



# 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

**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.

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

### 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.

The screenshot shows three terminal windows on a Linux system (seed@seedvm2004) displaying the following information:

- Terminal 1:** Shows Docker logs for a build process, listing 11 containers created, including hosts, routers, and various A/B containers. It also shows the command to attach to the A1-10.8.0.5 container.
- Terminal 2:** Shows the output of the command `docker ps`, listing the 11 containers with their IDs, images, status, ports, and commands. The containers include "seed-image-ubuntu-hosts", "B1-192.168.20.5", "A1-10.8.0.5", and others.
- Terminal 3:** Shows the output of the command `ifconfig`, displaying network interface statistics for eth0, eth1, and lo. The eth0 interface is connected to the 10.8.0.0/24 network, while eth1 is connected to the 192.168.20.0/24 network.

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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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.

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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,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  
  
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.

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```

seed@seedvms004: ~
#11 |A-10.8.0.99:PES1UG23CSB1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
[1] $>ip -br address
lo      UNKNOWN        UNKNOWN      127.0.0.1/8 :/1:128
eth0@lf26    UP    eth0@lf25      UP      192.168.20.5/24
N-A-10.8.0.99:PES1UG23CSB1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
C$>[1]
C
C
C
C
C
C
Attal
20.5
route
B2-1
A2-1
B-19
B1-1
A-10
A1-1
R-10
$>

$>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=2 ttl=127 time=60.8 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=5 ttl=127 time=65.1 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=6 ttl=127 time=473 ms
64 bytes from 150.171.22.12 (150.171.22.12): icmp_seq=7 ttl=127 time=283 ms
^C
--- ln-0002.ln-msedge.net ping statistics ---
7 packets transmitted, 4 received, 42.8571% packet loss, time 605ms
rtt min/avg/max/mdev = 60.840/220.465/473.119/171.267 ms
router-firewall:PES1UG23CS433:PranavHemanth:/
$>nslslookup 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:/
$>

```

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:/
$>nslslookup 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:

```

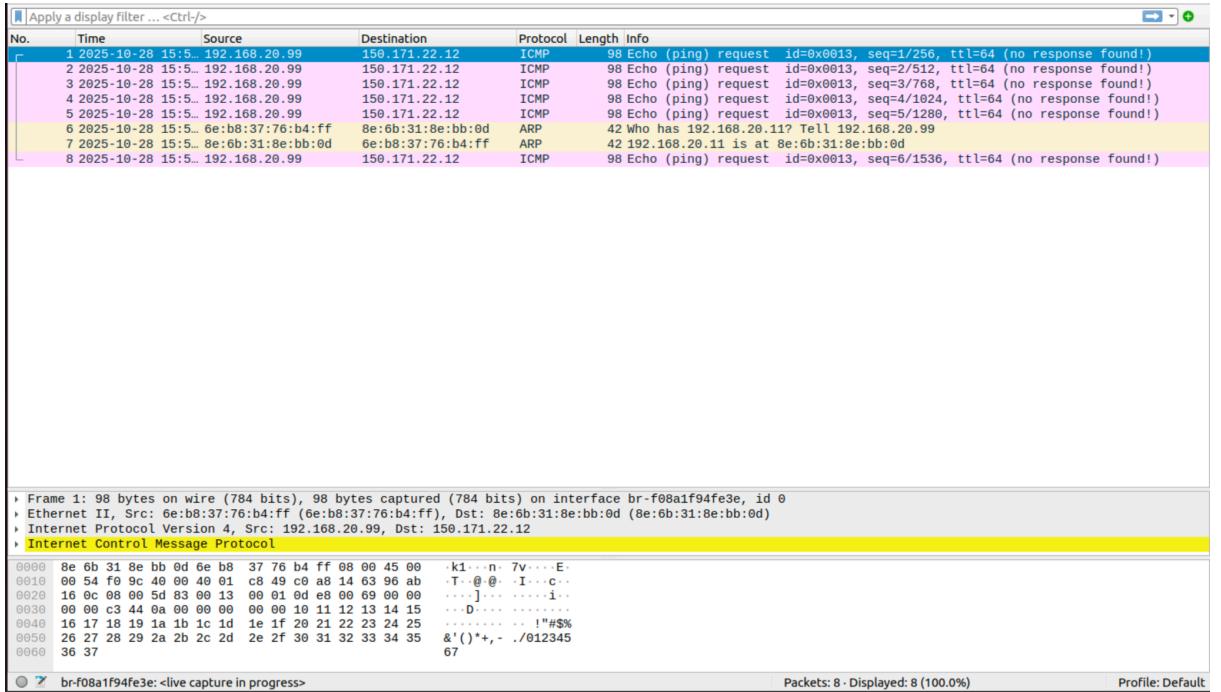
B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>ping www.linkedin.com
PING ln-0002.ln-msedge.net (150.171.22.12) 56(84) bytes of data.
^C
--- ln-0002.ln-msedge.net ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5098ms

B-192.168.20.99:PES1UG23CS433:PranavHemanth:/
$>

```

# Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

## 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)

The image shows three separate terminal windows, each titled 'seed@seedvm2004: ~'. The first window (A) shows an IP address of 127.0.0.1/8 on interface lo and 10.8.0.99/24 on interface eth0@if26. The second window (B1) shows an IP address of 127.0.0.1/8 on interface lo and 192.168.20.5/24 on interface eth0@if25. The third window (B2) shows an IP address of 127.0.0.1/8 on interface lo and 192.168.20.99/24 on interface eth0@if27.

```
A-10.8.0.99:PES1UG23CS433:PranavHemanth:/  
$>ip -br address  
lo          UNKNOWN      127.0.0.1/8 ::1/128  
eth0@if26    UP          10.8.0.99/24  
A-10.8.0.99:PES1UG23CS433:PranavHemanth:/  
$>  
  
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/  
$>ip -br address  
lo          UNKNOWN      127.0.0.1/8 ::1/128  
eth0@if25    UP          192.168.20.5/24  
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/  
$>  
  
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 do eth0 instead of eth1 as per the config of containers observed on the vm)

#### 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: 99.86.182.46  
  
router-firewall:PES1UG23CS433:PranavHemanth:/  
$>iptables -A FORWARD -i eth0 -d 99.86.182.0/24 -j DROP  
router-firewall:PES1UG23CS433:PranavHemanth:/  
$>
```

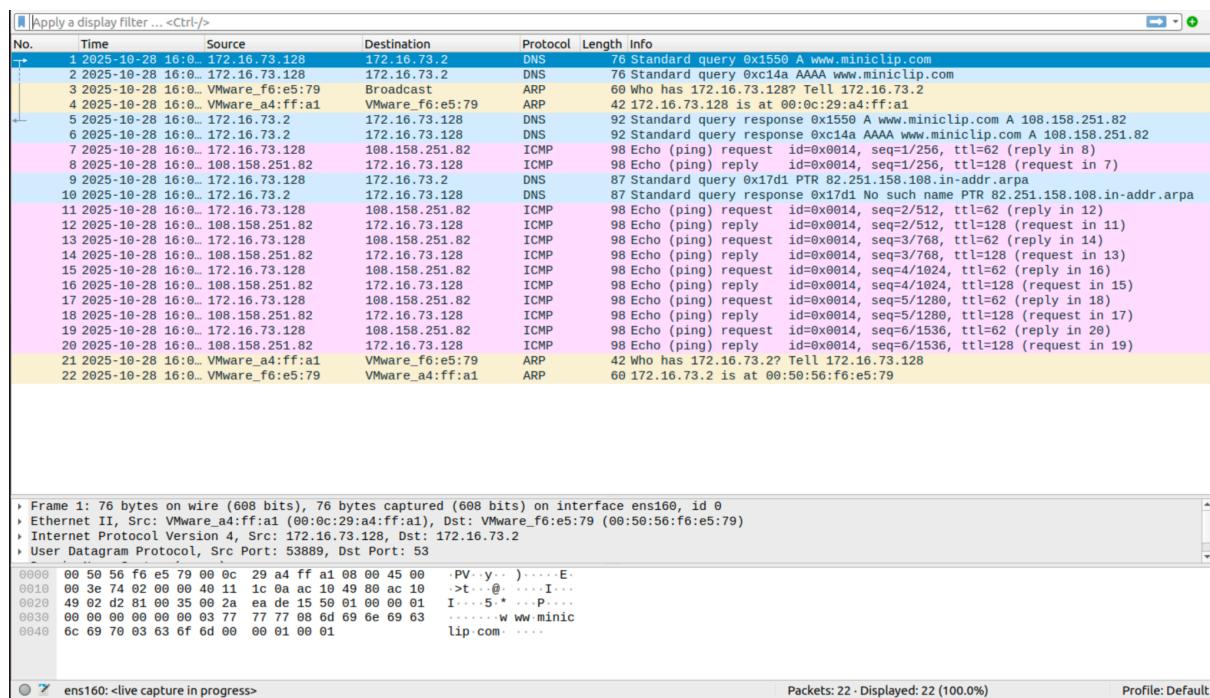
B-192.168.20.99:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=1/256, ttl=64 (no response found!)
2	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=2/512, ttl=64 (no response found!)
3	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=3/768, ttl=64 (no response found!)
4	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=4/1824, ttl=64 (no response found!)
5	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=5/1280, ttl=64 (no response found!)
6	2025-10-29 14:20:16:24:08:fa:c0:85	2a:22:68:28:41:5f	ARP	42	Who has 192.168.20.11? Tell 192.168.20.99	
7	2025-10-29 14:20:2a:22:68:28:41:5f	16:24:08:fa:c0:85	ARP	42	192.168.20.11 is at 2a:22:68:28:41:5f	
8	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=6/1536, ttl=64 (no response found!)
9	2025-10-29 14:20:19.20.99	13.227.249.5	ICMP	98	Echo (ping) request	id=0x0006, seq=7/1792, ttl=64 (no response found!)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-f08a1f94fe3e, id 0  
 > Ethernet II, Src: 16:24:08:fa:c0:85 (16:24:08:fa:c0:85), Dst: 2a:22:68:28:41:5f (2a:22:68:28:41:5f)  
 > Internet Protocol Version 4, Src: 192.168.20.99, Dst: 13.227.249.5  
 > Internet Control Message Protocol

```
0000  2a 22 68 28 41 5f 00 45 00  *"h(A_ $ ..... E.
0010  00 54 b6 81 40 00 40 01  ab 33 c0 a8 14 63 0d e3  .T-@0`- 3..c..
0020  f9 05 00 00 f8 4e 00 00  00 01 bd 24 02 69 00 00  ....N- ..$ 1..
0030  00 00 78 49 09 00 00 00  00 00 10 11 12 13 14 15  ..xI-.....
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25  .... .. !%"#
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35  &'(*+, - ./012345
0060  36 37

br-f08a1f94fe3e: <live capture in progress>
```

Packets: 9 · Displayed: 9 (100.0%)      Profile: Default

### 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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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:



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.

