Firewall Exploration Lab

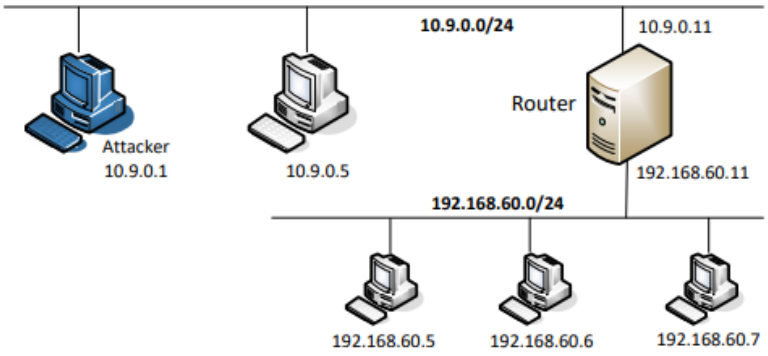
Introduction

The learning objective of this lab is two-fold: learning how firewalls work, and setting up a simple firewall for a network. Students will first implement a simple stateless packet-filtering firewall, which inspects packets, and decides whether to drop or forward a packet based on firewall rules. Through this implementation task, students can get the basic ideas on how firewall works.

Linux already has a built-in firewall, also based on netfilter. This firewall is called iptables. Students will be given a simple network topology, and are asked to use iptables to set up firewall rules to protect the network. Students will also be exposed to several other interesting applications of iptables.

Environment Setup Using Containers

In this lab, we need to use multiple machines. Their setup is depicted in the following figure. We will use containers to set up this lab environment.



Container Setup and Commands

Please download the Labsetup.zip file to your VM from the lab's website, unzip it, enter the **Labsetup** folder, and use the **docker-compose.yml** file to set up the lab environment. Detailed explanation of the content in this file and all the involved **Dockerfile** can be found from the user manual, which is linked.

Task 1: Implementing a Simple Firewall

The generated kernel module is in **hello.ko**. You can use the following commands to load the module, list all modules, and remove the module. Also, you can use "$**modinfo hello.ko**" to show information about a Linux Kernel module.

1. Compile the kernel module code to prepare it for insertion into the kernel. Ensure that the kernel module is correctly compiled and ready for use.
   Command: $**make**
   ```
   [04/15/24]seed@VM:~/.../kernel_module$ make
   ```

2. Check loaded modules with the command: **lsmod** and kernel messages with the command: **dmesg**.

Confirm the correct loading and operation of the kernel module

```
[04/15/24]seed@VM:~/.../kernel_module$ lsmod
```

3. And now running the dmesg command to show the diagnostics messages.

```
[04/15/24]seed@VM:~/.../kernel_module$ dmesg
```

Task 1.A Implementing a simple Kernel Module

The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded; when the module is removed from the kernel, it prints out "Bye-bye World!". The messages are not printed out on the screen; they are actually printed into the /var/log/syslog file. You can use "dmesg" to view the messages.

Load and unload the kernel module
Command: **insmod hell.ko**

```
[04/15/24]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[04/15/24]seed@VM:~/.../kernel_module$
```

Use the grep command to show the info about the kernel module.

```
[04/15/24]seed@VM:~/.../kernel_module$ lsmod | grep -i hello
```

And now using **rmmod** command to remove the kernel module info.

And once again when I use the grep command to check the output the hello message created kernel module isn't available anymore.

```
[04/15/24]seed@VM:~/.../kernel_module$ sudo rmmod hello
[04/15/24]seed@VM:~/.../kernel_module$ lsmod | grep -i hello
```

The generated in the **dmesg** window is stated as follows:

Use the command:

$dmesg → $dmesg -k -e → $dmesg -k -w

```
[04/15/24]seed@VM:~/.../kernel_module$ sudo dmesg --clear
[04/15/24]seed@VM:~/.../kernel_module$ dmesg
[04/15/24]seed@VM:~/.../kernel_module$ dmesg -k -e
[04/15/24]seed@VM:~/.../kernel_module$ dmesg -k -w
```

Use **modinfo** to get details about the kernel module which provides the information about the module's parameters and dependencies

```
[04/15/24]seed@VM:~/.../kernel_module$ modinfo video
```

Info of the **hello.ko** file is as follows:

```
[04/15/24]seed@VM:~/.../kernel_module$ modinfo hello.ko
```

Tack 1.B Implement a simple Firewall using Netfilter

In this task, we will write our packet filtering program as an LKM, and then insert in into the packet processing path inside the kernel. This cannot be easily done in the past before the netfilter was introduced into the Linux.

