Q1 Introduction to iptables
0 Points

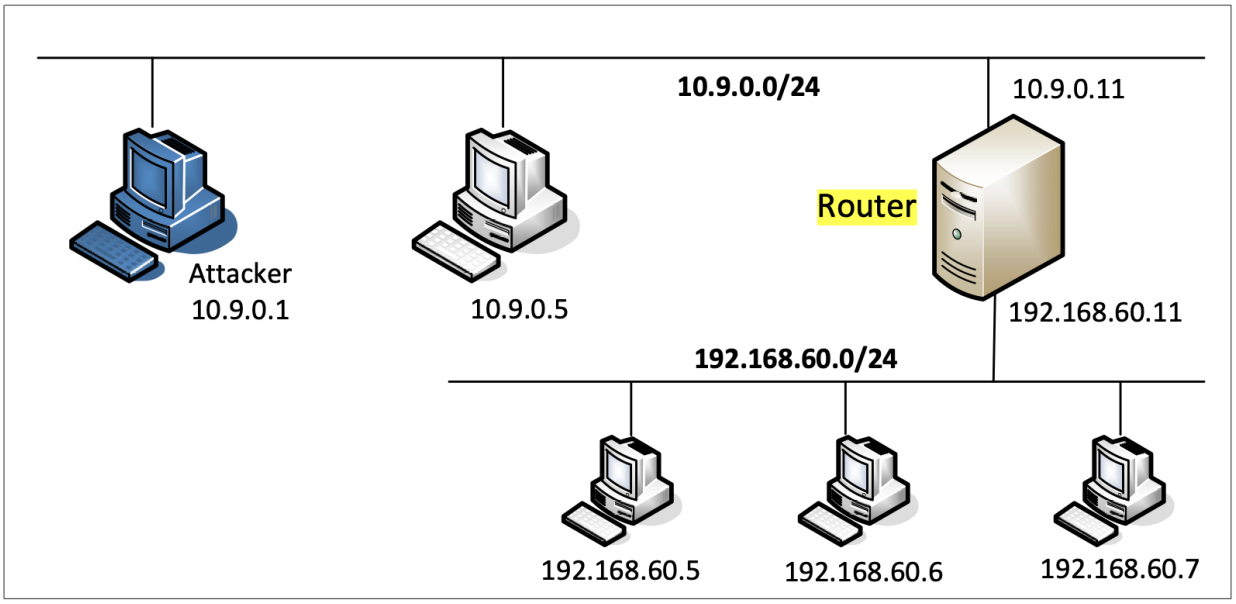
Introduction

We will only be using **tasks 2-4** from the SEED lab, with additional **Task A, B, and C** not in the PDF file. The full full lab detail can be found here: Firewall Exploration Lab

Earliest acceptance date: 30 April (+5% bonus)
Normal due date: 5 May
Latest acceptance date: 9 May (-4% points)

Note: Ensure all screenshots contain the full VM with the date and time.

Please follow the instructions in the Firewall Explorations Lab to setup the lab environment. Specifically, you would need to download the Labsetup.zip or Labsetup-arm.zip (for Apple Silicon) file and import it into the VM to set up the docker containers.



Note: Please skip Task 1 and move to Task 2 (page 8)

Q2 Task 2: Experimenting with Stateless Firewall Rules
35 Points

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.

Using IPtables

To add rules to the chains in each table, we use the iptables command, which is a quite powerful command. Students 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 we 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:

Table 1: `iptables` Tables and Chains

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

```
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 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.

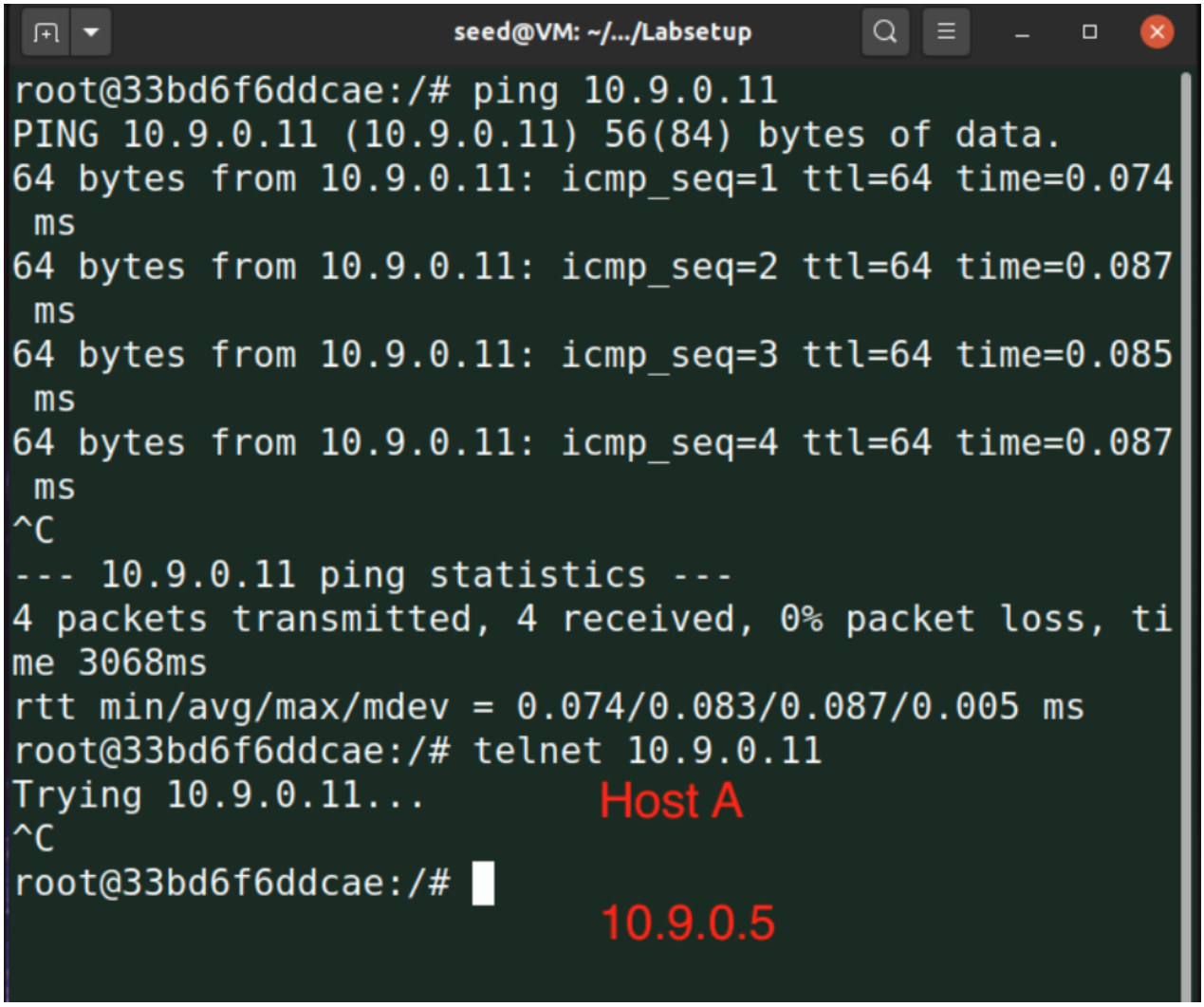
Q2.1 Task 2.A: Protecting the Router
5 Points

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.

