Lab 4 - Firewall

● Graded

[redacted]

Student

Alexander Lotero

Total Points

105 / 100 pts

Question 1

Introduction to iptables — **0** / 0 pts

✔ **+ 0 pts** Correct

Question 2

Task 2: Experimenting with Stateless Firewall Rules — **35** / 35 pts

2.1 Background of iptables — **0** / 0 pts

✔ **+ 0 pts** Correct

2.2 Using Iptables — **0** / 0 pts

✔ **+ 0 pts** Correct

2.3 Task 2.A: Protecting the Router — **5** / 5 pts

✔ **– 0 pts** Correct

2.4 Task 2.B: Protecting the Internal Network — **15** / 15 pts

✔ **– 0 pts** Correct

2.5 Task 2.C: Protecting Internal Servers — **15** / 15 pts

✔ **– 0 pts** Correct

Question 3

Task 3: Connection Tracking and Stateful Firewall — **30** / 30 pts

3.1 Task 3.A: Experiment with the Connection Tracking — **15** / 15 pts

✔ **– 0 pts** Correct

3.2 Task 3.B: Setting Up a Stateful Firewall — **15** / 15 pts

✔ **– 0 pts** Correct

Question 4

Task 4: Limiting Network Traffic — **10** / 10 pts

✔ **– 0 pts** Correct

Question 5

Task A - Multiple Choice

5 / 5 pts

✔ **– 0 pts** Correct

Question 6

Task B (not in the SEED Lab)

20 / 20 pts

✔ **– 0 pts** Correct

Question 7

Early/Date Submission Bonus

5 / 0 pts

✔ **+ 5 pts** Early Submission

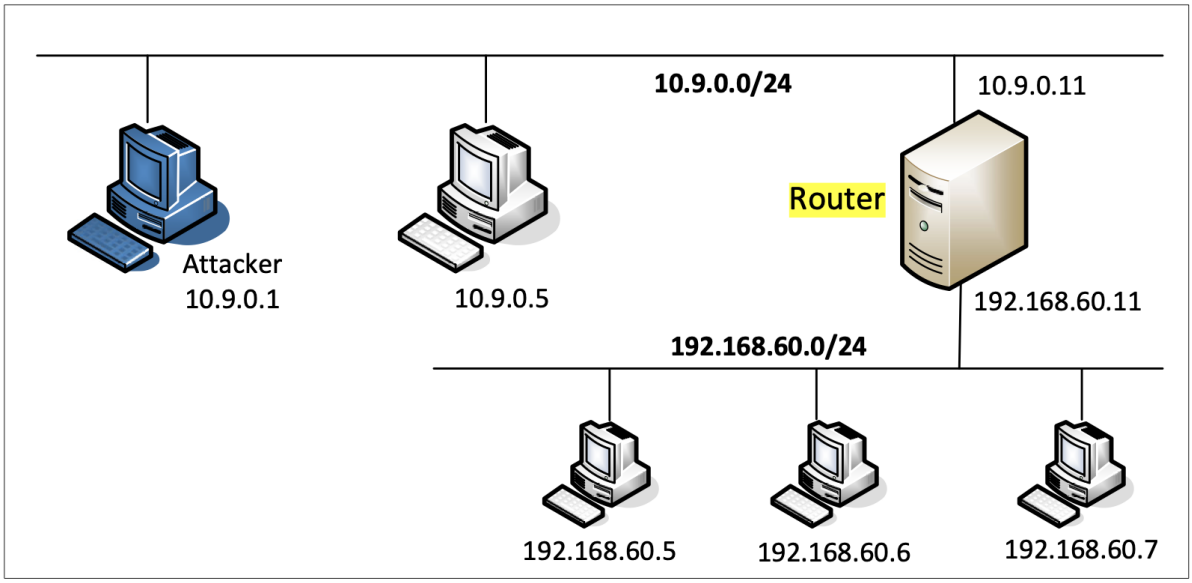
Q1 Introduction to iptables
0 Points

Introduction

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

Earliest acceptance date: 9 December (+5% bonus)
Normal due date: 14 December
Latest acceptance date: 24 December (-10% points)

Please follow the instructions in the Firewall Explorations Lab to setup the lab environment. Specifically, you would need to download the Labsetup.zip 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.

Q2.1 Background of iptables

0 Points

In the previous task *(which we skipped)*, 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.

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.

Q2.2 Using Iptables
0 Points

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.

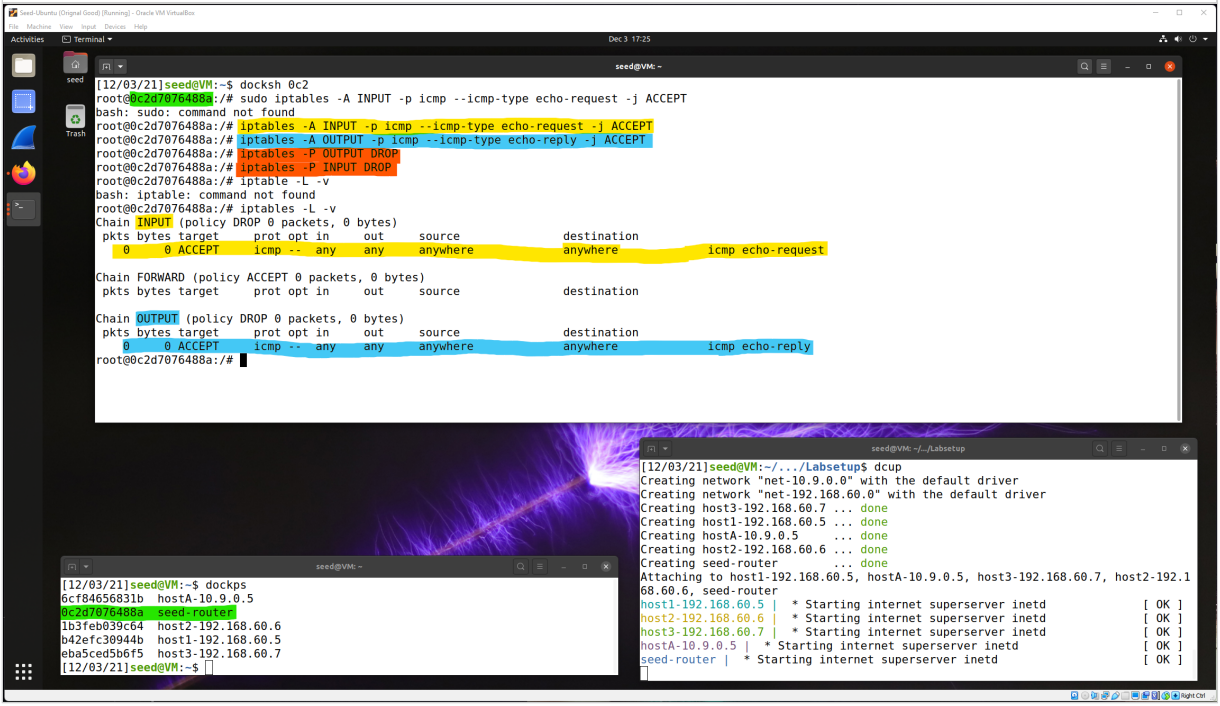
Q2.3 Task 2.A: Protecting the Router
5 Points

