Firewall Evasion

Seed Lab

By Aneesh Nagdev

December 2, 2025

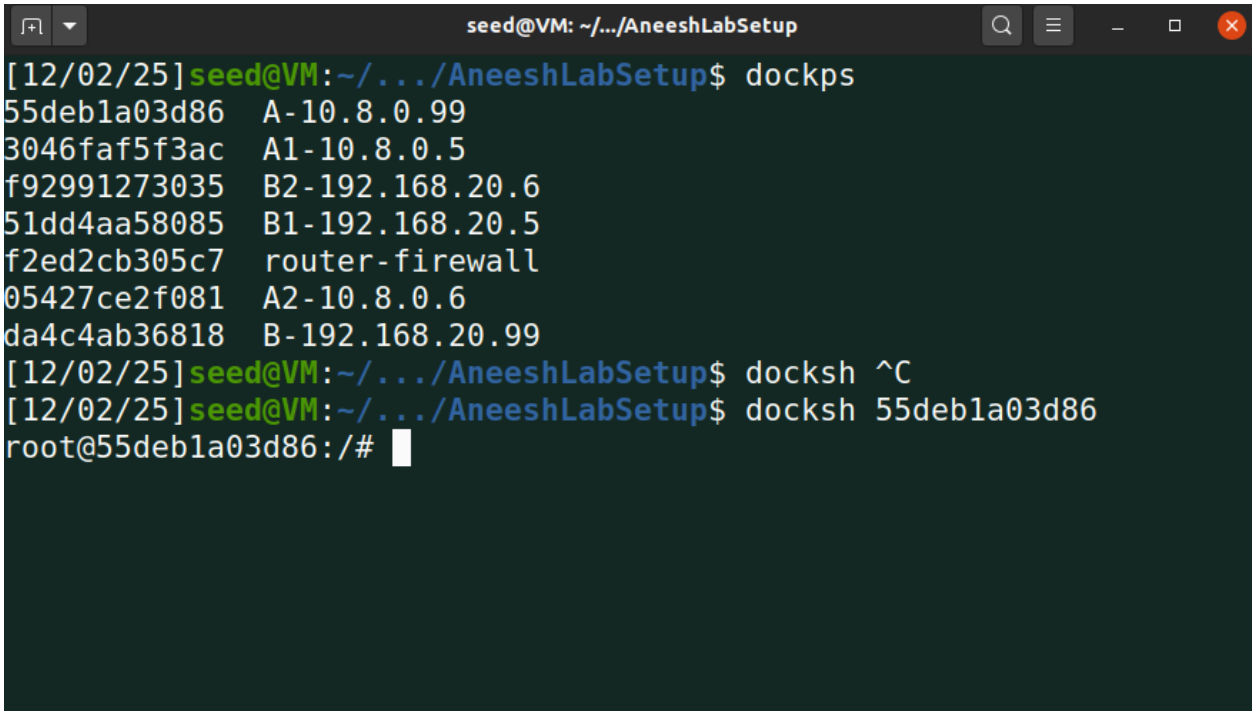
Lab setup:

Starting the container screenshot:

```
[12/02/25]seed@VM:~/.../AneeshLabSetup$ dcup
Creating network "net-10.8.0.0" with the default driver
Creating network "net-192.168.20.0" with the default driver
Building hostA
Step 1/5 : FROM handsonsecurity/seed-ubuntu:large
 ---> cecb04fbf1dd
Step 2/5 : ARG DEBIAN_FRONTEND=noninteractive
 ---> Running in f52b2cf66309
Removing intermediate container f52b2cf66309
 ---> a696ea0c3e94
Step 3/5 : RUN apt-get update     && apt-get -y install openssh-ser
ver     && rm -rf /var/lib/apt/lists/*
 ---> Running in 8c9a05660e03
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [1
28 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security/universe amd
64 Packages [1308 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/multiverse a
md64 Packages [33.1 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/restricted a
md64 Packages [4801 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 P
ackages [4432 kB]
```

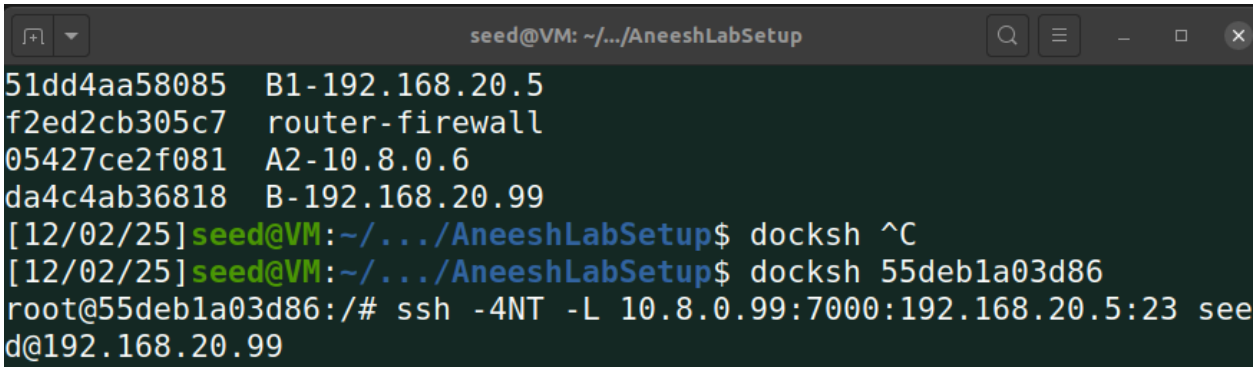

```
Creating B2-192.168.20.6 ... done
Creating A1-10.8.0.5     ... done
Attaching to B-192.168.20.99, A2-10.8.0.6, A-10.8.0.99, router-fire
wall, B2-192.168.20.6, B1-192.168.20.5, A1-10.8.0.5
A-10.8.0.99 |  * Starting internet superserver inetd              [
 OK ]
A-10.8.0.99 |  * Starting OpenBSD Secure Shell server sshd        [
 OK ]
A2-10.8.0.6 |  * Starting internet superserver inetd              [
 OK ]
B-192.168.20.99 |  * Starting internet superserver inetd          [
 OK ]
B-192.168.20.99 |  * Starting OpenBSD Secure Shell server sshd
                                                                  [
 OK ]
B2-192.168.20.6 |  * Starting internet superserver inetd          [
 OK ]
router-firewall |  * Starting internet superserver inetd          [
 OK ]
B1-192.168.20.5 |  * Starting internet superserver inetd          [
 OK ]
A1-10.8.0.5 |  * Starting internet superserver inetd              [
 OK ]
```

Connecting to host A container screenshot:



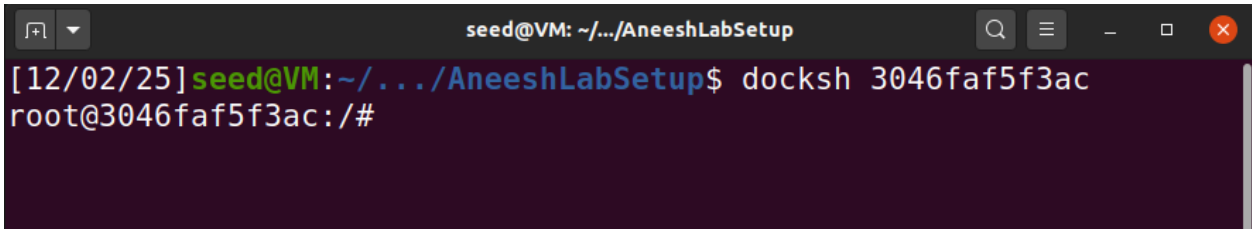
Task 1:

Command screenshot:

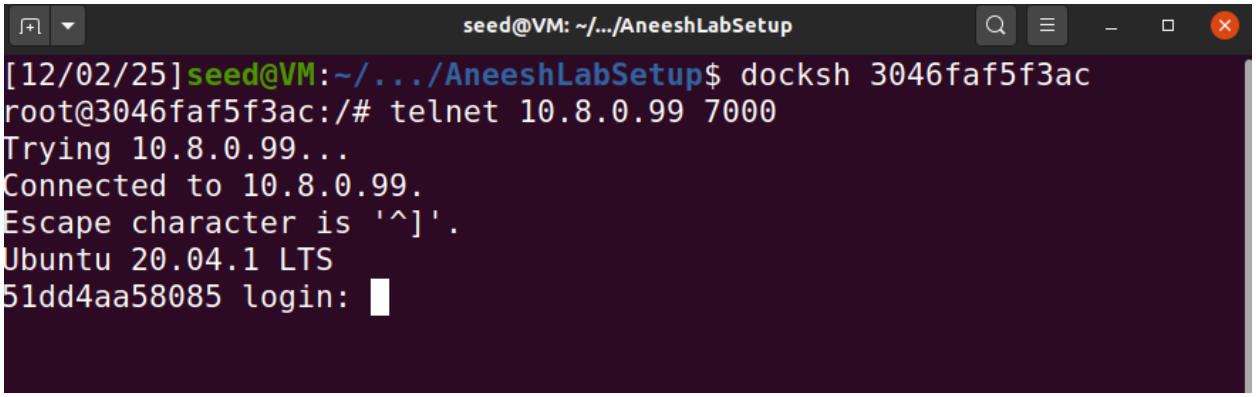


Explanation: Any host A1 or A2, if they want to telnet to host B1, they have to telnet to hostA's address or port 7000. HostA will send the packets to HostB and any packets that HostB will receive from HostA or from port 7000 will be forwarded to 192.168.20.5 address which is for HostB1 in the internet network.

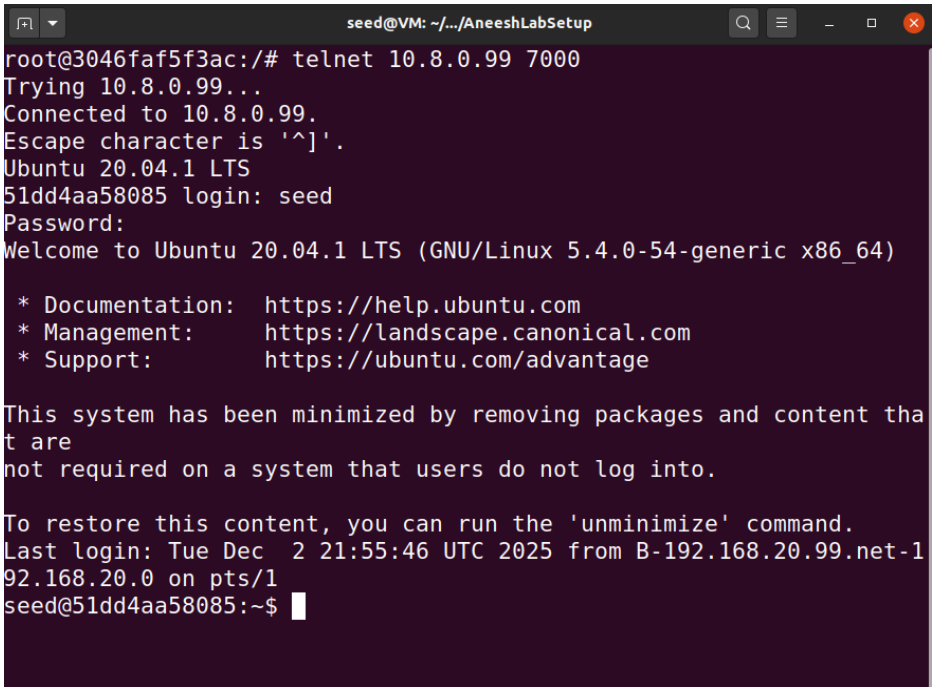
Connecting to A1 Host container screenshot:



Telnet to Host A using port 7000 screenshot:

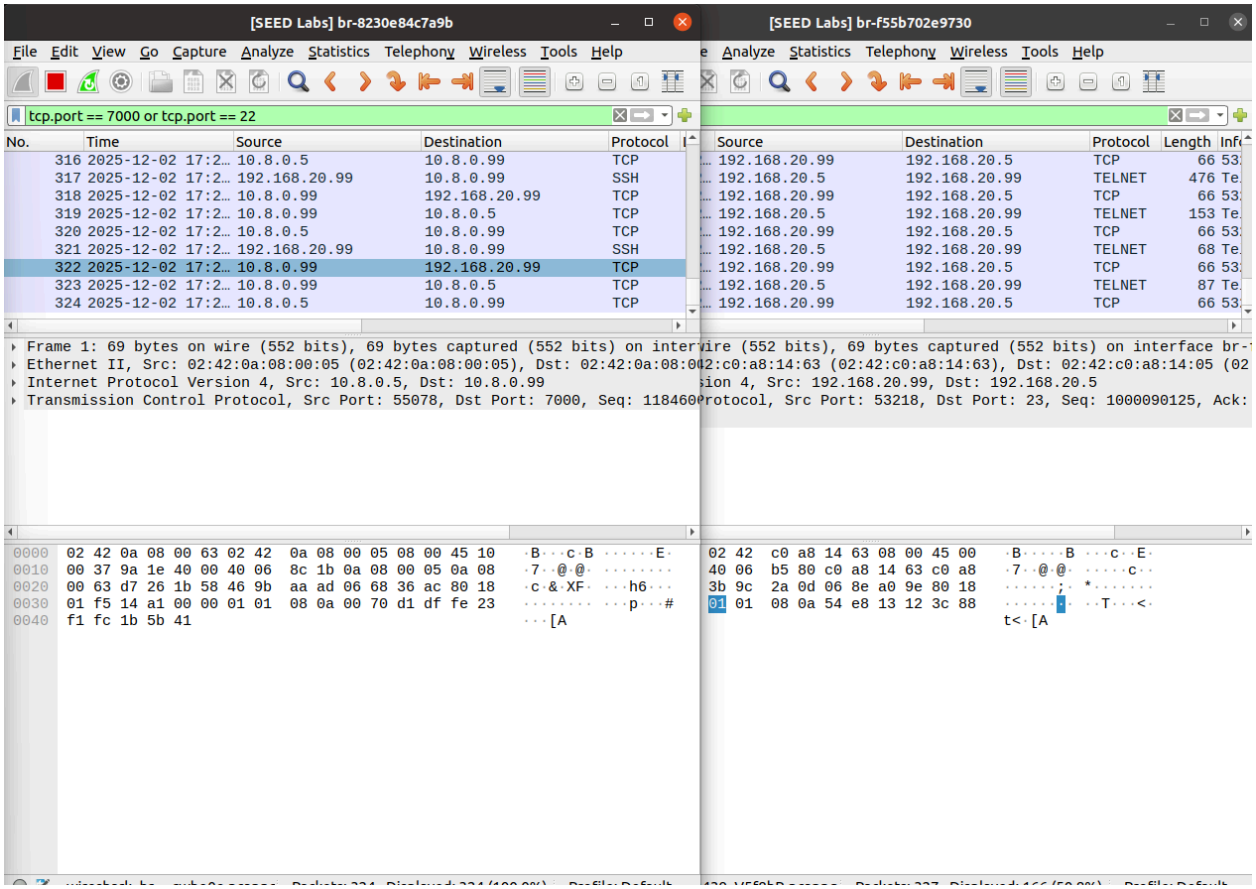


Putting the credentials screenshot:



Explanation: When I run telnet from A1 to 10.8.0.99:7000, the traffic is first sent to Host A, where SSH is listening on the forwarded port. SSH then encrypts this data and carries it inside its allowed TCP connection to Host B. Once it reaches B, SSH opens a separate internal Telnet connection to B1 (192.168.20.5:23) and delivers the traffic there. As a result, it appears to be connected directly to B1 even though the firewall would normally block this access.

Q1) How many TCP connections are involved?



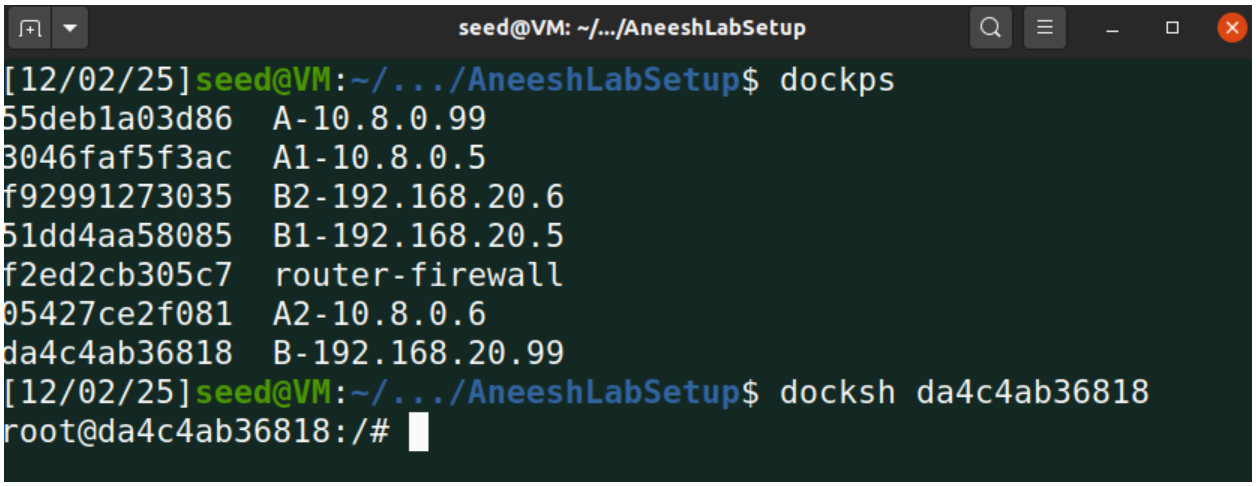
Ans) There are three TCP connections involved in the static port-forwarding process. The first connection is between the external host and Host A on port 7000. The second connection is the encrypted SSH session from Host A to Host B on port 22. The third connection is the internal Telnet connection from Host B to Host B1 on port 23.

Q2) Why can this tunnel successfully help users of the firewall rule?

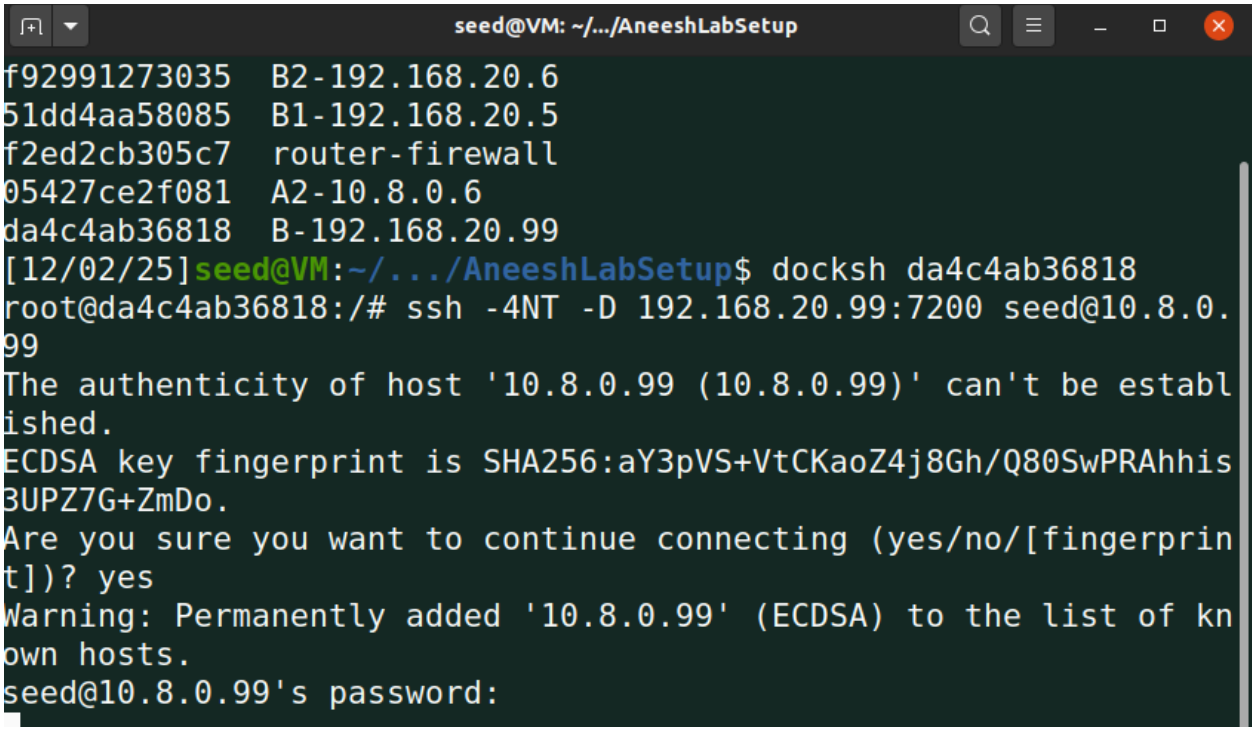
Ans) The tunnel evades the firewall because all Telnet traffic is carried inside an allowed SSh connection. The firewall only sees encrypted SSH packets going to port 22, which it permits, so it never detects or blocks the Telnet data inside the tunnel. Once the SSH connection reaches Host B, the Telnet traffic is delivered internally to B1, bypassing the firewall's restriction on direct access to internal services.

Task 2.1:

Connecting to Host B container Screenshot:

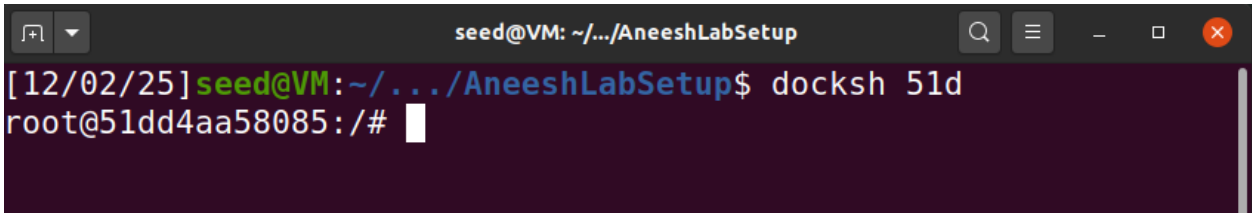


Running the command in Host B container:

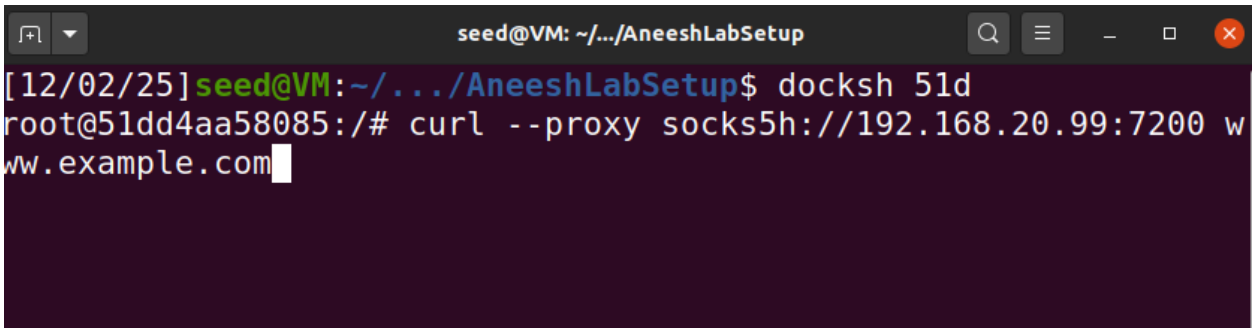


Explanation: we set up a dynamic port-forwarding tunnel from Host B to Host A. By using the -D option, B creates a SOCKS5 proxy that listens on port 7200 of its internal IP address (192.168.20.99). Any application on the internal network that sends traffic to this proxy will have its requests forwarded through the encrypted SSH tunnel to Host A (10.8.0.99), which then connects to the final destination on behalf of B. This allows internal hosts to reach blocked external websites by routing their traffic through A.

