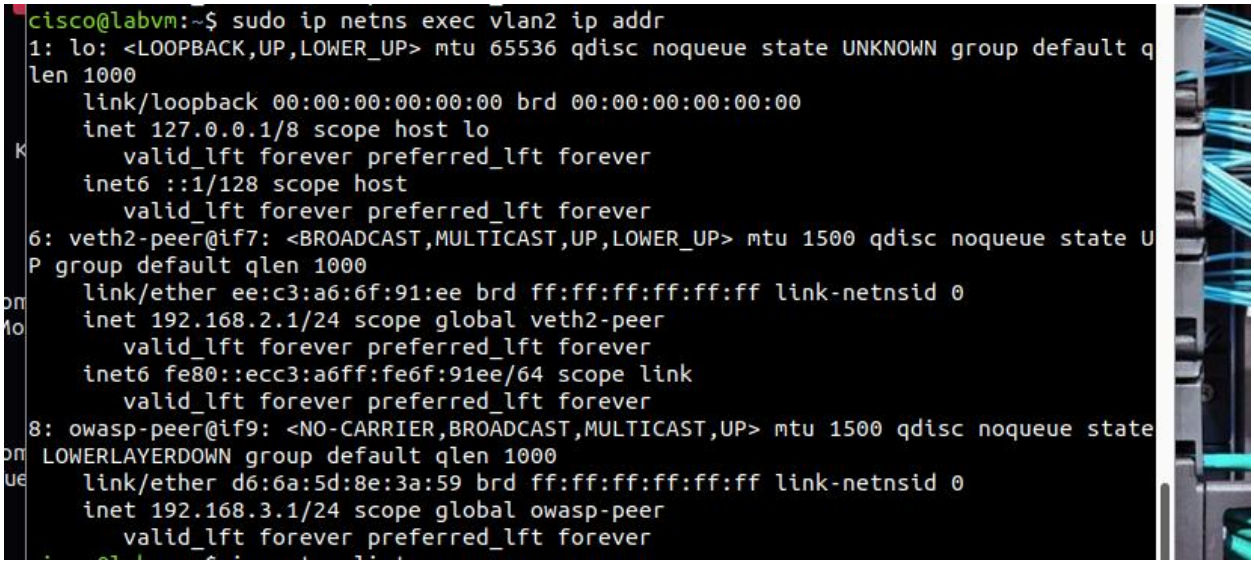
Exercise 2: VLAN Configuration Using Network Namespaces

Vlan1 and vlan2 created and verified with: ip netns list in the image below

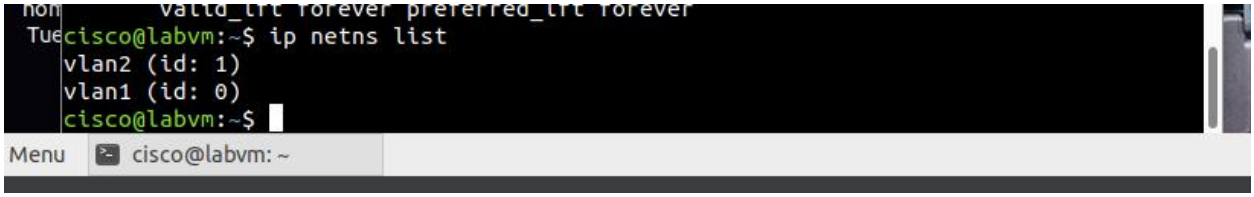
Checking vlan1 config



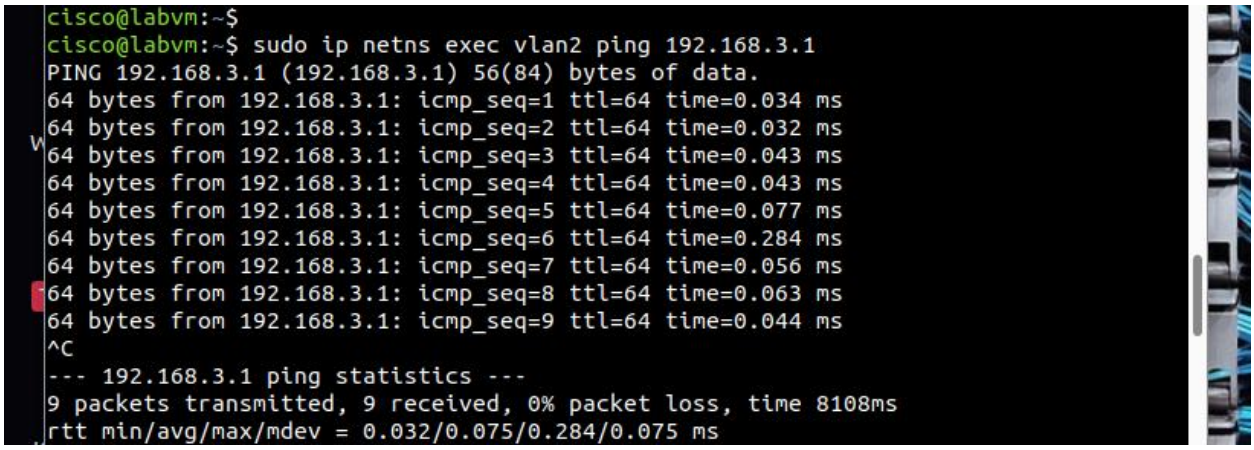
Checking vlan2 config



Vlan(s) creation confirmed



Pinging OWASP in the screenshot below:



How Network Namespaces Simulate VLANs in Linux

Network namespaces provide isolated environments for networking resources, effectively simulating the behavior of VLANs. Here's how they achieve this:

1. Network Isolation

- Each namespace has its own instance of the network stack, including:
- Interfaces

- Routing tables
- Firewall rules
- ARP tables
- Packets sent within a namespace remain isolated unless explicitly routed to another namespace, akin to VLAN isolation.

---

2. Virtual Ethernet Interfaces

- Virtual Ethernet (veth) pairs connect namespaces, behaving like a physical cable connecting two switches or devices in a VLAN.
- Each end of the veth pair exists in a different namespace, enabling controlled communication between namespaces.

3. IP Addressing and Subnets

- Within a namespace, IP addresses and subnets can be assigned just like in a VLAN.
- For example:
- Namespace `vlan1`: 192.168.1.0/24
- Namespace `vlan2`: 192.168.2.0/24
- This segmentation mirrors the behavior of VLANs, where devices in different VLANs cannot communicate without routing.

---

4. Bridge Configuration for Connectivity

- A Linux bridge can connect multiple namespaces, simulating a physical VLAN-aware switch.
- For example:
- Namespaces connected to the same bridge are part of the same broadcast domain.
- Traffic between namespaces on the bridge respects VLAN-like tagging or isolation.

---

5. Routing Between Namespaces

- Using routing or NAT, traffic between namespaces can be managed, similar to a router connecting VLANs.

Benefits of Using Network Namespaces for Network Isolation

1. Strong Isolation

- Each namespace has its own isolated networking stack, ensuring processes in one namespace cannot interfere with another.
- This makes it ideal for scenarios like:
- Multi-tenant environments
- Testing and development

2. Lightweight and Efficient

- Unlike traditional VLANs requiring physical switches or hypervisors, namespaces are entirely software-based.
- They are lightweight, requiring minimal resources compared to virtual machines.

3. Flexible and Programmable

- Namespaces can be dynamically created, configured, and removed using simple tools like `ip netns`.
- They integrate seamlessly with tools like Docker, Kubernetes, and Open vSwitch.

4. Granular Control

- Administrators can precisely control traffic, routing, and policies within each namespace.
- Network namespaces can be combined with features like:
- **iptables** for firewalling
- **tc** for traffic shaping
- **policy-based routing**

5. Simulate Complex Network Topologies

- Namespaces can mimic VLANs, subnets, routers, and firewalls, making them invaluable for:
- Network emulation
- Training and certification labs
- Testing network configurations and software

6. Cost-Effective

- No need for additional hardware or specialized VLAN-aware switches. Everything runs on a standard Linux system.

7. Integration with Modern Networking Tools

- Namespaces are compatible with advanced technologies like:
- **Container networking**: Docker and Kubernetes use namespaces for pod-level isolation.
- **SDN (Software-Defined Networking)**: Tools like Open vSwitch leverage namespaces for network virtualization.

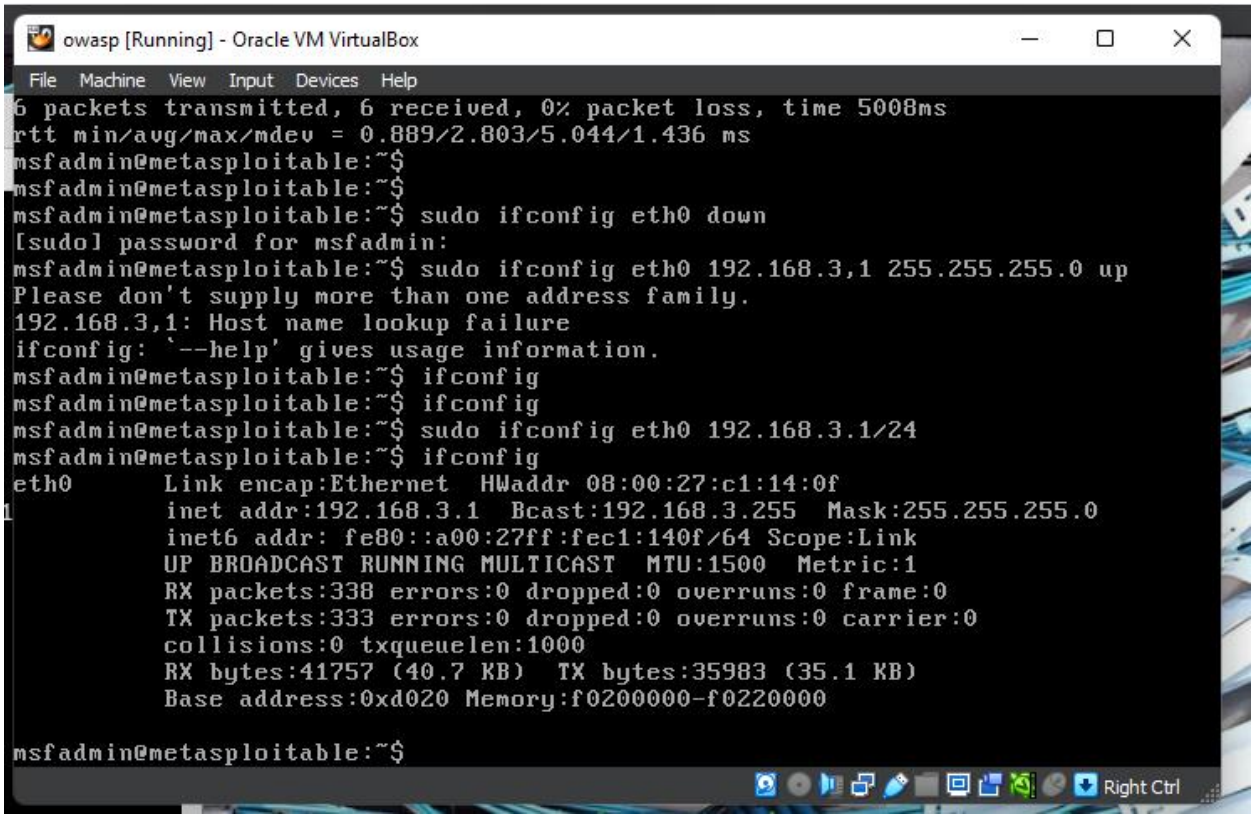
Use Cases

- **Testing**: Build isolated environments to test configurations without affecting production networks.
- **Multi-Tenant Solutions**: Create separate namespaces for each tenant in a shared environment.
- **Microservices**: Provide isolated network environments for individual containers.