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
```

(0) Share a screenshot of your rules to verify that you are able to input them correctly: `iptables -L -v`

▾ Q2.3-0.png Download

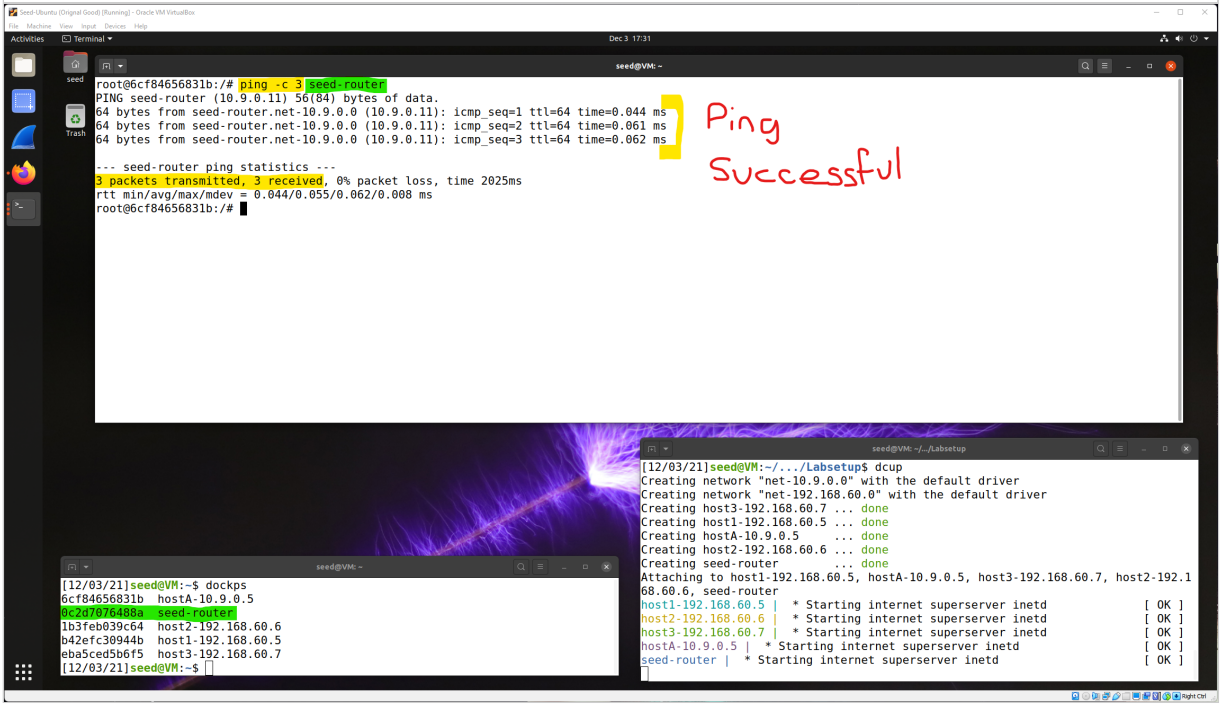


(1) Can you ping the router?

Upload your screenshots below

Q2.3-1.png

Download

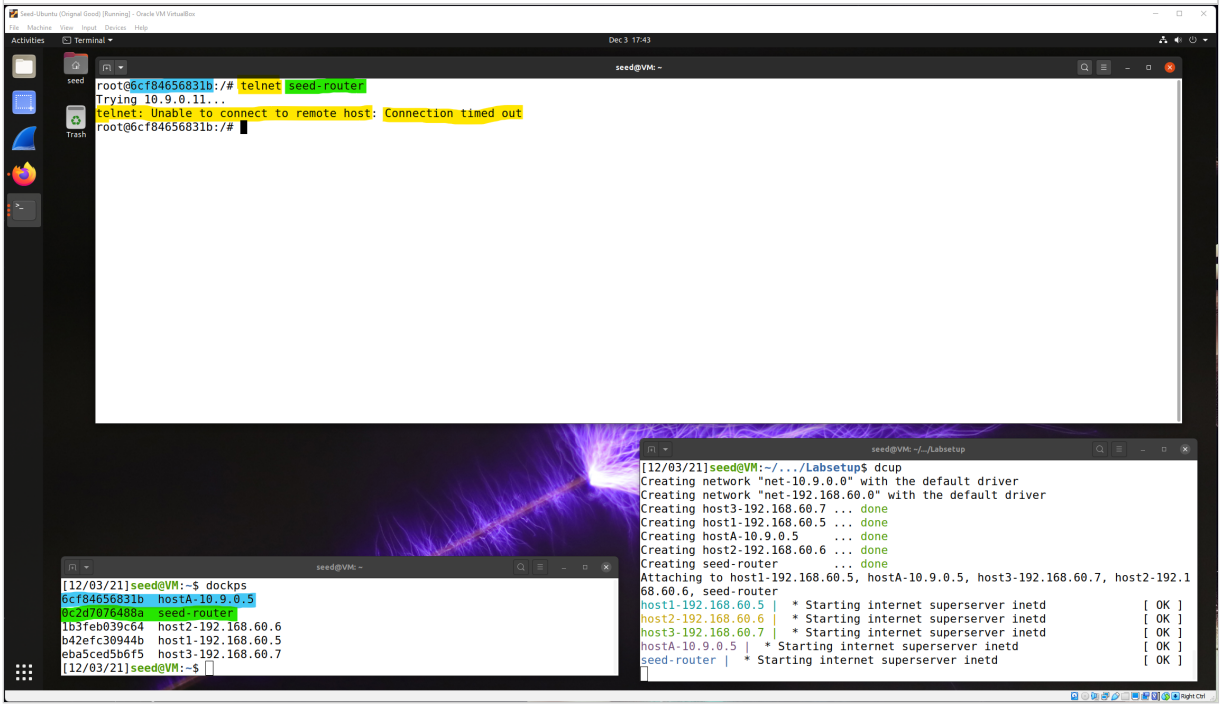


(2) Can you telnet into the router (a telnet server is running on all the containers; an account called seed was created on them with a password dees). Please report your observation and explain the purpose for each rule.

Upload your screenshots below

Q2.3-2.png

Download



Describe your observation and explain the purpose for each rule below

The telnet attempt from 10.9.0.5 to the seed-router was unsuccessful because our set of iptables rules only allow ping traffic ("--icmp-type echo-

request"). Thus, the telnet traffic is dropped, rendering our telnet attempt unsuccessful. This is further explained by examining what each rule does specifically;

"iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT": explicitly allows ping traffic coming to the router (input).
"iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT": explicitly allows ping traffic being sent by the router (output).
"iptables -P OUTPUT DROP": denies (drops, to be specific) all other traffic being sent by the router (output).
"iptables -P INPUT DROP": denies (drops, to be specific) all other traffic coming to the router (input).

These set of rules employ an explicit allow, implicit deny method.

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>
```

Q2.4 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.

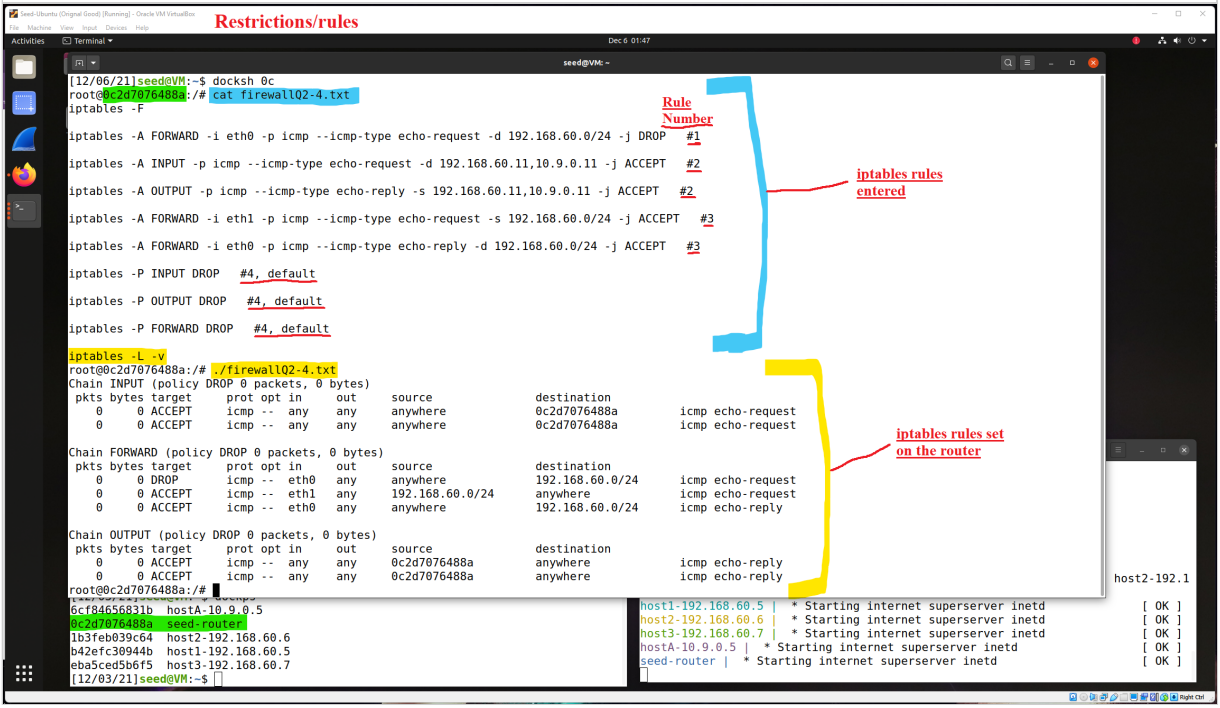
You will need to use the "-p icmp" options to specify the match options related to the ICMP protocol. You can run "iptables -p icmp -h" to find out all the ICMP match options. The following example drops the ICMP echo request.

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

Submit a screenshot of your iptable rules from the router `iptables -L -v`

Q2.4-Rules.png

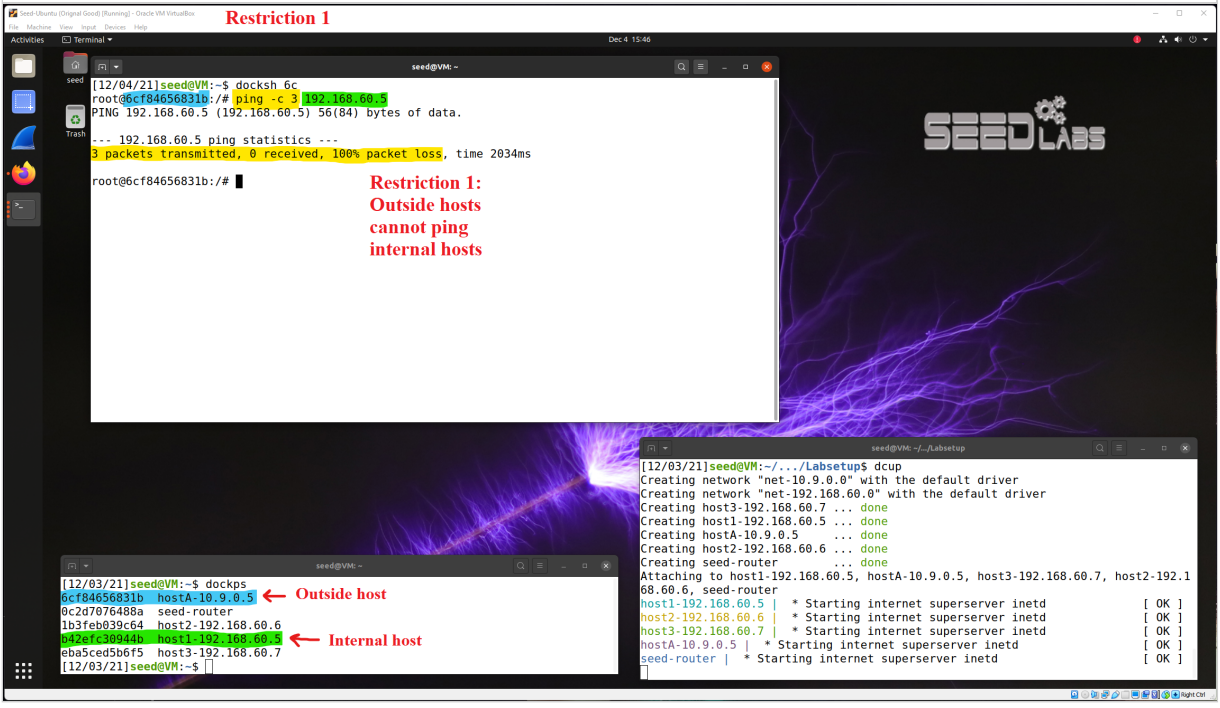
Download



Upload screenshots of showing that items 1-4 above function as specify. Please add a description if it's not very clear.