```
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
```

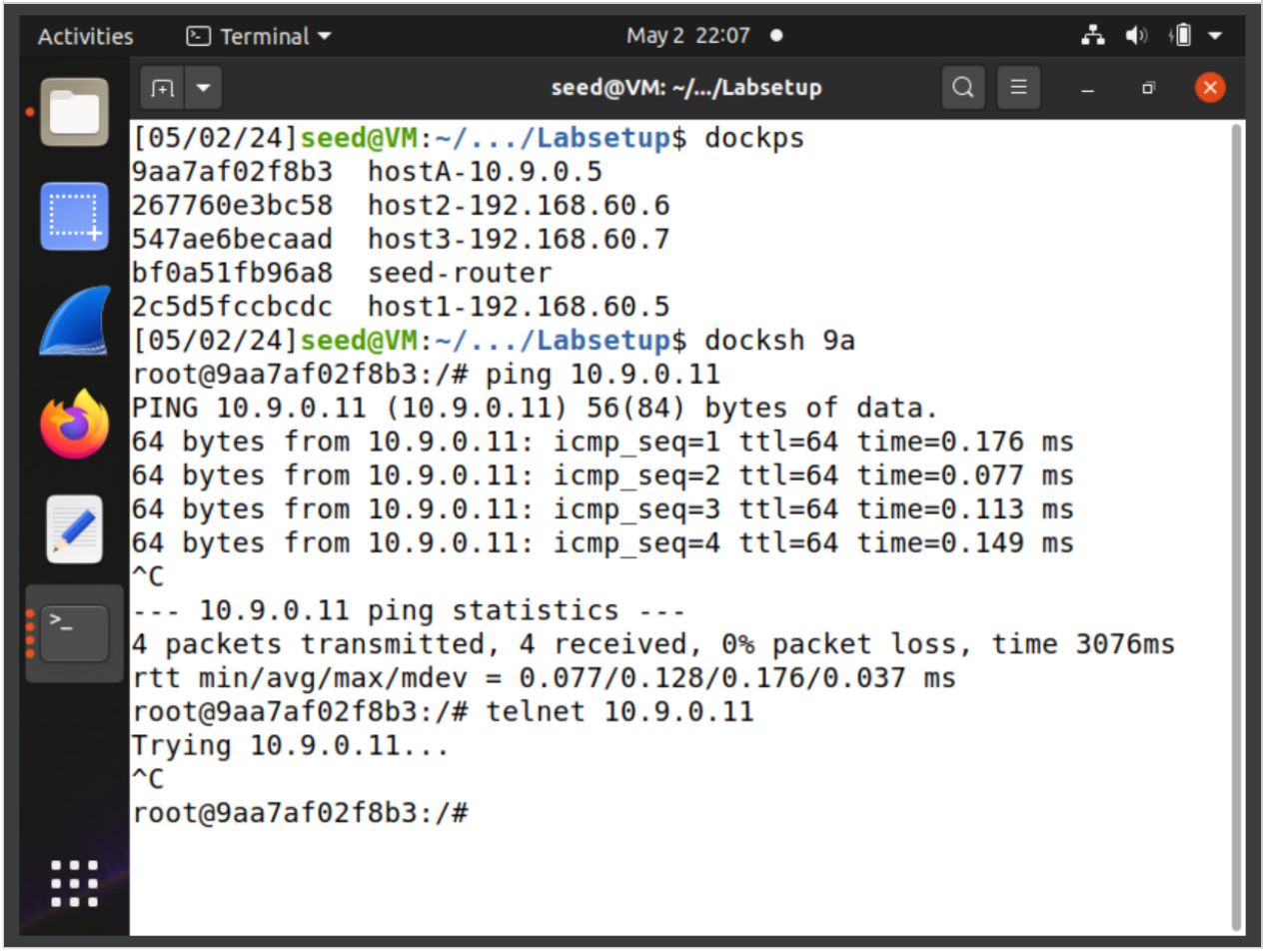
Expected Output

Note: Only ping packets are allowed, not telnet



Upload screenshots below (must show ping and telnet)

2A_ping_only.png Download



What is the purpose of each of the four rules?

1. If an incoming packet is an ICMP echo-request, let it pass through the firewall.
2. If an outgoing packet is an ICMP echo-reply, let it pass through the firewall.
3. If an outgoing packet does not match one of the rules, drop it.
4. If an incoming packet does not match one of the rules, drop it.

Answer the following questions

(1) Can you ping the router?

(2) Can you telnet into the router?

1. Yes
2. No

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

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

Another way to restore the states of all the tables is to restart the container. You can do it using the following command (you need to find the container's ID first):

```
$ docker restart <Container ID>
```

Q2.2 Task 2.B: Protecting the Internal Network
15 Points

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.

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

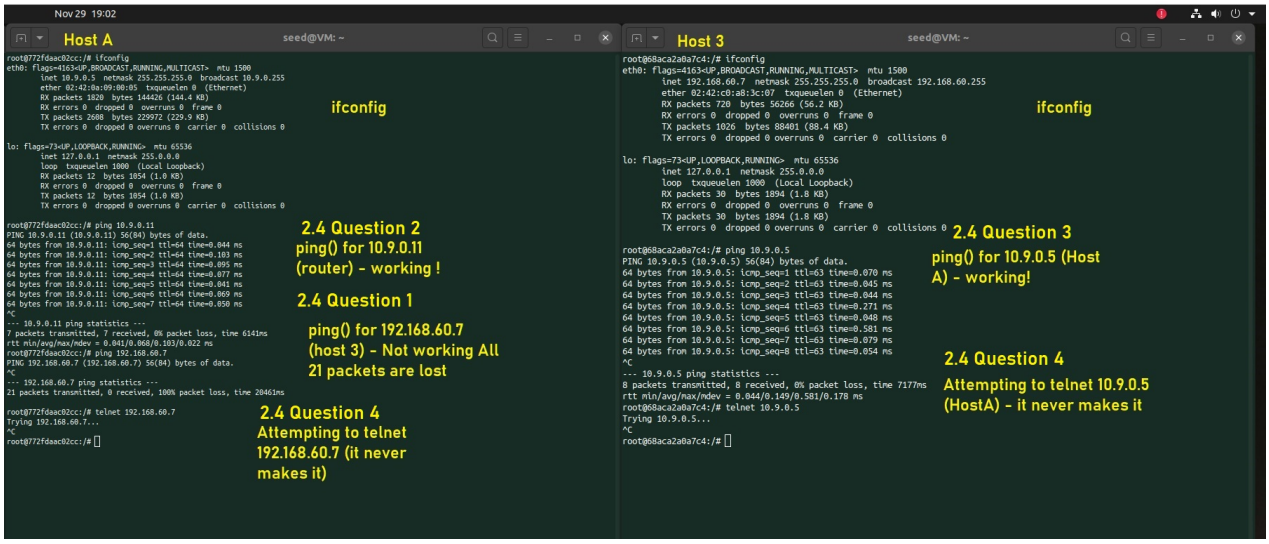
1. Outside hosts cannot ping internal hosts.
2. Outside hosts can ping the router.
3. Internal hosts can ping outside hosts.
4. All other packets between the internal and external networks should be blocked

Note: "Outside" refers to the internet, not just the 10.9.0.11/24 network.

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
```

Expected Output (Note: Ignore the 2.4 in the text)



Upload a screenshot of your iptable rules from the router

Run the `iptables -L -v` command to view iptable rules

2B_rules.png Download

```
root@bf0a51fb96a8:/# iptables -L -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
          icmp echo-request

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DROP       icmp --  eth0   eth1    anywhere             anywhere
          icmp echo-request
    0     0 DROP       icmp --  eth1   eth0    anywhere             anywhere
          icmp echo-reply
    0     0 ACCEPT     icmp --  eth1   eth0    anywhere             anywhere
          icmp echo-request
    0     0 ACCEPT     icmp --  eth0   eth1    anywhere             anywhere
          icmp echo-reply

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
          icmp echo-reply
```

