Firewall Exploration Lab

Contents

Lab Setup

Please download the Labsetup.zip file from the below link to your VM, unzip it, enter the Labsetup folder, and use the docker-compose.yml file to set up the lab environment.

https://seedsecuritylabs.org/Labs_20.04/Files/Firewall/Labsetup.zip

In this lab, we need to use multiple machines. Their setup is depicted in Figure 1. We will use containers to set up this lab environment..

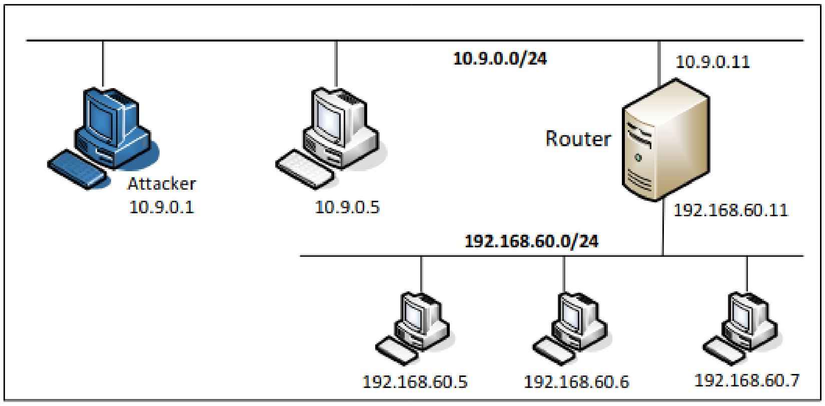


Figure 1: Lab Setup

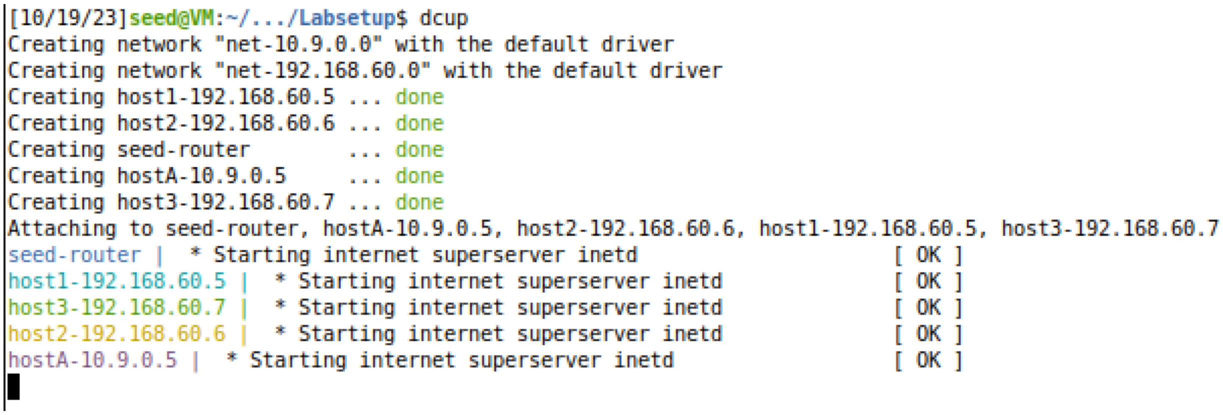
Lab Overview

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

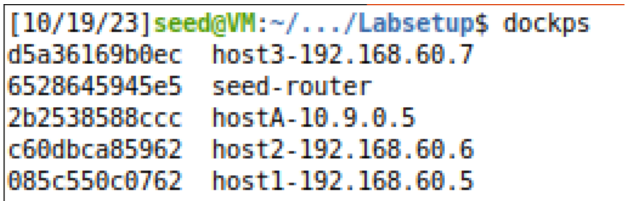
Through this implementation task, students can get basic ideas on how a firewall works. Actually, Linux already has a built-in firewall, also based on netfilter. This firewall is called iptables. Students will be given simple network topology, and are asked to use iptables to set up firewall rules to protect the network. Students will also be exposed to several other interesting applications of iptables.

This lab covers the following topics:

- Firewall
- Netfilter
- Loadable kernel module
- Using iptables to set up firewall rules
- Various applications of iptables



```
[10/19/23]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
Creating host1-192.168.60.5 ... done
Creating host2-192.168.60.6 ... done
Creating seed-router        ... done
Creating hostA-10.9.0.5     ... done
Creating host3-192.168.60.7 ... done
Attaching to seed-router, hostA-10.9.0.5, host2-192.168.60.6, host1-192.168.60.5, host3-192.168.60.7
seed-router |  * Starting internet superserver inetd                           [ OK ]
host1-192.168.60.5 |  * Starting internet superserver inetd                    [ OK ]
host3-192.168.60.7 |  * Starting internet superserver inetd                    [ OK ]
host2-192.168.60.6 |  * Starting internet superserver inetd                    [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                        [ OK ]
```



```
[10/19/23]seed@VM:~/.../Labsetup$ dockps
d5a36169b0ec  host3-192.168.60.7
6528645945e5  seed-router
2b2538588ccc  hostA-10.9.0.5
c60dbca85962  host2-192.168.60.6
085c550c0762  host1-192.168.60.5
```

Task 1: Implementing a Simple Firewall

In this task, we will implement a simple packet filtering type of firewall, which inspects each incoming and outgoing packet, and enforces the firewall policies set by the administrator. Since the packet processing is done within the kernel, the filtering must also be done within the kernel. Therefore, it seems that implementing such a firewall requires us to modify the Linux kernel. In the past, this had to be done by modifying and rebuilding the kernel. The modern Linux operating systems provide several new mechanisms to facilitate the manipulation of packets without rebuilding the kernel image. These two mechanisms are Loadable Kernel Module (LKM) and Netfilter.

Notes about containers - Since all the containers share the same kernel, kernel modules are global. Therefore, if we set a kernel model from a container, it affects all the containers and the host. For this reason, it does not matter where you set the kernel module. We will just set the kernel module from the host VM in this lab**.**

Another thing to keep in mind is that containers' IP addresses are virtual. Packets going to these virtual IP addresses may not traverse the same path as what is described in the Netfilter document.

Therefore, in this task, to avoid confusion, we will try to avoid using those virtual addresses. We do most tasks on the host VM. The containers are mainly for other tasks.

Task 1.A: Implement a Simple Kernel Module

LKM allows us to add a new module to the kernel at runtime. This new module enables us to extend the functionalities of the kernel, without rebuilding the kernel or even rebooting the computer. The packet filtering part of a firewall can be implemented as an LKM. In this task, we will get familiar with LKM.

The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded; when the module is removed from the kernel, it prints out "Bye-bye World!".

The messages are not printed out on the screen; they are actually printed into the /var/log/syslog file. You can use "dmesg" to view the messages.

hello. c - included in the lab setup files

```
#include <linux/module.h>
#include <linux/kernel.h>
int initialization(void)
{
    printk(KERN_INFO "Hello World!\n");
    return 0;
}
void cleanup(void)
{
    printk(KERN_INFO "Bye-bye World!.\n");
}
module_init(initialization);
module_exit(cleanup);
MODULE_LICENSE("GPL");
```

Note - Please use the VM’s Terminal and not the containers given in the lab setup for this sub task.

On the VM - Open two Terminal Tabs, one to load the module and the other to view the messages.

- Use one terminal window to view the messages

Command:

- **$ sudo dmesg -k -w**

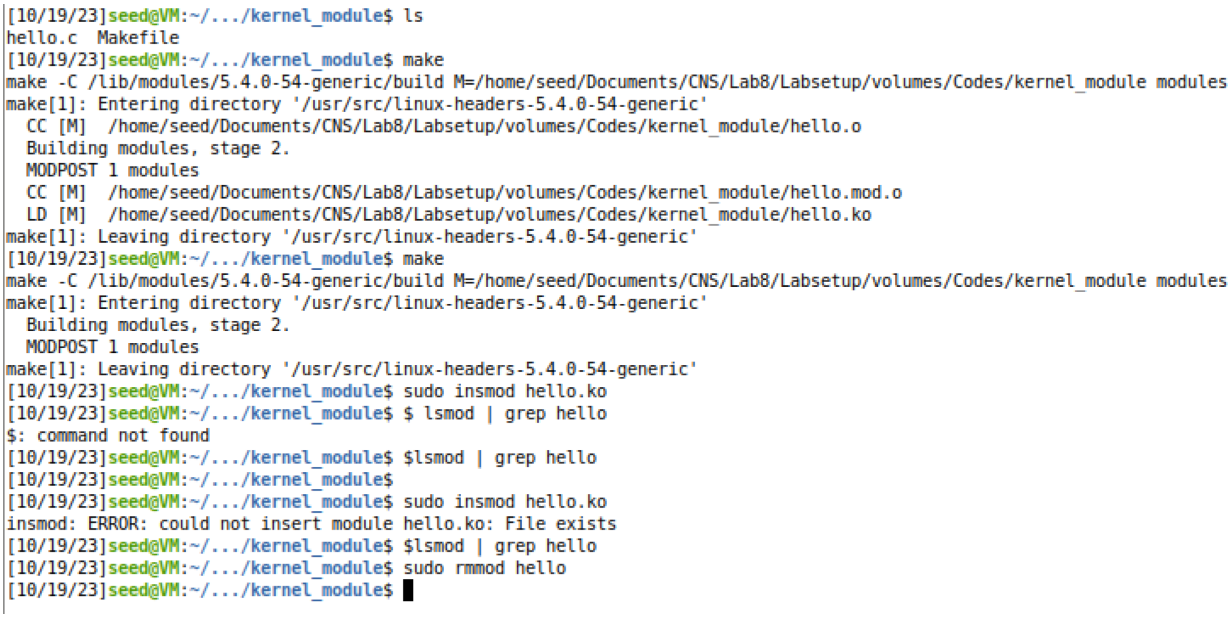
- The other to Load and Remove the Kernel

(**Change the directory to the kernel_module folder in 'Codes'** (given to the students) before executing the below)

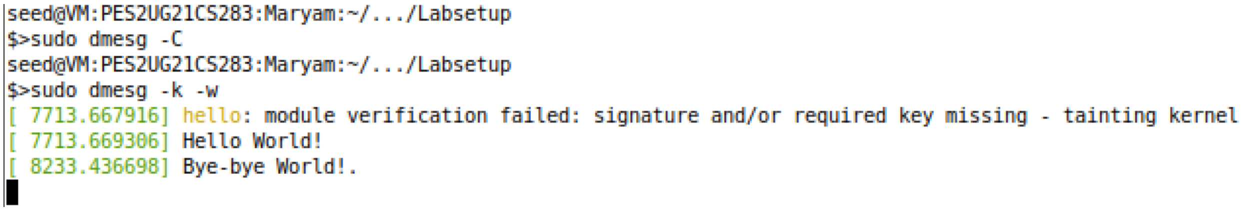
Command:

- **$ make**
  $ sudo insmod hello.ko (inserting a module)
  $ lsmod | grep hello (list modules)
  $ sudo rmmod hello

Provide screenshots of your terminal with the required explanations of the same.



```
[10/19/23]seed@VM:~/.../kernel_module$ ls
hello.c  Makefile
[10/19/23]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/kernel_module/hello.mod.o
  LD [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[10/19/23]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[10/19/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[10/19/23]seed@VM:~/.../kernel_module$ $ lsmod | grep hello
$: command not found
[10/19/23]seed@VM:~/.../kernel_module$ $lsmod | grep hello
[10/19/23]seed@VM:~/.../kernel_module$
[10/19/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
insmod: ERROR: could not insert module hello.ko: File exists
[10/19/23]seed@VM:~/.../kernel_module$ $lsmod | grep hello
[10/19/23]seed@VM:~/.../kernel_module$ sudo rmmod hello
[10/19/23]seed@VM:~/.../kernel_module$ 
```



```
seed@VM:PES2UG21CS283:Maryam:~/.../Labsetup
$>sudo dmesg -C
seed@VM:PES2UG21CS283:Maryam:~/.../Labsetup
$>sudo dmesg -k -w
[ 7713.667916] hello: module verification failed: signature and/or required key missing - tainting kernel
[ 7713.669306] Hello World!
[ 8233.436698] Bye-bye World!.
```

Explaination:

The code is an example of a Loadable Kernel Module (LKM) for the Linux kernel. It prints "Hello World!" when loaded and "Bye-bye World!" when removed, logging these messages in `/var/log/syslog`. We can view these messages using `sudo dmesg -k -w`. In a separate terminal, compile the module with `make`, insert it with `sudo insmod`, list loaded modules with `lsmod | grep hello`, and remove it with `sudo rmmod hello`. This example demonstrates the process of creating and managing kernel modules to extend Linux kernel functionality without rebooting.

Task 1.B: Implement a Simple Firewall Using Netfilter

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

Netfilter is designed to facilitate the manipulation of packets by authorized users. It achieves this goal by implementing a number of hooks in the Linux kernel. These hooks are inserted into various places, including the packet's incoming and outgoing paths. If we want to manipulate the incoming packets, we simply need to connect our own programs (within LKM) to the corresponding hooks.

Once an incoming packet arrives, our program will be invoked. Our program can decide whether this packet should be blocked or not; moreover, we can also modify the packets in the program.

In this task, you need to use LKM and Netfilter to implement a packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. We would like students to focus on the filtering part, the core of firewalls, so students are allowed to hardcode firewall policies in the program.

Tasks

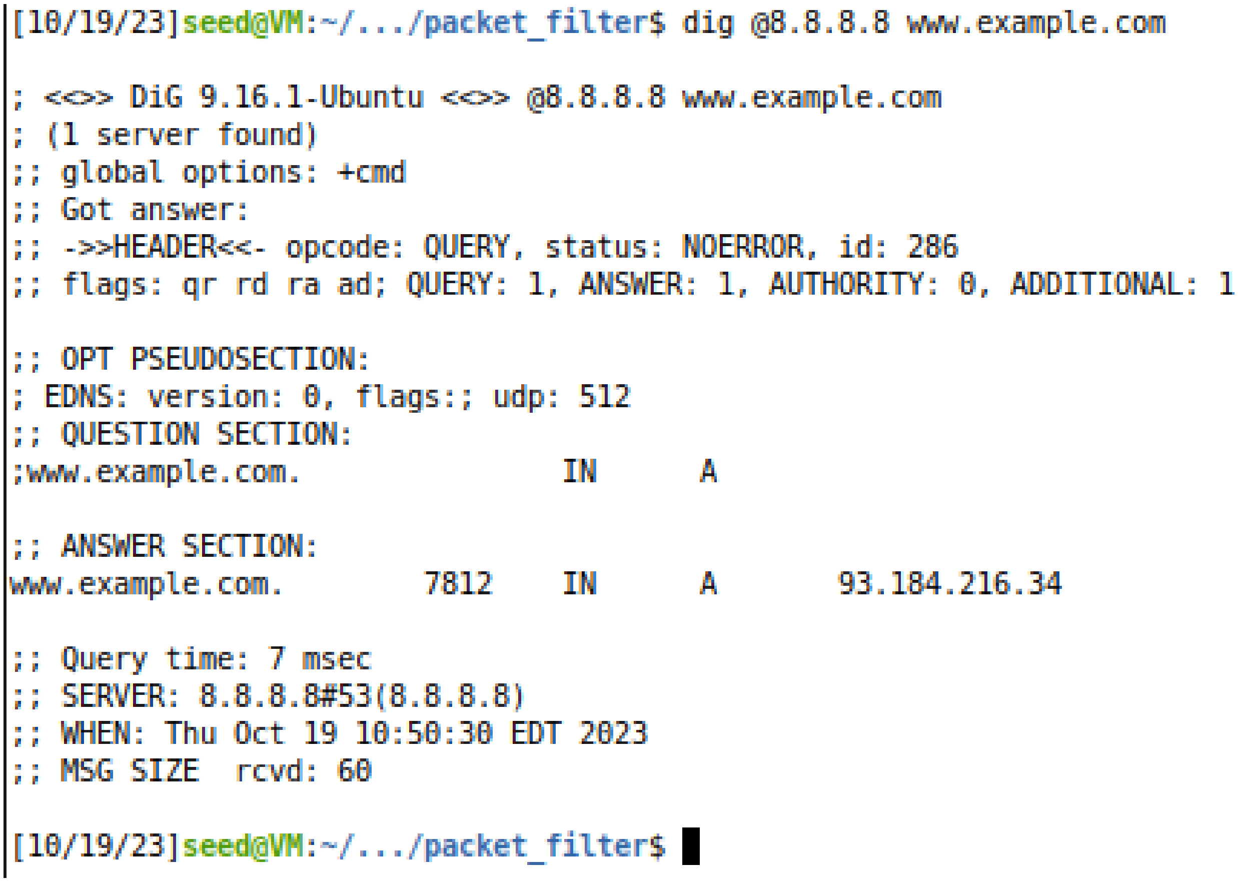
1. Compile the code using the provided Makefile. Load it into the kernel, and demonstrate that the firewall is working as expected. You can use the following command to generate UDP packets to 8.8.8.8, which is Google's DNS server. If your firewall works, your request will be blocked; otherwise, you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

On the VM Terminal type the following command to make sure www.example.com is reachable, if it is not consider changing 8.8.8.8 to 8.8.4.4

Command:

```
$ dig @8.8.8.8 www.example.com
```

Note your observations with screenshots



```
[10/19/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 286
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        7812    IN      A       93.184.216.34

;; Query time: 7 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Oct 19 10:50:30 EDT 2023
;; MSG SIZE  rcvd: 60

[10/19/23]seed@VM:~/.../packet_filter$ 
```

The IP address of www.example.com is 93.184.216.34

Open a new terminal to view kernel messages and execute the following -

Command:

$ sudo dmesg -k -w

Now on the earlier Terminal Window execute the following to insert the kernel object.

Note - Please change the make file, 'uncomment' the respective names (seedFilter, seedPrint, seedBlock) based on the task

Uncomment - 'obj-m += seedFilter.o' and comment the other two.

