

# CSCI262

## LAB 9 - FIREWALL EXPLORATION

### Description

This lab will introduce you how Linux built-in firewall, *iptables*, works. You will be given a simple network topology, and are asked to use *iptables* to set up firewall rules to protect the network. You also explore several amazing applications of *iptables* tool such as *conntrack* (connection tracking), *limit* (traffic monitoring), and *statistic* (load balancing).

### Learning Outcomes

By doing this hands-on configuration, you should be able to:

1. Investigate how *iptables* (Linux built-in-firewall) works
2. Use *iptables* to set up firewall rules
3. Implement special application connection track, traffic monitor, and load balance for the firewall

## Firewall Exploration

---

### Table of Contents

Description .....	1
Learning Outcomes.....	1
0. Environment Setup .....	3
1. TASK 1: Stateless Firewall Rules .....	5
1.1. What is iptables.....	5
1.2. How to use <i>iptables</i> .....	6
1.3. Task 1.A: Protecting the Router.....	8
1.4. Task 1.B: Protecting the Internal Network .....	15
1.5. Task 1.C: Protecting Internal Servers .....	16
2. TASK 2: Connection Tracking and Stateful Firewall .....	18
2.1. Task 2.A: Experiment with the Connection Tracking .....	18
2.2. Task 2.B: Setting Up a Stateful Firewall .....	19
Reference: .....	21

### Login Information:

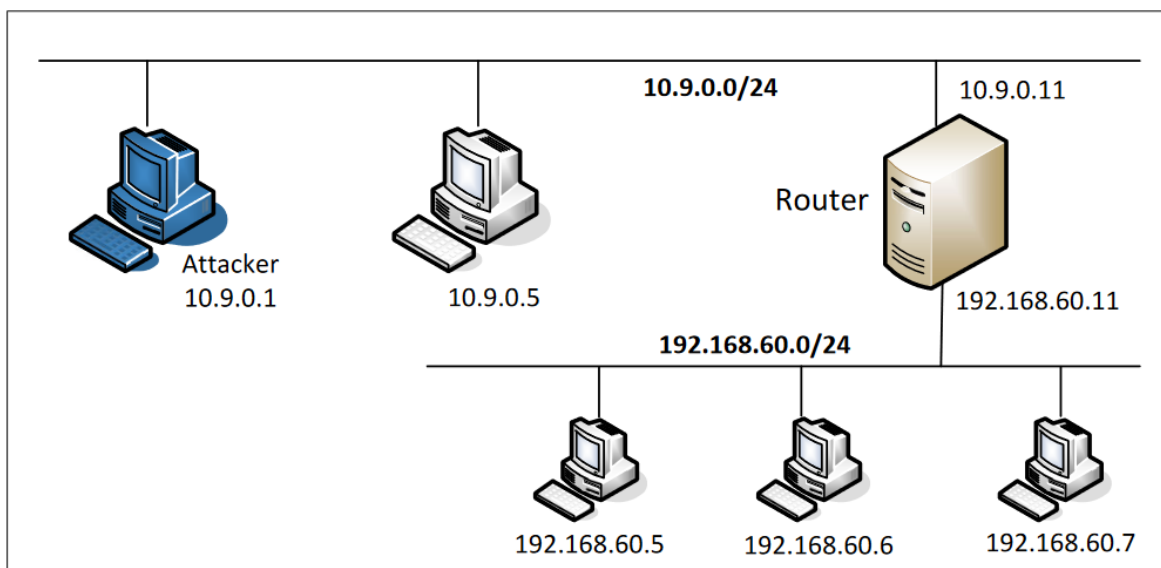
We will use the following account to log in to the virtual machine:

- ✓ Account ID: **seed**
- ✓ Password : **dees**

## Firewall Exploration

### 0. Environment Setup

In this lab, we will use Docker to build 5 to 6 virtual machines as shown in **Figure 1**. To build and enable the all Docker VM, please do as you did from Lab 6. To do this lab, please navigate to the folder /Desktop/Labsetup-Firewall\_Exploration. Please remember turn off all running containers (`docker kill $(docker ps -q)`) before we start to build (`dcbuild`) and enable all containers (`dcup`).



