PES UNIVERSITY

Department of Computer Science & Engineering

Computer Network Security

UE23CS343AB6

Assignment 8 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

Computer Network Security

UE23CS343AB6

Explanation of the config files as part of this lab setup:

This lab uses a small multi-container environment (router container + helper code) implemented with Docker images, source code and host-mounted volumes. The top-level tree is:

```
[pranavhemanth@Pranavs-MacBook-Pro-M3 CNS-S5 %cd Lab8                                ]
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab8 %tree                                     ]
.
├── docker-compose.yml
├── Hosts
│   ├── Dockerfile
│   └── sshd_config
├── UE22CS343AB6_9ca045b2-6740-42be-a3c2-4129f7514611_20241024102036.pdf
└── volumes
    ├── B-Socks-Client.py
    ├── B1-B2-Socks-Client.py
    └── socks_client.py

3 directories, 7 files
pranavhemanth@Pranavs-MacBook-Pro-M3 Lab8 %
```

Below is the role and purpose of each file and its configurations:

docker-compose.yml

Defines the multi-container SOCKS proxy lab environment with two network segments. It specifies services including hostA and hostB containers with TUN device access, along with a router/firewall container that implements iptables rules for traffic control. The configuration grants necessary capabilities (cap_add: ALL) for network operations, enables IP forwarding, and sets up routing between the net-10.8.0.0 and net-192.168.20.0 networks. This is the main configuration for launching the SOCKS proxy testing environment with docker-compose up.

Hosts/
Directory containing custom Docker container configuration for the lab hosts. Includes the Dockerfile for building the custom seed-image-ubuntu-hosts image and SSH configuration files to enable secure shell access between containers.

Hosts/Dockerfile
Build configuration for creating the custom Ubuntu host image used by all host containers in the lab. Specifies the base image, package installations, and system configurations needed for the SOCKS proxy experiments.

Hosts/sshd_config
SSH server configuration file that defines how the SSH service should run within the containers. Configures authentication methods, port settings, and security options for remote access between lab hosts.

volumes/
Host-side directory containing SOCKS client implementations for testing proxy server functionality. These Python scripts demonstrate how to connect through SOCKS proxies to access external resources.

volumes/B-Socks-Client.py
SOCKS5 client implementation configured to connect through a proxy running on localhost (0.0.0.0) port 8000. Demonstrates HTTP request routing through a local SOCKS proxy server to access external web resources.

volumes/B1-B2-Socks-Client.py
SOCKS5 client implementation specifically configured for hosts B1 and B2, connecting through a proxy at 192.168.20.99 port 8000. Shows how internal network hosts can route traffic through a designated proxy server.

volumes/socks_client.py
Generic SOCKS5 client template with placeholder values for proxy IP and server IP. Serves as a base implementation that can be customized for different proxy configurations within the lab environment.

Change shell name:

PS1="B-192.168.20.99:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="A-10.8.0.99:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="hostA-10.9.0.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="A1-10.8.0.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="A2-10.8.0.6:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="router-firewall:PES1UG23CS433:PranavHemanth:\w\n\$>"

PS1="B1-192.168.20.5:PES1UG23CS433:PranavHemanth:\w\n\$>"

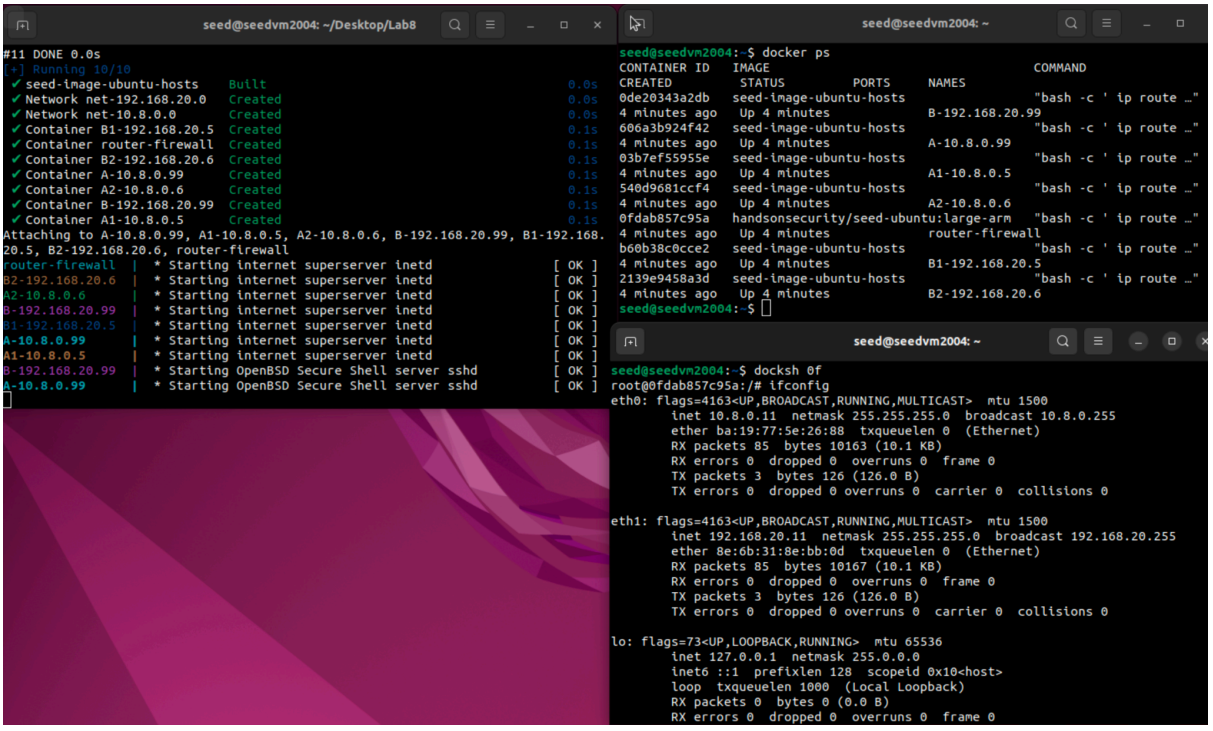
PS1="B2-192.168.20.6:PES1UG23CS433:PranavHemanth:\w\n\$>"

Task 0 : Get Familiar with the Lab Setup

Router configuration: setting up NAT . The following iptables command is included in the router configuration inside the docker-compose.yml file. This command sets up a NAT on the router for the traffic going out from its eth0 interface, except for the packets to 10.8.0.0/24. With this rule, for packets going out to the Internet, their source IP address will be replaced by the router's IP address 10.8.0.11. Packets going to 10.8.0.0/24 will not go through NAT .

iptables -t nat -A POSTROUTING ! -d 10.8.0.0/24 -j MASQUERADE -o eth0

In the above command, we assume that eth0 is the name assigned to the interface connecting the router to the 10.8.0.0/24 network. This is not guaranteed. The router has two Ethernet interfaces; when the router container is created, the name assigned to this interface might be eth1. You can find out the correct interface name using the following command. If the name is not eth0, you should make a change to the command above inside the docker-compose.yml file, and then restart the containers.



As we can see the interface connecting the router to 10.8.0.0 is eth0. Hence no changes are required

Step1: Run the below command and take a screenshot

```
# ip -br address
lo               UNKNOWN        127.0.0.1/8
eth1@if1907      UP             192.168.20.11/24
eth0@if1909      UP             10.8.0.11/24
```

1) Command: ip -br address (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
eth0@if22        UP             10.8.0.11/24
eth1@if24        UP             192.168.20.11/24
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Router configuration: Firewall rules. We have also added the following firewall rules on the router .
Please make sure that eth0 is the interface connected to the 10.8.0.0/24 network and that eth1 is the one connected to 192.168.20.0/24. If not, make changes accordingly.

```
// Ingress filtering: only allows SSH traffic
iptables -A FORWARD -i eth0 -p tcp -m conntrack \
         --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth0 -p tcp -j DROP

// Egress filtering: block www.example.com
iptables -A FORWARD -i eth1 -d 93.184.216.0/24 -j DROP
```

2) Command: iptables -A FORWARD -i eth0 -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth0 -p tcp -m conntrack --ctstate ESTABLISHED,RELATED
-j ACCEPT
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

3) Command: iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

4) Command: iptables -A FORWARD -i eth0 -p tcp -j DROP (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth0 -p tcp -j DROP
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Question: Explain what each of the firewall rules do.
Answer:

Ingress Filtering (router-firewall configuration)

Ingress filtering is used to control and secure incoming traffic entering the router from the external network. The following iptables rules demonstrate how specific packets are selectively allowed or blocked based on their state and protocol type.

1. Allow established and related connections

iptables -A FORWARD -i eth0 -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

Explanation: This rule permits traffic that is part of an already established or related connection- such as replies to SSH or HTTP requests initiated by internal hosts. It ensures that legitimate return packets can pass through the firewall.

Breakdown:

- -A FORWARD → Appends the rule to the FORWARD chain (used for packets being routed through the system).
- -i eth0 → Applies this rule to packets entering through the eth0 interface (external side).
- -p tcp → Matches only TCP packets.
- -m conntrack → Uses the connection tracking module to track existing sessions.
- --ctstate ESTABLISHED,RELATED → Matches packets that belong to an existing connection or are related (e.g., FTP data channels).
- -j ACCEPT → Allows these packets through the firewall.

This rule is crucial for maintaining the return path of valid sessions without reopening new inbound connections.

2. Allow new SSH connections

iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT

Explanation:

This rule allows incoming SSH traffic (TCP port 22) through the router, enabling secure remote access while still blocking other new inbound traffic.

Breakdown:

- -A FORWARD → Adds the rule to the FORWARD chain.
- -i eth0 → Applies to traffic coming from the external network via eth0.
- -p tcp → Matches TCP protocol packets.
- --dport 22 → Targets packets with destination port 22 (SSH).
- -j ACCEPT → Permits these packets to pass through.

This ensures administrators can remotely manage the system while maintaining controlled access.

3. Block all other inbound TCP traffic

iptables -A FORWARD -i eth0 -p tcp -j DROP

Explanation:

This final rule drops all other incoming TCP traffic that doesn't match the previous rules. It acts as a default deny policy, preventing unauthorized access attempts such as HTTP, HTTPS, or FTP connections from the external network.

Breakdown:

- -A FORWARD → Adds the rule to the FORWARD chain.
- -i eth0 → Applies to inbound traffic through eth0.
- -p tcp → Matches all TCP packets.
- -j DROP → Silently discards unmatched packets.

Together, these three rules implement a least privilege model:

- Allow only responses to legitimate internal traffic.
- Permit secure SSH access.
- Block all other unsolicited inbound connections.

We need to determine the current IP address ranges for the websites to be blocked. The IPs provided in previous lab versions are outdated and will fail

Step1.a: Run the below command and take a screenshot

5) Command: nslookup www.example.com (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.example.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
www.example.com canonical name = a1422.dscr.akamai.net.
Name:   a1422.dscr.akamai.net
Address: 103.132.16.19

router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Step1.b: Run the below command and take a screenshot

6) Command: nslookup www.linkedin.com (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.linkedin.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
www.linkedin.com        canonical name = ln-0002.ln-msedge.net.
Name:   ln-0002.ln-msedge.net
Address: 150.171.22.12

router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Step1.c: Run the below command and take a screenshot

7) Command: nslookup www.miniclip.com (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.miniclip.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
Name:   www.miniclip.com
Address: 13.227.249.5

router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Step2: Run the below command and take a screenshot

8) Command: iptables -t filter -L -n (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ES
TABLISHED
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:22
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0
DROP       all  --  0.0.0.0/0            93.184.216.0/24

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Please block two more websites and add the firewall rules to the setup files. The choice of websites are up to you. Keep in mind that most popular websites have multiple IP addresses that can change from time to time. After adding the rules, start the containers, and verify that all the ingress and egress firewall rules are working as expected.