Command:

$ make

$ sudo insmod seedFilter.ko

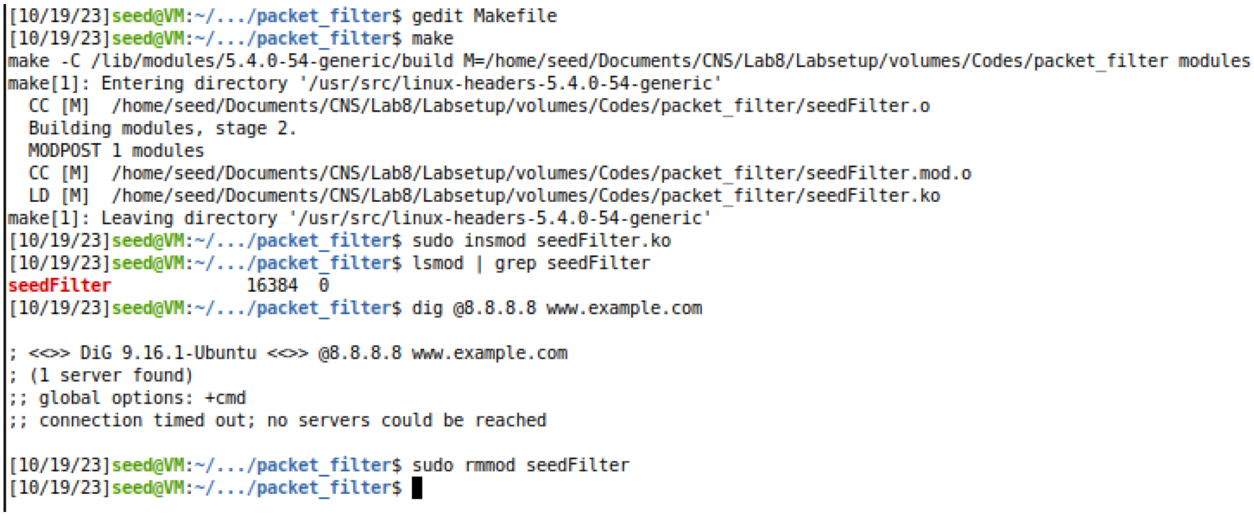
$ lsmod | grep seedFilter

After inserting execute the below and notice the difference -

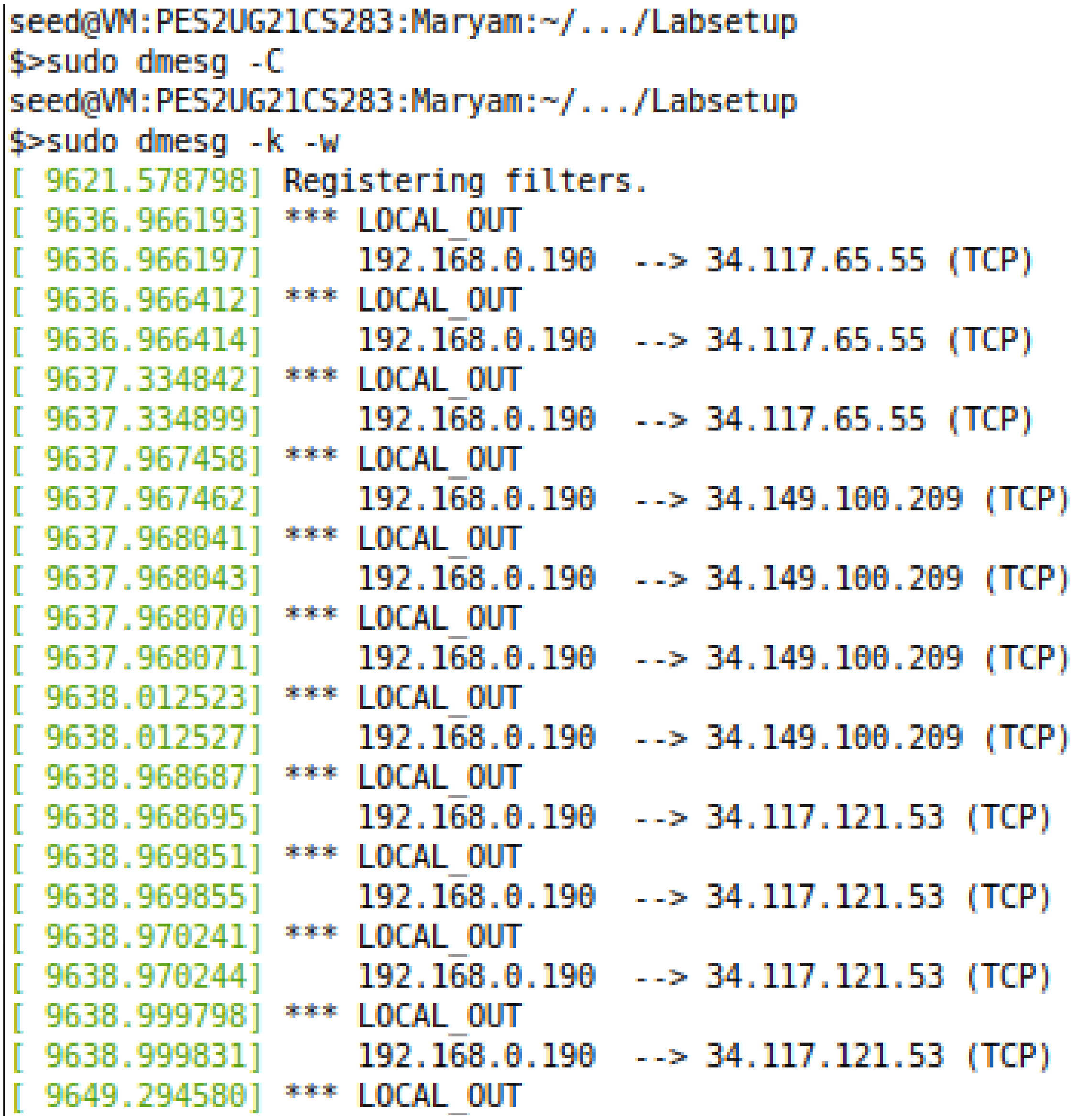
Command:

$ **dig @8.8.8.8 www.example.com**

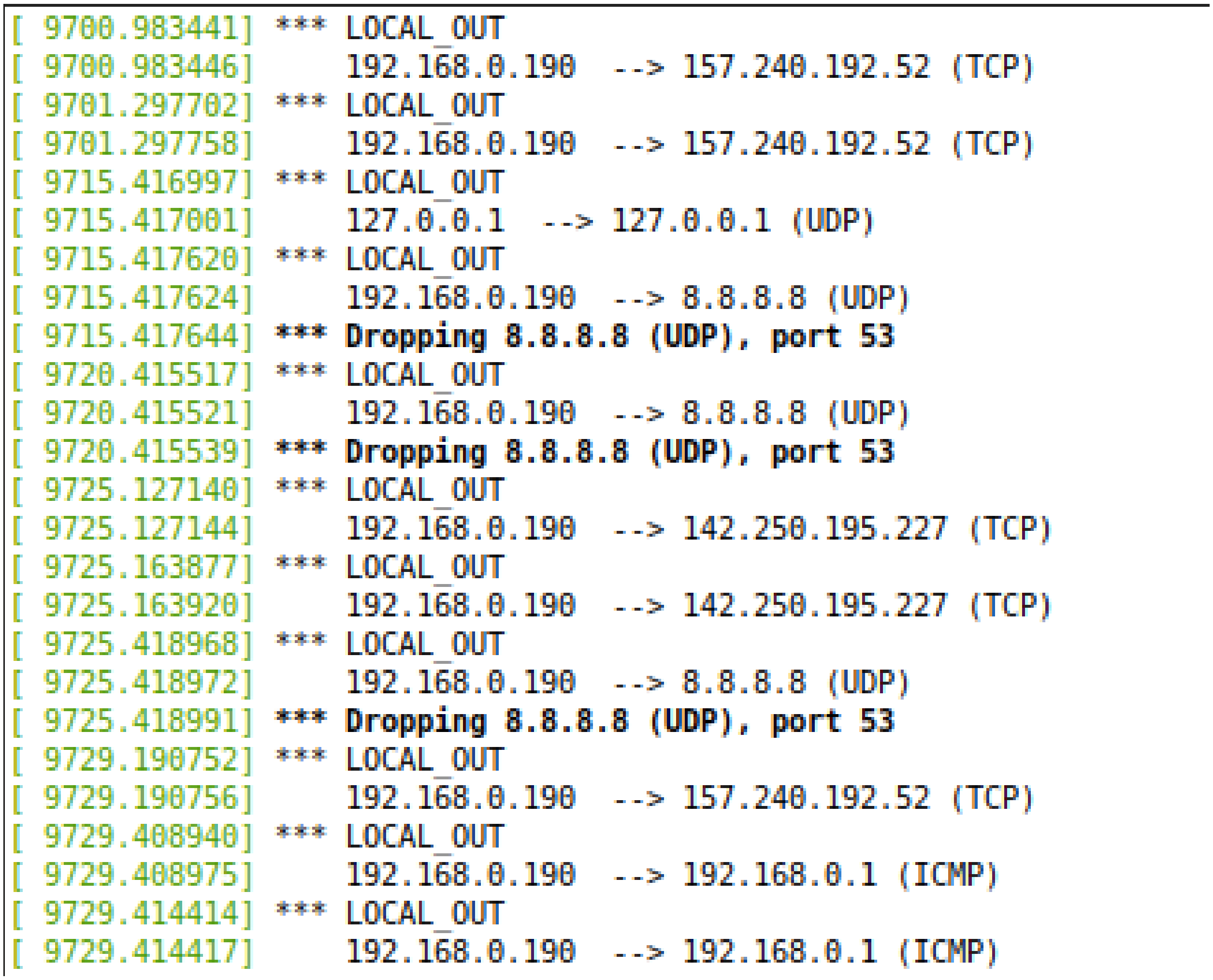
Take screenshots and explain your observations on both Terminal Windows.



```
[10/19/23]seed@VM:~/.../packet_filter$ gedit Makefile
[10/19/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[10/19/23]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[10/19/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
[10/19/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached

[10/19/23]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
[10/19/23]seed@VM:~/.../packet_filter$ 
```



```
seed@VM:PES2UG21CS283:Maryam:~/.../Labsetup
$>sudo dmesg -C
seed@VM:PES2UG21CS283:Maryam:~/.../Labsetup
$>sudo dmesg -k -w
[ 9621.578798] Registering filters.
[ 9636.966193] *** LOCAL_OUT
[ 9636.966197]     192.168.0.190  --> 34.117.65.55 (TCP)
[ 9636.966412] *** LOCAL_OUT
[ 9636.966414]     192.168.0.190  --> 34.117.65.55 (TCP)
[ 9637.334842] *** LOCAL_OUT
[ 9637.334899]     192.168.0.190  --> 34.117.65.55 (TCP)
[ 9637.967458] *** LOCAL_OUT
[ 9637.967462]     192.168.0.190  --> 34.149.100.209 (TCP)
[ 9637.968041] *** LOCAL_OUT
[ 9637.968043]     192.168.0.190  --> 34.149.100.209 (TCP)
[ 9637.968070] *** LOCAL_OUT
[ 9637.968071]     192.168.0.190  --> 34.149.100.209 (TCP)
[ 9638.012523] *** LOCAL_OUT
[ 9638.012527]     192.168.0.190  --> 34.149.100.209 (TCP)
[ 9638.968687] *** LOCAL_OUT
[ 9638.968695]     192.168.0.190  --> 34.117.121.53 (TCP)
[ 9638.969851] *** LOCAL_OUT
[ 9638.969855]     192.168.0.190  --> 34.117.121.53 (TCP)
[ 9638.970241] *** LOCAL_OUT
[ 9638.970244]     192.168.0.190  --> 34.117.121.53 (TCP)
[ 9638.999798] *** LOCAL_OUT
[ 9638.999831]     192.168.0.190  --> 34.117.121.53 (TCP)
[ 9649.294580] *** LOCAL_OUT
```



```
[ 9700.983441] *** LOCAL_OUT
[ 9700.983446]      192.168.0.190  --> 157.240.192.52 (TCP)
[ 9701.297702] *** LOCAL_OUT
[ 9701.297758]      192.168.0.190  --> 157.240.192.52 (TCP)
[ 9715.416997] *** LOCAL_OUT
[ 9715.417001]      127.0.0.1  --> 127.0.0.1 (UDP)
[ 9715.417620] *** LOCAL_OUT
[ 9715.417624]      192.168.0.190  --> 8.8.8.8 (UDP)
[ 9715.417644] *** Dropping 8.8.8.8 (UDP), port 53
[ 9720.415517] *** LOCAL_OUT
[ 9720.415521]      192.168.0.190  --> 8.8.8.8 (UDP)
[ 9720.415539] *** Dropping 8.8.8.8 (UDP), port 53
[ 9725.127140] *** LOCAL_OUT
[ 9725.127144]      192.168.0.190  --> 142.250.195.227 (TCP)
[ 9725.163877] *** LOCAL_OUT
[ 9725.163920]      192.168.0.190  --> 142.250.195.227 (TCP)
[ 9725.418968] *** LOCAL_OUT
[ 9725.418972]      192.168.0.190  --> 8.8.8.8 (UDP)
[ 9725.418991] *** Dropping 8.8.8.8 (UDP), port 53
[ 9729.190752] *** LOCAL_OUT
[ 9729.190756]      192.168.0.190  --> 157.240.192.52 (TCP)
[ 9729.408940] *** LOCAL_OUT
[ 9729.408975]      192.168.0.190  --> 192.168.0.1 (ICMP)
[ 9729.414414] *** LOCAL_OUT
[ 9729.414417]      192.168.0.190  --> 192.168.0.1 (ICMP)
```

Observation

We see that the seedFilter module is implemented.

We also see the UDP packet being dropped in the logs

Remove the module by executing **-**

$ sudo rmmod seedFilter

To clear the kernel messages execute -

$ dmesg -C

2. Hook the printInfo function to all of the netfilter hooks. Here are the macros of the hook numbers. Using your experiment results to help explain at what condition each of the hook functions be invoked.

   NF_INET_PRE_ROUTING

   NF_INET_LOCAL_IN

   NF_INET_FORWARD

   NF_INET_LOCAL_OUT

   NF_INET_POST_ROUTING

Similar to the previous task, open two terminal windows on your VM Host - one for viewing the kernel messages and the other for inserting the module.

On one window execute the following to view kernel messages -

Command:

```
$ sudo dmesg -k -w
```

Change the makefile as previously mentioned -

Uncomment - 'obj-m += seedPrint.o' and comment the other two.

Now on the other terminal window, insert the kernel module -

Command:

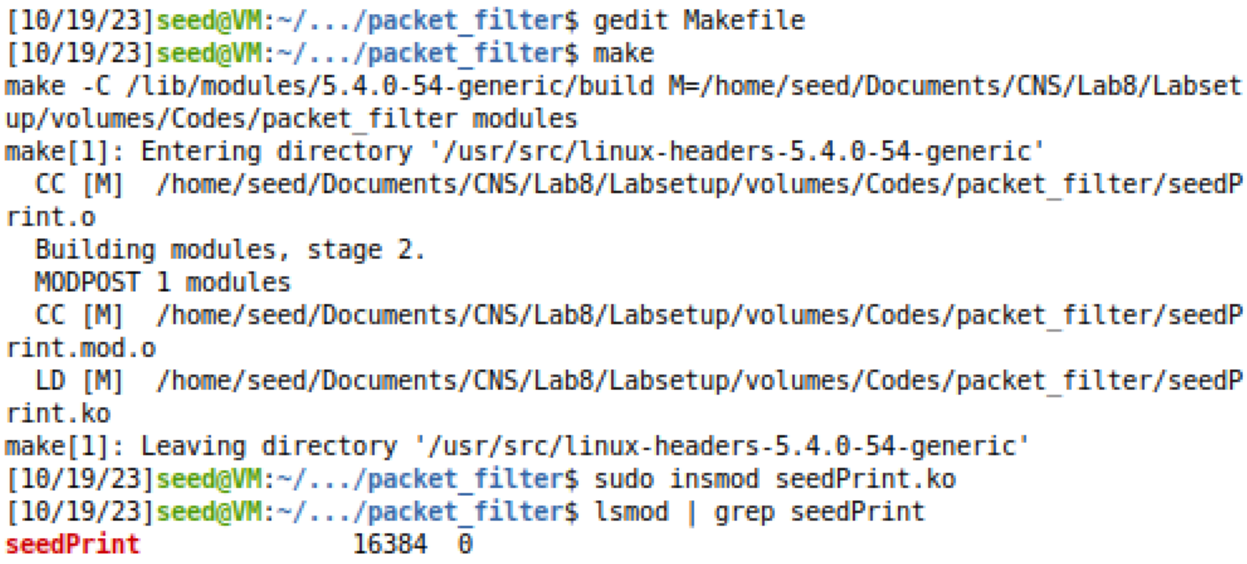
```
$ make
$ sudo insmod seedPrint.ko
$ lsmod | grep seedPrint
```

After inserting execute the below and notice the difference -

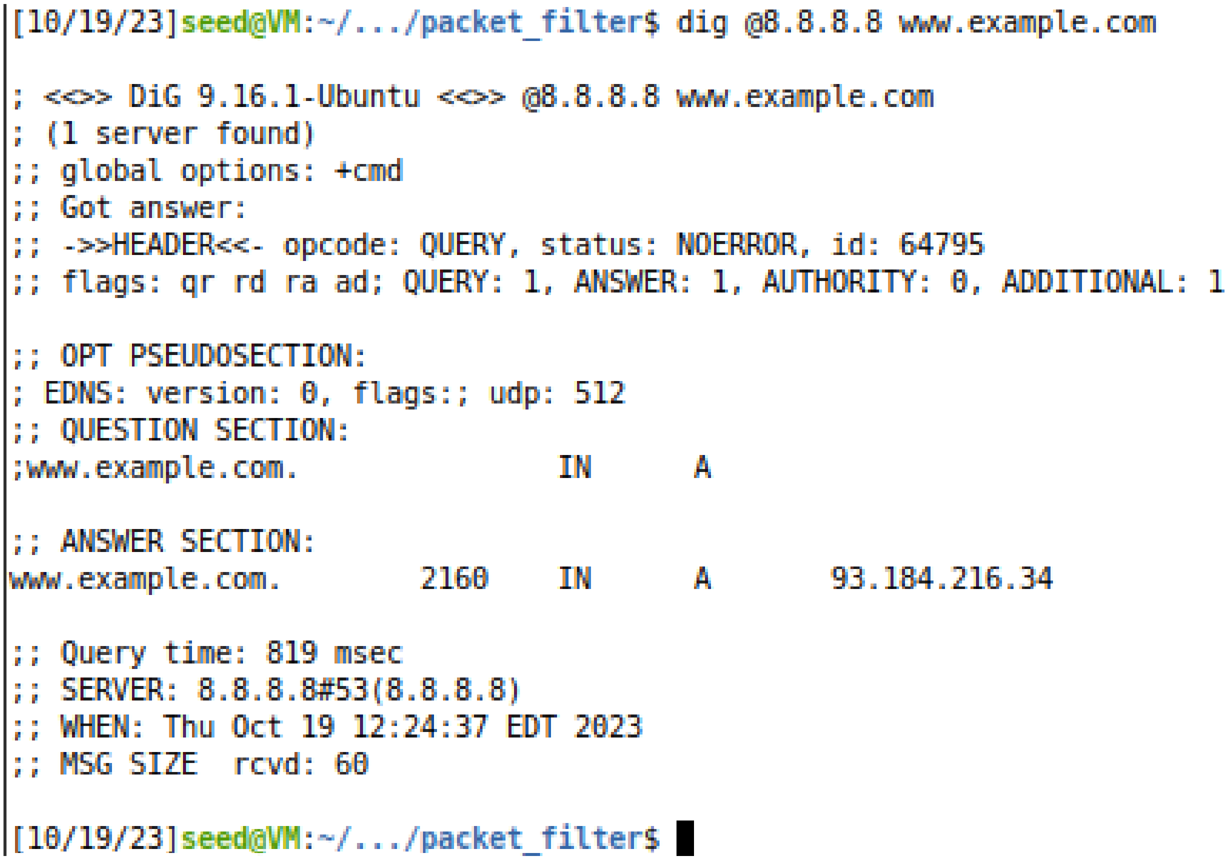
Command:

```
$ dig @8.8.8.8 www.example.com
```

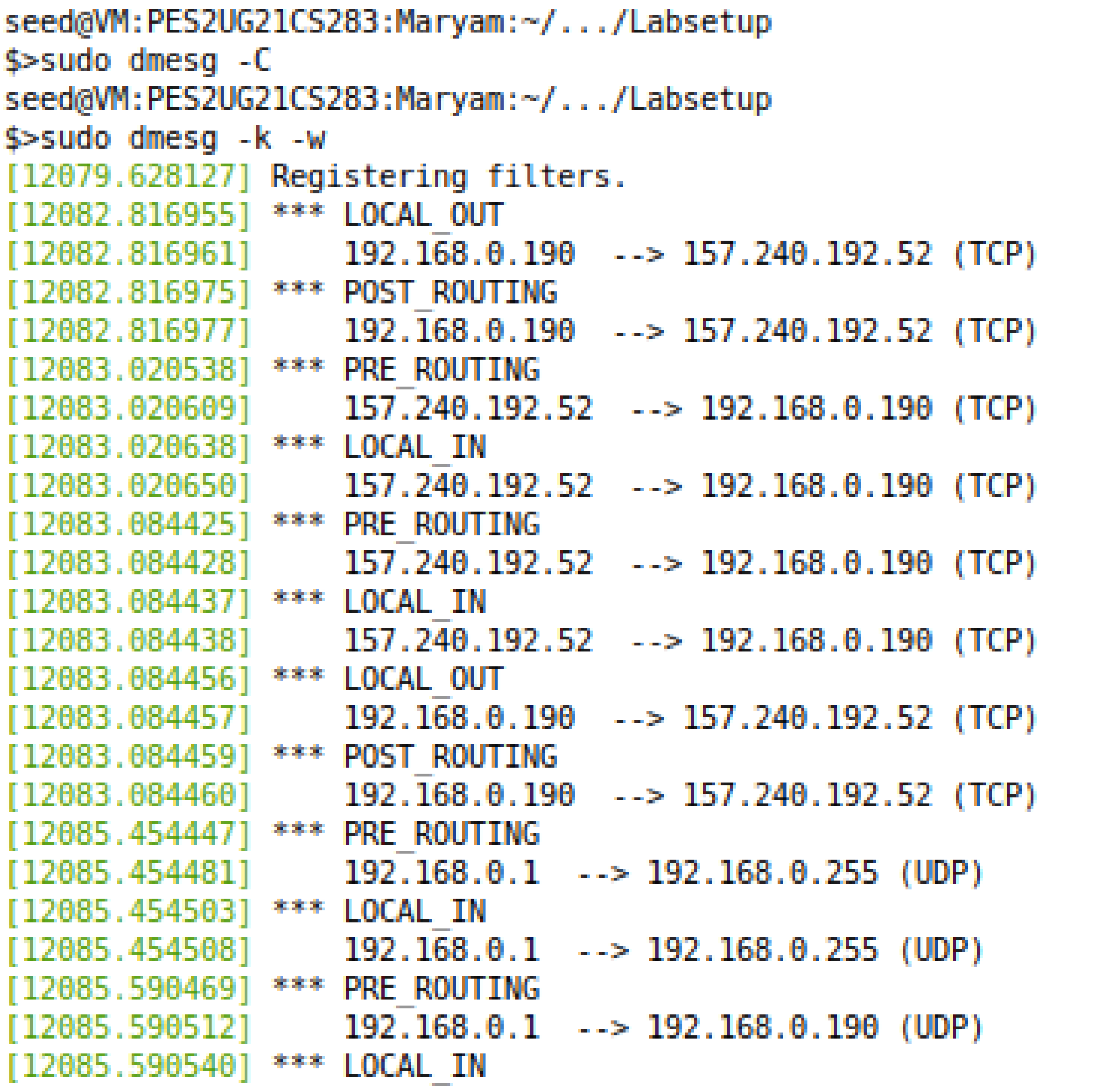
Take screenshots and explain your observations on both Terminal Windows.