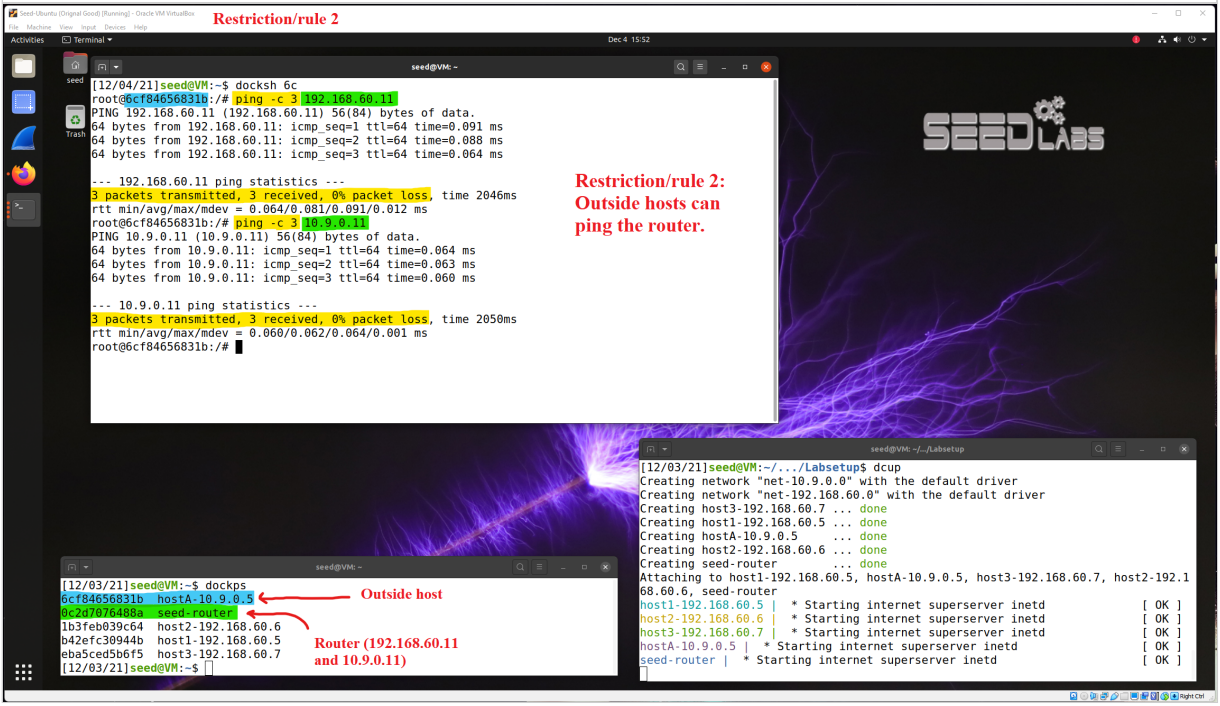
Q2.4-Restriction1.png

Download



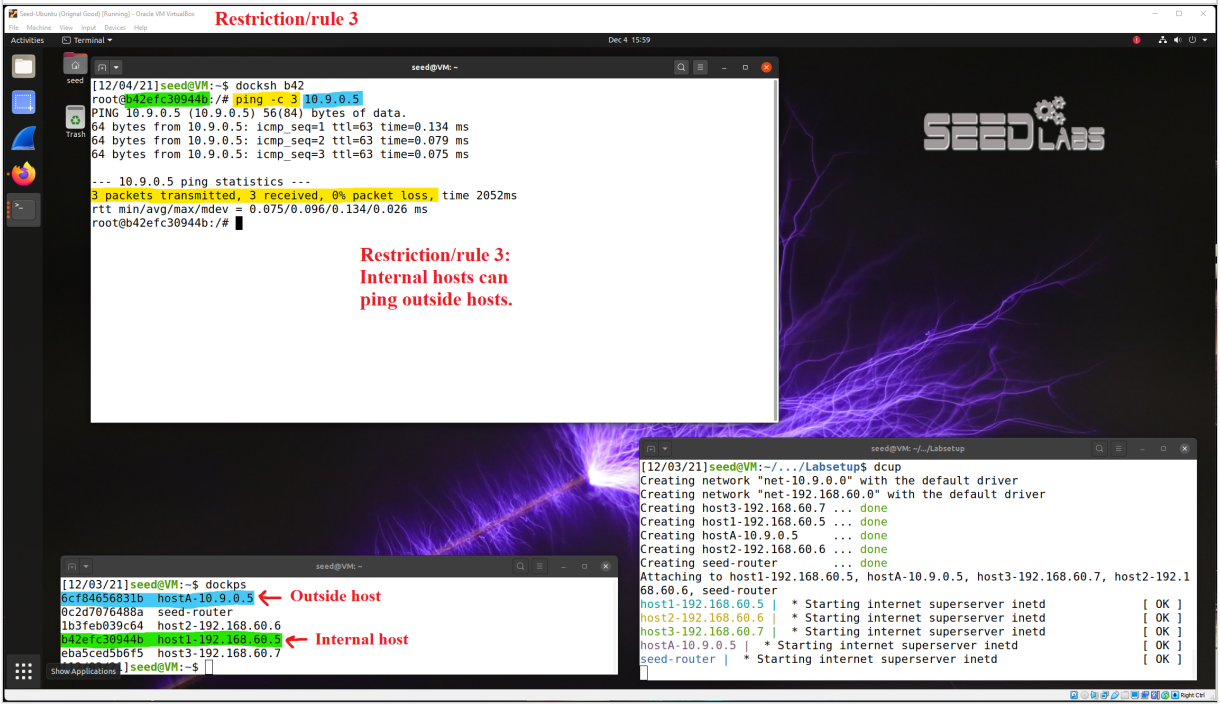
Q2.4-Restriction2.png

Download



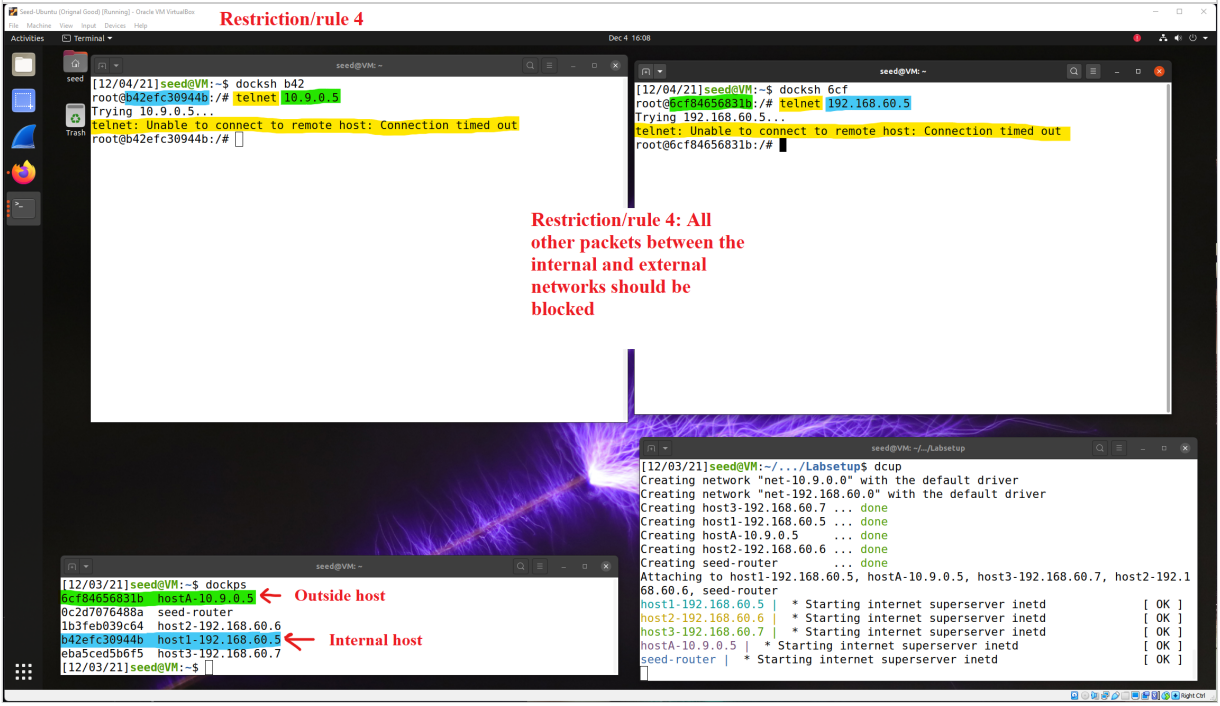
Q2.4-Restriction3.png

Download



Q2.4-Restriction4.png

Download



Q2.5 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

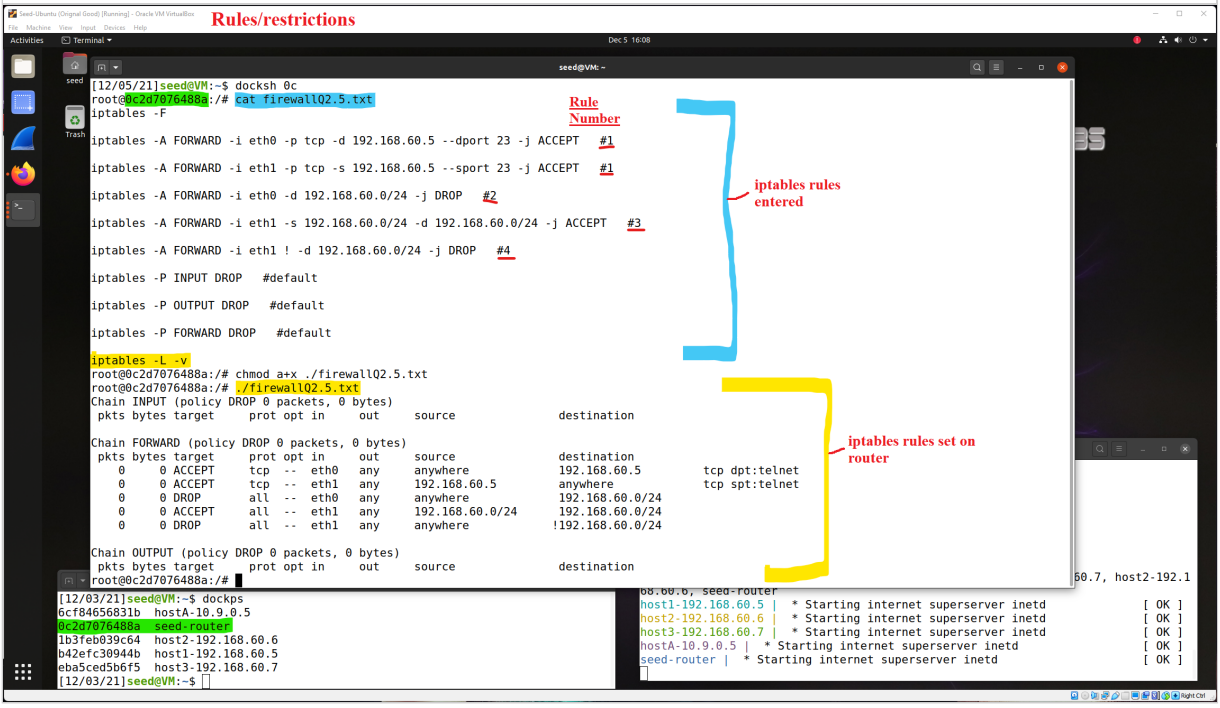
Note: "Outside" refers to the internet, not just the 10.9.0.11/24 network.