Upload screenshots showing proof of checking rules 1-4 above (Please label each items)

internal_ping_outside.png Download

```
[05/02/24]seed@VM:~/.../Labsetup$ dockps
9aa7af02f8b3  hostA-10.9.0.5
267760e3bc58  host2-192.168.60.6
547ae6becaad  host3-192.168.60.7
bf0a51fb96a8  seed-router
2c5d5fccbcdc  host1-192.168.60.5
[05/02/24]seed@VM:~/.../Labsetup$ docksh 26
root@267760e3bc58:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.138 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.184 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.219 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.071 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.071/0.153/0.219/0.055 ms
root@267760e3bc58:/#
```

internal_telnet_outside.png Download

```
[05/02/24]seed@VM:~/.../Labsetup$ dockps
9aa7af02f8b3  hostA-10.9.0.5
267760e3bc58  host2-192.168.60.6
547ae6becaad  host3-192.168.60.7
bf0a51fb96a8  seed-router
2c5d5fccbcdc  host1-192.168.60.5
[05/02/24]seed@VM:~/.../Labsetup$ docksh 26
root@267760e3bc58:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

outside_ping_internal.png Download

```
[05/02/24]seed@VM:~/.../Labsetup$ dockps
9aa7af02f8b3  hostA-10.9.0.5
267760e3bc58  host2-192.168.60.6
547ae6becaad  host3-192.168.60.7
bf0a51fb96a8  seed-router
2c5d5fccbcdc  host1-192.168.60.5
[05/02/24]seed@VM:~/.../Labsetup$ docksh 9a
root@9aa7af02f8b3:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
^C
--- 192.168.60.6 ping statistics ---
17 packets transmitted, 0 received, 100% packet loss, time 16393ms

root@9aa7af02f8b3:/#
```

outside_ping_router.png Download

```
[05/02/24]seed@VM:~/.../Labsetup$ dockps
9aa7af02f8b3  hostA-10.9.0.5
267760e3bc58  host2-192.168.60.6
547ae6becaad  host3-192.168.60.7
bf0a51fb96a8  seed-router
2c5d5fccbcdc  host1-192.168.60.5
[05/02/24]seed@VM:~/.../Labsetup$ docksh 9a
root@9aa7af02f8b3:/# ping 10.9.0.11
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=0.144 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.084 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.057 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.101 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.057/0.096/0.144/0.031 ms
root@9aa7af02f8b3:/#
```

outside_telnet_internal.png Download

