Name : Vinay V	SRN: PES1UG24CS840
Sub: CNS	LAB 7: Firewall Exploration Lab

Task 1: Implementing a Simple Firewall:

```
[10/15/25]seed@VM:~/.../kernel_module$ export PS1="PES1UG24CS840:VINAY:\w\$
 "
PES1UG24CS840:VINAY:~/.../kernel_module$ ls
hello.c  Makefile
PES1UG24CS840:VINAY:~/.../kernel_module$ - $ sudo dmesg -k -w
-: command not found
PES1UG24CS840:VINAY:~/.../kernel_module$ sudo dmesg -k -w
[    0.000000] Linux version 5.4.0-54-generic (buildd@lcy01-amd64-024) (gcc
 version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)) #60-Ubuntu SMP Fri Nov 6 10:
37:59 UTC 2020 (Ubuntu 5.4.0-54.60-generic 5.4.65)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.4.0-54-generic root
=UUID=a91f1a43-2770-4684-9fc3-b7abfd786c1d ro quiet splash
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point
 registers'
```

- The terminal navigates to the kernel_module directory containing the hello.c source file and Makefile.
- sudo dmesg -k -w is executed to **continuously monitor kernel logs** in real time.
- The kernel version is shown as 5.4.0-54-generic, confirming the active build environment.
- This setup ensures any printk() messages from the kernel module will be visible instantly.
- It acts as a **log monitoring window** for verifying module behavior.

```
PES1UG24CS840:VINAY:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labsetup/v
olumes/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labsetup/volumes/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labsetup/volumes/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/Labsetup/volumes/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
PES1UG24CS840:VINAY:~/.../kernel_module$ sudo insmod hello.ko
PES1UG24CS840:VINAY:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
PES1UG24CS840:VINAY:~/.../kernel_module$ sudo rmmod hello
```

- make successfully compiles the source code into hello.ko using the kernel build system.
- sudo insmod hello.ko inserts the kernel module without any errors.
- lsmod | grep hello confirms the module is loaded and running in the kernel.
- The size 16384 shows memory allocated for the module.
- sudo rmmod hello cleanly removes the module from the kernel.

```
ntu-software-local-file  pid=2
[   939.985208] hello: module ve
 key missing - tainting kernel
[   939.986103] Hello World!
[   956.439796] Bye-bye World!.
```

- Kernel log displays a message indicating module verification key is missing (expected for unsigned modules).
- The printk() message "Hello World!" appears after inserting the module.
- The message "Bye-bye World!" appears after removing the module.
- These messages confirm that the initialization and cleanup functions executed correctly.
- The test verifies successful loading and unloading of the kernel module.

Task 1.B: Implement a Simple Firewall Using Netfilter

```
seed@VM: ~/.../packet_filter
obj-m += seedFilter.o
#obj-m += seedPrint.o
#obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell
les

clean:
	make -C /lib/modules/$(shell
n

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter
```

- The `Makefile` has `seedFilter.o` uncommented, ensuring that only this module is built for packet filtering.
- The other modules `seedPrint.o` and `seedBlock.o` are commented out to avoid conflicts.
- This matches the task instruction of using `seedFilter.ko` for DNS traffic filtering.
- The `ins` and `rm` targets define shortcuts to insert and remove the module.
- Correct configuration ensures only the firewall module is loaded during testing

```
[10/15/25]seed@VM:~/.../packet_filter$ vim Makefile
[10/15/25]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/L
absetup/volumes/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-gener
ic'
  CC [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedFi
lter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedFi
lter.mod.o
  LD [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedFi
lter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generi
c'
[10/15/25]seed@VM:~/.../packet_filter$ export PS1="PES1UG24CS840:V
INAY\w\$ "
PES1UG24CS840:VINAY~/.../packet_filter$
PES1UG24CS840:VINAY~/.../packet_filter$ sudo insmod seedFilter.ko
PES1UG24CS840:VINAY~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
PES1UG24CS840:VINAY~/.../packet_filter$ dig @8.8.8.8 www.example.c
om

; <<>> 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

PES1UG24CS840:VINAY~/.../packet_filter$
```