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
```

Submit a screenshot of your iptable rules from the router `iptables -L -v`

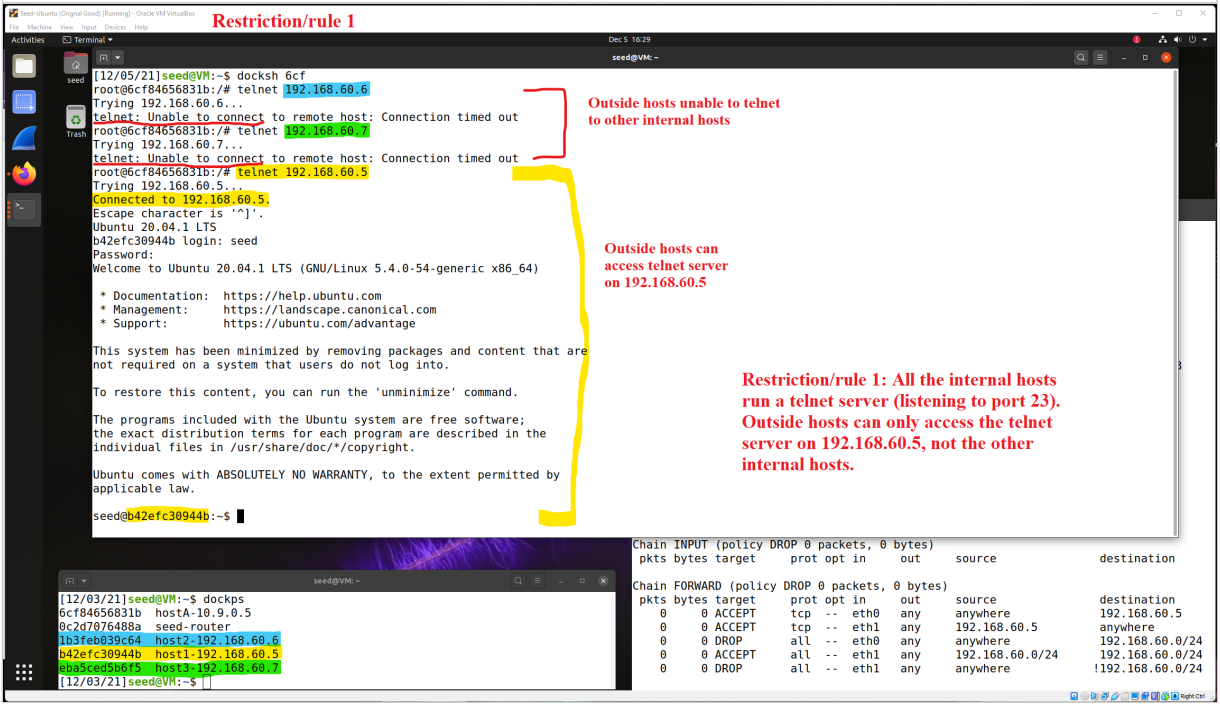
Q2.5-Rules.png Download



Upload screenshots of showing that items 1-5 above function as specify. Please add a description if it's not very clear.

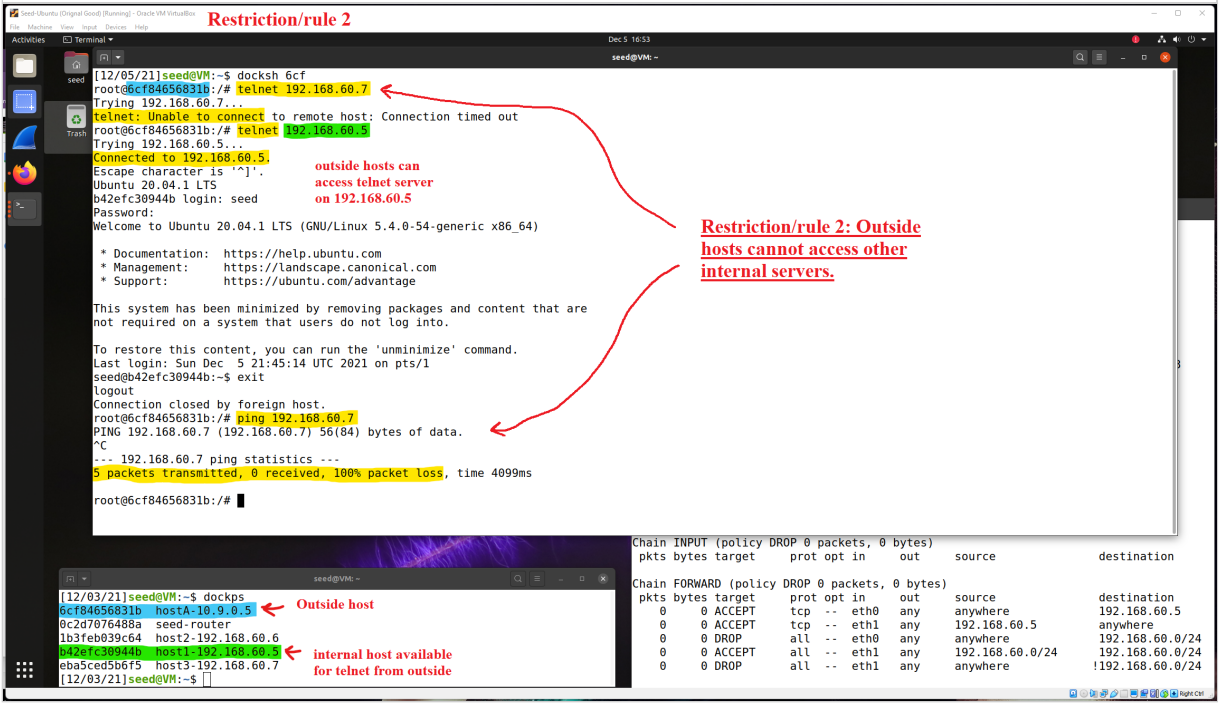
Q2.5-Restriction1.png

Download



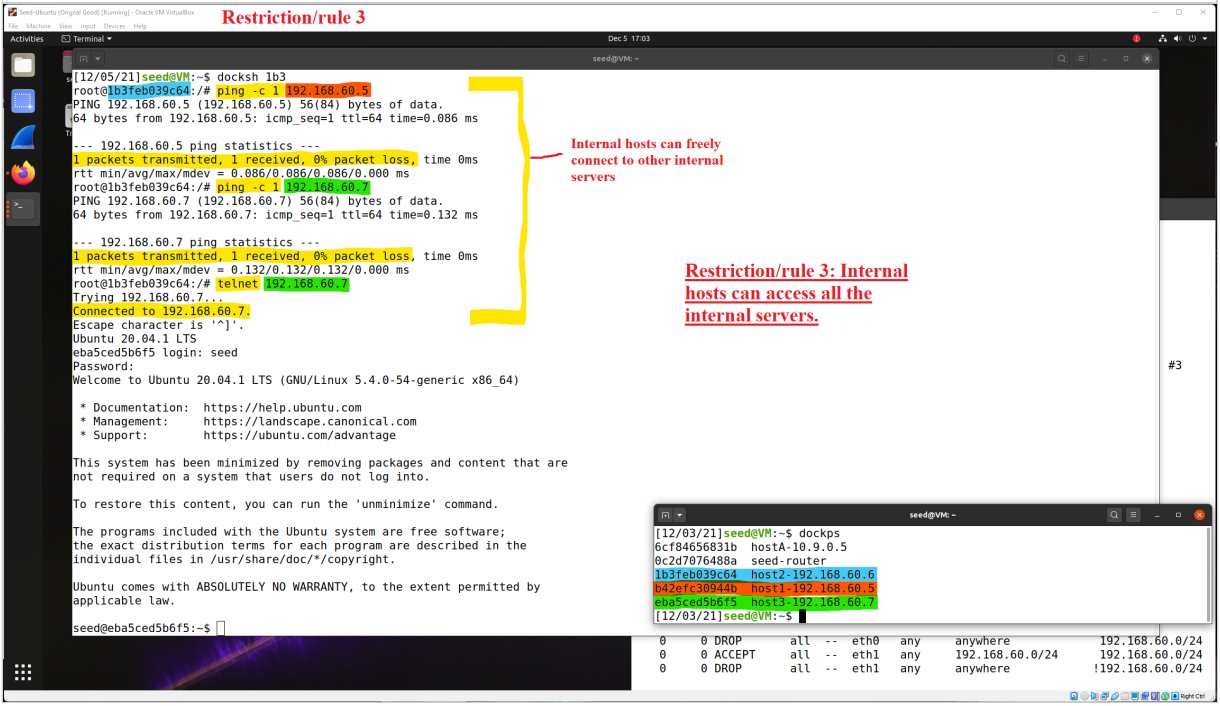
Q2.5-Restriction2.png

Download



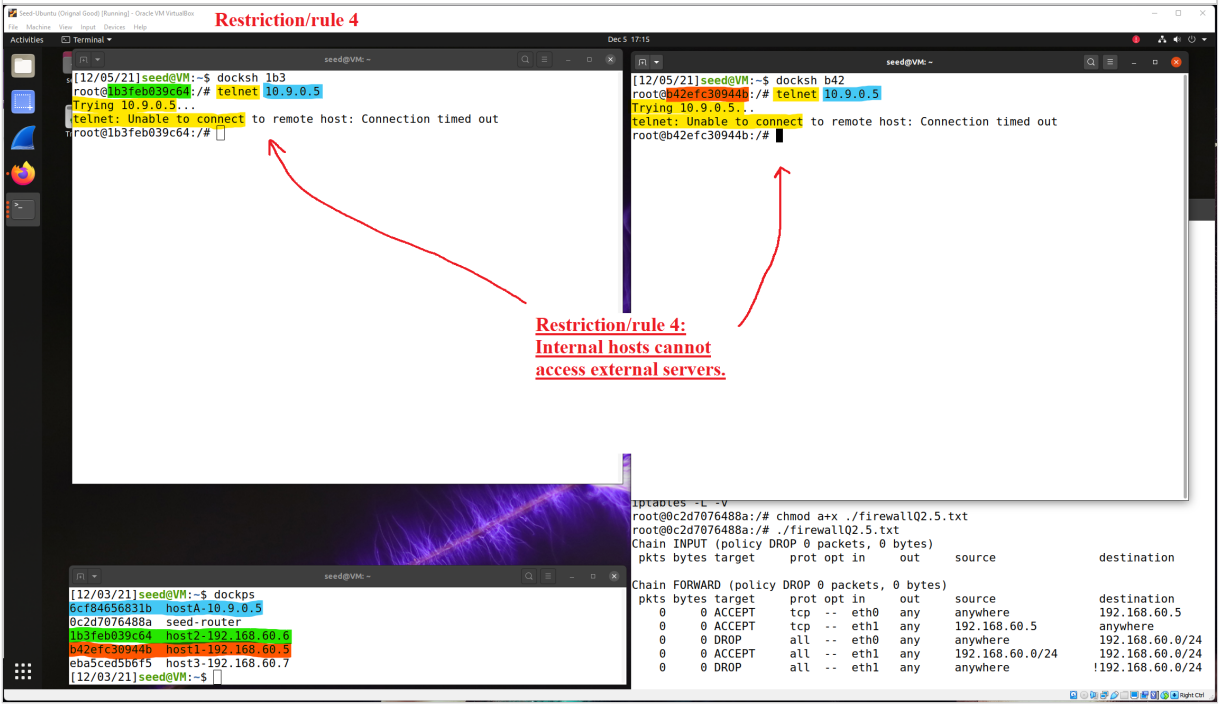
Q2.5-Restriction3.png

Download



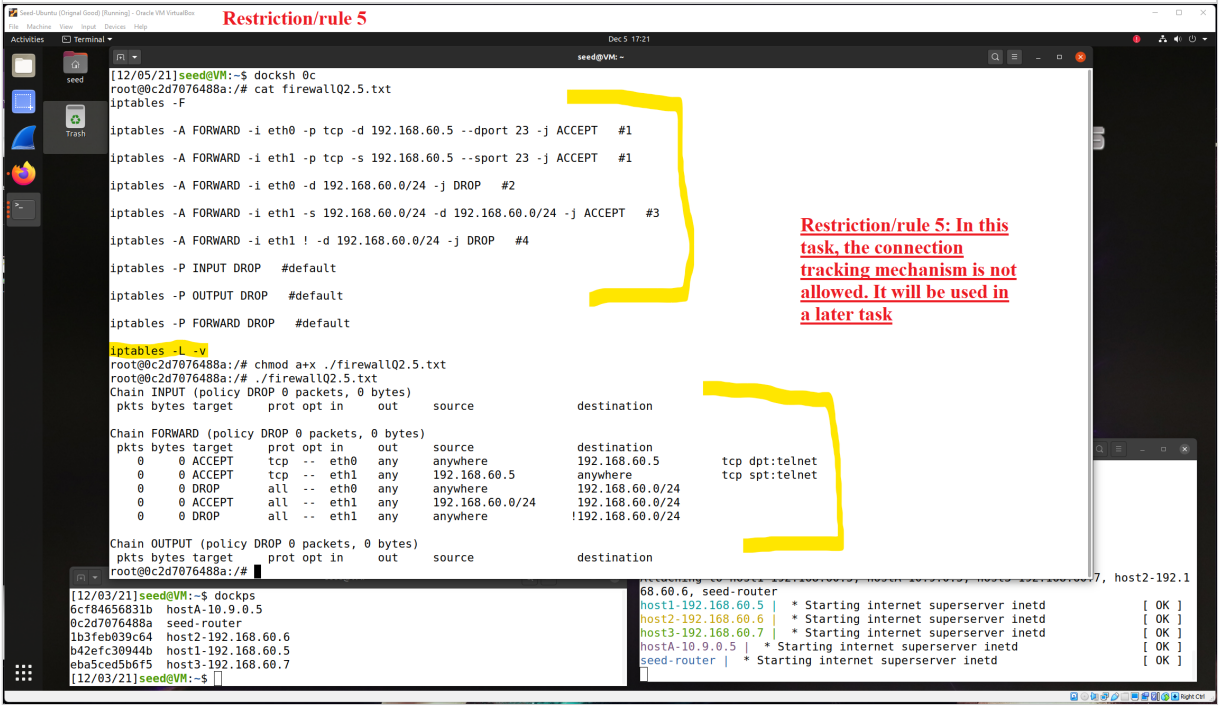
Q2.5-Restriction4.png

Download



Q2.5-Restriction5.png

Download



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

Q3 Task 3: Connection Tracking and Stateful Firewall

30 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
```