**Figure 1:** Network topology and IP plan for Firewall Exploration Lab

Then we can check the list of running containers by using `dockps` shown as the below figure.

```

seed@VM: ~/.../Labsetup-Firewall_Exploration
seed@VM: ~/.../Labsetup-Firewall_Explor... x seed@VM: ~/.../Labsetup-Firewall_Explor... x
[09/16/22]seed@VM:~/.../Labsetup-Firewall_Exploration$ dockps
d04328621085 host3-192.168.60.7
4028ba3c4da9 host2-192.168.60.6
f9108d390c70 seed-router
66b1a6a2b8a0 host1-192.168.60.5
c37649050951 hostA-10.9.0.5
[09/16/22]seed@VM:~/.../Labsetup-Firewall_Exploration$

```

**Explain about the network:**

## Firewall Exploration

---

You can see Figure 1, we have two networks (internal and external). Router (*seed-router*) has two network interface, *eth0* connects to external network (10.9.0.0/24), and *eth1* connects to the internal network (192.168.90.0/24). This router is configured for NAT between these two networks.

For external network, we have host A (10.9.0.5). For internal network, we have Host 1 (192.168.60.5), Host 2 (192.168.60.6) Host 3 (192.168.60.7)

Now we will practice the real tasks

## 1. TASK 1: Stateless Firewall Rules

Basically, 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.

### 1.1. What is iptables

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. There are several tables, each specifying the main purpose of the rules as shown in **Table 1**. For example, rules for packet filtering should be placed in the filter table, while rules for making changes to packets should be placed in the *nat* or *mangle* tables.

Each table contains several chains, each of which corresponds to a *netfilter* hook. Basically, each chain indicates where its rules are enforced. For example, rules on the *FORWARD* chain are enforced at the *NF\_INET\_FORWARD* hook, and rules on the *INPUT* chain are enforced at the *NF\_INET\_LOCAL\_IN* hook.

Each chain contains a set of firewall rules that will be enforced. When we set up firewalls, we add rules to these chains. For example, if we would like to block all incoming telnet traffic, we would add a rule to the *INPUT* chain of the filter table. If we would like to redirect all incoming telnet traffic to a different port on a different host, basically doing port forwarding, we can add a rule to the *INPUT* chain of the *mangle* table, as we need to make changes to packets.

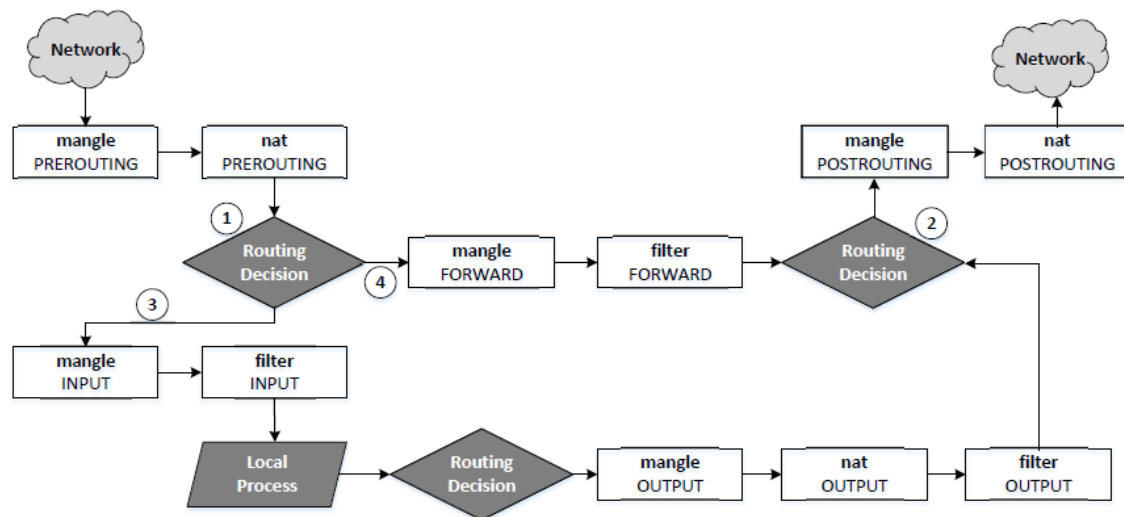
In summary, *iptables* is a simple *Linux* firewall using *netfilter*. The *iptables* uses chains which correspond to *netfilter* hooks. These chains belong to tables which the user can use to specify what actions to take when processing packets.