```
[05/02/24]seed@VM:~/.../Labsetup$ dockps
9aa7af02f8b3  hostA-10.9.0.5
267760e3bc58  host2-192.168.60.6
547ae6becaad  host3-192.168.60.7
bf0a51fb96a8  seed-router
2c5d5fccbcdc  host1-192.168.60.5
[05/02/24]seed@VM:~/.../Labsetup$ docksh 9a
root@9aa7af02f8b3:/# telnet 192.168.60.6
Trying 192.168.60.6...
```

Q2.3 Task 2.C: Protecting Internal Servers
15 Points

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.

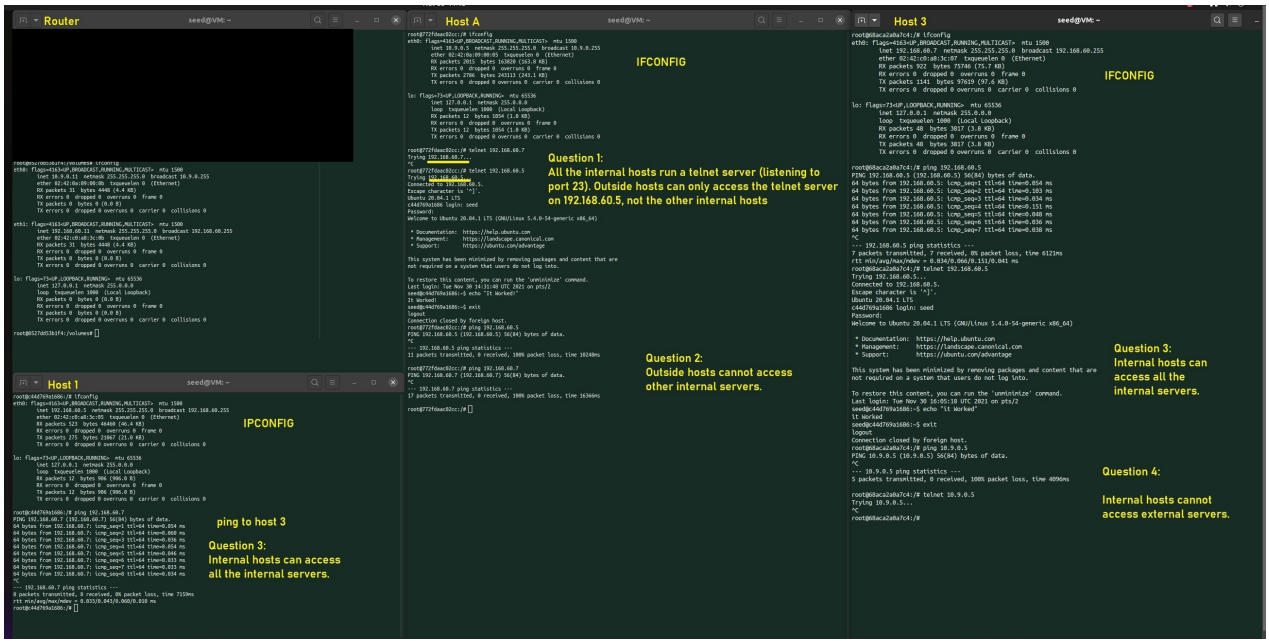
1. All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.
2. Outside hosts cannot access other internal servers.
3. Internal hosts can access all the internal servers.
4. Internal hosts cannot access external servers.
5. In this task, the connection tracking mechanism is not allowed. It will be used in a later task

Note: "Outside" refers to the internet, not just the 10.9.0.11/24 network.

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.

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

Expected Output



Upload a screenshot of your iptable rules from the router

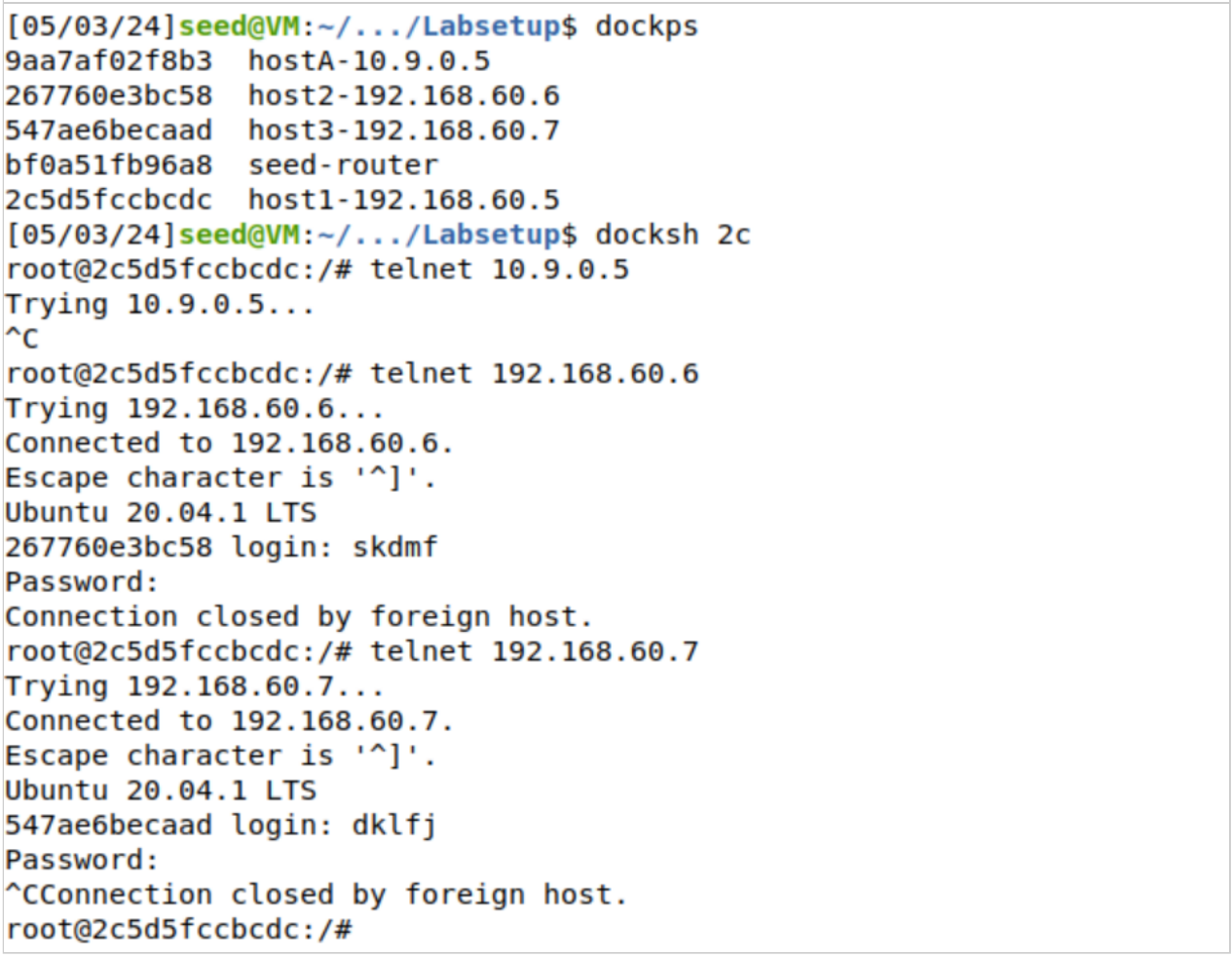
Run the `iptables -L -v` command to view iptable rules

▾ 2C_rules.png Download

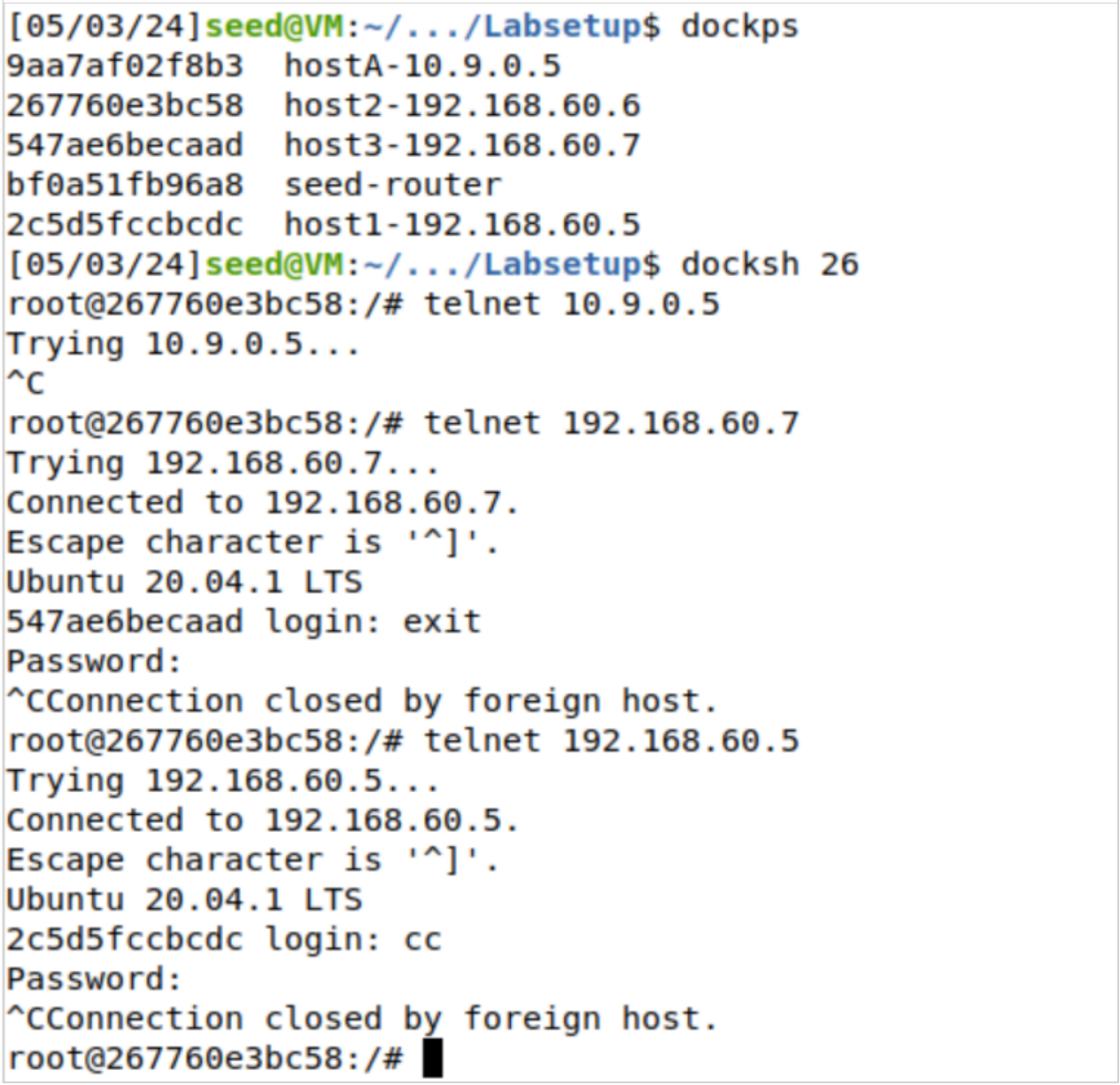
```
root@bf0a51fb96a8:/# iptables -L -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 ACCEPT     tcp  --  eth0   eth1    anywhere             192.168.60.
5           tcp dpt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth0    192.168.60.5         anywhere
           tcp spt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth1    192.168.60.5         anywhere
           tcp spt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth1    anywhere             192.168.60.
5           tcp dpt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth1    anywhere             192.168.60.
6           tcp dpt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth1    192.168.60.6         anywhere
           tcp spt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth1    192.168.60.7         anywhere
           tcp spt:telnet
    0     0 ACCEPT     tcp  --  eth1   eth1    anywhere             192.168.60.
7           tcp dpt:telnet