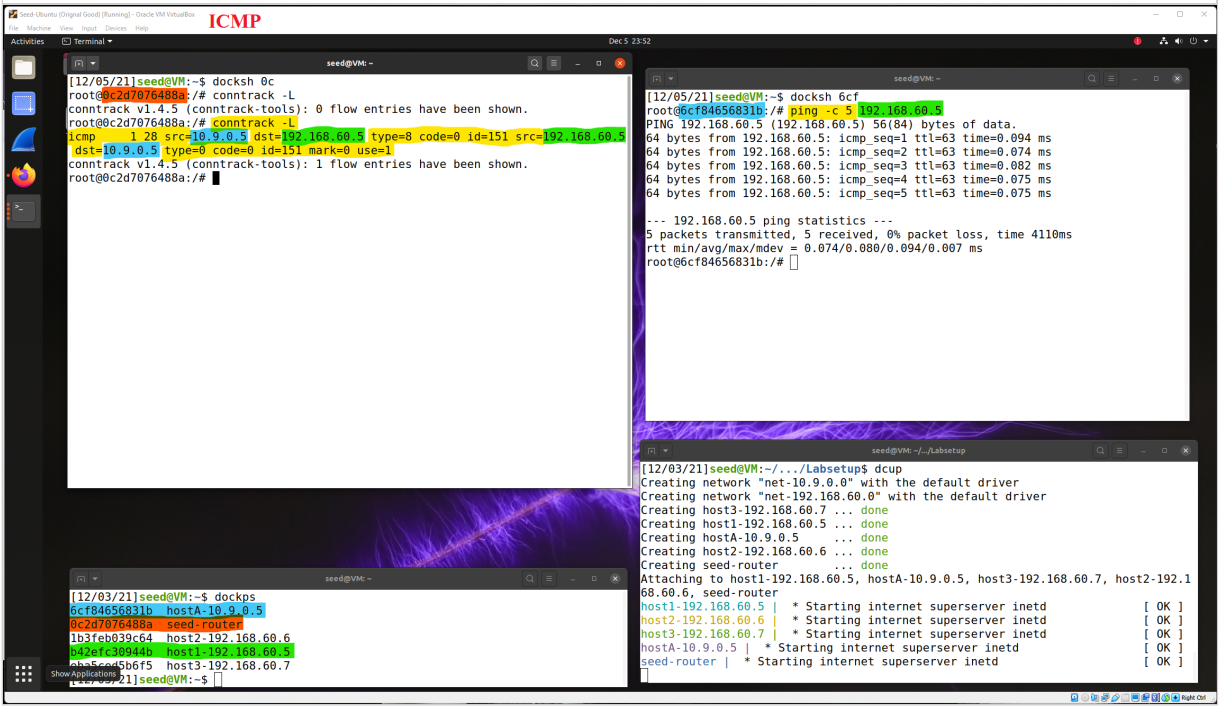
The goal of the task is to use a series of experiments to help students understand the connection concept in this tracking mechanism, especially for the ICMP and UDP protocols, because unlike TCP, they do not have connections. Please conduct the following experiments. For each experiment, please describe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

• 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

▾ Q3.1-ICMP.png Download



Describe your observation for ICMP

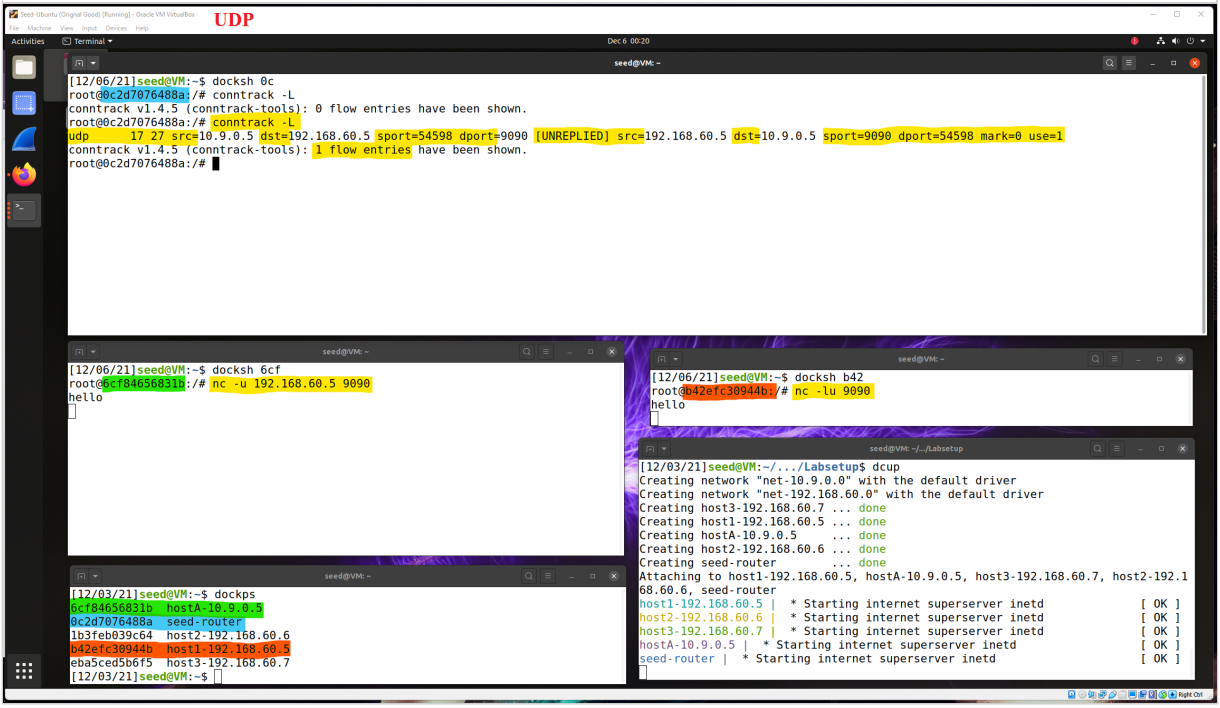
After examining the conntrack entry for the ping from 10.9.0.5 to 192.168.60.5, here are a few observations:
First, despite sending several ICMP packets as part of the ping process, conntrack only maintains a single flow entry to represent that instance of communication between the two hosts. This is likely to maintain scalable efficiency, particularly in routers that must track many connections and quickly locate the status of a connection in order to enforce stateful firewall rules.
Second, the ICMP connection is only kept for 30 seconds. The time remaining before the connection disappears from the conntrack table is on of the values listed as part of that connection's entry in the table. Once again, this is likely to preserve efficiency by removing entries quick enough to avoid tracking too many at once.