```
seed@seedvm2004:~$ ifconfig
br-6c250e60f50e: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.8.0.1 netmask 255.255.255.0 broadcast 10.8.0.255
        inet6 fe80::18d6:17ff:fe80:b6b7 prefixlen 64 scopeid 0x20<link>
            ether 1a:d6:17:80:b6:b7 txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

br-f08a1f94fe3e: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.20.1 netmask 255.255.255.0 broadcast 192.168.20.255
        inet6 fe80::f88a:7cff:fe80:7ea4 prefixlen 64 scopeid 0x20<link>
            ether fa:8a:7c:80:7e:a4 txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        inet6 fe80::6cb9:8dff:fea6:f690 prefixlen 64 scopeid 0x20<link>
            ether 6e:b9:8d:a6:f6:90 txqueuelen 0 (Ethernet)
            RX packets 2128 bytes 101746 (101.7 KB)
```

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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:fkGtseinXKS9urPzu8YEIMj8qz05MJzoi5ekV2x96MQ.  
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:

# Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

No.	Time	Source	Destination	Protocol	Length	Info
23	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065010 Ack=650926511 Win=64128 Len=0 TSval=365912...
24	2025-10-28 16:3...	10.8.0.99	192.168.20.99	SSHv2	126	Client: Encrypted packet (Len=60)
25	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	118	Server: Encrypted packet (Len=52)
26	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065070 Ack=650926563 Win=64128 Len=0 TSval=365912...
27	2025-10-28 16:3...	10.8.0.99	192.168.20.99	SSHv2	150	Client: Encrypted packet (Len=84)
28	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	94	Server: Encrypted packet (Len=28)
29	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065154 Ack=650926591 Win=64128 Len=0 TSval=365913...
30	2025-10-28 16:3...	10.8.0.99	192.168.20.99	SSHv2	178	Client: Encrypted packet (Len=112)
31	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	694	Server: Encrypted packet (Len=628)
32	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065266 Ack=650927219 Win=64128 Len=0 TSval=365913...
33	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	110	Server: Encrypted packet (Len=44)
34	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065266 Ack=650927263 Win=64128 Len=0 TSval=365913...
35	2025-10-28 16:3...	10.8.0.99	192.168.20.99	SSHv2	442	Client: Encrypted packet (Len=376)
36	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	174	Server: Encrypted packet (Len=108)
37	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	798	Server: Encrypted packet (Len=724)
38	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065642 Ack=650928095 Win=64128 Len=0 TSval=365913...
39	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (Len=60)
40	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065642 Ack=650928155 Win=64128 Len=0 TSval=365913...
41	2025-10-28 16:3...	10.8.0.99	192.168.20.99	SSHv2	134	Client: Encrypted packet (Len=68)
42	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (Len=60)
43	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065710 Ack=650928215 Win=64128 Len=0 TSval=365913...
44	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	134	Client: Encrypted packet (Len=68)
45	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (Len=60)
46	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065778 Ack=650928275 Win=64128 Len=0 TSval=365917...
47	2025-10-28 16:3...	10.8.0.99	192.168.20.99	SSHv2	134	Client: Encrypted packet (Len=68)
48	2025-10-28 16:3...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (Len=60)
49	2025-10-28 16:3...	10.8.0.99	192.168.20.99	TCP	66	57914 -> 22 [ACK] Seq=3560065846 Ack=650928335 Win=64128 Len=0 TSval=365917...
50	2025-10-28 16:3...	6e:b8:37:76:b4:ff	8e:b8:31:8e:bb:0d	ARP	42	Who has 192.168.20.11 Tell 192.168.20.99
51	2025-10-28 16:3...	8e:b8:31:8e:bb:0d	6e:b8:37:76:b4:ff	ARP	42	192.168.20.11 is at 8e:b8:31:8e:bb:0d

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface br-f08a1f94fe3e, id 0

Ethernet II, Src: 8e:b8:31:8e:bb:0d (8e:b8:31:8e:bb:0d), Dst: 6e:b8:37:76:b4:ff (6e:b8:37:76:b4:ff)

Internet Protocol Version 4, Src: 10.8.0.99, Dst: 192.168.20.99

Transmission Control Protocol, Src Port: 57914, Dst Port: 22, Seq: 3560063323, Len: 0

```
0000  6e b8 37 76 b4 ff 8e 6b 31 8e bb 0d 08 00 45 00  n 7v...k 1...E...
0010  00 3c 89 e6 40 00 3f 06 d2 5f 0a 08 00 63 c8 a8  .<..@?...c...
0020  14 63 e2 3a 00 16 d4 32 41 5b 00 00 00 a0 02  .c:-.-2 A[.....
0030  fa f0 df a4 00 00 02 04 b4 04 02 08 0a da 19  .....
0040  c2 af 00 00 00 00 01 03 03 07  .....
```

br-f08a1f94fe3e: <live capture in progress>

Packets: 51 · Displayed: 51 (100.0%)

Profile: Default

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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

No.	Time	Source	Destination	Protocol	Length	Info
379	2025-10-28 16:4...	10.8.0.99	192.168.20.99	SSHv2	110	Client: Encrypted packet (len=44)
380	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	66	22 → 42422 [ACK] Seq=959432460 Ack=757185151 Win=64128 Len=0 TSval=3003886...
381	2025-10-28 16:4...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (len=60)
382	2025-10-28 16:4...	192.168.20.99	192.168.20.99	TCP	92	8000 → 45256 [PSH, ACK] Seq=2826018541 Ack=3440673827 Win=65280 Len=26 TSv...
383	2025-10-28 16:4...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (len=60)
384	2025-10-28 16:4...	10.8.0.99	192.168.20.99	TCP	66	42422 → 22 [ACK] Seq=757185151 Ack=959432580 Win=64128 Len=0 TSval=3659344...
385	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	66	45256 → 8000 [ACK] Seq=3440673827 Ack=2826018567 Win=64128 Len=0 TSval=300...
386	2025-10-28 16:4...	10.8.0.99	192.168.20.99	SSHv2	134	Client: Encrypted packet (len=68)
387	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	75	45256 → 8000 [PSH, ACK] Seq=3440673827 Ack=2826018567 Win=64128 Len=9 TSva...
388	2025-10-28 16:4...	10.8.0.99	192.168.20.99	SSHv2	110	Client: Encrypted packet (len=44)
389	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	66	22 → 42422 [ACK] Seq=959432580 Ack=757185263 Win=64128 Len=0 TSval=3003939...
390	2025-10-28 16:4...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (len=60)
391	2025-10-28 16:4...	10.8.0.99	192.168.20.99	TCP	92	8000 → 45256 [PSH, ACK] Seq=2826018567 Ack=3440673836 Win=65280 Len=26 TSv...
392	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	66	45256 → 8000 [ACK] Seq=3440673836 Ack=2826018591 Win=64128 Len=0 TSval=300...
393	2025-10-28 16:4...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (len=60)
394	2025-10-28 16:4...	10.8.0.99	192.168.20.99	TCP	66	42422 → 22 [ACK] Seq=757185263 Ack=959432700 Win=64128 Len=0 TSval=3659397...
395	2025-10-28 16:4...	10.8.0.99	192.168.20.99	SSHv2	134	Client: Encrypted packet (len=68)
396	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	75	45256 → 8000 [PSH, ACK] Seq=3440673836 Ack=2826018593 Win=64128 Len=9 TSva...
397	2025-10-28 16:4...	10.8.0.99	192.168.20.99	SSHv2	110	Client: Encrypted packet (len=44)
398	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	66	22 → 42422 [ACK] Seq=959432700 Ack=757185375 Win=64128 Len=0 TSval=3003939...
399	2025-10-28 16:4...	10.8.0.99	192.168.20.99	SSHv2	126	Server: Encrypted packet (len=60)
400	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	92	8000 → 45256 [PSH, ACK] Seq=2826018593 Ack=3440673845 Win=65280 Len=26 TSv...
401	2025-10-28 16:4...	192.168.20.99	10.8.0.99	SSHv2	126	Server: Encrypted packet (len=60)
402	2025-10-28 16:4...	10.8.0.99	192.168.20.99	TCP	66	42422 → 22 [ACK] Seq=757185375 Ack=959432820 Win=64128 Len=0 TSval=3659397...
403	2025-10-28 16:4...	192.168.20.99	10.8.0.99	TCP	66	45256 → 8000 [ACK] Seq=3440673845 Ack=2826018619 Win=64128 Len=0 TSval=300...
404	2025-10-28 16:4...	6e:b8:31:8e:bb:0d	8e:b8:31:8e:bb:0d	ARP	42	Who has 192.168.20.11? Tell 192.168.20.11
405	2025-10-28 16:4...	8e:b8:31:8e:bb:0d	6e:b8:37:76:b4:ff	ARP	42	Who has 192.168.20.99? Tell 192.168.20.11
406	2025-10-28 16:4...	8e:b8:31:8e:bb:0d	6e:b8:37:76:b4:ff	ARP	42	192.168.20.11 is at 8e:b8:31:8e:bb:0d
407	2025-10-28 16:4...	8e:b8:31:8e:bb:0d	8e:b8:31:8e:bb:0d	ARP	42	192.168.20.99 is at 6e:b8:37:76:b4:ff
Frame 1: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface br-f08a1f94fe3e, id 0						
Ethernet II, Src: 8e:b8:31:8e:bb:0d (8e:b8:31:8e:bb:0d), Dst: 6e:b8:37:76:b4:ff (6e:b8:37:76:b4:ff)						
Internet Protocol Version 4, Src: 10.8.0.99, Dst: 192.168.20.99						
Transmission Control Protocol, Src Port: 57914, Dst Port: 22, Seq: 3560065846, Ack: 650928335, Len: 36						
0000	0e b8 37 76 b4 ff 8e 6b	31 8e bb 0d 08 45 18	n.7v..k 1....E.			
0010	00 58 8a 01 40 09 3f 06	d2 18 0a 00 00 63 c0 a8	.X...@?.....			
0020	14 63 e2 3a 00 16 d4 32	4b 36 26 cc 60 cf 80 18	.c:...2 K&:...			
0030	61 f5 df c0 00 00 01 01	08 0a da 1b 37 2a b3 09	.....7*..			
0040	25 aa a8 8e ac 94 d2 3d	99 ee 88 58 39 0c 0e b5	%.....V0,n			
0050	53 4a 6c 28 68 4f 99 2c	d1 2e 6f ea a0 00 a4 42	SJ1(ho., .o...n			
0060	3e fa fd b6 fe f6		>.....			

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.
A-10.8.0.99:PES1UG23CS433:PranavHemanth:/
$>
```

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:

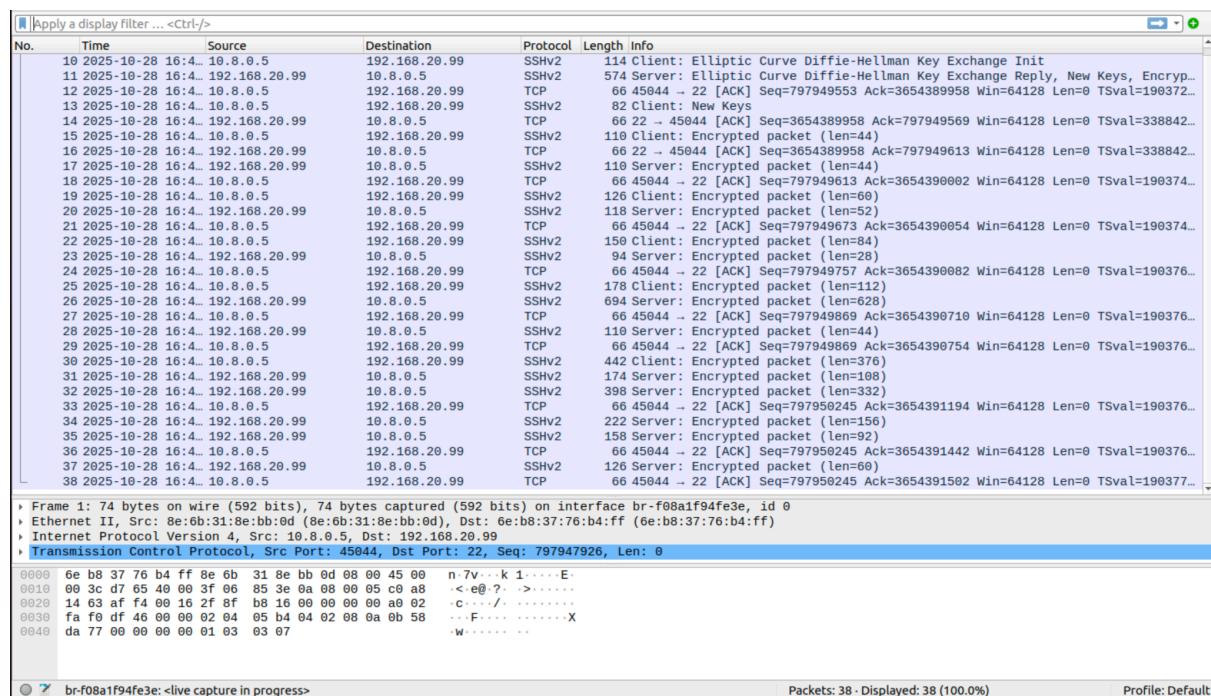
19)Command: ssh -L 0.0.0.0:8000:192.168.20.99:23 root@192.168.20.99

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:fkGtseinXKS9urPzu8YEIMj8qz05MJzo15ekV2x96MQ.  
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

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

No.	Time	Source	Destination	Protocol	Length	Info															
314	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
315	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	66	45844 -> 22 [ACK] Seq=797952553 Ack=3654397570 Win=501 Len=0 TSval=19048322...															
316	2025-10-28 16:4...	10.8.0.5	192.168.20.99	SSH	134	Client: Encrypted packet (len=68)															
317	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	75	40650 -> 8000 [PSH, ACK] Seq=176659004 Ack=1314149329 Win=64128 Len=9 TSval...															
318	2025-10-28 16:4...	10.8.0.5	192.168.20.99	SSH	110	Client: Encrypted packet (len=44)															
319	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	66	22 -> 45844 [ACK] Seq=3654397570 Ack=797952665 Win=501 Len=0 TSval=33885294...															
320	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
321	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	92	8000 -> 40650 [PSH, ACK] Seq=1314149329 Ack=176659013 Win=65280 Len=26 TSva...															
322	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
323	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	66	45844 -> 22 [ACK] Seq=797952665 Ack=3654397690 Win=501 Len=0 TSval=19048323...															
324	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	66	40650 -> 8000 [ACK] Seq=176659013 Ack=1314149355 Win=64128 Len=0 TSval=3388...															
325	2025-10-28 16:4...	10.8.0.5	192.168.20.99	SSH	134	Client: Encrypted packet (len=68)															
326	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	75	40650 -> 8000 [PSH, ACK] Seq=176659013 Ack=1314149355 Win=64128 Len=9 TSval...															
327	2025-10-28 16:4...	10.8.0.5	192.168.20.99	SSH	110	Client: Encrypted packet (len=44)															
328	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	66	22 -> 45844 [ACK] Seq=3654397690 Ack=797952777 Win=501 Len=0 TSval=33885508...															
329	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
330	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	92	8000 -> 40650 [PSH, ACK] Seq=176659022 Win=65280 Len=26 TSva...															
331	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	66	40650 -> 8000 [ACK] Seq=176659022 Ack=1314149381 Win=64128 Len=0 TSval=3388...															
332	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
333	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	66	45844 -> 22 [ACK] Seq=797952777 Ack=3654397810 Win=501 Len=0 TSval=19050461...															
334	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	134	Client: Encrypted packet (len=68)															
335	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	75	40650 -> 8000 [PSH, ACK] Seq=176659022 Ack=1314149381 Win=64128 Len=9 TSval...															
336	2025-10-28 16:4...	10.8.0.5	192.168.20.99	SSH	110	Client: Encrypted packet (len=44)															
337	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	66	22 -> 45844 [ACK] Seq=3654397810 Ack=797952889 Win=501 Len=0 TSval=33885508...															
338	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
339	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	92	8000 -> 40650 [PSH, ACK] Seq=176659031 Win=65280 Len=26 TSva...															
340	2025-10-28 16:4...	192.168.20.99	10.8.0.5	SSH	126	Server: Encrypted packet (len=60)															
341	2025-10-28 16:4...	10.8.0.5	192.168.20.99	TCP	66	45844 -> 22 [ACK] Seq=797952889 Ack=3654397930 Win=501 Len=0 TSval=19050461...															
342	2025-10-28 16:4...	192.168.20.99	10.8.0.5	TCP	66	40650 -> 8000 [ACK] Seq=176659031 Ack=1314149407 Win=64128 Len=0 TSval=3388...															
Frame 1: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface br-f08a1f94fe3e, id 0																					
Ethernet II, Src: 8e:6b:31:8e:bb:0d (8e:6b:31:8e:bb:0d), Dst: 6e:08:37:76:b4:ff (6e:08:37:76:b4:ff)																					
Internet Protocol Version 4, Src: 10.8.0.5, Dst: 192.168.20.99																					
Transmission Control Protocol, Src Port: 45044, Dst Port: 22, Seq: 797950245, Ack: 3654391502, Len: 60																					
0000	0e	b8	37	76	b4	ff	8e	bb	6d	08	00	45	16	n	7v	-k	1	....	E		
0010	00	70	d7	7b	40	00	3f	06	84	e4	0a	08	00	05	c9	a8	-p	(0?	....		
0020	14	63	af	f4	00	16	2f	8f	c1	25	d9	d1	96	ce	80	18	-c	....	/	%	....
0030	01	f5	df	7a	00	00	01	01	08	0a	0b	59	e3	f5	c9	f7	-z	....	Y	....	
0040	3c	2b	91	2d	ef	bc	e2	c2	08	78	4d	f9	a1	40	<+	*	....	xL	@	....	
0050	cd	a0	f2	14	ba	98	27	97	c8	ac	83	77	ee	22	81	f2	-w	....	W	....	
0060	66	81	f1	8c	d8	7e	f1	e5	72	d7	f9	5e	bf	eb	6a	32	f	....	r	....	j2