In summary, network namespaces in Linux provide an efficient and flexible way to simulate VLANs, offering isolation and control without the need for specialized hardware. They're particularly beneficial for development, testing, and containerized environments.

Exercise 3: IP Address Assignment and Subnetting in Linux

Exercise 4: Testing Connectivity Using Ping and Traceroute



```
ping: connect: Network is unreachable
cisco@labvm:~$ ping 192.168.3.1
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=1 ttl=64 time=8.33 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=64 time=4.12 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=64 time=1.71 ms
64 bytes from 192.168.3.1: icmp_seq=4 ttl=64 time=3.78 ms
64 bytes from 192.168.3.1: icmp_seq=5 ttl=64 time=2.16 ms
64 bytes from 192.168.3.1: icmp_seq=6 ttl=64 time=5.93 ms
64 bytes from 192.168.3.1: icmp_seq=7 ttl=64 time=3.77 ms
64 bytes from 192.168.3.1: icmp_seq=8 ttl=64 time=5.05 ms
64 bytes from 192.168.3.1: icmp_seq=9 ttl=64 time=2.76 ms
64 bytes from 192.168.3.1: icmp_seq=10 ttl=64 time=2.60 ms
64 bytes from 192.168.3.1: icmp_seq=11 ttl=64 time=3.09 ms
64 bytes from 192.168.3.1: icmp_seq=12 ttl=64 time=8.50 ms
64 bytes from 192.168.3.1: icmp_seq=13 ttl=64 time=2.93 ms
^C
--- 192.168.3.1 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12044ms
rtt min/avg/max/mdev = 1.710/4.209/8.498/2.101 ms
cisco@labvm:~$
```

Traceroute screenshot

```
13 packets transmitted, 13 received, 0% packet loss, time 12044ms
rtt min/avg/max/mdev = 1.710/4.209/8.498/2.101 ms
cisco@labvm:~$ traceroute 192.168.3.1
traceroute to 192.168.3.1 (192.168.3.1), 64 hops max
  1   192.168.3.1  11.332ms  1.170ms  0.892ms
cisco@labvm:~$
```

Menu cisco@labvm: ~

Exercise 5: Configuring `iptables` for Routing and Firewall Rules

Initial configuration that was changed to make it go inline with my original network configuration

```
cisco@labvm:~$ sudo iptables -A FORWARD -s 192.168.10.0/24 -d 192.168.20.0/24 -j
 ACCEPT
cisco@labvm:~$
cisco@labvm:~$ sudo iptables -A FORWARD -s 192.168.10.0/24 -d <OWASP_IP> -j DROP
bash: OWASP_IP: No such file or directory
cisco@labvm:~$ sudo iptables -A FORWARD -s 192.168.10.0/24 -d 192.168.20.100 -j
DROP
cisco@labvm:~$
cisco@labvm:~$
cisco@labvm:~$
```

Default iptables Policies

Set default policies to allow traffic that matches the rules and drop anything else:

-

```
cisco@labvm:~$ sudo iptables -P FORWARD ACCEPT
sudo iptables -P INPUT ACCEPT
sudo iptables -P OUTPUT ACCEPT
[sudo] password for cisco:
cisco@labvm:~$
```

To verify the rules you've added:

```
cisco@labvm:~$
cisco@labvm:~$
cisco@labvm:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 5 packets, 380 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

Chain OUTPUT (policy ACCEPT 6 packets, 456 bytes)
 pkts bytes target     prot opt in     out     source               destination
cisco@labvm:~$
```

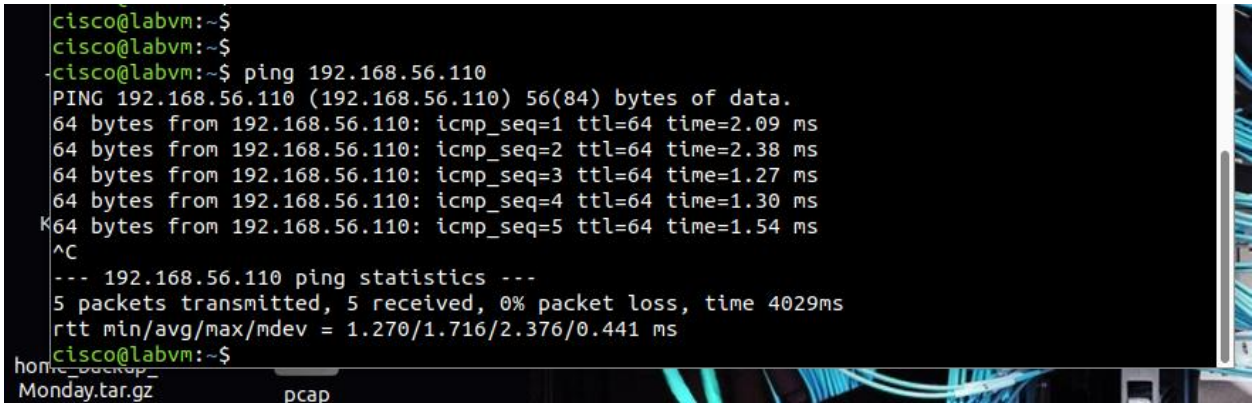
Test the Configuration

Perform the following tests to verify the behavior of your setup:

a. Ping Between Networks

```
cisco@labvm:~$ ping 192.168.56.108
PING 192.168.56.108 (192.168.56.108) 56(84) bytes of data.
64 bytes from 192.168.56.108: icmp_seq=1 ttl=64 time=4.05 ms
64 bytes from 192.168.56.108: icmp_seq=2 ttl=64 time=1.05 ms
64 bytes from 192.168.56.108: icmp_seq=3 ttl=64 time=1.36 ms
64 bytes from 192.168.56.108: icmp_seq=4 ttl=64 time=1.05 ms
64 bytes from 192.168.56.108: icmp_seq=5 ttl=64 time=1.92 ms
64 bytes from 192.168.56.108: icmp_seq=6 ttl=64 time=0.845 ms
64 bytes from 192.168.56.108: icmp_seq=7 ttl=64 time=1.21 ms
64 bytes from 192.168.56.108: icmp_seq=8 ttl=64 time=1.33 ms
^C
--- 192.168.56.108 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7012ms
rtt min/avg/max/mdev = 0.845/1.601/4.050/0.972 ms
cisco@labvm:~$
```