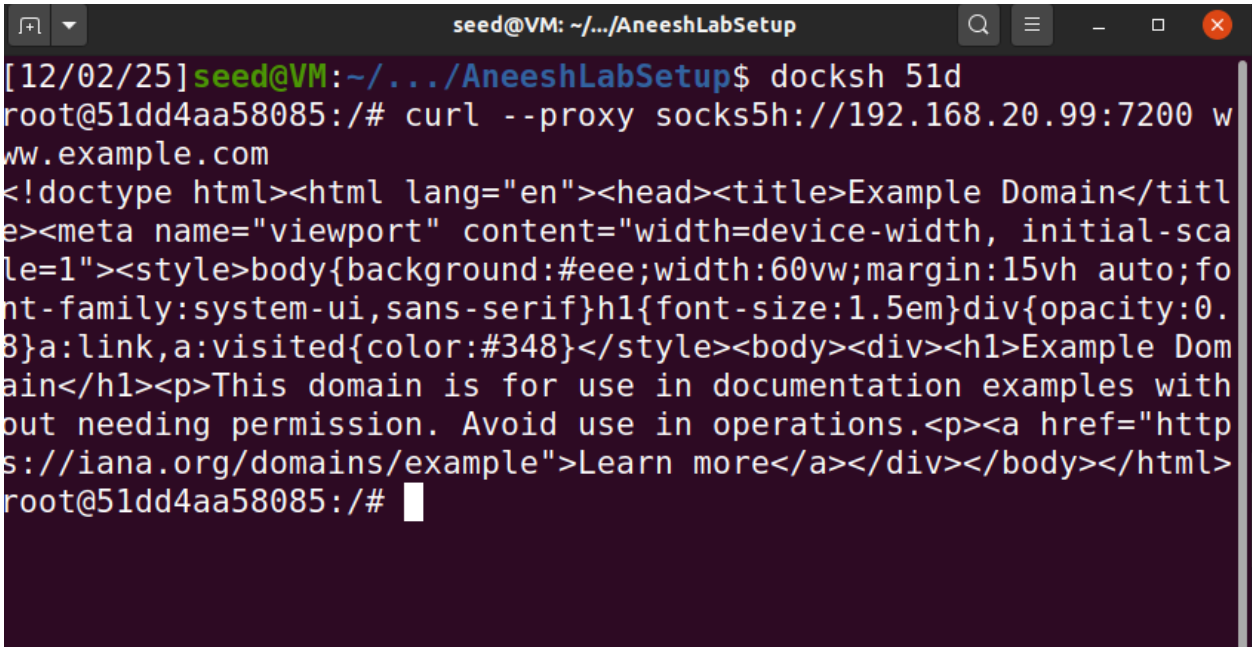
Connecting to Host B1 container screenshot:



Testing Dynamic port forwarding using curl command screenshot:

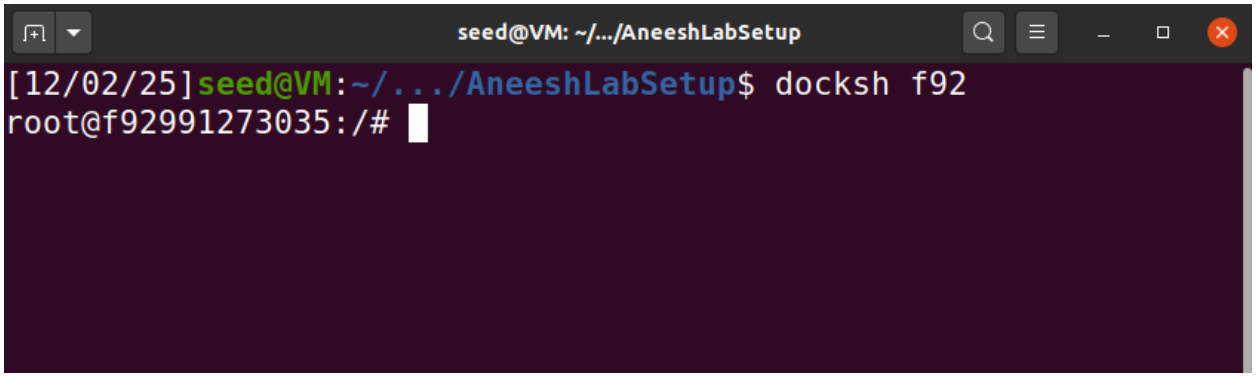


Running the command screenshot and output:

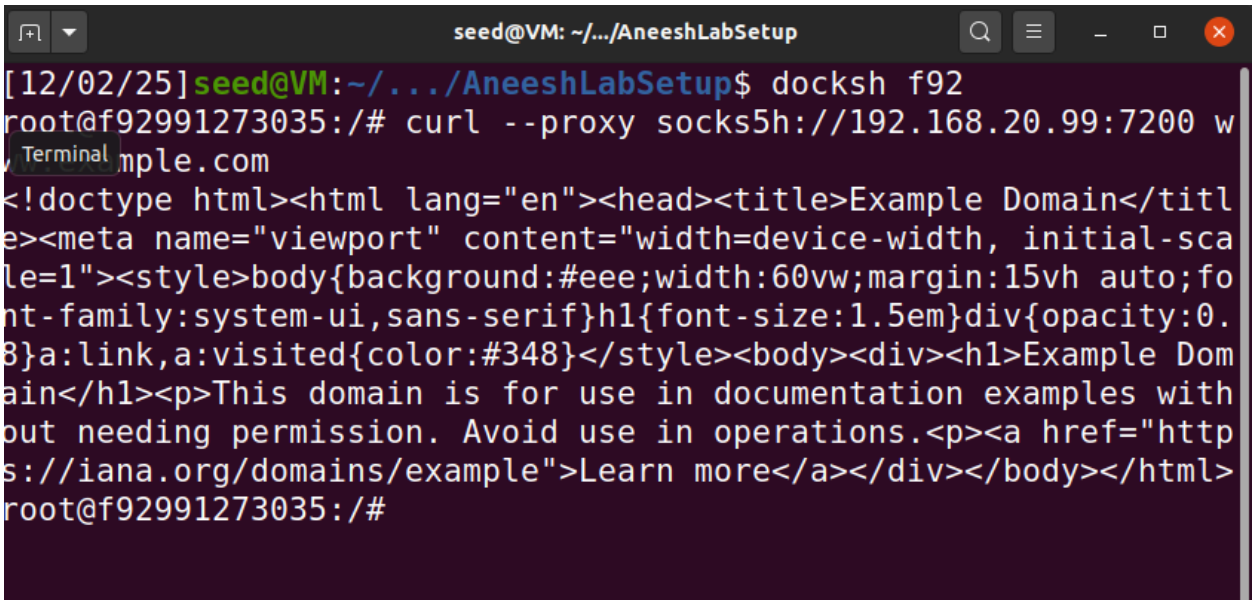


Explanation: As it can be seen from the screenshot above, receiving the HTML page from www.example.com confirms that the SOCKS5 proxy on Host B (listening at 192.188.20.99:7200) is functioning correctly. Although the firewall blocks direct access to www.example.com from the internal network, curl successfully retrieves the website because the request is routed through the SSH dynamic tunnel to Host A. Host A makes the actual external connection, and the response is delivered back through the encrypted SSH tunnel to B, allowing the blocked site to be accessed indirectly.

Connecting to Host B2 container screenshot:



Testing Dynamic port forwarding using curl command screenshot:



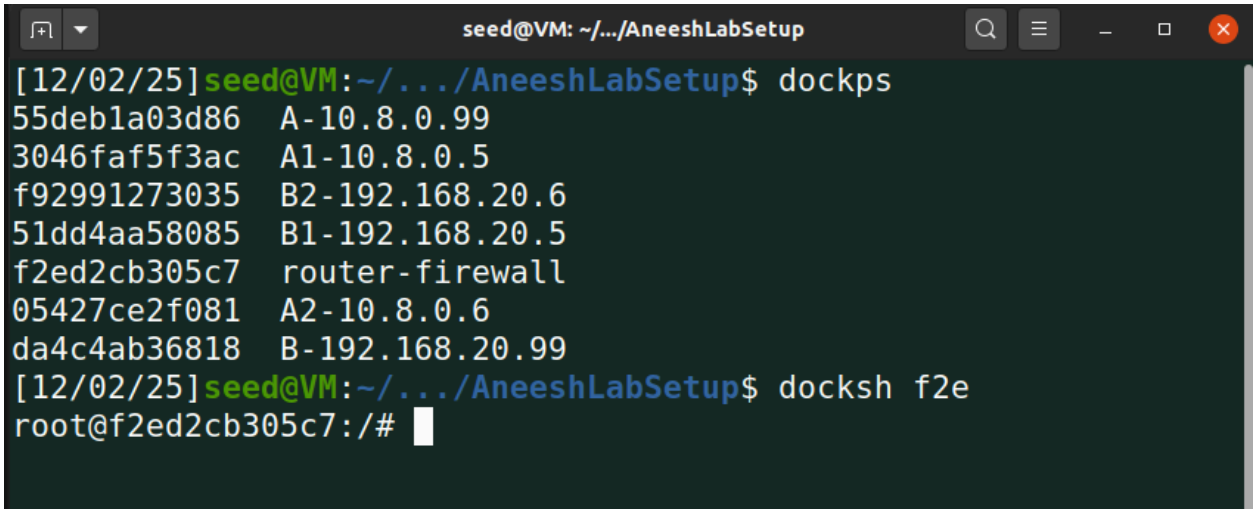
Explanation: We did it again just to make sure its dynamic using Host B2 this time and again the same output which confirms it.

Q1) Which computer establishes the actual connection with the intended web server?
Ans) The actual connection to the web server is established by Host A (10.8.0.99). Host B only sends its traffic to the local SOCKS proxy, which forwards it through the encrypted SSH tunnel. Host A receives these forwarded requests and performs the real outbound TCP connection to the website, bypassing the firewall that blocks B's direct access.

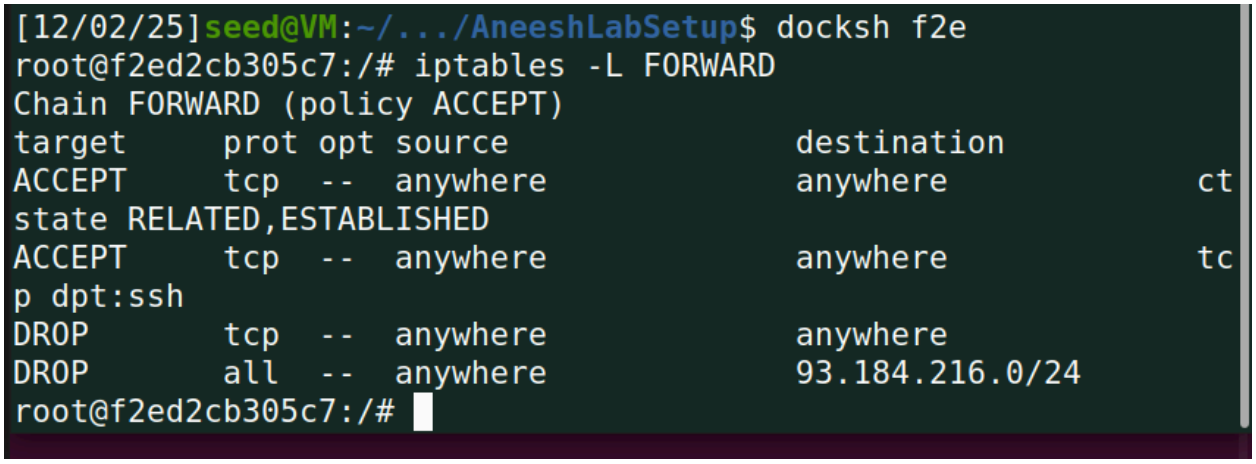
Q2) How does that computer know which server to connect to?
Ans) Host A knows the target server because the SOCKS5 protocol includes the destination hostname and port inside the request sent from curl to the proxy. WHen B receives the request on its SOCKS port (7200), it embeds the server address (www.example.com:80) in the SOCKS5 handshake and forwards it through the SSH tunnel. Host A reads this information from the SOCKS5 header and uses it to initiate the correct external TCP connection.