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:/
$>■
```

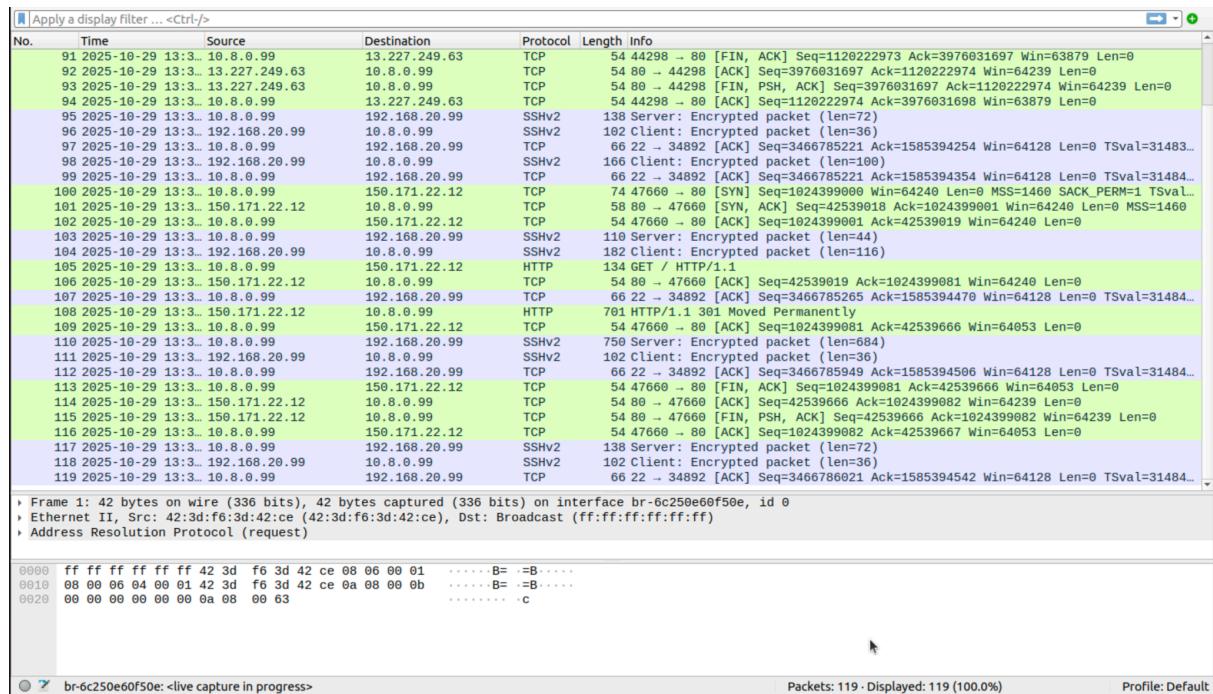
Wireshark:



## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:system-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 documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn 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

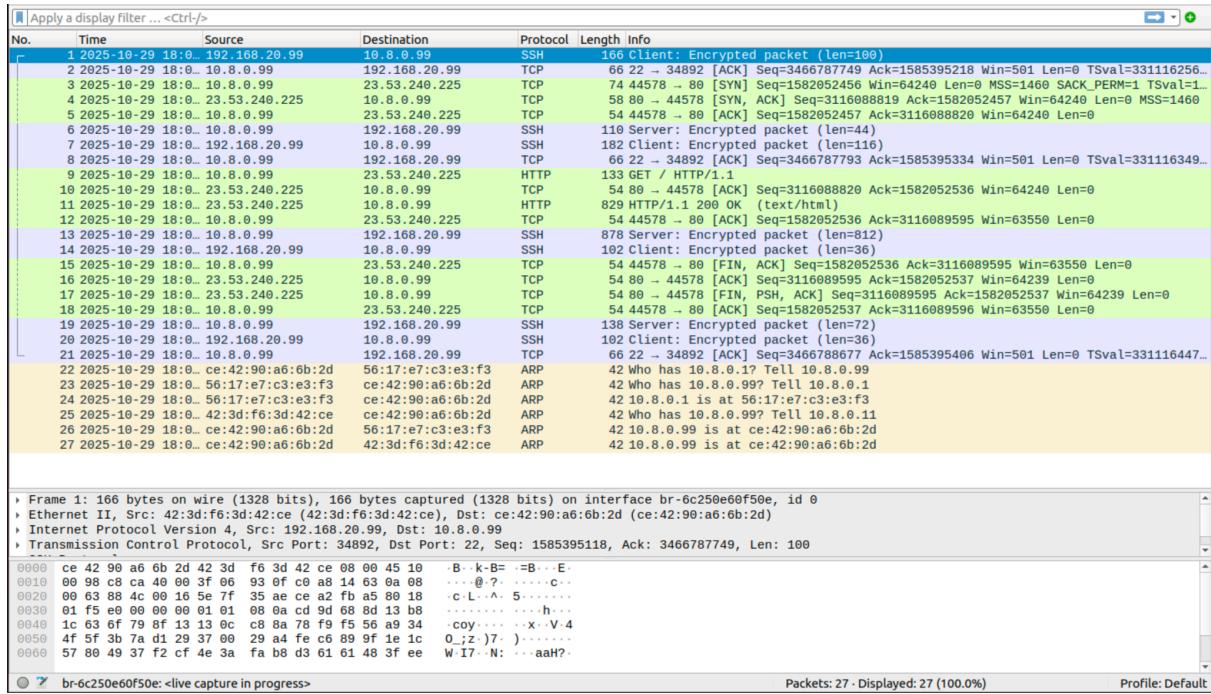
26)Command: curl -x socks5h://192.168.20.99:8000 http://www.example.com

B1-192.168.20.5:

# Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
B1-192.168.20.5:PES1UG23CS433:PranavHemanth:/
$>curl -x socks5h://192.168.20.99:8000 http://www.example.com
<!doctype html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:system-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 documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn 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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

No.	Time	Source	Destination	Protocol	Length	Info
4	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	74 8000	→ 51574	[SYN, ACK] Seq=4276151823 Ack=784013288 Win=65160 Len=0 MSS=1...
5	2025-10-29 18:0... 192.168.20.5	192.168.20.99	TCP	66 51574	→ 8000	[ACK] Seq=784013208 Ack=4276151824 Win=64256 Len=0 TSval=7451...
6	2025-10-29 18:0... 192.168.20.5	192.168.20.99	TCP	70 51574	→ 8000	[PSH, ACK] Seq=784013208 Ack=4276151824 Win=64256 Len=4 TSval...
7	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	66 8000	→ 51574	[ACK] Seq=4276151824 Ack=784013212 Win=65280 Len=0 TSval=2506...
8	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	68 8000	→ 51574	[PSH, ACK] Seq=4276151824 Ack=784013212 Win=65280 Len=2 TSval...
9	2025-10-29 18:0... 192.168.20.5	192.168.20.99	TCP	66 51574	→ 8000	[ACK] Seq=784013212 Ack=4276151826 Win=64256 Len=0 TSval=7451...
10	2025-10-29 18:0... 192.168.20.5	192.168.20.99	TCP	88 51574	→ 8000	[PSH, ACK] Seq=784013212 Ack=4276151826 Win=64256 Len=22 TSva...
11	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	160	Client: Encrypted packet (len=100)	
12	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22	→ 34892	[ACK] Seq=3466787749 Ack=1585395218 Win=501 Len=0 TSval=3311162...
13	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	66 8000	→ 51574	[ACK] Seq=4276151826 Ack=784013234 Win=65280 Len=0 TSval=2506...
14	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	110	Server: Encrypted packet (len=44)	
15	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	70 8000	→ 51574	[PSH, ACK] Seq=4276151826 Ack=784013234 Win=65280 Len=10 TSva...
16	2025-10-29 18:0... 192.168.20.5	192.168.20.99	HTTP	145	GET / HTTP/1.1	
17	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	66 8000	→ 51574	[ACK] Seq=4276151836 Ack=784013313 Win=65280 Len=0 TSval=2506...
18	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	182	Client: Encrypted packet (len=116)	
19	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22	→ 34892	[ACK] Seq=3466787793 Ack=1585395334 Win=501 Len=0 TSval=3311163...
20	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	878	Server: Encrypted packet (len=812)	
21	2025-10-29 18:0... 192.168.20.99	192.168.20.5	HTTP	841	HTTP/1.1 200 OK (text/html)	
22	2025-10-29 18:0... 192.168.20.5	192.168.20.99	TCP	66 51574	→ 8000	[FIN, ACK] Seq=784013313 Ack=4276152611 Win=64128 Len=0 TSval...
23	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	102	Client: Encrypted packet (len=36)	
24	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	138	Server: Encrypted packet (len=72)	
25	2025-10-29 18:0... 192.168.20.99	192.168.20.5	TCP	66 8000	→ 51574	[FIN, ACK] Seq=4276152611 Ack=784013314 Win=65280 Len=0 TSval...
26	2025-10-29 18:0... 192.168.20.5	192.168.20.99	TCP	66 51574	→ 8000	[ACK] Seq=784013314 Ack=4276152612 Win=64128 Len=0 TSval=7451...
27	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	102	Client: Encrypted packet (len=36)	
28	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22	→ 34892	[ACK] Seq=3466788677 Ack=1585395406 Win=501 Len=0 TSval=3311164...
29	2025-10-29 18:0... 16:24:08:fa:c0:85	5a:7a:a6:b8:5b:a8	ARP	42	Who has 192.168.20.5? Tell 192.168.20.99	
30	2025-10-29 18:0... 5a:7a:a6:b8:5b:a8	16:24:08:fa:c0:85	ARP	42	192.168.20.5 is at 5a:7a:a6:b8:5b:a8	
31	2025-10-29 18:0... 16:24:08:fa:c0:85	2a:22:68:28:41:5f	ARP	42	Who has 192.168.20.11? Tell 192.168.20.99	
32	2025-10-29 18:0... 2a:22:68:28:41:5f	16:24:08:fa:c0:85	ARP	42	192.168.20.11 is at 2a:22:68:28:41:5f	

▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-f08a1f94fe3e, id 0  
 ▶ Ethernet II, Src: 5a:7a:a6:b8:5b:a8 (5a:7a:a6:b8:5b:a8), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 ▶ Address Resolution Protocol (request)

```

0000 ff ff ff ff ff ff 5a 7a a6 b8 5b a8 08 06 00 01 .....Zz .[.....
0010 08 00 06 04 00 01 5a 7a a6 b8 5b a8 c8 a4 05 .....Zz .[.....
0020 00 00 00 00 00 00 c0 a8 14 63 .....c
  
```

br-f08a1f94fe3e:<live capture in progress>      Packets: 32 - Displayed: 32 (100.0%)      Profile: Default

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="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:system-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 documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a></p></div></body></html>
B2-192.168.20.6:PES1UG23CS433:PranavHemanth:/
$>
  
```

Wireshark on 10.8.0.1 network:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	166 Client: Encrypted packet (len=100)	100	
2	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466788677 Ack=1585395506 Win=501 Len=0 TSval=331448697...	100	
3	2025-10-29 18:0... 10.8.0.99	23.53.240.225	TCP	74 34880... 80 [SYN] Seq=2759549472 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1...	100	
4	2025-10-29 18:0... 23.53.240.225	10.8.0.99	TCP	58 80 → 34880 [SYN, ACK] Seq=3593525571 Ack=2759549473 Win=64240 Len=0 MSS=1460	100	
5	2025-10-29 18:0... 10.8.0.99	23.53.240.225	TCP	54 34880... 80 [ACK] Seq=2759549473 Ack=3593525572 Win=64240 Len=0	100	
6	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	110 Server: Encrypted packet (len=44)	44	
7	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	182 Client: Encrypted packet (len=116)	116	
8	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466788721 Ack=1585395622 Win=501 Len=0 TSval=331449207...	100	
9	2025-10-29 18:0... 10.8.0.99	23.53.240.225	HTTP	133 GET / HTTP/1.1	1	
10	2025-10-29 18:0... 23.53.240.225	10.8.0.99	TCP	54 80 → 34880 [ACK] Seq=3593525572 Ack=2759549552 Win=64240 Len=0	100	
11	2025-10-29 18:0... 23.53.240.225	10.8.0.99	HTTP	829 HTTP/1.1 200 OK (text/html)	1	
12	2025-10-29 18:0... 10.8.0.99	23.53.240.225	TCP	54 34880... 80 [ACK] Seq=2759549552 Ack=3593526347 Win=63550 Len=0	100	
13	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	878 Server: Encrypted packet (len=812)	812	
14	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	102 Client: Encrypted packet (len=36)	36	
15	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466789533 Ack=1585395658 Win=501 Len=0 TSval=331449257...	100	
16	2025-10-29 18:0... 10.8.0.99	23.53.240.225	TCP	54 34880... 80 [FIN, ACK] Seq=2759549552 Ack=3593526347 Win=63550 Len=0	100	
17	2025-10-29 18:0... 23.53.240.225	10.8.0.99	TCP	54 80 → 34880 [ACK] Seq=3593526347 Ack=2759549553 Win=64239 Len=0	100	
18	2025-10-29 18:0... 23.53.240.225	10.8.0.99	TCP	54 80 → 34880 [FIN, PSH, ACK] Seq=3593526347 Ack=2759549553 Win=64239 Len=0	100	
19	2025-10-29 18:0... 10.8.0.99	23.53.240.225	TCP	54 34880... 80 [ACK] Seq=2759549553 Ack=3593526348 Win=63550 Len=0	100	
20	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	138 Server: Encrypted packet (len=72)	72	
21	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	182 Client: Encrypted packet (len=36)	36	
22	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466789605 Ack=1585395694 Win=501 Len=0 TSval=331449331...	100	
23	2025-10-29 18:0... 56:17:e7:c3:e3:f3	ce:42:90:a6:6b:2d	ARP	42 Who has 10.8.0.99? Tell 10.8.0.1	1	
24	2025-10-29 18:0... ce:42:90:a6:6b:2d	56:17:e7:c3:e3:f3	ARP	42 10.8.0.99 is at ce:42:90:a6:6b:2d	1	

Wireshark on 192.168.20.1 network:

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-29 18:0... f6:8a:b4:36:a6:bb	Broadcast	ARP	42 Who has 192.168.20.99? Tell 192.168.20.99	6	
2	2025-10-29 18:0... 16:24:88:fa:c0:85	f6:8a:b4:36:a6:bb	ARP	42 192.168.20.99 is at 16:24:88:fa:c0:85	1	
3	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	74 60180... 8000 [SYN] Seq=400174379 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=...	100	
4	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	74 8000 → 60180 [SYN, ACK] Seq=489131853 Ack=400174380 Win=65160 Len=0 MSS=1460...	100	
5	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	66 60180... 8000 [ACK] Seq=400174380 Ack=489131854 Win=64256 Len=0 TSval=3606702...	100	
6	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	70 60180... 8000 [PSH, ACK] Seq=400174380 Ack=489131854 Win=64256 Len=4 TSval=36...	100	
7	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	68 8000 → 60180 [ACK] Seq=489131854 Ack=400174384 Win=65280 Len=0 TSval=1868609...	100	
8	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	68 8000 → 60180 [PSH, ACK] Seq=489131854 Ack=400174384 Win=65280 Len=2 TSval=18...	100	
9	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	66 60180... 8000 [ACK] Seq=400174384 Ack=489131854 Win=64256 Len=0 TSval=3606702...	100	
10	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	88 60180... 8000 [PSH, ACK] Seq=400174384 Ack=489131856 Win=64256 Len=22 TSval=3...	100	
11	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	166 Client: Encrypted packet (len=100)	100	
12	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466788677 Ack=1585395506 Win=501 Len=0 TSval=331448697...	100	
13	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	66 8000 → 60180 [ACK] Seq=489131854 Ack=400174406 Win=65280 Len=0 TSval=1868609...	100	
14	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	110 Server: Encrypted packet (len=44)	44	
15	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	76 8000 → 60180 [PSH, ACK] Seq=489131856 Ack=400174406 Win=65280 Len=10 TSval=1...	100	
16	2025-10-29 18:0... 192.168.20.6	192.168.20.99	HTTP	145 GET / HTTP/1.1	1	
17	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	66 8000 → 60180 [ACK] Seq=489131866 Ack=400174405 Win=65280 Len=0 TSval=1868614...	100	
18	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	182 Client: Encrypted packet (len=116)	116	
19	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466788721 Ack=1585395622 Win=501 Len=0 TSval=331449207...	100	
20	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	878 Server: Encrypted packet (len=612)	612	
21	2025-10-29 18:0... 192.168.20.99	192.168.20.6	HTTP	841 HTTP/1.1 200 OK (text/html)	1	
22	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	66 60180... 8000 [FIN, ACK] Seq=400174405 Ack=489132641 Win=64128 Len=0 TSval=36...	100	
23	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	102 Client: Encrypted packet (len=36)	36	
24	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466789533 Ack=1585395658 Win=501 Len=0 TSval=331449257...	100	
25	2025-10-29 18:0... 10.8.0.99	192.168.20.99	SSH	138 Server: Encrypted packet (len=72)	72	
26	2025-10-29 18:0... 192.168.20.99	192.168.20.6	TCP	66 8000 → 60180 [FIN, ACK] Seq=489132641 Ack=400174406 Win=65280 Len=0 TSval=18...	100	
27	2025-10-29 18:0... 192.168.20.6	192.168.20.99	TCP	66 60180... 8000 [ACK] Seq=400174406 Ack=489132642 Win=64128 Len=0 TSval=3606708...	100	
28	2025-10-29 18:0... 192.168.20.99	10.8.0.99	SSH	102 Client: Encrypted packet (len=36)	36	
29	2025-10-29 18:0... 10.8.0.99	192.168.20.99	TCP	66 22 → 34892 [ACK] Seq=3466789605 Ack=1585395694 Win=501 Len=0 TSval=331449331...	100	

Now lets try with [www.linkedin.com](http://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:

Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface br-6c250e60f50e, id 0

Ethernet II, Src: 42:3d:f6:3d:42:ce (42:3d:f6:3d:42:ce), Dst: ce:42:90:a6:b2:2d (ce:42:90:a6:b2:2d)

Internet Protocol Version 4, Src: 192.168.20.99, Dst: 10.8.0.99

Transmission Control Protocol, Src Port: 34892, Dst Port: 22, Seq: 1585396558, Ack: 3466792821, Len: 100

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	166	Client: Encrypted packet (len=100)
2	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466792821 Ack=1585396658 Win=501 Len=0 TSval=331817800...
3	2025-10-29 18:1...	10.8.0.99	184.18.41.41	TCP	74	54744 → 80 [SYN] Seq=2667314667 Win=64240 Len=0 MSS=1408 SACK_PERM=1 TSval=2...
4	2025-10-29 18:1...	104.18.41.41	10.8.0.99	TCP	56	80 → 54744 [SYN, ACK] Seq=3093350615 Ack=2667314668 Win=64240 Len=0 MSS=1400
5	2025-10-29 18:1...	10.8.0.99	184.18.41.41	TCP	54	54744 → 80 [ACK] Seq=2667314668 Ack=3093350616 Win=64240 Len=0
6	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	118	Server: Encrypted packet (len=44)
7	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	182	Client: Encrypted packet (len=116)
8	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466792865 Ack=1585396774 Win=501 Len=0 TSval=331817915...
9	2025-10-29 18:1...	10.8.0.99	184.18.41.41	HTTP	134	GET / HTTP/1.1
10	2025-10-29 18:1...	104.18.41.41	10.8.0.99	TCP	54	80 → 54744 [ACK] Seq=3093350616 Ack=2667314748 Win=64240 Len=0
11	2025-10-29 18:1...	104.18.41.41	10.8.0.99	HTTP	973	HTTP/1.1 301 Moved Permanently
12	2025-10-29 18:1...	10.8.0.99	184.18.41.41	TCP	54	54744 → 80 [ACK] Seq=2667314748 Ack=3093351535 Win=63411 Len=0
13	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	1822	Server: Encrypted packet (len=956)
14	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	102	Client: Encrypted packet (len=36)
15	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466793821 Ack=1585396810 Win=501 Len=0 TSval=331818273...
16	2025-10-29 18:1...	10.8.0.99	184.18.41.41	TCP	54	54744 → 80 [FIN, ACK] Seq=2667314748 Ack=3093351535 Win=63411 Len=0
17	2025-10-29 18:1...	104.18.41.41	10.8.0.99	TCP	54	80 → 54744 [FIN, PSH, ACK] Seq=3093351535 Ack=2667314749 Win=64239 Len=0
18	2025-10-29 18:1...	104.18.41.41	10.8.0.99	TCP	54	54744 → 80 [ACK] Seq=2667314749 Ack=3093351536 Win=63411 Len=0
19	2025-10-29 18:1...	10.8.0.99	184.18.41.41	TCP	54	54744 → 80 [ACK] Seq=2667314749 Ack=3093351536 Win=63411 Len=0
20	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	138	Server: Encrypted packet (len=72)
21	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	102	Client: Encrypted packet (len=36)
22	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466793893 Ack=1585396846 Win=501 Len=0 TSval=331818324...
23	2025-10-29 18:1...	ce:42:90:a6:b2:2d	ce:42:90:a6:b2:2d	ARP	42	Who has 10.8.0.99? Tell 10.8.0.1
24	2025-10-29 18:1...	ce:42:90:a6:b2:2d	56:17:e7:c3:e3:f3	ARP	42	10.8.0.99 is at ce:42:90:a6:b2:2d

Packets: 24 · Displayed: 24 (100.0%) Profile: Default

### Wireshark on 192.168.20.1 network:

Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface br-f08a1f94fe3e, id 0

Ethernet II, Src: 16:24:08:fa:c0:85 (16:24:08:fa:c0:85), Dst: 2a:22:68:28:41:5f (2a:22:68:28:41:5f)

Internet Protocol Version 4, Src: 192.168.20.99, Dst: 10.8.0.99

Transmission Control Protocol, Src Port: 34892, Dst Port: 22, Seq: 1585396558, Ack: 3466792821, Len: 100

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	166	Client: Encrypted packet (len=100)
2	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466792821 Ack=1585396658 Win=501 Len=0 TSval=331817800...
3	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	118	Server: Encrypted packet (len=44)
4	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	182	Client: Encrypted packet (len=116)
5	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466792865 Ack=1585396774 Win=501 Len=0 TSval=331817915...
6	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	1822	Server: Encrypted packet (len=956)
7	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	102	Client: Encrypted packet (len=36)
8	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466793821 Ack=1585396810 Win=501 Len=0 TSval=331818273...
9	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	138	Server: Encrypted packet (len=72)
10	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	102	Client: Encrypted packet (len=36)
11	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466793893 Ack=1585396846 Win=501 Len=0 TSval=331818324...

Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface br-f08a1f94fe3e, id 0

Ethernet II, Src: 16:24:08:fa:c0:85 (16:24:08:fa:c0:85), Dst: 2a:22:68:28:41:5f (2a:22:68:28:41:5f)

Internet Protocol Version 4, Src: 192.168.20.99, Dst: 10.8.0.99

Transmission Control Protocol, Src Port: 34892, Dst Port: 22, Seq: 1585396558, Ack: 3466792821, Len: 100

No.	Time	Source	Destination	Protocol	Length	Info
0000	2a 22 68 28 41 5f	16 24 08 fa c0 85	08 fa c0 85 08 00 45 10	**(A..._S ..... E...		
0018	00 98 c8 de 40 00 40 00 91	fb c0 a8 14 63 0a 08	..@_@.....c...			
0028	00 63 88 4c 00 16 5e 7f	3b 4e ce a3 0f 75 80 18	.c.L..^ ..N..u..			
0038	01 f5 e0 00 00 00 01 01	08 0a cd a1 1c f5 13 c4	.....			
0048	c4 00 76 2a f7 36 88 79	d3 65 68 17 52 c0 61 93	:v*.6 y ..eh.R.a...			
0050	17 89 50 22 c9 c7 bc 5b	b3 07 b2 05 58 b1 df	..P". ..[ ...XX...			
0060	dc a2 21 3b 75 58 5f 68	96 10 23 da 52 29 b7 b6	..!;uX_h ..# R) ..			

Packets: 11 · Displayed: 11 (100.0%) Profile: Default

# Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

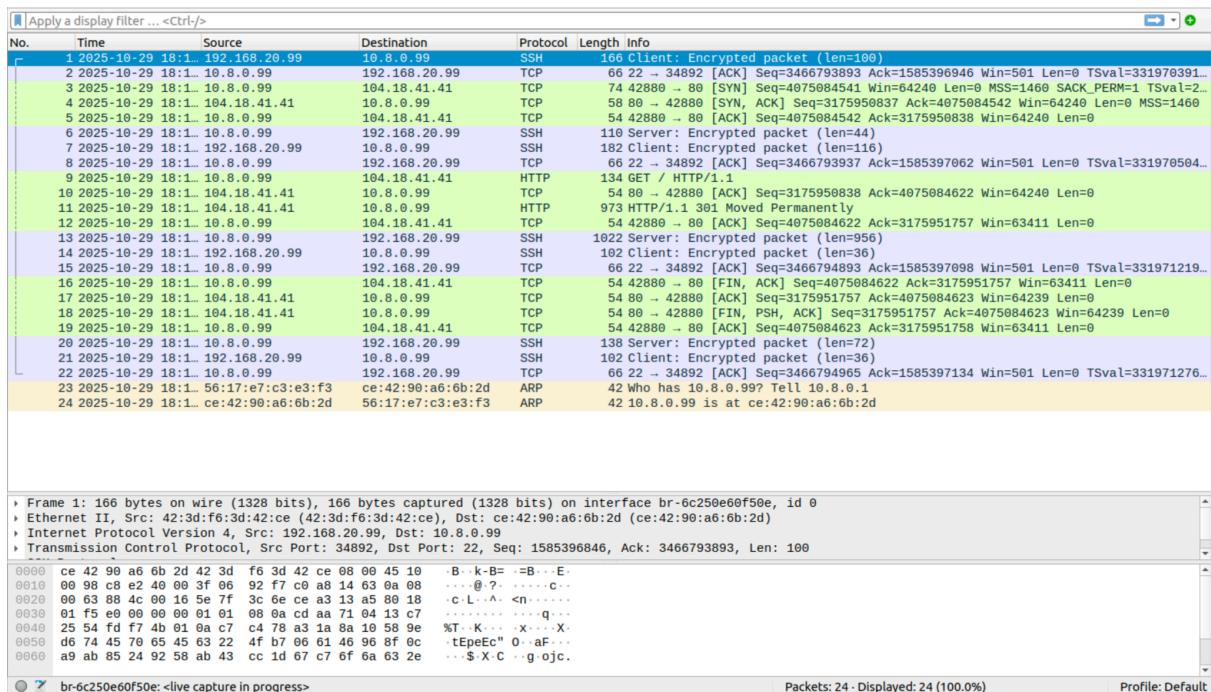
Step3: Now run the below on container B1 and B2