**Table 1:** *iptables* Tables and Chains

## Firewall Exploration

Table	Chain	Functionality
filter	INPUT FORWARD OUTPUT	Packet filtering
nat	PREROUTING INPUT OUTPUT POSTROUTING	Modifying source or destination network addresses
mangle	PREROUTING INPUT FORWARD OUTPUT POSTROUTING	Packet content modification

Apart from lecture knowledge, to know more about the background of iptables components, please see Figure 2 and read the below link.



**Figure 2:** Network packet traversal through iptables

<https://linuxhint.com/control-network-traffic-using-iptables/>

## 1.2. How to use iptables

## Firewall Exploration

---

To add rules to the chains in each table, we use the *iptables* command, which is a quite powerful command. You can find the manual of *iptables* by typing "*man iptables*" or easily find many tutorials from online. What makes *iptables* complicated is the many command-line arguments that we need to provide when using the command. However, if you understand the structure of these command-line arguments, we will find out that the command is not that complicated.

In a typical *iptables* command, we add a rule to or remove a rule from one of the chains in one of the tables, so we need to specify a table name (the default is filter), a chain name, and an operation on the chain. After that, we specify the rule, which is basically a pattern that will be matched with each of the packets passing through. If there is a match, an action will be performed on this packet. The general structure of the command is depicted in the following:

```
iptables -t <table> -<operation> <chain> <rule> -j <target>
-----
          Table           Chain       Rule       Action
```

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

```
// List all the rules in a table (without line number)
iptables -t nat -L -n

// List all the rules in a table (with line number)
iptables -t filter -L -n --line-numbers

// Delete rule No. 2 in the INPUT chain of the filter table
iptables -t filter -D INPUT 2

// Drop all the incoming packets that satisfy the <rule>
iptables -t filter -A INPUT <rule> -j DROP
```

**NOTE:** 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

## Firewall Exploration

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.

### 1.3. Task 1.A: Protecting the Router

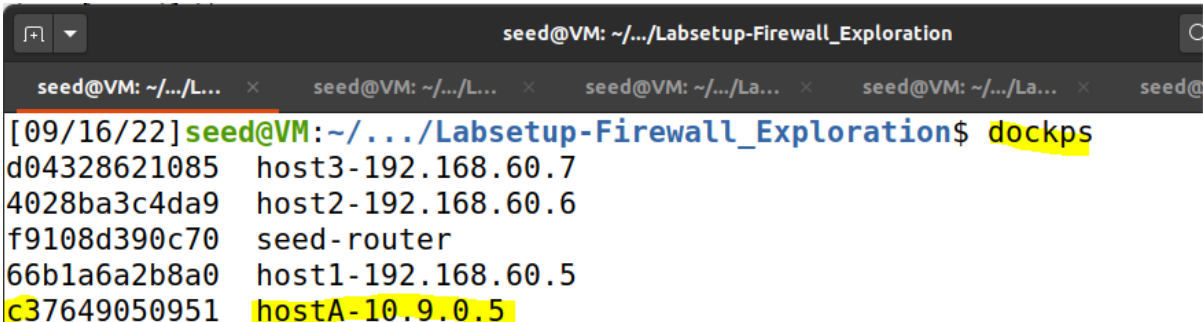
In this task, we will *set up rules to prevent outside machines from accessing the router machine, except ping*. Please execute the following `iptables` command on the router container, and then try to access it from 10.9.0.5. (1) Can you ping the router? (2) Can you telnet into the router (a telnet server is running on all the containers; an account called `seed` was created on them with a password `dees`). Please report your observation and explain the purpose for each rule.

```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -P OUTPUT DROP      # ← Set default rule for OUTPUT
iptables -P INPUT DROP       # ← Set default rule for INPUT
```