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

root@bf0a51fb96a8:/#
```

Upload screenshots showing proof of checking rules 1-5 above (Please label each items)

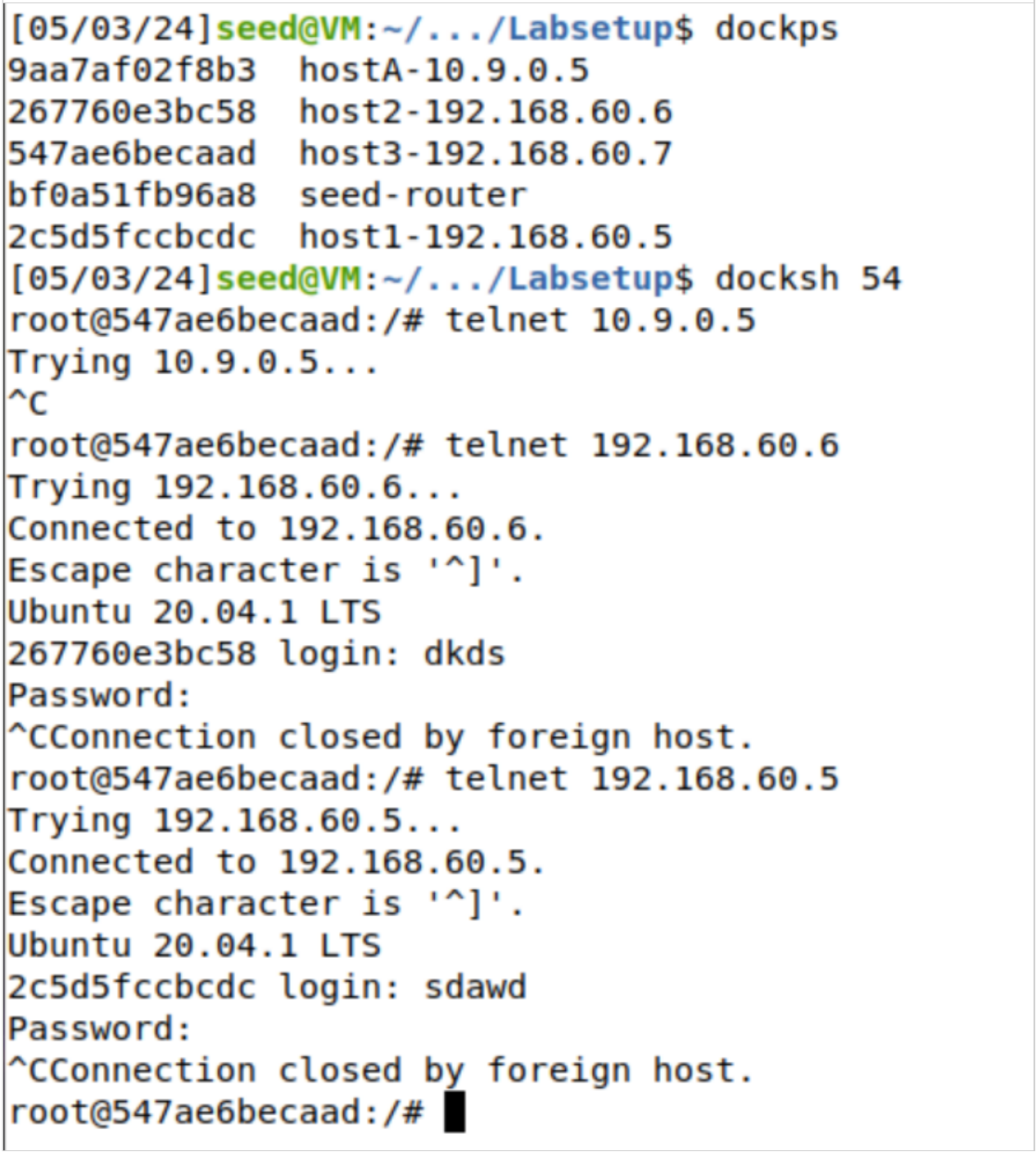
2C_192.168.60.5_telnet_outside_and_internal.png Download



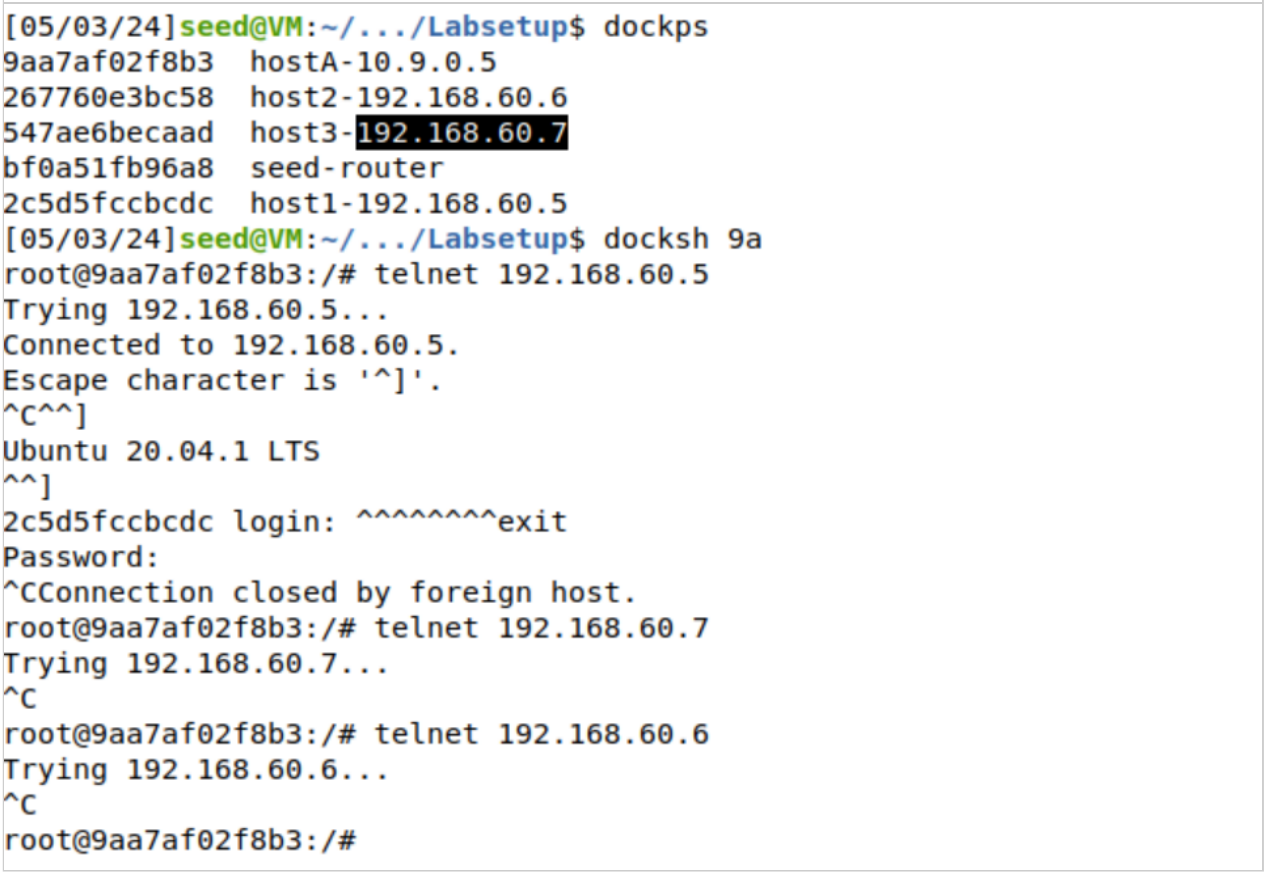
2C_192.168.60.6_telnet_outside_and_internal.png Download



2C_192.168.60.7_telnet_outside_and_internal.png Download



2C_outside_telnet_internal.png Download



When you are done with this task, please remember to clean the table (`e iptables -F`) or restart the container before moving on to the next task.

Q3 Task 3: Connection Tracking and Stateful Firewall

25 Points

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.

Q3.1 Task 3.A: Experiment with the Connection Tracking
15 Points

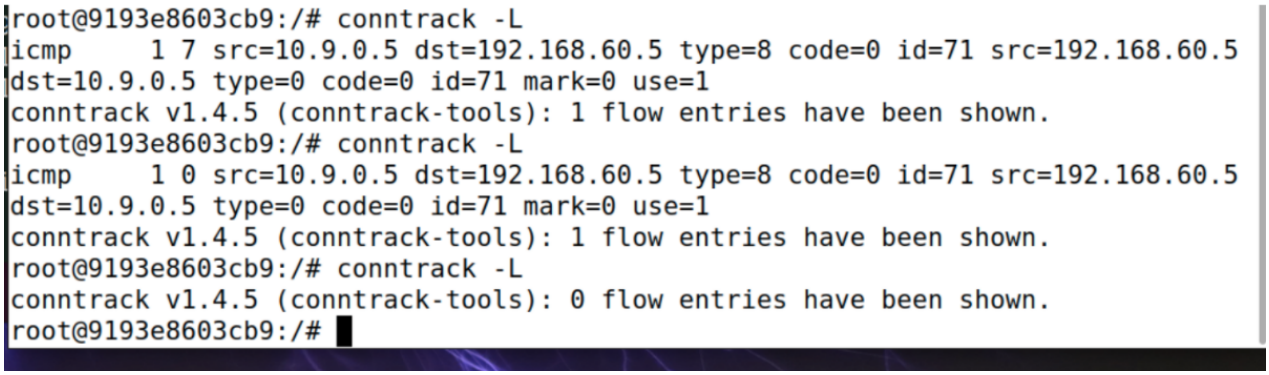
To support stateful firewalls, we need to be able to track connections. This is achieved by the conntrack mechanism inside the kernel. In this task, we will conduct experiments related to this module, and get familiar with the connection tracking mechanism. In our experiment, we will check the connection tracking information on the router container. This can be done using the following command:

```
# conntrack -L
```

The goal of the task is to use a series of experiments to help students 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.

Hint: Use `sysctl net.netfilter` to see the timeout settings, and verify.

Expected Output for ICMP only (similar output for UDP and TCP)



• 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
```

Upload your screenshots for ICMP connection tracking

3A_connection_tracking_icmp.png Download

```
root@bf0a51fb96a8:/# conntrack -L
icmp     1 27 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=119 src=192.168.60
.5 dst=10.9.0.5 type=0 code=0 id=119 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
icmp     1 26 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=119 src=192.168.60
.5 dst=10.9.0.5 type=0 code=0 id=119 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
icmp     1 25 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=119 src=192.168.60
.5 dst=10.9.0.5 type=0 code=0 id=119 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
icmp     1 24 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=119 src=192.168.60
.5 dst=10.9.0.5 type=0 code=0 id=119 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
icmp     1 23 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=119 src=192.168.60
.5 dst=10.9.0.5 type=0 code=0 id=119 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=119 src=192.168.60
.5 dst=10.9.0.5 type=0 code=0 id=119 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

Describe your observation for ICMP connection tracking. Be sure to specify how long the tracking information is kept once a connection has been established.

The ICMP connection tracking information is kept for 30 seconds

• 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>
```

Upload your screenshots for UDP connection tracking

3A_connection_tracking_udp.png Download

```
root@bf0a51fb96a8:/# conntrack -L
udp      17 25 src=10.9.0.5 dst=192.168.60.5 sport=43285 dport=9090 [UNREPLIED]
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=43285 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
udp      17 23 src=10.9.0.5 dst=192.168.60.5 sport=43285 dport=9090 [UNREPLIED]
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=43285 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
udp      17 22 src=10.9.0.5 dst=192.168.60.5 sport=43285 dport=9090 [UNREPLIED]
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=43285 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
udp      17 21 src=10.9.0.5 dst=192.168.60.5 sport=43285 dport=9090 [UNREPLIED]
 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=43285 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@bf0a51fb96a8:/#
```

Describe your observation for UDP connection tracking. Be sure to specify how long the tracking information is kept once a connection has been established.

The UDP connection tracking information is kept for 30 seconds

• TCP experiment: Run the following command and check the connection tracking information on the router. Describe your observation. Be sure to specify how long the tracking information is kept once a connection has been established.

```
// 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>
```

Upload your screenshots for TCP connection tracking

3A_connection_tracking_tcp.png Download

```
root@bf0a51fb96a8:/# conntrack -L
tcp      6 431985 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=48806 dport=9
090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=48806 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/# conntrack -L
tcp      6 431995 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=48806 dport=9
090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=48806 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@bf0a51fb96a8:/#
```

Describe your observation for TCP connection tracking. Be sure to specify how long the tracking information is kept.

The UDP connection tracking information is kept for 432000 seconds

Q3.2 Task 3.B: Setting Up a Stateful Firewall
10 Points

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.

```
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.

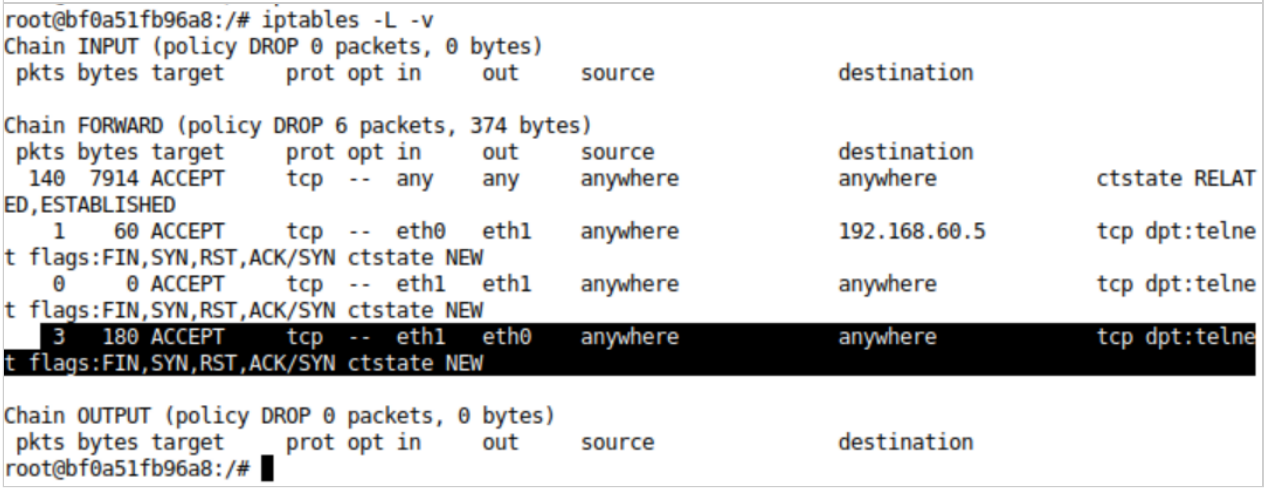
```
iptables -P FORWARD DROP
```

Please rewrite the firewall rules in Q2.3 Task 2.C, but this time, we will add a rule allowing internal hosts to visit any external server (this was not allowed in Task 2.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). When you are done with this task, remember to clear all the rules.

Upload your IPTables rules `iptables -L -v` and highlight the new rule to allow internal hosts to visit any external server

3B_rules.png

Download



Q4 Task 4: Limiting Network Traffic

10 Points

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.

```
$ iptables -m limit -h
limit match options:
--limit avg           max average match rate: default 3/hour
                      [Packets per second unless followed by
                      /sec /minute /hour /day postfixes]
--limit-burst number    number to match in a burst, default 5
```

Please run the following commands on router, and then ping 192.168.60.5 from 10.9.0.5. Describe your observation. Please conduct the experiment with and without the second rule, and then explain whether the second rule is needed or not, and why.