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

No.	Time	Source	Destination	Protocol	Length	Info											
1	2025-10-29 18:1...	192.168.20.99	192.168.20.6	TCP	74	42960 → 8000 [SYN] Seq=1347634965 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=...											
2	2025-10-29 18:1...	192.168.20.99	192.168.20.6	TCP	74	8000 → 42960 [SYN, ACK] Seq=3462966333 Ack=1347634966 Win=65160 Len=0 MSS=14...											
3	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	66	42960 → 8000 [ACK] Seq=1347634966 Ack=3462966334 Win=64256 Len=0 TSval=36125...											
4	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	70	42960 → 8000 [PSH, ACK] Seq=1347634966 Ack=3462966334 Win=64256 Len=4 TSval=...											
5	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	66	8000 → 42960 [ACK] Seq=3462966334 Ack=1347634970 Win=65280 Len=0 TSval=18744...											
6	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	68	8000 → 42960 [PSH, ACK] Seq=3462966334 Ack=1347634970 Win=65280 Len=2 TSval=...											
7	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	66	42960 → 8000 [ACK] Seq=1347634970 Ack=3462966336 Win=64256 Len=0 TSval=36125...											
8	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	89	42960 → 8000 [PSH, ACK] Seq=1347634976 Ack=3462966336 Win=64256 Len=23 TSval=...											
9	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	166	Client: Encrypted packet (len=100)											
10	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466794965 Ack=1585397234 Win=501 Len=0 TSval=332036789...											
11	2025-10-29 18:1...	192.168.20.99	192.168.20.6	TCP	66	8000 → 42960 [ACK] Seq=3462966336 Ack=1347634993 Win=65280 Len=0 TSval=18744...											
12	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	118	Server: Encrypted packet (len=44)											
13	2025-10-29 18:1...	192.168.20.99	192.168.20.6	TCP	76	8000 → 42960 [PSH, ACK] Seq=3462966336 Ack=1347634993 Win=65280 Len=10 TSval=...											
14	2025-10-29 18:1...	192.168.20.6	192.168.20.99	HTTP	146	GET / HTTP/1.1											
15	2025-10-29 18:1...	192.168.20.6	192.168.20.6	TCP	66	8000 → 42960 [ACK] Seq=3462966346 Ack=1347635073 Win=65280 Len=0 TSval=18744...											
16	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	182	Client: Encrypted packet (len=116)											
17	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466795009 Ack=1585397350 Win=501 Len=0 TSval=332036893...											
18	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	1822	Server: Encrypted packet (len=72)											
19	2025-10-29 18:1...	192.168.20.99	192.168.20.6	HTTP	985	HTTP/1.1 301 Moved Permanently											
20	2025-10-29 18:1...	192.168.20.99	192.168.20.99	TCP	66	42960 → 8000 [FIN, ACK] Seq=13467635073 Ack=3462967265 Win=64128 Len=0 TSval=...											
21	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	182	Client: Encrypted packet (len=36)											
22	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466795065 Ack=1585397386 Win=501 Len=0 TSval=332037249...											
23	2025-10-29 18:1...	192.168.20.99	192.168.20.6	TCP	66	8000 → 42960 [ACK] Seq=3462967265 Ack=1347635074 Win=65280 Len=0 TSval=18744...											
24	2025-10-29 18:1...	10.8.0.99	192.168.20.99	SSH	138	Server: Encrypted packet (len=72)											
25	2025-10-29 18:1...	192.168.20.99	192.168.20.6	TCP	66	8000 → 42960 [FIN, ACK] Seq=13467635074 Ack=1347635074 Win=65280 Len=0 TSval=...											
26	2025-10-29 18:1...	192.168.20.6	192.168.20.99	TCP	66	42960 → 8000 [ACK] Seq=1347635074 Ack=3462967265 Win=64128 Len=0 TSval=36125...											
27	2025-10-29 18:1...	192.168.20.99	10.8.0.99	SSH	102	Client: Encrypted packet (len=36)											
28	2025-10-29 18:1...	10.8.0.99	192.168.20.99	TCP	66	22 → 34892 [ACK] Seq=3466796037 Ack=1585397422 Win=501 Len=0 TSval=332037304...											
▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface br-f08a1f94fe3e, id 0																	
Ethernet II, Src: f0:8a:b4:36:a6:bb (f0:8a:b4:36:a6:bb), Dst: 16:24:08:fa:c0:85 (16:24:08:fa:c0:85)																	
Internet Protocol Version 4, Src: 192.168.20.6, Dst: 192.168.20.99																	
Transmission Control Protocol, Src Port: 42960, Dst Port: 8000, Seq: 1347634965, Len: 0																	
0000	16	24	08	fa	c9	85	f6	8a	b4	36	a6	bb	00	45	00	\$ ..... 6 ..... E	
0010	00	3c	b4	16	40	00	40	00	dc	eb	c9	a8	14	06	c8	a8	< . @ @ .
0020	14	63	a7	d0	1f	48	50	53	47	15	00	00	00	00	a0	02	.c .. @PS G.....
0030	fa	f0	a9	e8	00	00	02	04	b5	b4	04	02	08	00	15	88	]D .....
0040	5d	44	00	00	00	00	01	03	03	07							

br-f08a1f94fe3e: <live capture in progress>

Packets: 28 · Displayed: 28 (100.0%)

Profile: Default

### 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.

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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 -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:

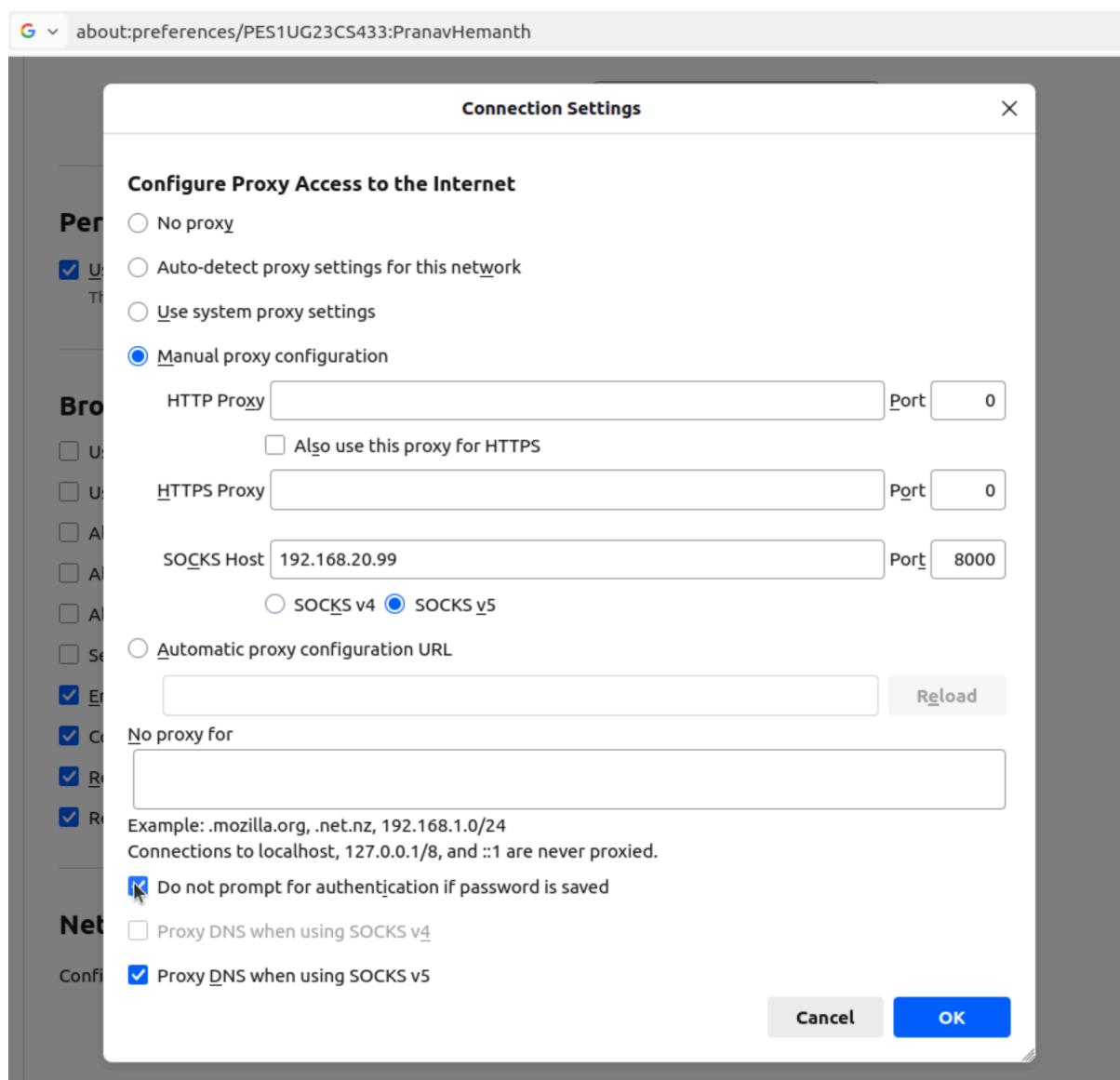
- Traffic between B and A is encrypted SSH on port 22.
- Outbound connections to external servers originate from A's IP, not from B or B1/B2.

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

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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

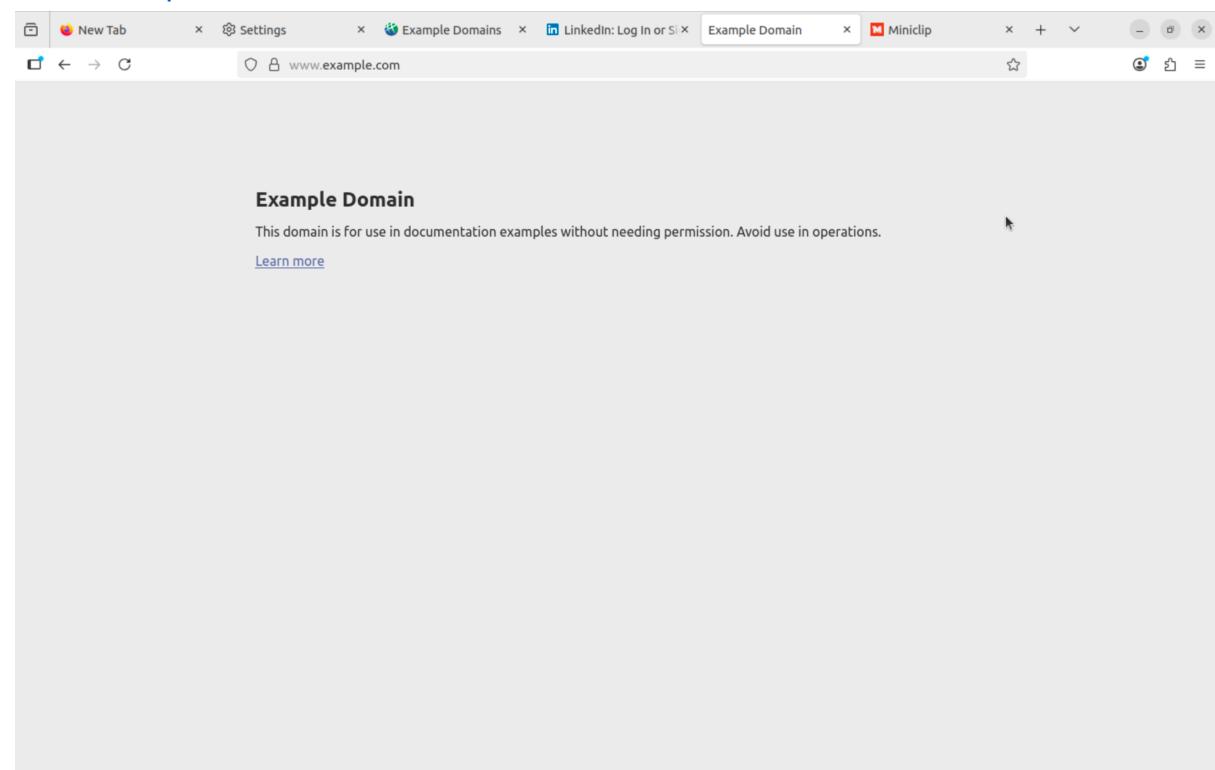
Please note, we are still using the ssh connection established in the previous Task 2.1.