For this sub-task, I will implement hands-on step by step as follows:

#### First, we access all containers:

After enabling all containers from previous section, open the new terminal (ctrl+alt+T), using `dockps` command, we can see the list of containers.



```
seed@VM: ~/.../Labsetup-Firewall_Exploration
seed@VM: ~/.../L... x seed@VM: ~/.../L... x seed@VM: ~/.../La... x seed@VM: ~/.../La... x seed@
[09/16/22] seed@VM: ~/.../Labsetup-Firewall_Exploration$ dockps
d04328621085 host3-192.168.60.7
4028ba3c4da9 host2-192.168.60.6
f9108d390c70 seed-router
66b1a6a2b8a0 host1-192.168.60.5
c37649050951 hostA-10.9.0.5
```



## Firewall Exploration

Before accessing each container, we need to take notes the network, container name, container ID, and IP as shown in Figure 1. We will use the information later.

Network	Container name	Container ID	IP
<b>External</b>	hostA-10.9.0.5	c37649050951	10.9.0.5
<b>Internal - External</b>	seed-router	f9108d390c70	192.168.60.11, 10.9.0.11
<b>Internal</b>	host1-192.168.60.5	66b1a6a2b8a0	192.168.60.5
	host2-192.168.60.6	4028ba3c4da9	192.168.60.6
	host3-192.168.60.7	d04328621085	192.168.60.7

**TIPS:** Change title for each tab

Before you access to all containers, the problem is that we find hard to recognize which container is. You can see all are similar like this.

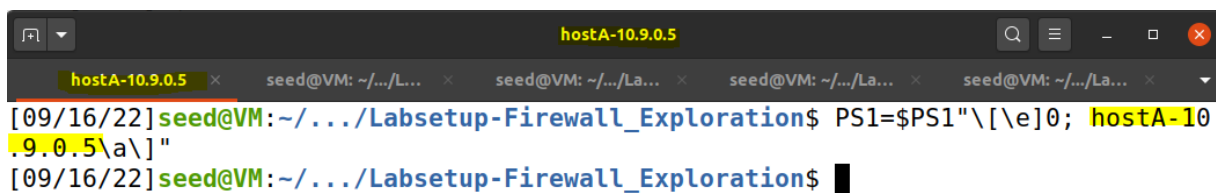


They have the same title. To make it easy to recognize, we will change the title for each container by assigning environment variable PS1 as the below syntax:

```
PS1=$PS1"\ [\e] 0;Title_Name\a\]"
```

For example, for hostA-10.9.0.5, we will execute the command as below

```
PS1=$PS1"\ [\e] 0; hostA-10.9.0.5\a\]"
```



Now you can change with other change the title for remaining terminal tabs (4 more) as below.

## Firewall Exploration

```

[09/16/22] seed@VM:~/.../Labsetup-Firewall_Exploration$

```

Now we can access (*docksh*) to each container that matches the title of each tab-terminal. The below is for hostA-10.9.0.5, you can do for other.

```

[09/16/22] seed@VM:~/.../Labsetup-Firewall_Exploration$ dockps
d04328621085  host3-192.168.60.7
4028ba3c4da9  host2-192.168.60.6
f9108d390c70  seed-router
66b1a6a2b8a0  host1-192.168.60.5
c37649050951  hostA-10.9.0.5
[09/16/22] seed@VM:~/.../Labsetup-Firewall_Exploration$ docksh c3
root@c37649050951:/#

```

### Second, implement the sub-task (Protecting the Router)

Back to the purpose of this sub-task is to *set up rules to prevent outside machines from accessing the router machine, except ping*.