```
iptables -A FORWARD -s 10.9.0.5 -m limit \
        --limit 10/minute --limit-burst 5 -j ACCEPT
iptables -A FORWARD -s 10.9.0.5 -j DROP
```

Expected Output with second rule

```
root@33bd6f6ddcae:/# ping 192.168.60.5
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
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.135 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.161 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.111 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.113 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.105 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.111 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.116 ms
^C
--- 192.168.60.5 ping statistics ---
24 packets transmitted, 8 received, 66.6667% packet loss, time 2353
1ms
rtt min/avg/max/mdev = 0.083/0.116/0.161/0.021 ms
```

Upload your screenshots below

4.png Download

```
root@9aa7af02f8b3:/# ping 192.168.60.5
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=0.1
16 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.0
55 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.1
03 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.2
56 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.1
03 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.1
50 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.
155 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.
056 ms
```

Explain your observation on whether the second rule is needed or not, and why

After the initial burst of 5 packets, the firewall starts dropping packets so that the limit of 10 packets per minute is upheld. The second rule is needed because otherwise the firewall does not drop packets so that the limit is upheld (since the default rule on FORWARD is to accept everything).

Q5 Task A - Multiple Choice

5 Points

Q5.1
1 Point

This Chain applies for all packets that are addressed to the firewall

- (●) INPUT
- ( ) OUTPUT
- ( ) FORWARD

Q5.2
1 Point

This chain applies for all packets originating from firewall and going out of the server

- ( ) INPUT
- (●) OUTPUT
- ( ) FORWARD

Q5.3
1 Point

This chain applies for all packets passing through the firewall from other hosts on the network. The host with iptables is neither the source nor destination of the packet; mainly used to route packets through the machines on the network.

- ( ) INPUT
- ( ) OUTPUT
- (●) FORWARD

Q5.4
1 Point

This jump target does not respond to a packet at all and does nothing with the packet. If an attack sends a packet, they would not get any response.

- (●) DROP
- ( ) REJECT

Q5.5
1 Point

This jump target responds with an ICMP Destination Unreachable back to the source. This indicates that a server exists, which is beneficial for troubleshooting and for attackers.

- ( ) DROP
- (●) REJECT

Q6 Task B (not in SEED Lab PDF)
5 Points

iptables Logging

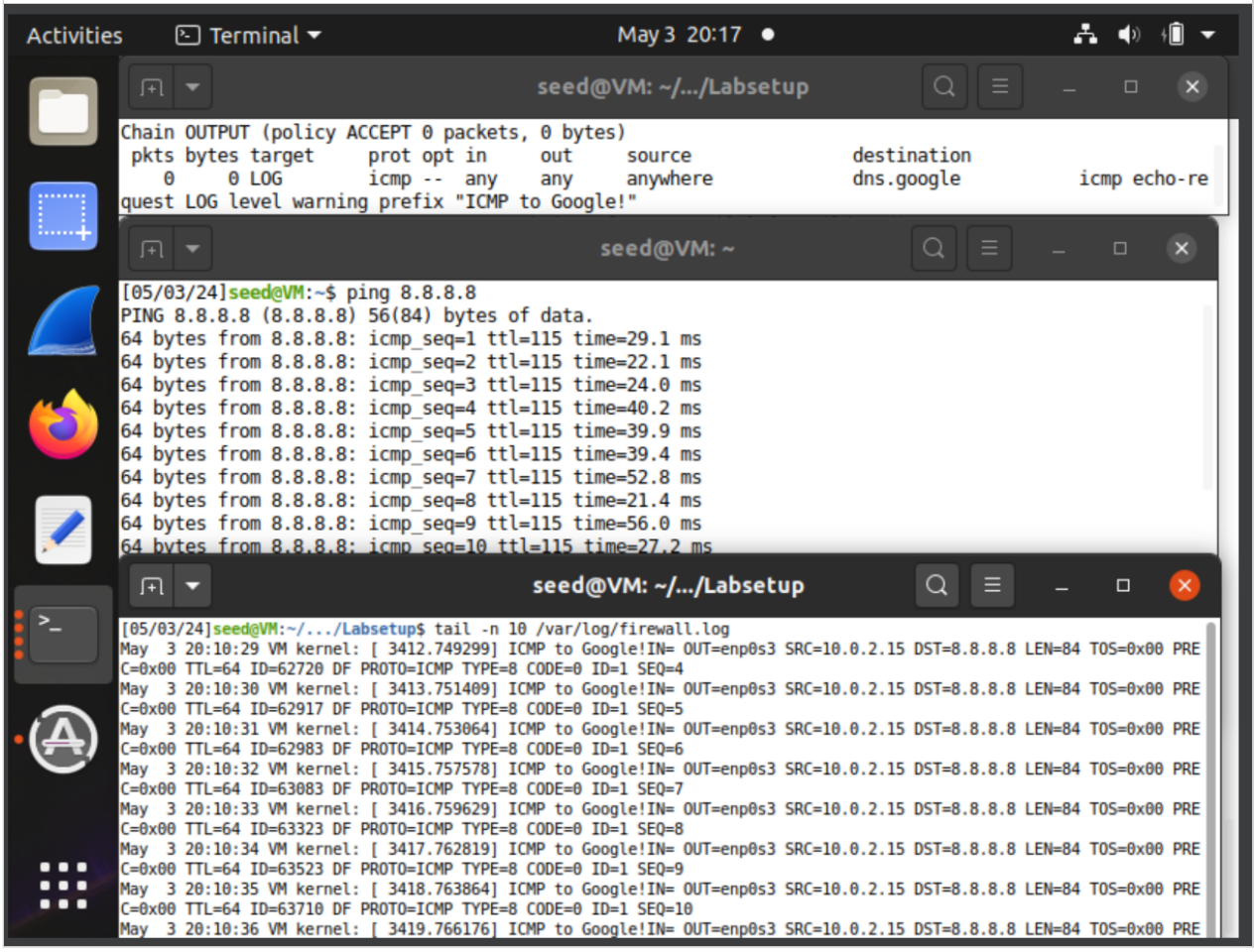