• 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

▾ Q3.1-UDP.png Download



Describe your observation for UDP

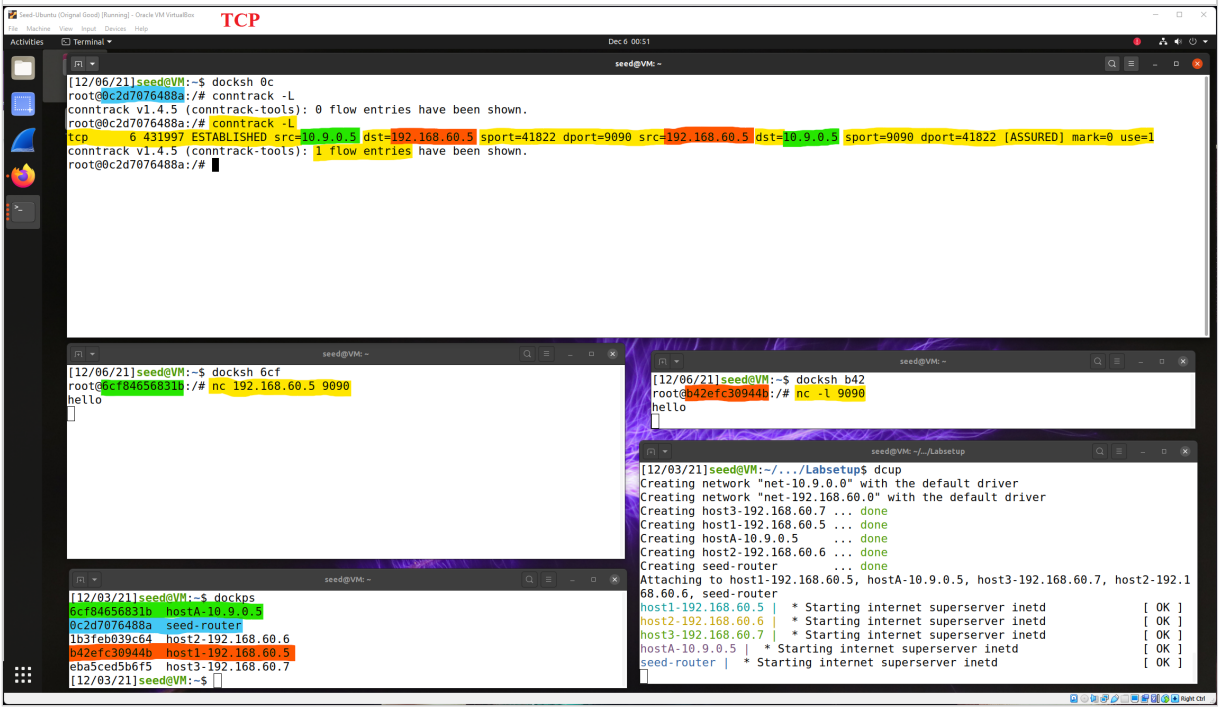
The UDP entry in the conntrack table contains important details about the "connection," including; source IP and port, destination IP and port, time before the entry is forgotten, etc. Notably however, unlike the ICMP entry, this UDP "connection" entry includes an "[UNREPLIED]" flag. This is likely due to the "fire-and-forget" nature of UDP. A connection isn't established, rather one host is sending packets to another. "[UNREPLIED]" refers to one-way traffic, this communication session has only seen traffic in one direction. The UDP entry is maintained in the conntrack table for 30 seconds. This timer is reset every time a new packet is sent, in this case, whenever the enter/return key is pressed on the 10.9.0.5 machine.