From the router-11, we use `iptables -L` to list all the current rule

```

root@f9108d390c70:/# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

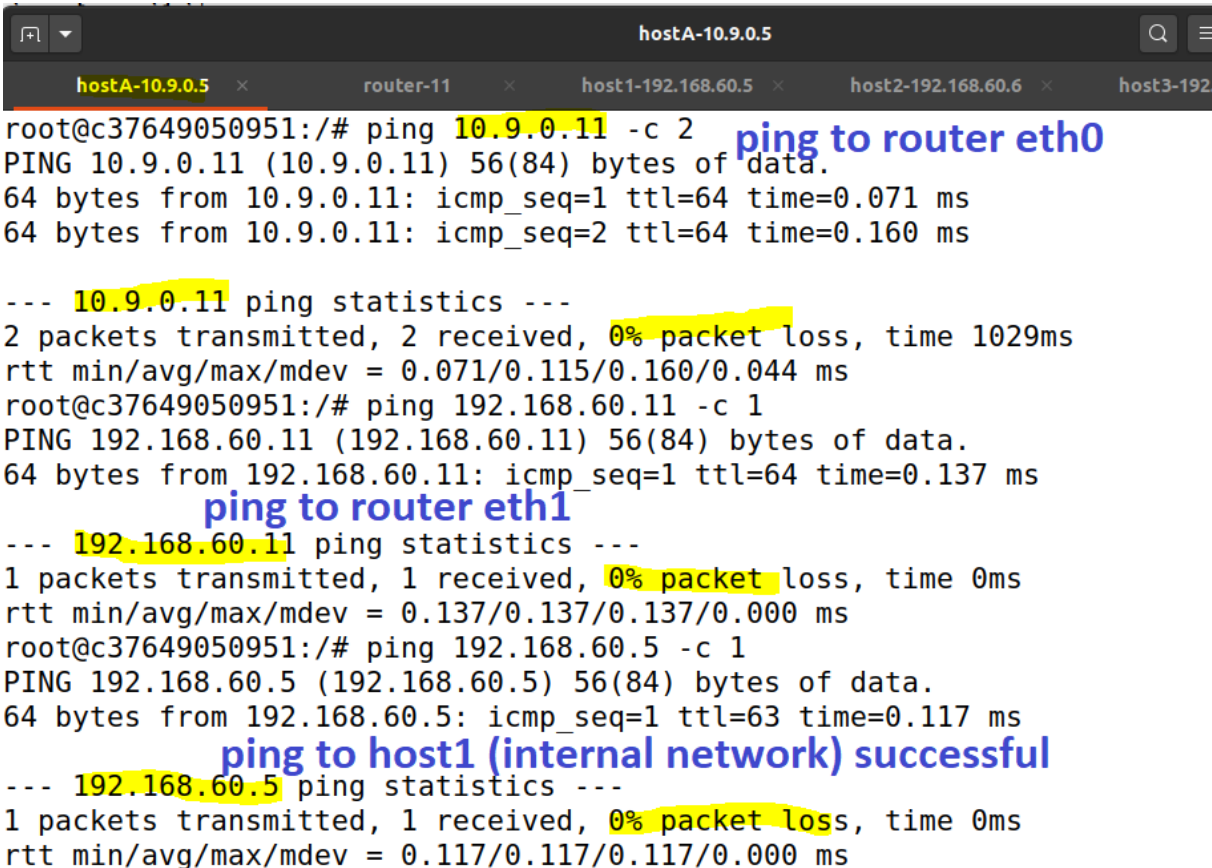
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
root@f9108d390c70:/#

```

## Firewall Exploration

You can see that we do not apply any rules for the network. So, we can test *ping* and *telnet* from host A (external network) to router or internal network (host 1, 2, 3) before applying the rules. It should work.

**Step 1:** ping from host A to router (both network interfaces) and internal networks (host1) successfully



```
root@c37649050951:/# ping 10.9.0.11 -c 2
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.071 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.160 ms

--- 10.9.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1029ms
rtt min/avg/max/mdev = 0.071/0.115/0.160/0.044 ms
root@c37649050951:/# ping 192.168.60.11 -c 1
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.137 ms

--- 192.168.60.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.137/0.137/0.137/0.000 ms
root@c37649050951:/# ping 192.168.60.5 -c 1
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.117 ms

--- 192.168.60.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.117/0.117/0.117/0.000 ms
```

ping to router eth0

ping to router eth1

ping to host1 (internal network) successful

**Step 2:** telnet from host A to router and host 1

## Firewall Exploration

```
hostA-10.9.0.5
hostA-10.9.0.5 x router-11 x host1-192.168.60.5 x host2-192.168.60.6 x host3-192.168.60.7 x
root@c37649050951:/# telnet 10.9.0.11
Trying 10.9.0.11...
Connected to 10.9.0.11.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f9108d390c70 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;
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.

telnet done
seed@f9108d390c70:~$
```