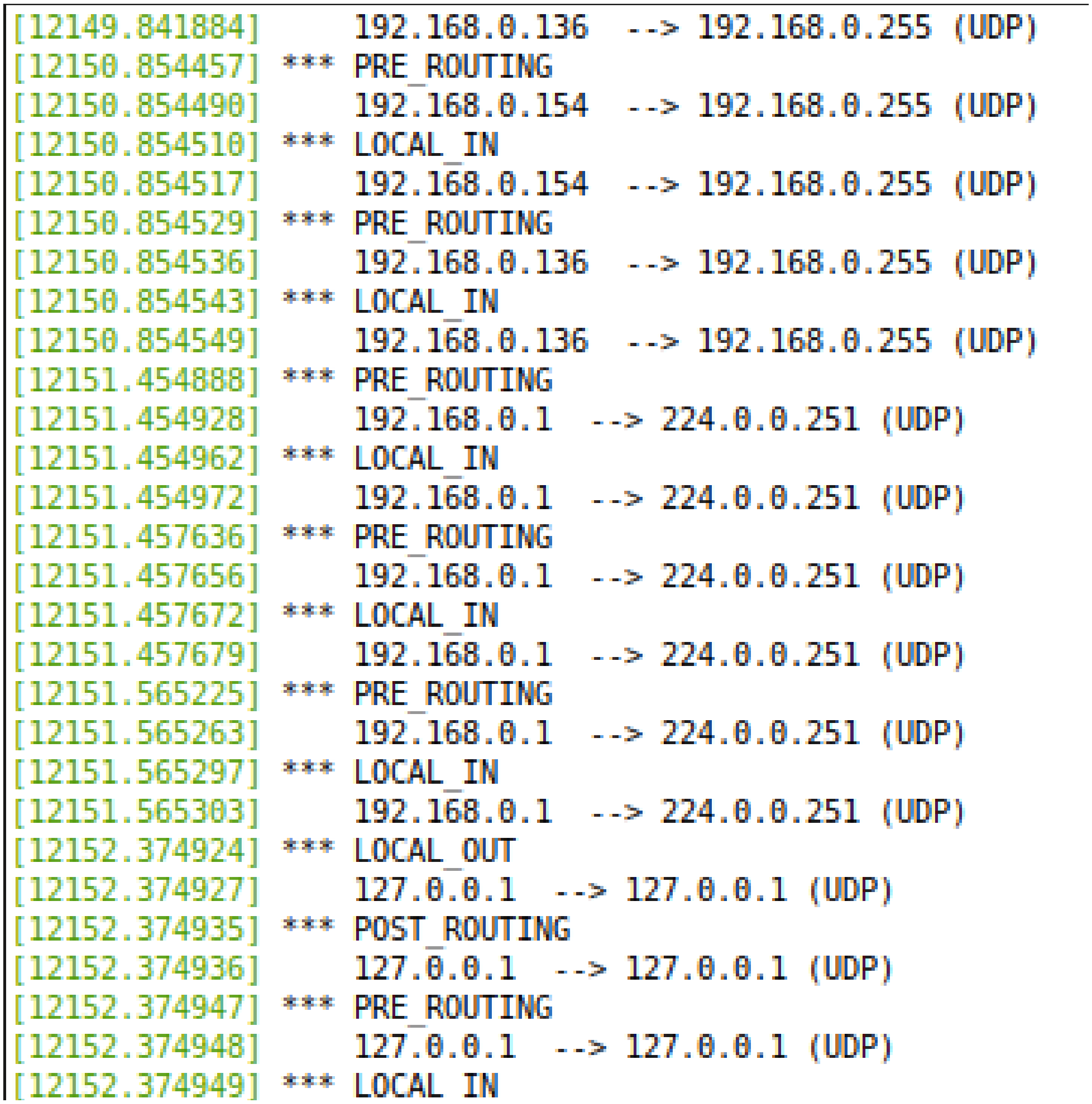
```
[10/19/23]seed@VM:~/.../packet_filter$ gedit Makefile
[10/19/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Documents/CNS/Lab8/Labset
up/volumes/Codes/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedP
rint.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedP
rint.mod.o
  LD [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedP
rint.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[10/19/23]seed@VM:~/.../packet_filter$ sudo insmod seedPrint.ko
[10/19/23]seed@VM:~/.../packet_filter$ lsmod | grep seedPrint
seedPrint              16384  0
```



```
[10/19/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64795
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        2160    IN      A       93.184.216.34

;; Query time: 819 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Oct 19 12:24:37 EDT 2023
;; MSG SIZE  rcvd: 60

[10/19/23]seed@VM:~/.../packet_filter$ 
```



```
seed@VM:PES2UG21CS283:Maryam:~/.../Labsetup
$>sudo dmesg -C
seed@VM:PES2UG21CS283:Maryam:~/.../Labsetup
$>sudo dmesg -k -w
[12079.628127] Registering filters.
[12082.816955] *** LOCAL_OUT
[12082.816961]     192.168.0.190  --> 157.240.192.52 (TCP)
[12082.816975] *** POST_ROUTING
[12082.816977]     192.168.0.190  --> 157.240.192.52 (TCP)
[12083.020538] *** PRE_ROUTING
[12083.020609]     157.240.192.52  --> 192.168.0.190 (TCP)
[12083.020638] *** LOCAL_IN
[12083.020650]     157.240.192.52  --> 192.168.0.190 (TCP)
[12083.084425] *** PRE_ROUTING
[12083.084428]     157.240.192.52  --> 192.168.0.190 (TCP)
[12083.084437] *** LOCAL_IN
[12083.084438]     157.240.192.52  --> 192.168.0.190 (TCP)
[12083.084456] *** LOCAL_OUT
[12083.084457]     192.168.0.190  --> 157.240.192.52 (TCP)
[12083.084459] *** POST_ROUTING
[12083.084460]     192.168.0.190  --> 157.240.192.52 (TCP)
[12085.454447] *** PRE_ROUTING
[12085.454481]     192.168.0.1  --> 192.168.0.255 (UDP)
[12085.454503] *** LOCAL_IN
[12085.454508]     192.168.0.1  --> 192.168.0.255 (UDP)
[12085.590469] *** PRE_ROUTING
[12085.590512]     192.168.0.1  --> 192.168.0.190 (UDP)
[12085.590540] *** LOCAL_IN
```



```
[12149.841884]     192.168.0.136  --> 192.168.0.255 (UDP)
[12150.854457] *** PRE_ROUTING
[12150.854490]     192.168.0.154  --> 192.168.0.255 (UDP)
[12150.854510] *** LOCAL_IN
[12150.854517]     192.168.0.154  --> 192.168.0.255 (UDP)
[12150.854529] *** PRE_ROUTING
[12150.854536]     192.168.0.136  --> 192.168.0.255 (UDP)
[12150.854543] *** LOCAL_IN
[12150.854549]     192.168.0.136  --> 192.168.0.255 (UDP)
[12151.454888] *** PRE_ROUTING
[12151.454928]     192.168.0.1  --> 224.0.0.251 (UDP)
[12151.454962] *** LOCAL_IN
[12151.454972]     192.168.0.1  --> 224.0.0.251 (UDP)
[12151.457636] *** PRE_ROUTING
[12151.457656]     192.168.0.1  --> 224.0.0.251 (UDP)
[12151.457672] *** LOCAL_IN
[12151.457679]     192.168.0.1  --> 224.0.0.251 (UDP)
[12151.565225] *** PRE_ROUTING
[12151.565263]     192.168.0.1  --> 224.0.0.251 (UDP)
[12151.565297] *** LOCAL_IN
[12151.565303]     192.168.0.1  --> 224.0.0.251 (UDP)
[12152.374924] *** LOCAL_OUT
[12152.374927]     127.0.0.1  --> 127.0.0.1 (UDP)
[12152.374935] *** POST_ROUTING
[12152.374936]     127.0.0.1  --> 127.0.0.1 (UDP)
[12152.374947] *** PRE_ROUTING
[12152.374948]     127.0.0.1  --> 127.0.0.1 (UDP)
[12152.374949] *** LOCAL_IN
```

This is a Print filter. We are able to get the output.

Remove the module by executing -

```
$ sudo rmmod seedPrint
```

3. Implement two more hooks to achieve the following:

   (1) preventing other computers to ping the VM, and

   (2) preventing other computers from telnetting into the VM.

   Please implement two different hook functions, but register them to the same netfilter hook. You should decide what hook to use. Telnet's default port is TCP port 23. To test it, you can start the containers, go to 10.9.0.5, run the following commands (10.9.0.1 is the IP address assigned to the VM; for the sake of simplicity, you can hardcode this IP address in your firewall rules):

For this Task you need the docker container - Open the Host 10.9.0.5

Similar to the previous task,this time we need to open three terminal windows on your VM Host - one for viewing the kernel messages, one for inserting the module and the docker container that acts as an external machine.

On one window execute the following to view kernel messages -

Command:

```
$ sudo dmesg -k -w
```

Change the makefile as previously mentioned -

Uncomment - 'obj-m += seedBlock.o' and comment the other two.

Now on the other terminal window, insert the kernel module -

Command:

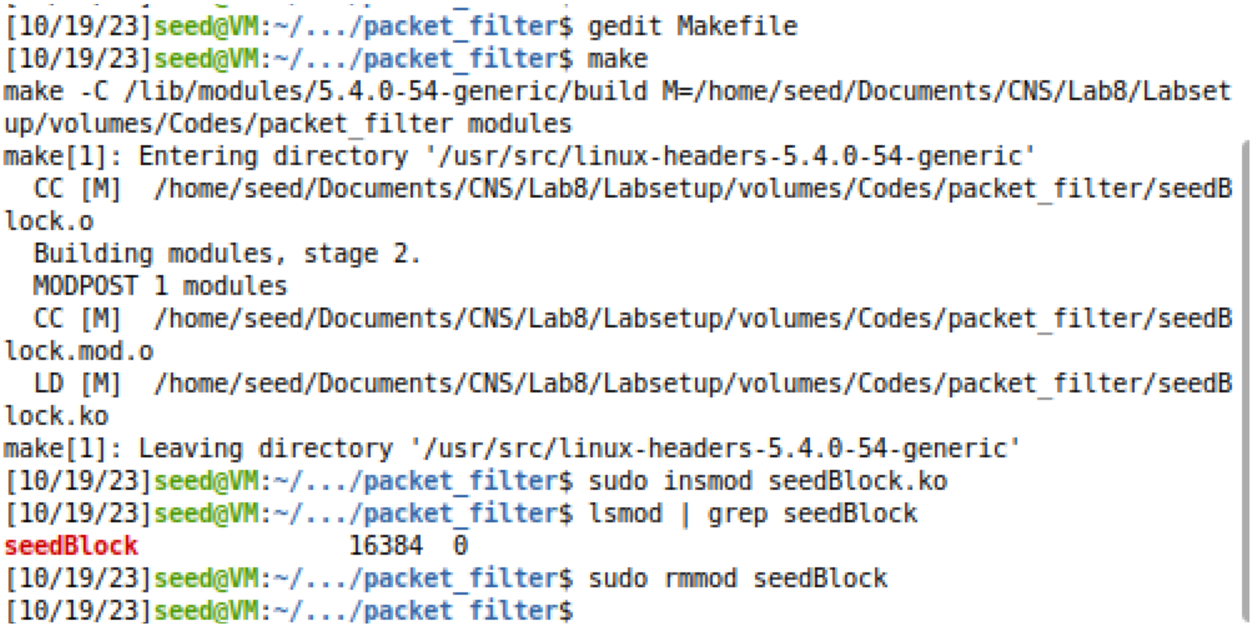
```
$ make
$ sudo insmod seedBlock.ko
$ lsmod | grep seedBlock
```

On the Host A - 10.9.0.5 terminal try -

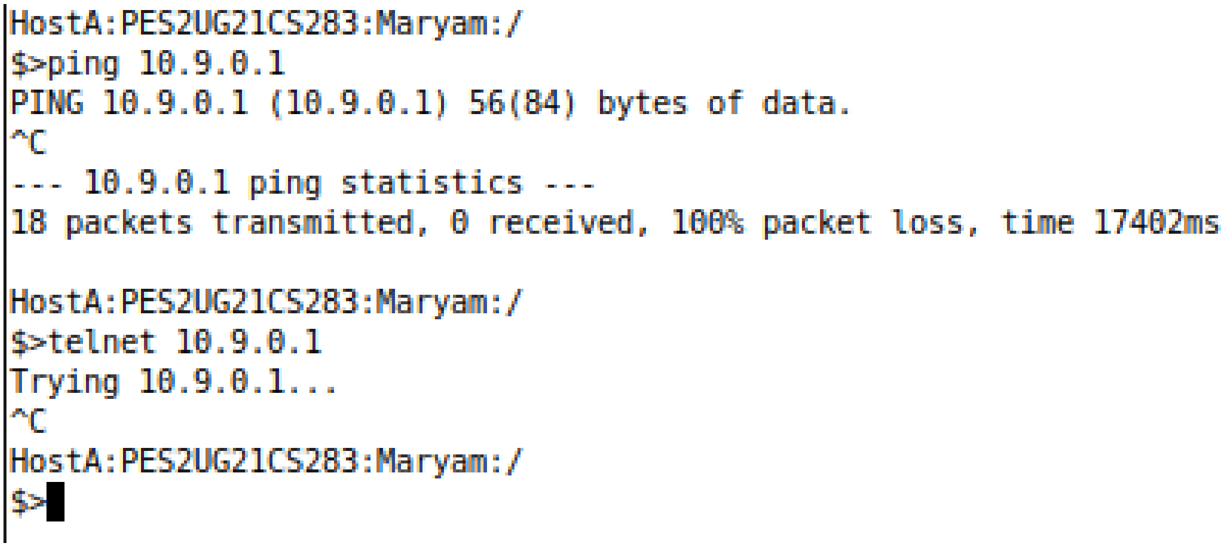
Command:

```
# ping 10.9.0.1
# telnet 10.9.0.1
```

Take screenshots and explain your observations on allTerminal Windows.