Ping the OWASP VM

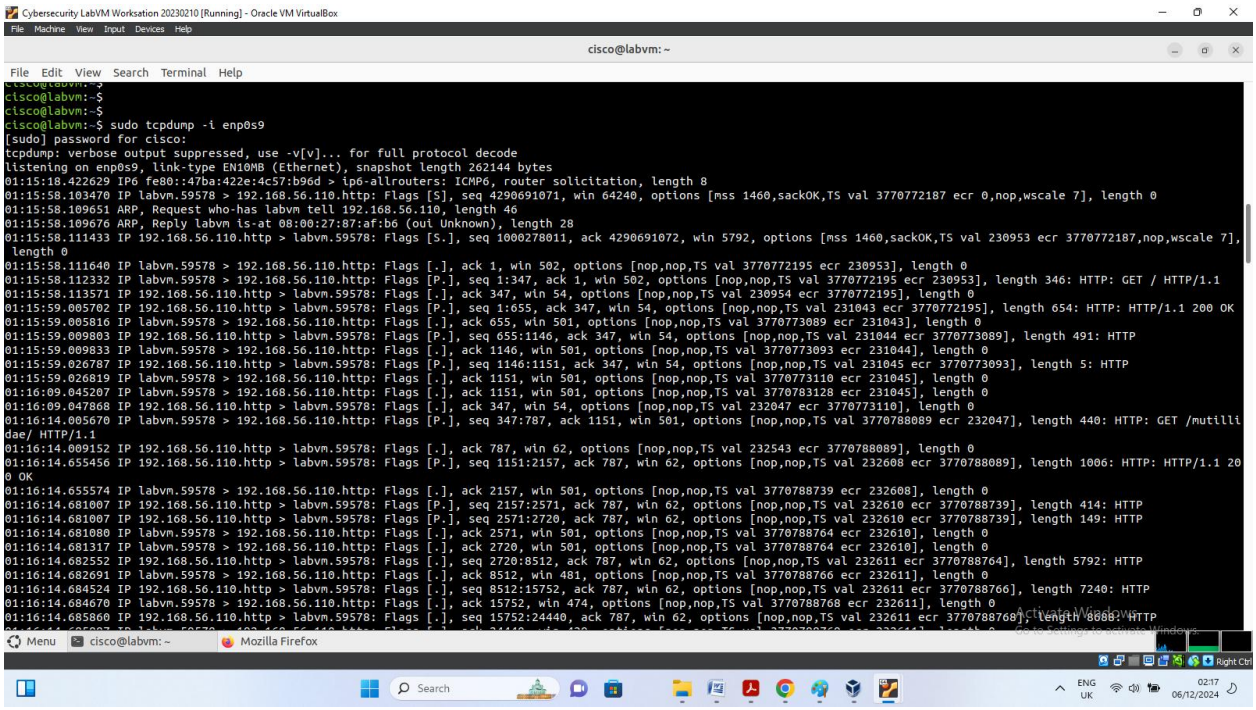


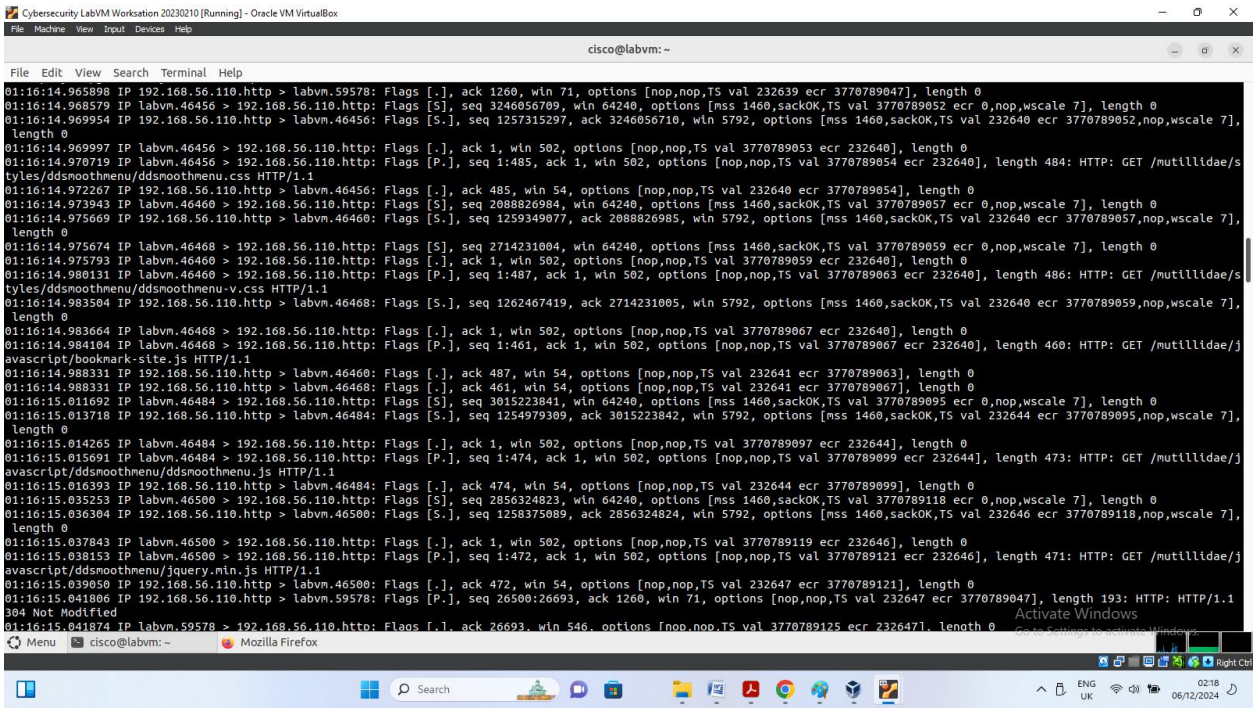
Using iptables:
`iptables` can simulate routing by enabling IP forwarding and setting rules to control traffic flow between networks. It acts as a firewall, allowing, blocking, or redirecting traffic based on source, destination, and protocols.

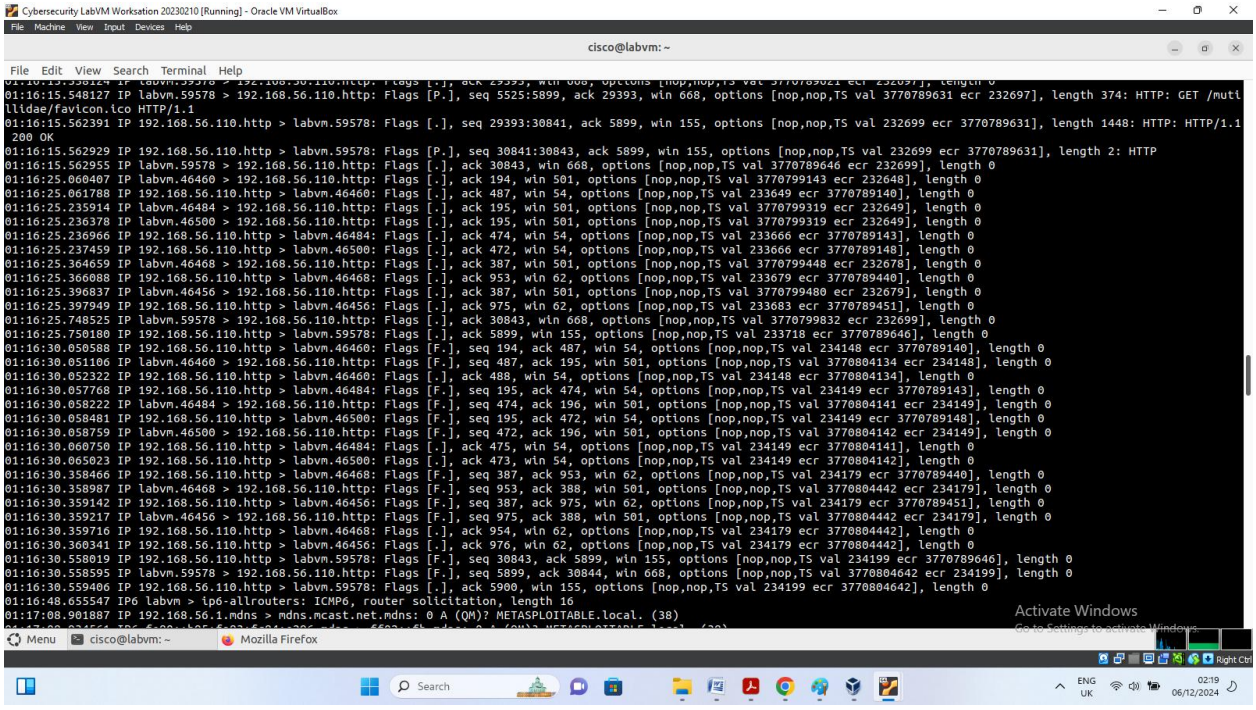
Common mistakes:

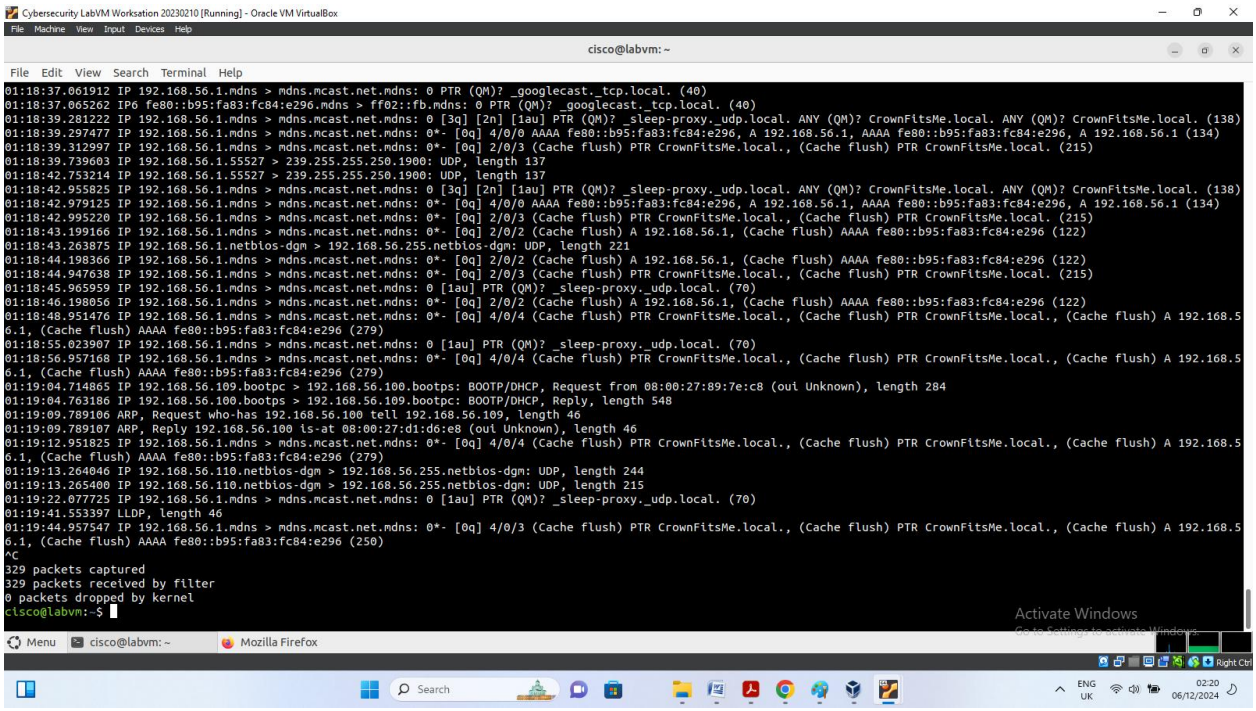
1. Forgetting to enable IP forwarding.
2. Incorrect rule order (iptables processes rules sequentially).
3. Missing default policies, leading to unintended traffic behavior.
4. Overlooking persistence (rules reset after reboot).
5. Blocking essential services (e.g., SSH).