Task a: Compile the sample code using the provided Makefile. Load it into the kernel, and demonstrate that the firewall is working as expected. You can use the following command to generate UDP packets to 8.8.8.8, which is Google's DNS server. If your firewall works, your request will be blocked; otherwise, you will get a response. dig @8.8.8.8 www.example.com

The complete sample code is called **seedFilter.c**, which is included in the lab setup files (inside the Files/packet_filter folder). Please do the following tasks (do each of them separately):

1. Ensure if the network is functioning before applying firewall rules.
   Hence, verify network connectivity by pinging towards the destination.
   Command: $**ping 8.8.8.8** and **$ping example.com**

   ```
   [04/15/24]seed@VM:~/.../packet_filter$ ping 8.8.8.8
   ```

   We can see that we are getting the positive results

   ```
   [04/15/24]seed@VM:~/.../packet_filter$ ping example.com
   ```

2. Now use **dig** to send DNS queries and generate UDP traffic.
   It tests network functionality before firewall rule implementation.
   Command: **dig @8.8.8.8 www.example.com**

   ```
   [04/15/24]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com
   ```

3. Now execute the makefile for the **seedFilter.c** file.
   Run the following commands: **make**

   ```
   [04/15/24]seed@VM:~/.../packet_filter$ make
   ```

4. After running the make command, run the following command:
   $sudo insmod seedFilter.ko
   And check what is the output generated in the **dmesg** dialogue box

   ```
   [04/15/24]seed@VM:~/.../kernel_module$ dmesg
   [04/15/24]seed@VM:~/.../kernel_module$ dmesg -k -e
   [04/15/24]seed@VM:~/.../kernel_module$ dmesg -k -w
   ```

5. Use the **grep** command to search the output for the term "seed"
   Command: $**lsmod | grep -i seed**
   This output indicates that the kernel module named "seedFilter" is loaded into the kernel, it is 16,384 bytes in size, and it is currently not being used by any other modules.

   ```
   [04/15/24]seed@VM:~/.../packet_filter$ lsmod | grep -i seed
   ```

6. Use **dig** command for querying DNS name servers.
   Command: $**dig @8.8.8.8** www.example.com

```
[04/15/24]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com
```

   DNS lookup for www.example.com using Google's DNS server at 8.8.8.8. The output shows that the DNS server was found, but the connection timed out, meaning no response was received.

7. Use **$rmmod seedFilter** after checking that we have successfully implemented a firewall.

```
[04/15/24]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
```

   Now check the output on the **dmseg** dialogue box and mention what does this log entry signify.

Creating hooks for the application

Task b: Hook the **printInfo** function to all of the **netfilter** hooks. Here are the macros of the hook numbers. Using your experiment results to help explain at what condition will each of the hook function be invoked.

Implement two more hooks to achieve the following:

(1) preventing other computers to ping the VM
(2) preventing other computers to telnet into the VM. Please implement two different hook functions, but register them to the same netfilter hook. You should decide what hook to use. Telnet's default port is TCP port 23.

1. Go to host-A server and try to ping the server
   Command: **$docksh hostA-10.9.0.5**
   /# ping 10.9.0.1

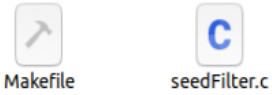
```
[06/08/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@422534e71442:/# ping 10.9.0.1
```

2. Trying to log into telnet server

```
root@422534e71442:/# telnet 10.9.0.1
```

   Mention your observation for ping and telnet requests.