- The module `seedFilter.ko` was successfully compiled and inserted into the kernel.
- `lsmod | grep seedFilter` confirms the module is loaded.
- `dig @8.8.8.8 www.example.com` fails with *"connection timed out; no servers could be reached"*, which means packets were blocked by the firewall.
- This behavior proves that the Netfilter-based LKM is intercepting and filtering outgoing UDP packets to port 53 (DNS).
- The custom prompt `PS1` change is unrelated to firewall function but shows environment customization.

```
tLoop=true )
[ 4065.192343] Registering filters.
[ 4122.315559] *** LOCAL_OUT
[ 4122.315563]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4122.316024] *** LOCAL_OUT
[ 4122.316026]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 4122.316034] *** Dropping 8.8.8.8 (UDP), port 53
[ 4127.317141] *** LOCAL_OUT
[ 4127.317149]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 4127.317280] *** Dropping 8.8.8.8 (UDP), port 53
[ 4132.317815] *** LOCAL_OUT
[ 4132.317821]     10.0.2.15  --> 8.8.8.8 (UDP)
[ 4132.317849] *** Dropping 8.8.8.8 (UDP), port 53
[ 4186.597559] *** LOCAL_OUT
[ 4186.597570]     10.0.2.15  --> 34.107.243.93 (TCP)
[ 4207.388497] *** LOCAL_OUT
```

- Kernel logs show `Registering filters.` confirming that the firewall hooks were initialized.

- Multiple lines indicate:
- **LOCAL_OUT** hook is triggered for outgoing packets.
- Packets to `8.8.8.8` on UDP port 53 are **dropped** as per the hardcoded policy.
- Other packets (e.g., to 127.0.0.1 or a TCP destination) are allowed through.
- The selective blocking demonstrates that Netfilter hook logic is working as intended.
- Logging at this stage provides visibility into which packets are intercepted and filtered.

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

```
#obj-m += seedFilter.o
obj-m += seedPrint.o
#obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell unam
les
clean:
	make -C /lib/modules/$(shell unam
n

ins:
```

Firefox Web Browser

- The Makefile is updated to **uncomment** `seedPrint.o` and comment out the other modules.
- This ensures only the print-based Netfilter hook module is compiled and inserted.
- This step is required to **observe packet flow** through all Netfilter hooks without filtering.
- The `ins` and `rm` targets remain the same, simplifying module insertion/removal.
- Correct setup allows observation of hook invocation sequence during DNS resolution.

```
PES1UG24CS840:VINAY~/.../packet_filter$ sudo dmesg -C
PES1UG24CS840:VINAY~/.../packet_filter$ vim Makefile
PES1UG24CS840:VINAY~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/L
absetup/volumes/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-gener
ic'
  CC [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedPr
int.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedPr
int.mod.o
  LD [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedPr
int.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generi
c'
PES1UG24CS840:VINAY~/.../packet_filter$  sudo insmod seedPrint.ko
PES1UG24CS840:VINAY~/.../packet_filter$ lsmod | grep seedPrint
seedPrint              16384  0
PES1UG24CS840:VINAY~/.../packet_filter$ dig @8.8.8.8 www.example.c
om

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
```

- Kernel messages are cleared (`sudo dmesg -C`) before starting the test.
- `make` compiles `seedPrint.ko` successfully using the kernel build environment.
- `sudo insmod seedPrint.ko` inserts the module without errors.
- `lsmod | grep seedPrint` confirms the module is active in the kernel.
- The setup is ready to capture packet traversal information for DNS requests.

```
PES1UG24CS840:VINAY~/.../packet_filter$ dig @8.8.8.8 www.example.c
om

; <<>> 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: 27115
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL:
 1

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

;; ANSWER SECTION:
www.example.com.        263     IN      CNAME   www.example.com-v4
.edgesuite.net.
www.example.com-v4.edgesuite.net. 19684 IN CNAME a1422.dscr.akamai
.net.
a1422.dscr.akamai.net.  20      IN      A       49.44.113.16
a1422.dscr.akamai.net.  20      IN      A       49.44.113.9

;; Query time: 35 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Oct 15 13:44:21 EDT 2025
;; MSG SIZE  rcvd: 154

PES1UG24CS840:VINAY~/.../packet_filter$
```

- Running `dig @8.8.8.8 www.example.com` returns a **valid DNS response**, proving no filtering is applied.
- The purpose of this test is not to block traffic but to **observe hook invocation**.
- The query resolves successfully, confirming that the module only logs, not drops packets.
- This sets the stage to correlate packet movement with hook messages in kernel logs.

```
[10/15/25]seed@VM:~/.../packet_filter$ sudo dmesg -k -w
[ 4453.637965] Registering filters.
[ 4473.470060] *** LOCAL_OUT
[ 4473.470064]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4473.470076] *** POST_ROUTING
[ 4473.470078]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4473.470090] *** PRE_ROUTING
[ 4473.470091]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4473.470093] *** LOCAL IN
```

