The exercises are designed for students to finish in an individual capacity. The exercises are not designed to be completed in tutorial sessions but rather to give you some tasks and a starting point to continue and complete on your own.

1 Lab Setup

Please download a **fresh copy** of the SecureCorp project for this lab.

Run the command below on the GNS3 VM shell to download the project. If you SSH into the VM from your host OS terminal, you can simply copy and paste the command instead of typing it manually.

```
gdown 1hhDuc2S7MMdz2KO6tfr85PSx7W4t87Pp ; sudo bash ./install_SecureCorp.sh
```

Alternatively, you can use the link below to download the same project. However, if you are connected to the **Monash Wi-Fi**, this method may not work. In that case, please use a mobile hotspot. (single command)

```
wget https://sniffnrun.com/install_SecureCorp.sh --no-check-certificate ; \
sudo bash ./install_SecureCorp.sh
```

Open SecureCorp network configuration in GNS3 and start all nodes.

2 Lab Tasks

Mikrotik RouterOS firewall is quite similar to Linux `iptables` firewall. They both use Tables, Chains, Rules and Target/Actions.

Table	**Chains**
filter	INPUT OUTPUT FORWARD
nat	OUTPUT PREROUTING POSTROUTING
mangle	INPUT OUTPUT FORWARD PREROUTING POSTROUTING
raw	OUTPUT PREROUTING

Table 1: Tables and Chains

- Tables: Tables are files that join similar actions. A table consists of several chains.
- Chains: A chain is a string of rules. When a packet is received, the firewall finds the appropriate table, then runs it through the chain of rules until it finds a match.
- Rules: A rule is a statement that tells the system what to do with a packet. Rules can block one type of packet, or forward another type of packet.
- Target/Action: A target is a decision of what to do with a packet. Typically, this is to accept it, drop it, or reject it.

For the ease of understanding, we will rename the 3 interfaces of the `Perimeter-Firewall` as below (username `admin` and password admin):

```
/interface
set "ether1" name="wan"
set "ether2" name="lan"
set "ether3" name="dmz"
```

1. **Packet Filtering**

   Firewall filters are used to allow or block specific packets forwarded to your local network, originated from your router, or destined to the router.

   There are two methods on how to set up filtering:

   - Allow specific traffic and drop everything else
   - Drop only malicious traffic, everything else is allowed.

   Both methods have pros and cons. However, from a security point of view first method is much more secure, but requires administrator input whenever traffic for new service needs to be accepted. This strategy provides good control over the traffic and reduces the possibility of a breach because of service misconfiguration.

   In this lab we will follow the first approach. run the following commands on Perimeter-Firewall to add implicit deny rules to all `chains` in `filter` table, to block all traffic.

```
/ip firewall filter
add action=drop chain=forward comment="Drop all forwarding traffic"
add action=drop chain=input comment="Drop all input traffic"
add action=drop chain=output comment="Drop all output traffic"
```

   Run the following command to view the existing rules in the filter table.

```
/ip firewall filter print stats
```

   Modern firewalls supports both stateful and stateless packet filtering. We will configure rules of both types on the `Perimeter-Firewall`.

   (a) **Stateless Packet Filtering**

   i. Allow access to Web server in DMZ from public Internet (WAN)

```
/ip firewall filter add action=accept chain=forward in-interface=wan \
dst-address=10.10.2.80 dst-port=443,80 protocol=tcp \
comment="Allow access to Web server in DMZ from public Internet (WAN)"
```