3. A seedFilter.c file has been provided to you, place this file inside the path //Labsetup/Files/Packet_Filter



   The codeblock for **seedFilter.c** is here as follows:

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>
#include <linux/icmp.h>

static struct nf_hook_ops hook1a, hook1b, hook1c, hook1d, hook1e, hook2, hook3, hook4;


unsigned int blocktelnet(void *priv, struct sk_buff *skb,
                         const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct tcphdr *tcph;

   u16  port   = 23;
   char ip[16] = "10.9.0.1";
```

4. This code snippet follows:

```c
hook2.hook = blockUDP;
hook2.hooknum = NF_INET_POST_ROUTING;
hook2.pf = PF_INET;
hook2.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook2);

hook3.hook = blockICMP;
hook3.hooknum = NF_INET_PRE_ROUTING;
hook3.pf = PF_INET;
hook3.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook3);

hook4.hook = blocktelnet;
hook4.hooknum = NF_INET_PRE_ROUTING;
hook4.pf = PF_INET;
hook4.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook4);
```

   Mention what does the **hook3** and **hook4** specify.

5. Use **make** command to create kernel runtime codes

```
[06/08/24]seed@VM:~/.../packet_filter$ make
```

6. Use **ls** command to check the files generated by the make command:

```
[06/08/24]seed@VM:~/.../packet_filter$ ls
```

7. Use insmod command to load the module
   Command: $**sudo insmod seedFilter.ko**

```
[06/08/24]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
```

   Mention what is the output generated in the **dmseg** dialogue box.

8. Now try to ping the VM from your host (which is hostA-10.9.0.5)
   Command: **/# ping 10.9.0.1**

```
root@422534e71442:/# ping 10.9.0.1
```

9. In the **dmesg** dialogue box, check the output and mention your findings.

10. Similarly with the telnet,
    Once you try to access via telnet to VM (ip address to be 10.9.0.1)
    Command: **/# telnet 10.9.0.1**

    ```
    root@422534e71442:/# telnet 10.9.0.1
    ```

    Are you able to telnet? Why?

11. In the **dmesg** dialogue box, check the output and mention your findings.

Task 2: Experimenting with Stateless Firewall Rules

In the previous task, we had a chance to build a simple firewall using netfilter. Actually, Linux already has a built-in firewall, also based on netfilter. This firewall is called iptables. Technically, the kernel part implementation of the firewall is called **Xtables**, while iptables is a user-space program to configure the firewall. However, iptables is often used to refer to both the kernel-part implementation and the user-space program.

You can find the manual of iptables by typing **$man iptables**
Background of iptables, in this task, we will use iptables to set up a firewall. The iptables firewall is designed not only to filter packets, but also to make changes to packets. To help manage these firewall rules for different purposes, iptables organizes all rules using a hierarchical structure: table, chain, and rules.

The rule is the most complicated part of the iptables command. We will provide additional information later when we use specific rules. In the following, we list some commonly used commands:

1. List all the rules in a table (without line number)
   Command: **iptables -t nat -L -n**
   ```
   [06/08/24]seed@VM:~/.../Labsetup$ sudo iptables -t nat -L -n
   ```

2. List all the rules in a table (with line number)
   Command: **iptables -t filter -L -n --line-numbers**
   ```
   [06/08/24]seed@VM:~/.../Labsetup$ sudo iptables -t filter -L -n --line-numbers
   ```

Docker relies on **iptables** to manage the networks it creates, so it adds many rules to the **nat** table. When we manipulate **iptables** rules, we should be careful not to remove Docker rules. For example, it will be quite dangerous to run the "**iptables -t nat -F**" command, because it removes all the rules in the **nat** table, including many of the Docker rules. That will cause trouble to Docker containers. Doing this for the **filter** table is fine, because Docker does not touch this table.

Task 2.A: Protecting the Router

1. In this task, we will set up rules to prevent outside machines from accessing the router achine, except ping. Please execute the following **iptables** command on the router container, and then try to access it from **10.9.0.5**.
   a) Access the Router Container:
   Command: **$docksh seed-router**

   Now try to ping the HostA having ip address as: 10.9.0.5
   And we can see that we are able to ping it.

```
[06/08/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@28d666d49167:/# ping 10.9.0.5
```

   b) And telnet the same, we see that we are able to connect to the HostA.