Task 2.2:

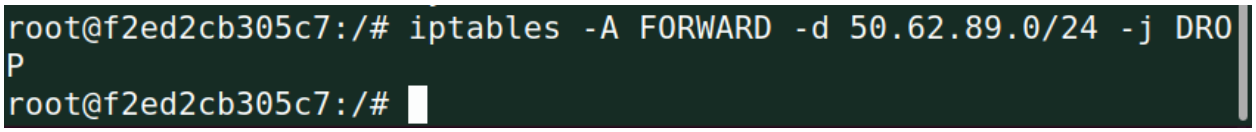
Connecting to the router firewall container screenshot:



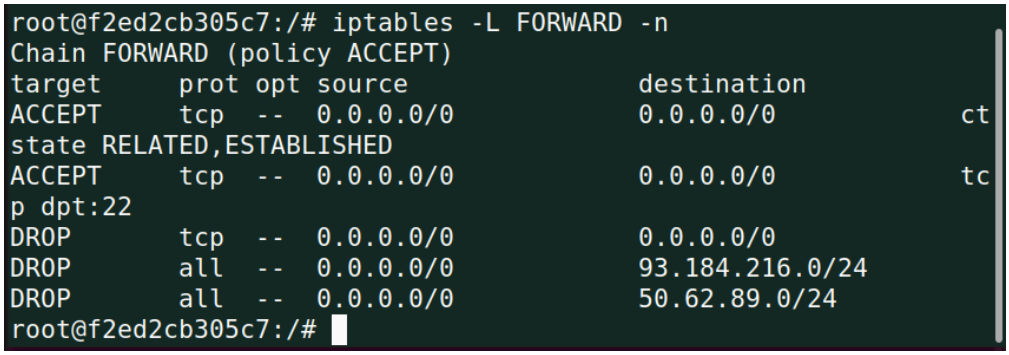
Checking the iptables screenshot:



Appending new website IP address screenshot:

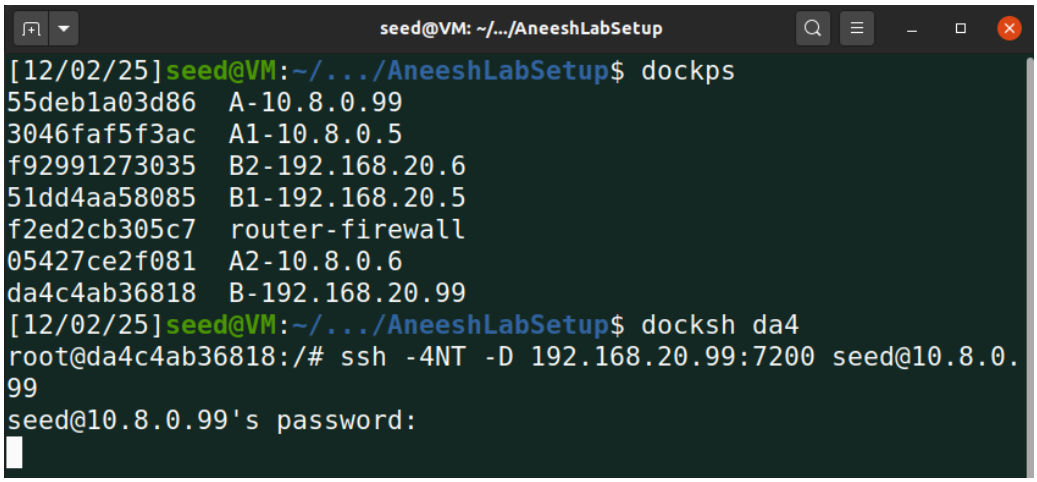


Checking if it was appended screenshot:

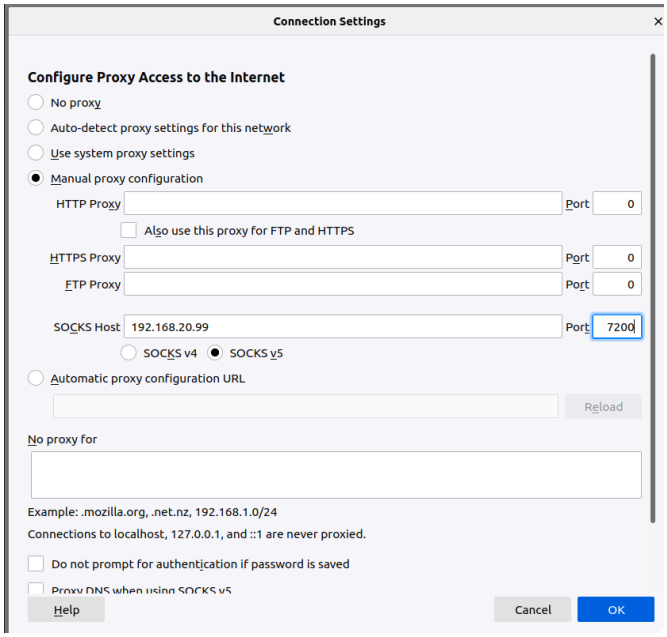


Explanation: As it can be seen from the screenshot above that it was successfully added.

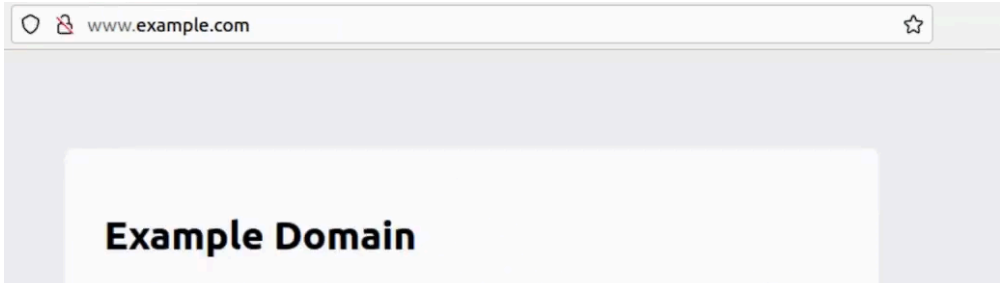
Connecting to Host B container and running dynamic proxy command again:



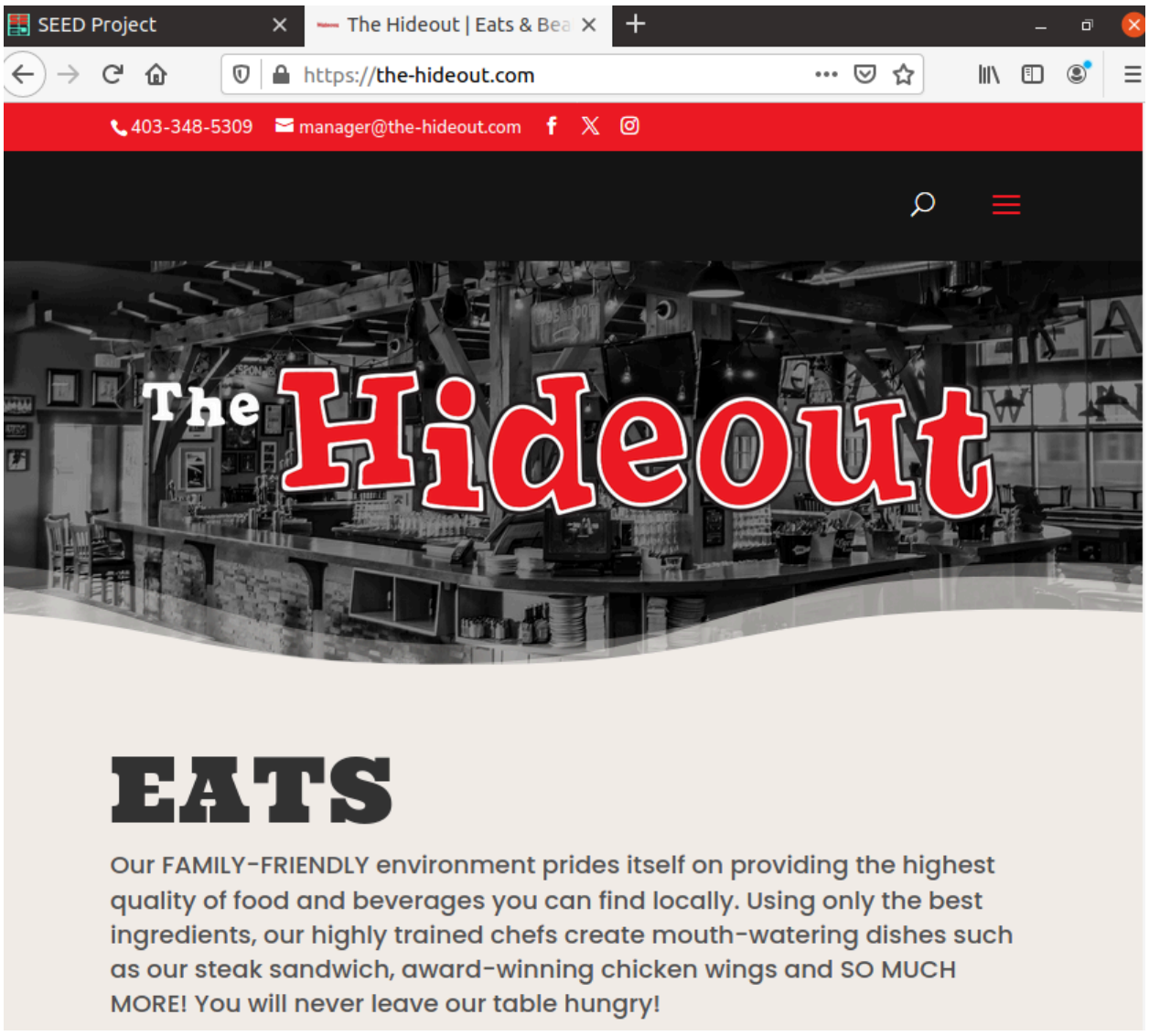
Setting the proxy manually in the browser screenshot:



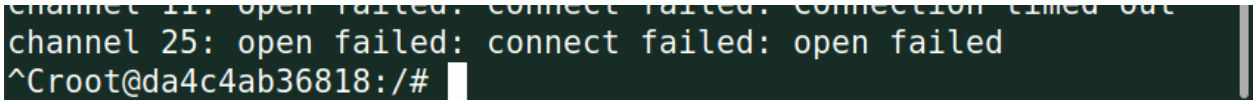
Checking example domain screenshot:



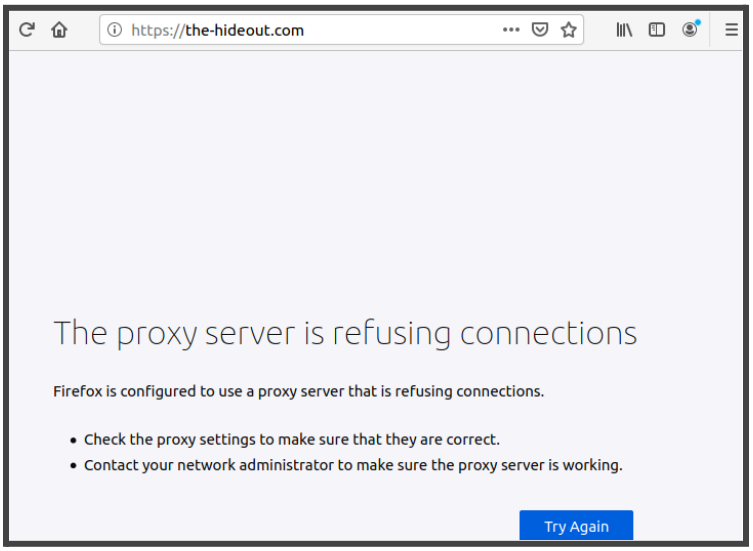
Checking the second website if its connected:



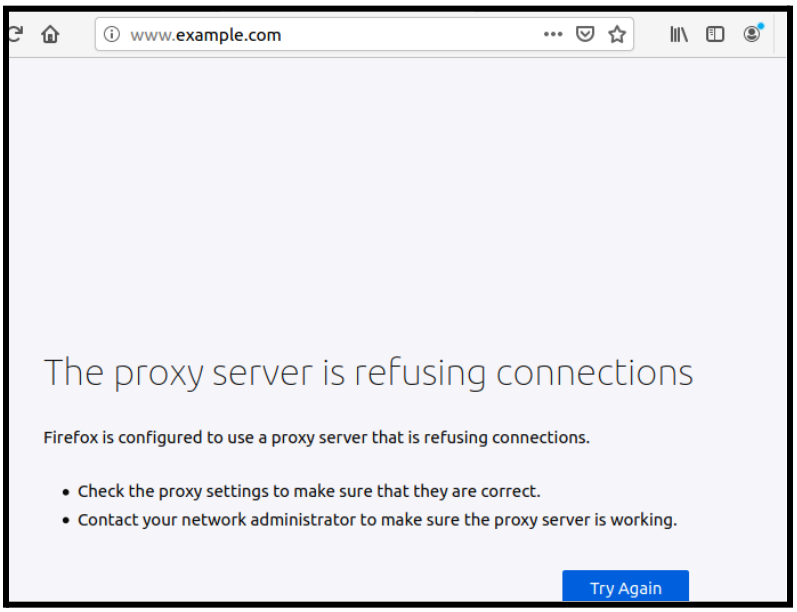
Stopping the dynamic proxy container screenshot:



Refreshing the website to check if its connected to dynamic proxy server:



Checking our example website again:



Explanation: While the dynamic port-forwarding tunnel was active, all browser traffic was routed through the SOCKS5 proxy running on Host B at 192.168.20.99:7200. Because the proxy forwarded requests through the SSH tunnel to host A, the browser was able to access the websites that were blocked by the firewall's rules, including the additional site blocked via the iptables rule. After stopping the proxy with CTRL+C, the SOCKS5 service on B immediately stopped listening on port 7200. The browser was still configured to use this proxy, so when it attempted to refresh the page, it had no functioning proxy to connect to. As a result, the browser displayed "proxy server is refusing connection." Without the proxy running, the traffic could no longer be forwarded through the SSH tunnel and instead followed the normal route, where the firewall correctly blocked both websites due to the configured filtering rules.

Task 2.3:

Changing the port in SocksClient.py file screenshot:

Open *socks_client.py ~/Downloads/Labsetup/AneeshLabSetup Save

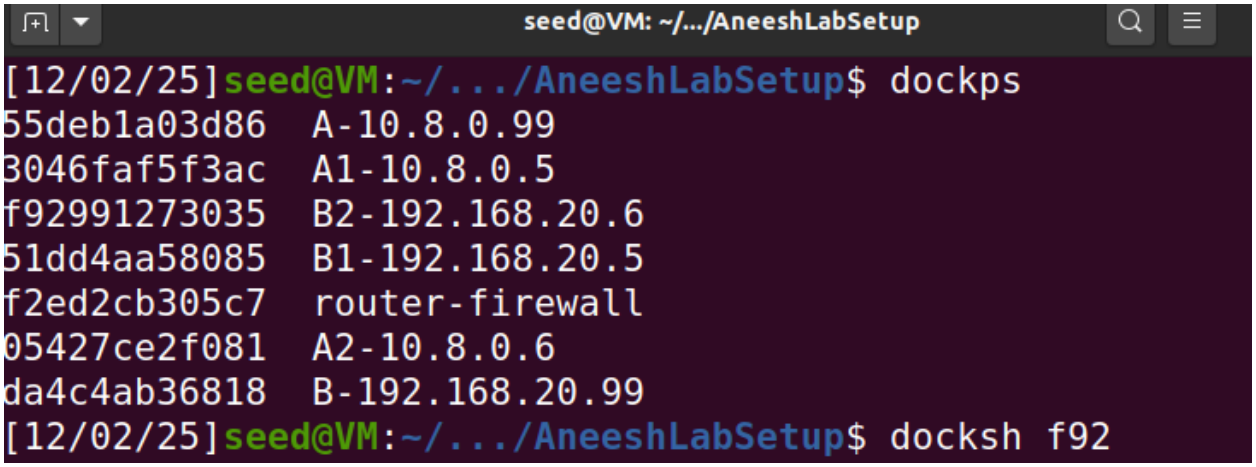
```
#!/bin/env python3

import socks

s = socks.socksocket()

# Set the proxy
s.set_proxy(socks.SOCKS5, "192.168.20.99", 7200)

# 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)
```

Running the python script screenshot:

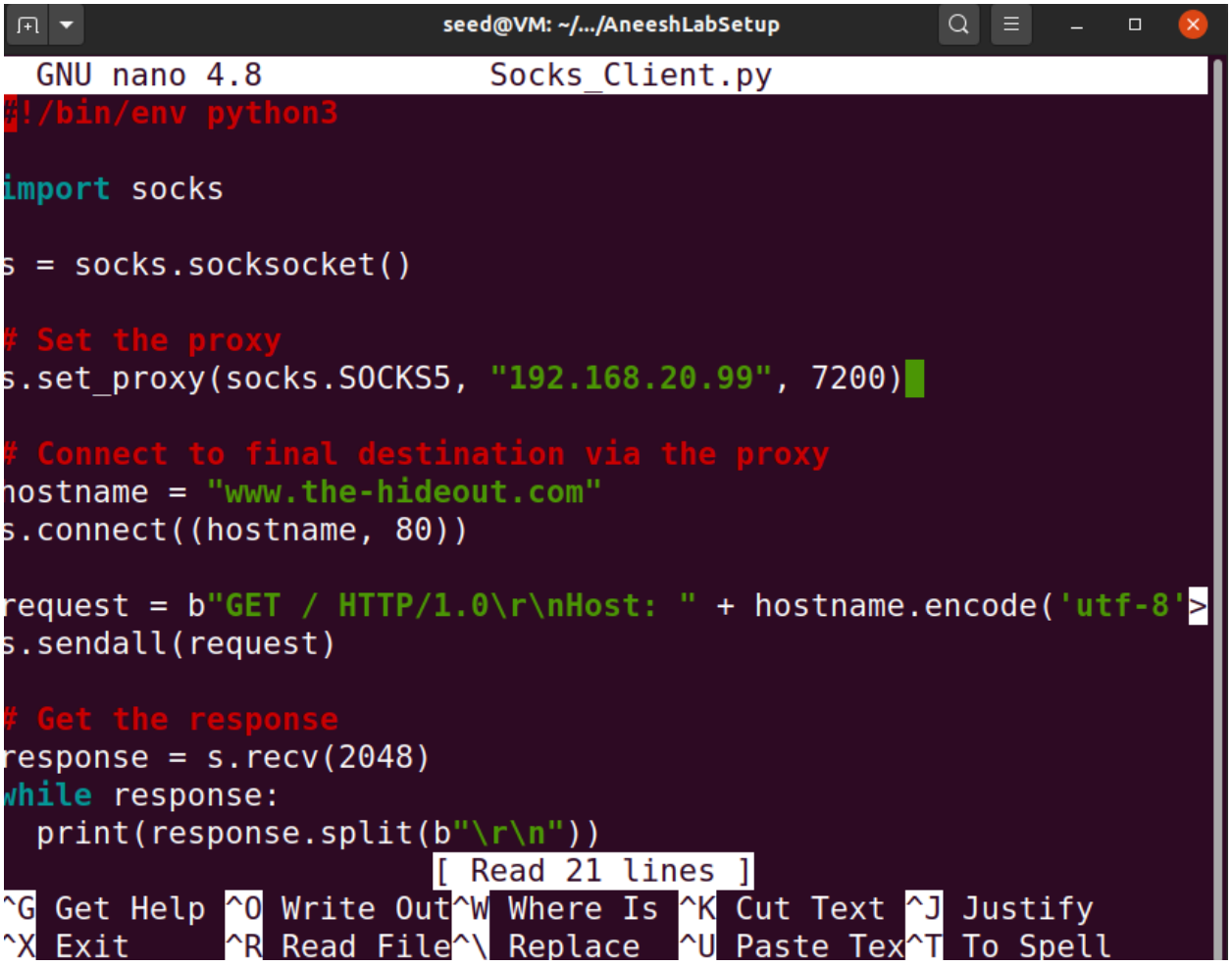
seed@VM: ~/.../AneeshLabSetup

```
root@51dd4aa58085:/home/seed# ls
SocksClient.py
root@51dd4aa58085:/home/seed# ./SocksClient.py
[b'HTTP/1.0 200 OK', b'Content-Type: text/html', b'ETag: "bc2473
a18e003bdb249eba5ce893033f:1760028122.592274"', b'Last-Modified:
 Thu, 09 Oct 2025 16:42:02 GMT', b'Cache-Control: max-age=86000'
, b'Date: Tue, 02 Dec 2025 23:41:04 GMT', b'Content-Length: 513'
, b'Connection: close', b'X-N: S', b'', b'<!doctype html><html l
ang="en"><head><title>Example Domain</title><meta name="viewport
" content="width=device-width, initial-scale=1"><style>body{back
ground:#eee;width:60vw;margin:15vh auto;font-family:system-ui,sa
ns-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{col
or:#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/e
xample">Learn more</a></div></body></html>\n']
root@51dd4aa58085:/home/seed#
```