Exercise 6: Monitoring Traffic Using `tcpdump`

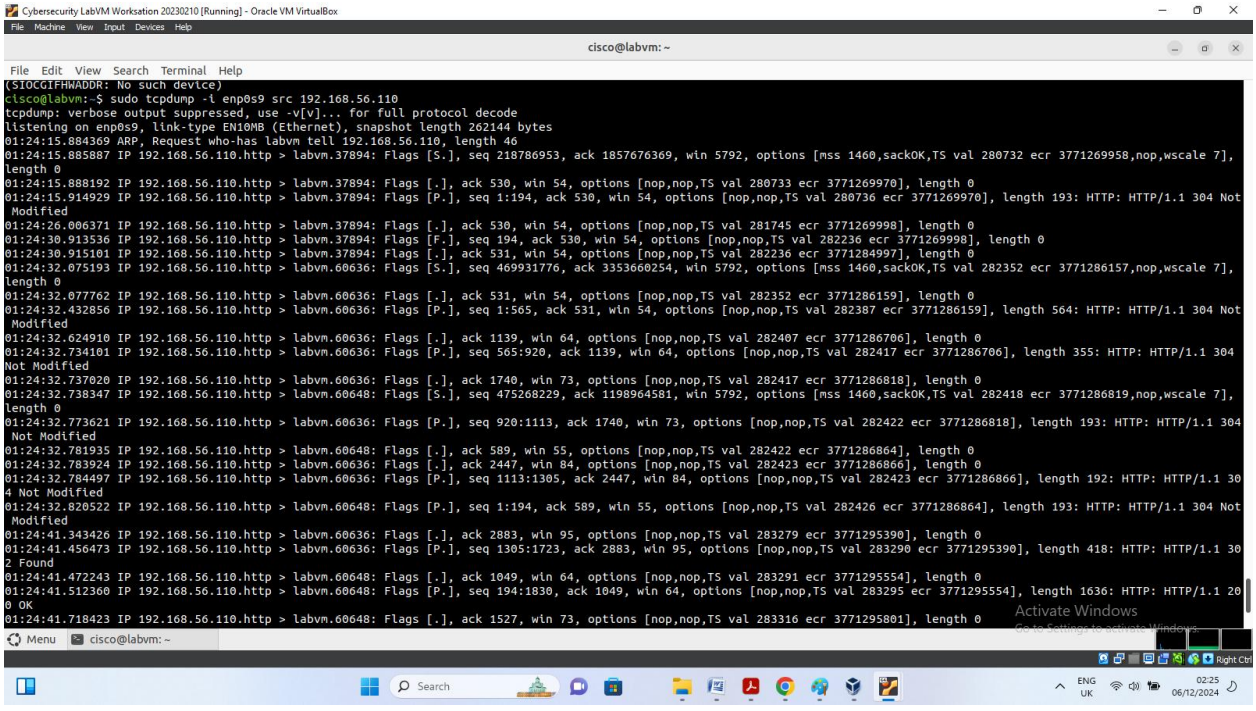


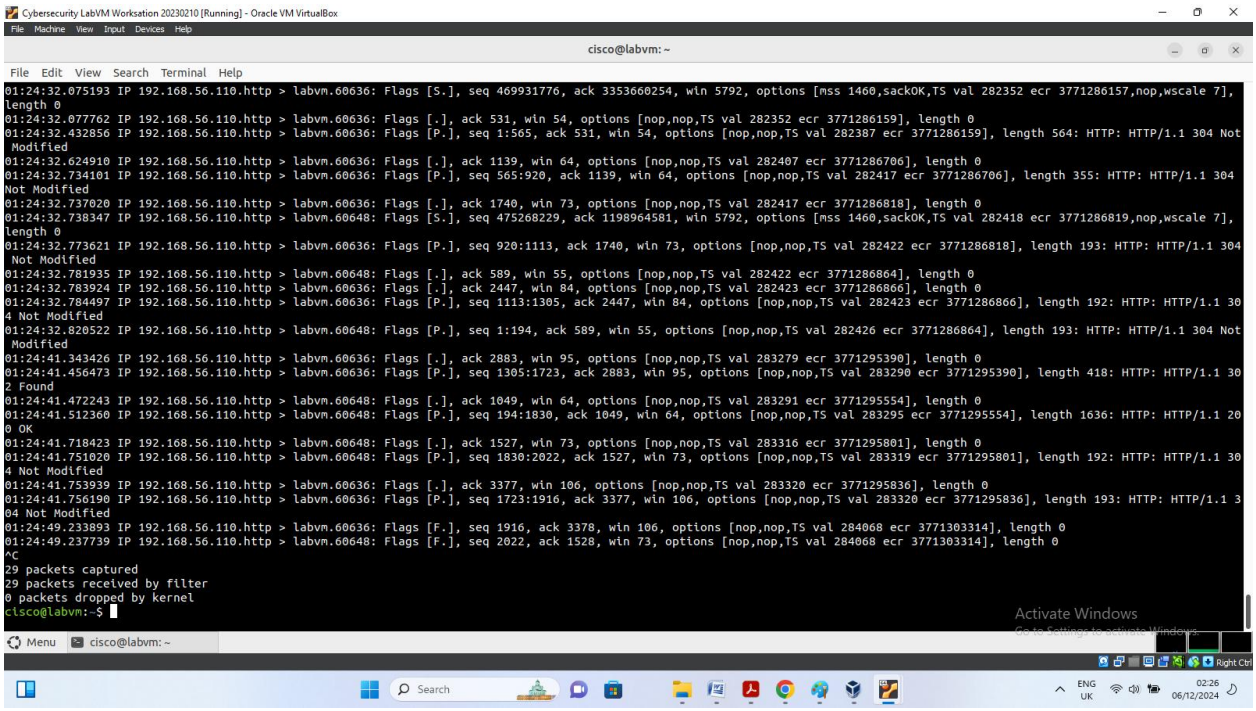

Cybersecurity LabVM Worksation 20230210 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
cisco@labvm: ~
File Edit View Search Terminal Help
01:16:14.965898 IP 192.168.56.110.http > labvm.59578: Flags [.], ack 1260, win 71, options [nop,nop,TS val 232639 ecr 3770789047], length 0
01:16:14.968579 IP labvm.46456 > 192.168.56.110.http: Flags [S], seq 3246056709, win 64240, options [mss 1460,sackOK,TS val 3770789052 ecr 0,nop,wscale 7], length 0
01:16:14.969954 IP 192.168.56.110.http > labvm.46456: Flags [S.], seq 1257315297, ack 3246056710, win 5792, options [mss 1460,sackOK,TS val 232640 ecr 3770789052,nop,wscale 7],
length 0
01:16:14.969997 IP labvm.46456 > 192.168.56.110.http: Flags [.], ack 1, win 502, options [nop,nop,TS val 3770789053 ecr 232640], length 0
01:16:14.970719 IP labvm.46456 > 192.168.56.110.http: Flags [P.], seq 1:485, ack 1, win 502, options [nop,nop,TS val 3770789054 ecr 232640], length 484: HTTP: GET /mutillidae/s
tyles/ddsmoothmenu/ddsmoothmenu.css HTTP/1.1
01:16:14.972267 IP 192.168.56.110.http > labvm.46456: Flags [.], ack 485, win 54, options [nop,nop,TS val 232640 ecr 3770789054], length 0
01:16:14.973943 IP labvm.46460 > 192.168.56.110.http: Flags [S], seq 2088826984, win 64240, options [mss 1460,sackOK,TS val 3770789057 ecr 0,nop,wscale 7], length 0
01:16:14.975669 IP 192.168.56.110.http > labvm.46460: Flags [S.], seq 1259349077, ack 2088826985, win 5792, options [mss 1460,sackOK,TS val 232640 ecr 3770789057,nop,wscale 7],
length 0
01:16:14.975674 IP labvm.46468 > 192.168.56.110.http: Flags [S], seq 2714231004, win 64240, options [mss 1460,sackOK,TS val 3770789059 ecr 0,nop,wscale 7], length 0
01:16:14.975793 IP labvm.46460 > 192.168.56.110.http: Flags [.], ack 1, win 502, options [nop,nop,TS val 3770789059 ecr 232640], length 0
01:16:14.980131 IP labvm.46460 > 192.168.56.110.http: Flags [P.], seq 1:487, ack 1, win 502, options [nop,nop,TS val 3770789063 ecr 232640], length 486: HTTP: GET /mutillidae/s
tyles/ddsmoothmenu/ddsmoothmenu-v.css HTTP/1.1
01:16:14.983504 IP 192.168.56.110.http > labvm.46468: Flags [S.], seq 1262467419, ack 2714231005, win 5792, options [mss 1460,sackOK,TS val 232640 ecr 3770789059,nop,wscale 7],
length 0
01:16:14.983664 IP labvm.46468 > 192.168.56.110.http: Flags [.], ack 1, win 502, options [nop,nop,TS val 3770789067 ecr 232640], length 0
01:16:14.984104 IP labvm.46468 > 192.168.56.110.http: Flags [P.], seq 1:461, ack 1, win 502, options [nop,nop,TS val 3770789067 ecr 232640], length 460: HTTP: GET /mutillidae/j
avascript/bookmark-site.js HTTP/1.1
01:16:14.988331 IP 192.168.56.110.http > labvm.46460: Flags [.], ack 487, win 54, options [nop,nop,TS val 232641 ecr 3770789063], length 0
01:16:14.988331 IP 192.168.56.110.http > labvm.46468: Flags [.], ack 461, win 54, options [nop,nop,TS val 232641 ecr 3770789067], length 0
01:16:15.011692 IP labvm.46484 > 192.168.56.110.http: Flags [S], seq 3015223841, win 64240, options [mss 1460,sackOK,TS val 3770789095 ecr 0,nop,wscale 7], length 0
01:16:15.013718 IP 192.168.56.110.http > labvm.46484: Flags [S.], seq 1254979309, ack 3015223842, win 5792, options [mss 1460,sackOK,TS val 232644 ecr 3770789095,nop,wscale 7],
length 0
01:16:15.014265 IP labvm.46484 > 192.168.56.110.http: Flags [.], ack 1, win 502, options [nop,nop,TS val 3770789097 ecr 232644], length 0
01:16:15.015691 IP labvm.46484 > 192.168.56.110.http: Flags [P.], seq 1:474, ack 1, win 502, options [nop,nop,TS val 3770789099 ecr 232644], length 473: HTTP: GET /mutillidae/j
avascript/ddsmoothmenu/ddsmoothmenu.js HTTP/1.1
01:16:15.016393 IP 192.168.56.110.http > labvm.46484: Flags [.], ack 474, win 54, options [nop,nop,TS val 232644 ecr 3770789099], length 0
01:16:15.035253 IP labvm.46500 > 192.168.56.110.http: Flags [S], seq 2856324823, win 64240, options [mss 1460,sackOK,TS val 3770789118 ecr 0,nop,wscale 7], length 0
01:16:15.036304 IP 192.168.56.110.http > labvm.46500: Flags [S.], seq 1258375089, ack 2856324824, win 5792, options [mss 1460,sackOK,TS val 232646 ecr 3770789118,nop,wscale 7],
length 0
01:16:15.037843 IP labvm.46500 > 192.168.56.110.http: Flags [.], ack 1, win 502, options [nop,nop,TS val 3770789119 ecr 232646], length 0
01:16:15.038153 IP labvm.46500 > 192.168.56.110.http: Flags [P.], seq 1:472, ack 1, win 502, options [nop,nop,TS val 3770789121 ecr 232646], length 471: HTTP: GET /mutillidae/j
avascript/ddsmoothmenu/jquery.min.js HTTP/1.1
01:16:15.039050 IP 192.168.56.110.http > labvm.46500: Flags [.], ack 472, win 54, options [nop,nop,TS val 232647 ecr 3770789121], length 0
01:16:15.041806 IP 192.168.56.110.http > labvm.59578: Flags [P.], seq 26500:26693, ack 1260, win 71, options [nop,nop,TS val 232647 ecr 3770789047], length 193: HTTP: HTTP/1.1
304 Not Modified
01:16:15.041874 IP labvm.59578 > 192.168.56.110.http: Flags [.], ack 26693, win 546, options [nop,nop,TS val 3770789125 ecr 232647], length 0
Activate Windows
Menu
cisco@labvm: ~
Mozilla Firefox
Right Ctrl
Search
ENG
UK
02:18
06/12/2024