```
root@28d666d49167:/# telnet 10.9.0.5
```

2. Set Up iptables Rules:
   Inside the router container, you will set up iptables rules to block all incoming connections except for ICMP (ping) requests.
   Execute the following commands:

   a) Flush all existing rules:
   Command: $**iptables -F** and **$iptables -X**

```
[06/08/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@28d666d49167:/# iptables -F
root@28d666d49167:/# iptables -X
```

   b) Default policy to drop all incoming connections:
   Command:
   $iptables -P INPUT DROP
   $iptables -P FORWARD DROP
   $iptables -P OUTPUT ACCEPT

```
root@28d666d49167:/# iptables -P INPUT DROP
root@28d666d49167:/# iptables -P FORWARD DROP
root@28d666d49167:/# iptables -P OUTPUT ACCEPT
root@28d666d49167:/#
```

   c) Allow incoming and outgoing ping requests
   Command: **iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT**
   iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

```
root@28d666d49167:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@28d666d49167:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@28d666d49167:/#
```

   d) Set default policy to drop all incoming connections

Command: **iptables -P INPUT DROP**

```
root@28d666d49167:/# iptables -P INPUT DROP
```

e) Set default policy to drop all outgoing connections
Command: **iptables -P OUTPUT DROP**

```
root@28d666d49167:/# iptables -P OUTPUT DROP
```

f) To check the status of the Firewall rules:
Run the following command: iptables -t filter -L -n

```
root@a3126f91c4db:/# iptables -t filter -L -n
```

3. Access the Host container
Command: **$docksh hostA-10.9.0.5**

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
```

4. Test Connectivity from hostA (10.9.0.5) towards the router (10.9.0.11)
Command: **ping 10.9.0.11**

```
root@53636f154f6c:/# ping 10.9.0.11
```

Are you able to ping?

5. Try to use the telnet command and check the result
Command: **telnet 10.9.0.11**

```
root@53636f154f6c:/# telnet 10.9.0.11
```

Are you able to Telnet?

6. Cleanup

Before moving on to the next task, please restore the filter table to its original state by running the following commands:

a) Command: **iptables -F**
b) Command: **iptables -P OUTPUT ACCEPT**
c) Command: **iptables -P INPUT ACCEPT**

```
root@a3126f91c4db:/# iptables -F
root@a3126f91c4db:/# iptables -P OUTPUT ACCEPT
root@a3126f91c4db:/# iptables -P INPUT ACCEPT
root@a3126f91c4db:/#
```

7. To check the status, you can run the following command
   Command: **iptables -t filter -L -n**

```
root@a3126f91c4db:/# iptables -t filter -L -n
```

Task 2.B: Protecting the Internal Network

In this task, we will set up firewall rules on the router to protect the internal network 192.168.60.0/24. We need to use the **FORWARD** chain for this purpose.

We want to implement a firewall to protect the internal network. More specifically, we need to enforce the following restrictions on the ICMP traffic:

a) Outside hosts cannot ping internal hosts.
b) Outside hosts can ping the router.
c) Internal hosts can ping outside hosts.

   Run the following commands:

```
root@a3126f91c4db:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
root@a3126f91c4db:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
root@a3126f91c4db:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
root@a3126f91c4db:/# iptables -P FORWARD DROP
```

   Thus, the final rules integrated are as follows:

```
root@a3126f91c4db:/# iptables -t filter -L -n --line-numbers
```

   Process:

   a) Outside hosts cannot ping internal hosts.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
```

   b) Outside hosts can ping the router.

```
root@53636f154f6c:/# ping 10.9.0.11
```

   c) Internal hosts can ping outside hosts.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
```

Remember: Before moving on to the next task, please restore the filter table to its original state.

Task 2.C: Protecting Internal Servers

In this task, we want to protect the TCP servers inside the internal network (192.168.60.0/24). More specifically, we would like to achieve the following objectives.

a) All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.
b) Outside hosts cannot access other internal servers.
c) Internal hosts can access all the internal servers.
d) Internal hosts cannot access external servers.