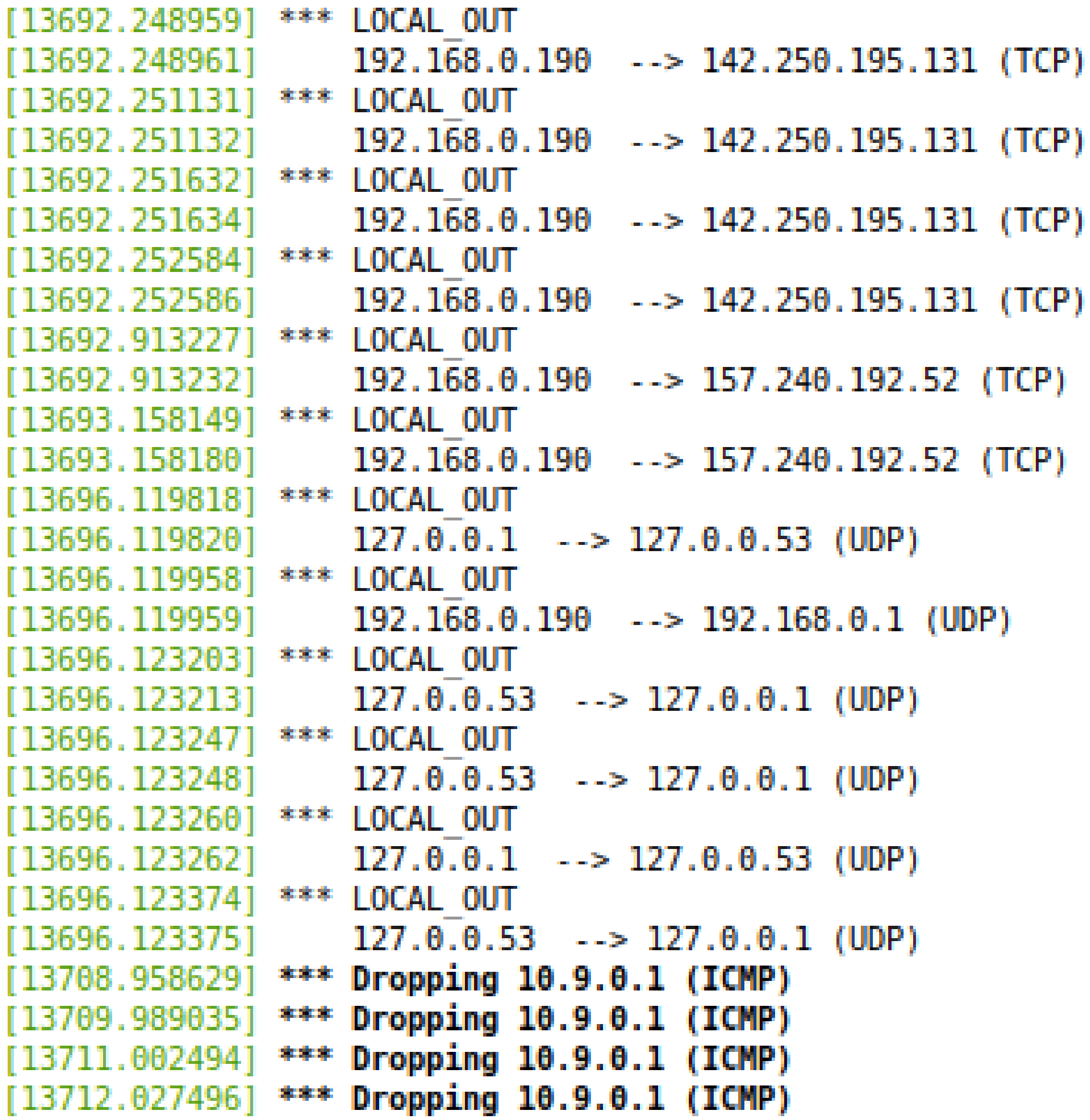


```
[10/19/23]seed@VM:~/.../packet_filter$ gedit Makefile
[10/19/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Documents/CNS/Lab8/Labset
up/volumes/Codes/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedB
lock.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedB
lock.mod.o
  LD [M]  /home/seed/Documents/CNS/Lab8/Labsetup/volumes/Codes/packet_filter/seedB
lock.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[10/19/23]seed@VM:~/.../packet_filter$ sudo insmod seedBlock.ko
[10/19/23]seed@VM:~/.../packet_filter$ lsmod | grep seedBlock
seedBlock              16384  0
[10/19/23]seed@VM:~/.../packet_filter$ sudo rmmod seedBlock
[10/19/23]seed@VM:~/.../packet_filter$
```



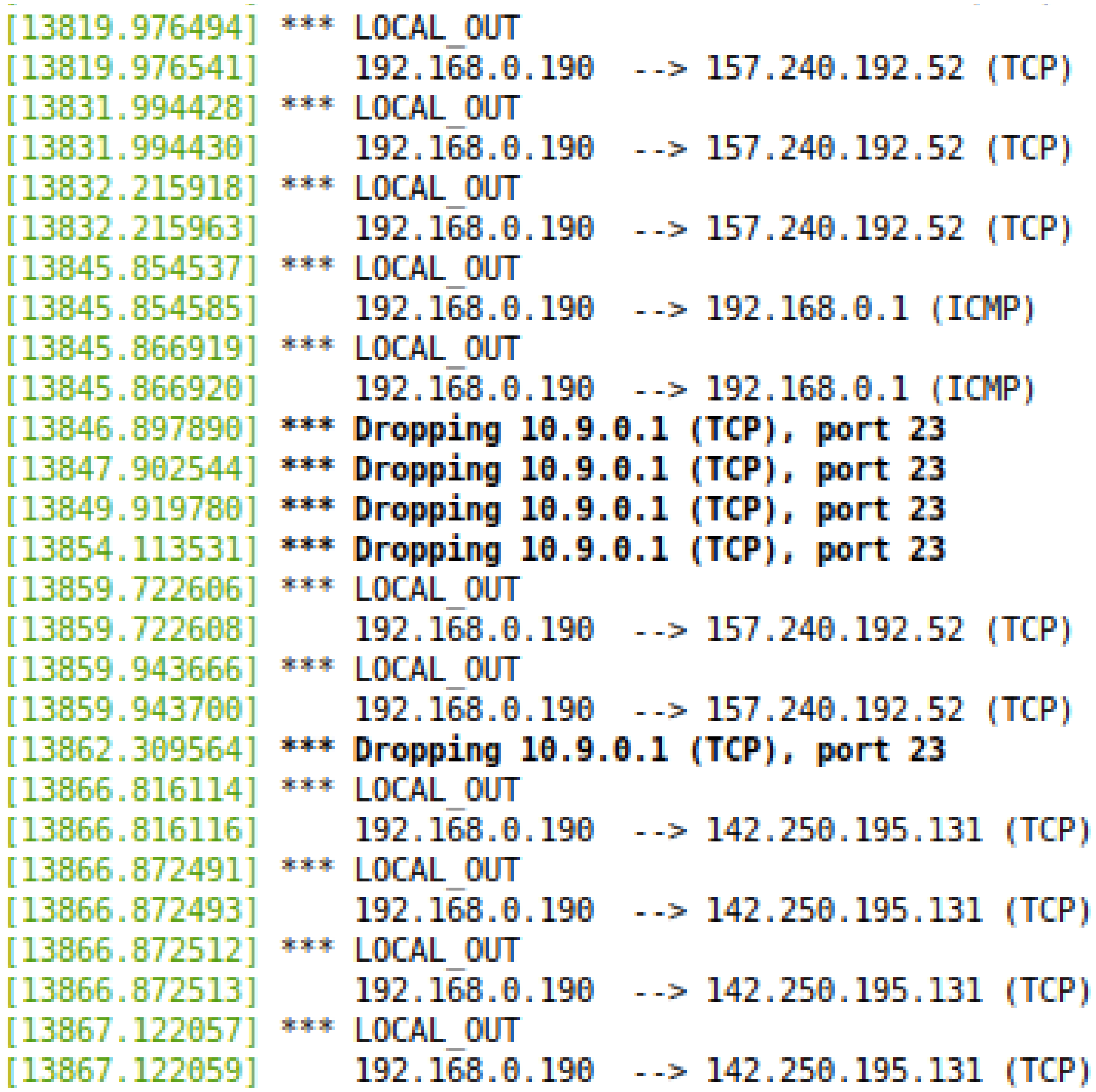
```
HostA:PES2UG21CS283:Maryam:/
$>ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
18 packets transmitted, 0 received, 100% packet loss, time 17402ms

HostA:PES2UG21CS283:Maryam:/
$>telnet 10.9.0.1
Trying 10.9.0.1...
^C
HostA:PES2UG21CS283:Maryam:/
$>
```

ping 10.9.0.1



```
[13692.248959] *** LOCAL_OUT
[13692.248961]      192.168.0.190  --> 142.250.195.131 (TCP)
[13692.251131] *** LOCAL_OUT
[13692.251132]      192.168.0.190  --> 142.250.195.131 (TCP)
[13692.251632] *** LOCAL_OUT
[13692.251634]      192.168.0.190  --> 142.250.195.131 (TCP)
[13692.252584] *** LOCAL_OUT
[13692.252586]      192.168.0.190  --> 142.250.195.131 (TCP)
[13692.913227] *** LOCAL_OUT
[13692.913232]      192.168.0.190  --> 157.240.192.52 (TCP)
[13693.158149] *** LOCAL_OUT
[13693.158180]      192.168.0.190  --> 157.240.192.52 (TCP)
[13696.119818] *** LOCAL_OUT
[13696.119820]      127.0.0.1  --> 127.0.0.53 (UDP)
[13696.119958] *** LOCAL_OUT
[13696.119959]      192.168.0.190  --> 192.168.0.1 (UDP)
[13696.123203] *** LOCAL_OUT
[13696.123213]      127.0.0.53  --> 127.0.0.1 (UDP)
[13696.123247] *** LOCAL_OUT
[13696.123248]      127.0.0.53  --> 127.0.0.1 (UDP)
[13696.123260] *** LOCAL_OUT
[13696.123262]      127.0.0.1  --> 127.0.0.53 (UDP)
[13696.123374] *** LOCAL_OUT
[13696.123375]      127.0.0.53  --> 127.0.0.1 (UDP)
[13708.958629] *** Dropping 10.9.0.1 (ICMP)
[13709.989035] *** Dropping 10.9.0.1 (ICMP)
[13711.002494] *** Dropping 10.9.0.1 (ICMP)
[13712.027496] *** Dropping 10.9.0.1 (ICMP)
```

telnet 10.9.0.1



```
[13819.976494] *** LOCAL_OUT
[13819.976541]     192.168.0.190  --> 157.240.192.52 (TCP)
[13831.994428] *** LOCAL_OUT
[13831.994430]     192.168.0.190  --> 157.240.192.52 (TCP)
[13832.215918] *** LOCAL_OUT
[13832.215963]     192.168.0.190  --> 157.240.192.52 (TCP)
[13845.854537] *** LOCAL_OUT
[13845.854585]     192.168.0.190  --> 192.168.0.1 (ICMP)
[13845.866919] *** LOCAL_OUT
[13845.866920]     192.168.0.190  --> 192.168.0.1 (ICMP)
[13846.897890] *** Dropping 10.9.0.1 (TCP), port 23
[13847.902544] *** Dropping 10.9.0.1 (TCP), port 23
[13849.919780] *** Dropping 10.9.0.1 (TCP), port 23
[13854.113531] *** Dropping 10.9.0.1 (TCP), port 23
[13859.722606] *** LOCAL_OUT
[13859.722608]     192.168.0.190  --> 157.240.192.52 (TCP)
[13859.943666] *** LOCAL_OUT
[13859.943700]     192.168.0.190  --> 157.240.192.52 (TCP)
[13862.309564] *** Dropping 10.9.0.1 (TCP), port 23
[13866.816114] *** LOCAL_OUT
[13866.816116]     192.168.0.190  --> 142.250.195.131 (TCP)
[13866.872491] *** LOCAL_OUT
[13866.872493]     192.168.0.190  --> 142.250.195.131 (TCP)
[13866.872512] *** LOCAL_OUT
[13866.872513]     192.168.0.190  --> 142.250.195.131 (TCP)
[13867.122057] *** LOCAL_OUT
[13867.122059]     192.168.0.190  --> 142.250.195.131 (TCP)
```

When we try to ping and telnet from hostA, it is blocked.

We can see 100% filter loss

Remove the module by executing **-**

$ sudo rmmod seedBlock

To clear the kernel messages execute -

$ dmesg -C

Task 2: Experimenting with Stateless Firewall Rules

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

Background of iptables

In this task, we will use iptables to set up a firewall. The iptables firewall is designed not only to filter packets, but also to make changes to packets. To help manage these firewall rules for different purposes, iptables organizes all rules using a hierarchical structure: table, chain, and rules. 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

Task 2.A: Protecting the Router

Before proceeding ahead with this task, make sure to **open all the docker containers**.

In this task, we will set up rules to prevent outside machines from accessing the router machine, except ping.

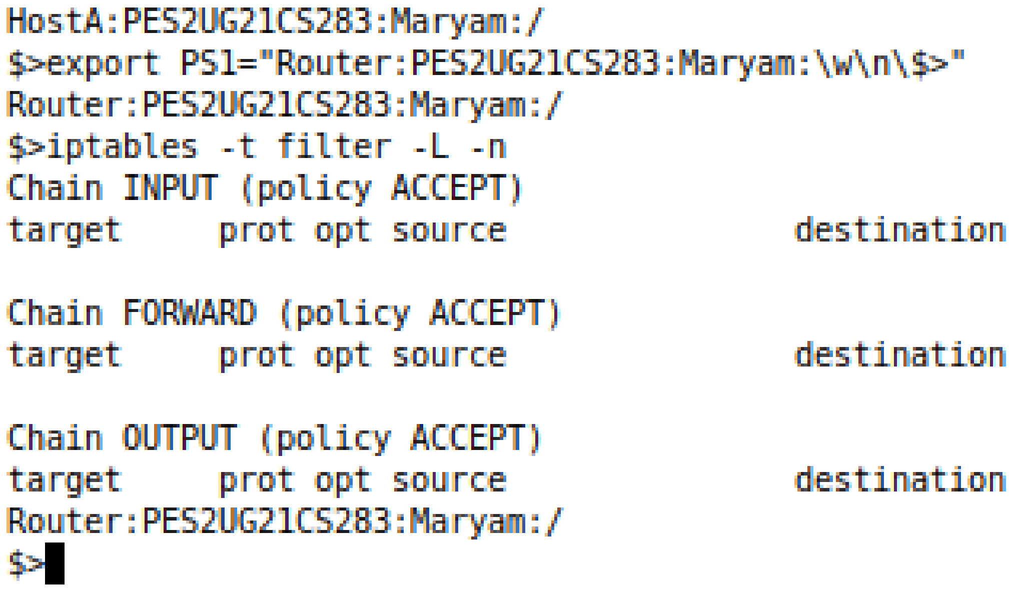
On the **Router Container (seed-router)**, execute the following Iptables commands.

In order to view the current policies run the below command

Command:

# iptables -t filter -L -n

Now please execute the following iptables command on the router container, and then try to access it from 10.9.0.5.



```
HostA:PES2UG21CS283:Maryam:/
$>export PS1="Router:PES2UG21CS283:Maryam:\w\n\$>"
Router:PES2UG21CS283:Maryam:/
$>iptables -t filter -L -n
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
Router:PES2UG21CS283:Maryam:/
$>
```

On seed-router run -

Command:

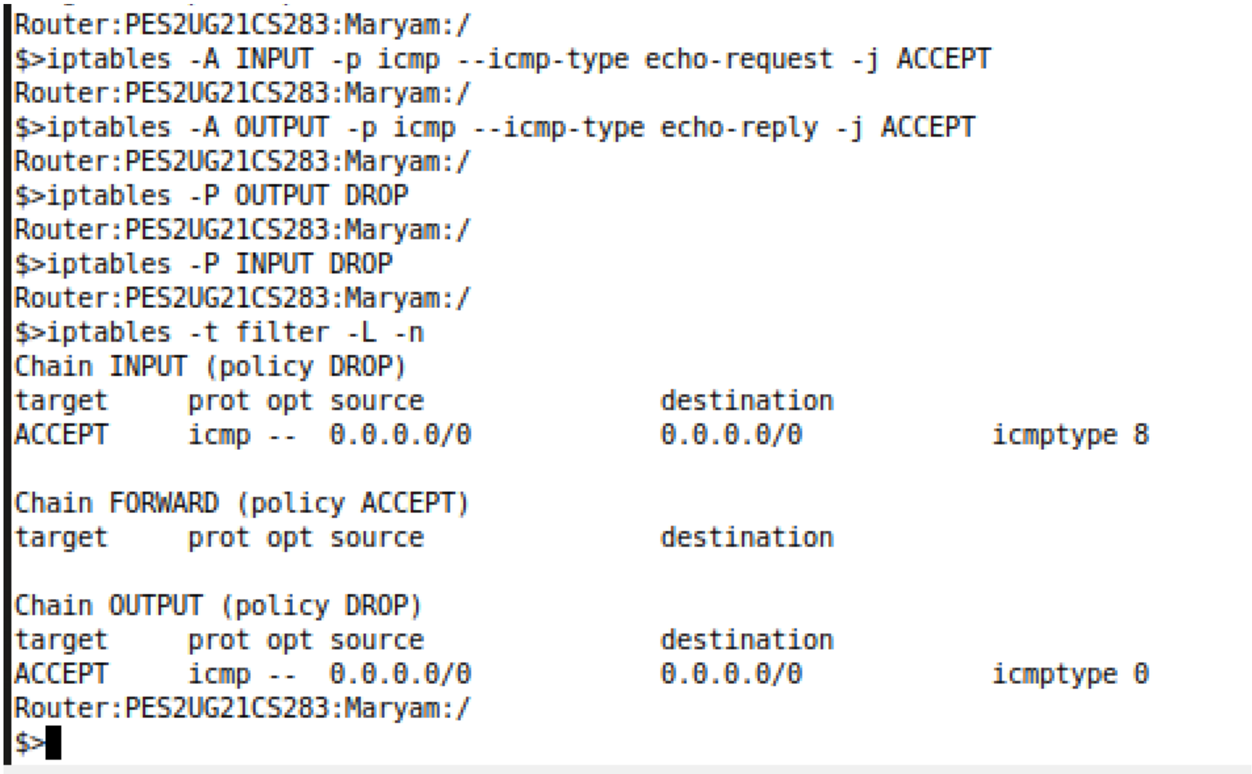
# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

# iptables -P OUTPUT DROP

# iptables -P INPUT DROP

# iptables -t filter -L -n



```
Router:PES2UG21CS283:Maryam:/
$>iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
Router:PES2UG21CS283:Maryam:/
$>iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
Router:PES2UG21CS283:Maryam:/
$>iptables -P OUTPUT DROP
Router:PES2UG21CS283:Maryam:/
$>iptables -P INPUT DROP
Router:PES2UG21CS283:Maryam:/
$>iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
Router:PES2UG21CS283:Maryam:/
$>
```

We can see that all the rules have been added to the ip routing table

Now try to access (ping and telnet) the router from Host A - 10.9.0.5

Command:

# ping seed-router

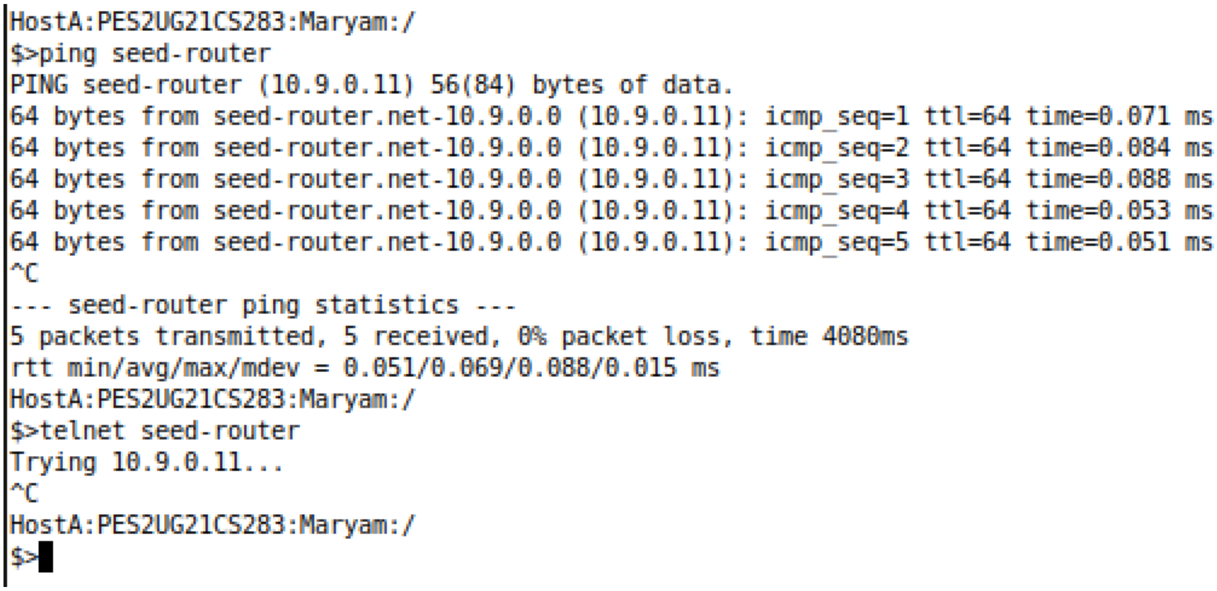
# telnet seed-router

Questions :

(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 with screenshots and explain the purpose for each rule



```
HostA:PES2UG21CS283:Maryam:/
$>ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.071 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.084 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.088 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.051 ms
^C
--- seed-router ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4080ms
rtt min/avg/max/mdev = 0.051/0.069/0.088/0.015 ms
HostA:PES2UG21CS283:Maryam:/
$>telnet seed-router
Trying 10.9.0.11...
^C
HostA:PES2UG21CS283:Maryam:/
$>
```

We observe that the ping is successful, but the telnet is not successful according to the iptables

Cleanup

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

# iptables -F

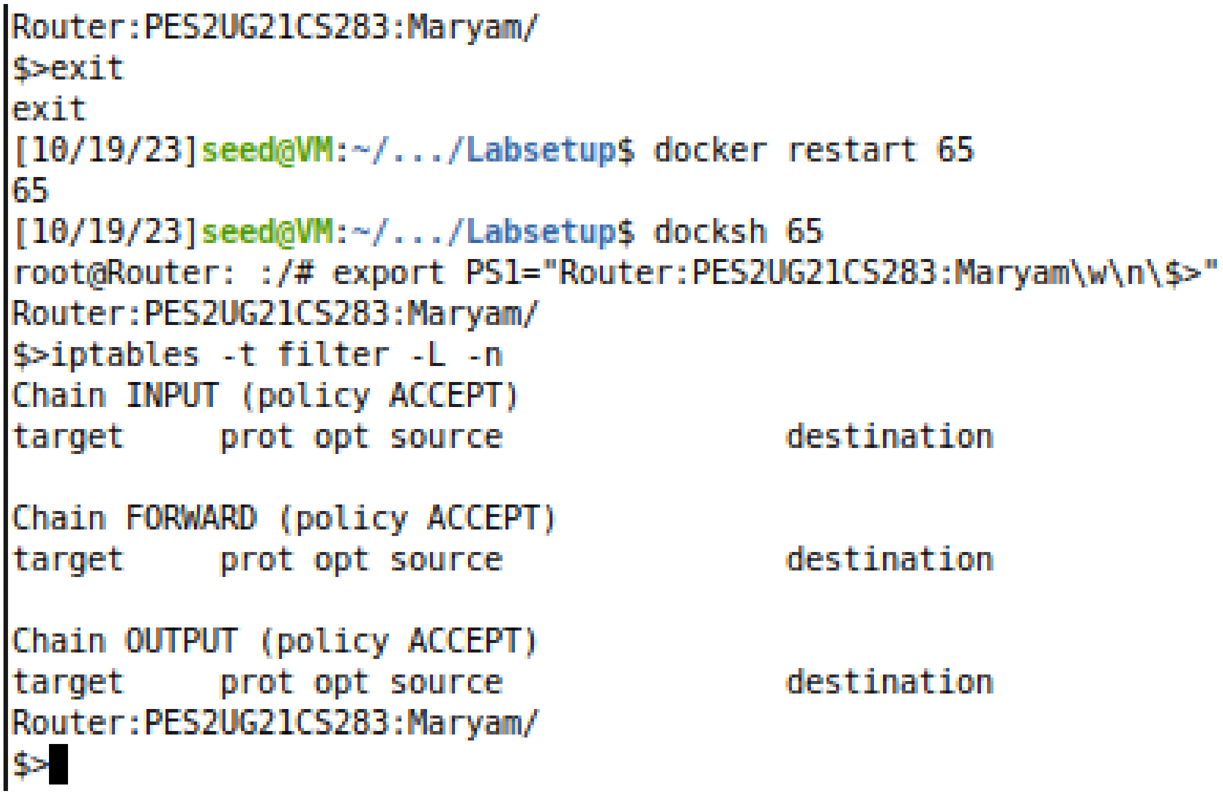
# iptables -P OUTPUT ACCEPT

# 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>



```
Router:PES2UG21CS283:Maryam/
$>exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$>iptables -t filter -L -n
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
Router:PES2UG21CS283:Maryam/
$>
```

All the rules have been cleared

Task 2.B: Protecting the Internal Network

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

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.