Step3: On the router-firewall container: This IP address is for www.linkedin.com

9) Command: iptables -A FORWARD -i eth1 -d 13.107.42.0/24 -j DROP (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth1 -d 13.107.42.0/24 -j DROP
router-firewall:PES1UG23CS433:PranavHemanth:/
$>ping www.linkedin.com
PING ln-0002.ln-msedge.net (150.171.22.12) 56(84) bytes of data.
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=1 ttl=127 time=72.3 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=2 ttl=127 time=124 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=3 ttl=127 time=66.2 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=4 ttl=127 time=53.7 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=5 ttl=127 time=86.2 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=6 ttl=127 time=136 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=7 ttl=127 time=68.5 ms
^C
--- ln-0002.ln-msedge.net ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 53.652/86.693/135.689/28.960 ms
router-firewall:PES1UG23CS433:PranavHemanth:/
```

Doesnt work as linkedin uses different cdns.

It resolved to 150.171.22.12 to return the page. Lets block that

Confirming the ip series:

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.linkedin.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
www.linkedin.com        canonical name = ln-0002.ln-msedge.net.
Name:   ln-0002.ln-msedge.net
Address: 150.171.22.12

router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

Lets block it:

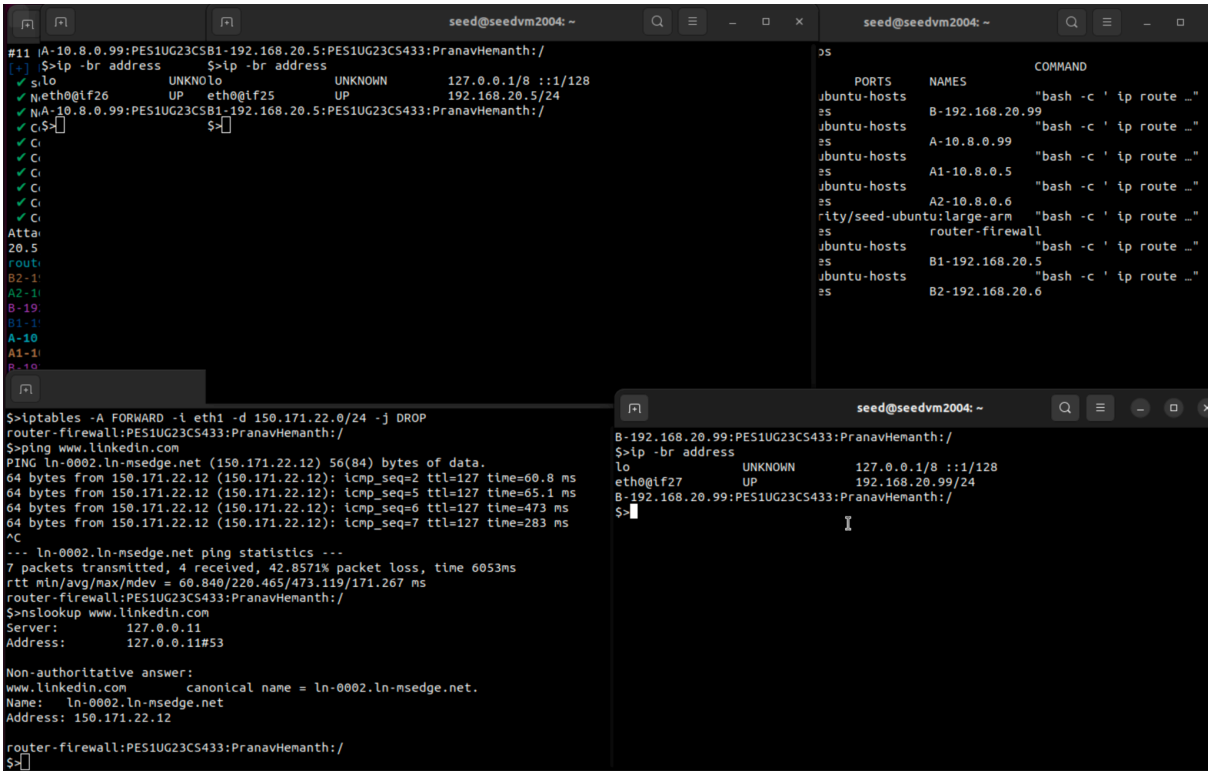
10) Command: iptables -A FORWARD -i eth1 -d 150.171.22.0/24 -j DROP (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth1 -d 150.171.22.0/24 -j DROP
router-firewall:PES1UG23CS433:PranavHemanth:/
$>ping www.linkedin.com
PING ln-0002.ln-msedge.net (150.171.22.12) 56(84) bytes of data.
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=1 ttl=127 time=74.1 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=2 ttl=127 time=99.7 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=3 ttl=127 time=71.8 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=4 ttl=127 time=43.6 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=5 ttl=127 time=40.1 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=6 ttl=127 time=95.7 ms
^C
--- ln-0002.ln-msedge.net ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 40.082/70.818/99.672/22.915 ms
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

It still doesnt appear to work

We need to ping using a device on the subnet. pinging from the router itself doesnt work as it will use the OUTPUT chain for that and not FORWARD chain.

However as instructed in the document A, B, B1 none of the containers also work (ping is still successful). On further investigation we can see that they are all on eth0 and not eth1.



So we can modify the command to block on eth0

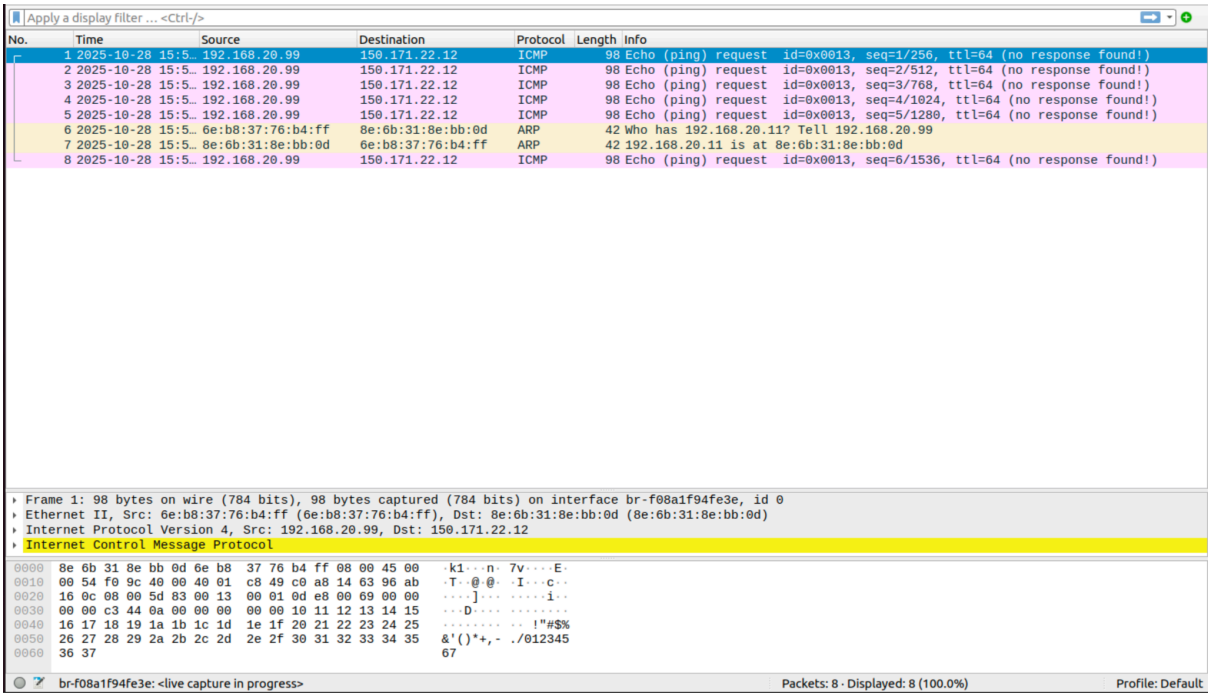
11) Command: iptables -A FORWARD -i eth0 -d 150.171.22.0/24 -j DROP (On router-firewall)

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.linkedin.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
www.linkedin.com        canonical name = ln-0002.ln-msedge.net.
Name:   ln-0002.ln-msedge.net
Address: 150.171.22.12

router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth0 -d 150.171.22.0/24 -j DROP
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

On pinging from container B:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ping www.linkedin.com
PING ln-0002.ln-msedge.net (150.171.22.12) 56(84) bytes of data.
^C
--- ln-0002.ln-msedge.net ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5098ms

B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Explanation:
This rule blocks any outgoing (egress) traffic headed to the IP address range 150.171.22.0/24, which belongs to www.linkedin.com. It prevents internal users or devices from accessing any host within that subnet.

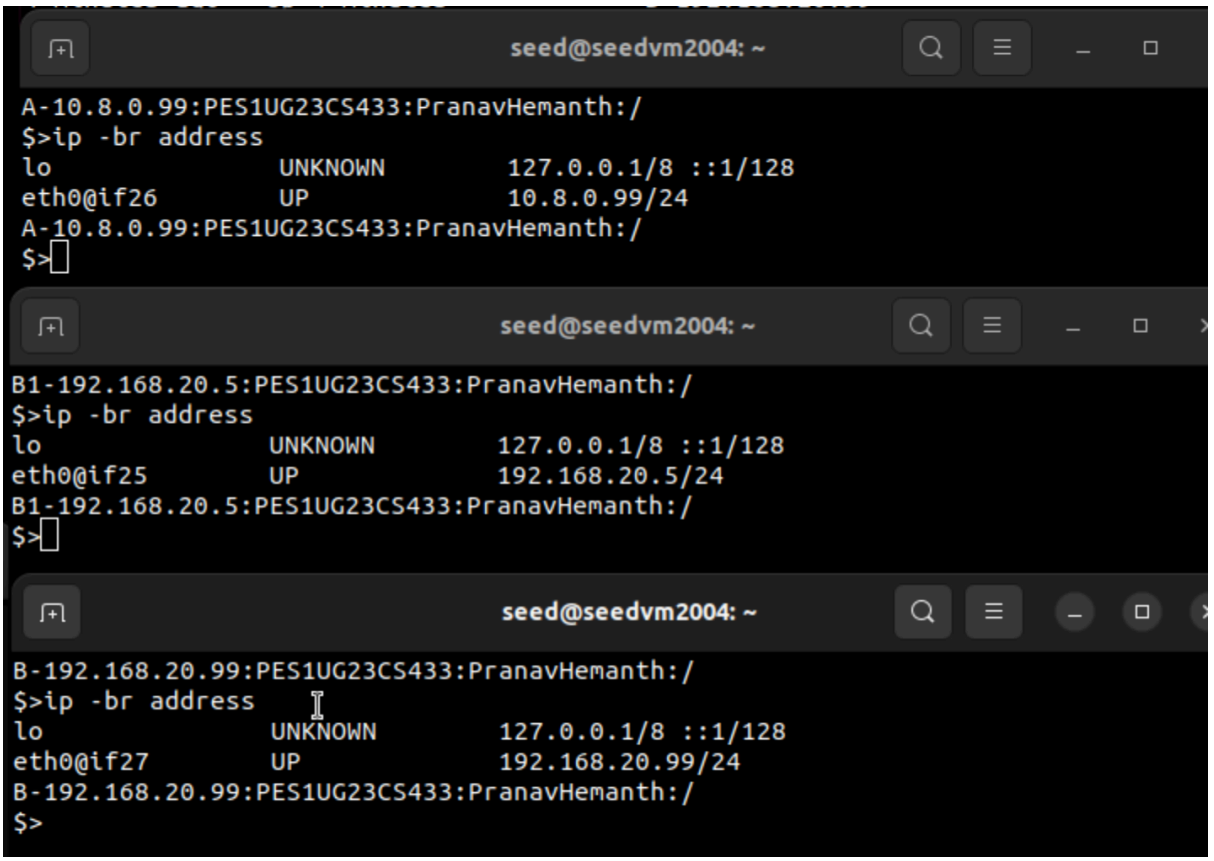
Flags explained:

- -A FORWARD → Adds (appends) the rule to the FORWARD chain.
- -i eth0 → Applies to packets leaving through the eth0 interface.
- -d 150.171.22.0/24 → Matches packets destined for this IP range.
- -j DROP → Drops these packets, blocking the connection.

Summary:
This is an egress filter that stops outbound access to a specific external network, improving security and preventing unauthorized external communication.

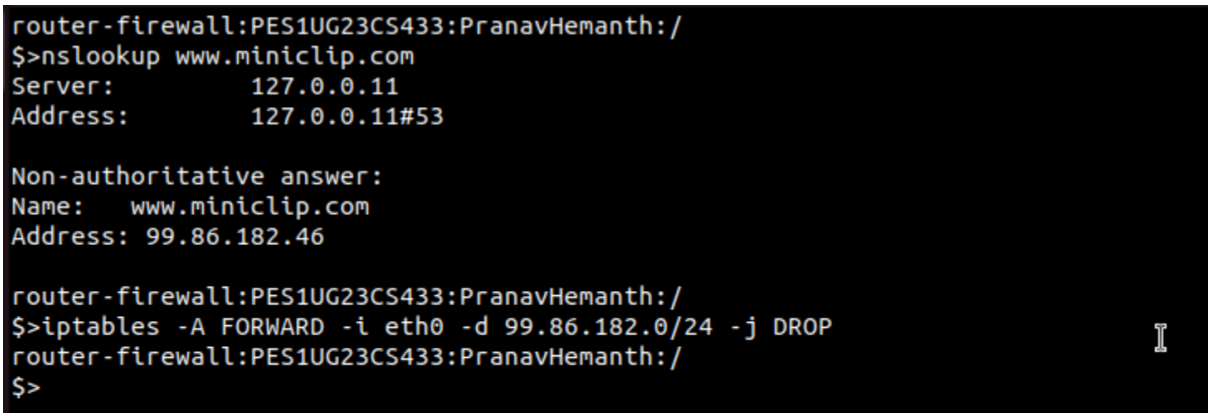
Step4: On the router-firewall container: This IP address is for www.miniclip.com

12) Command: iptables -A FORWARD -i eth0 -d 13.249.221.0/24 -j DROP (On router-firewall)



(We do eth0 instead of eth1 as per the config of containers observed on the vm)

router-firewall:

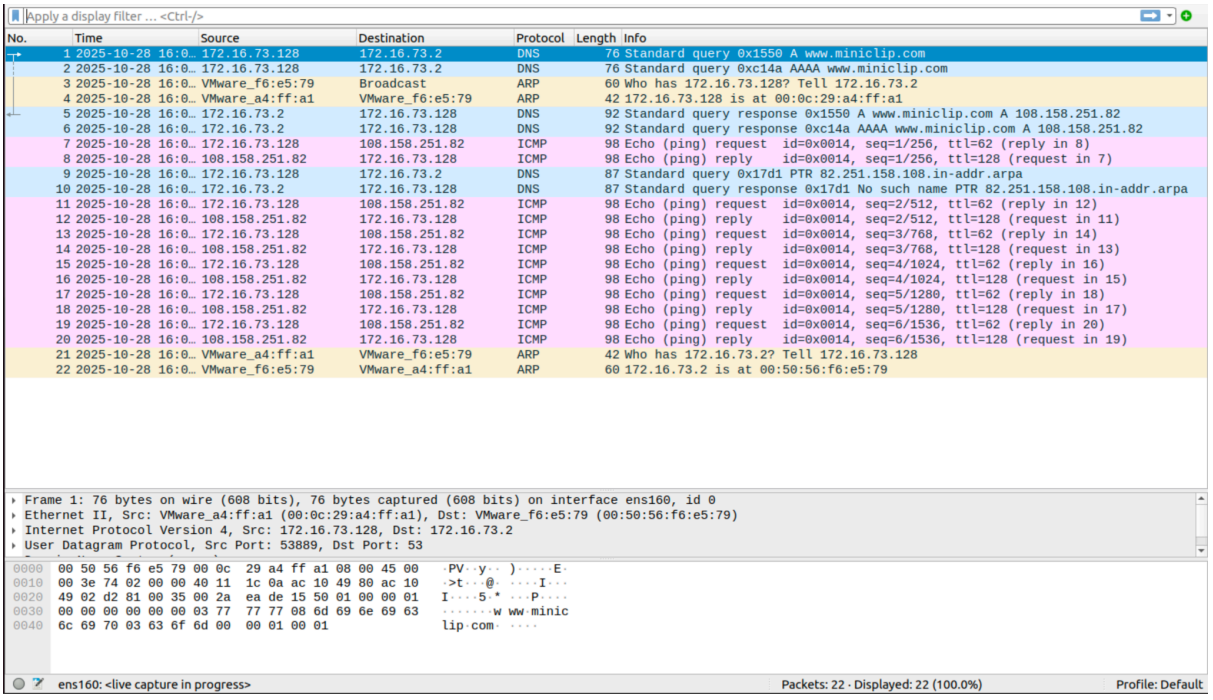


B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ping www.miniclip.com
PING www.miniclip.com (108.158.251.82) 56(84) bytes of data.
64 bytes from 108.158.251.82 (108.158.251.82): icmp_seq=1 ttl=126 time=36.5 ms
64 bytes from 108.158.251.82 (108.158.251.82): icmp_seq=2 ttl=126 time=25.1 ms
64 bytes from 108.158.251.82 (108.158.251.82): icmp_seq=3 ttl=126 time=105 ms
64 bytes from 108.158.251.82 (108.158.251.82): icmp_seq=4 ttl=126 time=33.2 ms
64 bytes from 108.158.251.82 (108.158.251.82): icmp_seq=5 ttl=126 time=27.0 ms
64 bytes from 108.158.251.82 (108.158.251.82): icmp_seq=6 ttl=126 time=57.0 ms
^C
--- www.miniclip.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 25.149/47.290/104.875/27.760 ms
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
eth0@if27        UP             192.168.20.99/24
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

We can see that miniclip instead resolved from a different ip. This is why its hard to block large sites

Wireshark:



Lets try again:

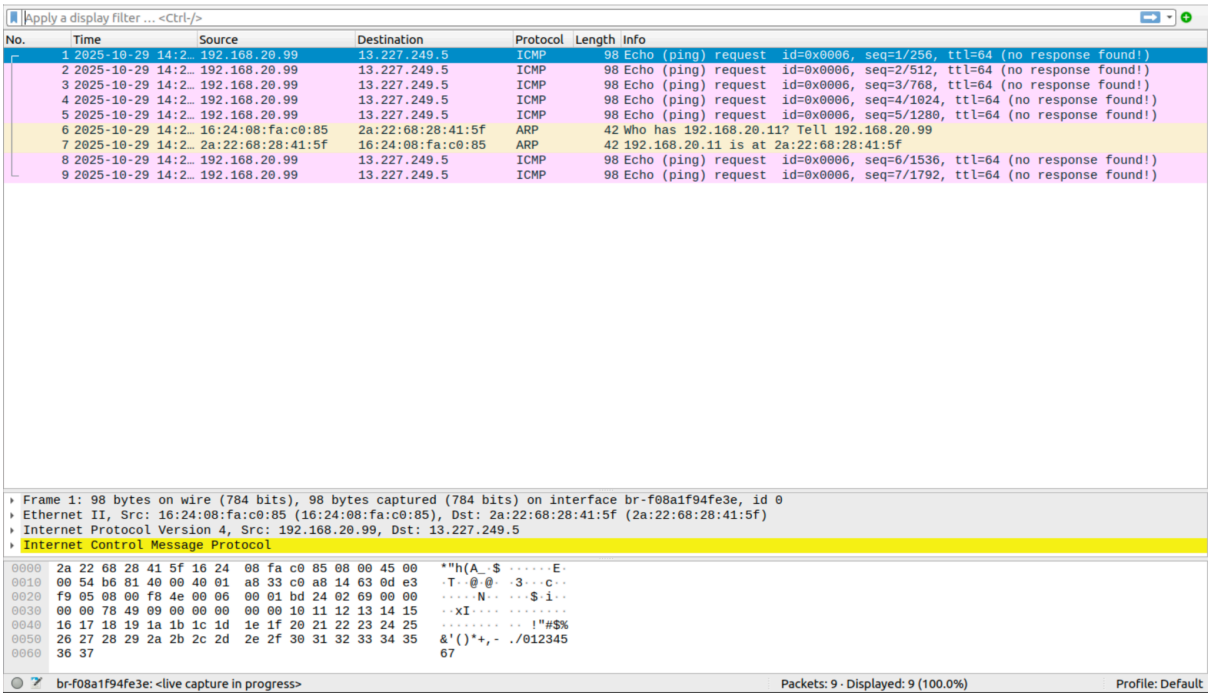
router-firewall:

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.miniclip.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
Name:   www.miniclip.com
Address: 13.227.249.63

router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth0 -d 13.227.249.0/24 -j DROP
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:22
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0
DROP       all  --  0.0.0.0/0            93.184.216.0/24
DROP       all  --  0.0.0.0/0            150.171.22.0/24
DROP       all  --  0.0.0.0/0            104.18.41.0/24
DROP       all  --  0.0.0.0/0            150.171.22.0/24
DROP       all  --  0.0.0.0/0            13.227.249.0/24

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ping www.miniclip.com
PING www.miniclip.com (13.227.249.5) 56(84) bytes of data.
^C
--- www.miniclip.com ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6131ms

B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Explanation:
This rule blocks any outgoing (egress) traffic destined for the IP range 13.227.249.0/24, which corresponds to www.miniclip.com. It prevents internal hosts from accessing any servers within that subnet.

Flags explained:

- -A FORWARD → Adds (appends) the rule to the FORWARD chain.
- -i eth0 → Applies to packets leaving through the eth0 interface.
- -d 13.227.249.0/24 → Matches packets whose destination lies within this IP range.
- -j DROP → Drops these packets, effectively blocking the connection.

Summary:
This rule serves as an egress filter that restricts outbound access to a specific external network, helping enforce browsing policies and enhance network security.

Step4: On the router-firewall container: This IP address is for www.example.com

13) Command: iptables -A FORWARD -i eth0 -d 13.249.221.0/24 -j DROP (On router-firewall)

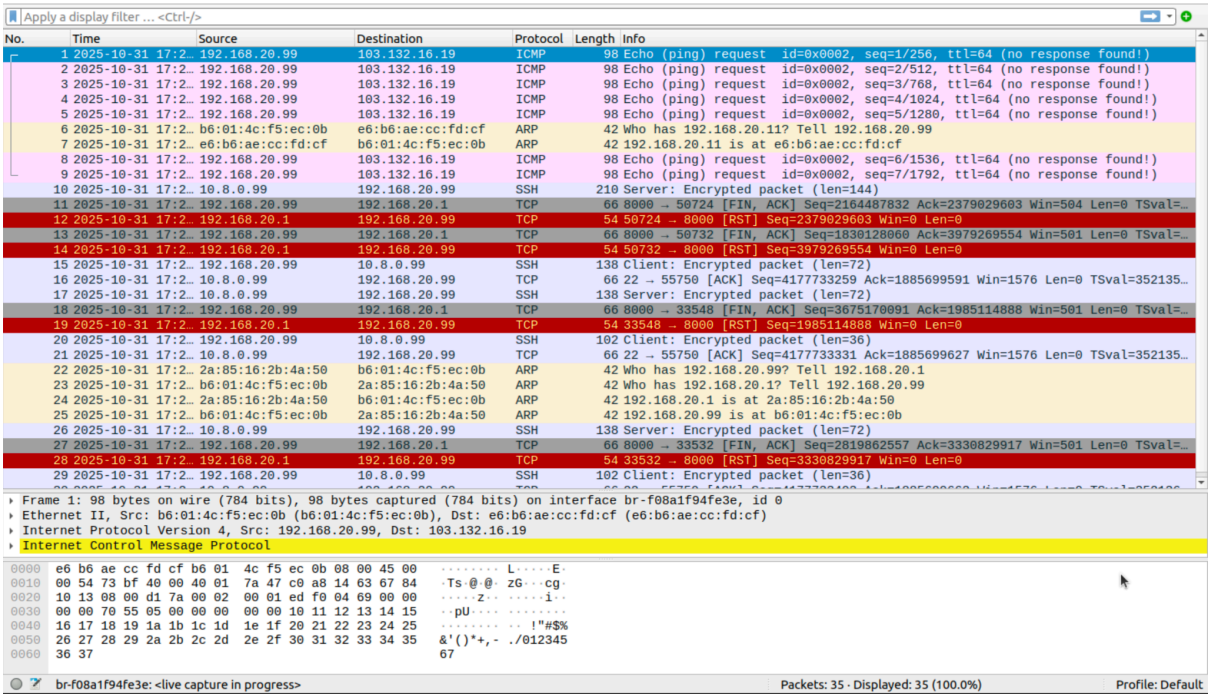
router-firewall:

```
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslookup www.example.com
Server:         127.0.0.11
Address:        127.0.0.11#53

Non-authoritative answer:
www.example.com canonical name = a1422.dscr.akamai.net.
Name:   a1422.dscr.akamai.net
Address: 103.132.16.16

router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -A FORWARD -i eth0 -d 103.132.16.0/24 -j DROP
router-firewall:PES1UG23CS433:PranavHemanth:/
$>iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:22
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0
DROP       all  --  0.0.0.0/0            93.184.216.0/24
DROP       all  --  0.0.0.0/0            103.132.16.0/24

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
router-firewall:PES1UG23CS433:PranavHemanth:/
$>
```

B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ping www.example.com
PING a1422.dscr.akamai.net (103.132.16.19) 56(84) bytes of data.
^C
--- a1422.dscr.akamai.net ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6146ms

B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Explanation:
This rule blocks any outgoing (egress) traffic destined for the IP range 103.132.16.0/24, which belongs to www.example.com. It prevents internal systems from reaching any servers within that subnet.

Flags explained:

- -A FORWARD → Adds (appends) the rule to the FORWARD chain.
- -i eth0 → Applies to packets leaving through the eth0 interface.
- -d 103.132.16.0/24 → Matches packets whose destination is within this IP range.
- -j DROP → Drops the packets, blocking the connection.

Summary:
This rule acts as an egress filter to stop outbound connections to a specific external subnet, helping control external communications and improve overall network security.

Explanation:

Because I ran the experiment on my home network rather than the PES network, DNS resolution returned different public addresses for the blocked sites. On my network, www.linkedin.com resolved to 150.171.22.12 and www.miniclip.comresolved to 13.227.249.5. I therefore repeated the experiment using these IP addresses to ensure the firewall rules were tested against the addresses actually reachable from my environment.

Observation:

First, the router-firewall was configured with iptables rules to block network traffic to specific destinations. These rules ensured that any traffic directed toward www.linkedin.com and www.miniclip.com was dropped. To validate this configuration, we attempted to ping both websites from the internal hosts B (192.168.20.99), B1 (192.168.20.5), and B2 (192.168.20.6). The domain names resolved to the IP addresses 150.171.22.12 and 13.227.249.5, which fall within the blocked address range. As expected, the ping requests failed, packets were transmitted but none were received, resulting in 100% packet loss. This confirms that the iptables rules on the router-firewall are effectively inspecting and filtering forwarded packets, successfully dropping those destined for the blocked networks.

Task 1: Static Port Forwarding

The firewall in the lab setup prevents outside machines from connecting to any TCP server on the internal network, other than the SSH server . In this task, we would like to use static port forwarding to evade this restriction. More specifically, we will use ssh to create a static port forwarding tunnel between host A (on the external network) and host B (on the internal network), so whatever data received on A's port X will be sent to B, from where the data is forwarded to the target T's port Y . In the following command, we use ssh to create such a tunnel.

```
$ ssh -4NT -L <A's IP>:<A's port X>:<T's IP>:<T's port Y>  <user id>@<B's IP>