The above is shown that we telnet to the router (10.9.0.11) successfully by log with account seed and password dees. You can do similarly to telnet to host 1 (192.168.60.5). After testing telnet, use *exit* command to logout the session.

```
seed@f9108d390c70:~$ exit
logout
Connection closed by foreign host.
root@c37649050951:/#
```

**Step 3:** set up rules to prevent outside machines from accessing the router machine, except ping.

From router container, run these commands:

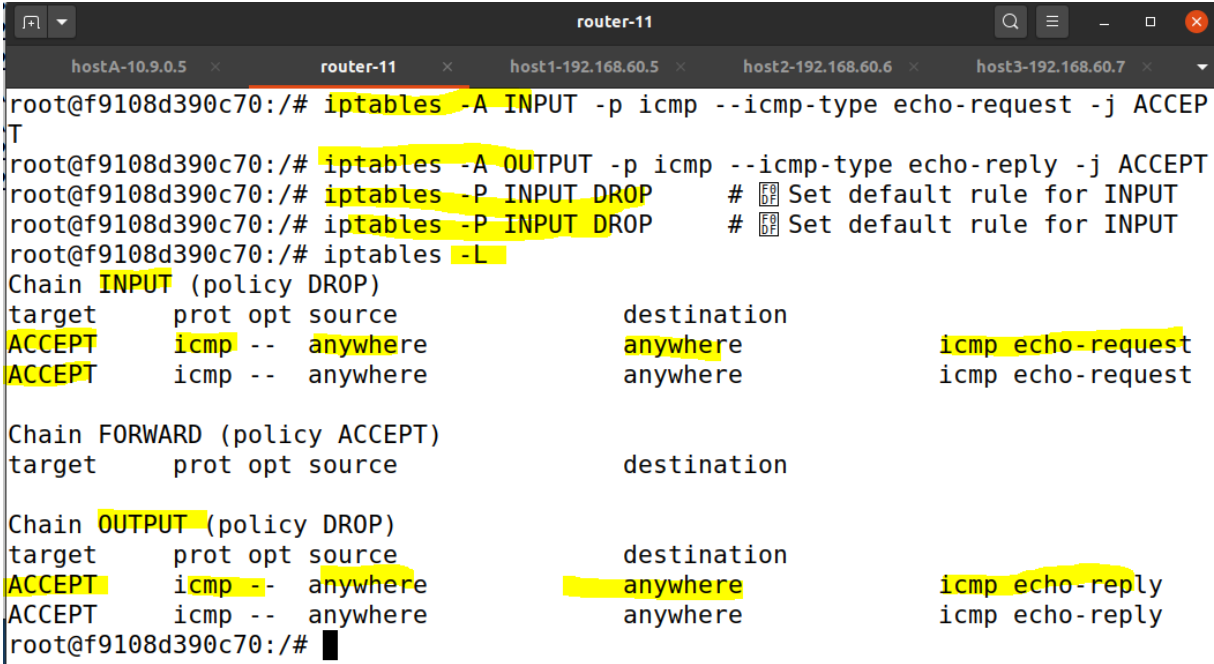
```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

## Firewall Exploration

```
iptables -P OUTPUT DROP    # ← Set default rule for OUTPUT
iptables -P INPUT DROP     # ← Set default rule for INPUT
```

Explain the above commands: (1) allows ping (request icmp packet) from external to internal network; (2) allows ping (reply icmp packet) from internal to external network; (3) blocks (drop) any packets from internal to external networks; (4) blocks (drop) any packets from external to internal networks;

In summary, 4 above commands will allow ping but blocks everything else.