Execute the following iptables commands on the **seed-router container** -

Commands:

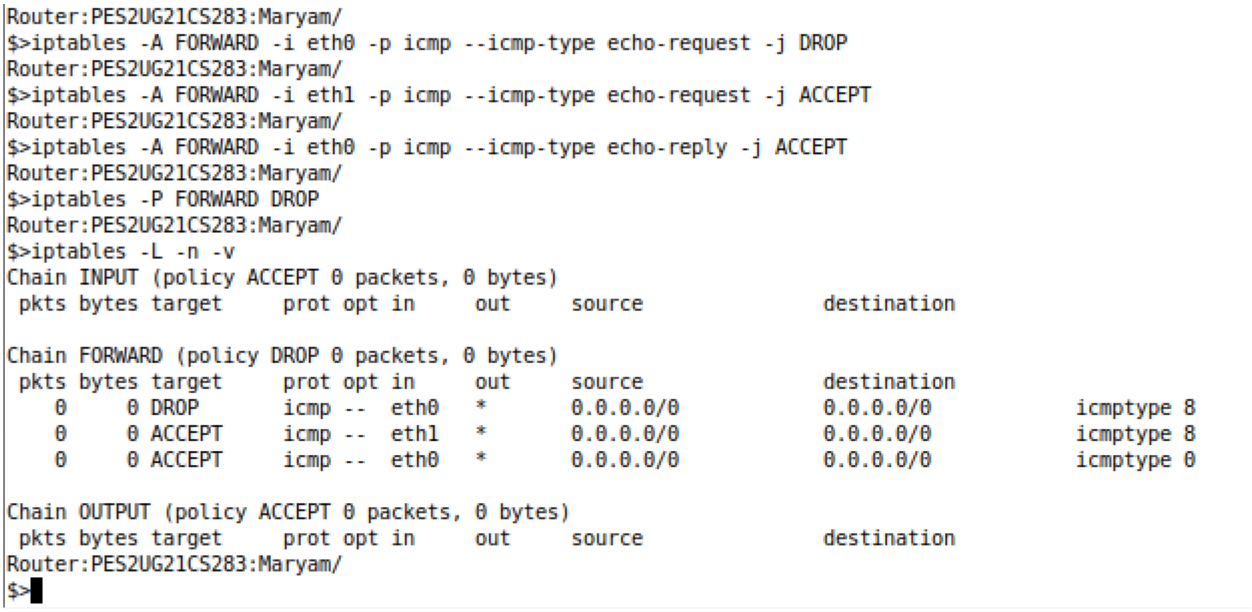
# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP

# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT

# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT

# iptables -P FORWARD DROP

# iptables -L -n -v



```
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -P FORWARD DROP
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 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 DROP       icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0            icmptype 8
    0     0 ACCEPT     icmp --  eth1   *       0.0.0.0/0            0.0.0.0/0            icmptype 8
    0     0 ACCEPT     icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
Router:PES2UG21CS283:Maryam/
$>
```

We can see that all the rules have been added to the ip routing table

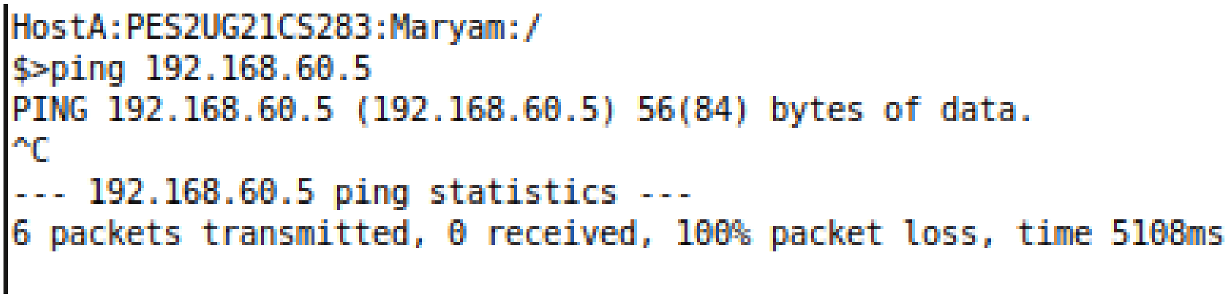
Now we shall see if these **restrictions have been enforced** in the network.

1. Outside hosts cannot ping internal hosts.

On Host A - 10.9.0.5 execute

Command:

# ping 192.168.60.5



```
HostA:PES2UG21CS283:Maryam:/
$>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5108ms
```

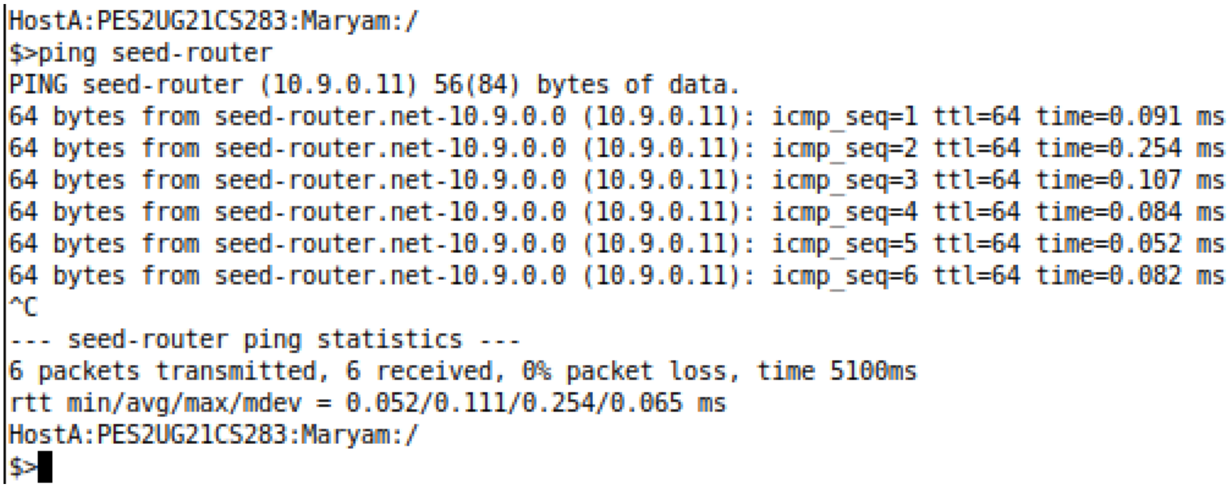
2. Outside hosts can ping the router.

The ping to IP 192.168.60.5 was unsuccessful

On Host A - 10.9.0.5 execute

Command:

# ping seed-router



```
HostA:PES2UG21CS283:Maryam:/
$>ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.091 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.254 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.107 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.084 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl=64 time=0.052 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=6 ttl=64 time=0.082 ms
^C
--- seed-router ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5100ms
rtt min/avg/max/mdev = 0.052/0.111/0.254/0.065 ms
HostA:PES2UG21CS283:Maryam:/
$>
```

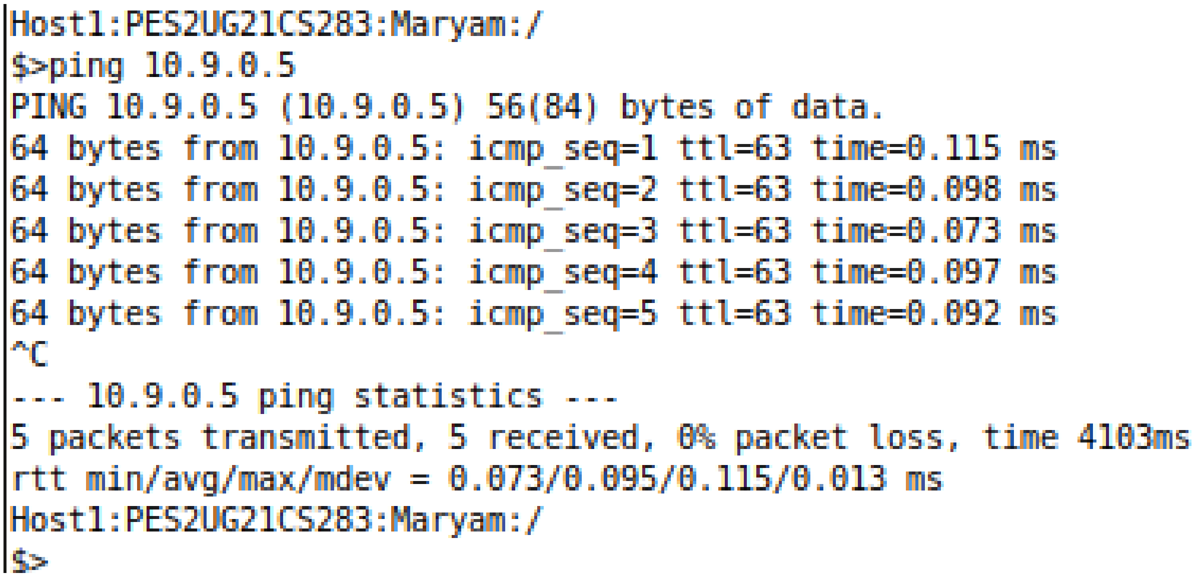
The ping to IP 192.168.60.5 was unsuccessful but it works for seed router.

3. Internal hosts can ping Outside Hosts.

On host1-192.168.60.5 execute

Command:

# ping 10.9.0.5



```
Host1:PES2UG21CS283:Maryam:/
$>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=0.115 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.098 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.073 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.097 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.092 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4103ms
rtt min/avg/max/mdev = 0.073/0.095/0.115/0.013 ms
Host1:PES2UG21CS283:Maryam:/
$>
```

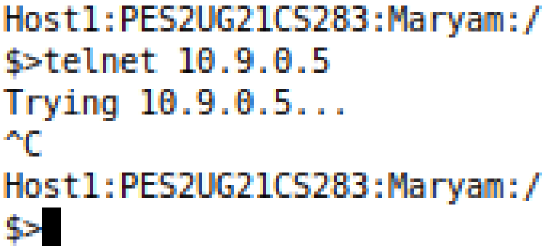
The ping to IP 192.168.60.5 was successful

4. All other packets between the internal and external networks should be blocked.

On host1-192.168.60.5 execute

Command:

# telnet 10.9.0.5



```
Host1:PES2UG21CS283:Maryam:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
^C
Host1:PES2UG21CS283:Maryam:/
$>
```

The telnet to IP 192.168.60.5 was unsuccessful

Please report your observation with screenshots and explain the purpose for each rule

Cleanup

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

On seed-router

# iptables -F

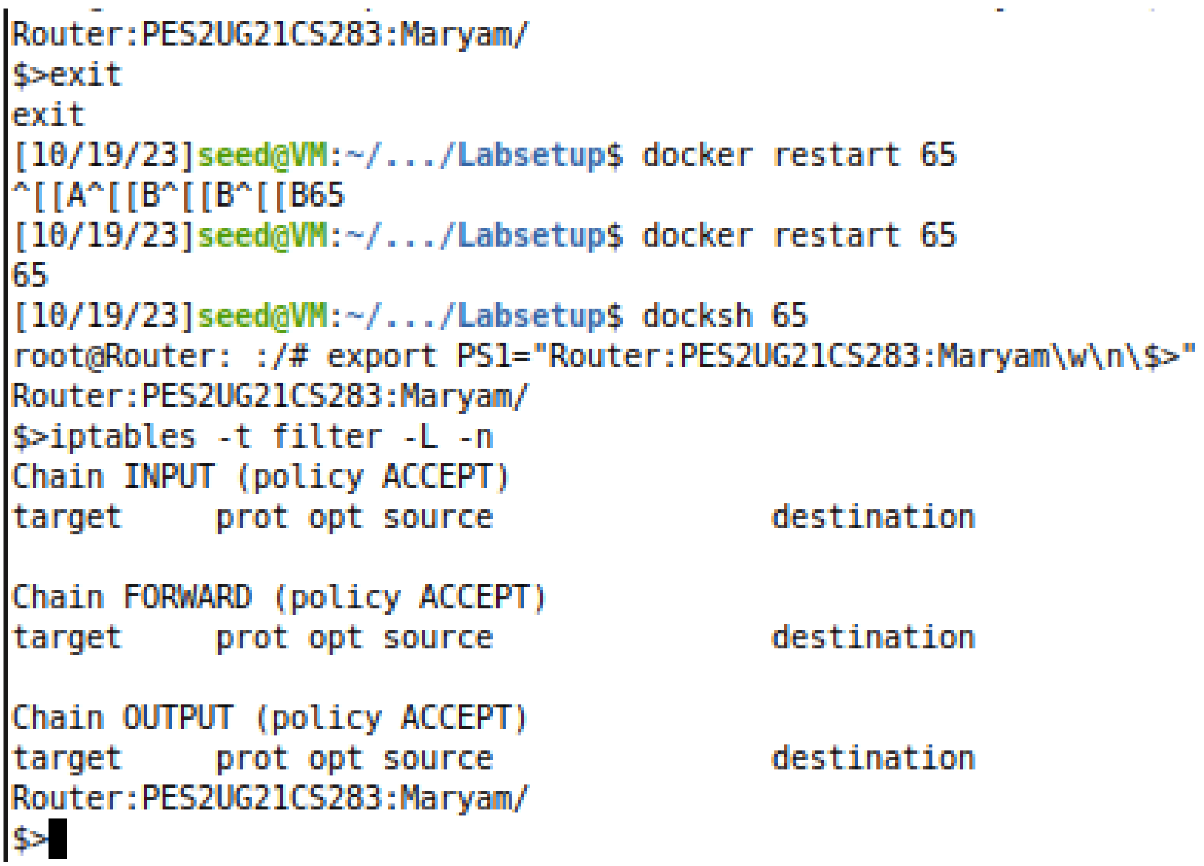
# iptables -P OUTPUT ACCEPT

# 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>



```
Router:PES2UG21CS283:Maryam/
$>exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
^[[A^[[B^[[B^[[B65
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$>iptables -t filter -L -n
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
Router:PES2UG21CS283:Maryam/
$>
```

Restart the docker

Task 2.C: Protecting Internal Servers

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

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.

Execute the following iptables commands on the **seed-router container** -

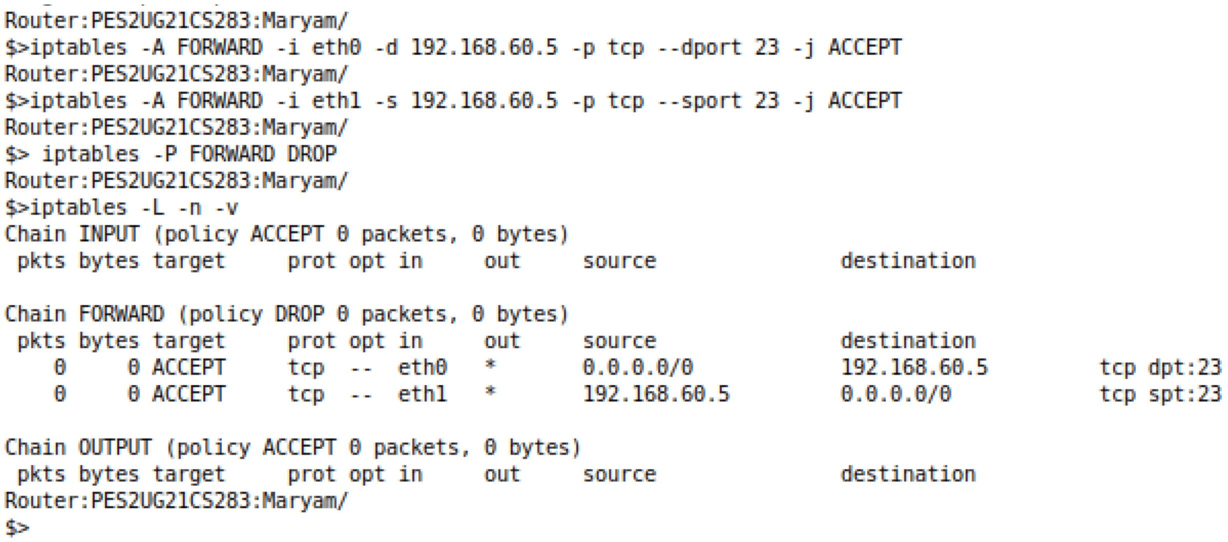
Commands:

# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT

# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT

# iptables -P FORWARD DROP

# iptables -L -n -v



```
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
Router:PES2UG21CS283:Maryam/
$> iptables -P FORWARD DROP
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 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   *       0.0.0.0/0            192.168.60.5         tcp dpt:23
    0     0 ACCEPT     tcp  --  eth1   *       192.168.60.5         0.0.0.0/0            tcp spt:23

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
Router:PES2UG21CS283:Maryam/
$>
```

We can see that all the rules have been added to the ip routing table

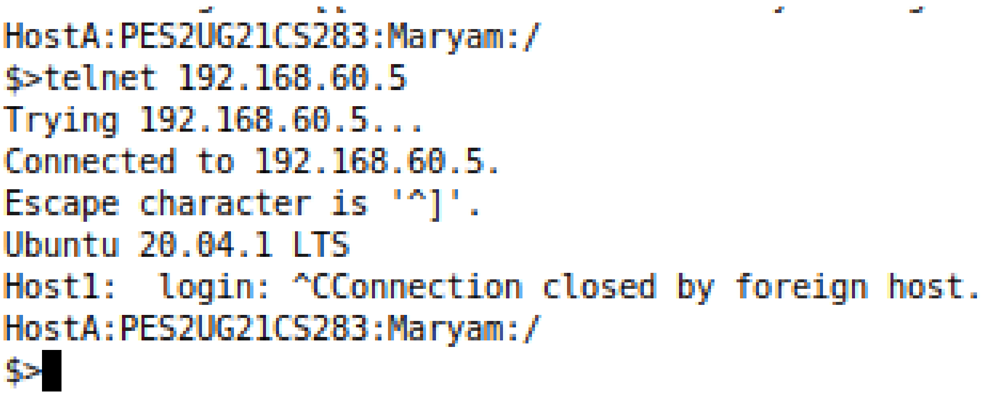
Now we shall see if these **restrictions have been enforced** in the network.