Run the following commands:

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
root@a3126f91c4db:/# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
root@a3126f91c4db:/# iptables -P FORWARD DROP
```

Thus, the final rules integrated are as follows:

```
root@a3126f91c4db:/# iptables -t filter -L -n --line-numbers
```

a) Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# telnet 192.168.60.5
```

b) Outside hosts cannot access other internal servers.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# telnet 192.168.60.6
```

c) Internal hosts can access all the internal servers.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# telnet 192.168.60.6
root@3df15df73e87:/# telnet 192.168.60.7
Trying 192.168.60.7...
```

d) Internal hosts cannot access external servers

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# telnet 10.9.0.5
[06/09/24]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@c181e6851621:/# telnet 10.9.0.5
[06/09/24]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@b173e9034362:/# telnet 10.9.0.5
```

Mention the output for the telnet connection. Are you able to telnet ?

Remember: Before moving on to the next task, please restore the filter table to its original state.

Task 3: Connection Tracking and Stateful Firewall

In the previous task, we have only set up stateless firewalls, which inspect each packet independently. However, packets are usually not independent; they may be part of a TCP connection, or they may be ICMP packets triggered by other packets. Treating them independently does not take into consideration the context of the packets, and can thus lead to inaccurate, unsafe, or complicated firewall rules. For example, if we would like to allow TCP packets to get into our network only if a connection was made first, we cannot achieve that easily using stateless packet filters, because when the firewall examines each individual TCP packet, it has no idea whether the packet belongs to an existing connection or not, unless the

firewall maintains some state information for each connection. If it does that, it becomes a stateful firewall.

Task 3.A Experiment with the Connection Tracking

To support stateful firewalls, we need to be able to track connections. This is achieved by the **conntrack** mechanism inside the kernel.

In this task, we will conduct experiments related to this module, and get familiar with the connection tracking mechanism. In our experiment, we will check the connection tracking information on the router container. This can be done using the following

Command: **conntrack -L**

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# conntrack -L
```

1. **ICMP experiment:** Run the following command and check the connection tracking information on the router.

   On 10.9.0.5, send out ICMP packets

   Command: **ping 192.168.60.5**

   ```
   root@a3126f91c4db:/# ping 192.168.60.5
   ```

   Now run the conntrack command to check the flow entries

   Command: **conntrack -L**

   ```
   root@a3126f91c4db:/# conntrack -L
   ```

   To check the ICMP timeout status, run the following command:

   cat /proc/sys/net/netfilter/nf_conntrack_icmp_timeout

   ```
   root@a3126f91c4db:/# cat /proc/sys/net/netfilter/nf_conntrack_icmp_timeout
   ```

2. **UDP experiment:** Run the following command and check the connection tracking information on the router.

   On 192.168.60.5, start a netcat UDP server

   ```
   [06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
   root@3df15df73e87:/# nc -lu 9090
   ```

   Now, On 10.9.0.5, send out UDP packets

   ```
   [06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
   root@53636f154f6c:/# nc -u 192.168.60.5 9090
   Hello
   ```

   Note: run the following command and make sure you press **Shift+Enter** key to type a message. And then press the **Enter** key.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# nc -lu 9090
Hello
```

To check the UDP timeout status, run the following commands:

cat /proc/sys/net/netfilter/nf_conntrack_udp_timeout

cat /proc/sys/net/netfilter/nf_conntrack_udp_timeout_stream

```
root@53636f154f6c:/# cat /proc/sys/net/netfilter/nf_conntrack_udp_timeout
root@53636f154f6c:/# cat /proc/sys/net/netfilter/nf_conntrack_udp_timeout_stream
```

3. **TCP experiment:** Run the following command and check the connection tracking information on the router.

   On 192.168.60.5, start a netcat TCP server

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# nc -l 9090
```

   On 10.9.0.5, send out TCP packets

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# nc 192.168.60.5 9090
Hello..
```

   Note: run the following command and make sure you press **Shift+Enter** key to type a message. And then press the **Enter** key.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# nc -l 9090
Hello..
```

   To check the TCP timeout status, run the following commands:

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_established

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_syn_sent

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_syn_recv

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_fin_wait

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_close_wait

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_last_ack

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_time_wait

   cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_close

Remember: Before moving on to the next task, please restore the filter table to its original state.