Cybersecurity LabVM Worksation 20230210 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
cisco@labvm: ~
File Edit View Search Terminal Help
01:16:15.548127 IP labvm.59578 > 192.168.56.110.http: Flags [P.], seq 5525:5899, ack 29393, win 668, options [nop,nop,TS val 3770789631 ecr 232697], length 374: HTTP: GET /muti
llidae/favicon.ico HTTP/1.1
01:16:15.562391 IP 192.168.56.110.http > labvm.59578: Flags [.], seq 29393:30841, ack 5899, win 155, options [nop,nop,TS val 232699 ecr 3770789631], length 1448: HTTP: HTTP/1.1
200 OK
01:16:15.562929 IP 192.168.56.110.http > labvm.59578: Flags [P.], seq 30841:30843, ack 5899, win 155, options [nop,nop,TS val 232699 ecr 3770789631], length 2: HTTP
01:16:15.562955 IP labvm.59578 > 192.168.56.110.http: Flags [.], ack 30843, win 668, options [nop,nop,TS val 3770789646 ecr 232699], length 0
01:16:25.060407 IP labvm.46460 > 192.168.56.110.http: Flags [.], ack 194, win 501, options [nop,nop,TS val 3770799143 ecr 232648], length 0
01:16:25.061788 IP 192.168.56.110.http > labvm.46460: Flags [.], ack 487, win 54, options [nop,nop,TS val 233649 ecr 3770789140], length 0
01:16:25.235914 IP labvm.46484 > 192.168.56.110.http: Flags [.], ack 195, win 501, options [nop,nop,TS val 3770799319 ecr 232649], length 0
01:16:25.236378 IP labvm.46500 > 192.168.56.110.http: Flags [.], ack 195, win 501, options [nop,nop,TS val 3770799319 ecr 232649], length 0
01:16:25.236966 IP 192.168.56.110.http > labvm.46484: Flags [.], ack 474, win 54, options [nop,nop,TS val 233666 ecr 3770789143], length 0
01:16:25.237459 IP 192.168.56.110.http > labvm.46500: Flags [.], ack 472, win 54, options [nop,nop,TS val 233666 ecr 3770789148], length 0
01:16:25.364659 IP labvm.46468 > 192.168.56.110.http: Flags [.], ack 387, win 501, options [nop,nop,TS val 3770799448 ecr 232678], length 0
01:16:25.366088 IP 192.168.56.110.http > labvm.46468: Flags [.], ack 953, win 62, options [nop,nop,TS val 233679 ecr 3770789440], length 0
01:16:25.396837 IP labvm.46456 > 192.168.56.110.http: Flags [.], ack 387, win 501, options [nop,nop,TS val 3770799480 ecr 232679], length 0
01:16:25.397949 IP 192.168.56.110.http > labvm.46456: Flags [.], ack 975, win 62, options [nop,nop,TS val 233683 ecr 3770789451], length 0
01:16:25.748525 IP labvm.59578 > 192.168.56.110.http: Flags [.], ack 30843, win 668, options [nop,nop,TS val 3770799832 ecr 232699], length 0
01:16:25.750180 IP 192.168.56.110.http > labvm.59578: Flags [.], ack 5899, win 155, options [nop,nop,TS val 233718 ecr 3770789646], length 0
01:16:30.050588 IP 192.168.56.110.http > labvm.46460: Flags [F.], seq 194, ack 487, win 54, options [nop,nop,TS val 234148 ecr 3770789140], length 0
01:16:30.051106 IP labvm.46460 > 192.168.56.110.http: Flags [F.], seq 487, ack 195, win 501, options [nop,nop,TS val 3770804134 ecr 234148], length 0
01:16:30.052322 IP 192.168.56.110.http > labvm.46460: Flags [.], ack 488, win 54, options [nop,nop,TS val 234148 ecr 3770804134], length 0
01:16:30.057768 IP 192.168.56.110.http > labvm.46484: Flags [F.], seq 195, ack 474, win 54, options [nop,nop,TS val 234149 ecr 3770789143], length 0
01:16:30.058222 IP labvm.46484 > 192.168.56.110.http: Flags [F.], seq 474, ack 196, win 501, options [nop,nop,TS val 3770804141 ecr 234149], length 0
01:16:30.058481 IP 192.168.56.110.http > labvm.46500: Flags [F.], seq 195, ack 472, win 54, options [nop,nop,TS val 234149 ecr 3770789148], length 0
01:16:30.058759 IP labvm.46500 > 192.168.56.110.http: Flags [F.], seq 472, ack 196, win 501, options [nop,nop,TS val 3770804142 ecr 234149], length 0
01:16:30.060750 IP 192.168.56.110.http > labvm.46484: Flags [.], ack 475, win 54, options [nop,nop,TS val 234149 ecr 3770804141], length 0
01:16:30.065023 IP 192.168.56.110.http > labvm.46500: Flags [.], ack 473, win 54, options [nop,nop,TS val 234149 ecr 3770804142], length 0
01:16:30.358466 IP 192.168.56.110.http > labvm.46468: Flags [F.], seq 387, ack 953, win 62, options [nop,nop,TS val 234179 ecr 3770789440], length 0
01:16:30.358987 IP labvm.46468 > 192.168.56.110.http: Flags [F.], seq 953, ack 388, win 501, options [nop,nop,TS val 3770804442 ecr 234179], length 0
01:16:30.359142 IP 192.168.56.110.http > labvm.46456: Flags [F.], seq 387, ack 975, win 62, options [nop,nop,TS val 234179 ecr 3770789451], length 0
01:16:30.359217 IP labvm.46456 > 192.168.56.110.http: Flags [F.], seq 975, ack 388, win 501, options [nop,nop,TS val 3770804442 ecr 234179], length 0
01:16:30.359716 IP 192.168.56.110.http > labvm.46468: Flags [.], ack 954, win 62, options [nop,nop,TS val 234179 ecr 3770804442], length 0
01:16:30.360341 IP 192.168.56.110.http > labvm.46456: Flags [.], ack 976, win 62, options [nop,nop,TS val 234179 ecr 3770804442], length 0
01:16:30.558019 IP 192.168.56.110.http > labvm.59578: Flags [F.], seq 30843, ack 5899, win 155, options [nop,nop,TS val 234199 ecr 3770789646], length 0
01:16:30.558595 IP labvm.59578 > 192.168.56.110.http: Flags [F.], seq 5899, ack 30844, win 668, options [nop,nop,TS val 3770804642 ecr 234199], length 0
01:16:30.559406 IP 192.168.56.110.http > labvm.59578: Flags [.], ack 5900, win 155, options [nop,nop,TS val 234199 ecr 3770804642], length 0
01:16:48.655547 IP6 labvm > ip6-allrouters: ICMP6, router solicitation, length 16
01:17:08.901887 IP 192.168.56.1.mdns > mdns.mcast.net.mdns: 0 A (QM)? METASPLOITABLE.local. (38)
Activate Windows
Menu
cisco@labvm: ~
Mozilla Firefox
Right Ctrl
Search
ENG
UK
02:19
06/12/2024