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:

```
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:/  
$>
```

[www.example.com](http://www.example.com):



# Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

[www.linkedin.com:](http://www.linkedin.com)

The screenshot shows the LinkedIn homepage. At the top, there are several navigation links: Top Content, People, Learning, Jobs, and Games. A prominent "Agree & Join LinkedIn" button is visible, along with a link to "User Agreement", "Privacy Policy", and "Cookie Policy". Below this, there's a "Sign in with Google" button and a "Sign in with email" button. A large, stylized illustration of a person sitting at a desk with a laptop, surrounded by books and papers, serves as the background. A tooltip for the "Sign in with Google" button explains that it allows users to sign in using their Google account, noting that no more passwords are required. Below the sign-in area, there's a section titled "Explore top LinkedIn content" with categories like Career, Productivity, Finance, Soft Skills & Emotional Intelligence, and Project Management.

[www.miniclip.com:](http://www.miniclip.com)

The screenshot shows the Miniclip homepage. The header features the Miniclip logo and navigation links for Publishing, Our Story, Games, Web Games, Careers, and Support. The main banner is a vibrant purple and blue design featuring a pool table and billiard balls. It prominently displays the text "Celebrating over 190 Billion Matches played in 10 amazing years!" and "10 YEARS". A large "PLAY NOW" button is centered below the banner. At the bottom of the page, there's a cookie consent message asking for permission to use cookies, with options to "Personalize", "Decline", or "Accept".

router-firewall (tcpdump):

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
op,TS val 2058470970 ecr 3521767407], length 1060
17:32:32.053790 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 384192, win 2567, options [nop,nop,TS val 3521767536
ecr 2058470970], length 0
17:32:32.053842 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 384192:385324, ack 6788881, win 8807, options [nop,n
op,TS val 2058470970 ecr 3521767536], length 1132
17:32:32.053876 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 385324, win 2563, options [nop,nop,TS val 3521767536
ecr 2058470970], length 0
17:32:32.054472 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [P.], seq 6788881:6789093, ack 385324, win 2571, options [nop,
nop,TS val 3521767817 ecr 2058470970], length 212
17:32:32.054522 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [.], ack 6789093, win 8807, options [nop,nop,TS val 2058471251
ecr 3521767817], length 0
17:32:34.030096 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385324:385400, ack 6789093, win 8807, options [nop,n
op,TS val 2058473246 ecr 3521767817], length 76
17:32:34.030190 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 385400, win 2571, options [nop,nop,TS val 3521769812
ecr 2058473246], length 0
17:32:34.032560 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385400:385460, ack 6789093, win 8807, options [nop,n
op,TS val 2058473249 ecr 3521769812], length 60
17:32:34.032595 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 385460, win 2571, options [nop,nop,TS val 3521769815
ecr 2058473249], length 0
17:32:34.033112 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385460:385496, ack 6789093, win 8807, options [nop,n
op,TS val 2058473249 ecr 3521769815], length 36
17:32:34.033143 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 385496, win 2571, options [nop,nop,TS val 3521769815
ecr 2058473249], length 0
17:32:34.039199 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [P.], seq 6789093:6789165, ack 385496, win 2571, options [nop,
nop,TS val 3521769875 ecr 2058473249], length 72
17:32:34.0393268 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [.], ack 6789165, win 8807, options [nop,nop,TS val 2058473309
ecr 3521769875], length 0
17:32:34.0394322 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385496:385532, ack 6789165, win 8807, options [nop,n
op,TS val 2058473311 ecr 3521769875], length 36
17:32:34.0394357 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 385532, win 2571, options [nop,nop,TS val 3521769877
ecr 2058473311], length 0
17:32:39.0356053 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [P.], seq 6789165:6789233, ack 385532, win 2571, options [nop,
nop,TS val 3521774838 ecr 2058473311], length 68
17:32:39.0357602 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385532:385600, ack 6789233, win 8807, options [nop,n
op,TS val 2058478274 ecr 3521774838], length 68
17:32:39.0357689 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [P.], seq 6789233:6789269, ack 385600, win 2571, options [nop,
nop,TS val 3521774840 ecr 2058478274], length 36
17:32:39.0357741 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385600:385636, ack 6789269, win 8807, options [nop,n
op,TS val 2058478274 ecr 3521774840], length 36
17:32:39.0357920 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [P.], seq 385636:385672, ack 6789269, win 8807, options [nop,n
op,TS val 2058478274 ecr 3521774840], length 36
17:32:39.0358263 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [.], ack 385672, win 2571, options [nop,nop,TS val 3521774840
ecr 2058478274], length 0
17:32:39.0358586 IP A-10.8.0.99.net->10.8.0.0.ssh > B-192.168.20.99.net-192.168.20.0.55750: Flags [P.], seq 6789269:6789305, ack 385672, win 2571, options [nop,
nop,TS val 3521774841 ecr 2058478274], length 36
17:32:39.402501 IP B-192.168.20.99.net->192.168.20.0.55750 > A-10.8.0.99.net-10.8.0.0.ssh: Flags [.], ack 6789305, win 8807, options [nop,nop,TS val 2058478319
ecr 3521774841], length 0
```

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:



## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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.

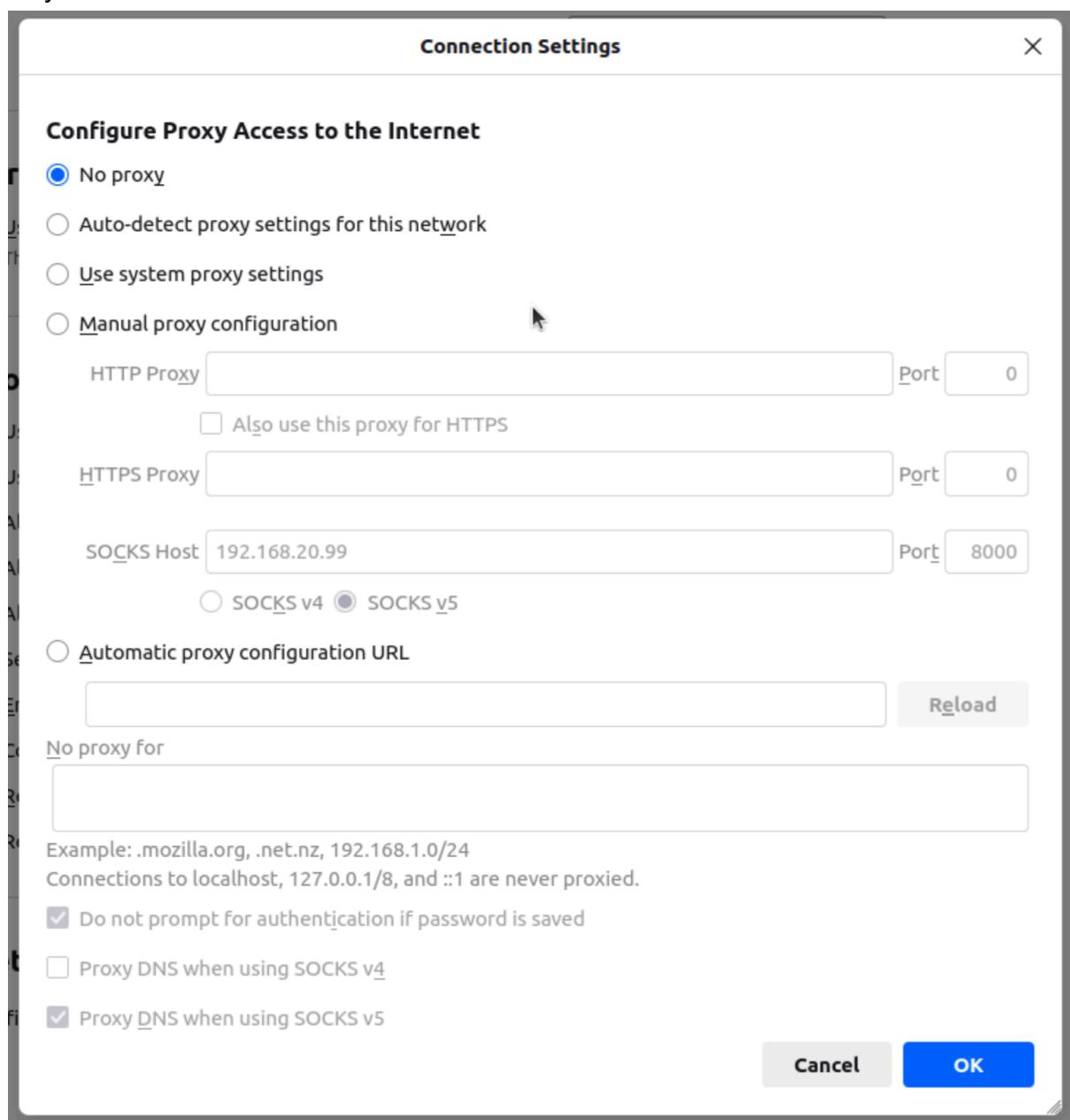
This happened because once the SSH tunnel was broken:

- 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. Host 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.

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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.

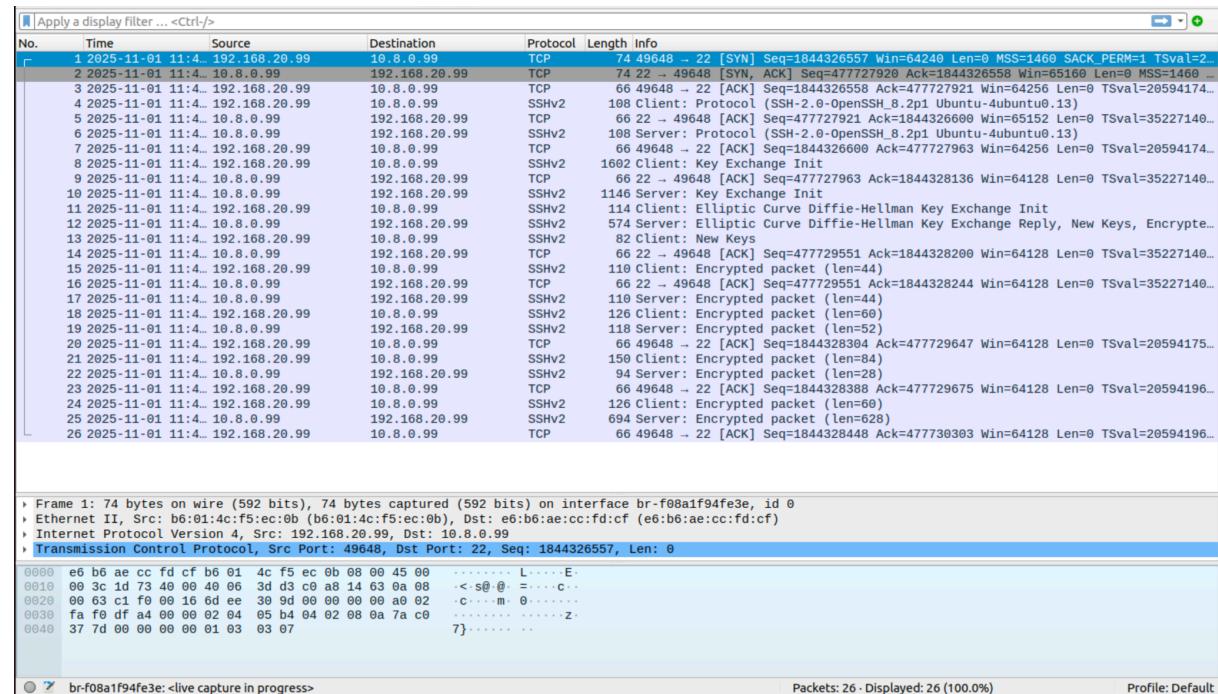
Step1: Run in container B to set up the ssh shell with dynamic port forwarding enabled:

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](#)