// -4: use IPv4 only, or we will see some error message.
// -N: do not execute a remote command.
// -T: disable pseudo-terminal allocation (save resources).
```

Regarding A's IP , typically we use 0.0.0.0, indicating that our port forwarding will listen to the connection from all the interfaces on A. If we want to limit the connections to a particular interface, we should use that interface's IP address. For example, if we want to limit the connection to the loopback interface, so only the program on the local host can use this port forwarding, we can use 127.0.0.1:<port> or simply omit the IP address (the default IP address is 127.0.0.1).

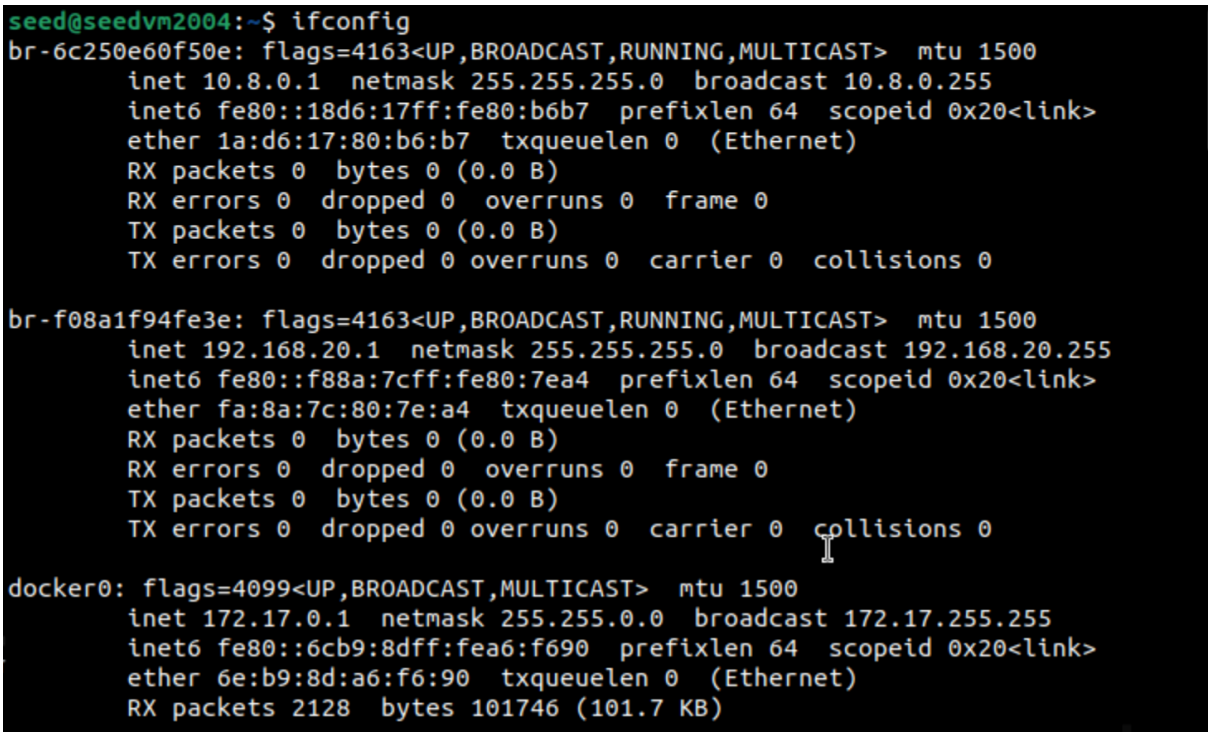
Please use static port forwarding to create a tunnel between the external network and the internal network, so we can telnet into the server on B. Please demonstrate that you can do such telnet from hosts A, A1 and A2.

Questions:

1. **How many TCP connections are involved in this entire process? You should run wireshark or tcpdump to capture the network traffic, and then point out all the involved TCP connections from the captured traffic.**
   Answer: There are two primary TCP connections involved in this process. The first is the SSH connection, visible in Wireshark as the SYN-ACK handshake, which establishes the encrypted tunnel. The second is the Telnet connection from 192.168.20.99 to 10.8.0.99 on port 8000, seen as a separate three-way handshake in Wireshark. Although the Telnet traffic is forwarded through the SSH tunnel, it remains a distinct TCP session encapsulated within the first one.

2. **Why can this tunnel successfully help users evade the firewall rule specified in the lab setup?**
   Answer: This tunnel bypasses port-based firewall restrictions because all traffic is encapsulated within the encrypted SSH session. Normally, a firewall configured to block Telnet traffic (port 23) would inspect packets targeting that port. However, in this setup, the Telnet data is sent to port 8000 on host A and then transmitted through the existing SSH tunnel over port 22. Since the firewall only detects encrypted SSH traffic on an allowed port, it cannot inspect or block the Telnet packets hidden inside, effectively allowing them to pass through undetected.

Keep wireshark open and select the interface with the IP address of 192.168.20.1. Wireshark is to be opened on the host VM.



So we can keep captures from br-f08a1f94fe3e open in wireshark

Step1: Run the below command on container A:

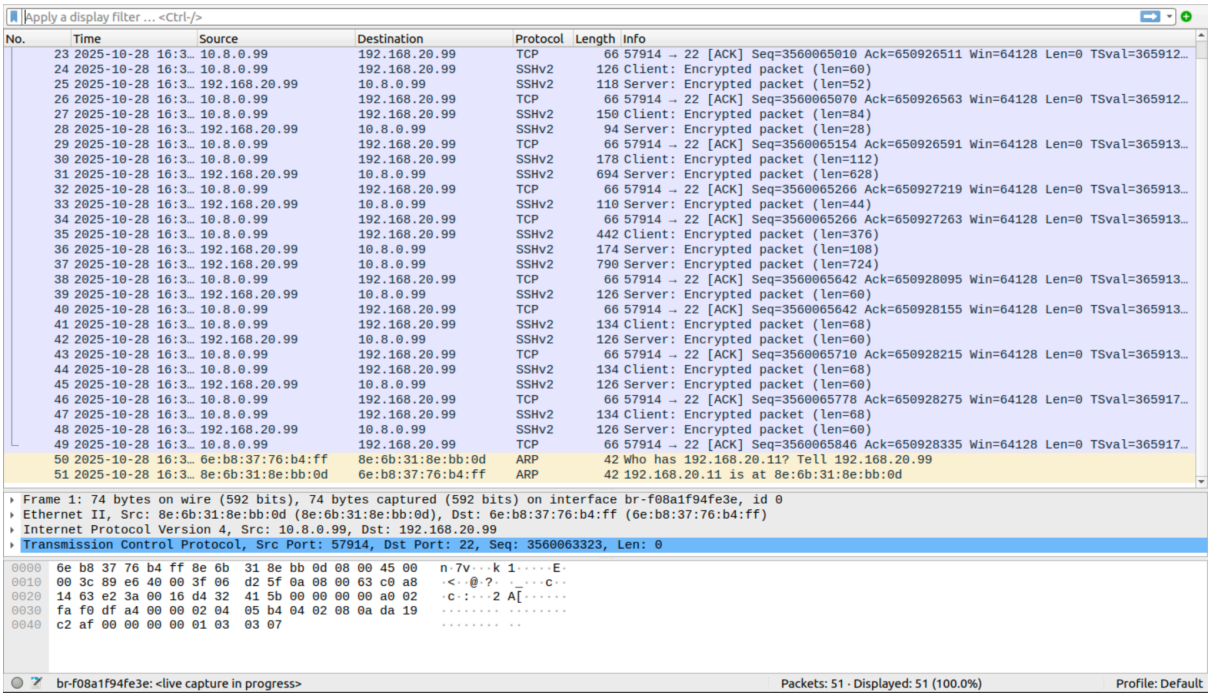
14)Command: ssh -L 0.0.0.0:8000:192.168.20.99:23 root@192.168.20.99

15)password: dees

A-10.8.0.99:

```
A-10.8.0.99:PES1UG23CS433:PranavHemanth:/
$>ssh -L 0.0.0.0:8000:192.168.20.99:23 root@192.168.20.99
The authenticity of host '192.168.20.99 (192.168.20.99)' can't be established.
ECDSA key fingerprint is SHA256:fkGtseinXKS9urPzu8YEIMj8qzO5MJzoi5ekV2x96MQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.20.99' (ECDSA) to the list of known hosts.
root@192.168.20.99's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@0de20343a2db:~#
```

Wireshark:



Step2: Run the below command on container A:

16)Command: telnet 10.8.0.99 8000

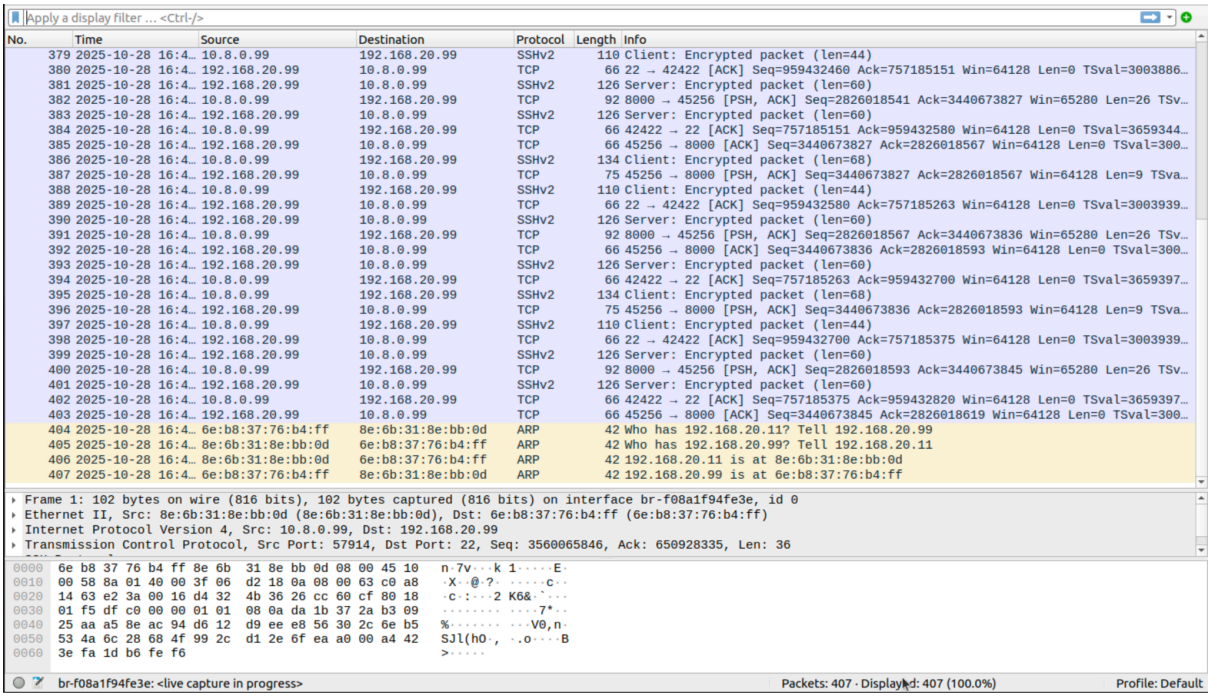
17)username: seed; password: dees

18) Command: ifconfig

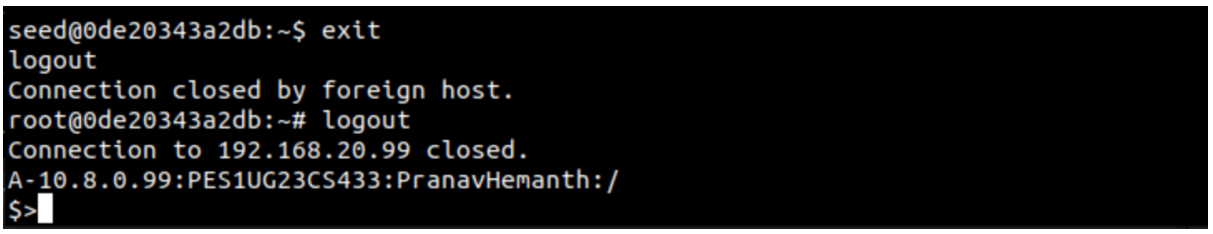
A-10.8.0.99:

```
root@0de20343a2db:~# telnet 10.8.0.99 8000
Trying 10.8.0.99...
Connected to 10.8.0.99.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
0de20343a2db login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@0de20343a2db:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.20.99  netmask 255.255.255.0  broadcast 192.168.20.255
        ether 6e:b8:37:76:b4:ff  txqueuelen 0  (Ethernet)
        RX packets 324  bytes 40765 (40.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 202  bytes 24511 (24.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 117  bytes 7896 (7.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 117  bytes 7896 (7.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

seed@0de20343a2db:~$
```

Wireshark:



Step3: To exit the connection type ‘exit’ and then ctrl+D



Explanation:

In this setup, static port forwarding was configured from host A (10.8.0.99) by establishing an SSH connection to host B (192.168.20.99). Using the SSH command, we logged into host B and created a listener on host A (port 8000, bound to all interfaces). This listener was configured to forward any incoming connections through the established SSH tunnel to port 23 on host B.

The initial Wireshark capture confirmed the establishment of the SSHv2 tunnel between 10.8.0.99 and 192.168.20.99. To verify functionality, we initiated a Telnet connection to 10.8.0.99:8000 from the shell on host B. Wireshark recorded a new TCP three-way handshake from 192.168.20.99 to 10.8.0.99:8000, indicating that the listener on host A received the request and successfully forwarded the Telnet data back through the encrypted SSH tunnel to host B’s port 23.

This “loopback” test was successful, the Telnet login prompt for host B appeared, confirming that the static port forwarding configuration was functioning correctly. Finally, both the Telnet and SSH sessions were gracefully terminated, as reflected by the FIN ACK sequence in the closing Wireshark capture.

SAME TASK WITH A1:

Step1: Run the below command on container A:

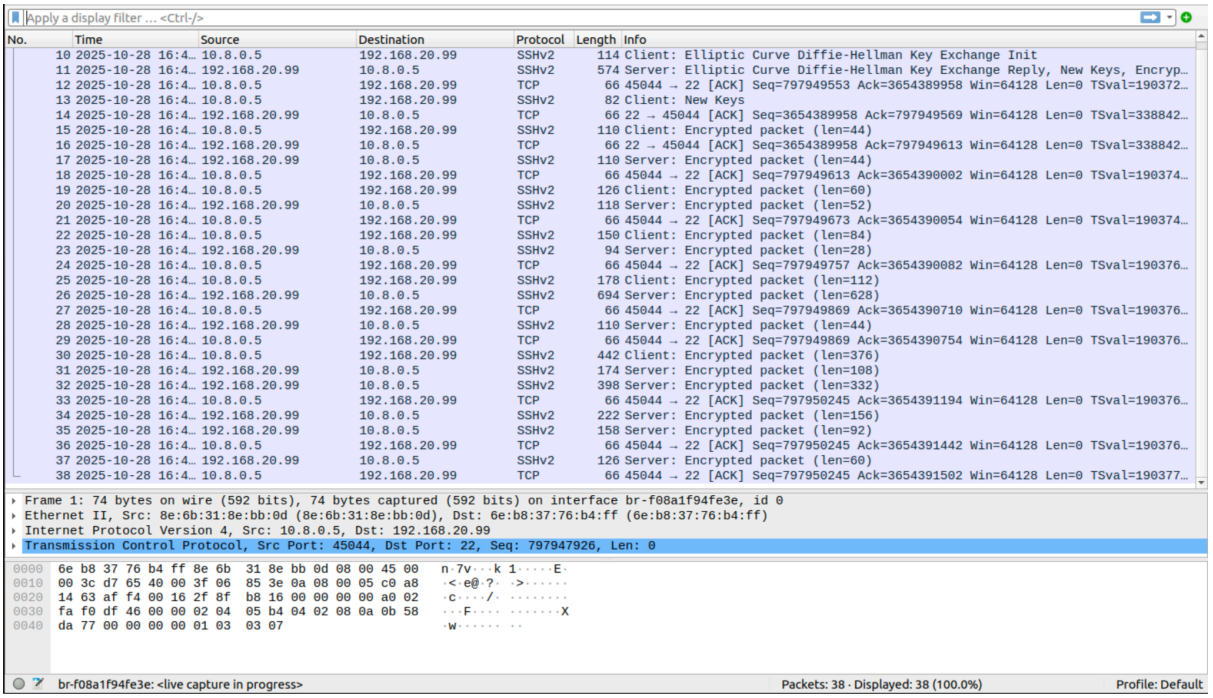
19)Command: ssh -L 0.0.0.0:8000:192.168.20.99:23 root@192.168.20.99

20)password: dees

A1-10.8.0.5:

```
A1-10.8.0.5:PES1UG23CS433:PranavHemanth:/
$>ssh -L 0.0.0.0:8000:192.168.20.99:23 root@192.168.20.99
The authenticity of host '192.168.20.99 (192.168.20.99)' can't be established.
ECDSA key fingerprint is SHA256:fkGtseinXKS9urPzu8YEIMj8qzO5MJzoi5ekV2x96MQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.20.99' (ECDSA) to the list of known hosts.
root@192.168.20.99's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Oct 28 16:41:04 2025 from 10.8.0.99
root@0de20343a2db:~#
```

Wireshark:



Step2: Run the below command on container A:

21)Command: telnet 10.8.0.99 8000

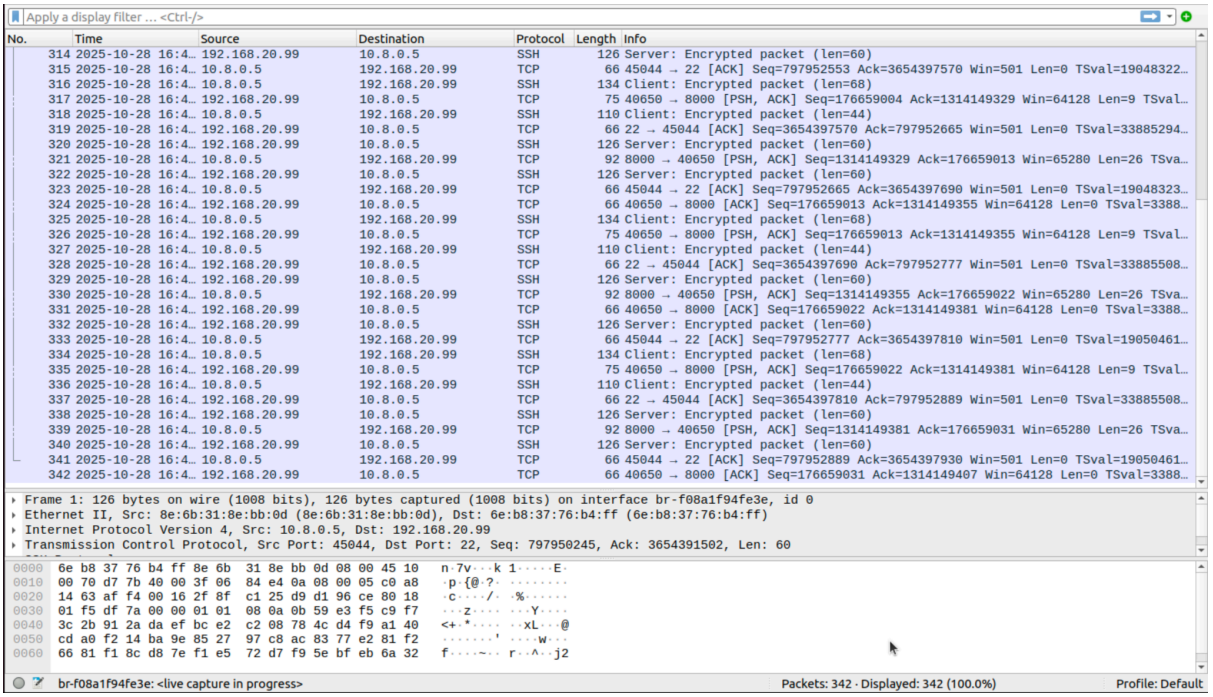
22)username: seed; password: dees

23) Command: ifconfig

A1-10.8.0.5:

```
root@0de20343a2db:~# telnet 10.8.0.5 8000
Trying 10.8.0.5...
Connected to 10.8.0.5.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
0de20343a2db login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-160-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Oct 28 16:41:11 UTC 2025 from 0de20343a2db on pts/3
seed@0de20343a2db:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.20.99  netmask 255.255.255.0  broadcast 192.168.20.255
        ether 6e:b8:37:76:b4:ff  txqueuelen 0  (Ethernet)
        RX packets 578  bytes 64743 (64.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 458  bytes 52870 (52.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 290  bytes 18621 (18.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 290  bytes 18621 (18.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

seed@0de20343a2db:~$
```

Wireshark:



Step3: To exit the connection type ‘exit’ and then ctrl+D