• 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>
```

Upload your screenshots for TCP

Q3.1-TCP.png Download



Describe your observation for TCP

The TCP connection entry retains much of the same relavent information as the UDP entry without the "[UNREPLIED]" with the addition of the "ESTABLISHED" and "[ASSURED]" flags. I suspect that the "ESTABLISHED" reflects the fact that the TCP 3-way handshake has been completed successfully. "[ASSURED]" is the counterpart to "[UNREPLIED]," referring to the fact that traffic has been identified going both directions.
The TCP connection appears to remain in the conntrack table for 12 hours. The countdown continues until a new packet is sent, in this case the enter/return key has been pressed on the 10.9.0.5 machine has been pressed. Each new packet refreshes the counter back to 12 hours. The timer only reflects 12 hours while both hosts remain connected. After one or both of the communicating parties disconnects, the timer decreases to 2 minutes. After that 2 minutes, the entry is removed from the conntrack table.

Q3.2 Task 3.B: Setting Up a Stateful Firewall
15 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.

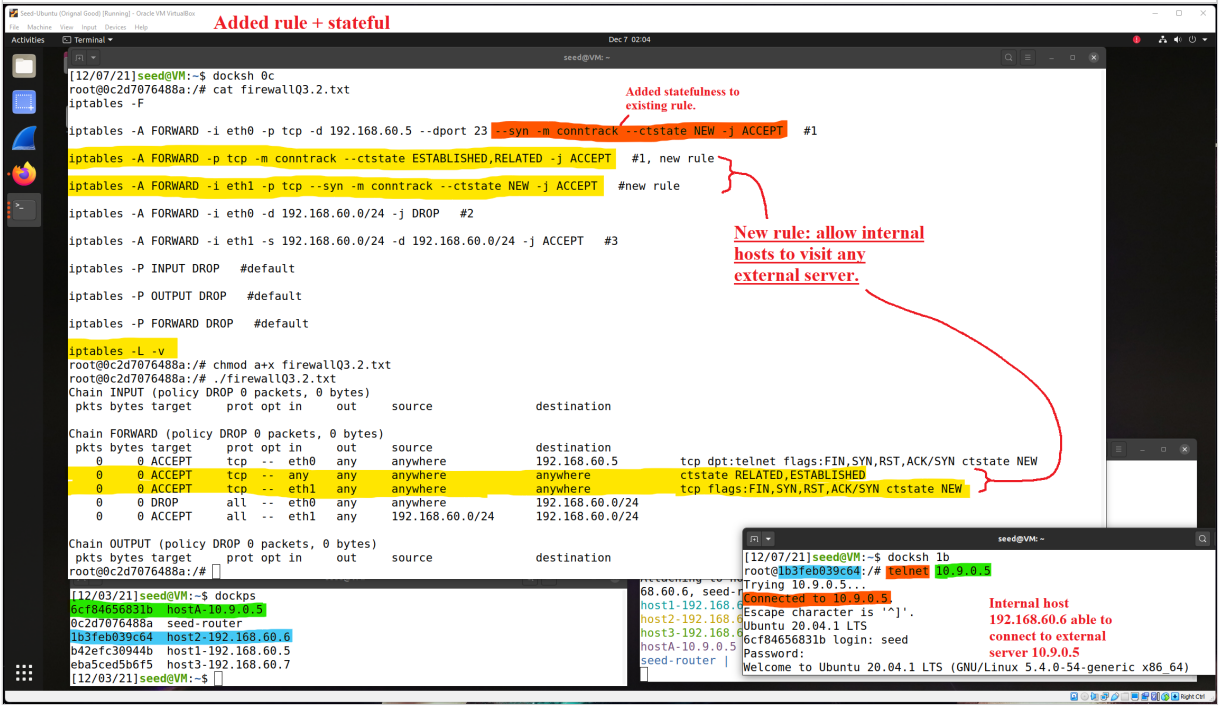
```
iptables -P FORWARD DROP
```

Please rewrite the firewall rules in Q2.5 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). 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.

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

Q3.2.png

Download



Explain the advantages and disadvantages of each approach.

The immediate advantage of using connection tracking is flexibility. Particularly in the case of TCP connections, it enables the option to allow connections to be allowed only if initiated from a particular side (e.g .the internal network) and not the other (e.g. the outside network). Disadvantages of connection tracking, or rather advantages of stateful firewalls include independence and potential simplicity. By independence, I am referring to the fact that a firewall using conntrack relies on the contrack table to function properly. If something goes wrong with the table, e.g the table fills up, then the firewall rules may become difficult to enforce properly. In the case of simplicity, and while this may be more of an opinion than a fact, because a firewall without connection tracking is a "dumb" firewall, it can be easier to read and write "dumb" rules.

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.

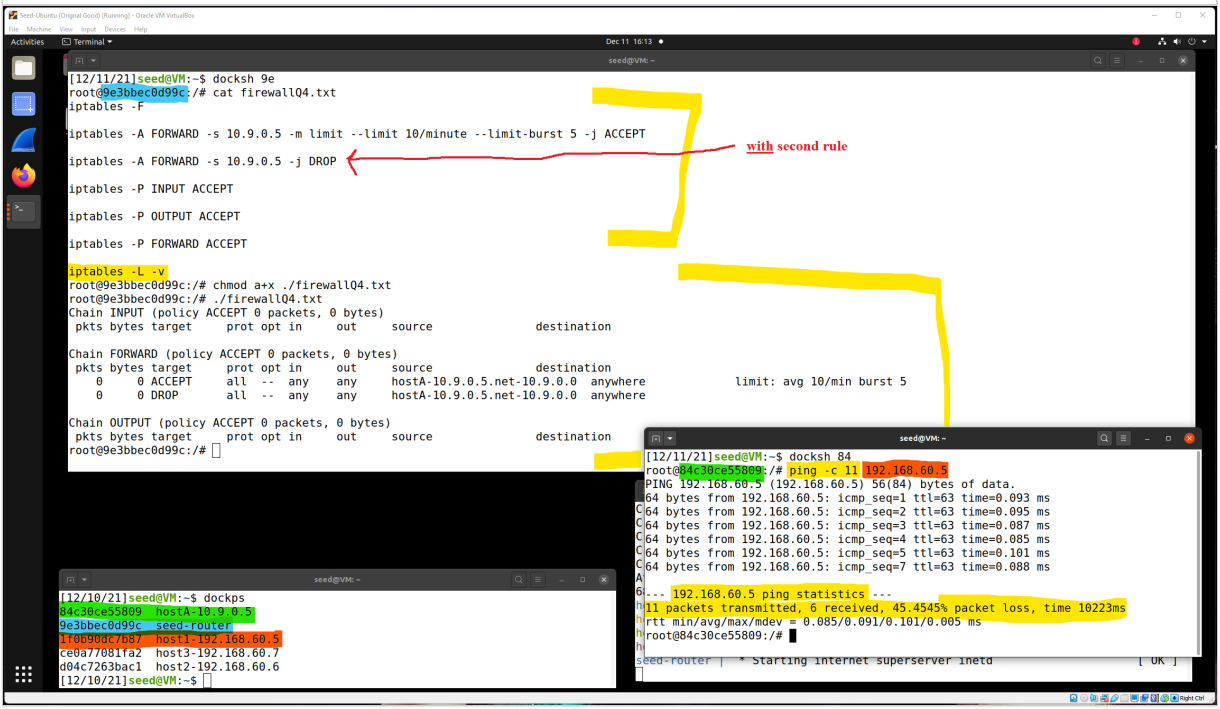
```
$ iptables -m limit -h
limit match options:
--limit avg           max average match rate: default 3/hour
                      [Packets per second unless followed by
                      /sec /minute /hour /day postfixes]
--limit-burst number    number to match in a burst, default 5
```

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.

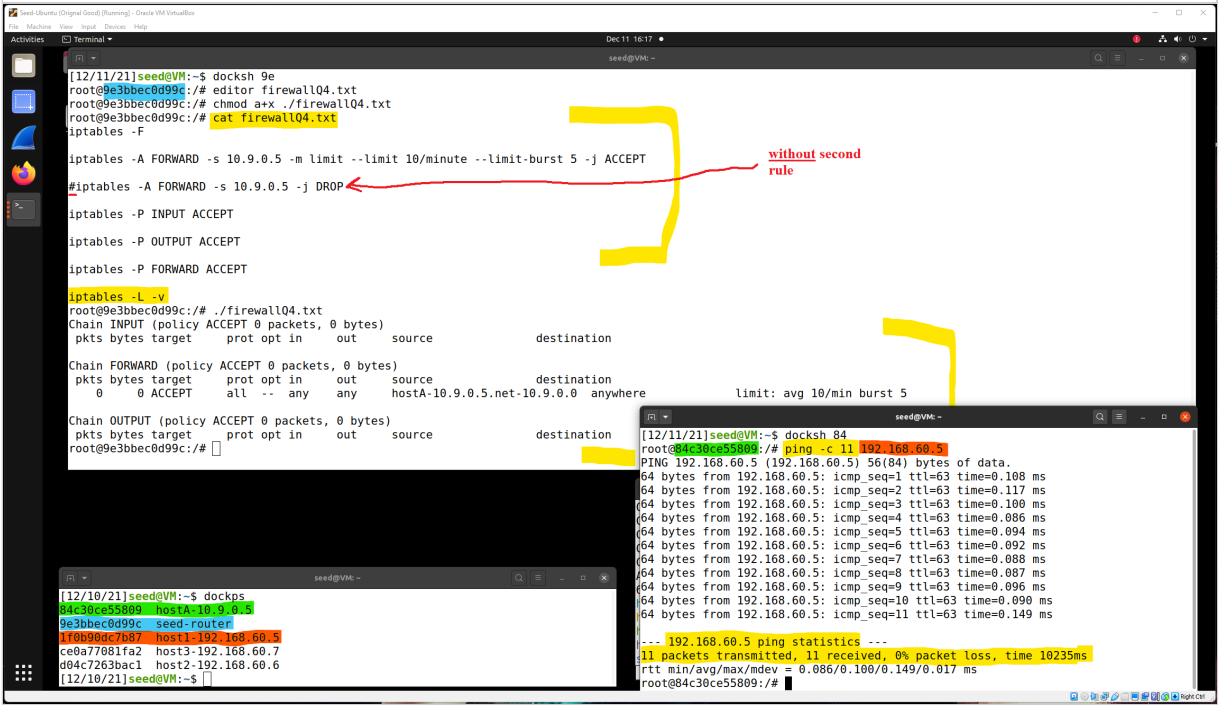
```
iptables -A FORWARD -s 10.9.0.5 -m limit \
        --limit 10/minute --limit-burst 5 -j ACCEPT
iptables -A FORWARD -s 10.9.0.5 -j DROP
```

Upload your IPTables rules `iptables -L -v`

Q4-with.png Download



Q4-without.png Download



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

As you can see from the provided screenshots, the second rule is required. Without the second rule, ping from 10.9.0.5 to 192.168.60.5 appear to be allowed without any enforcement of the limit. With the second rule, the ping packets are actually limited. In the included screenshot "Q4-with.png," you can see that of the 11 packets sent by 10.9.0.5, only 6 response packets were received (45.4545% packet loss). This demonstrates that the traffic from 10.9.0.5 to 192.168.60.5 is being properly enforced based on the iptables rules entered provided.

Q5 Task A - Multiple Choice
5 Points

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

- ◯ INPUT
- ◯ OUTPUT
- ◯ FORWARD

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

- ◯ INPUT
- ◯ OUTPUT
- ◯ FORWARD

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

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.

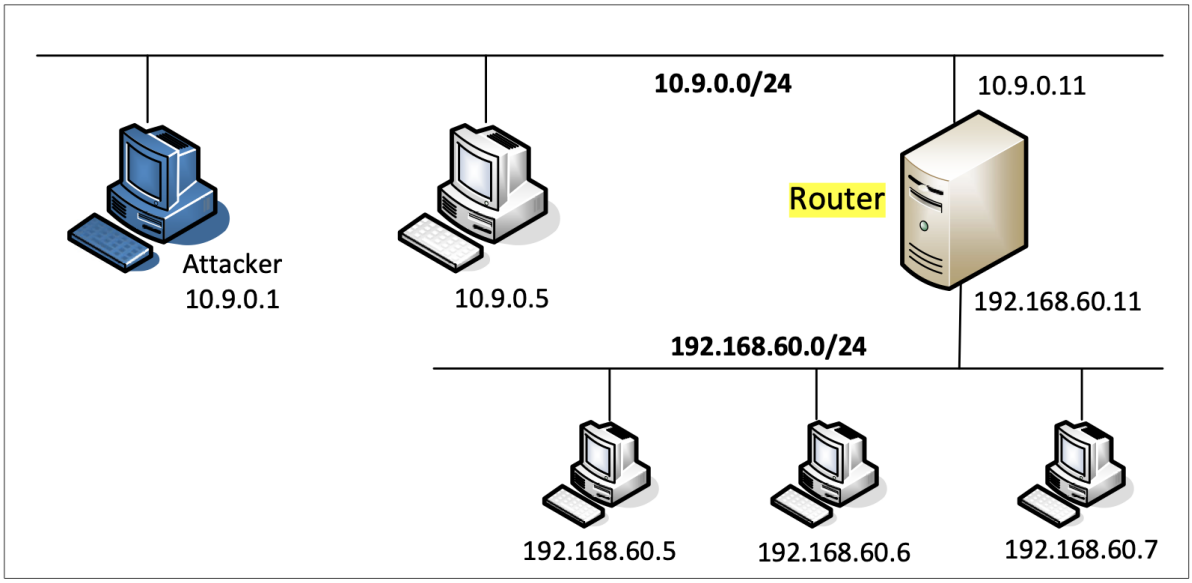
- ◯ DROP
- ◯ REJECT

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 the SEED Lab)
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 -F

#Rule1:
iptables -A FORWARD -i eth1 -s 192.168.60.0/24 -d 10.9.0.0/24 -m conntrack --ctstate NEW -j ACCEPT   #1
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT   #1

#Rule 2:
iptables -A INPUT -p icmp --icmp-type echo-request -d 192.168.60.11,10.9.0.11 -m conntrack --ctstate NEW,ESTABLISHED,RELATED -j ACCEPT   #2
iptables -A OUTPUT -p icmp --icmp-type echo-reply -s 192.168.60.11,10.9.0.11 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT   #2
```

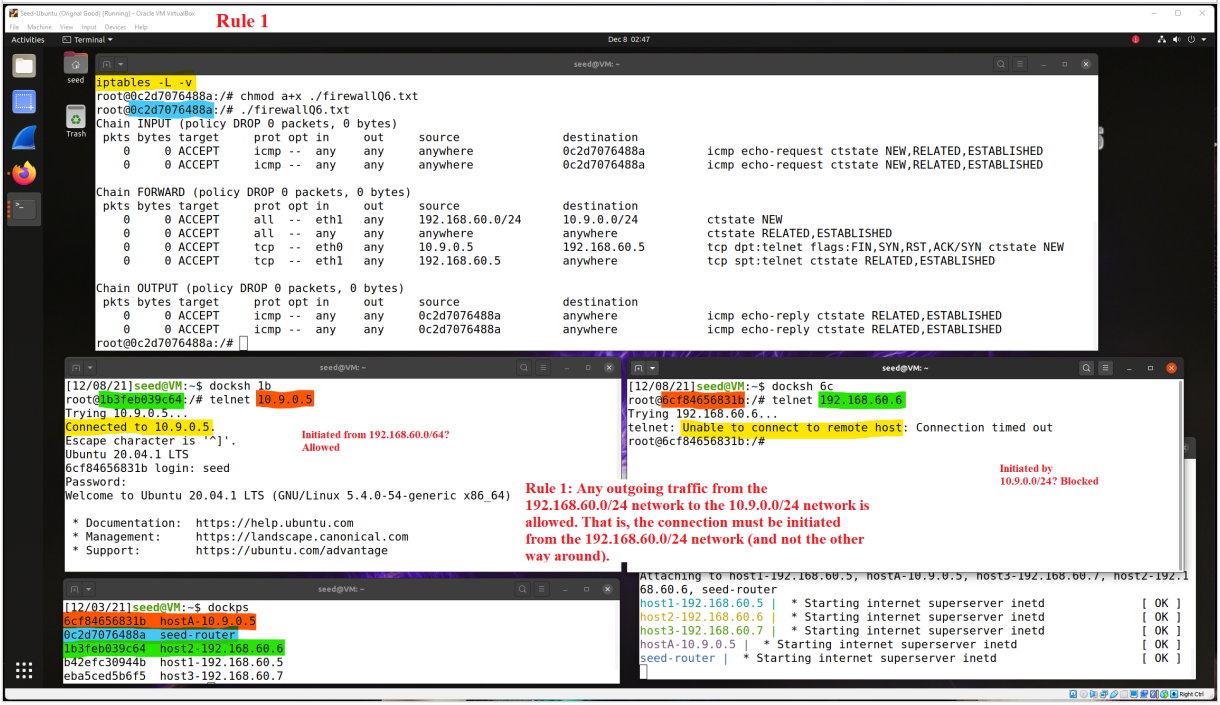
```
#Rule 3:
iptables -A FORWARD -i eth0 -p tcp -s 10.9.0.5 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT   #3
iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 --sport 23 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT   #3

#Default rules:
iptables -P INPUT DROP   #default
iptables -P OUTPUT DROP   #default
iptables -P FORWARD DROP   #default

iptables -L -v
```