Task 3.B: Setting Up a Stateful Firewall

Now we are ready to set up firewall rules based on connections. In the following example, the "**-m conntrack**" option indicates that we are using the **conntrack** module, which is a very important module for **iptables**; it tracks connections, and **iptables** replies on the tracking information to build stateful firewalls. The **--ctsate ESTABLISHED,RELATED** indicates that whether a packet belongs to an **ESTABLISHED** or **RELATED** connection. The rule allows TCP packets belonging to an existing connection to pass through.

1. First check the ip address of the eth0 and eth1
   Command: **ifconfig**

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# ifconfig
```

2. Add the rules to accept incoming SYN packet by running the following commands:
   Rule 1:
   Command: **iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT**
   This rule allows new TCP connections (indicated by --syn) to port 23 (Telnet) on the host 192.168.60.5 if they are coming from the external network interface eth0.

   Rule 2:
   Command: **iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT**
   This rule allows new TCP connections (indicated by --syn) coming from the internal network interface eth1.

   Rule 3:
   Command: **iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT**
   This rule allows traffic for already established and related TCP connections.

   Rule 4:
   Command**: iptables -A FORWARD -p tcp -j DROP**
   This rule drops all other TCP traffic that doesn't match the above rules.

   Rule 5:
   Command: **iptables -P FORWARD ACCEPT**
   This rule sets the default policy for the FORWARD chain to ACCEPT, meaning any packet that doesn't match any rule in the FORWARD chain will be accepted.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntr
ack --ctstate NEW   -j ACCEPT
root@a3126f91c4db:/# iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW   -j ACCE
PT
root@a3126f91c4db:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED   -j AC
CEPT
root@a3126f91c4db:/# iptables -A FORWARD -p tcp -j DROP
root@a3126f91c4db:/# iptables -P FORWARD ACCEPT
root@a3126f91c4db:/#
root@a3126f91c4db:/# iptables -t filter -L -n --line-numbers
```

a) Check whether you can access external host (10.9.0.5) from internal server (192.168.0.5)

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# telnet 10.9.0.5
```

b) Check whether if you can access the internal server (192.168.0.5) from an external host (10.9.0.5)

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# telnet 192.168.60.5
```

c) As we have set the rules for the dest. Ip 192.168.60.5, try to access the internal server (192.168.60.6 or 192.168.60.7) from the external host (10.9.0.5).

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# telnet 192.168.60.6
```

d) Let's see if the added rule allows internal hosts to visit any external server.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# nc -u 10.9.0.5 9090
hello
```

And listen to the external server (10.9.0.5)

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# nc -ul 9090
hello
```

Remember: Before moving on to the next task, please restore the filter table to its original state.

Task 4: Limiting Network Traffic

In addition to blocking packets, we can also limit the number of packets that can pass through the firewall. This can be done using the **limit** module of **iptables**. In this task, we will use this module to limit how many packets from **10.9.0.5** are allowed to get into the internal network. You can use "**iptables -m limit -h**" to see the manual.

1. Run the command: **iptables -m limit -h**

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# iptables -m limit -h
```

2. Now run the following command:
   iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j
 ACCEPT
```

3. Now try to ping the external host from the internal server and check the status of the sequence.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# ping 10.9.0.5
```

   You can see that there isn't any drop in the icmp sequence.

4. Run the following command into the seed-router:
   iptables -A FORWARD -s 10.9.0.5 -j DROP

```
root@a3126f91c4db:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@a3126f91c4db:/#
```

5. Now check the status of the icmp_sequence.

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# ping 10.9.0.5
```

   You can observe that the icmp_sequence drops some of the packets and doesn't transmit all the packets.

Remember: Before moving on to the next task, please restore the filter table to its original state.

Task 5: Load Balancing

The iptables is very powerful. In addition to firewalls, it has many other applications. We will not be able to cover all its applications in this lab, but we will experiment with one of the applications, load balancing. In this task, we will use it to load balance three UDP servers running in the internal network.

Let's first start the server on each of the hosts: 192.168.60.5, 192.168.60.6, and 192.168.60.7 with the command: **nc -luk 8080**

1. Host 192.168.60.5

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@3df15df73e87:/# nc -luk 8080
```

2. Host 192.168.60.6

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@c181e6851621:/# nc -luk 8080
```

3. Host 192.168.60.7

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@b173e9034362:/# nc -luk 8080
```

We can use the statistic module to achieve load balancing. You can type the following command to get its manual. You can see there are two modes: random and nth. We will conduct experiments using both of them.