```
seed@0de20343a2db:~$ exit
logout
Connection closed by foreign host.
root@0de20343a2db:~# logout
Connection to 192.168.20.99 closed.
A1-10.8.0.5:PES1UG23CS433:PranavHemanth:/
$>
```

Explanation:

In this setup, static port forwarding was configured from host A1 (10.8.0.5) by establishing an SSH connection to host B (192.168.20.99). Using the SSH command, we logged into host B and created a listener on A1 (port 8000, bound to all interfaces). This listener was configured to forward any incoming connections through the encrypted SSH tunnel to port 23 on host B.

The Wireshark capture verified the successful establishment of the SSHv2 tunnel between 10.8.0.5 and 192.168.20.99. To test the configuration, we initiated a Telnet connection to 10.8.0.99:8000 from A1. Wireshark showed a new TCP three-way handshake originating from 10.8.0.5 to 10.8.0.99:8000, confirming that the listener on A1 received the connection request and correctly forwarded the Telnet traffic through the SSH tunnel to B’s Telnet service on port 23.

The Telnet login prompt for host B appeared, confirming that the static port forwarding setup was working as intended. Finally, both the Telnet and SSH sessions were terminated, as indicated by the FIN ACK packets captured in Wireshark, marking the end of both TCP connections.

Task 2: Dynamic Port Forwarding

In the static port forwarding, each port-forwarding tunnel forwards the data to a particular destination. If we want to forward data to multiple destinations, we need to set up multiple tunnels. For example, using port forwarding, we can successfully visit the blocked example.com website, but what if the firewall blocks many other sites, how do we avoid tediously establishing an SSH tunnel for each site? We can use dynamic port forwarding to solve this problem.

Task 2.1: Setting Up Dynamic Port Forwarding

We can use ssh to create a dynamic port-forwarding tunnel between B and A. We run the following command on host B. In dynamic port forwarding, B is often called proxy.

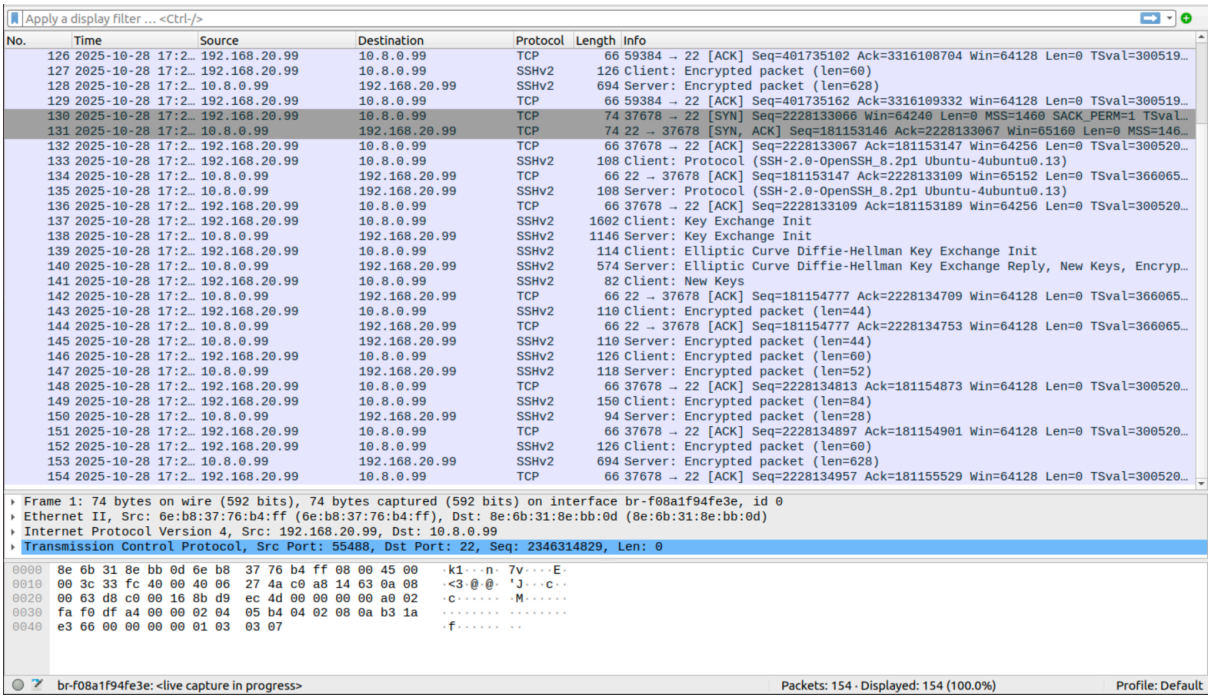
Step1: Run in container B to set up the ssh shell with dynamic port forwarding enabled:

24)Command: ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N

B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root@10.8.0.99's password:
bind [0.0.0.0]:8000: Address already in use
channel_setup_fwd_listener_tcpip: cannot listen to port: 8000
Could not request local forwarding.
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ssh -4 -D 0.0.0.0:8080 root@10.8.0.99 -f -N
root@10.8.0.99's password:
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Question: What is the significance of using 0.0.0.0 in the command?
Answer: The use of 0.0.0.0 in the command specifies that the proxy should listen on all available network interfaces of the local machine. This means that any device within the same local network can connect to host B's IP address on port 8000 and utilize it as a proxy. In effect, this configuration turns host B into a proxy gateway, allowing connected devices to route their traffic through the established SSH tunnel to 10.8.0.99**.**

We specify a proxy option, so curl will send its HTTP request to the proxy B, which listens on port X. The proxy forwards the data received on this port to the other end of the tunnel (host A), from where the data will be further forwarded to the target website. The type of proxy is called SOCKS version 5, so that is why we specify socks5h.

Please demonstrate that you can visit all the blocked websites using curl from hosts B, B1, and B2 on the internal network.

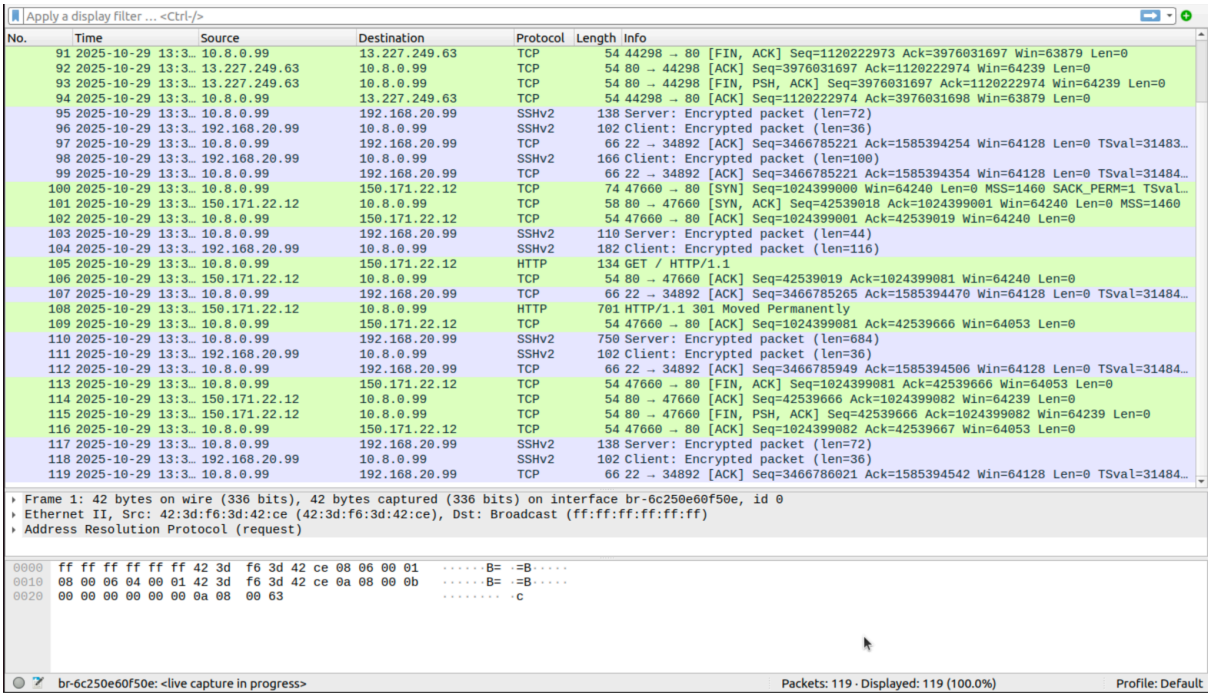
Step2: Now run the below on container B

25)Command: curl -x socks5h://0.0.0.0:8000 http://www.example.com

B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root@10.8.0.99's password:
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://0.0.0.0:8000 http://www.example.com
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="vi
ewport" content="width=device-width, initial-scale=1"><style>body{background:#ee
e;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5e
m}div{opacity:0.8}a:link,a:visited{color:#348}</style><body><div><h1>Example Dom
ain</h1><p>This domain is for use in documentation examples without needing perm
ission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Le
arn more</a></div></body></html>
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://0.0.0.0:8000 http://www.linkedin.com
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
```

Wireshark:



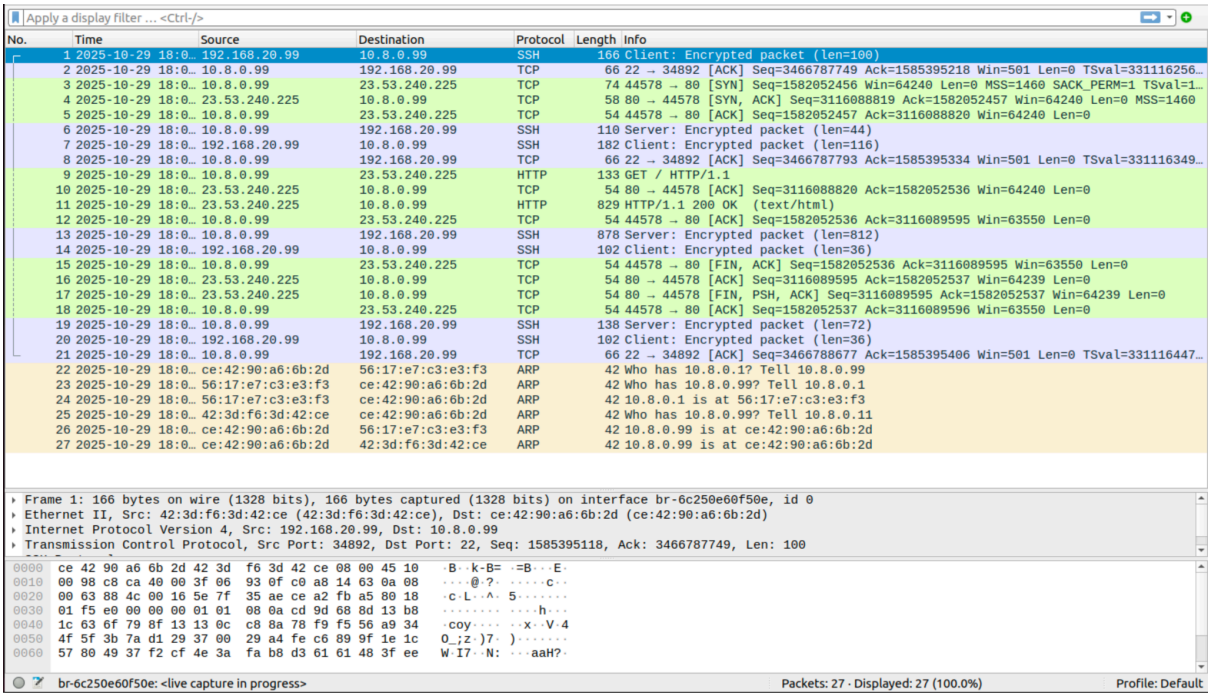
Step3: Now run the below on container B1 and B2

26)Command: curl -x socks5h://192.168.20.99:8000 http://www.example.com

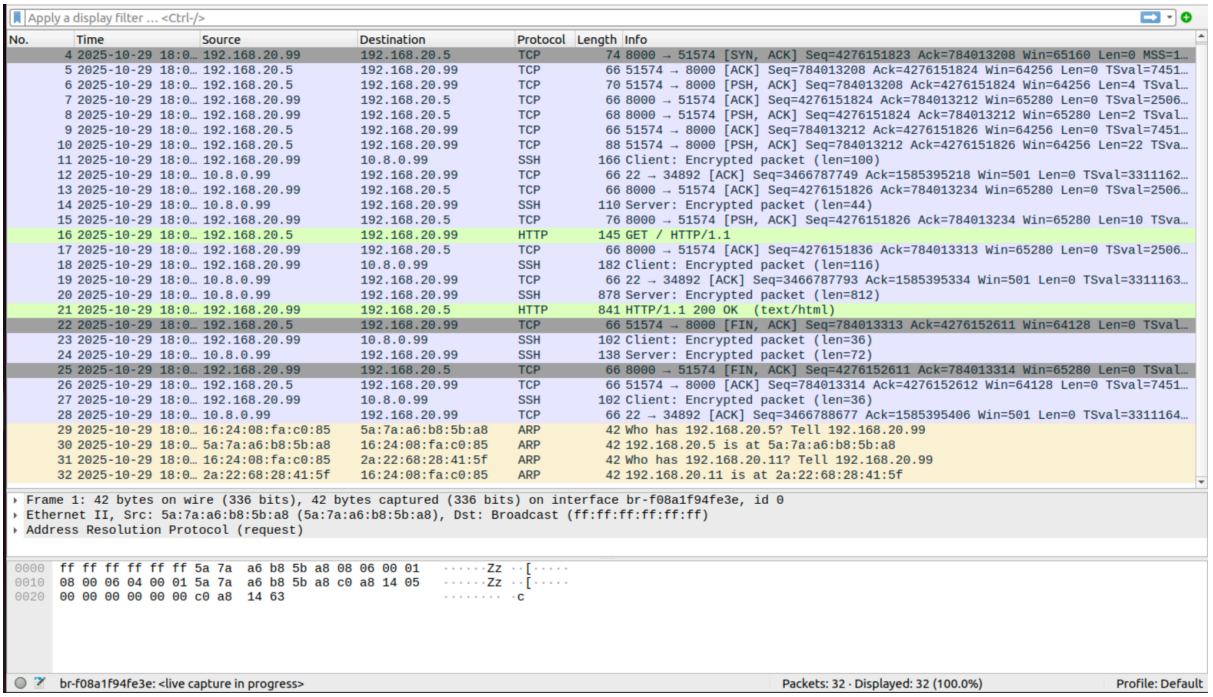
B1-192.168.20.5:

```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://192.168.20.99:8000 http://www.example.com
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="vi
ewport" content="width=device-width, initial-scale=1"><style>body{background:#ee
e;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5e
m}div{opacity:0.8}a:link,a:visited{color:#348}</style><body><div><h1>Example Dom
ain</h1><p>This domain is for use in documentation examples without needing perm
ission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Le
arn more</a></div></body></html>
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark on 10.8.0.1 network:



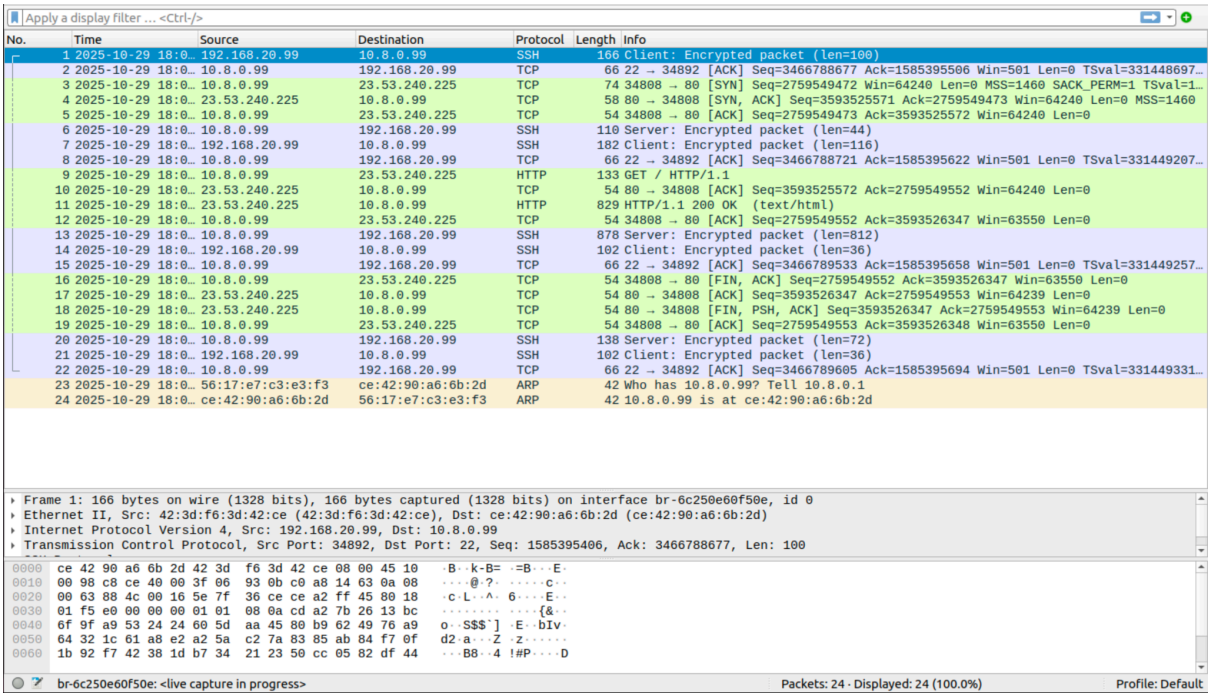
Wireshark on 192.168.20.1 network:



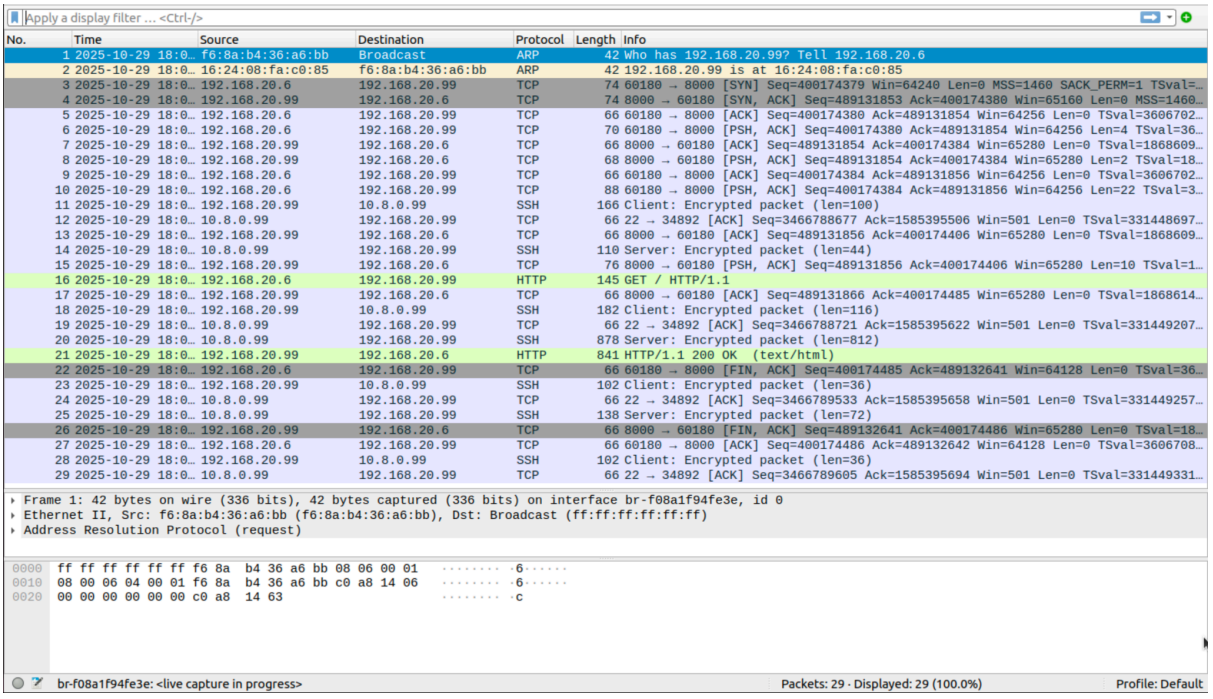
B2-192.168.20.6:

```
B2-192.168.20.6:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://192.168.20.99:8000 http://www.example.com
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="vi
ewport" content="width=device-width, initial-scale=1"><style>body{background:#ee
e;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5e
m}div{opacity:0.8}a:link,a:visited{color:#348}</style><body><div><h1>Example Dom
ain</h1><p>This domain is for use in documentation examples without needing perm
ission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Le
arn more</a></div></body></html>
B2-192.168.20.6:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:



Now lets try with www.linkedin.com

Step4: Now run the below on container B

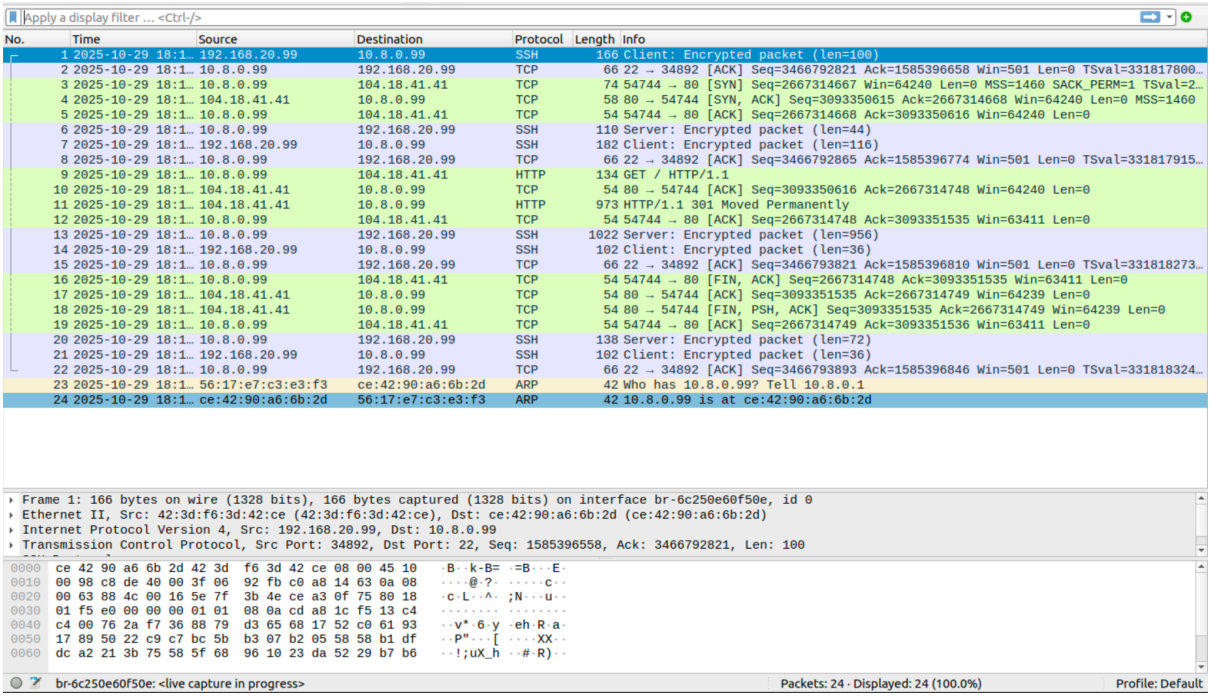
27)Command: curl -x socks5h://0.0.0.0:8000 http://www.linkedin.com

B-192.168.20.99:

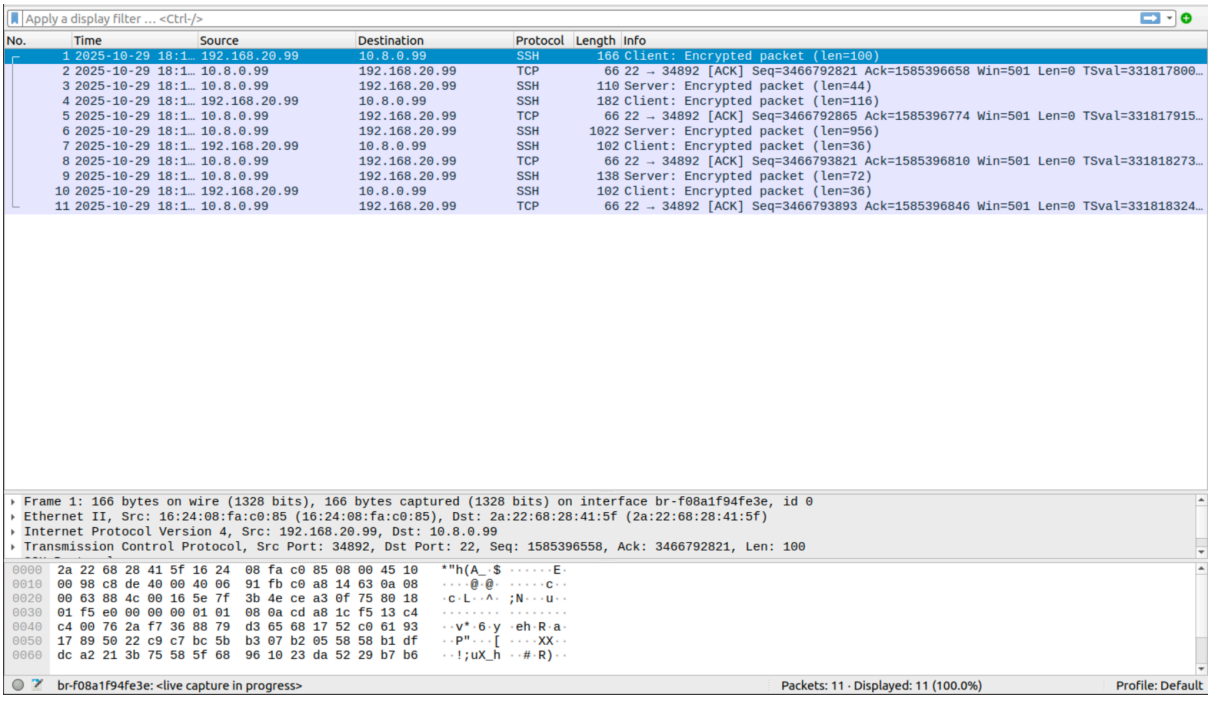
Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://0.0.0.0:8000 http://www.linkedin.com
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:



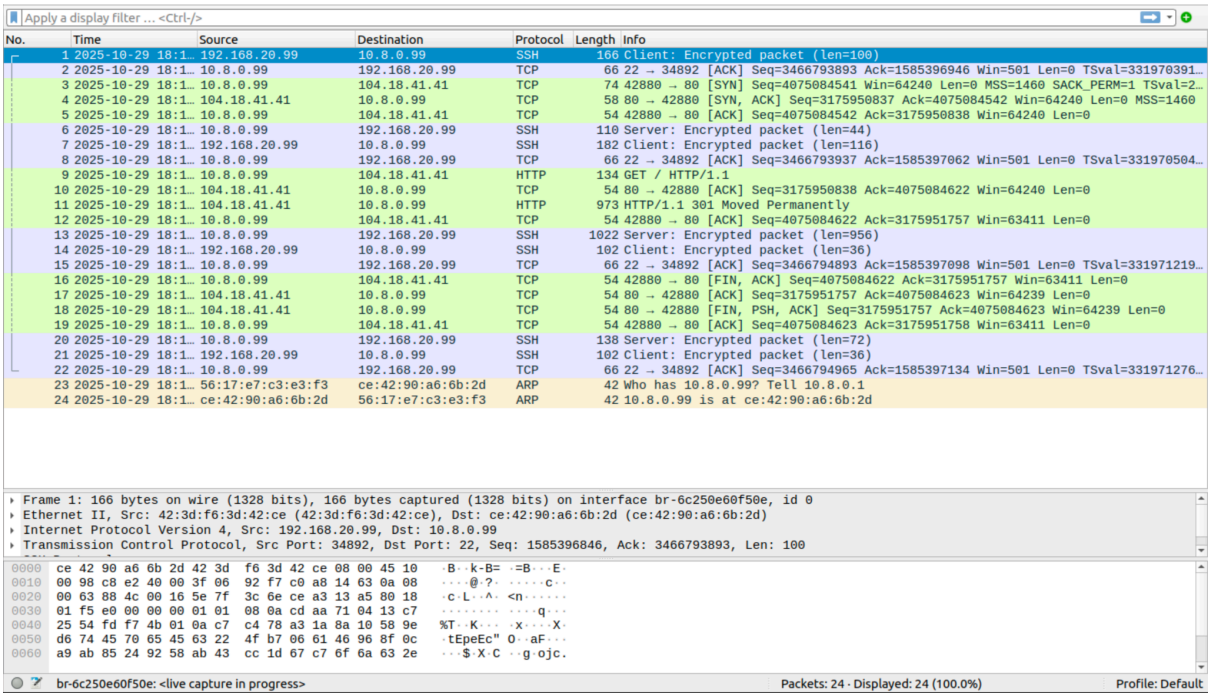
Step3: Now run the below on container B1 and B2

28)Command: curl -x socks5h://192.168.20.99:8000 http://www.linkedin.com

B1-192.168.20.5:

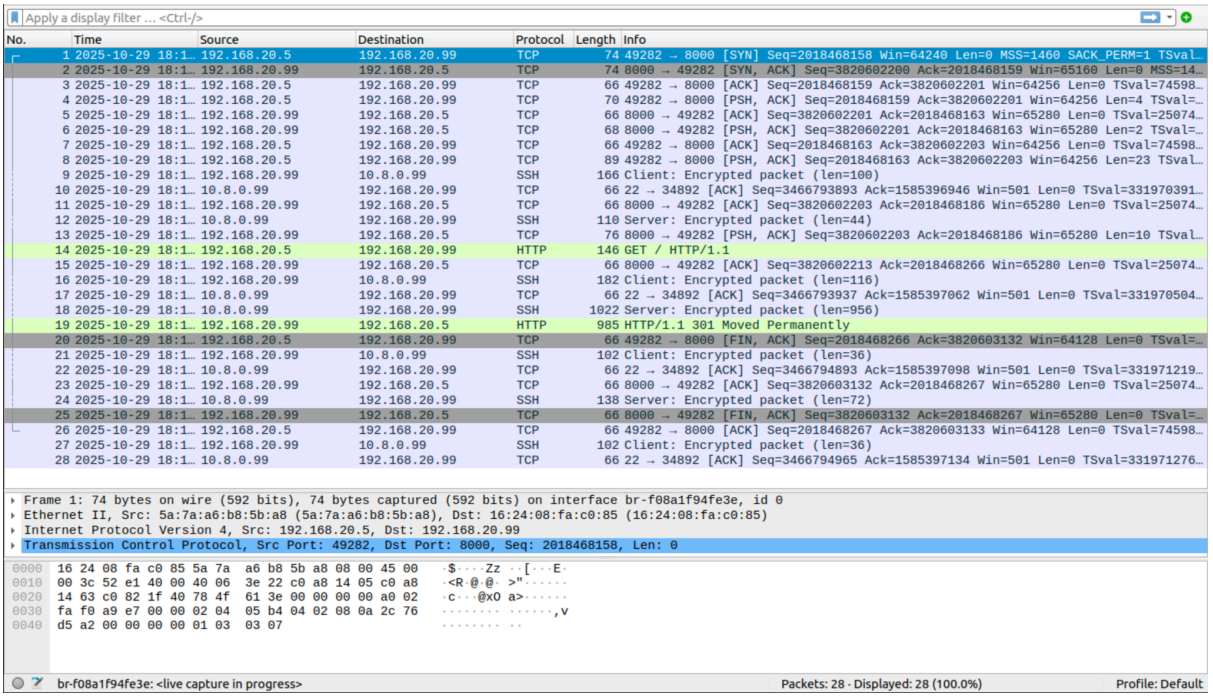
```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://192.168.20.99:8000 http://www.linkedin.com
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:

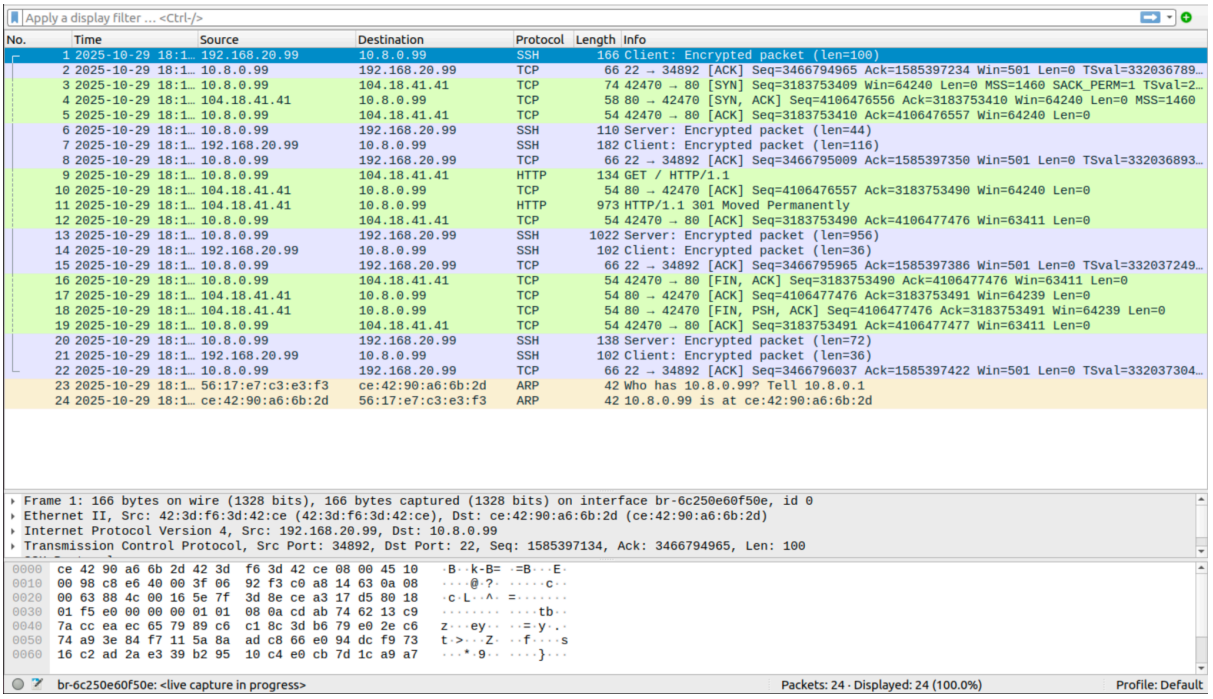
Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6



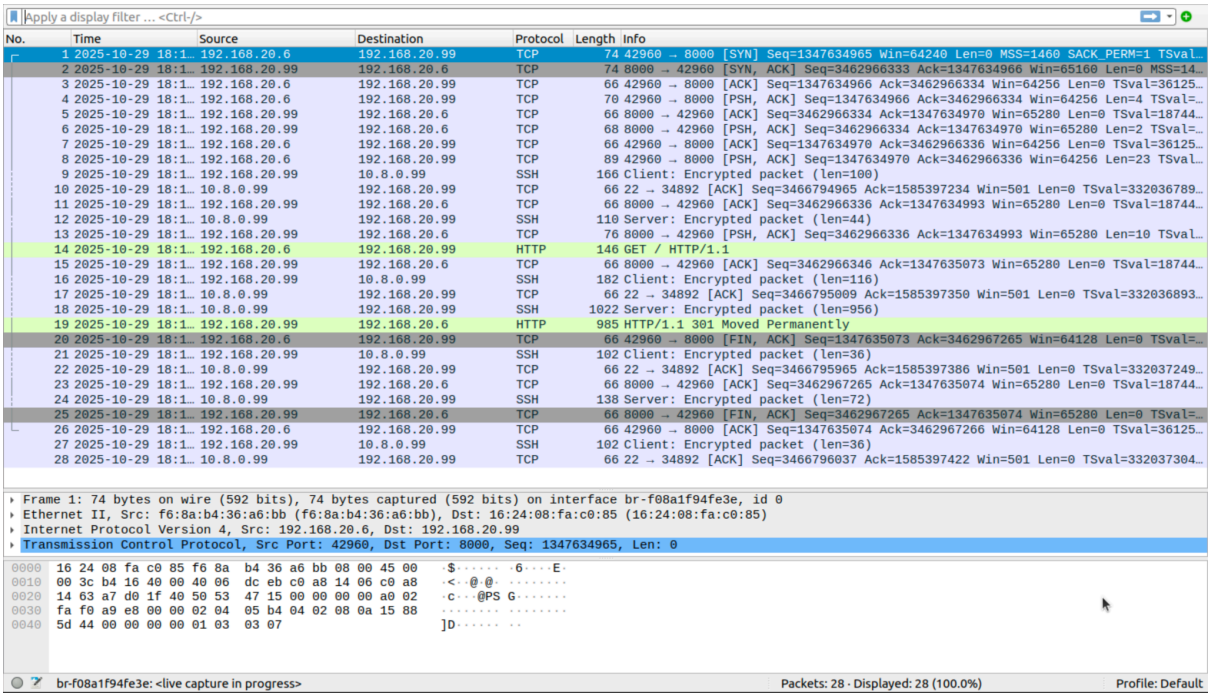
B2-192.168.20.6:

```
B2-192.168.20.6:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://192.168.20.99:8000 http://www.linkedin.com
B2-192.168.20.6:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:



Questions:

1. **Which computer establishes the actual connection with the intended web server?**
   Answer: The external host A (10.8.0.99) establishes the actual connection with the target web server. When host B acts as the SOCKS proxy and receives HTTP requests from internal hosts (like B1 and B2), it forwards those requests through the SSH tunnel to host A. Host A, which sits on the external network, then initiates the outgoing TCP connections to the destination web servers (such as www.example.com or www.linkedin.com).

2. **How does this computer know which server it should connect to?**
   Answer: Host A knows the final destination because the SOCKS5 protocol includes this information inside each client request. When curl (or any SOCKS-capable application) sends data to the proxy on B:8000, it first sends a SOCKS handshake specifying the remote server's hostname and port (for example, "www.example.com:80"). Host B passes this information through the SSH tunnel to host A, which then reads these details and establishes a direct TCP connection to that web server.

Explanation:
In this task, we set up dynamic port forwarding using SSH to transform host B (192.168.20.99) into a SOCKS5 proxy server.

The command: ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N creates an encrypted SSH tunnel between B and A, with host B listening on port 8000 for incoming SOCKS connections.

The -D flag enables dynamic port forwarding, while 0.0.0.0 makes the proxy accessible on all network interfaces, allowing other internal hosts (B1 and B2) to use it as well. When curl on B, B1, or B2 sends an HTTP request using the SOCKS proxy (via -x socks5h://...), the data first travels to host B's proxy port 8000.

From there, it is forwarded securely through the SSH tunnel to A, which then connects to the target website. This process effectively allows all internal hosts to reach blocked external sites, like linkedin.com or example.com, by tunneling their traffic through the allowed SSH channel.

Wireshark captures confirm this behavior:

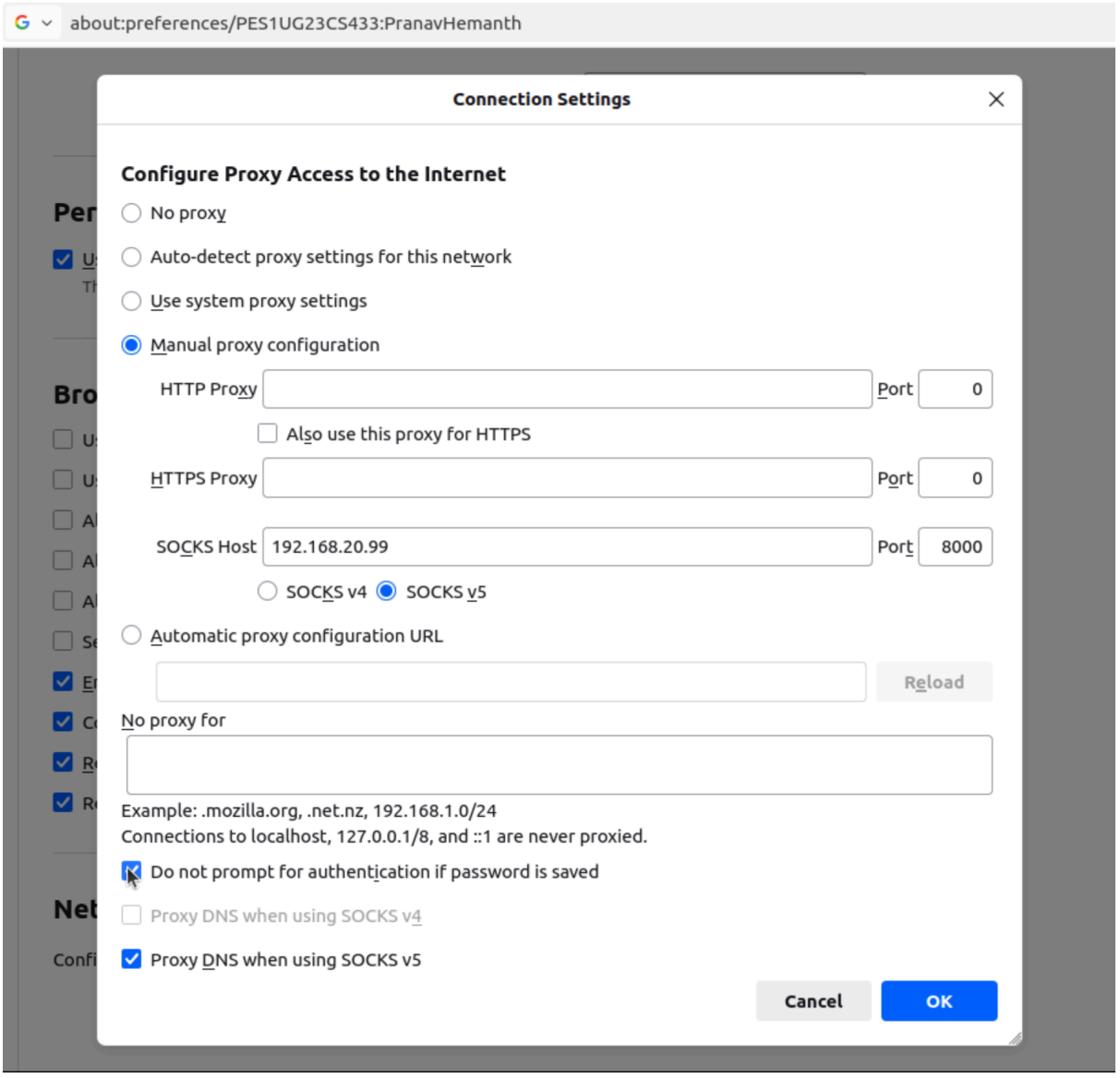
- Traffic between B and A is encrypted SSH on port 22.
- Outbound connections to external servers originate from A's IP, not from B or B1/B2.

This demonstrates how dynamic port forwarding provides a flexible and secure way to bypass egress filtering and access multiple destinations through a single SSH tunnel.

Task 2.2: Testing the Tunnel Using Browser

We can also test the tunnel using a real browser, instead of using curl. Although it is hard to run a browser inside a container, in the docker setup, by default, the host machine is always attached to any network created inside docker, and the first IP address on that network is assigned to the host machine. For example, in our setup, the host machine is the SEED VM; its IP address on the internal network 192.168.20.0/24 is 192.168.20.1.

To use the dynamic port forwarding, we need to configure Firefox's proxy setting. To get to the setting page, we can type about:preferences in the URL field or click the Preference menu item. On the General page, find the "Network Settings" section, click the Settings button, and a window will pop up. Follow the figure to set up the SOCKS proxy.