- The log shows `Registering filters.` followed by hook messages for:
- `LOCAL_OUT` – packet generated locally (outgoing).
- `POST_ROUTING` – after routing decision.
- `PRE_ROUTING` – before routing decision (loopback traffic).
- `LOCAL_IN` – packet destined for local system.
- This confirms **all five major Netfilter hooks** are triggered at different packet processing stages

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

```
#obj-m += seedFilter.o
#obj-m += seedPrint.o
obj-m += seedBlock.o
all:
	make -C /lib/modules/$(shell uname -r)
les

clean:
	make -C /lib/modules/$(shell uname -r)
n
```

- The `Makefile` is updated to **uncomment `seedBlock.o`** and comment out other modules (`seedFilter.o`, `seedPrint.o`).
- This ensures only the blocking firewall module is compiled and inserted.
- This module contains logic to block **ICMP packets** (for ping) and **TCP port 23 traffic** (for Telnet).
- Proper Makefile setup is essential for Task 3, where inbound traffic is filtered at kernel level.

```
PES1UG24CS840:VINAY~/.../packet_filter$ sudo rmmod seedPrint
PES1UG24CS840:VINAY~/.../packet_filter$ vim Makefile
PES1UG24CS840:VINAY~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/L
absetup/volumes/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-gener
ic'
  CC [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedBl
ock.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedBl
ock.mod.o
  LD [M]  /home/seed/Desktop/Labsetup/volumes/packet_filter/seedBl
ock.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generi
c'
PES1UG24CS840:VINAY~/.../packet_filter$ sudo insmod seedBlock.ko
PES1UG24CS840:VINAY~/.../packet_filter$ lsmod | grep seedBlock
seedBlock              16384  0
PES1UG24CS840:VINAY~/.../packet_filter$
```

- The previous module (`seedPrint`) is removed using `sudo rmmod`.
- The `seedBlock` module is compiled with `make` and inserted using `sudo insmod seedBlock.ko`.
- `lsmod | grep seedBlock` confirms successful loading of the module.
- At this stage, the kernel is ready to **block ICMP echo requests** and **Telnet traffic** based on the programmed firewall rules.

```
[10/15/25]seed@VM:~/.../packet_filter$ sudo dmesg -k -w
[ 4453.637965] Registering filters.
[ 4473.470060] *** LOCAL_OUT
[ 4473.470064]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4473.470076] *** POST_ROUTING
[ 4473.470078]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4473.470090] *** PRE_ROUTING
[ 4473.470091]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 4473.470093] *** LOCAL_IN
[ 4473.470095]     127.0.0.1  --> 127.0.0.1 (UDP)
```

Terminal

- `sudo dmesg -k -w` is used to monitor real-time kernel logs.
- `Registering filters.` confirms the firewall hooks have been initialized.
- Hooks such as `LOCAL_OUT`, `POST_ROUTING`, `PRE_ROUTING`, and `LOCAL_IN` are triggered when traffic flows.
- When external hosts attempt to communicate, the firewall intercepts packets and **drops ICMP and Telnet packets** according to rules.

```
HOSTA:PES1UG24CS840:VINAY/
$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 ---
35 packets transmitted, 0 received, 100% packet loss, time 35483ms

HOSTA:PES1UG24CS840:VINAY/
$telnet 10.9.0.1
Trying 10.9.0.1...
```

- On **Host A (10.9.0.5)**, `ping 10.9.0.1` fails with **100% packet loss** — showing ICMP echo requests are dropped.
- `telnet 10.9.0.1` fails to connect (no response), confirming TCP port 23 traffic is blocked.
- This matches the firewall behavior coded in `seedBlock.o`.
- The blocking occurs at the **`PRE_ROUTING` hook**, which is appropriate for filtering inbound packets before they reach the local machine.

Task 2: Experimenting with Stateless Firewall Rules:

```
Seed-Attacker:PES1UG24CS840:VINAY/
$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
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -P OUTPUT DROP
Seed-Attacker:PES1UG24CS840:VINAY/
$ iptables -P INPUT DROP
Seed-Attacker:PES1UG24CS840:VINAY/
$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            icmp
type 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            icmp
type 0
```

- Initially, the router's `INPUT` and `OUTPUT` policies were set to `ACCEPT` for all traffic.