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.

On host A - 10.9.0.5

Command:

# telnet 192.168.60.5



```
HostA:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Host1:  login: ^CConnection closed by foreign host.
HostA:PES2UG21CS283:Maryam:/
$>
```

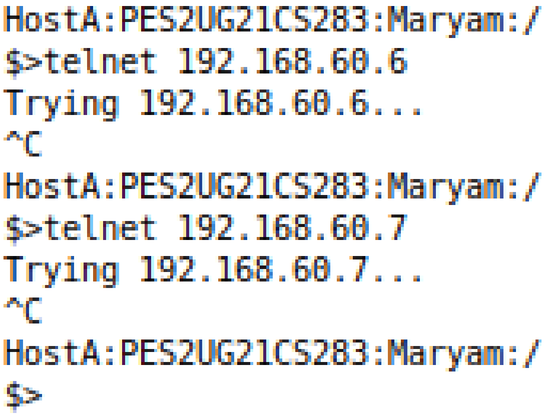
2. Outside hosts cannot access other internal servers.

On host A - 10.9.0.5

Command:

# telnet 192.168.60.6

# telnet 192.168.60.7



```
HostA:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.6
Trying 192.168.60.6...
^C
HostA:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.7
Trying 192.168.60.7...
^C
HostA:PES2UG21CS283:Maryam:/
$>
```

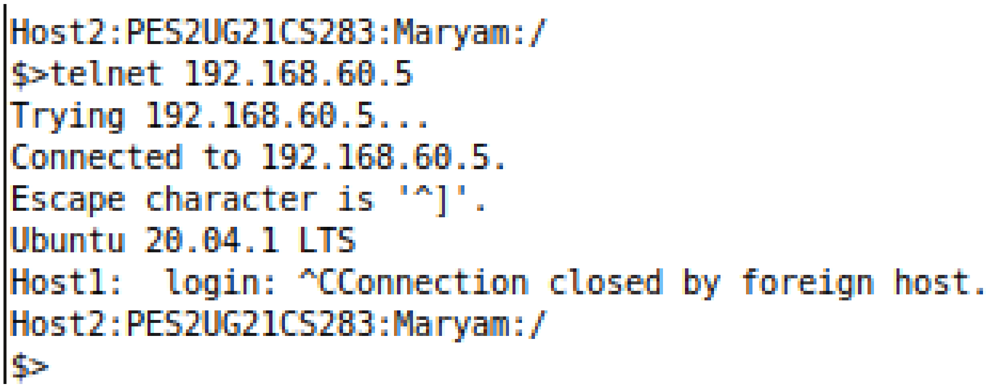
telnet is successful only for specified IP address from host A

3. Internal hosts can access all the internal servers.

On host2 - 192.168.60.6

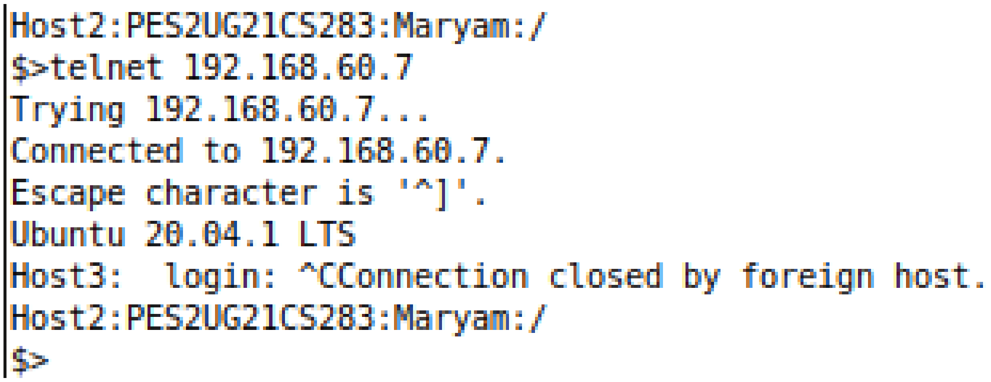
Command:

# telnet 192.168.60.5



```
Host2:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Host1:  login: ^CConnection closed by foreign host.
Host2:PES2UG21CS283:Maryam:/
$>
```

# telnet 192.168.60.7



```
Host2:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Host3:  login: ^CConnection closed by foreign host.
Host2:PES2UG21CS283:Maryam:/
$>
```

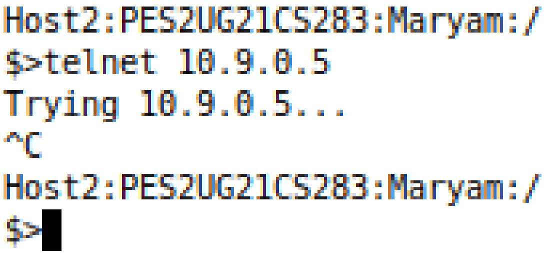
telnet works from host 2 since filter was set only to host A

4. Internal hosts cannot access external servers.

On host2 - 192.168.60.6

Command:

# telnet 10.9.0.5



```
Host2:PES2UG21CS283:Maryam:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
^C
Host2:PES2UG21CS283:Maryam:/
$>
```

telnet is successful only for specified IP address from host 2

Please report your observation with screenshots and explain the purpose for each rule

Cleanup

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

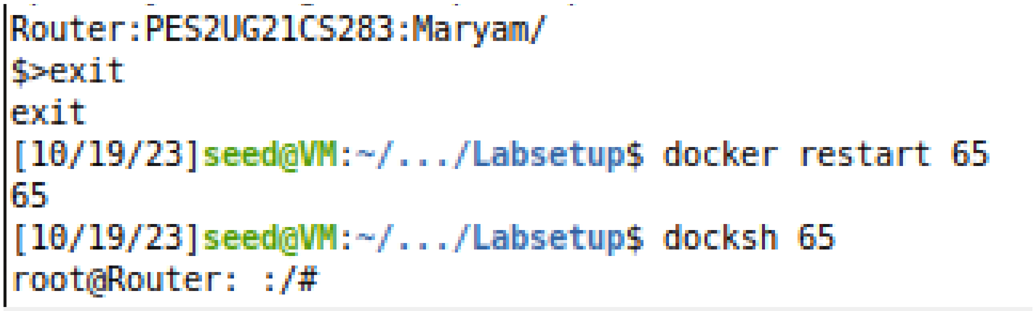
On seed-router

```
# 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>
```



```
Router:PES2UG21CS283:Maryam/
$>exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/#
```

Task 3: Connection Tracking and Stateful Firewall

In the previous task, we have only set up stateless firewalls, which inspect each packet independently. However, packets are usually not independent; they may be part of a TCP connection, or they may be ICMP packets triggered by other packets. Treating them independently does not take into consideration the context of the packets, and can thus lead to inaccurate, unsafe, or complicated firewall rules.

For example, if we would like to allow TCP packets to get into our network only if a connection was made first, we cannot achieve that easily using stateless packet filters, because when the firewall examines each individual TCP packet, it has no idea whether the packet belongs to an existing connection or not, unless the firewall maintains some state information for each connection. If it does that, it becomes a stateful firewall.

Task 3.A: Experiment with the Connection Tracking

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

This can be done using the following command:

```
# conntrack -L
```

Conntrack is a connection tracking command in linux

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:

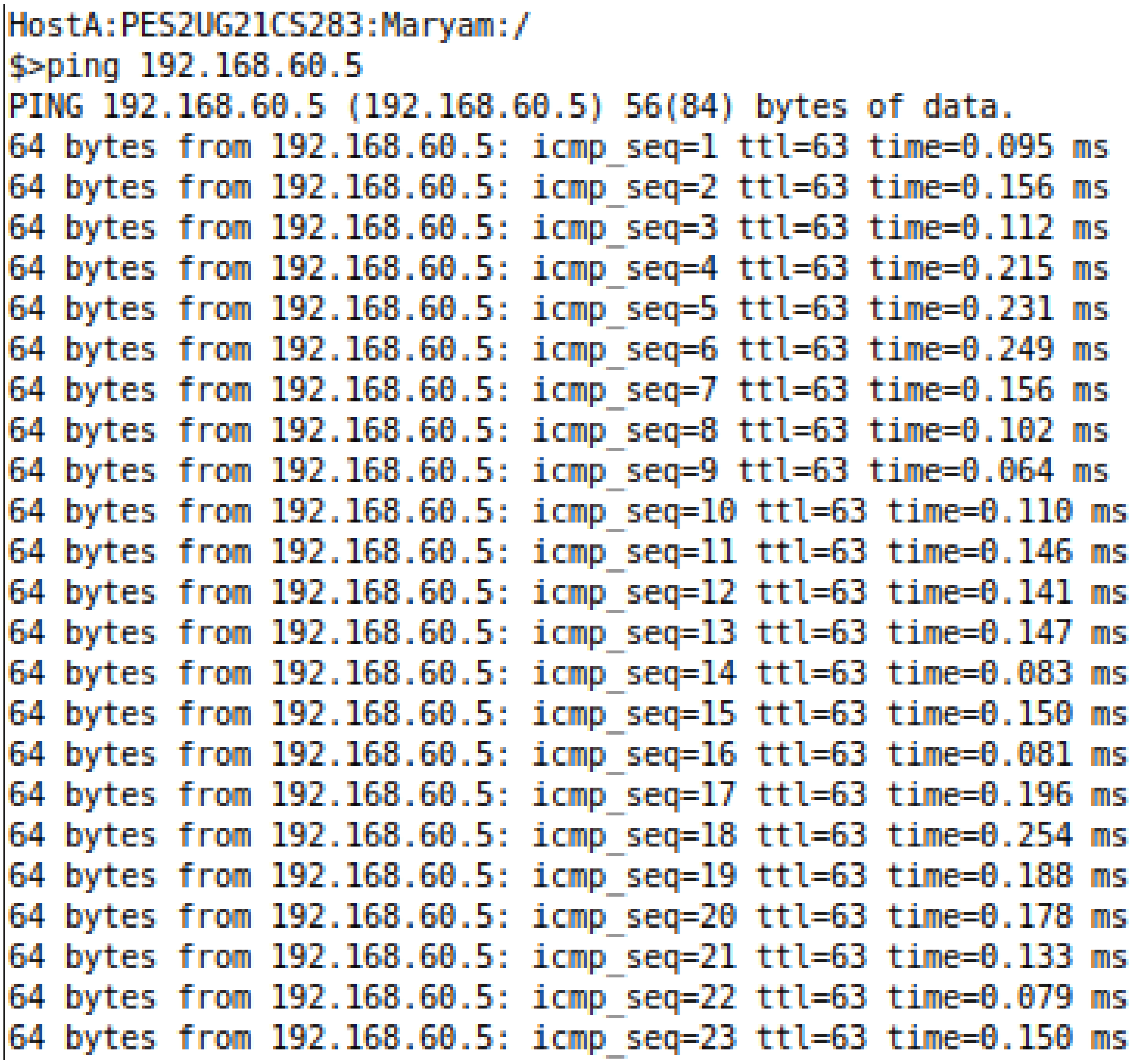
Run the following command and check the connection tracking information on the router. Describe your observation. How long can the ICMP connection state be kept?

On host A - 10.9.0.5

Command:

```
# ping 192.168.60.5
```

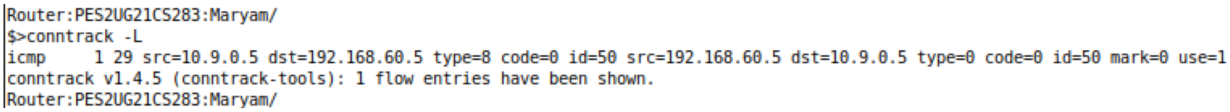
Let the ping run for sometime, then stop it (Ctrl + C)



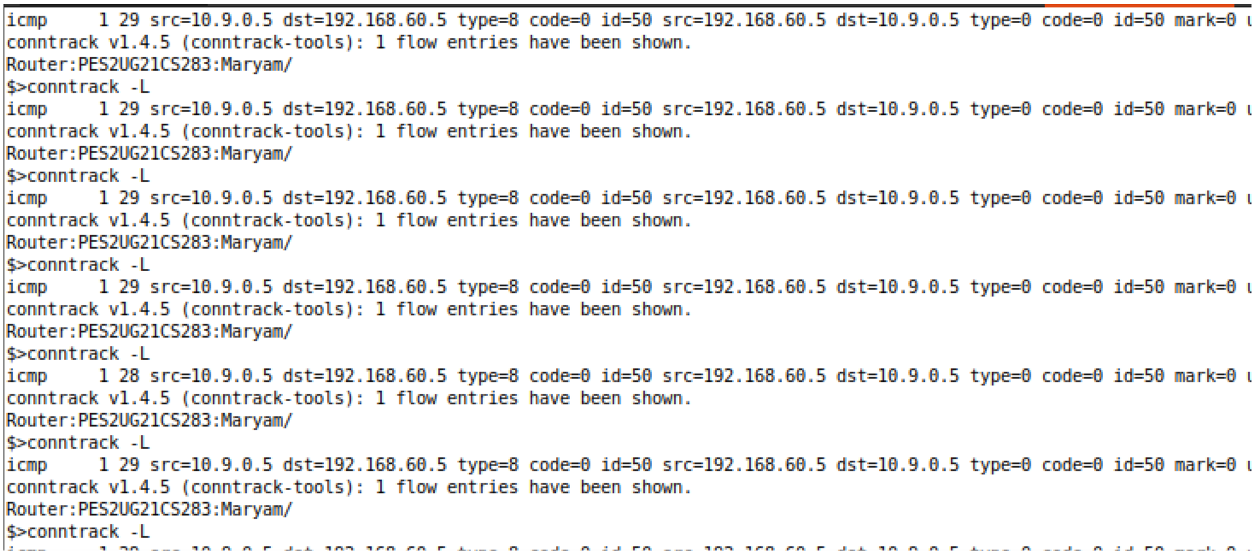
```
HostA:PES2UG21CS283:Maryam:/
$>ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.095 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.156 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.112 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.215 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.231 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.249 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.156 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.102 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.064 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.110 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.146 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.141 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.147 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.083 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.150 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.081 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.196 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.254 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.188 ms
64 bytes from 192.168.60.5: icmp_seq=20 ttl=63 time=0.178 ms
64 bytes from 192.168.60.5: icmp_seq=21 ttl=63 time=0.133 ms
64 bytes from 192.168.60.5: icmp_seq=22 ttl=63 time=0.079 ms
64 bytes from 192.168.60.5: icmp_seq=23 ttl=63 time=0.150 ms
```

Immediately move to the seed router and run

# conntrack -L



```
Router:PES2UG21CS283:Maryam/
$>conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
```



```
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 u
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 u
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 u
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 u
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
icmp     1 28 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 u
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=50 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=50 mark=0 u
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
```

Keep executing the above command, and you shall see the timer decreasing (time remaining for the connection)

We see decreasing numbers i.e. 29 to 28

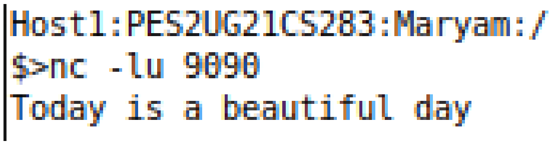
<u>UDP experiment</u>:

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

On host 1 - 192.168.60.5

Command:

# nc -lu 9090



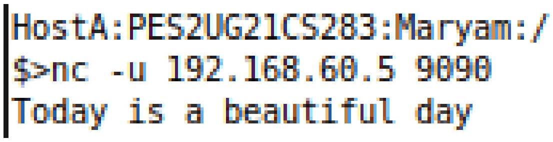
```
Host1:PES2UG21CS283:Maryam:/
$>nc -lu 9090
Today is a beautiful day
```

On host A - 10.9.0.5

Command:

# nc -u 192.168.60.5 9090

<type something and hit return>



```
HostA:PES2UG21CS283:Maryam:/
$>nc -u 192.168.60.5 9090
Today is a beautiful day
```

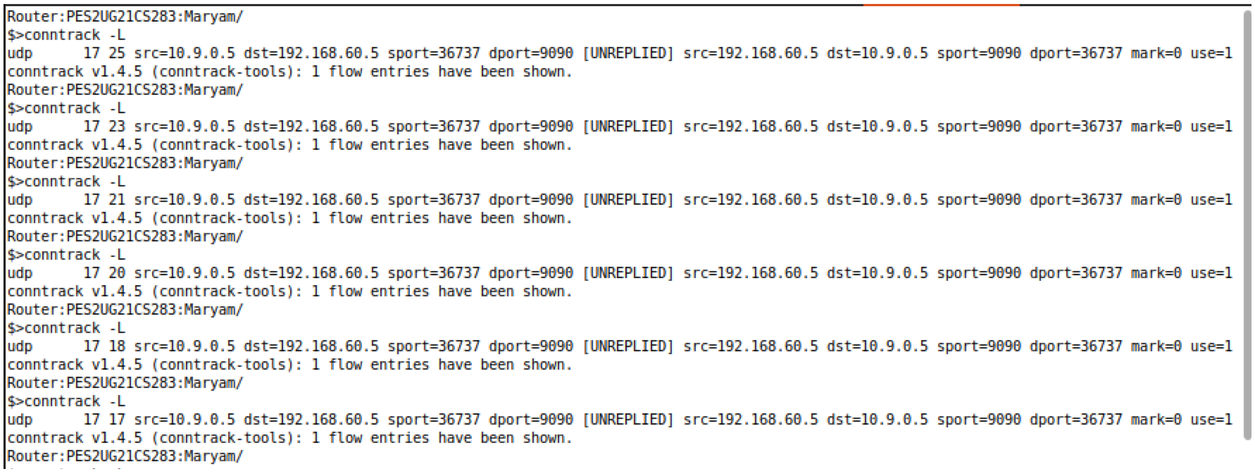
On seed router

# conntrack -L

Execute the above command continuously and see if you spot the change in the timer.

Close the UDP connection and execute conntrack -L on the router container.