```
root@f9108d390c70:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@f9108d390c70:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@f9108d390c70:/# iptables -P INPUT DROP    # Set default rule for INPUT
root@f9108d390c70:/# iptables -P INPUT DROP    # Set default rule for INPUT
root@f9108d390c70:/# iptables -L

Chain INPUT (policy DROP)
target    prot opt source                destination            icmp echo-request
ACCEPT    icmp -- anywhere             anywhere               icmp echo-request
ACCEPT    icmp -- anywhere             anywhere               icmp echo-request

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy DROP)
target    prot opt source                destination            icmp echo-reply
ACCEPT    icmp -- anywhere             anywhere               icmp echo-reply
ACCEPT    icmp -- anywhere             anywhere               icmp echo-reply
root@f9108d390c70:/#
```

Then using *iptables -L* command to list all the rules. You can see firewall accept all request and reply echo *icmp* packets from internal and external network and reversly.

**Step 4:** ping and telnet again (do Step 1 and Step 2 for testing)

From host A, we ping to host 1. It is working

However, when we telnet to router. It is not allowed. So, this means that we have completed this sub-task successfully.

## Firewall Exploration

```

hostA-10.9.0.5  ×  router-11  ×  host1-192.168.60.5  ×  host2-192.168.60.6  ×
root@c37649050951:/# ping 192.168.60.5 -c 1
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.083 ms

--- 192.168.60.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.083/0.083/0.083/0.000 ms
root@c37649050951:/# ping 192.168.60.5 -c 1
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.105 ms

--- 192.168.60.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.105/0.105/0.105/0.000 ms
root@c37649050951:/# telnet 10.9.0.11
Trying 10.9.0.11...

```

**We can ping**

**We cannot telnet**

```

root@c37649050951:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out

```

### Step 5: **IMPORTANT!!!** Cleanup

To do the next task, we need to clean up all the rules from the router, there are two ways.

1<sup>st</sup>, you can run these commands

```

iptables -F
iptables -P OUTPUT ACCEPT
iptables -P INPUT ACCEPT

```

2<sup>nd</sup>, restart the docker

```

docker restart <container id>

```



## Firewall Exploration

---

To do this, from the router (10.9.0.11) do the above commands, then check iptables list by using command *iptables -L*

```
root@f9108d390c70:/# iptables -F
root@f9108d390c70:/# iptables -P OUTPUT ACCEPT
root@f9108d390c70:/# iptables -P INPUT ACCEPT
root@f9108d390c70:/# iptables -L
Chain INPUT (policy ACCEPT)
target          prot opt source                               destination

Chain FORWARD (policy ACCEPT)
target          prot opt source                               destination

Chain OUTPUT (policy ACCEPT)
target          prot opt source                               destination
root@f9108d390c70:/#
```

---

You will do this step regularly whenever you start a new task.

Now, you can do by yourself the remaining tasks.

### 1.4. Task 1.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.

The directions of packets in the INPUT and OUTPUT chains are clear: packets are either coming into (for INPUT) or going out (for OUTPUT). This is not true for the FORWARD chain, because it is bi-directional: packets going into the internal network or going out to the external network all go through this chain.

To specify the direction, we can add the interface options using "-i xyz" (coming in from the xyz interface) and/or "-o xyz" (going out from the xyz interface).

The interfaces for the internal and external networks are different. You can find out the interface names via the "ip addr" command.

1. Outside hosts cannot ping internal hosts.

## Firewall Exploration

---

2. Outside hosts can ping the router.
3. Internal hosts can ping outside hosts.
4. All other packets between the internal and external networks should be blocked

You will need to use the "-p icmp" options to specify the match options related to the ICMP protocol. You can run "iptables -p icmp -h" to find out all the ICMP match options. The following example drops the ICMP echo request