```
[1/2] python3
#!/bin/env python3
import socks
s = socks.socksocket()
# Set the proxy
s.set_proxy(socks.SOCKS5, "0.0.0.0", 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 14 lines ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Close ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

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 2025 05:44:51 GMT', b'Content-Length: 513', b'Via: HTTP/1.1 forward.http.proxy:3128', b'Connection: close', b'', b'<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:sans-serif}</style><body><div><h1>Example Domain</h1><p>This domain is for use in documentation examples without needing permission. Avoid use in operations.<p><a href="https://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:



```
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:fkGtseinXKS9urPzu8YEIMj8qz05MJzoI5ekV2x96MQ.
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
$>
```

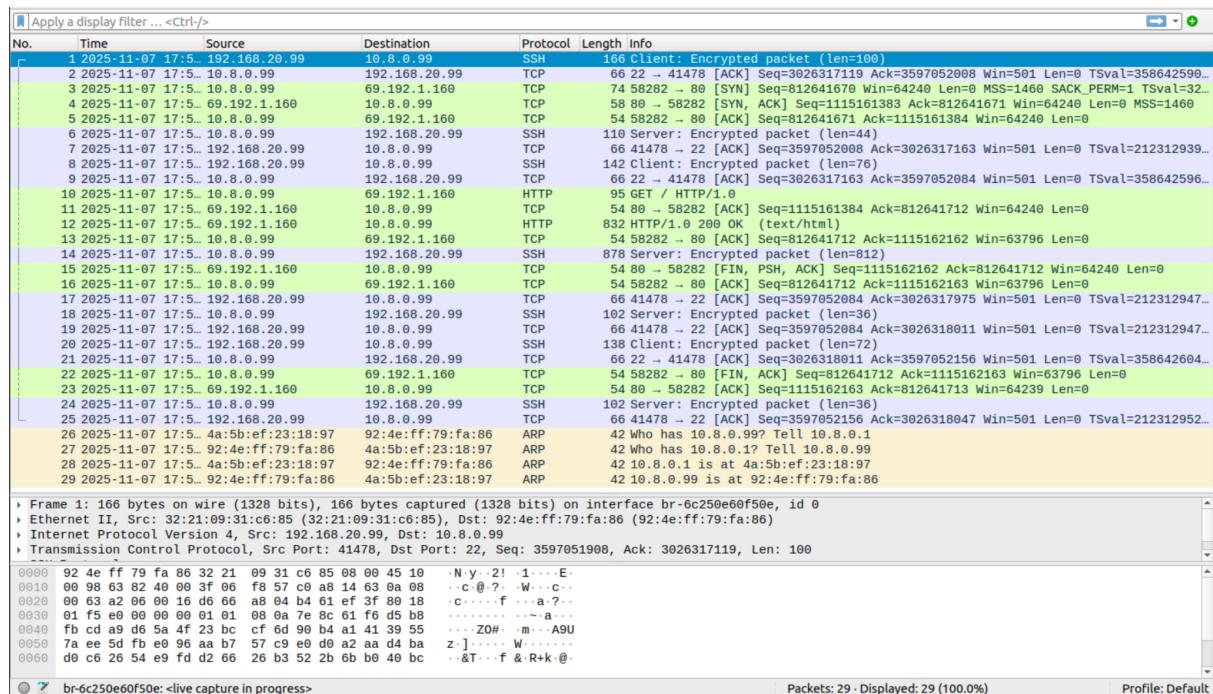
Wireshark on 10.8.0.1 network:



# Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

```
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: "bc2473a18e003bdb249eba5ce893033f: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'Content-Length: 513', b'Connection: close', b'X-N: S', b'', b'<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:system-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 documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a></div></body></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:

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

No.	Time	Source	Destination	Protocol	Length	Info
3	2025-11-07 17:5.. 192.168.20.5	192.168.20.99	TCP	66	47274 -> 8000 [ACK] Seq=1408269814 Ack=45878684 Win=64256 Len=0 TSval=37175...	
4	2025-11-07 17:5.. 192.168.20.5	192.168.20.99	TCP	69	47274 -> 8000 [PSH, ACK] Seq=1408269814 Ack=45878684 Win=64256 Len=3 TSval=...	
5	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	66	8000 -> 47274 [ACK] Seq=45878684 Ack=1408269817 Win=65280 Len=0 TSval=15355...	
6	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	68	8000 -> 47274 [PSH, ACK] Seq=45878684 Ack=1408269817 Win=65280 Len=2 TSval=...	
7	2025-11-07 17:5.. 192.168.20.99	192.168.20.99	TCP	66	47274 -> 8000 [ACK] Seq=1408269817 Ack=45878686 Win=64256 Len=0 TSval=37175...	
8	2025-11-07 17:5.. 192.168.20.99	192.168.20.99	TCP	88	47274 -> 8000 [PSH, ACK] Seq=1408269817 Ack=45878686 Win=64256 Len=22 TSval=...	
9	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	SSH	166	Client: Encrypted packet (len=108)	
10	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	TCP	66	22 -> 41478 [ACK] Seq=3026317119 Ack=3597052008 Win=501 Len=0 TSval=3586425...	
11	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	66	8000 -> 47274 [ACK] Seq=45878686 Ack=1408269839 Win=65280 Len=0 TSval=15355...	
12	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	SSH	110	Server: Encrypted packet (len=44)	
13	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	TCP	66	41478 -> 22 [ACK] Seq=1408269817 Ack=45878686 Win=64256 Len=0 TSval=37175...	
14	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	76	8000 -> 47274 [PSH, ACK] Seq=45878686 Ack=1408269839 Win=65280 Len=10 TSval=...	
15	2025-11-07 17:5.. 192.168.20.5	192.168.20.99	HTTP	107	GET / HTTP/1.0	
16	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	66	8000 -> 47274 [ACK] Seq=45878696 Ack=1408269884 Win=65280 Len=0 TSval=15355...	
17	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	SSH	142	Client: Encrypted packet (len=76)	
18	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	TCP	66	22 -> 41478 [ACK] Seq=3026317163 Ack=3597052084 Win=501 Len=0 TSval=3586425...	
19	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	SSH	878	Server: Encrypted packet (len=812)	
20	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	HTTP	844	HTTP/1.0 200 OK (text/html)	
21	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	TCP	66	41478 -> 22 [ACK] Seq=3026317975 Win=501 Len=0 TSval=2123129...	
22	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	SSH	102	Server: Encrypted packet (len=36)	
23	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	TCP	66	41478 -> 22 [ACK] Seq=3597052084 Ack=3026318011 Win=501 Len=0 TSval=2123129...	
24	2025-11-07 17:5.. 192.168.20.5	192.168.20.99	TCP	66	47274 -> 8000 [ACK] Seq=1408269880 Ack=45879474 Win=64128 Len=0 TSval=37175...	
25	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	66	8000 -> 47274 [FIN, ACK] Seq=45879474 Ack=1408269888 Win=65280 Len=0 TSval=...	
26	2025-11-07 17:5.. 192.168.20.5	192.168.20.99	TCP	66	47274 -> 8000 [FIN, ACK] Seq=1408269888 Ack=45879475 Win=64128 Len=0 TSval=...	
27	2025-11-07 17:5.. 192.168.20.99	192.168.20.5	TCP	66	8000 -> 47274 [ACK] Seq=45879475 Ack=1408269881 Win=65280 Len=0 TSval=15355...	
28	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	SSH	138	Client: Encrypted packet (len=72)	
29	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	TCP	66	22 -> 41478 [ACK] Seq=3026318011 Ack=3597052156 Win=501 Len=0 TSval=3586426...	
30	2025-11-07 17:5.. 10.8.0.99	192.168.20.99	SSH	102	Server: Encrypted packet (len=36)	
31	2025-11-07 17:5.. 192.168.20.99	10.8.0.99	TCP	66	41478 -> 22 [ACK] Seq=3597052156 Ack=3026318047 Win=501 Len=0 TSval=2123129...	
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface br-f08a1f94fe3e, id 0						
Ethernet II, Src: 52:56:a4:ea:c6:fd (52:56:a4:ea:c6:fd), Dst: b6:01:4c:f5:ec:0b (b6:01:4c:f5:ec:0b)						
Internet Protocol Version 4, Src: 192.168.20.5, Dst: 192.168.20.99						
Transmission Control Protocol, Src Port: 47274, Dst Port: 8000, Seq: 1408269813, Len: 0						
00:00	b6 01 4c f5 ec 0b 52 56 a4 ea c6 fd 08 00 45 00	-L .. -RV .. E..				
00:10	00 3d 63 17 40 00 48 00 2d ec c0 a8 14 65 c8 a8	-< @ @ ..				
00:20	14 63 b8 aa 1f 40 53 70 7d f5 00 00 00 a0 02	-c .. @S ..				
00:30	fa f0 a9 e7 00 00 02 04 b5 b4 02 08 0a dd 95	-H ..				
00:40	c3 48 00 00 00 00 01 03 03 07					
br-f08a1f94fe3e:<live capture in progress>						
Packets: 31 · Displayed: 31 (100.0%)						
Profile: Default						

### 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

## Aug -Dec 2025 Assignment SUBMISSION\_UE23CS343AB6

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 tunneled 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.