Data from OWASP





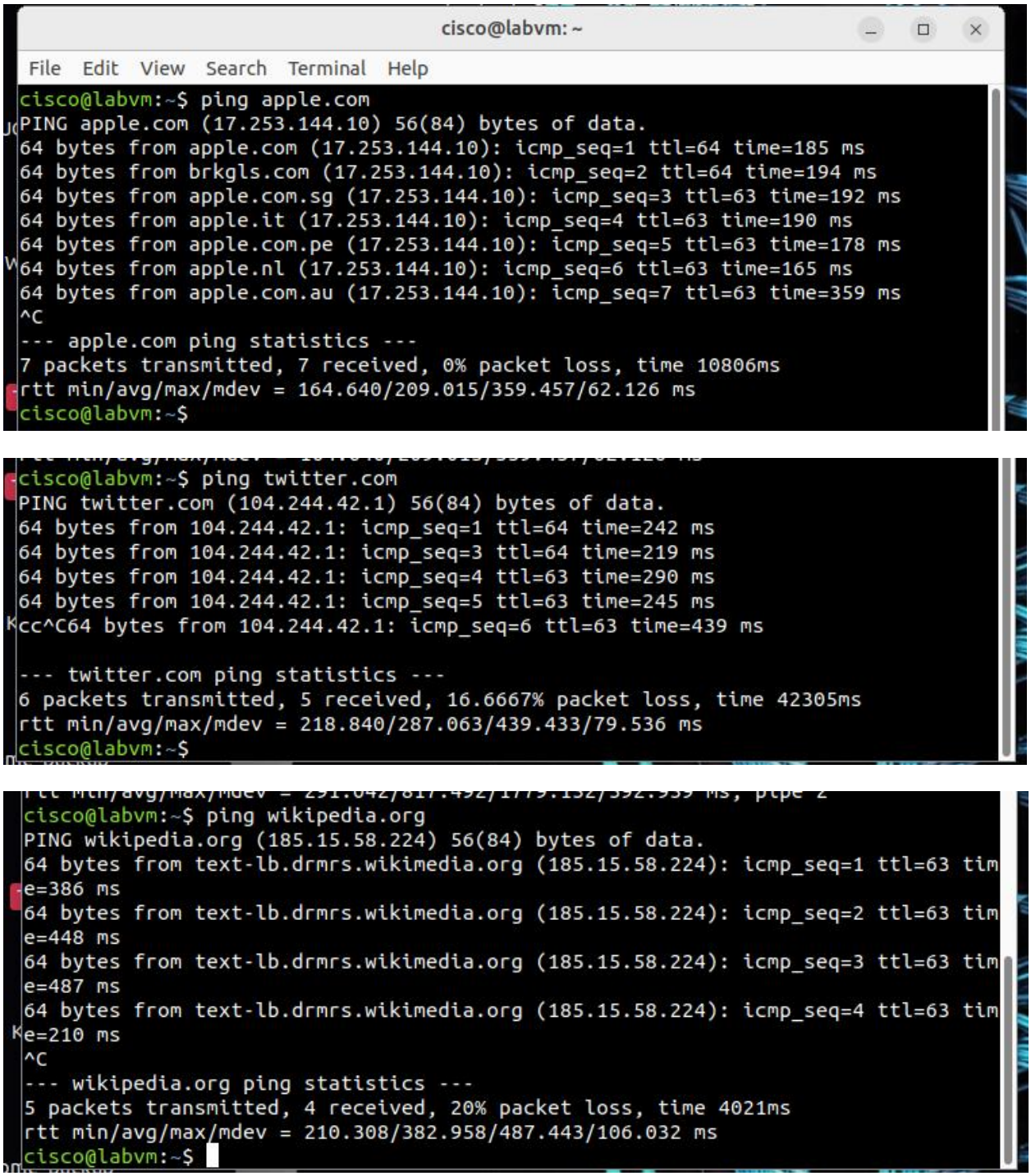
INT303: Networking Fundamentals – Lab 5

Please note that professional report required for submission is attached immediately after the lab test screen shots and explanation

Step 1: Choose 10 Popular Websites

Include a mix of domain types for variety:

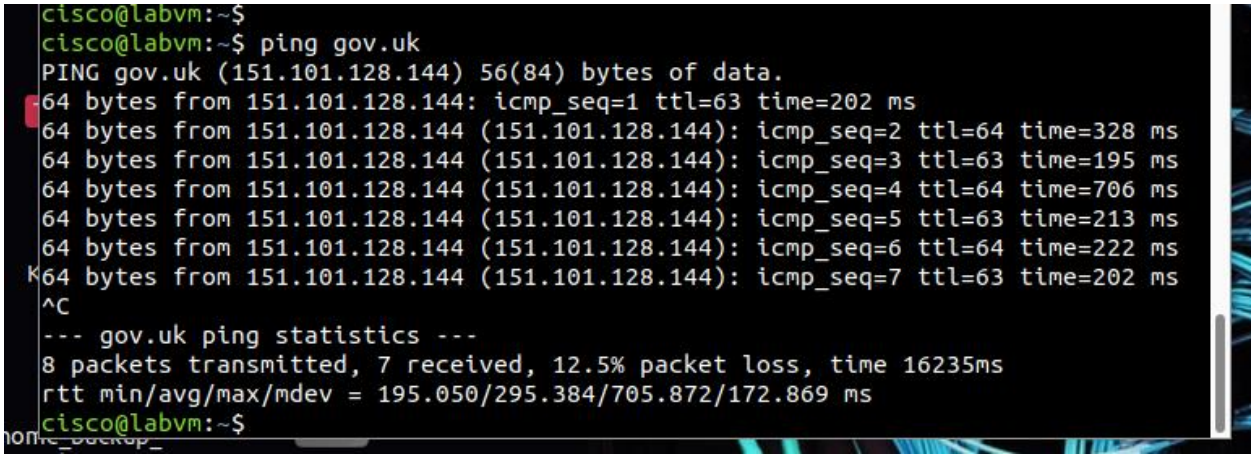
1. **apple.com** (.com - Technology)
2. **twitter.com** (.com - Social Media)
3. **wikipedia.org** (.org - Nonprofit)
4. **harvard.edu** (.edu - Educational Institution)
5. **bbc.co.uk** (.co.uk - Media, UK)
6. **gov.uk** (.gov.uk - Government, UK)
7. **cnn.com** (.com - News Media)
8. **un.org** (.org - United Nations)
9. **spiegel.de** (.de - German News Media)
10. **google.com** (.com - Technology)

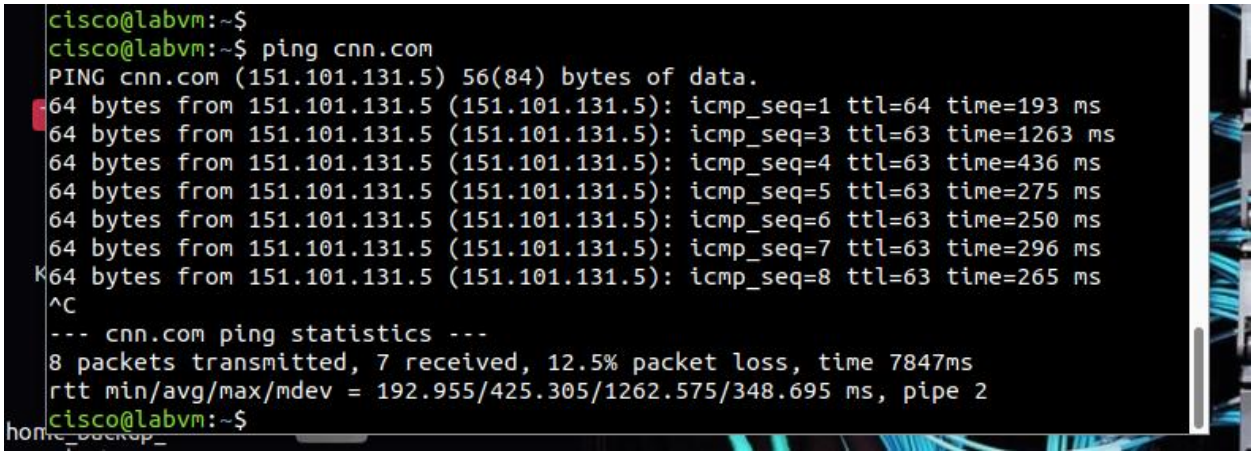

cisco@labvm: ~
File Edit View Search Terminal Help
cisco@labvm:~$ ping apple.com
PING apple.com (17.253.144.10) 56(84) bytes of data.
64 bytes from apple.com (17.253.144.10): icmp_seq=1 ttl=64 time=185 ms
64 bytes from brkgls.com (17.253.144.10): icmp_seq=2 ttl=64 time=194 ms
64 bytes from apple.com.sg (17.253.144.10): icmp_seq=3 ttl=63 time=192 ms
64 bytes from apple.it (17.253.144.10): icmp_seq=4 ttl=63 time=190 ms
64 bytes from apple.com.pe (17.253.144.10): icmp_seq=5 ttl=63 time=178 ms
64 bytes from apple.nl (17.253.144.10): icmp_seq=6 ttl=63 time=165 ms
64 bytes from apple.com.au (17.253.144.10): icmp_seq=7 ttl=63 time=359 ms
^C
--- apple.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 10806ms
rtt min/avg/max/mdev = 164.640/209.015/359.457/62.126 ms
cisco@labvm:~$
cisco@labvm:~$ ping twitter.com
PING twitter.com (104.244.42.1) 56(84) bytes of data.
64 bytes from 104.244.42.1: icmp_seq=1 ttl=64 time=242 ms
64 bytes from 104.244.42.1: icmp_seq=3 ttl=64 time=219 ms
64 bytes from 104.244.42.1: icmp_seq=4 ttl=63 time=290 ms
64 bytes from 104.244.42.1: icmp_seq=5 ttl=63 time=245 ms
cc^C64 bytes from 104.244.42.1: icmp_seq=6 ttl=63 time=439 ms
--- twitter.com ping statistics ---
6 packets transmitted, 5 received, 16.6667% packet loss, time 42305ms
rtt min/avg/max/mdev = 218.840/287.063/439.433/79.536 ms
cisco@labvm:~$
cisco@labvm:~$ ping wikipedia.org
PING wikipedia.org (185.15.58.224) 56(84) bytes of data.
64 bytes from text-lb.drmrs.wikimedia.org (185.15.58.224): icmp_seq=1 ttl=63 time=386 ms
64 bytes from text-lb.drmrs.wikimedia.org (185.15.58.224): icmp_seq=2 ttl=63 time=448 ms
64 bytes from text-lb.drmrs.wikimedia.org (185.15.58.224): icmp_seq=3 ttl=63 time=487 ms
64 bytes from text-lb.drmrs.wikimedia.org (185.15.58.224): icmp_seq=4 ttl=63 time=210 ms
^C
--- wikipedia.org ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4021ms
rtt min/avg/max/mdev = 210.308/382.958/487.443/106.032 ms
cisco@labvm:~$

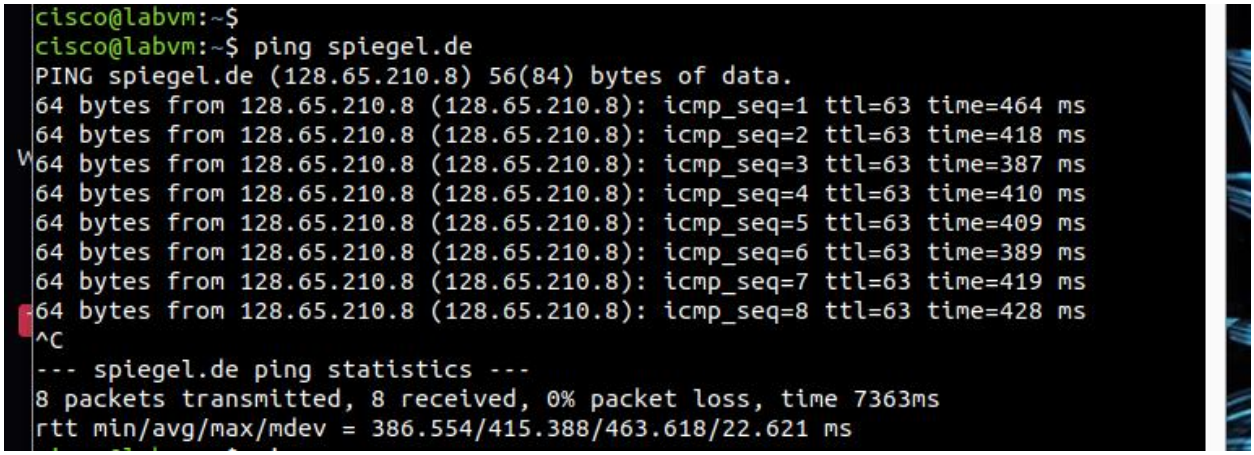
```
cisco@labvm:~$
cisco@labvm:~$ ping harvard.edu
PING harvard.edu (192.0.66.20) 56(84) bytes of data.
64 bytes from 192.0.66.20 (192.0.66.20): icmp_seq=1 ttl=63 time=179 ms
64 bytes from 192.0.66.20 (192.0.66.20): icmp_seq=2 ttl=63 time=172 ms
64 bytes from 192.0.66.20 (192.0.66.20): icmp_seq=3 ttl=63 time=190 ms
64 bytes from 192.0.66.20 (192.0.66.20): icmp_seq=4 ttl=63 time=204 ms
^C
--- harvard.edu ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 8374ms
rtt min/avg/max/mdev = 171.726/185.988/203.628/12.051 ms
cisco@labvm:~$
```