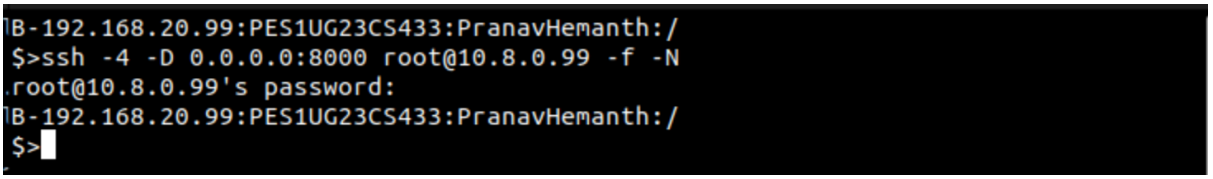
Once the proxy is configured, we can then browse any website. The requests and

replies will go through the SSH tunnel. Since the host VM can reach the Internet directly, to make sure that our web browsing traffic has gone through the tunnel, you should do the following

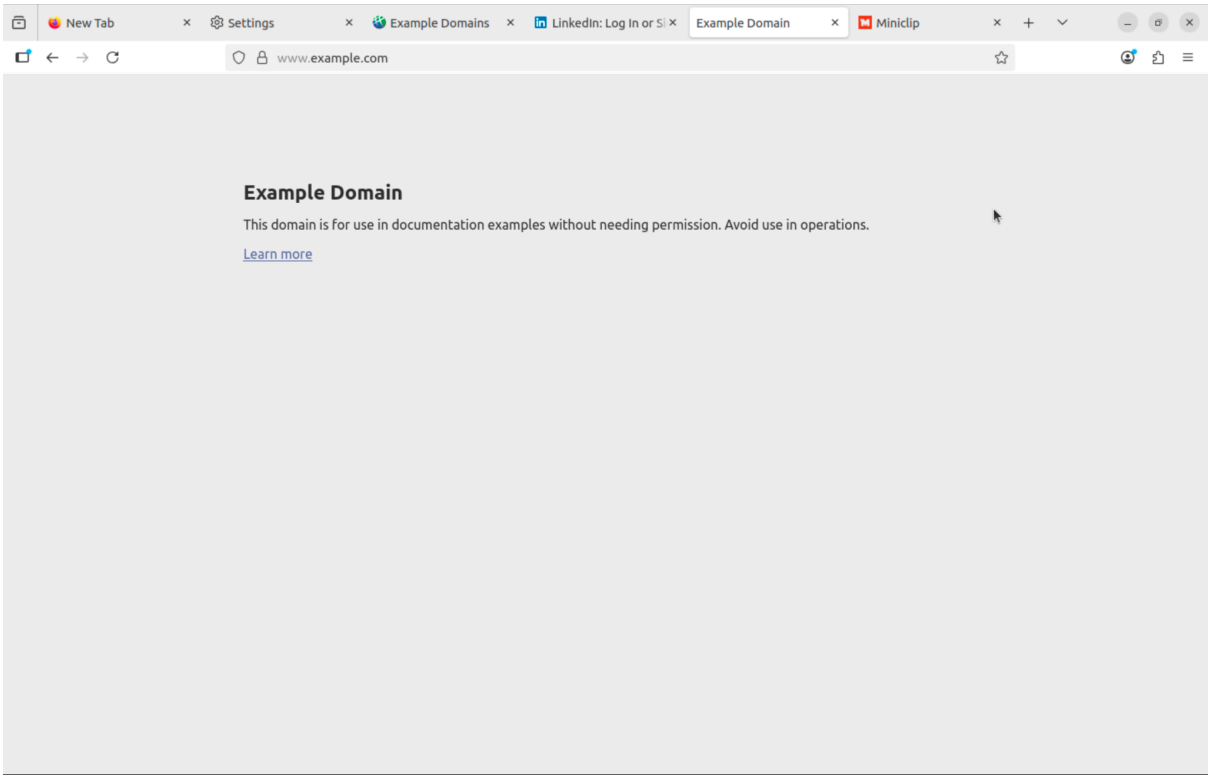
(1) run tcpdump on the router-firewall, and point out the traffic involved in the entire port forwarding process.
Answer: Refer the explanation/observation after the screenshot of tcpdump below in task

(2) Break the SSH tunnel, and then try to browse a website. Describe your observation.
Answer: refer to the explanation/observation after the screenshot in task below

Please note, we are still using the ssh connection established in the previous Task 2.1.
Access the websites as you normally would in firefox and provide screenshots of your observations.
To close the ssh shell that is in the background created using the previous “ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N” command:

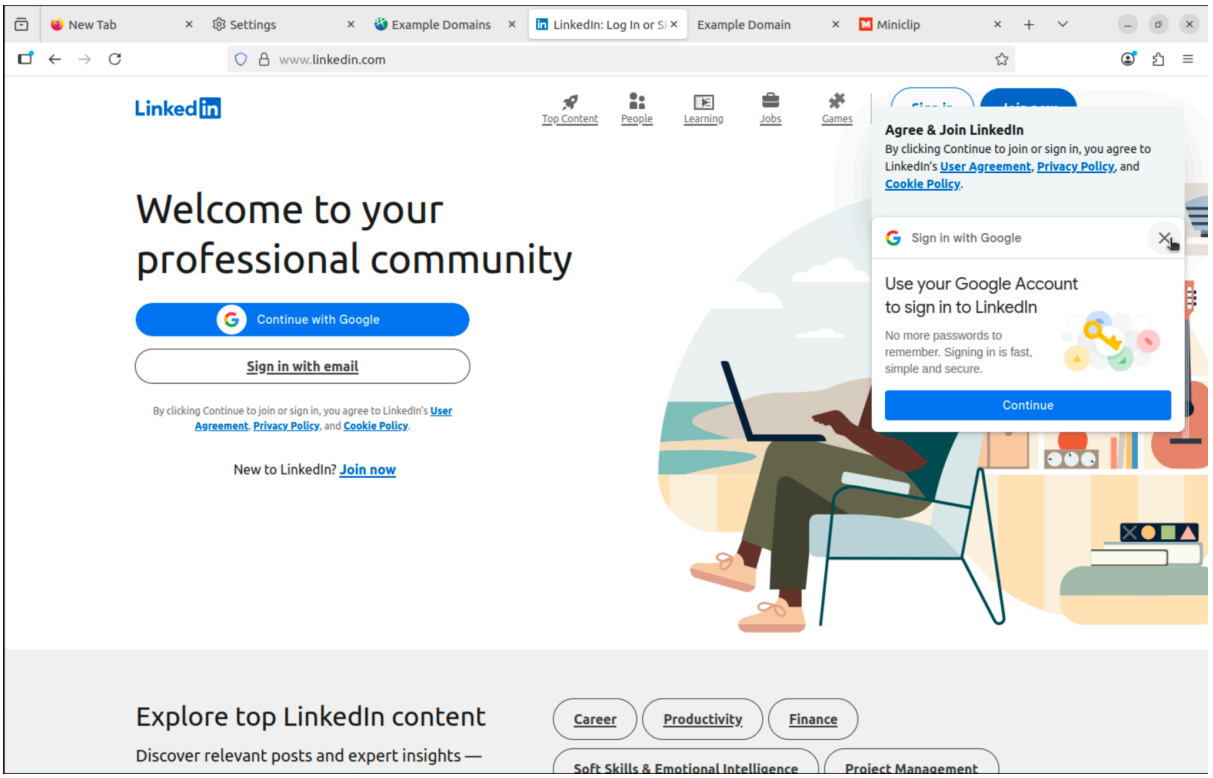


www.example.com:

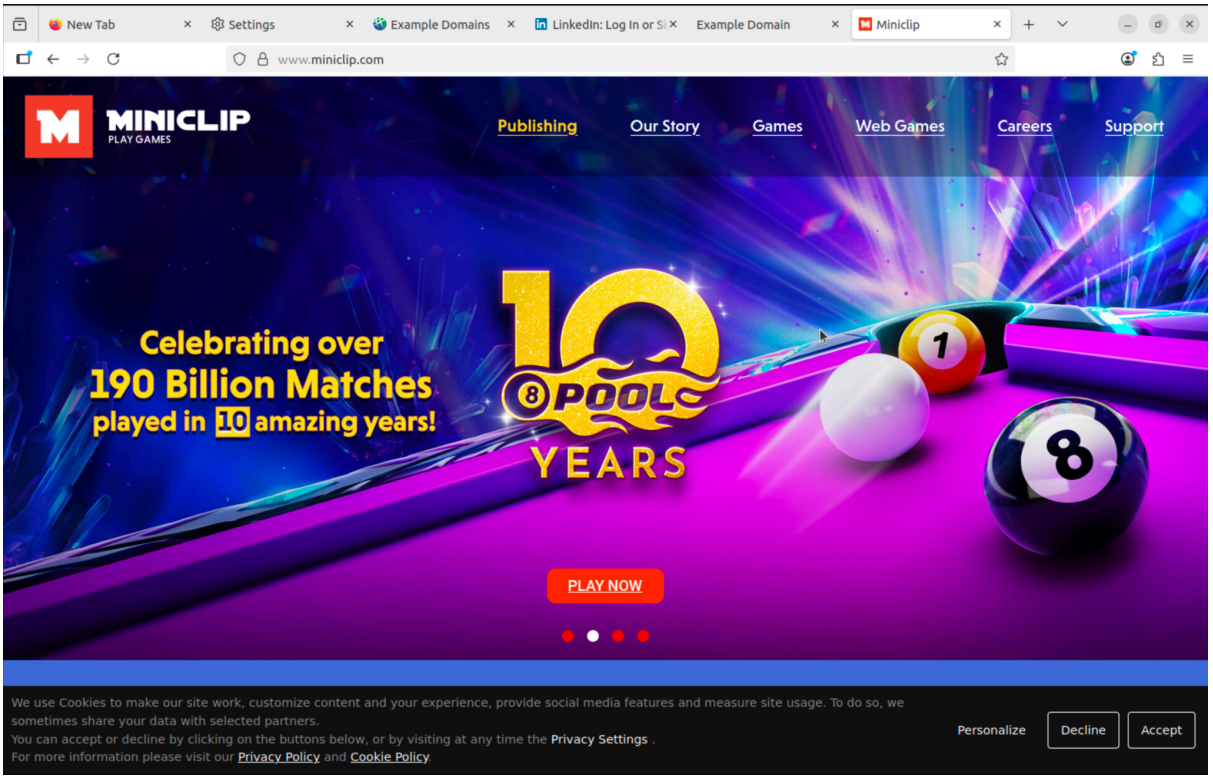


Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

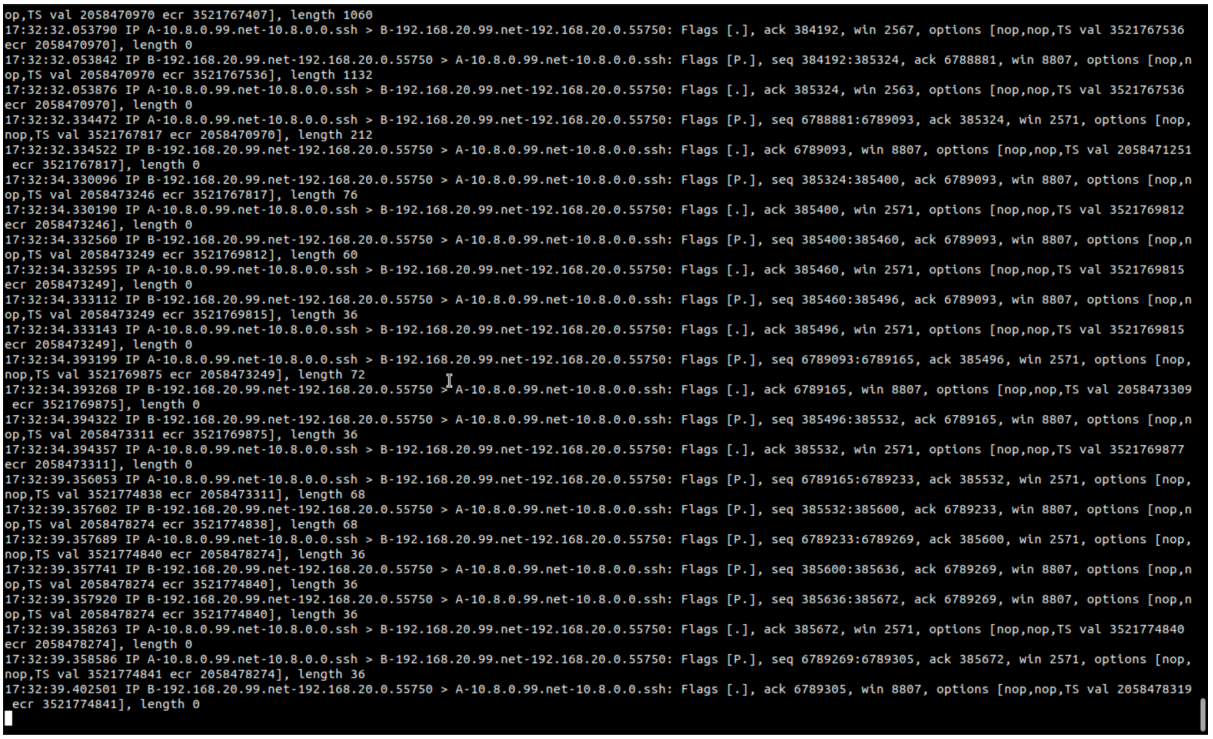
www.linkedin.com:



www.miniclip.com:



router-firewall (tcpdump):



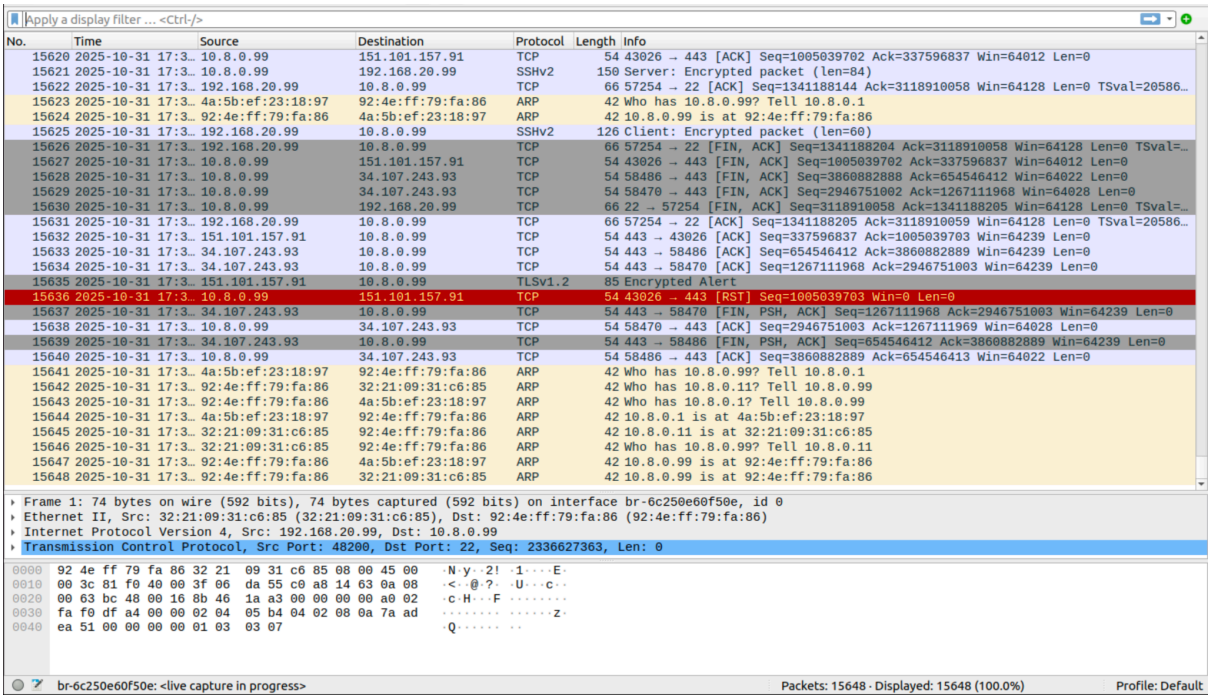
When tcpdump is run on the router-firewall during the browser test, the captured traffic showed the following:

- Encrypted SSH packets flowing between 192.168.20.99 (host B) and 10.8.0.99 (host A) on TCP port 22.
- Outbound HTTP/HTTPS connections originating from 10.8.0.99 to the destination IPs of visited websites.
- No direct traffic from internal hosts (B, B1, or B2) to the external websites was observed.

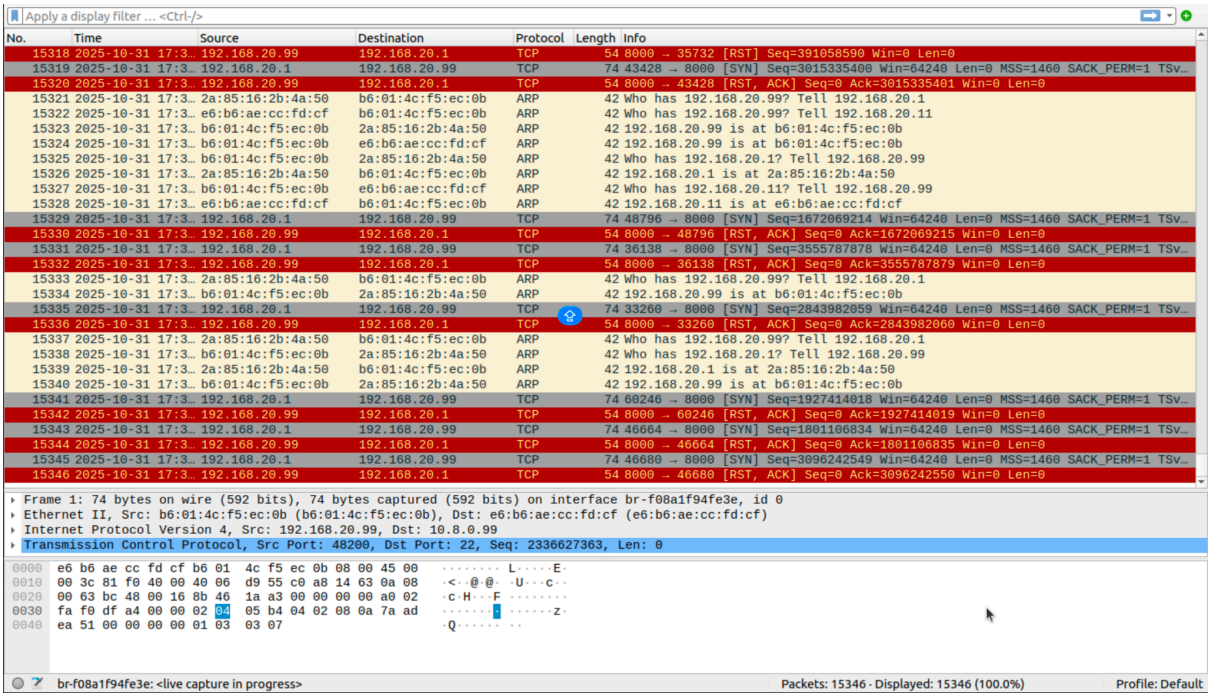
This indicates that the router-firewall only handled SSH-encrypted traffic between B and A, and all actual web requests and responses were encapsulated within the tunnel.

Hence, tcpdump verifies that the SSH tunnel hides internal browsing activity from the local network, the firewall sees only encrypted packets, not the contents of the web sessions.

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:



Step1: Execute the following iptables commands on the seed-router container-

29)Command: ps -eaf | grep "ssh"

30)Command: kill [corresponding pid of the process]

B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ps -eaf | grep "ssh"
root          39       1  0 16:52 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          51       1  0 17:01 ?        00:00:00 [ssh] <defunct>
root          58       1  0 17:28 ?        00:00:00 [ssh] <defunct>
root          65       1  0 17:34 ?        00:00:00 ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root          70      41  0 17:35 pts/1    00:00:00 grep ssh
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

B-192.168.20.99:

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ps -eaf | grep "ssh"
root          39       1  0 16:52 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          51       1  0 17:01 ?        00:00:00 [ssh] <defunct>
root          58       1  0 17:28 ?        00:00:00 [ssh] <defunct>
root          65       1  0 17:34 ?        00:00:00 ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root          70      41  0 17:35 pts/1    00:00:00 grep ssh
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>kill 65
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ps -eaf | grep "ssh"
root          39       1  0 16:52 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          51       1  0 17:01 ?        00:00:00 [ssh] <defunct>
root          58       1  0 17:28 ?        00:00:00 [ssh] <defunct>
root          65       1  0 17:34 ?        00:00:00 [ssh] <defunct>
root          72      41  0 17:35 pts/1    00:00:00 grep ssh
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Observation:
When the browser was configured with the SOCKS proxy and a website was accessed, the page loaded successfully.

Wireshark and tcpdump traces show that:

- All traffic between host B and host A was encrypted SSH traffic on port 22, with no direct HTTP/HTTPS packets visible on the local network.
- Outbound connections to external web servers were made from host A, confirming that A handled the actual browsing traffic.

This demonstrates successful dynamic port forwarding, where the SSH tunnel securely carries web traffic between the internal and external networks.

When the SSH tunnel between host B and host A was intentionally terminated and the browser attempted to load a website again, the page failed to load.

The browser either displayed a “Proxy connection failed” or “Unable to connect to the server” message.

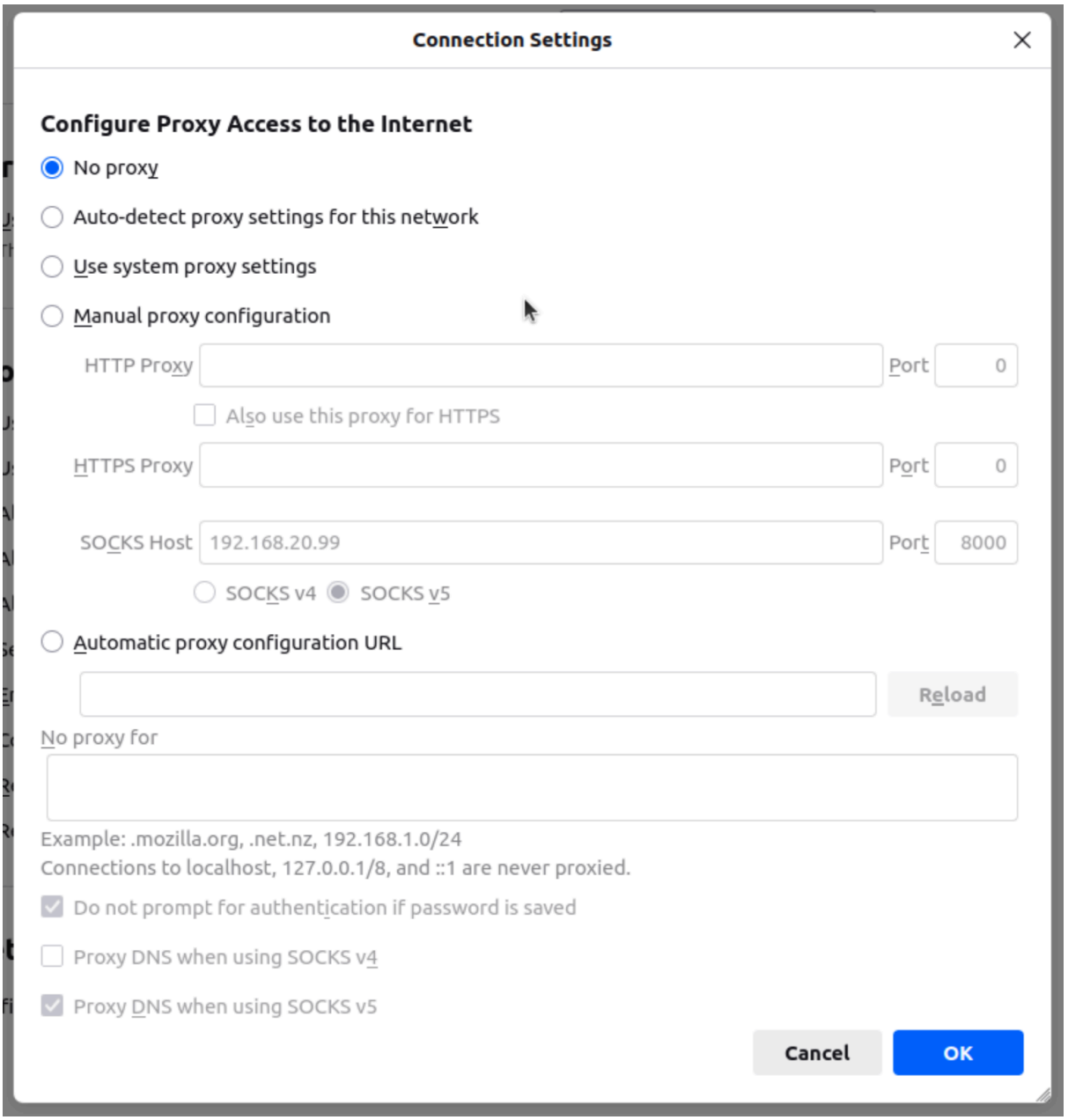
This happened because once the SSH tunnel was broken:

- The SOCKS proxy at 192.168.20.99:8000 was no longer active.
- The browser continued to send requests to that proxy port, but there was no SSH process to forward them to host A.