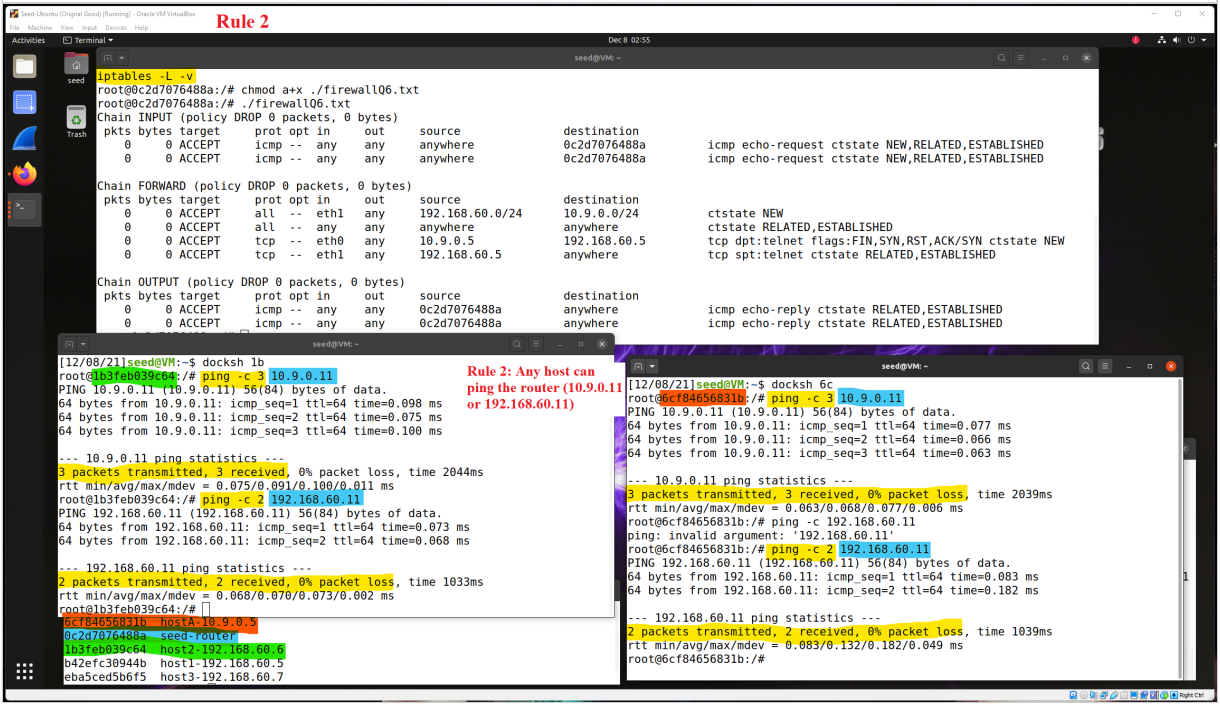
Take a screenshot of `iptables -L -v`. Ensure your screenshot is of the entire VM.

Q6-rule1.png Download



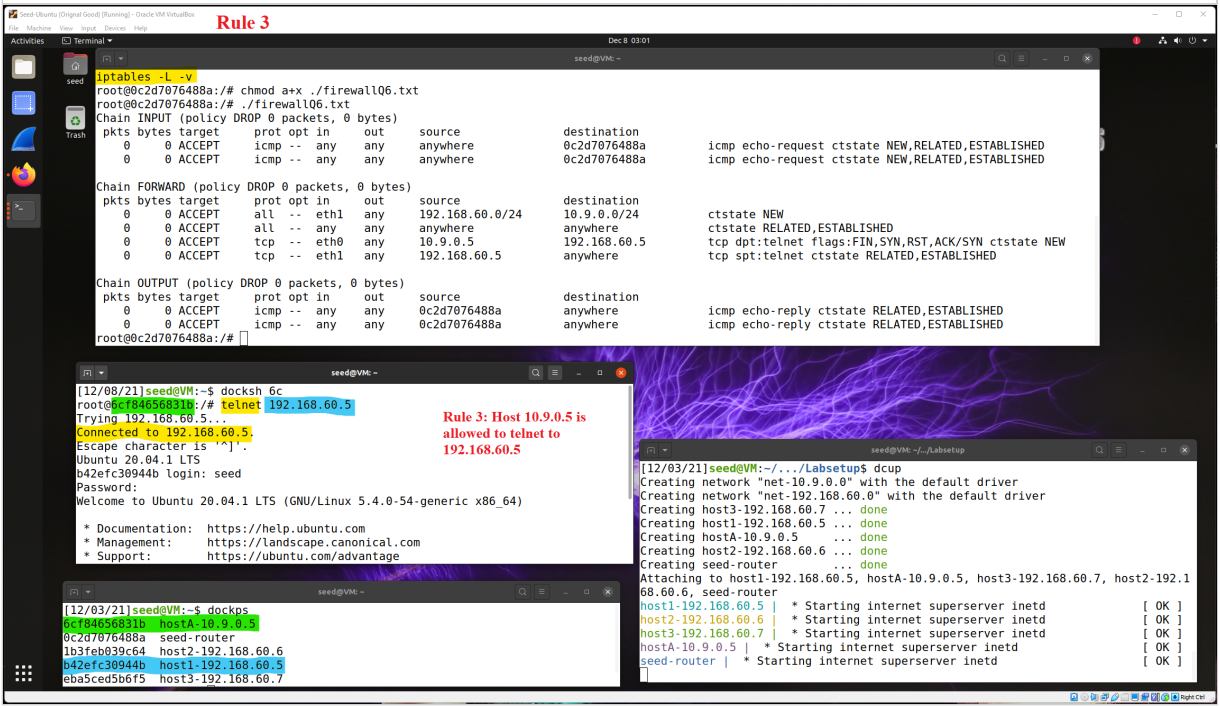
Q6-rule2.png

Download

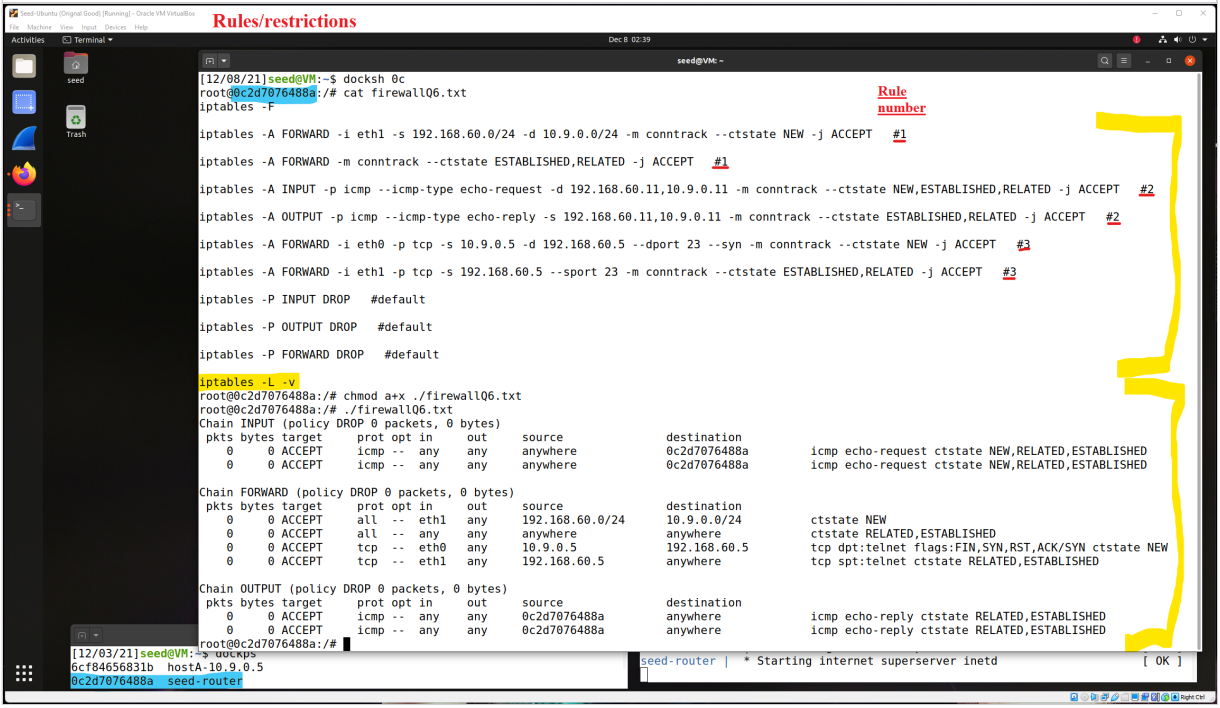


Q6-rule3.png

Download



Q6-rules.png Download



Q7 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 10% deducted for late submission.

Submissions more than 10 days late are not accepted without a medical or work approved reason.