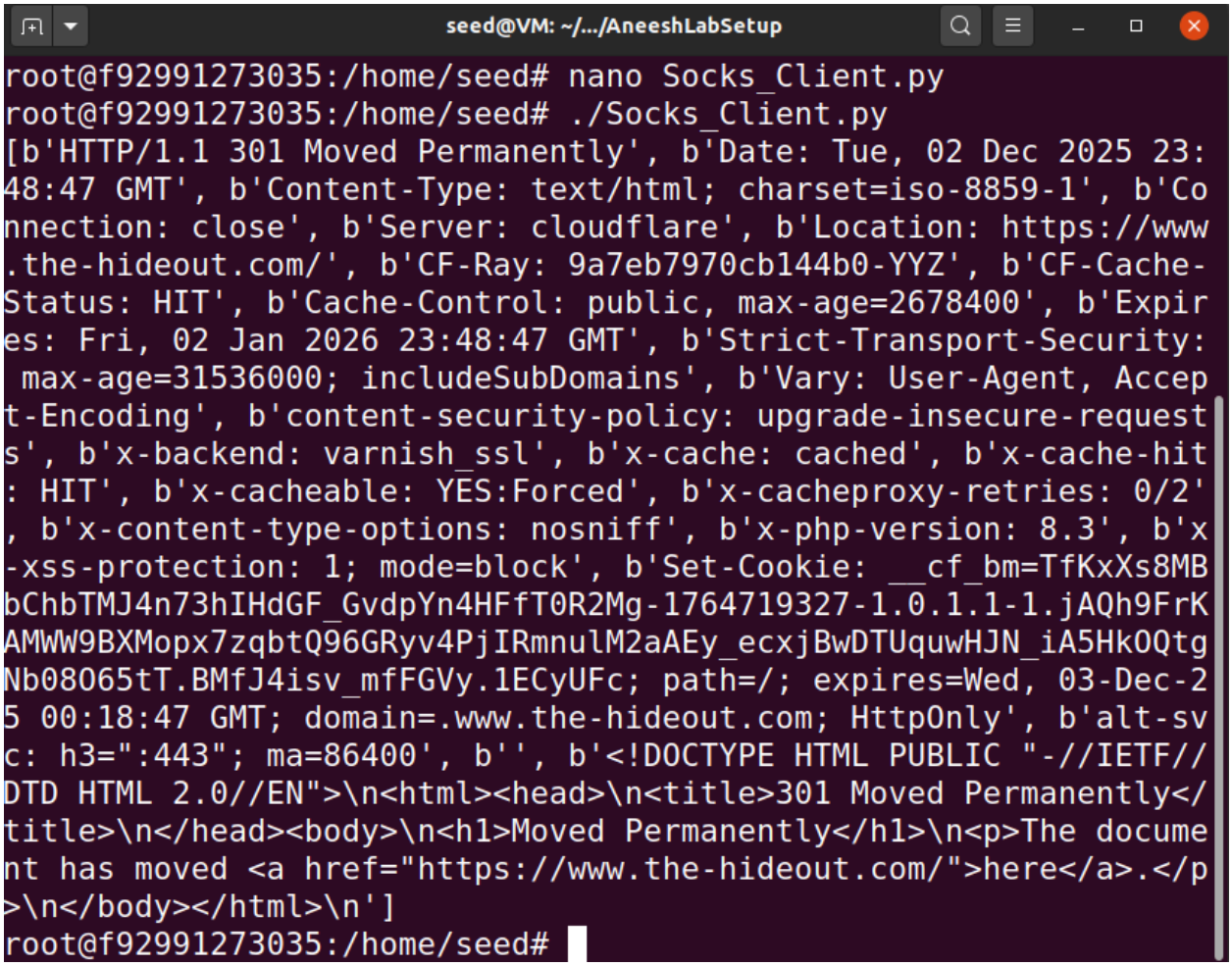
Connecting to the Host B2 container screenshot:



Python script code with different website screenshot:



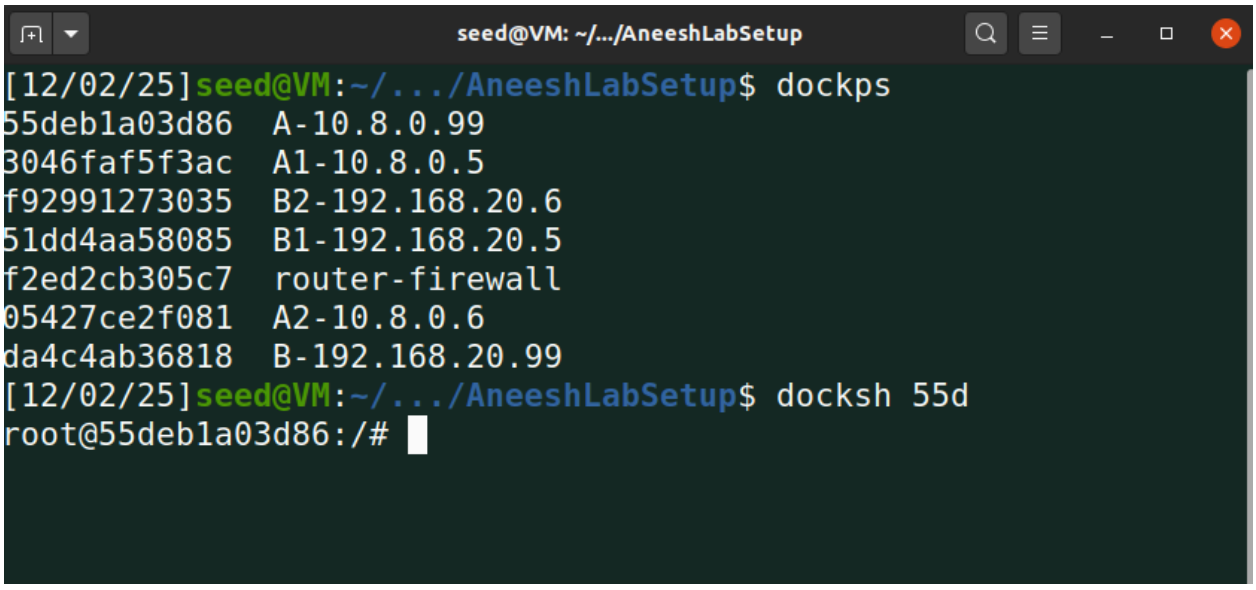
Running the python script screenshot:



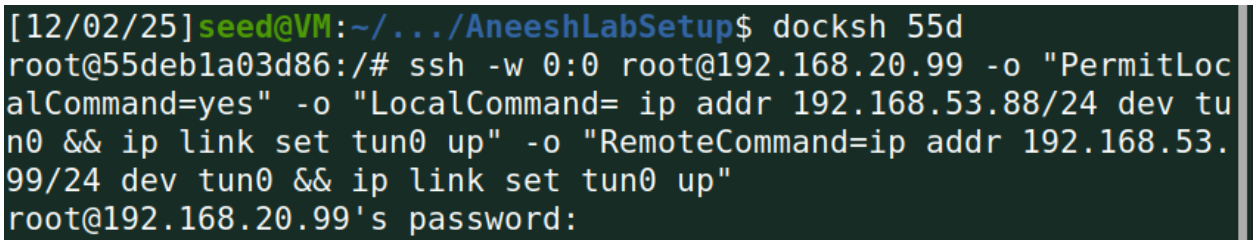
Explanation: Running the same python SOCKS5 client on Host B and Host B2 successfully retrieved the HTML content from both www.example.com and www.the-hideout.com. Although these websites were blocked by the firewalls' rules, the script bypassed the restrictions by forwarding all traffic to the SOCKS5 proxy running on host B. The proxy sent the destination hostname and port through the encrypted SSH tunnel to host A, which established the actual external TCP connections. The web servers' responses were then returned through the tunnel to the internal hosts. This confirms that any internal container using the SOCKS proxy can reach blocked websites by routing its traffic through Host A instead of directly through the firewall.

Task 3.1:

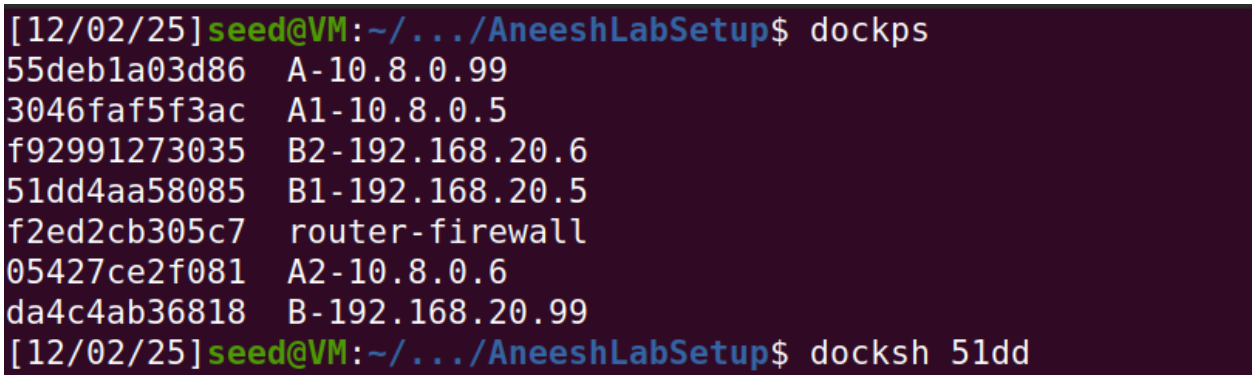
Starting container as a Host A screenshot:



Creating the VPN tunnel using SSH screenshot:



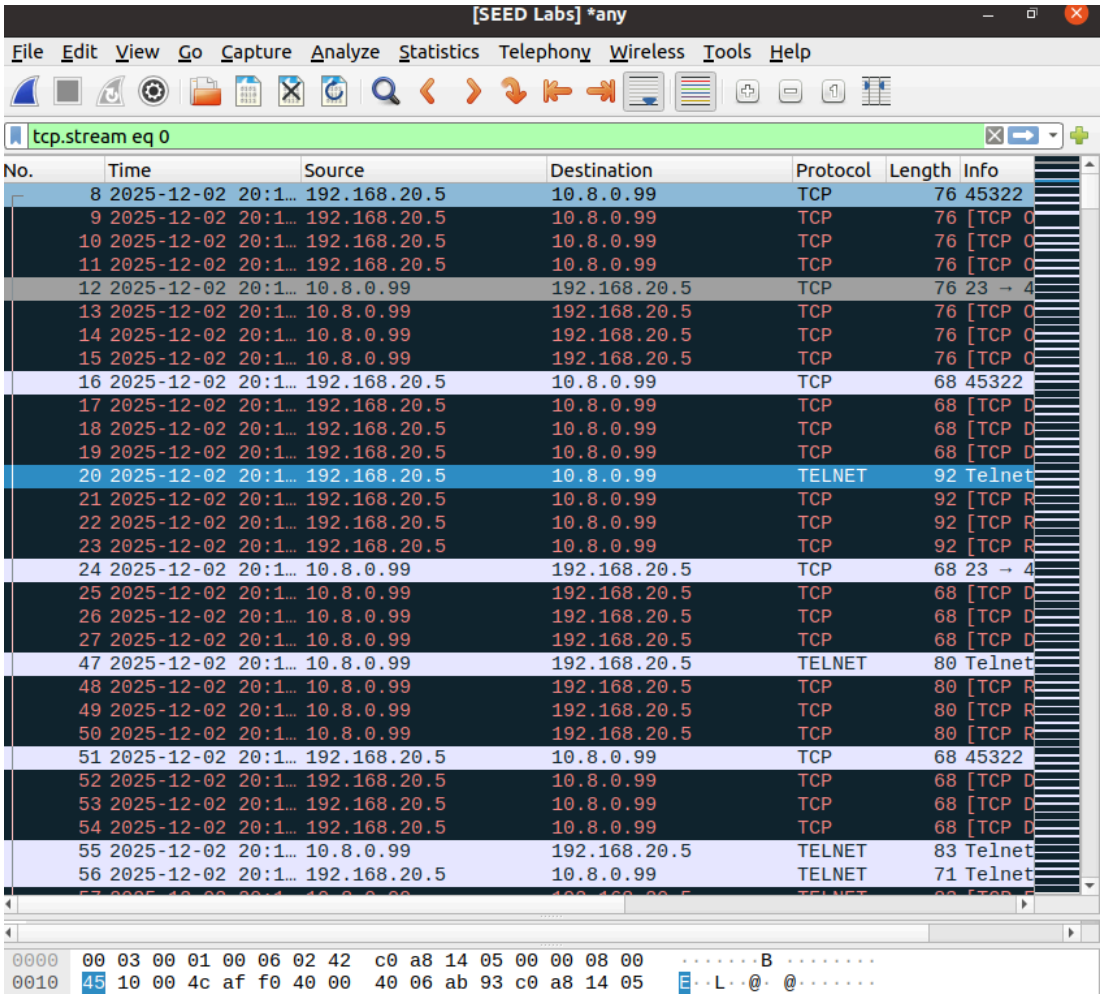
Connecting to Host B1 for telnet command:



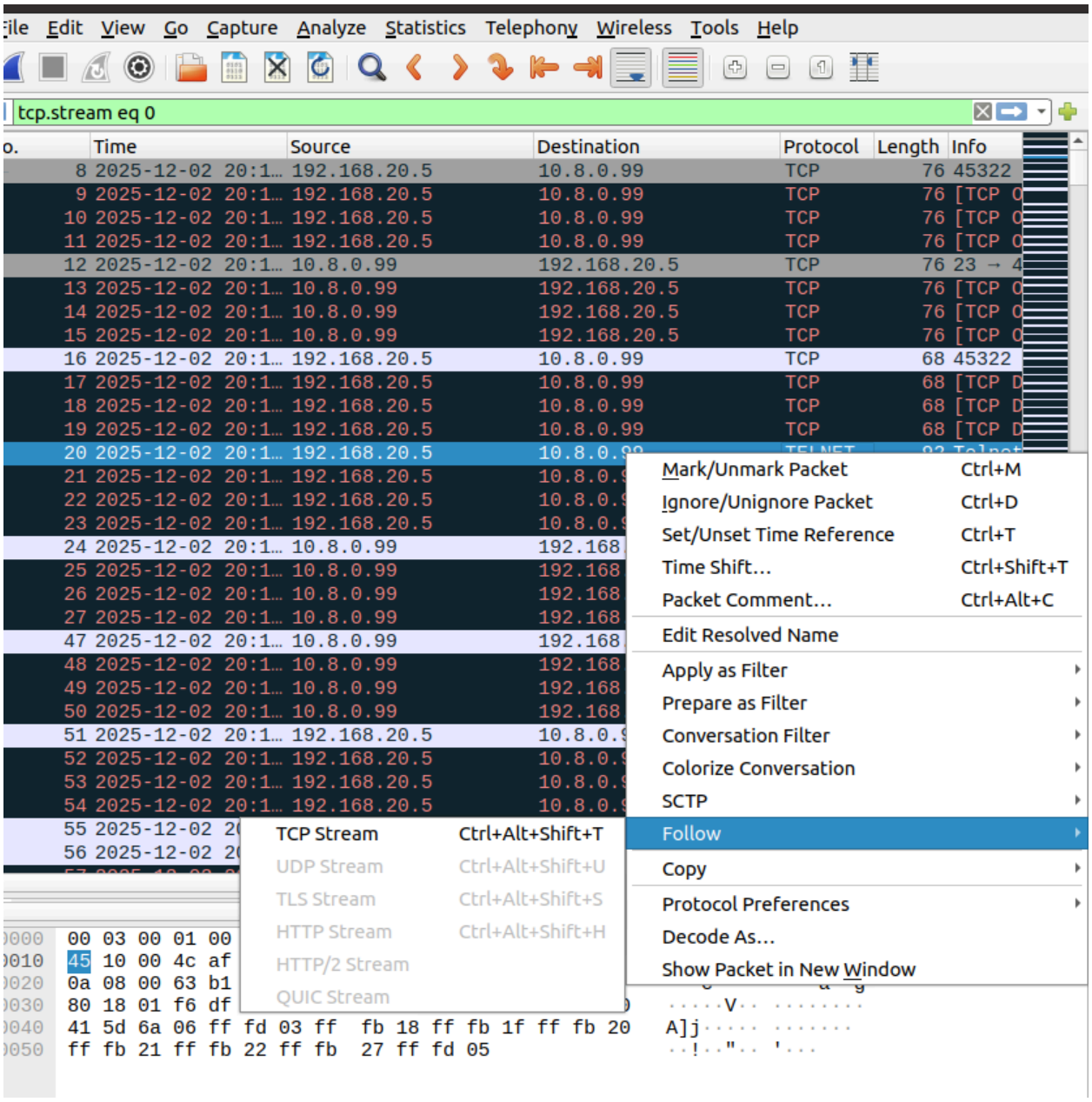
Running the Telnet Command:

```
root@51dd4aa58085:/# telnet 10.8.0.99
Trying 10.8.0.99...
Connected to 10.8.0.99.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
55deb1a03d86 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64
)

 * 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.
seed@55deb1a03d86:~$
```

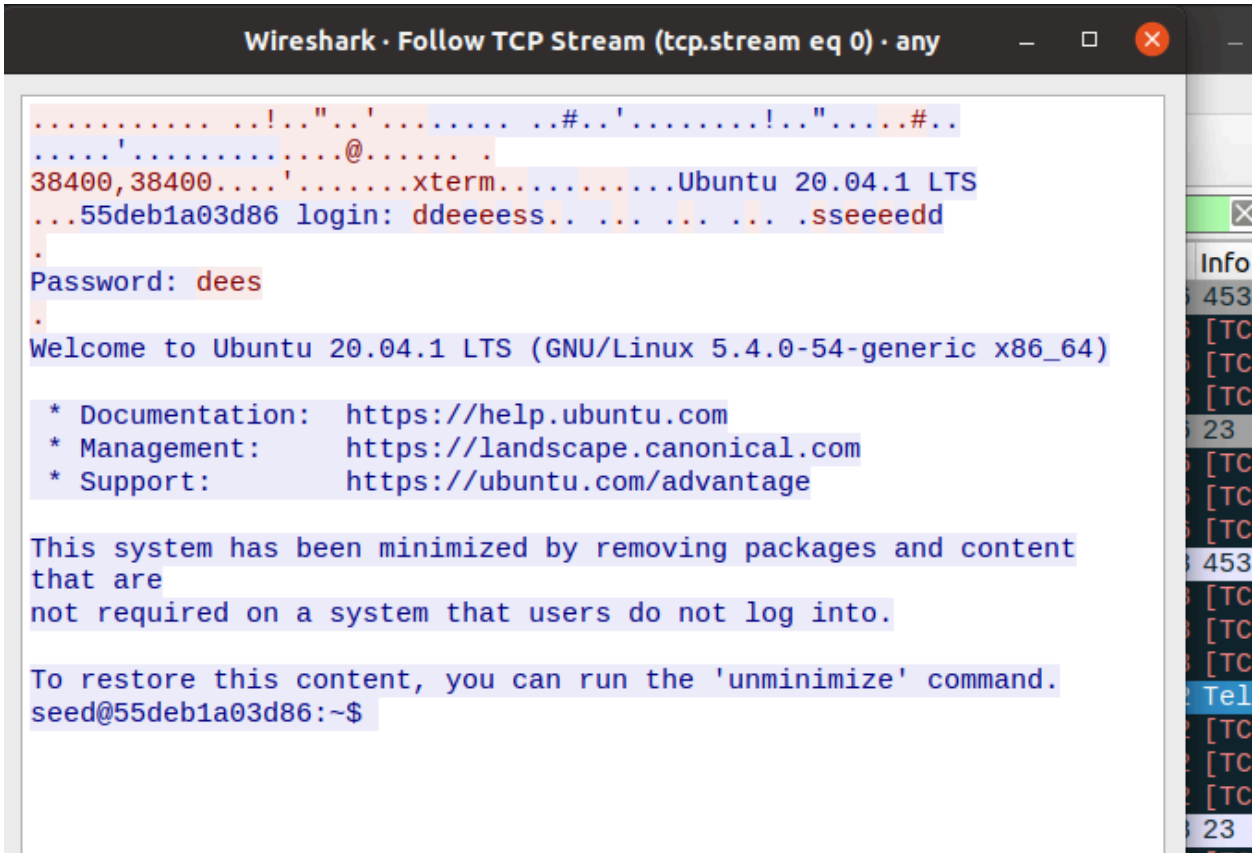
Checking the wireshark once the command was ran:



Following the Telnet Protocol in wireshark screenshot:



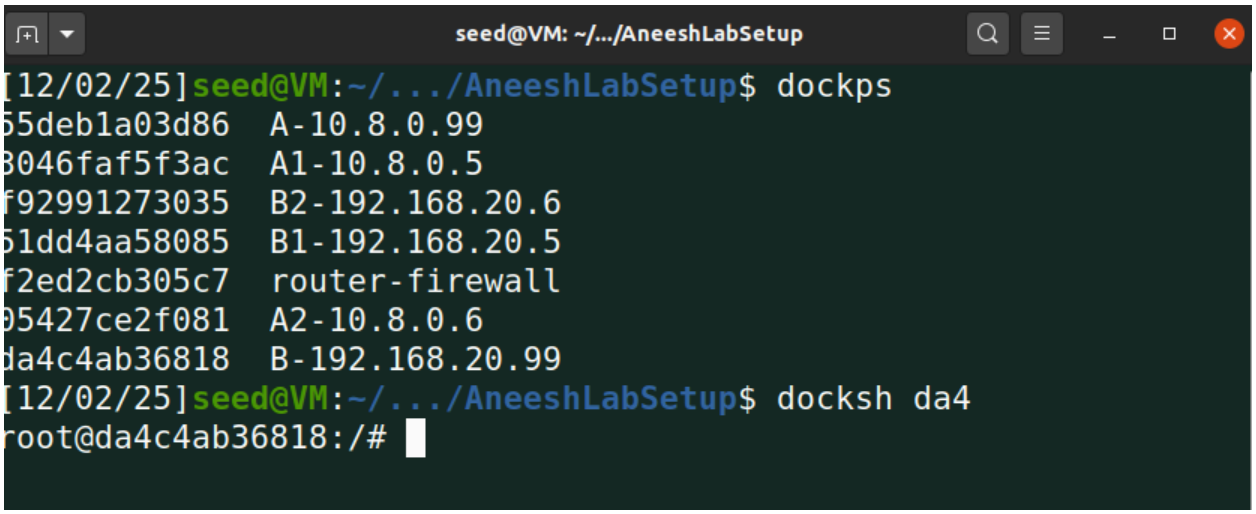
Checking the TCP Stream screenshot:



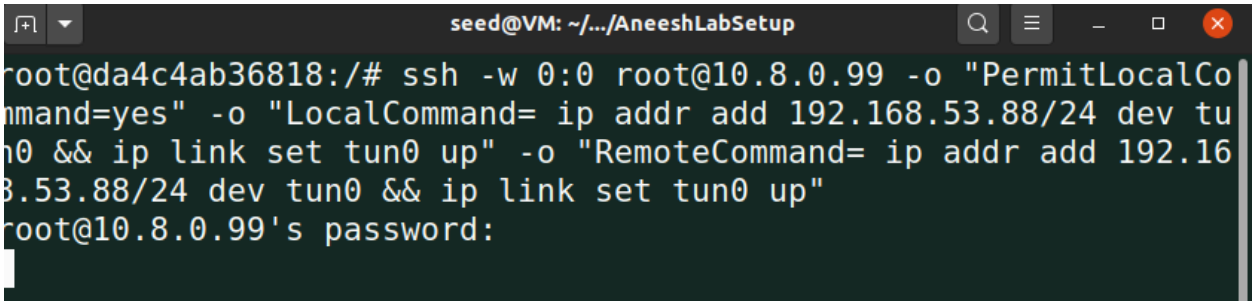
Explanation: When the Telnet connection was initiated from Host B1 to external host A through the VPN tunnel, the firewall normally blocks this type of traffic, the VPN encapsulates all packets inside an encrypted SSH connection between A and B. Because of this, the firewall did not see any direct Telnet packets or SYN attempts. It only observed encrypted SSH packets flowing through the tunnel. This confirms that the Telnet session was successfully routed through the tun0 interfaces created by the SSH. When I opened Wireshark on the internal interface and located the Telnet packets before they were encrypted by the VPN. When I followed the TCP stream, Wireshark reconstructed the complete Telnet session. Even the username and password were visible in plaintext within the stream. This is because all data including credentials are sent as plaintext without encryption as Telnet doesn't provide that. Anyone with access to the network path could capture and read this information if the Telnet traffic is not protected by a tunnel.