```
iptables -A FORWARD -p icmp --icmp-type echo-request -j DROP
```

In your lab report, please include your rules and screenshots to demonstrate that your firewall works as expected. When you are done with this task, please remember to clean the table or restart the container before moving on to the next task.

### 1.5. Task 1.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.

1. 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.
2. Outside hosts cannot access other internal servers.
3. Internal hosts can access all the internal servers.
4. Internal hosts cannot access external servers.
5. In this task, the connection tracking mechanism is not allowed. It will be used in a later task

You will need to use the "-p tcp" options to specify the match options related to the TCP protocol. You can run "iptables -p tcp -h" to find out all the TCP match options. The following example allows the TCP packets coming from the interface *eth0* if their source port is 5000.



## Firewall Exploration

---

```
iptables -A FORWARD -i eth0 -p tcp --sport 5000 -j ACCEPT
```

When you are done with this task, please remember to clean the table or restart the container before moving on to the next task.

## 2. TASK 2: 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.

To sum-up, the goal of this task is to understand: (1) what the difference is between stateless and *stateful* firewalls; (2) what limitation stateless firewalls have; (3) how stateless deals with TCP, UDP and ICMP connections; (4) *iptables* and the *conntrack* module.

### 2.1. Task 2.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.

The goal is to understand *iptables* and the *conntrack* module; and viewing *conntrack* timeout configurations. In our experiment, we will check the connection tracking information on the router container. This can be done using the following command:

```
# conntrack -L
```

## Firewall Exploration

---

The goal of the task is to use a series of experiments to help you understand the connection concept in this tracking mechanism, especially for the ICMP and UDP protocols, because unlike TCP, they do not have connections. Please conduct the following experiments. For each experiment, please describe your observation, along with your explanation.

- ICMP experiment: Run the following command and check the connection tracking information on the router. Describe your observation. How long is the ICMP connection state be kept?

```
// On 10.9.0.5, send out ICMP packets  
# ping 192.168.60.5
```

- UDP experiment: Run the following command and check the connection tracking information on the router. Describe your observation. How long is the UDP connection state be kept?

```
// On 192.168.60.5, start a netcat UDP server  
# nc -lu 9090  
// On 10.9.0.5, send out UDP packets  
# nc -u 192.168.60.5 9090  
<type something, then hit return>
```

- TCP experiment: Run the following command and check the connection tracking information on the router. Describe your observation. How long is the TCP connection state be kept?

```
// On 192.168.60.5, start a netcat TCP server  
# nc -l 9090  
// On 10.9.0.5, send out TCP packets  
# nc 192.168.60.5 9090  
<type something, then hit return>
```

## 2.2. Task 2.B: Setting Up a Stateful Firewall

---

## Firewall Exploration

---

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* relies on the tracking information to build *stateful* firewalls. The `--ctstate 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.

```
iptables -A FORWARD -p tcp -m conntrack \  
--ctstate ESTABLISHED,RELATED -j ACCEPT
```

The rule above does not cover the SYN packets, which do not belong to any established connection. Without it, we will not be able to create a connection in the first place. Therefore, we need to add a rule to accept incoming SYN packet:

```
iptables -A FORWARD -p tcp -i eth0 --dport 8080 --syn \  
-m conntrack --ctstate NEW -j ACCEPT
```

Finally, we will set the default policy on FORWARD to drop everything. This way, if a packet is not accepted by the two rules above, they will be dropped.

```
iptables -P FORWARD DROP
```

Please rewrite the firewall rules in Task 1.C, but this time, we will add a rule allowing internal hosts to visit any external server (this was not allowed in Task 1.C). After you write the rules using the connection tracking mechanism, think about how to do it without using the connection tracking mechanism (you do not need to actually implement them). Based on these two sets of rules, compare these two different approaches, and explain the advantage and disadvantage of each approach. When you are done with this task, remember to clear all the rules

---

**BONUS tasks:** Limiting Network Traffic and Load Balancing. Please refer to the link to the SEED project from reference (end of this document) to have some suggestions.

Enjoy your happy lab!

**Reference:**

1. [https://seedsecuritylabs.org/Labs\\_20.04/Files/Firewall/Firewall.pdf](https://seedsecuritylabs.org/Labs_20.04/Files/Firewall/Firewall.pdf)
2. <https://linuxhint.com/control-network-traffic-using-iptables/>