   ii. Allow access to external DNS servers from LAN and DMZ

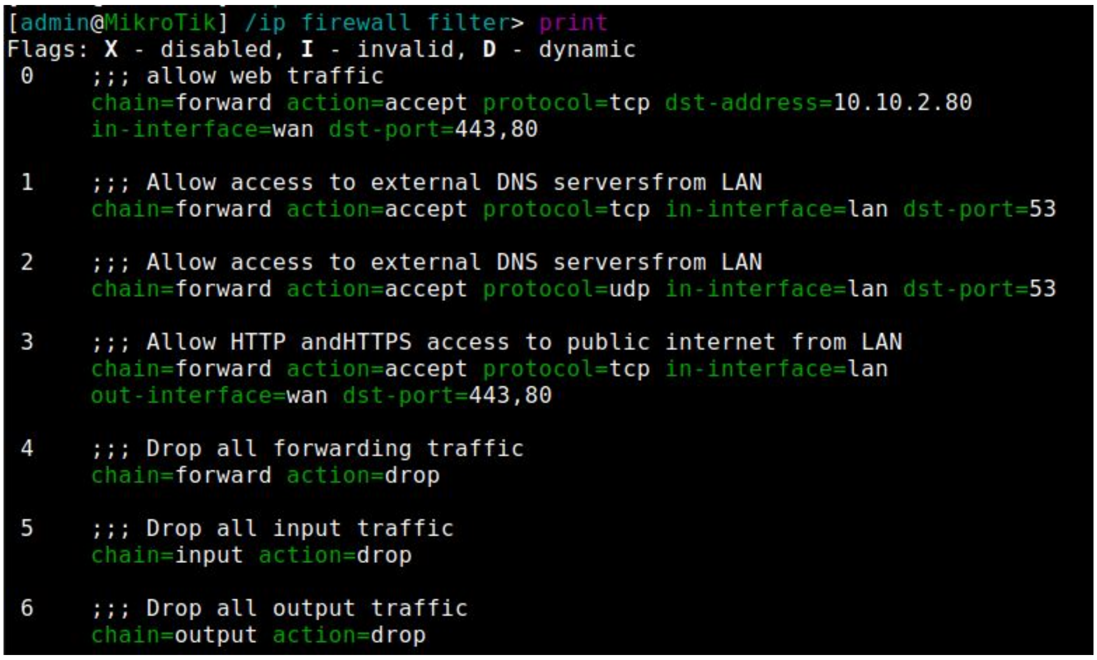
```
/ip firewall filter add action=accept chain=forward in-interface=lan \
dst-port=53 protocol=tcp comment="Allow access to external DNS \
servers from LAN"

/ip firewall filter add action=accept chain=forward in-interface=lan \
dst-port=53 protocol=udp comment="Allow access to external DNS servers \
from LAN"
```

   iii. Allow HTTP and HTTPS access to public internet from LAN

```
/ip firewall filter add action=accept chain=forward in-interface=lan \
out-interface=wan dst-port=443,80 protocol=tcp comment="Allow HTTP and \
HTTPS access to public internet from LAN"
```

   iv. Move the implicit deny rule to the end of the rule list

```
/ip firewall filter
print
move <current position> <destination position>
```



```
[admin@MikroTik] /ip firewall filter> print
Flags: X - disabled, I - invalid, D - dynamic
 0    ;;; allow web traffic
      chain=forward action=accept protocol=tcp dst-address=10.10.2.80
      in-interface=wan dst-port=443,80

 1    ;;; Allow access to external DNS serversfrom LAN
      chain=forward action=accept protocol=tcp in-interface=lan dst-port=53

 2    ;;; Allow access to external DNS serversfrom LAN
      chain=forward action=accept protocol=udp in-interface=lan dst-port=53

 3    ;;; Allow HTTP andHTTPS access to public internet from LAN
      chain=forward action=accept protocol=tcp in-interface=lan
      out-interface=wan dst-port=443,80

 4    ;;; Drop all forwarding traffic
      chain=forward action=drop

 5    ;;; Drop all input traffic
      chain=input action=drop

 6    ;;; Drop all output traffic
      chain=output action=drop
```

Figure 1: Firewall rules

(b) **Stateful Packet Filtering**

Using the above stateless filtering rules, we only allowed traffic flow one direction . These rules will not allow the reply traffic from the destination. For an example, a DNS request from LAN to an external DNS server will be allowed, however, the reply from the DNS server will not reach the requester as traffic is not allowed to that side. You can test this by sending a DNS query to Google DNS server from `Internal-Client`.

Now if you create another rule for reverse traffic using stateless filtering rules, anyone from public internet will be able to reach the LAN on DNS ports, which is not secure.