Take screenshots and explain your observations.



```
Router:PES2UG21CS283:Maryam/
$>conntrack -L
udp      17 25 src=10.9.0.5 dst=192.168.60.5 sport=36737 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36737 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
udp      17 23 src=10.9.0.5 dst=192.168.60.5 sport=36737 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36737 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
udp      17 21 src=10.9.0.5 dst=192.168.60.5 sport=36737 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36737 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
udp      17 20 src=10.9.0.5 dst=192.168.60.5 sport=36737 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36737 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
udp      17 18 src=10.9.0.5 dst=192.168.60.5 sport=36737 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36737 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
udp      17 17 src=10.9.0.5 dst=192.168.60.5 sport=36737 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36737 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
```

We get a constant output 17 when we run the conntrack -L command

Whatever we type on host A is seen on Host 1

TCP experiment:

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

On host 1 - 192.168.60.5

Command:

# nc -l 9090

On host A - 10.9.0.5

Command:

# nc 192.168.60.5 9090

<type something and hit return>

On seed router

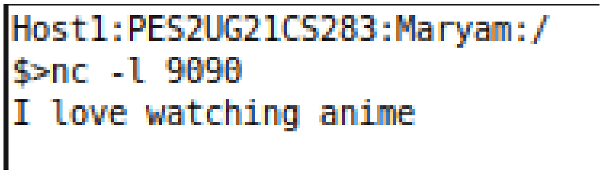
# conntrack -L

Execute the above command continuously and see if you spot the change in the timer.

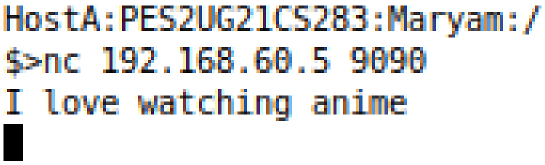
Close the TCP connection and execute conntrack -L on the router container.

Do you spot any difference?

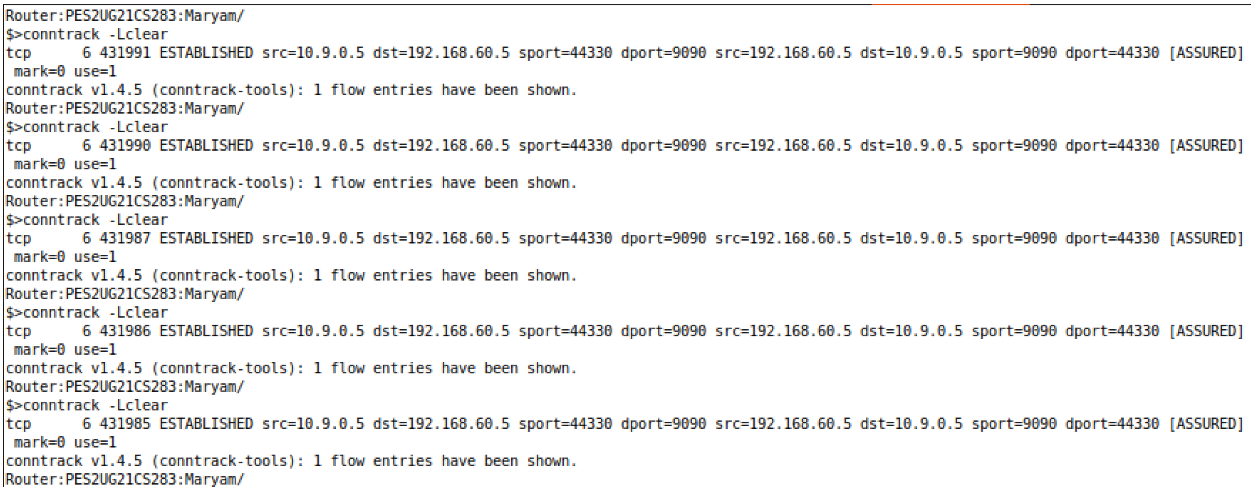
Take screenshots and explain your observations.



```
Host1:PES2UG21CS283:Maryam:/
$>nc -l 9090
I love watching anime
```

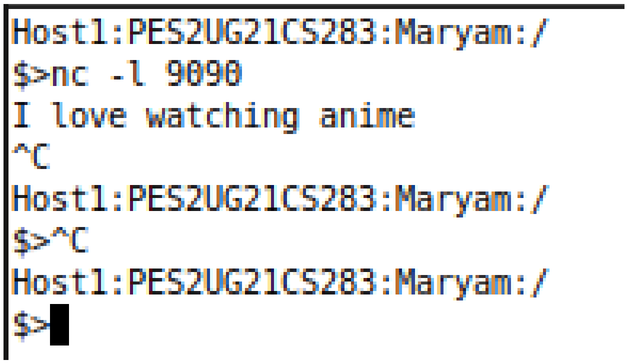


```
HostA:PES2UG21CS283:Maryam:/
$>nc 192.168.60.5 9090
I love watching anime
```

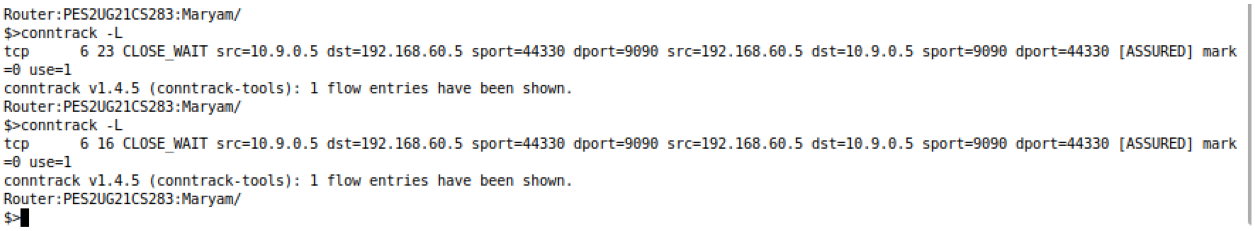


```
Router:PES2UG21CS283:Maryam/
$>conntrack -Lclear
tcp      6 431991 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED]
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -Lclear
tcp      6 431990 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED]
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -Lclear
tcp      6 431987 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED]
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -Lclear
tcp      6 431986 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED]
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -Lclear
tcp      6 431985 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED]
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
```

We get different sequence number



```
Host1:PES2UG21CS283:Maryam:/
$>nc -l 9090
I love watching anime
^C
Host1:PES2UG21CS283:Maryam:/
$>^C
Host1:PES2UG21CS283:Maryam:/
$>
```



```
Router:PES2UG21CS283:Maryam/
$>conntrack -L
tcp      6 23 CLOSE_WAIT src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED] mark
=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>conntrack -L
tcp      6 16 CLOSE_WAIT src=10.9.0.5 dst=192.168.60.5 sport=44330 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=44330 [ASSURED] mark
=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
Router:PES2UG21CS283:Maryam/
$>
```

When we close one of the connection and try the same thing, it shows connection is closed

Task 3.B: Setting Up a Stateful Firewall

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

For this task we have to rewrite the firewall rules in 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).

On seed-router execute

Commands:

# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT

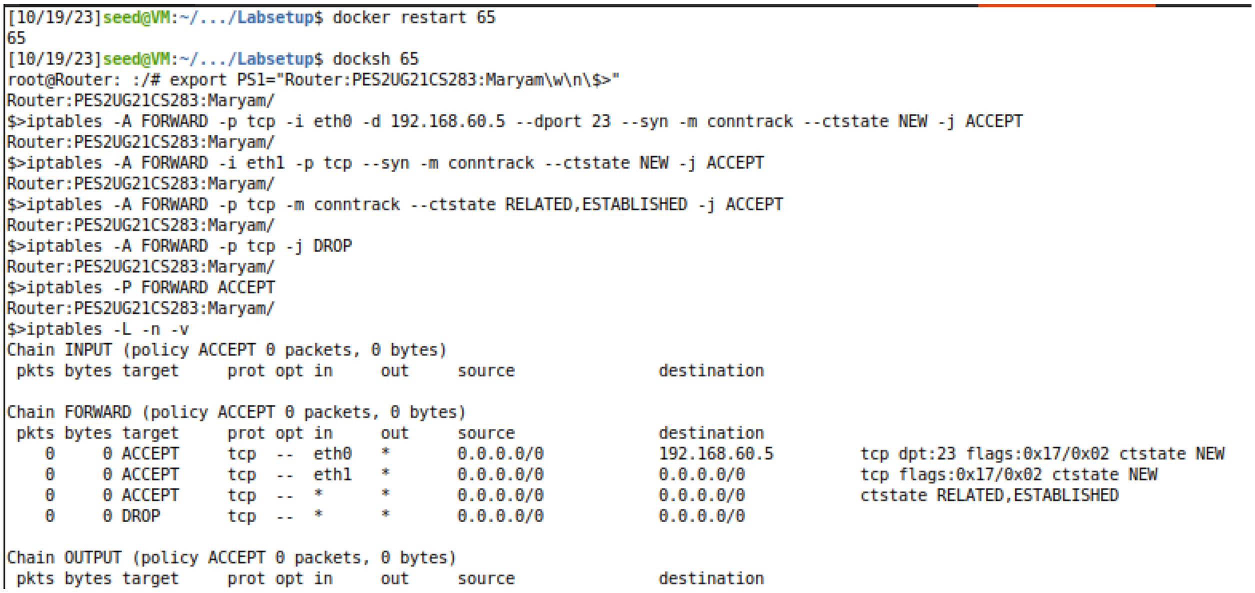
# iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT

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

# iptables -A FORWARD -p tcp -j DROP

# iptables -P FORWARD ACCEPT

# iptables -L -n -v



```
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -p tcp -j DROP
Router:PES2UG21CS283:Maryam/
$>iptables -P FORWARD ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  eth0   *       0.0.0.0/0            192.168.60.5         tcp dpt:23 flags:0x17/0x02 ctstate NEW
    0     0 ACCEPT     tcp  --  eth1   *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x02 ctstate NEW
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
    0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0

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

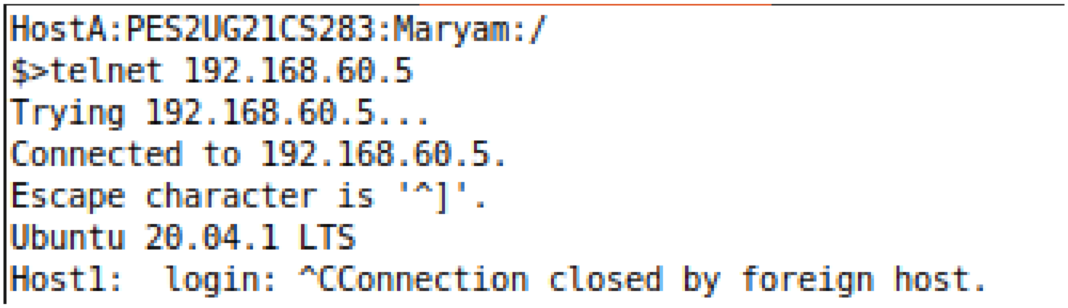
Now we shall see if these **restrictions have been enforced** in the network.

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.

On host A - 10.9.0.5

Command:

# telnet 192.168.60.5



```
HostA:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Host1:  login: ^CConnection closed by foreign host.
```

telnet is successful only for specified IP address from host A

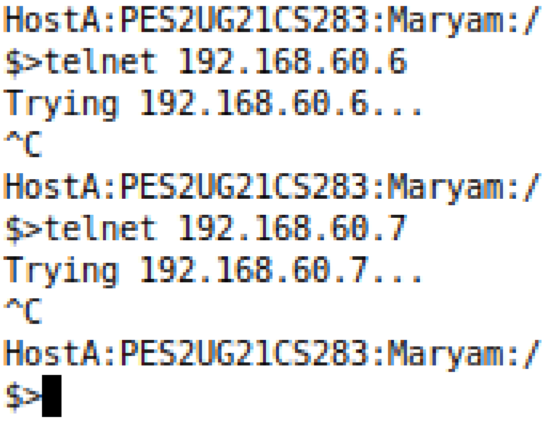
2. Outside hosts cannot access other internal servers.

On host A - 10.9.0.5

Command:

# telnet 192.168.60.6

# telnet 192.168.60.7



```
HostA:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.6
Trying 192.168.60.6...
^C
HostA:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.7
Trying 192.168.60.7...
^C
HostA:PES2UG21CS283:Maryam:/
$>
```

telnet is successful only for specified IP address from host A

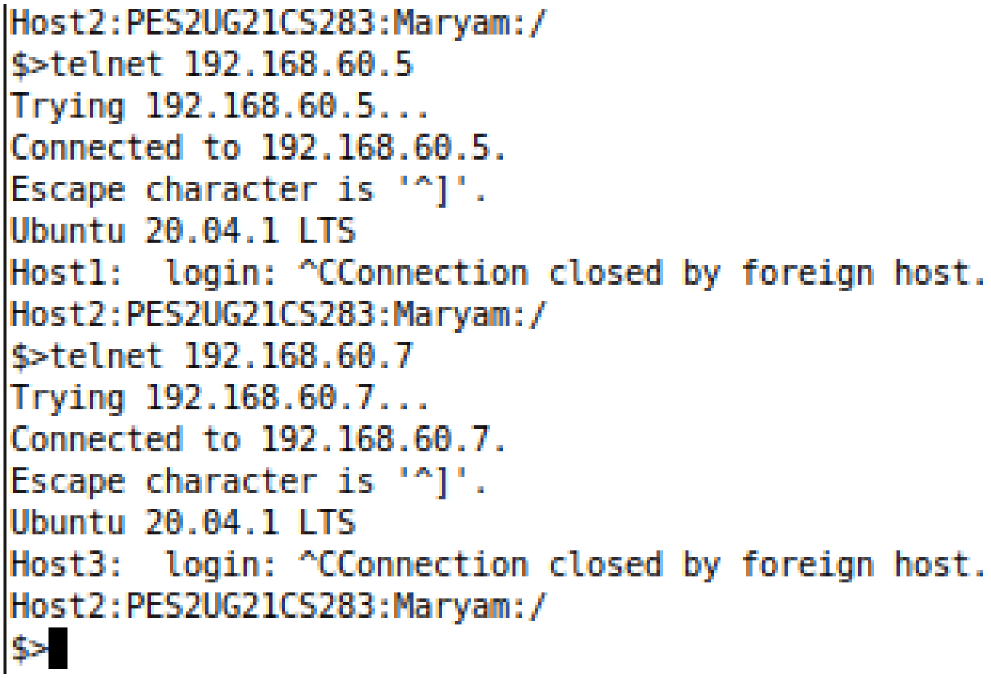
3. Internal hosts can access all the internal servers.

On host2 - 192.168.60.6

Command:

# telnet 192.168.60.5

# telnet 192.168.60.7



```
Host2:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Host1:  login: ^CConnection closed by foreign host.
Host2:PES2UG21CS283:Maryam:/
$>telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Host3:  login: ^CConnection closed by foreign host.
Host2:PES2UG21CS283:Maryam:/
$>
```

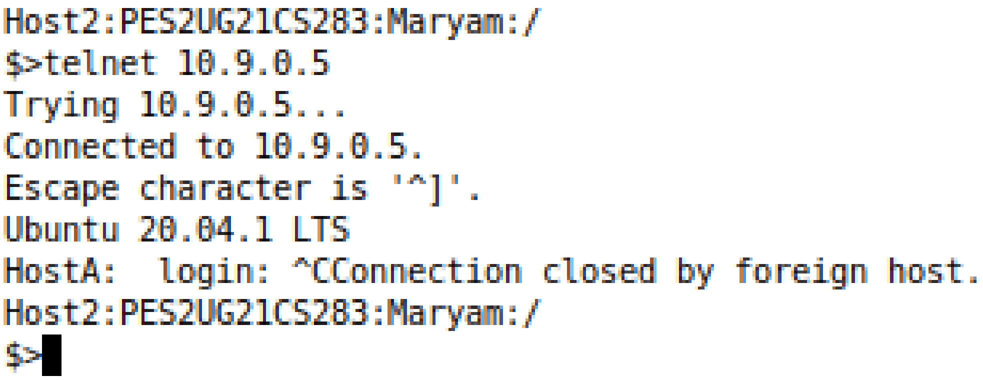
telnet is successful for all IP addresses from host 2

4. Internal hosts can access external servers.

On host2 - 192.168.60.6

Command:

# telnet 10.9.0.5



```
Host2:PES2UG21CS283:Maryam:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
HostA:  login: ^CConnection closed by foreign host.
Host2:PES2UG21CS283:Maryam:/
$>
```

telnet is successful for all IP addresses from host 2

Please report your observation with screenshots and explain the purpose for each rule

Highlight the differences between stateless and stateful firewalls based on your observation

Stateless firewalls filter traffic based on individual packet attributes without considering connection state. Stateful firewalls use connection tracking to maintain state information, allowing them to make context-aware decisions about whether to allow or block traffic based on connection states, making them more secure and suitable for complex security requirements.

Cleanup

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

On seed-router

```
# 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>
```

Task 4: Limiting Network Traffic

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

Please run the following commands on the 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.

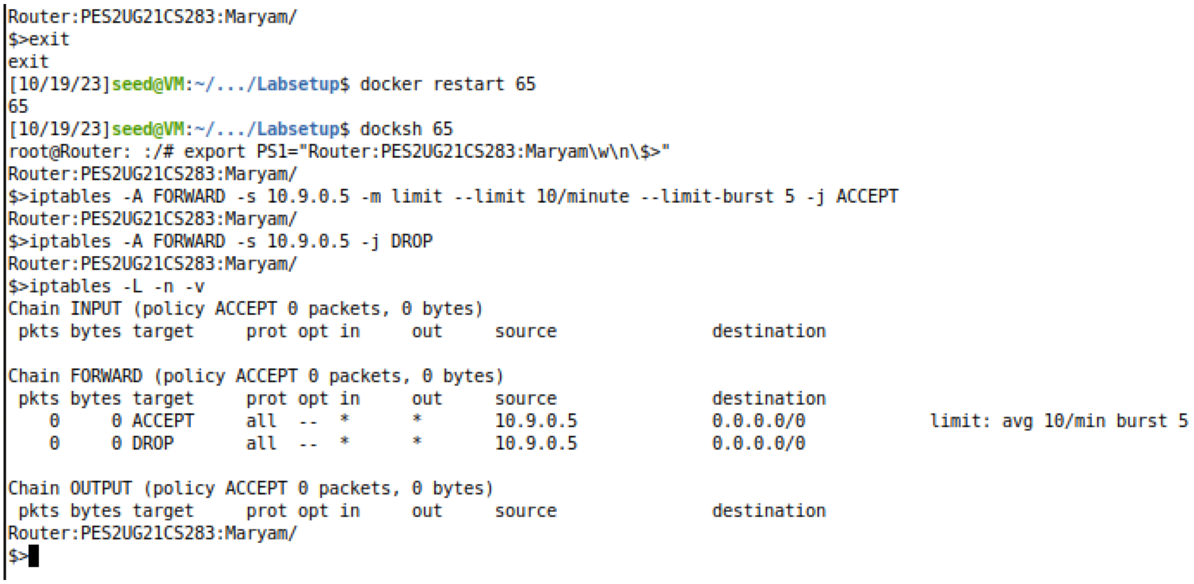
On seed router execute -

Command:

# 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

# iptables -L -n -v



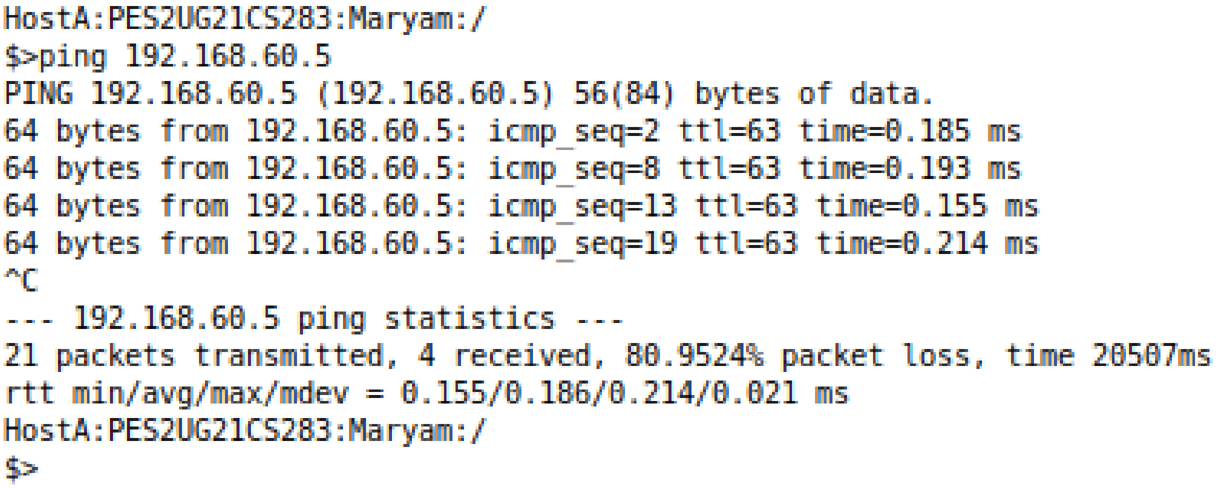
```
Router:PES2UG21CS283:Maryam/
$>exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -A FORWARD -s 10.9.0.5 -j DROP
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  *      *       10.9.0.5             0.0.0.0/0            limit: avg 10/min burst 5
    0     0 DROP       all  --  *      *       10.9.0.5             0.0.0.0/0