- As a result, all requests were dropped locally, and no external connections were made.

The failure confirms that the browser's internet access fully depended on the active SSH tunnel. Once the tunnel was stopped, all traffic routed through the proxy immediately ceased, demonstrating the tunnel's central role in handling and securing outbound web traffic.

Step1: Cleanup. After this task, please make sure to remove the proxy setting from Firefox by checking the "No proxy" option. Without a proper cleanup, future tasks may be affected.



Explanation:
In this task, the web browser is configured to use the SOCKS proxy created in the previous step (Task 2.1) to verify that the tunnel works for real-world applications, not just command-line tools like curl.

The browser's network settings are modified to use a SOCKS5 proxy pointing to host B (192.168.20.99) on port 8000, where the SSH tunnel is listening. Once configured, all HTTP and HTTPS requests made by the browser are automatically redirected through this proxy. When a website is accessed (for example, www.example.com or www.linkedin.com), the browser sends the request to host B's SOCKS port. B forwards it through the encrypted SSH tunnel to host A (10.8.0.99). Host A then connects directly to the destination web server and returns the response back through the tunnel to B, which delivers it to the browser.

This confirms that the dynamic port forwarding setup functions as a full-fledged VPN-like tunnel. From the browser's perspective, it appears as if the requests are being made directly from host A's external network, effectively bypassing local network restrictions or filters. Wireshark traces show that all traffic between B and A is encrypted SSH traffic on port 22, while the actual web traffic (HTTP/HTTPS) occurs only between A and the target sites.

This proves the tunnel's end-to-end operation and verifies that any application supporting SOCKS proxies (like browsers) can securely send traffic through the established SSH tunnel.

Task 2.3: Writing a SOCKS Client Using Python

For port forwarding to work, we need to specify where the data should be forwarded to (the final destination). In the static case, this piece of information is provided when we set up the tunnel, i.e., it is hard-wired into the tunnel setup. In the dynamic case, the final destination is dynamic, not specified during the setup, so how can the proxy know where to forward the data?
Applications using a dynamic port forwarding proxy must tell the proxy where to forward their data. This is done through an additional protocol between the application and the proxy. A common protocol for such a purpose is the SOCKS (Socket Secure) protocol, which becomes a de facto proxy standard.

Since the application needs to interact with the proxy using the SOCKS protocol, the application software must have a native SOCKS support in order to use SOCKS proxies. Both Firefox and curl have such a support, but we cannot directly use this type of proxy for the telnet program, because it does not provide a native SOCKS support. In this task, we implement a very simple SOCKS client program using Python.

Please complete this program, and use it to access http://www.example.com from hosts B, B1, and B2. The code given above is only for sending HTTP requests, not HTTPS requests (sending HTTPS requests are much more complicated due to the TLS handshake). For this task, students only need to send HTTP requests.

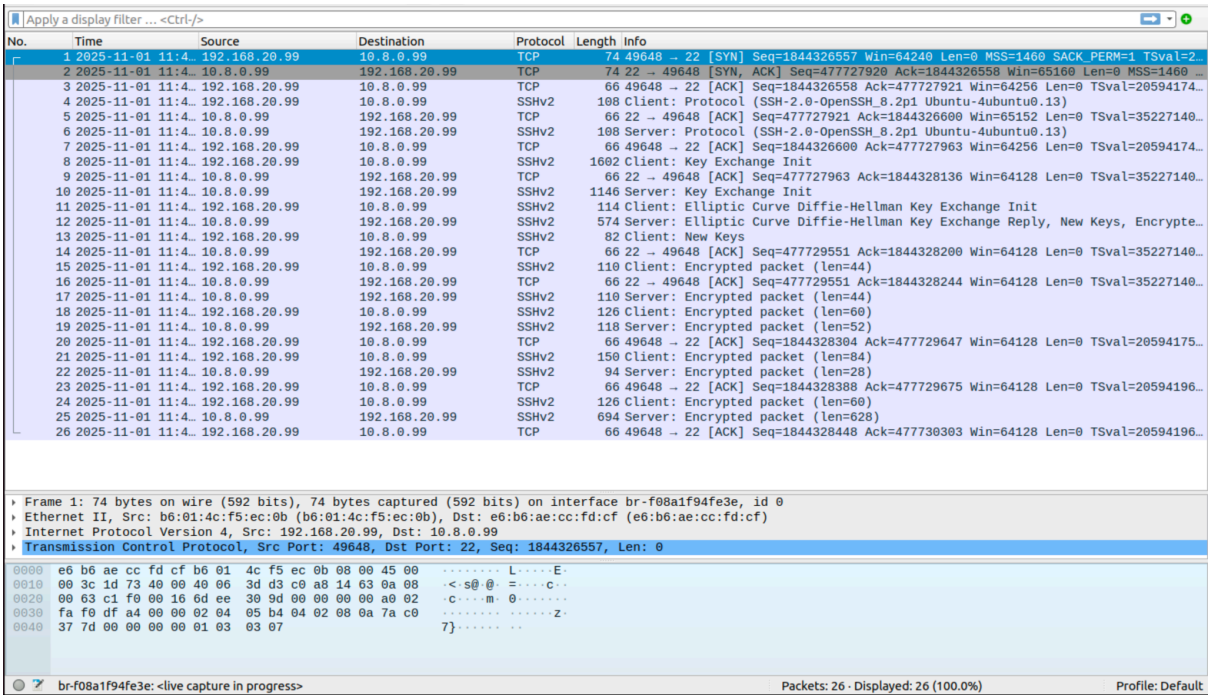
Step1: Run in container B to set up the ssh shell with dynamic port forwarding enabled:

31)Command: ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N

B-192.168.20.99:

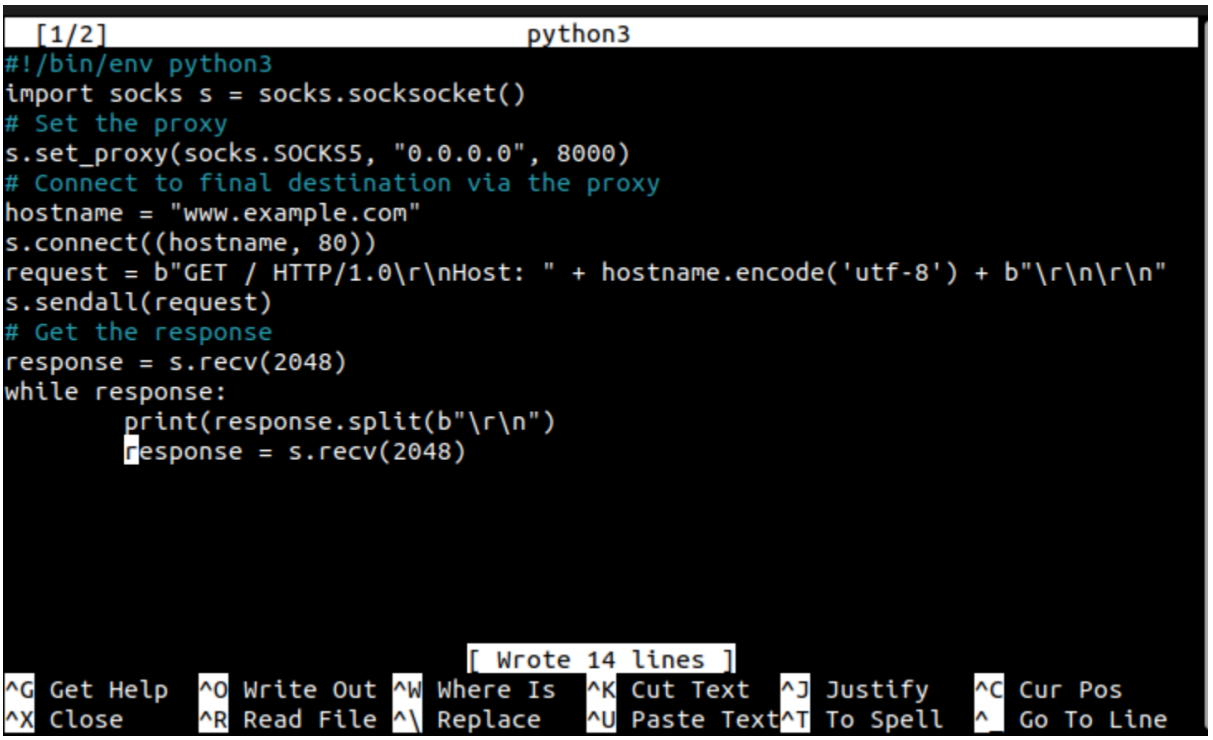
```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root@10.8.0.99's password:
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



Step2: Run in container B and paste given code in the instruction document:

32)Command: nano python3 B-Socks-Client.py




```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>nano python3 B-Socks-Client.py
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>
```

Step3: Run in container B:

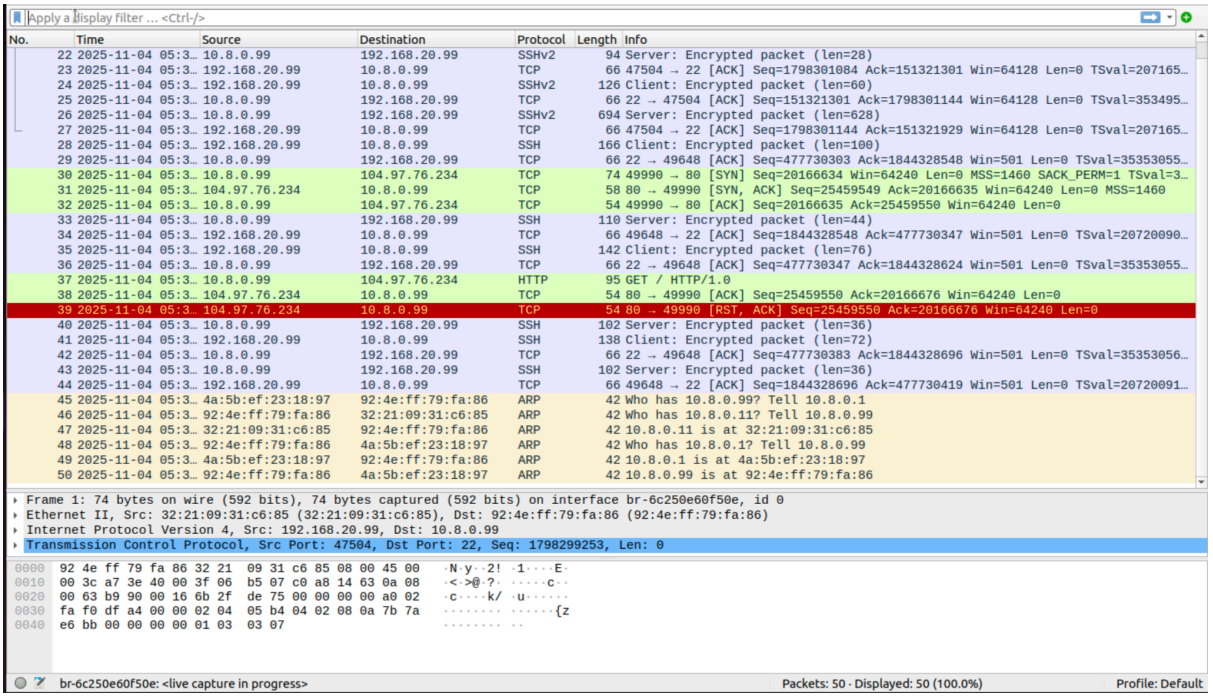
33)Command: python3 B-Socks-Client.py

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>nano B-Socks-Client.py
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>python3 B-Socks-Client.py
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

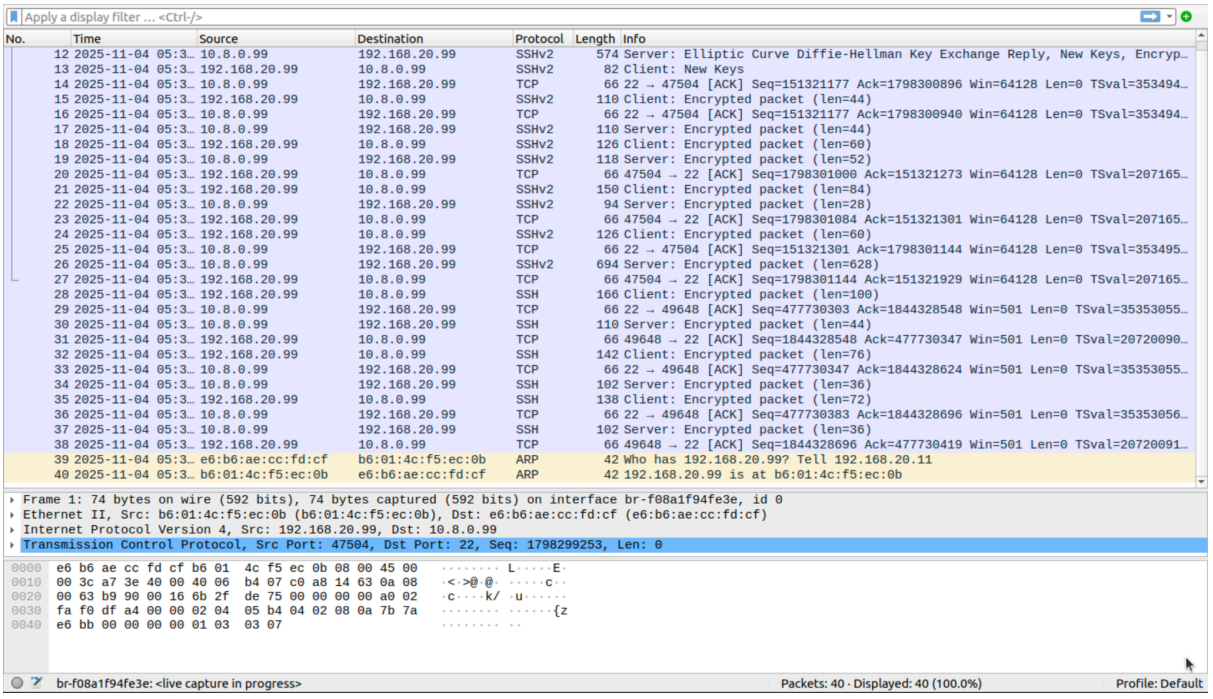
```
$>python3 B-Socks-Client.py
[b'HTTP/1.0 200 OK', b'Accept-Ranges: none', b'Content-Type: text/html', b'ETag:
 "bc2473a18e003bdb249eba5ce893033f:1760028122.592274"', b'Last-Modified: Thu, 09
 Oct 2025 16:42:02 GMT', b'Cache-Control: max-age=86000', b'Date: Tue, 04 Nov 20
25 05:44:51 GMT', b'Content-Length: 513', b'Via: HTTP/1.1 forward.http.proxy:312
8', b'Connection: close', b'', b'<!doctype html><html lang="en"><head><title>Exa
mple Domain</title><meta name="viewport" content="width=device-width, initial-sc
ale=1"><style>body{background:#eee;width:60vw;margin:15vh auto;font-family:syste
m-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{color:#348}<
/style><body><div><h1>Example Domain</h1><p>This domain is for use in documentat
ion examples without needing permission. Avoid use in operations.<p><a href="htt
ps://iana.org/domains/example">Learn more</a></div></body></html>\n']
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

Wireshark on 10.8.0.1 network:

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6



Wireshark on 192.168.20.1 network:



Step4: Run in container B1 or B2 and paste given code in the instruction document:

34)Command: nano python3 B1-B2-Socks-Client.py or nano B1-B2-Socks-Client.py

```
  GNU nano 4.8                    B1-B2-Socks-Client.py
#!/bin/env python3
import socks
s = socks.socksocket()
# Set the proxy
s.set_proxy(socks.SOCKS5, "192.168.20.99", 8000)
# Connect to final destination via the proxy
hostname = "www.example.com"
s.connect((hostname, 80))
request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b"\r\n\r\n"
s.sendall(request)
# Get the response
response = s.recv(2048)
while response:
        print(response.split(b"\r\n"))
        response = s.recv(2048)


                        [ Wrote 15 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
```

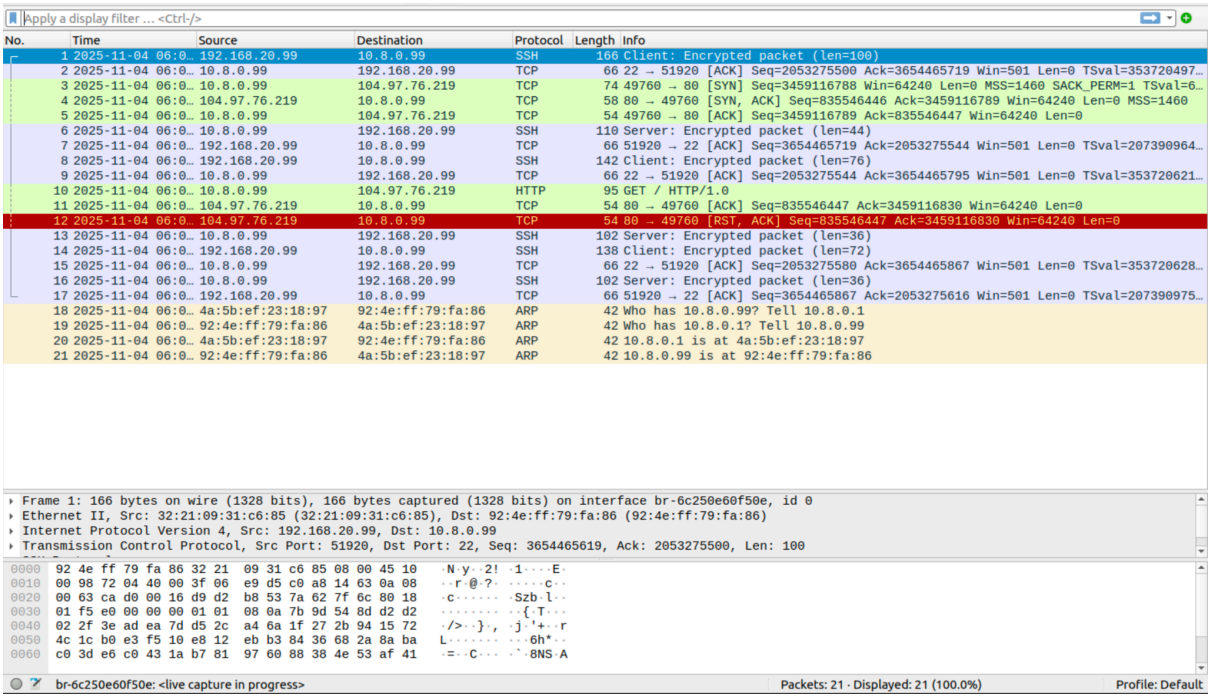
```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>nano B1-B2-Socks-Client.py
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>
```

Step5: Run in container B1:

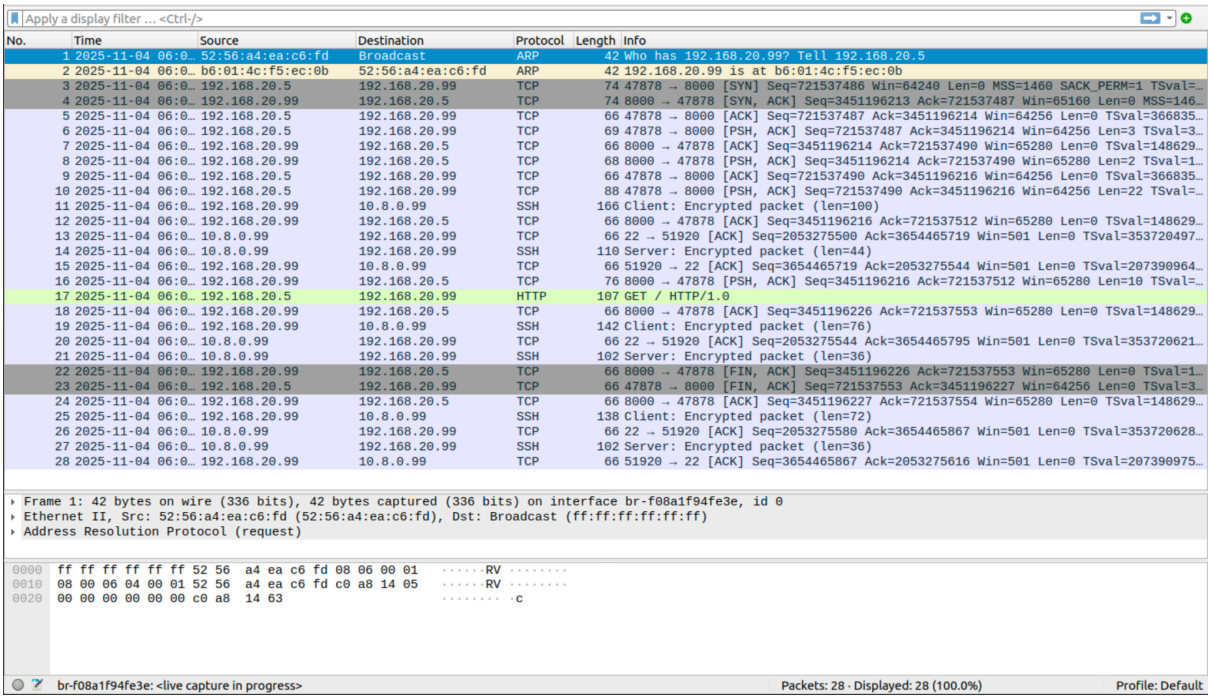
35)Command: python3 B1-B2-Socks-Client.py

```
$>python3 B1-B2-Socks-Client.py
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
The authenticity of host '10.8.0.99 (10.8.0.99)' can't be established.
ECDSA key fingerprint is SHA256:fkGtseinXKS9urPzu8YEIMj8qzO5MJzoi5ekV2x96MQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.8.0.99' (ECDSA) to the list of known hosts.
root@10.8.0.99's password:
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>python3 B1-B2-Socks-Client.py
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:



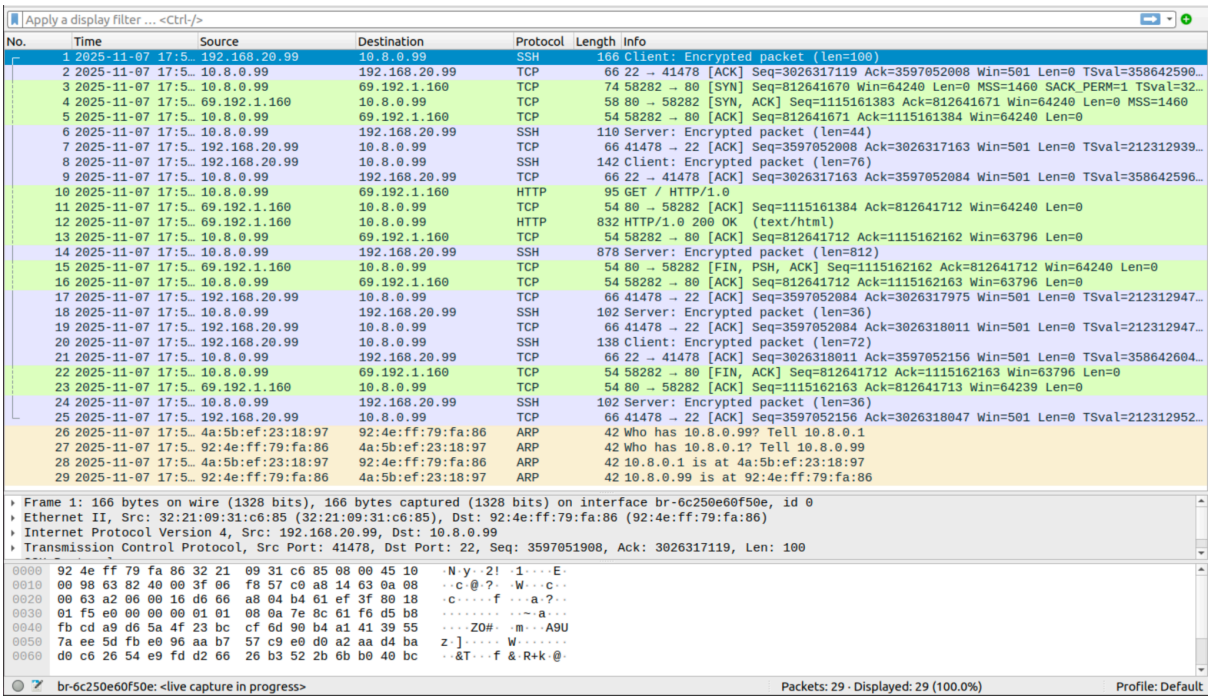
Trying again since the script didnt return a page:

Step5: Run in container B1:

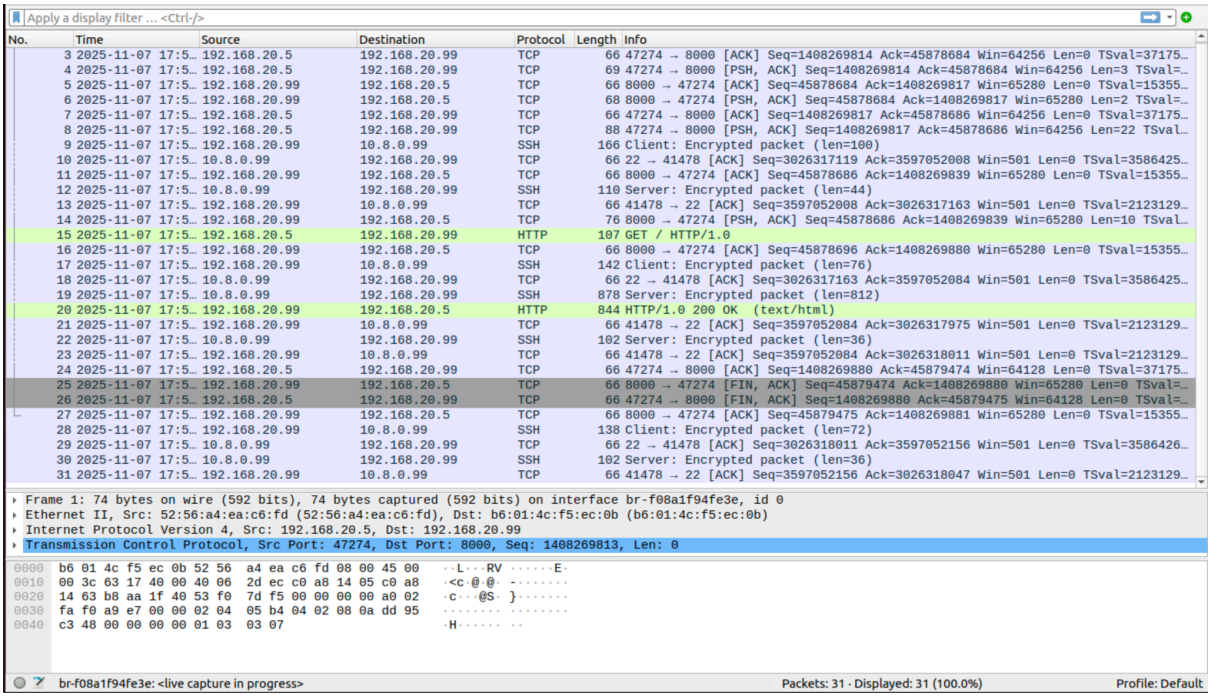
36)Command: python3 B1-B2-Socks-Client.py

```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>python3 B1-B2-Socks-Client.py
[b'HTTP/1.0 200 OK', b'Content-Type: text/html', b'ETag: "bc2473a18e003bdb249eba
5ce893033f:1760028122.592274"', b'Last-Modified: Thu, 09 Oct 2025 16:42:02 GMT',
 b'Cache-Control: max-age=86000', b'Date: Fri, 07 Nov 2025 17:58:48 GMT', b'Cont
ent-Length: 513', b'Connection: close', b'X-N: S', b'', b'<!doctype html><html l
ang="en"><head><title>Example Domain</title><meta name="viewport" content="width
=device-width, initial-scale=1"><style>body{background:#eee;width:60vw;margin:15
vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:li
nk,a:visited{color:#348}</style><body><div><h1>Example Domain</h1><p>This domain
 is for use in documentation examples without needing permission. Avoid use in o
perations.<p><a href="https://iana.org/domains/example">Learn more</a></div></bo
dy></html>\n']
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

Wireshark on 10.8.0.1 network:



Wireshark on 192.168.20.1 network:



Explanation:
In this task, a simple Python-based SOCKS5 client was implemented to demonstrate how an application without native SOCKS support can communicate through a SOCKS proxy. Host B (192.168.20.99) was configured as a SOCKS5 proxy using dynamic port forwarding (ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N). The Python client script connected to this proxy and performed the SOCKS5 handshake, specifying the destination (www.example.com:80). The proxy then forwarded the request through the encrypted SSH tunnel to host A (10.8.0.99), which established the actual HTTP connection with the external web server and returned the response back through the tunnel. Wireshark traces confirmed that all traffic between B and A was encrypted SSH traffic on port 22, and outbound HTTP requests originated from host A. This verifies that the SOCKS client correctly used the proxy to securely tunnel web requests through the dynamic port forwarding setup.

Step6: To close the ssh shell that is in the background created using the previous “ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N”

37)Command: (On B1)

ps -eaf | grep "ssh"

```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>ps -eaf | grep ssh
root          43       1  0 04:20 ?        00:00:00 ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root          49      24  0 18:00 pts/1    00:00:00 grep ssh
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

kill [corresponding pid of the process]

```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>kill 43
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>ps -eaf | grep ssh
root          43       1  0 04:20 ?        00:00:00 [ssh] <defunct>
root          51      24  0 18:02 pts/1    00:00:00 grep ssh
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

38)Command: (On B)

ps -eaf | grep "ssh"

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>ps -eaf | grep ssh
root          39       1  0 Nov06 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          51       1  0 Nov06 ?        00:00:00 [ssh] <defunct>
root          58       1  0 Nov06 ?        00:00:00 [ssh] <defunct>
root          65       1  0 00:02 ?        00:00:00 [ssh] <defunct>
root          83       1  0 00:16 ?        00:00:00 [ssh] <defunct>
root         114       1  0 03:40 ?        00:00:00 [ssh] <defunct>
root         145       1  0 03:49 ?        00:00:00 [ssh] <defunct>
root         156       1  0 03:51 ?        00:00:00 [ssh] <defunct>
root         170       1  0 03:58 ?        00:00:00 [ssh] <defunct>
root         192       1  0 17:12 ?        00:00:00 [ssh] <defunct>
root         203       1  0 17:13 ?        00:00:00 [ssh] <defunct>
root         218       1  0 17:46 ?        00:00:00 ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root         229      41  0 18:02 pts/1    00:00:00 grep ssh
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

kill [corresponding pid of the process]

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>ps -eaf | grep ssh
root          39       1  0 Nov06 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          51       1  0 Nov06 ?        00:00:00 [ssh] <defunct>
root          58       1  0 Nov06 ?        00:00:00 [ssh] <defunct>
root          65       1  0 00:02 ?        00:00:00 [ssh] <defunct>
root          83       1  0 00:16 ?        00:00:00 [ssh] <defunct>
root         114       1  0 03:40 ?        00:00:00 [ssh] <defunct>
root         145       1  0 03:49 ?        00:00:00 [ssh] <defunct>
root         156       1  0 03:51 ?        00:00:00 [ssh] <defunct>
root         170       1  0 03:58 ?        00:00:00 [ssh] <defunct>
root         192       1  0 17:12 ?        00:00:00 [ssh] <defunct>
root         203       1  0 17:13 ?        00:00:00 [ssh] <defunct>
root         218       1  0 17:46 ?        00:00:00 ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N
root         229      41  0 18:02 pts/1    00:00:00 grep ssh
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>kill 218
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>ps -eaf | grep ssh
root          39       1  0 Nov06 ?        00:00:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          51       1  0 Nov06 ?        00:00:00 [ssh] <defunct>
root          58       1  0 Nov06 ?        00:00:00 [ssh] <defunct>
root          65       1  0 00:02 ?        00:00:00 [ssh] <defunct>
root          83       1  0 00:16 ?        00:00:00 [ssh] <defunct>
root         114       1  0 03:40 ?        00:00:00 [ssh] <defunct>
root         145       1  0 03:49 ?        00:00:00 [ssh] <defunct>
root         156       1  0 03:51 ?        00:00:00 [ssh] <defunct>
root         170       1  0 03:58 ?        00:00:00 [ssh] <defunct>
root         192       1  0 17:12 ?        00:00:00 [ssh] <defunct>
root         203       1  0 17:13 ?        00:00:00 [ssh] <defunct>
root         218       1  0 17:46 ?        00:00:00 [ssh] <defunct>
root         231      41  0 18:03 pts/1    00:00:00 grep ssh
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

Now try running the python files again in the containers

Step7: Run in container B:

39)Command: python3 B-Socks-Client.py

```
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>python3 B-Socks-Client.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/socks.py", line 787, in connect
    super(socksocket, self).connect(proxy_addr)
ConnectionRefusedError: [Errno 111] Connection refused

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "B-Socks-Client.py", line 8, in <module>
    s.connect((hostname, 80))
  File "/usr/local/lib/python3.8/dist-packages/socks.py", line 47, in wrapper
    return function(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/socks.py", line 800, in connect
    raise ProxyConnectionError(msg, error)
socks.ProxyConnectionError: Error connecting to SOCKS5 proxy 0.0.0.0:8000: [Errno 111] Connection refused
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

Step8: Run in container B1:

40)Command: python3 B1-B2-Socks-Client.py

```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>python3 B1-B2-Socks-Client.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/socks.py", line 787, in connect
    super(socksocket, self).connect(proxy_addr)
ConnectionRefusedError: [Errno 111] Connection refused

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "B1-B2-Socks-Client.py", line 8, in <module>
    s.connect((hostname, 80))
  File "/usr/local/lib/python3.8/dist-packages/socks.py", line 47, in wrapper
    return function(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/socks.py", line 800, in connect
    raise ProxyConnectionError(msg, error)
socks.ProxyConnectionError: Error connecting to SOCKS5 proxy 192.168.20.99:8000: [Errno 111] Connection refused
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/home/seed
$>
```

Explanation:

After terminating the SSH processes on hosts B and B1 using the ps -eaf | grep "ssh" and kill commands, the dynamic port forwarding tunnels were closed. When the Python SOCKS client scripts (B-Socks-Client.py and B1-B2-Socks-Client.py) were executed again, the connections failed with a “Connection refused” error. This occurred because the SOCKS proxy at port 8000 was no longer active—without the SSH tunnel, there was no process listening to forward requests through host A. As a result, all connection attempts from the Python clients were rejected locally, confirming that the SOCKS proxy relied entirely on the active SSH tunnel for communication. Once the tunnel was terminated, the secure forwarding path to external web servers was lost, demonstrating the dependency of the SOCKS-based connection on the SSH dynamic port forwarding setup.

Task 3: Comparing SOCKS5 Proxy and VPN

Both SOCKS5 proxy (dynamic port forwarding) and VPN are commonly used in creating tunnels to bypass firewalls, as well as to protect communications. Many VPN service providers provide both types of services. Sometimes, when a VPN service provider tells you that it provides the VPN service, but in reality, it is just a SOCKS5 proxy. Although both technologies can be used to solve the same problem, they do have significant differences. Please compare these two technologies, describing their differences, pros and cons.

Explanation**:**

Both SOCKS5 proxy (dynamic port forwarding) and VPN technologies create secure tunnels that can be used to bypass firewalls and protect data in transit. Although their purposes often overlap, they operate at different network layers and provide different levels of protection.

A SOCKS5 proxy functions at the application layer.

When SSH dynamic port forwarding (ssh -D) is used, the client establishes an encrypted SSH tunnel to a remote host and listens locally on a SOCKS port (for example : 8000).

Only applications explicitly configured to use this proxy (such as browsers or curl) will have their traffic forwarded through the tunnel. The remote host then initiates the real connection to the destination web server on behalf of the client. Hence, the SOCKS5 proxy provides per-application tunnelling rather than full-system coverage.

A VPN, in contrast, operates at the network layer.

It creates a virtual network interface (for example : tun0) and routes all IP packets through an encrypted tunnel between the client and the VPN server. Once the VPN is active, every program on the system- browser, ping, SSH, or any other- sends its traffic through the tunnel automatically. VPNs therefore provide system-wide encryption and routing, offering stronger privacy and broader protection than a SOCKS proxy.

Comparison:

Feature	SOCKS5 Proxy (SSH Dynamic Port Forwarding)	VPN (Virtual Private Network)

Layer of operation	Application layer (L5–L7)	Network layer (L3)
Traffic scope	Only proxy-aware apps	All system traffic
Encryption	Provided only if tunnelled through SSH	Always encrypted (OpenVPN, IPsec, WireGuard, etc.)
Configuration	Simple – ssh -D and browser proxy setting	Requires VPN client, keys and root access
Performance	Faster (minimal overhead)	Slightly slower (due to encryption)
Firewall bypass	Effective if SSH port 22 is allowed	More robust – encapsulates all traffic
Privacy scope	Per-application IP masking	Full device IP masking and DNS protection
Typical use	Tunnelling browser or specific apps	Secure remote access and full-network privacy

Advantages and Disadvantages:

SOCKS5 Proxy

Advantages:

- Lightweight and easy to set up with a single SSH command.
- Low latency since only selected traffic is forwarded.

- Works well for quickly bypassing restricted sites.

Disadvantages:

- Protects only configured applications.
- No inherent encryption outside of SSH.
- Does not conceal DNS or non-SOCKS traffic.

VPN

Advantages:

- Encrypts and routes all traffic leaving the device.
- Provides complete anonymity and protection on public networks.
- Transparent to all applications.

Disadvantages:

- Slightly higher setup complexity.
- Encryption overhead may reduce throughput.
- May require administrative privileges and specific ports.

Demonstration

The behavior and differences between SOCKS5 proxy and VPN were already demonstrated in the previous tasks.

In Task 2.1 and Task 2.2, we implemented and tested SSH dynamic port forwarding (SOCKS5 proxy) using the command:

```
ssh -D 0.0.0.0:8000 root@10.8.0.99 -f -N
```

During those tests, the browser was configured to use the SOCKS5 proxy at 192.168.20.99:8000, and visiting https://whatismyipaddress.com showed the external IP of Host A (10.8.0.99).

This confirmed that the web traffic was being securely relayed through the SSH tunnel.

Packet captures (tcpdump) on the router-firewall showed only SSH (port 22) packets and no visible HTTP traffic, proving that the data was encapsulated inside the SSH session and only proxy-enabled applications (like the browser) were affected.

In contrast, in Lab 9, we set up a VPN tunnel using the scripts tun_client_select.py and tun_server_select.py.

There, all applications, including ping, browser, and telnet, successfully operated through the VPN, demonstrating full-system tunnelling.

The tcpdump output revealed encapsulated UDP/TUN packets rather than plain traffic, confirming that every packet from the client passed through the encrypted VPN tunnel.

Based on these demonstrations:

- The SOCKS5 proxy encrypted and forwarded traffic only for proxy-configured applications (application-level tunnelling).
- The VPN encrypted and forwarded all network traffic from the client, providing system-wide, end-to-end security and routing.

These observations clearly show that while both mechanisms use tunnelling for secure communication, a SOCKS5 proxy operates at the application level, whereas a VPN functions at the network layer, offering broader protection and coverage.

Conclusion

Both SOCKS5 and VPN technologies provide secure tunnelling to bypass firewalls and hide internal traffic.

However, a SOCKS5 proxy offers a lightweight, per-application solution ideal for quick or specific tasks, while a VPN delivers comprehensive, system-wide encryption and privacy suitable for full-network protection.

In short:

SOCKS5 Proxy → Fast and simple (app-level)
VPN → Secure and complete (network-level)

Conclusion:

This lab successfully demonstrated various techniques to bypass firewall restrictions using SSH tunneling and SOCKS proxies. Through static port forwarding, we established fixed tunnels for specific destinations, while dynamic port forwarding enabled flexible, SOCKS5-based tunneling for multiple sites through a single encrypted SSH connection. Using both command-line tools like curl and a web browser, we verified that blocked websites could be accessed securely through these tunnels. The implementation of a custom Python SOCKS client further illustrated how applications without native proxy support can communicate via a SOCKS proxy. Finally, by comparing SOCKS5 proxies with VPNs, we understood the key differences between application-layer and network-layer tunneling. Overall, this lab highlighted how SSH-based port forwarding enhances privacy, enables secure communication, and effectively evades firewall-imposed restrictions.