Note: this exercise is performed on the Guest VM (the first VM that you log into, not the docker containers. Your prompt should say `seed@[hostname]`. Also, there are other rules here, ignore them, or if they get messed up, restart the VM to restore the rules.

Sometimes you need to log certain packets. Using the jump target LOG (`-j LOG`). Write a rule to log all outgoing icmp packets sent to 8.8.8.8 with the message "ICMP to Google!" (using `--log-prefix`)

1. Enable logging by adding the line `kern.warn /var/log/firewall.log` to the file `/etc/rsyslog.conf` (be sure to use sudo)
2. Restart rsyslog: `sudo service rsyslog restart`
3. Create traffic to be logged (using an iptables rule with `-j LOG`)
4. The log is saved to `/var/log/firewall.log`

Submit a screenshot with the following (full VM with date and time):
1. Your `iptables -L -v` with the LOG rule
2. Your ping from the terminal
3. Log file showing the pings (you can use something such as `tail -n 10 /var/log/firewall.log`)

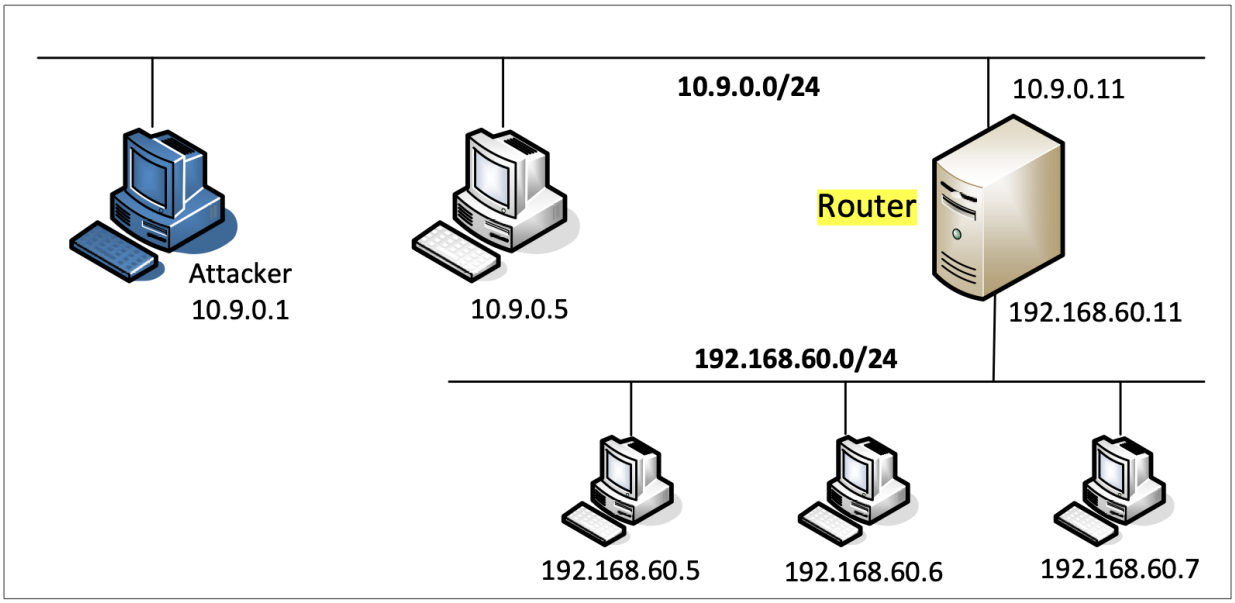
▾ Task B.png Download



When done, remove the rule (`-D`), or just restart the entire VM.

Q7 Task C (not in the SEED Lab PDF)
20 Points

Flush all your rules (`iptables -F`), and write the following rules **only** on the Router.



All traffic should be DROP'ed except for the following rules:
1. Any outgoing traffic from the 192.168.60.0/24 network to the 10.9.0.0/24 network is allowed. That is, the connection must be initiated from the 192.168.60.0/24 network (and not the other way around).
2. Any host can ping the router (10.9.0.11 or 192.168.60.11)
3. Host 10.9.0.5 is allowed to telnet to 192.168.60.5

All rules must be stateful

Your complete iptable rules. **Please write the rules in the same order given above (1,2, then 3).** Hint: do not forget the default policy.

iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

1. iptables -A FORWARD -i eth1 -o eth0 -d 10.9.0.0/24 -s 192.168.60.0/24 -m conntrack --ctstate NEW -j ACCEPT

2. iptables -A INPUT -p icmp --icmp-type echo-request -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

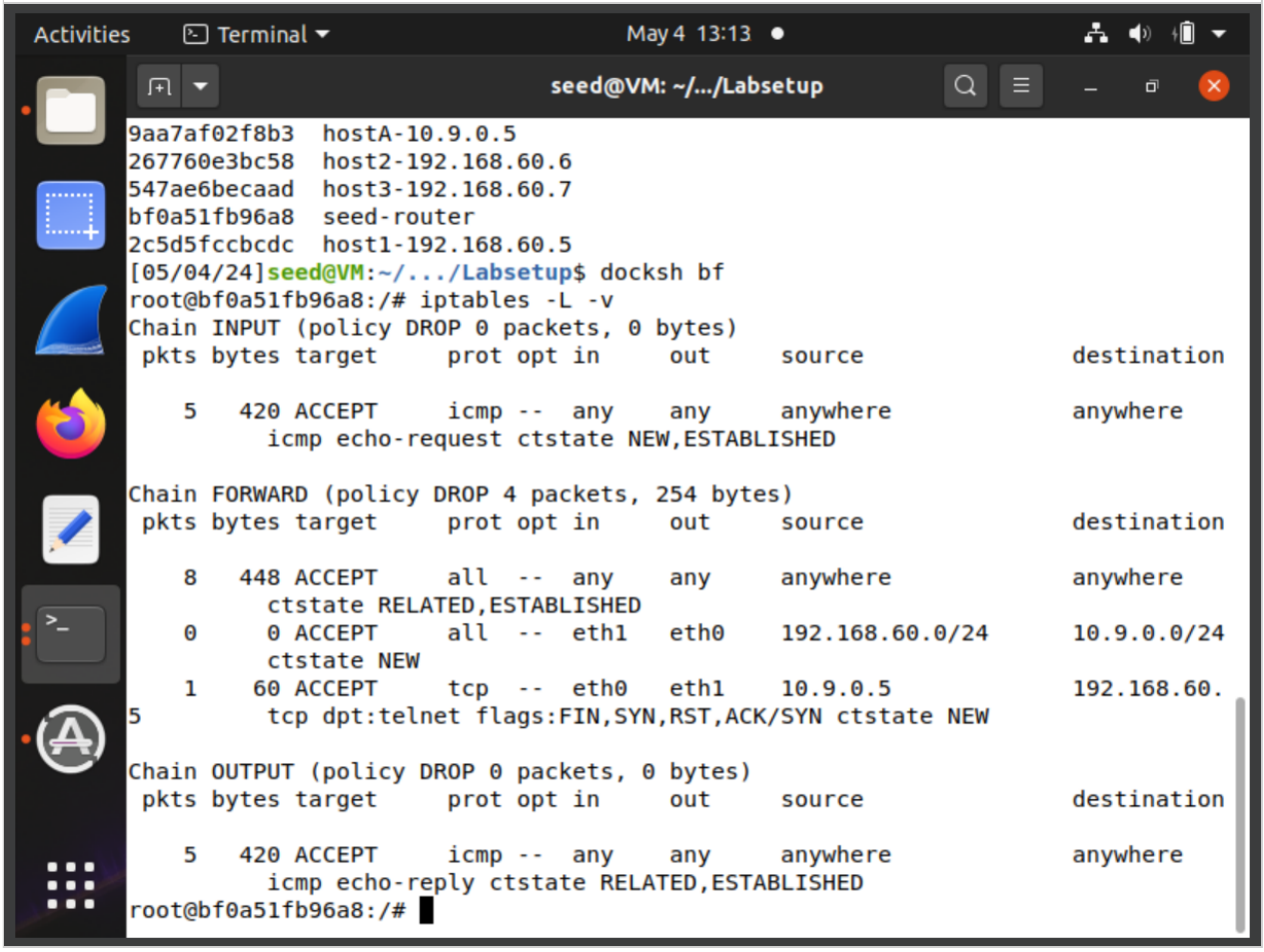
iptables -A OUTPUT -p icmp --icmp-type echo-reply -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

3. iptables -A FORWARD -p tcp -i eth0 -o eth1 -s 10.9.0.5 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT

iptables -P FORWARD DROP
iptables -P OUTPUT DROP
iptables -P INPUT DROP

Take a screenshot of `iptables -L -v`. Ensure your screenshot is of the entire VM.

Task C.png Download



Q8 Early/Date Submission Bonus

0 Points

Bonus points for early or late submission will be added here. You may submit up to five days early for an extra 5% bonus points added to the grade of this assignment, or up to 4% deducted for late submission.

Submissions more than 4 days late are not accepted without an approved reason.