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
Router:PES2UG21CS283:Maryam/
$>
```

On host A - 10.9.0.5

Command:

# ping 192.168.60.5

Please report your observation with screenshots and explain the purpose for each rule



```
HostA:PES2UG21CS283:Maryam:/
$>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=2 ttl=63 time=0.185 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.193 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=0.214 ms
^C
--- 192.168.60.5 ping statistics ---
21 packets transmitted, 4 received, 80.9524% packet loss, time 20507ms
rtt min/avg/max/mdev = 0.155/0.186/0.214/0.021 ms
HostA:PES2UG21CS283:Maryam:/
$>
```

We observe that there is packet loss as we have limited the network traffic

Clean the rules and now perform the same task without the second rule -

On seed router execute -

Command:

# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT

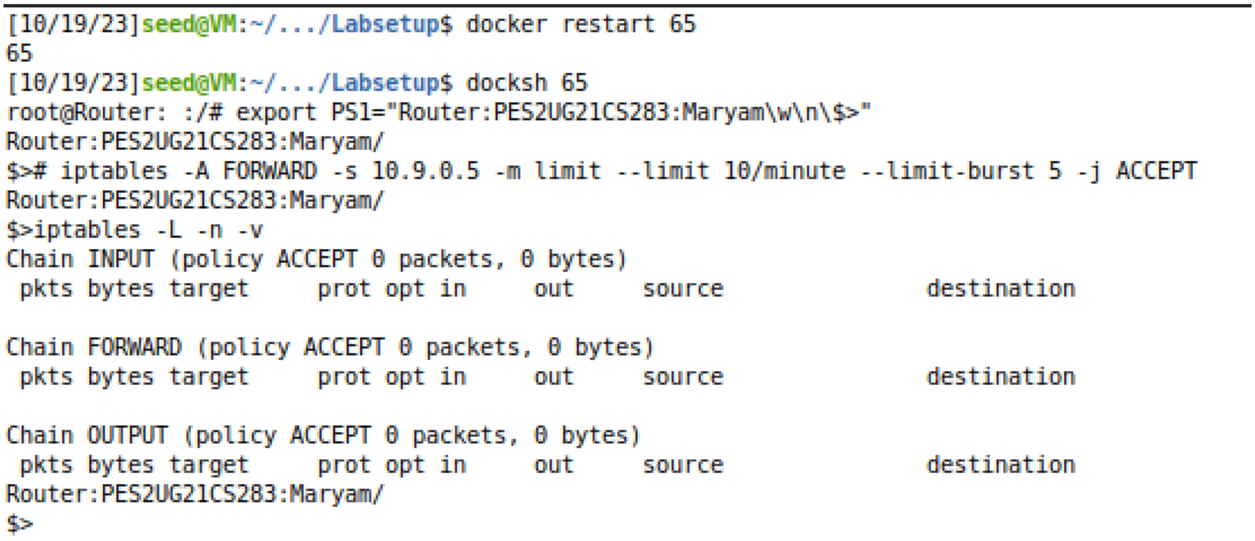
# iptables -L -n -v

On host A - 10.9.0.5

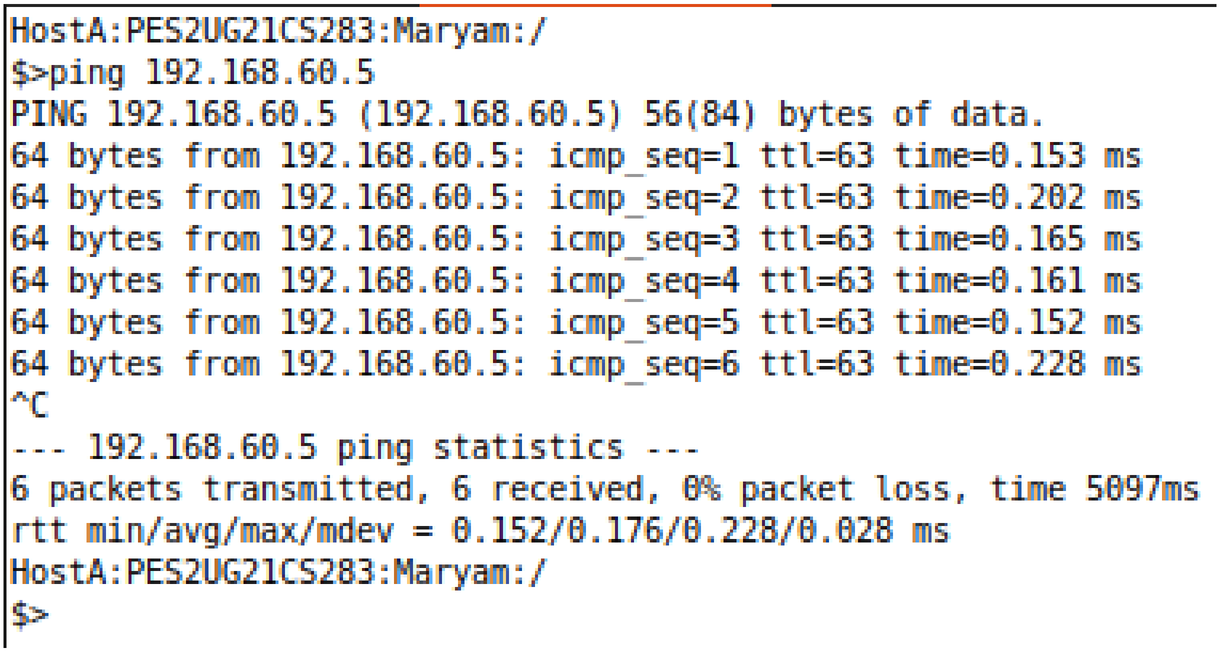
Command:

# ping 192.168.60.5

Please report your observation with screenshots and explain the purpose for each rule



```
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$># iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
Router:PES2UG21CS283:Maryam/
$>
```



```
HostA:PES2UG21CS283:Maryam:/
$>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=0.153 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.202 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.165 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.161 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.152 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.228 ms
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5097ms
rtt min/avg/max/mdev = 0.152/0.176/0.228/0.028 ms
HostA:PES2UG21CS283:Maryam:/
$>
```

Ping is successful on removing the second rule. All packets are received.

Clean the rules before proceeding to the next task

Task 5: Load Balancing

The iptables are very powerful. In addition to firewalls, it has many other applications. We will not be able to cover all its applications in this lab, but we will be experimenting with one of the applications, load balancing. In this task, we will use it to balance three UDP servers running in the internal network. Let’s first start the server on each of the hosts: 192.168.60.5, 192.168.60.6, and 192.168.60.7 (the -k option indicates that the server can receive UDP datagrams from multiple hosts):

```
# nc -luk 8080
```

We can use the statistic module to achieve load balancing.

There are two modes: random and nth. We will conduct experiments using both of them.

Using the nth mode (round-robin) - On the router container, we set the following rule, which applies to all the UDP packets going to port 8080. The nth mode of the statistic module is used; it implements a round-robin load balancing policy:

In this subtask we shall implement policies to equally divide the incoming packets between the three interval servers.

On the seed-router container:

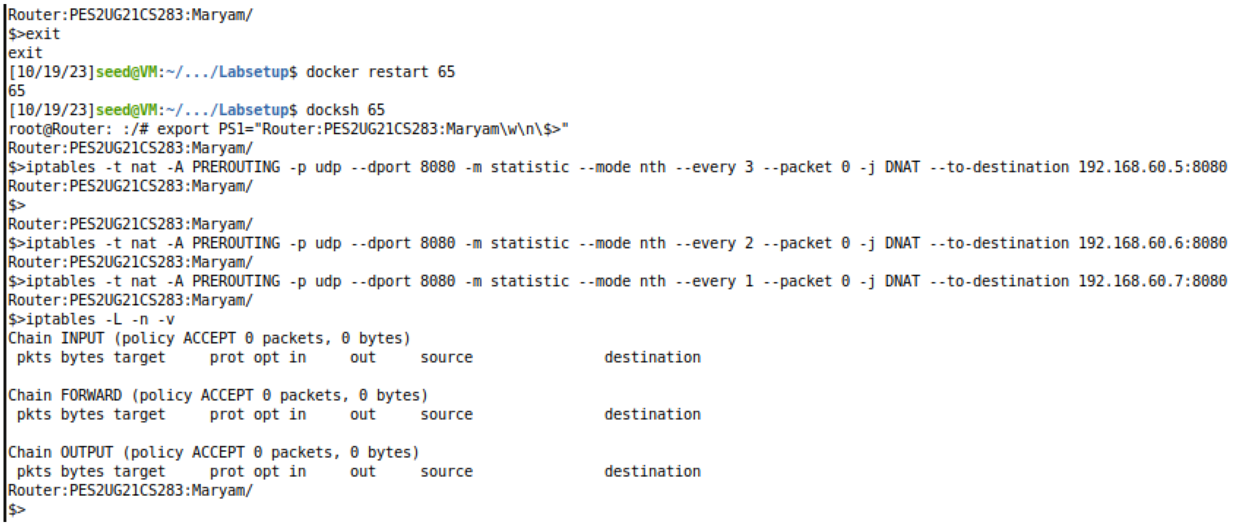
Command:

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

# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080

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

# iptables -L -n -v



```
Router:PES2UG21CS283:Maryam/
$>exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
Router:PES2UG21CS283:Maryam/
$>
Router:PES2UG21CS283:Maryam/
$>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080
Router:PES2UG21CS283:Maryam/
$>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 1 --packet 0 -j DNAT --to-destination 192.168.60.7:8080
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
Router:PES2UG21CS283:Maryam/
$>
```

On host1 - 192.168.60.5, host2 - 192.168.60.6 and host3 - 192.168.60.7 start the server

Command:

```
# nc -luk 8080
```

On host A - 10.9.0.5

Command:

```
# nc -u 10.9.0.11 8080
```

< enter 3 words, wait 30 seconds before entering the next word>

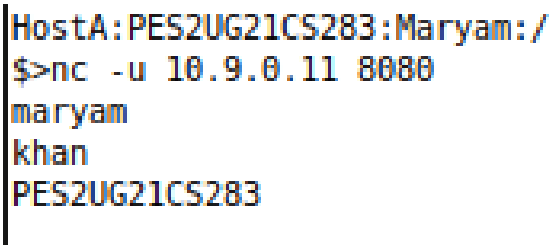
For example :

```
# nc -u 10.9.0.11 8080
Hello 1
Wait 30 seconds
Hello 2
Wait 30 seconds
Hello 3
```

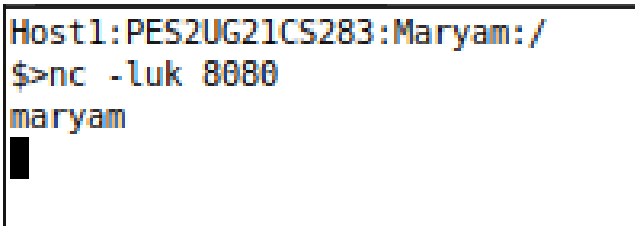
‘Hello 1’ appears in the host 1 terminal, ‘Hello 2’ appears in the host 2 terminal etc.

If you do not have a waiting period all the three statements will be considered as a single packet, so please wait for sometime before entering some text.

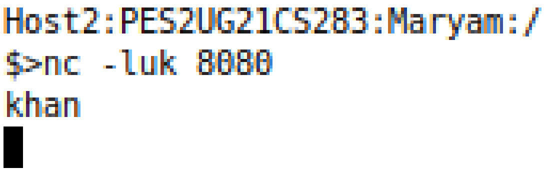
Students should enter their First Name, Last Name and SRN.



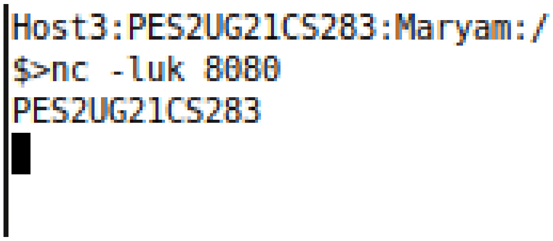
```
HostA:PES2UG21CS283:Maryam:/
$>nc -u 10.9.0.11 8080
maryam
khan
PES2UG21CS283
```



```
Host1:PES2UG21CS283:Maryam:/
$>nc -luk 8080
maryam
```



```
Host2:PES2UG21CS283:Maryam:/
$>nc -luk 8080
khan
█
```



```
Host3:PES2UG21CS283:Maryam:/
$>nc -luk 8080
PES2UG21CS283
█
```

All the three words are divided in the three terminals, this is called load balancing

Clean the rules before proceeding to the next task

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

On the seed-router container:

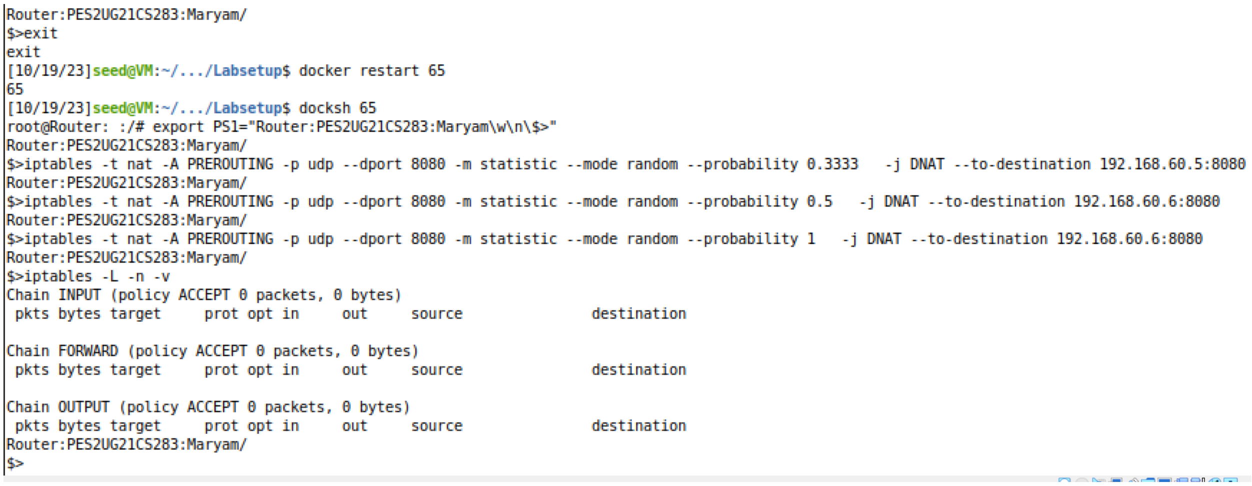
Command:

# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.3333 -j DNAT --to-destination 192.168.60.5:8080

# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-destination 192.168.60.6:8080

# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1 -j DNAT --to-destination 192.168.60.6:8080

# iptables -L -n -v



```
Router:PES2UG21CS283:Maryam/
$>exit
exit
[10/19/23]seed@VM:~/.../Labsetup$ docker restart 65
65
[10/19/23]seed@VM:~/.../Labsetup$ docksh 65
root@Router: :/# export PS1="Router:PES2UG21CS283:Maryam\w\n\$>"
Router:PES2UG21CS283:Maryam/
$>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.3333   -j DNAT --to-destination 192.168.60.5:8080
Router:PES2UG21CS283:Maryam/
$>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5   -j DNAT --to-destination 192.168.60.6:8080
Router:PES2UG21CS283:Maryam/
$>iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1   -j DNAT --to-destination 192.168.60.6:8080
Router:PES2UG21CS283:Maryam/
$>iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
Router:PES2UG21CS283:Maryam/
$>
```

On host1 - 192.168.60.5, host2 - 192.168.60.6, host3 - 192.168.60.7 start the server

Command:

# nc -luk 8080

On host A - 10.9.0.5

Command:

# nc -u 10.9.0.11 8080

< enter 3 words, wait 30 seconds before entering the next word>

For example :

# nc -u 10.9.0.11 8080

Hello 1

Wait 30 seconds

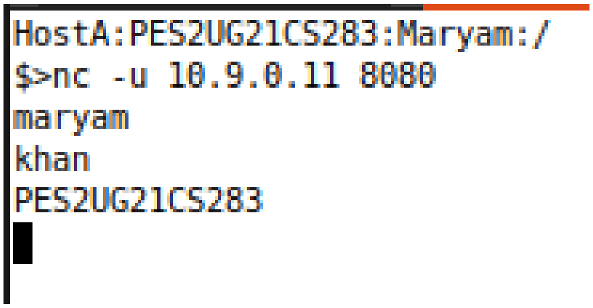
Hello 2

Wait 30 seconds

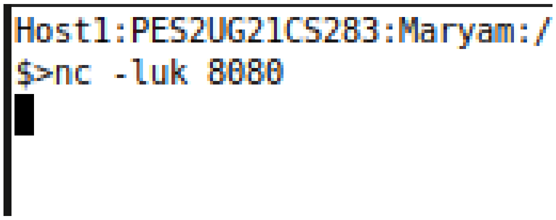
Hello 3

This time each server **<u>may not get the same amount of traffic</u>**.

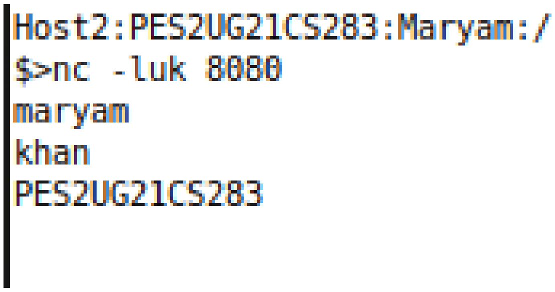
Please provide some explanation for the rules with the appropriate screenshots.



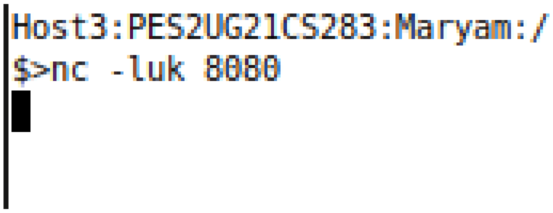
```
HostA:PES2UG21CS283:Maryam:/
$>nc -u 10.9.0.11 8080
maryam
khan
PES2UG21CS283
```



```
Host1:PES2UG21CS283:Maryam:/
$>nc -luk 8080
```



```
Host2:PES2UG21CS283:Maryam:/
$>nc -luk 8080
maryam
khan
PES2UG21CS283
```



```
Host3:PES2UG21CS283:Maryam:/
$>nc -luk 8080
```

We notice that the load is balanced in some random way.