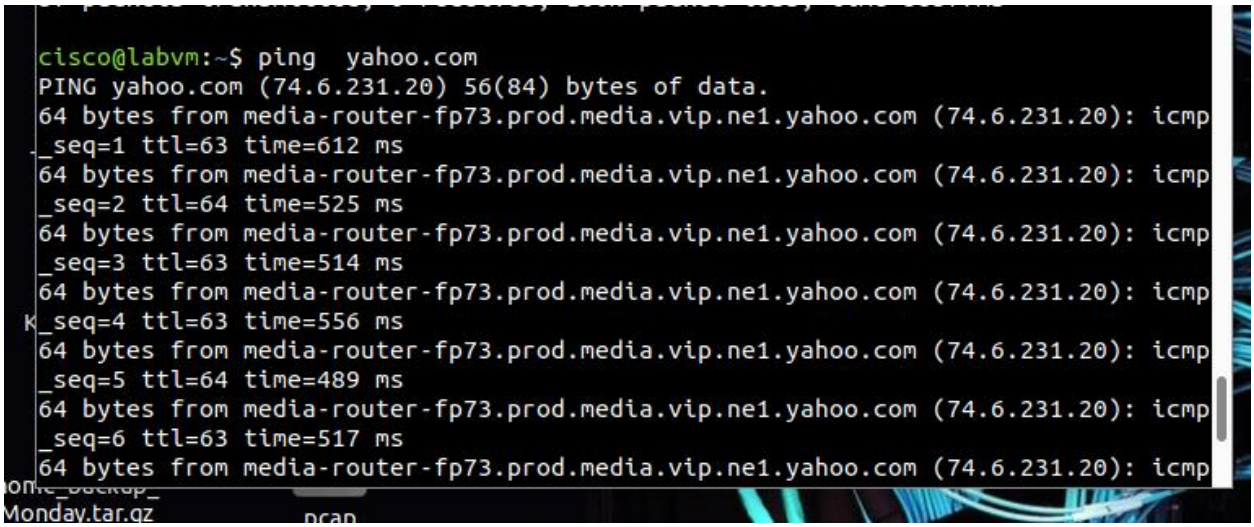
```
cisco@labvm:~$ ping bbc.co.uk
PING bbc.co.uk (151.101.0.81) 56(84) bytes of data.
64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=1 ttl=63 time=457 ms
64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=2 ttl=63 time=215 ms
64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=3 ttl=63 time=228 ms
64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=4 ttl=63 time=231 ms
64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=5 ttl=63 time=231 ms
c64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=6 ttl=63 time=218 ms
c64 bytes from 151.101.0.81 (151.101.0.81): icmp_seq=7 ttl=63 time=216 ms
^C
--- bbc.co.uk ping statistics ---
8 packets transmitted, 7 received, 12.5% packet loss, time 11762ms
rtt min/avg/max/mdev = 215.369/256.733/457.421/82.175 ms
cisco@labvm:~$
```

```
cisco@labvm:~$ ping google.com
PING google.com (142.250.185.14) 56(84) bytes of data.
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=1 ttl=63 time
=203 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=2 ttl=64 time
=205 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=3 ttl=64 time
=207 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=4 ttl=63 time
=206 ms
c64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=5 ttl=63 tim
e=202 ms
64 bytes from mad41s11-in-f14.1e100.net (142.250.185.14): icmp_seq=6 ttl=64 time
=196 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5024ms
rtt min/avg/max/mdev = 196.070/203.176/207.361/3.618 ms
```


cisco@labvm:~$
cisco@labvm:~$ ping gov.uk
PING gov.uk (151.101.128.144) 56(84) bytes of data.
64 bytes from 151.101.128.144: icmp_seq=1 ttl=63 time=202 ms
64 bytes from 151.101.128.144 (151.101.128.144): icmp_seq=2 ttl=64 time=328 ms
64 bytes from 151.101.128.144 (151.101.128.144): icmp_seq=3 ttl=63 time=195 ms
64 bytes from 151.101.128.144 (151.101.128.144): icmp_seq=4 ttl=64 time=706 ms
64 bytes from 151.101.128.144 (151.101.128.144): icmp_seq=5 ttl=63 time=213 ms
64 bytes from 151.101.128.144 (151.101.128.144): icmp_seq=6 ttl=64 time=222 ms
64 bytes from 151.101.128.144 (151.101.128.144): icmp_seq=7 ttl=63 time=202 ms
^C
--- gov.uk ping statistics ---
8 packets transmitted, 7 received, 12.5% packet loss, time 16235ms
rtt min/avg/max/mdev = 195.050/295.384/705.872/172.869 ms
cisco@labvm:~$


cisco@labvm:~$
cisco@labvm:~$ ping cnn.com
PING cnn.com (151.101.131.5) 56(84) bytes of data.
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=1 ttl=64 time=193 ms
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=3 ttl=63 time=1263 ms
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=4 ttl=63 time=436 ms
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=5 ttl=63 time=275 ms
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=6 ttl=63 time=250 ms
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=7 ttl=63 time=296 ms
64 bytes from 151.101.131.5 (151.101.131.5): icmp_seq=8 ttl=63 time=265 ms
^C
--- cnn.com ping statistics ---
8 packets transmitted, 7 received, 12.5% packet loss, time 7847ms
rtt min/avg/max/mdev = 192.955/425.305/1262.575/348.695 ms, pipe 2
cisco@labvm:~$


cisco@labvm:~$
cisco@labvm:~$ ping spiegel.de
PING spiegel.de (128.65.210.8) 56(84) bytes of data.
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=1 ttl=63 time=464 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=2 ttl=63 time=418 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=3 ttl=63 time=387 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=4 ttl=63 time=410 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=5 ttl=63 time=409 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=6 ttl=63 time=389 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=7 ttl=63 time=419 ms
64 bytes from 128.65.210.8 (128.65.210.8): icmp_seq=8 ttl=63 time=428 ms
^C
--- spiegel.de ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7363ms
rtt min/avg/max/mdev = 386.554/415.388/463.618/22.621 ms



Step 3: Document the IP Addresses

3. Create a table to record the IP addresses for each website you ping.

Serial number	Website	Ip Address
1	Apple.com	17.253.144.10
2	Twitter.com	104.244.42.1
3	Wikipedia.org	185.15.58.224
4	Harvard.edu	192.0.66.20
5	Google.com	142.250.185.14
6	Bbc.co.uk	151.101.0.81
7	Gov.uk	151.101.128.144
8	Cnn.com	151.101.131.5
9	Spiegel.de	128.65.210.8
10	Sfczonkwa.com.ng	108.166.183.137

Step 4: Classify the IP Addresses

5. Determine the **class** (A, B, or C) of each IP address based on its first octet

Let's classify each IP address based on its first octet:

1. **17.253.144.10**
    - **First Octet:** 17
    - **Class:** A (1-126)
2. **104.244.42.1**
    - **First Octet:** 104

    - **Class:** A (1-126)
3. **185.15.58.224**
    - **First Octet:** 185
    - **Class:** B (128-191)
4. **192.0.66.20**
    - **First Octet:** 192
    - **Class:** C (192-223)
5. **142.250.185.14**
    - **First Octet:** 142
    - **Class:** B (128-191)
6. **151.101.0.81**
    - **First Octet:** 151
    - **Class:** B (128-191)
7. **151.101.128.144**
    - **First Octet:** 151
    - **Class:** B (128-191)
8. **151.101.131.5**
    - **First Octet:** 151
    - **Class:** B (128-191)
9. **128.65.210.8**
    - **First Octet:** 128
    - **Class:** B (128-191)
10. **108.166.183.137**
    - **First Octet:** 108
    - **Class:** A (1-126)

IP Address	Class
17.253.144.10	A
104.244.42.1	A
185.15.58.224	B
192.0.66.20	C
142.250.185.14	B
151.101.0.81	B
151.101.128.144	B
151.101.131.5	B
128.65.210.8	B
108.166.183.137	A

Here's a quick overview of IP address classes based on their first octet:

- **Class A:** 1-126
- **Class B:** 128-191
- **Class C:** 192-223

Common Reserved IP Ranges

Reserved IPs are defined by the Internet Assigned Numbers Authority (IANA) in RFC 1918 and other RFCs for private and special-use purposes. These ranges are not routable on the public Internet and are primarily used in internal networks.

Here are the key ranges:

Private IP Ranges (RFC 1918):

1. **10.0.0.0/8**
    - Range: 10.0.0.0 – 10.255.255.255
    - Total IPs: ~16.7 million
    - Usage: Large private networks (e.g., enterprises or ISPs).
2. **172.16.0.0/12**

- Range: 172.16.0.0 – 172.31.255.255
- Total IPs: ~1 million
- Usage: Medium-sized private networks.

3. **192.168.0.0/16**
   - Range: 192.168.0.0 – 192.168.255.255
   - Total IPs: ~65,536
   - Usage: Small/home networks.

Other Special Reserved IP Ranges:

1. **Loopback (127.0.0.0/8)**
   - Range: 127.0.0.1 – 127.255.255.255
   - Usage: Testing and internal host communication (e.g., `localhost`).
2. **Link-Local (169.254.0.0/16)**
   - Usage: Automatic private IP addressing when no DHCP is available.
3. **Multicast (224.0.0.0/4)**
   - Range: 224.0.0.0 – 239.255.255.255
   - Usage: Multicast traffic (e.g., video streams).
4. **Documentation (192.0.2.0/24, 198.51.100.0/24, 203.0.113.0/24)**
   - Usage: Example addresses for documentation.
5. **Carrier-Grade NAT (100.64.0.0/10)**
   - Range: 100.64.0.0 – 100.127.255.255
   - Usage: NAT in ISP-level networks.
6. **Broadcast Address (255.255.255.255)**
   - Usage: Network-wide broadcasts.