Task 5.A: Using the nth mode (round-robin)

On the router container, we set the following rule, which applies to all the UDP packets going to port **8080**. The **nth** mode of the **statistic** module is used; it implements a round-robin load balancing policy: for every three packets, pick the packet 0 (i.e., the first one), change its destination IP address and port number to **192.168.60.5** and **8080**, respectively. The modified packets will continue on its journey.

1. Command:
   iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --e
very 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
```

2. On 10.9.0.5, run the command:

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
```

3. Now let's add more rules to the router container, so all the three internal hosts get the equal number of packets.
   Command: **iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080**

   This rule will change the destination of every 2nd UDP packet destined for port 8080 to 192.168.60.6:8080. This is useful for distributing traffic in a round-robin fashion.

   Command: **iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 1 --packet 0 -j DNAT --to-destination 192.168.60.7:8080**

   This rule will change the destination of every UDP packet destined for port 8080 to 192.168.60.7:8080.

4. Run the commands and see how the rules have been configured:

```
root@a3126f91c4db:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth -
very 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080
root@a3126f91c4db:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth -
very 1 --packet 0 -j DNAT --to-destination 192.168.60.7:8080
root@a3126f91c4db:/# iptables -t nat -L -n --line-numbers
```

5. Now run the command again on 10.9.0.5;

a) Message as “hello”

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
```

b) Message as “hello hello hello”

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo "hello hello hello" | nc -u 10.9.0.11 8080
```

c) Again, message as “hello hello hello”

```
[06/09/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo hello | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo "hello hello hello" | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo "hello hello hello" | nc -u 10.9.0.11 8080
```

d) Message as hello1

```
root@53636f154f6c:/# echo hello1 | nc -u 10.9.0.11 8080
```

e) Message as hello2

```
root@53636f154f6c:/# echo hello2 | nc -u 10.9.0.11 8080
```

f) Message as hello3

```
root@53636f154f6c:/# echo hello3 | nc -u 10.9.0.11 8080
```

Remember: Before moving on to the next task, please restore the filter table to its original state.

Task 5.B: Using the random mode

Let’s use a different mode to achieve the load balancing. The following rule will select a matching packet with the probability P. You need to replace P with a probability number.

Run the following commands in the seed-router.

Command:

1. **iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.5:8080**
2. **iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-destination 192.168.60.6:8080**

3. **iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1.0 -j DNAT --to-destination 192.168.60.7:8080**

Please use this mode to implement your load balancing rules, so each internal server get roughly the same amount of traffic

```
[06/10/24]seed@VM:~/.../Labsetup$ docksh seed-router
root@a3126f91c4db:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random
--probability 0.33 -j DNAT --to-destination 192.168.60.5:8080
root@a3126f91c4db:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random
--probability 0.5 -j DNAT --to-destination 192.168.60.6:8080
root@a3126f91c4db:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random
--probability 1.0 -j DNAT --to-destination 192.168.60.7:8080
root@a3126f91c4db:/# iptables -t nat -L -n --line-numbers
```

1. Run the following message/or of your own:

```
[06/10/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# echo "H11" | nc -u 10.9.0.11 8080
```

2. Message:

```
[06/10/24]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@53636f154f6c:/# echo "H11" | nc -u 10.9.0.11 8080
^C
root@53636f154f6c:/# echo "Hey! How's everything?" | nc -u 10.9.0.11 8080
```

You can try certain set of messages of your own and can check the implementation of load balancing according to the probability.

Deliverables

Please submit the lab report in a single document named *CS690-LAB1-yourlastname*. Please submit a DOC document and/or a PDF file in case the format is not compatible across the platform. The lab report should include:

1. Title, author(s)
2. Table of Content
3. The detailed steps and results using text description and/or screenshots that answer the above questions and demonstrate your lab progress.
4. A summary of your own reflection of the lab exercise, such as:
   a) What is the purpose of the lab in your own words?
   b) What do you learn? Do you achieve the objectives?
   c) Is this lab hard or easy? Are the lab instructions clear?
   d) Any other feedback?