Task 3.2:

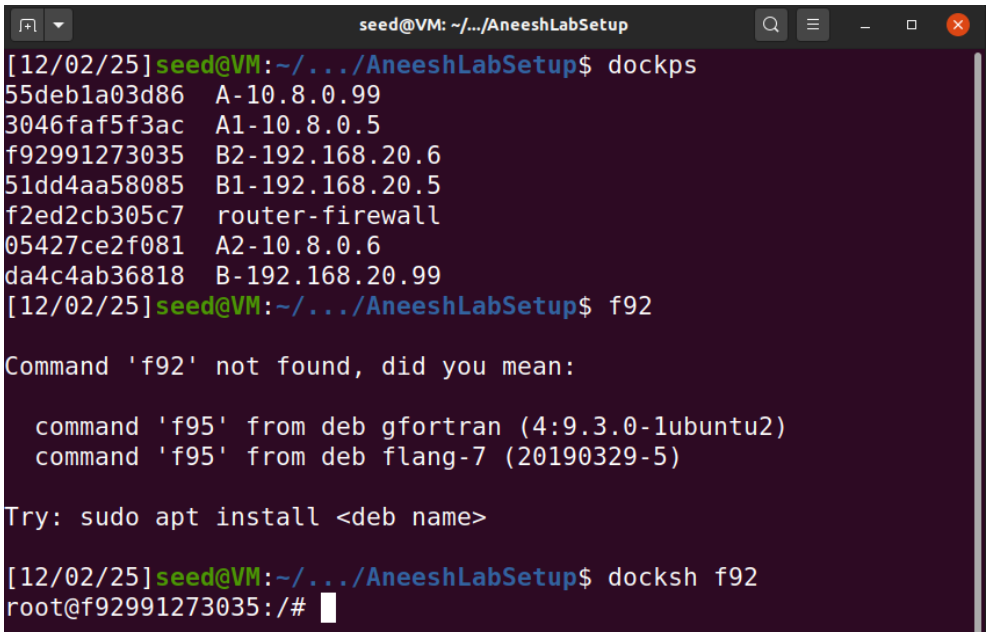
Connecting to Host B container screenshot:



Running the SSH command screenshot on Host B:



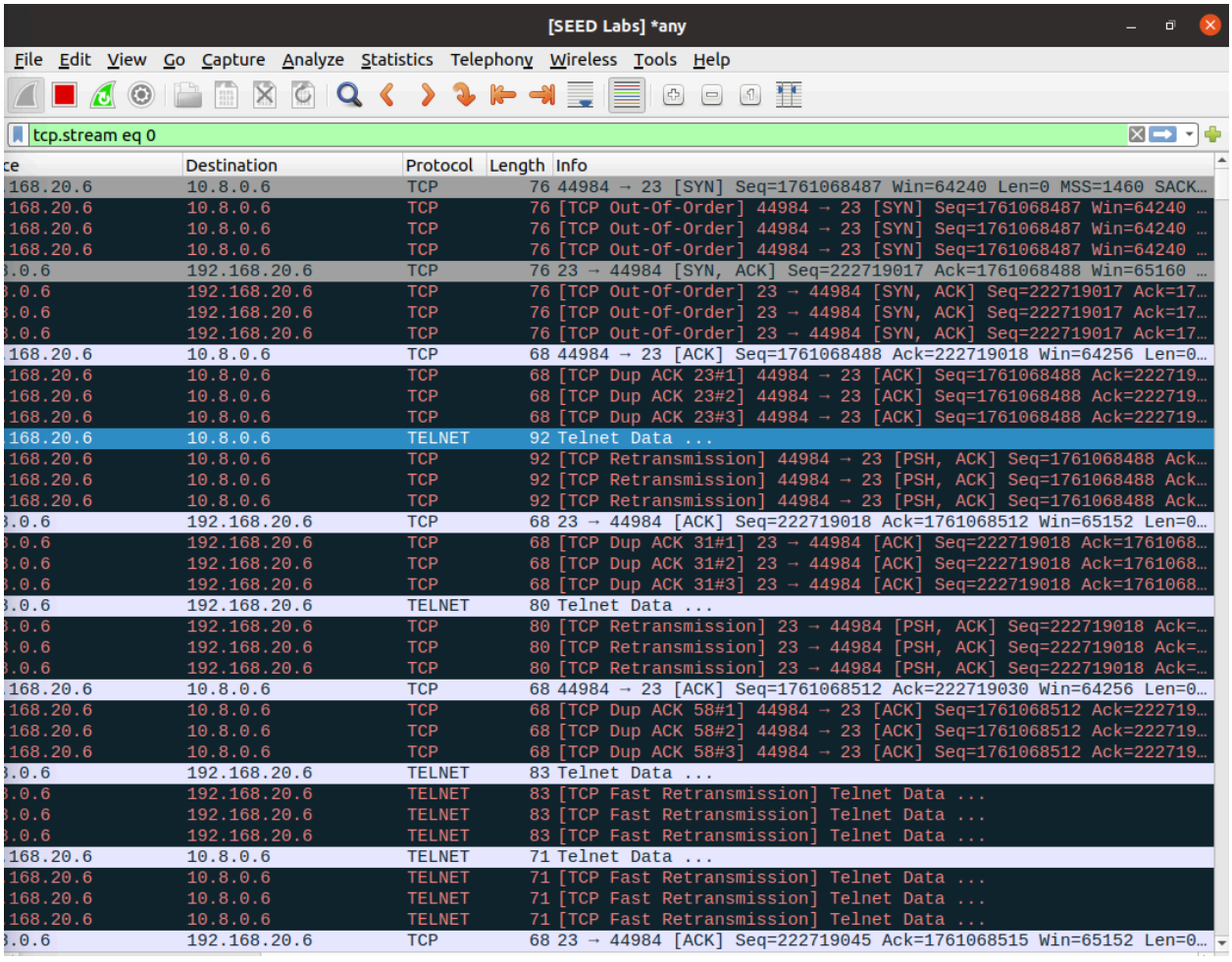
Connecting to Host B2 container screenshot:



Running the Telnet command on HostB container:

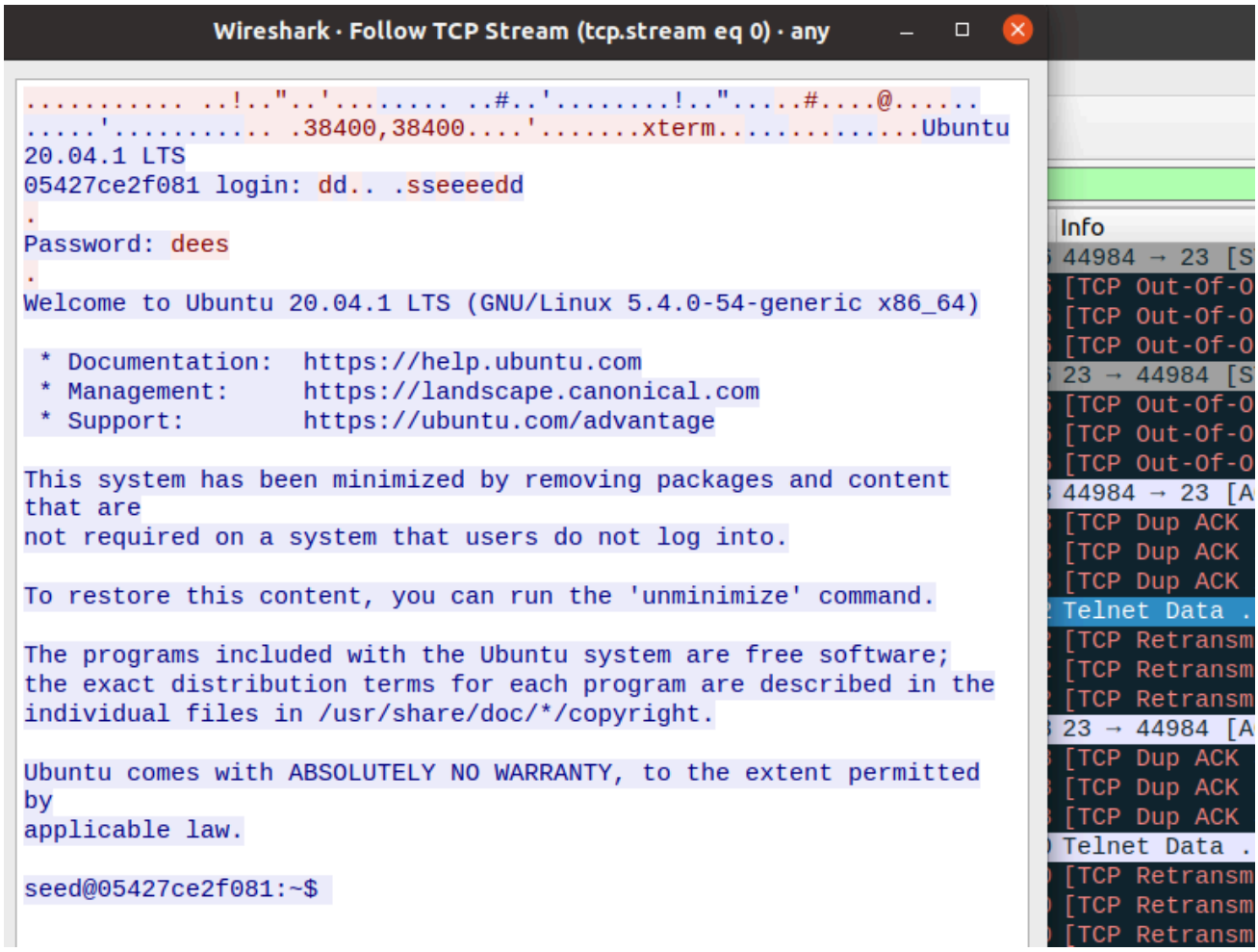
```
[12/02/25]seed@VM:~/.../AneeshLabSetup$ docksh f92
root@f92991273035:/# telnet 10.8.0.6
Trying 10.8.0.6...
Connected to 10.8.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
05427ce2f081 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64
)

 * 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;
```

Wireshark screenshot after running the Telnet command:



Explanation: In the screenshot above it can be seen that on wireshark the internal interface shows the full TCP handshake and Telnet Data because this traffic is captured before entering the VPN tunnel. Once encapsulated the Telnet packets become encrypted inside SSH, so only the inner unencrypted traffic is visible on the internal side.

Checking TCP Stream of Telnet screenshot:



Explanation: The packets are not blocked by the firewall because all the traffic from B, B1 and B2 is routed through the VPN tunnel instead of going directly to the blocked websites. When the VPN is active, the internal hosts send their packets to tun0, where they are encapsulated and encrypted inside an SSH connection to A. The firewall only sees SSH traffic between B and A, which is allowed, and never sees the actual HTTP requests to the blocked IP ranges. It then makes the external web connection on behalf of the internal hosts, so the firewall's egress filters are completely bypassed.

Task 4:

A SOCKS5 proxy forwards traffic on an application by application basis so only programs that are explicitly configured to use the proxy such as curl, browser or custom scripts will actually send their traffic through it. It does support tunneling for specific connections but does not protect all system traffic. The main advantages of SOCKS5 are that it is lightweight, easy to set up, and very effective for bypassing egress filters, but it does not encrypt any traffic at all beyond the SSH tunnel and also requires each application to have SOCKS support.

Whereas, A vpn creates a virtual network interface (tun0) that routes all system traffic through the tunnel automatically. Any program, even those without proxy support for example telnet can use the tunnel without modification. The pros of a VPNs are that it provides full network-level protection, hides all internal traffic from the firewall, and works transparently for every application. The cons are that they require more configuration and can introduce more overhead compared to SOCKS. For example in Task 3, when I created VPN tunneling every program on B, B1 and B2 automatically used the tunnel without any extra config.

Overall, I think SOCKS5 is useful when you only need to tunnel specific applications while the VPN is more powerful because it is transparent and protects all traffic without much setup.