Importance of Reserved IPs in Network Configurations

1. **Network Segmentation and Scalability**
   - Reserved IPs, especially private ranges, enable organizations to create isolated networks. Without them, IPv4 exhaustion would prevent the scalability of modern networks.
2. **Security**
   - Private IPs prevent external access by default since they are not routable over the public Internet. This makes them ideal for internal systems.
3. **Cost-Effectiveness**
   - Organizations can reuse these IP ranges internally without acquiring public IPs for every device.
4. **IP Conservation**
   - Private IPs help mitigate the exhaustion of IPv4 space. Technologies like NAT allow private networks to share a single public IP for external communication.
5. **Multicast and Routing Functions**
   - Reserved ranges for multicast and loopback ensure proper routing and application behavior. For example, multicast addresses are used for real-time applications like video conferencing.
6. **Testing and Documentation**

- Reserved ranges like 192.0.2.0/24 help avoid conflicts in testing and training environments.

How to Identify Reserved IPs in Your Network

To check if any IPs in your network fall into reserved ranges:

1. Identify all subnets in use.
2. Compare them against reserved ranges (e.g., using tools like IP calculators or scripting with Python).
3. Cross-reference against DHCP and static IP configurations.

Step 5: Calculate the Number of Devices Each IP Can Support

Device Capacity for Each IP Class

1. **Class A:**
   - Formula: 224−22^{24} - 2
   - Calculation: 16,777,216−2=16,777,21416,777,216 - 2 = 16,777,214 devices
2. **Class B:**
   - Formula: 216−22^{16} - 2
   - Calculation: 65,536−2=65,53465,536 - 2 = 65,534 devices
3. **Class C:**
   - Formula: 28−22^{8} - 2
   - Calculation: 256−2=254256 - 2 = 254 devices

Bonus Exercise: Subnetting

Class B Network Subnetting

Let's subdivide a Class B network into smaller subnets.

- **Original Class B Network:**
  - Total devices: 65,53465,534
- **Subnetting Example 1: /24 Subnet Mask (Subnetting a Class B into Class C sized subnets)**
  - New Subnet Mask: 255.255.255.0 (/24)
  - Subnets created: 28=2562^{8} = 256 subnets
  - Devices per subnet: 28−2=2542^{8} - 2 = 254 devices
- **Subnetting Example 2: /26 Subnet Mask**
  - New Subnet Mask: 255.255.255.192 (/26)
  - Subnets created: 210=1,0242^{10} = 1,024 subnets
  - Devices per subnet: 26−2=622^{6} - 2 = 62 devices

Class C Network Subnetting

Let's consider subnet masks like /26 and /28 for Class C networks.

- **Original Class C Network:**
  - Total devices: 254254
- **Subnetting Example 1: /26 Subnet Mask**
  - New Subnet Mask: 255.255.255.192 (/26)
  - Subnets created: 22=42^{2} = 4 subnets
  - Devices per subnet: 26−2=622^{6} - 2 = 62 devices
- **Subnetting Example 2: /28 Subnet Mask**
  - New Subnet Mask: 255.255.255.240 (/28)
  - Subnets created: 24=162^{4} = 16 subnets
  - Devices per subnet: 24−2=142^{4} - 2 = 14 devices

By performing subnetting, we can break down larger networks into smaller, more manageable subnets, each with its own device capacity. This enhances network organization and scalability.

Step 6: Determine the Subnet Mask

Default Subnet Masks for Each IP Address Class

1. **Class A:**
   - Default Subnet Mask: 255.0.0.0
2. **Class B:**
   - Default Subnet Mask: 255.255.0.0
3. **Class C:**
   - Default Subnet Mask: 255.255.255.0

Bonus Exercise: Custom Subnetting and Its Impact

Scenario: Custom Subnetting for Network Segmentation

Custom subnetting is often required for network segmentation, which helps in improving network performance, security, and management. By altering the subnet mask, we can create smaller subnets within a larger network, each with its own set of IP addresses.

For example, in an organization, different departments (e.g., HR, Sales, IT) might need separate subnets to isolate their traffic and enhance security.

Impact on Device Capacity and Network Organization

1. **Subnet Mask /25 (255.255.255.128):**
   - **Class C Network:**
     - Creates 2 subnets.

        - Each subnet has 128 addresses (2^7) minus 2 for network and broadcast addresses.
        - Usable hosts per subnet: 126
2. **Subnet Mask /27 (255.255.255.224):**
    - **Class C Network:**
        - Creates 8 subnets.
        - Each subnet has 32 addresses (2^5) minus 2 for network and broadcast addresses.
        - Usable hosts per subnet: 30

Example Calculation

Original Class C Network (Default Mask /24):

- Total usable addresses: 254

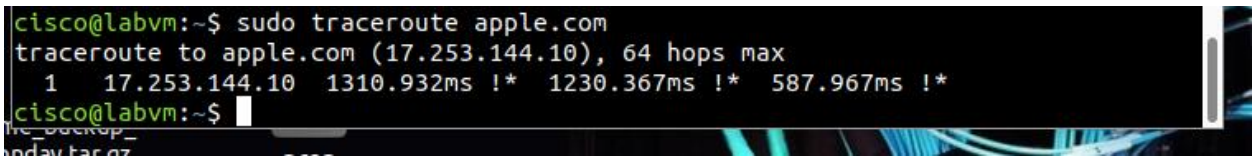
With Subnet Mask /25:

- Number of Subnets: 2
- Usable Hosts per Subnet: 126

With Subnet Mask /27:

- Number of Subnets: 8
- Usable Hosts per Subnet: 30

Step 7: Advanced IP Tools (Bonus Exercises)



1. **First (and Only) Hop (17.253.144.10):**
    - The IP address `17.253.144.10` is directly associated with `apple.com`.
    - **Response Times:** The response times for this hop are 1310.932ms, 1230.367ms, and 587.967ms. These times indicate how long it took for the packets to travel to and from this address.
    - *Annotations (!*):* The `!*` symbol indicates an administrative filter preventing the traceroute from progressing. This usually means that further probing is blocked or the destination is configured not to respond to further traceroute queries.

Understanding the Output

- **Single Hop:** The traceroute shows only one hop, which directly reaches the destination IP address `17.253.144.10`.

- **Response Times:** The relatively high response times (especially the first two) suggest either network latency or load on the destination server.
- *Annotations (!*):* The `!*` annotation signifies that this hop is filtered or blocked, commonly by a firewall or other network security mechanism.

This traceroute scan reveals that the packet reaches `apple.com` (17.253.144.10) in a single hop with notable response times. The `!*` annotation suggests that further probing is blocked, likely due to security measures at the destination. This indicates the presence of administrative controls, ensuring that network probing tools like traceroute cannot gather more detailed routing information.

Report of IP Address Analysis and Subnetting

1. Introduction

The purpose of this lab is to analyze and classify IP addresses, calculate their device capacities, and explore subnetting techniques for improved network organization. IP address analysis and subnetting are critical for managing network scalability, security, and efficiency. This report examines a set of websites, their corresponding IP addresses, and how advanced tools like traceroute and subnetting strategies can optimize real-world network configurations.

2. Website List and IP Addresses

Serial Number	Website	IP Address
1	apple.com	17.253.144.10
2	twitter.com	104.244.42.1
3	wikipedia.org	185.15.58.224
4	harvard.edu	192.0.66.20
5	google.com	142.250.185.14
6	bbc.co.uk	151.101.0.81
7	gov.uk	151.101.128.144
8	cnn.com	151.101.131.5
9	spiegel.de	128.65.210.8
10	sfczonkwa.com.ng	108.166.183.137

3. IP Address Classification

The IP addresses were classified based on their first octet, using the IPv4 class system:

IP Address	First Octet	Class	Range
17.253.144.10	17	A	1-126
104.244.42.1	104	A	1-126
185.15.58.224	185	B	128-191
192.0.66.20	192	C	192-223
142.250.185.14	142	B	128-191
151.101.0.81	151	B	128-191
151.101.128.144	151	B	128-191
151.101.131.5	151	B	128-191
128.65.210.8	128	B	128-191
108.166.183.137	108	A	1-126

Characteristics of Classes:

- **Class A**: Large networks (e.g., enterprises); supports 224−22^{24} - 2224−2 devices.
- **Class B**: Medium-sized networks; supports 216−22^{16} - 2216−2 devices.
- **Class C**: Small networks; supports 28−22^8 - 228−2 devices.

4. Number of Devices Supported

IP Address	Class	Device Capacity (Usable Hosts)
17.253.144.10	A	16,777,214
104.244.42.1	A	16,777,214
185.15.58.224	B	65,534
192.0.66.20	C	254
142.250.185.14	B	65,534
151.101.0.81	B	65,534
151.101.128.144	B	65,534
151.101.131.5	B	65,534
128.65.210.8	B	65,534
108.166.183.137	A	16,777,214

5. Subnet Masks

Default subnet masks and possible custom subnetting scenarios:

IP Address	Class	Default Subnet Mask	Custom Subnetting	Usable Hosts
17.253.144.10	A	255.0.0.0	/24 (Class C size subnet)	254
104.244.42.1	A	255.0.0.0	/22	1,022
185.15.58.224	B	255.255.0.0	/26	62
192.0.66.20	C	255.255.255.0	/27	30

6. Advanced Tools and Analysis

Traceroute Analysis

- **Target**: apple.com (17.253.144.10)
- **Response Times**:
  - 1310.932ms, 1230.367ms, 587.967ms
- **Annotations (`!*`)**: Indicates administrative filters, likely firewalls, blocking further probing.

Significance of Tools:

- **Traceroute**: Identifies network hops and latency, useful for troubleshooting.
- **GeoIP Analysis**: Maps IP addresses to physical locations, aiding in network security and performance monitoring.

---

7. Conclusion

This analysis highlights the importance of IP address management and subnetting in modern networking. Classifying IP addresses, calculating device capacities, and utilizing custom subnet masks ensure optimal network performance, scalability, and security. Tools like traceroute and GeoIP analysis further enhance troubleshooting and efficiency in real-world scenario