Firewall Evasion Lab

Lab Environment Setup

Please download the Labsetup.zip file from the link given below :

https://seedsecuritylabs.org/Labs_20.04/Networking/Firewall_Evasion/

Follow the instructions in the lab setup document to set up the lab environment.

Task 0 : Get Familiar with the Lab Setup

We will conduct a series of experiments in this chapter. These experiments need to use several computers in two separate networks. The experiment setup is depicted in Figure 1. We use docker containers for these machines. Readers can find the container setup file from the website of this lab. In this lab, the network 10.8.0.0/24 serves as an external network, while 192.168.20.0/24 serves as the internal network.
The host 10.8.0.1 is not a container; this IP address is given to the host machine (i.e., the VM in our case). This machine is the gateway to the Internet. To reach the Internet from the hosts in both 192.168.20.0/24 and 10.8.0.0/24 networks, packets must be routed to 10.8.0.1. The routing has already been set up.

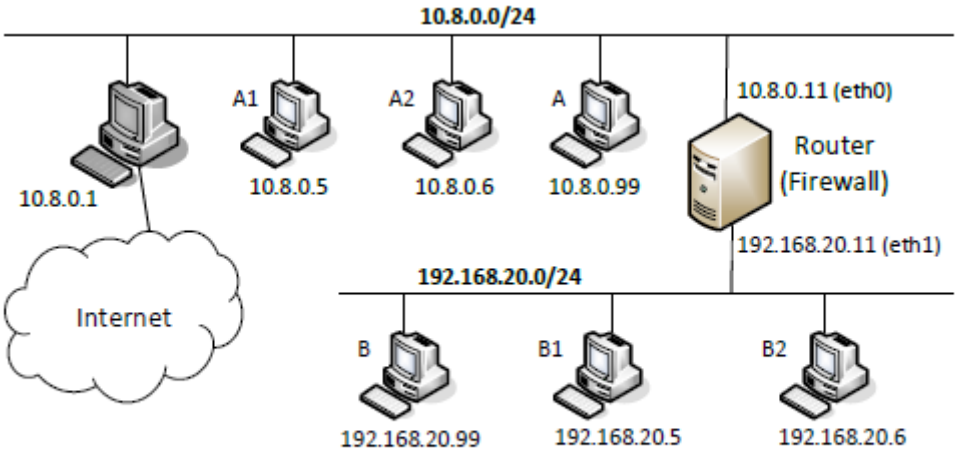


Figure 1: Network setup

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.

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
```

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
```

Q. Explain what each of the firewall rules do.

Lab task. 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.

On the router-firewall container:

iptables -A FORWARD -i eth1 -d 13.107.42.0/24 -j DROP
This IP address was for www.linkedin.com

iptables -A FORWARD -i eth1 -d 13.249.221.0/24 -j DROP
This IP address was for www.miniclip.com

To verify that the websites have been blocked, ping them from any machine in the internal network ie. **B/B1/B2**.

 # ping www.linkedin.com
 # ping www.miniclip.com

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).

Lab task. 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.

Please answer the following 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.
(2) Why can this tunnel successfully help users evade the firewall rule specified in the lab setup?

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.

Run the below command on container A:

 # ssh -L 0.0.0.0:8000:192.168.20.99:23 root@192.168.20.99

password: dees

take a screenshot

Run the below command on container A:

 # telnet 10.8.0.99 8000

username: seed password: dees

 # ifconfig

take a screenshot

take a screenshot of wireshark output

to exit the connection type 'exit' and then ctrl+D

REPEAT THE TASK ONCE MORE WITH EITHER HOST A1 OR A2 AND PUT THE SCREENSHOTS

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.

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

> **# ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N**
> take a screenshot

Q. What is the significance of using 0.0.0.0 in the command?

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

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

Please also answer the following questions:

(1) Which computer establishes the actual connection with the intended web server?
(2) How does this computer know which server it should connect to?

Run in container B:

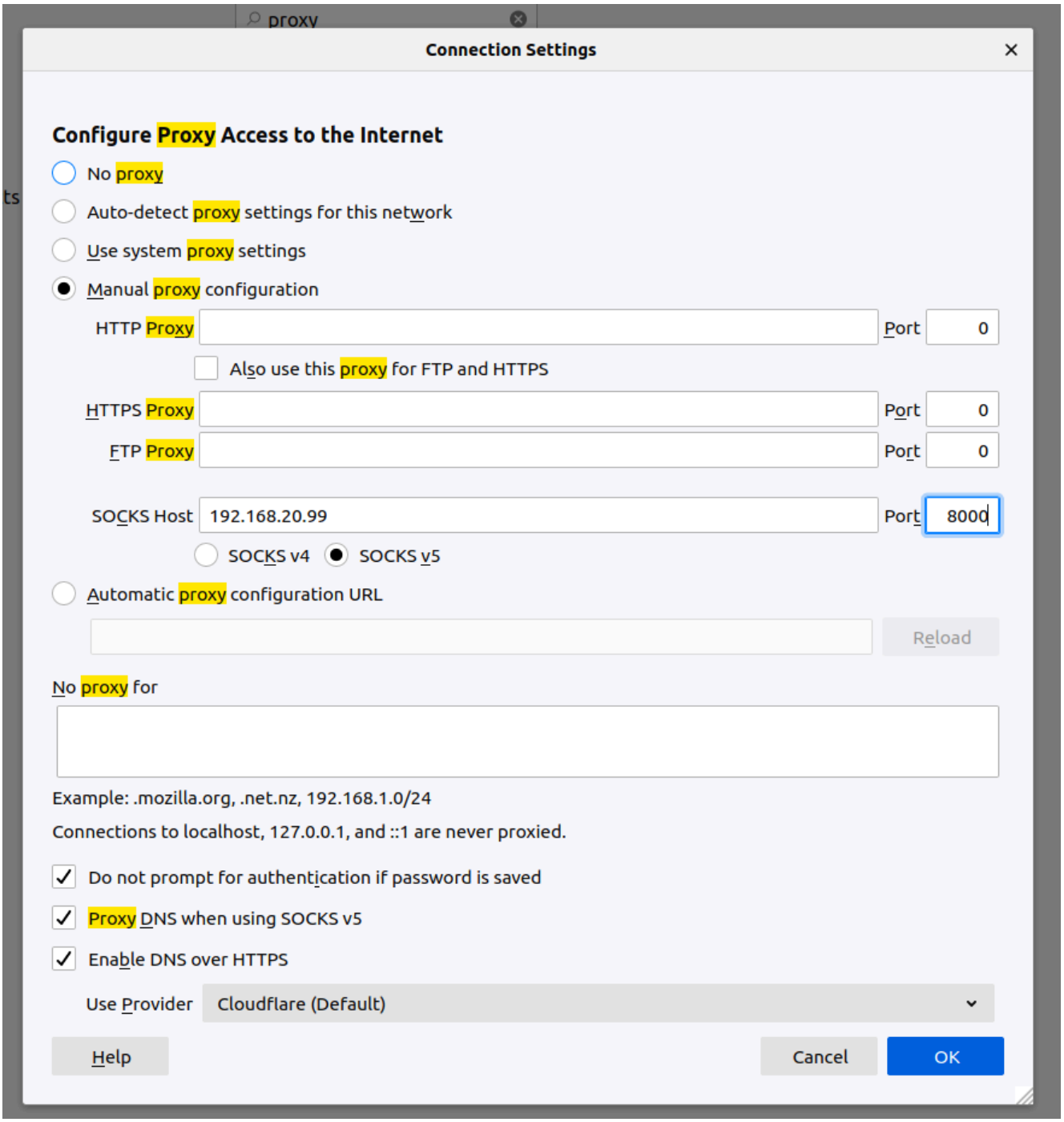
> **# curl -x socks5h://0.0.0.0:8000 http://www.example.com**
> take a screenshot

To access the websites from hosts **B1,B2** run the following command in the corresponding containers:

> **# curl -x socks5h://192.168.20.99:8000 http://www.example.com**
> take a screenshot

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.



Lab task. Once the proxy is configured, we can then browse any website. The requests and replies will
go through the SSH tunnel. Since the host VM can reach the Internet directly, to make sure that our web browsing traffic has gone through the tunnel, you should do the following:

(1) run tcpdump on the router-firewall, and point out the traffic involved in the entire port forwarding process.
take a screenshot
(2) Break the SSH tunnel, and then try to browse a website. Describe your observation.

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 :

After showing that you can browse the site using the browser run the below commands and try again.

Run the below commands in container B:

 # ps -eaf | grep "ssh"

take a screenshot

 # kill [corresponding pid of the process]

take a screenshot

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 labs may be affected.

Task 2.3: Writing a SOCKS Client Using Python

For port forwarding to work, we need to specify where the data should be forwarded to (the final destination). In the static case, this piece of information is provided when we set up the tunnel, i.e., it is hard-wired into the tunnel setup. In the dynamic case, the final destination is dynamic, not specified during the setup, so how can the proxy know where to forward the data?
Applications using a dynamic port forwarding proxy must tell the proxy where to forward their data. This is done through an additional protocol between the application and the proxy. A common protocol for such a purpose is the SOCKS (Socket Secure) protocol, which becomes a de facto proxy standard.
Since the application needs to interact with the proxy using the SOCKS protocol, the application software must have a native SOCKS support in order to use SOCKS proxies. Both Firefox and curl have such a support, but we cannot directly use this type of proxy for the telnet program, because it does not provide a native SOCKS support. In this task, we implement a very simple SOCKS client program

using Python.

Lab task. 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.

Run in container B:

 # ssh -4 -D 0.0.0.0:8000 root@10.8.0.99 -f -N

take a screenshot

Run in container B :

 # nano python3 B-Socks-Client.py (or you can create a file using file explorer or gedit)

paste the Code:

```
#!/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)
```

 # python3 B-Socks-Client.py

take a screenshot

Run in containers B1 or B2:

 # nano B1-B2-Socks-Client.py (or you can create a file using file explorer or gedit)

paste the code:

```
#!/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)
```

 # python3 B1-B2-Socks-Client.py

take a screenshot

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 :

 # ps -eaf | grep "ssh"

 # kill [corresponding pid of the process]

Now try running the python files again in the containers

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.

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.