A securely implemented rule should only allow return traffic for a request initiated from LAN. This is where stateful filtering rules can be helpful. Use the following command to allow all return traffic for an established connection as well as the traffic required for connection establishment.

```
/ip firewall filter \
add action=accept chain=forward connection-state=established,related \
comment="Allow established and related connections"
```

Move this rule above all the ”Drop all” rules.

2. **Firewall Evasion with SSH Tunnels**

Some enterprise networks enforce egress filtering, blocking all HTTP and HTTPS traffic to public internet from internal network. In this task we will perform an firewall evasion technique to avoid such restrictions.

(a) Install Lynx browser on Internal-Client and access https://youtube.com. (some of these may already be installed)

```
apt update
apt install nano lynx openssh-client -y
lynx https://youtube.com
```

(b) Block all HTTP/HTTPS forwarding traffic on the Perimeter-Firewall by disabling the forwarding rule created.

```
\ip firewall filter
print
disable <rule-id>
```

(c) Install the following on the **External-Attacker**. We are going to use the **External-Attacker** as the SSH-Server. (some of these may already be installed)

```
apt update
apt install nano openssh-server -y
```

Change the SSH port of SSH Server by editing `/etc/ssh/sshd_config` and modifying `Port 22` as `Port 53`. Add the below lines to the file to allow SSH using Root user. (this line should already be there)

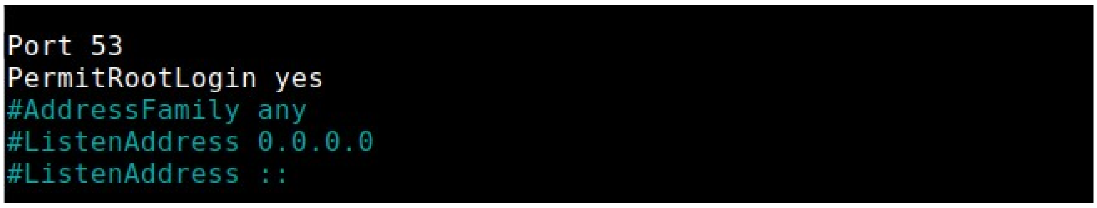
```
PermitRootLogin yes
```

Reset the password of the root user using the below command. You will have to repeat the new password to confirm. Make sure to remember the password you set.

```
passwd
```

Restart SSH server using the below command.

```
service ssh restart
```



```
Port 53
PermitRootLogin yes
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
```

Figure 2: SSH on Port 53 and Root Login Permitted

(d) Since TCP port 53 is allowed from LAN to public internet, if tunnel the HTTPS traffic for `youtube.com` through the port TCP 53 we can evade the firewall restrictions.

Use the following command on the Internal-Client to create a SSH tunnel to the SSH-Server (use the password you set in the previous step). This command connects to the `SSH-Server` with SSH and bind the local port 443 with the SSH tunnel, the server will forward the traffic of SSH tunnel to `youtube.com` on TCP port 53.

```
ssh root@<ssh-server-IP> -p 53 -N -f -L 443:youtube.com:443
```

The `-N` and `-f` options will avoid getting a SSH shell to the remote machine and will keep the session in the background.

(e) Now we need to configure proxy on `Lynx` to send the traffic to the SSH tunnel. Run the below command to edit Lynx configuration file and use `Ctrl+W` to find the line `https_proxy`.

```
nano /etc/lynx/lynx.cfg
```

Un-comment the line and change it to the below.

```
https_proxy:https://127.0.0.1:443
```

Now execute the below to command to access youtube.com again. You should be able to visit the website. We are visiting youtube.com using SSH-Server via the SSH tunnel.

```
lynx https://youtube.com
```

Note: You may need to press Y a couple of times to accept the SSL certificate warnings.

Analyse the traffic between the two firewalls using Wireshark. Discuss your observations.

Group Discussion and Reflection:

1. What measures can be used to avoid the above evasion technique?
2. What are the best practices for writing good firewall rules?
3. Discuss the uses of `nat, mangle` and `raw` tables.

Further Reading: Mikrotik Documentation.