- Two rules were added:
- `-A INPUT -p icmp --icmp-type echo-request -j ACCEPT` → allows incoming ping requests.
- `-A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT` → allows outgoing ping replies.
- The default policies for `INPUT` and `OUTPUT` were then changed to `DROP`, which blocks all other traffic except the allowed ICMP types.
- Listing rules with `iptables -t filter -L -n` shows ICMP as allowed but other traffic implicitly denied.

```
HOSTA:PES1UG24CS840:VINAY/
$ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.199 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.119 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.332 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.181 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.174 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.197 ms
^C
--- 10.9.0.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5115ms
rtt min/avg/max/mdev = 0.119/0.200/0.332/0.064 ms
HOSTA:PES1UG24CS840:VINAY/
$telnet 10.9.0.11
Trying 10.9.0.11...
```

- `ping 10.9.0.11` from Host A succeeds — ICMP echo requests and replies are permitted due to the added iptables rules.
- `telnet 10.9.0.11` fails (hangs with no connection) because the `INPUT` and `OUTPUT` default policies are set to `DROP` and no rule allows Telnet (TCP port 23).
- This confirms selective access: **ICMP allowed**, **Telnet blocked**.

```
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -F
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -P OUTPUT ACCEPT
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -P INPUT ACCEPT
Seed-Attacker:PES1UG24CS840:VINAY/
$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
Seed-Attacker:PES1UG24CS840:VINAY/
```

- `iptables -F` clears all existing rules.
- `iptables -P OUTPUT ACCEPT` and `iptables -P INPUT ACCEPT` restore the default behavior, allowing all traffic again.
- Verifying with `iptables -t filter -L -n` shows both `INPUT` and `OUTPUT` policies reverted to `ACCEPT`.
- This cleanup is necessary before starting the next experiment.

Questions:

1. **Can you ping the router?**

   - When executing
   - `ping 10.9.0.11` the ping **succeeded** with 0% packet loss.
   - **Reason:**
   - The router's iptables rules explicitly allow:
     - Incoming ICMP echo requests
     - Outgoing ICMP echo replies
   - So ping traffic is permitted through the firewall.

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).**

   - When executing
   - telnet 10.9.0.11 the connection **failed** (hung), meaning no TCP connection was established.
   - **Reason:**
   - The default iptables policy for both INPUT and OUTPUT was set to DROP, and no rule was added to allow Telnet (TCP port 23).
   - Thus, the connection was blocked by the firewall.

Task 2.B: Protecting the Internal Network

```
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j D
ROP
Seed-Attacker:PES1UG24CS840:VINAY/
$ iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j
ACCEPT
Seed-Attacker:PES1UG24CS840:VINAY/
$iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACC
EPT
Seed-Attacker:PES1UG24CS840:VINAY/
$ iptables -P FORWARD DROP
Seed-Attacker:PES1UG24CS840:VINAY/
$ 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
```

- The `iptables` commands are applied on the **seed-router** to enforce ICMP traffic rules between internal and external networks.
- `-A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP` ensures external hosts cannot ping internal hosts.
- `-A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT` allows internal hosts to ping external hosts.
- `-A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT` allows router to respond to external pings.
- Finally, the **default policy** is set to `DROP` for the FORWARD chain, blocking all other packet types between networks.

```
HOSTA:PES1UG24CS840:VINAY/
$ 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 ---
44 packets transmitted, 0 received, 100% packet loss, time 44063ms

HOSTA:PES1UG24CS840:VINAY/
$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=0.283 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl
=64 time=0.166 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl
=64 time=0.205 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl
=64 time=0.328 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=5 ttl
=64 time=0.190 ms
^C
--- seed-router ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4055ms
rtt min/avg/max/mdev = 0.166/0.234/0.328/0.061 ms
HOSTA:PES1UG24CS840:VINAY/
$
```

- The outside host attempts to **ping an internal host (192.168.60.5)** but receives 100% packet loss.
- This confirms the firewall is **blocking incoming ICMP** to the internal network.
- Next, the outside host successfully pings the router (`seed-router`) — packets are received with no loss.
- This confirms that the firewall allows pings **to the router only**, as per the rule.
- The result matches the intended protection policy.

.

```
Host1:PES1UG24CS840:Vinay/
$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=0.159 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.251 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.214 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.207 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.222 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.199 ms
^C
--- 10.9.0.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5125ms
rtt min/avg/max/mdev = 0.159/0.208/0.251/0.027 ms
Host1:PES1UG24CS840:Vinay/
$telnet 10.9.0.5
Trying 10.9.0.5...
```

- The internal host successfully pings the outside host (10.9.0.5), showing 0% packet loss.
- This verifies that outbound ICMP traffic from internal to external is **allowed**.
- A `telnet` command to the outside host is attempted next.
- The connection hangs, proving **non-ICMP traffic is blocked** between networks.
- This confirms the final rule of the firewall: only ICMP ping (with specific direction) is permitted.

Task 2.C: Protecting Internal Servers

```
SEED-ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j
ACCEPT
SEED-ROUTER:PES1UG24CS840:VINAY/
$ iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j
 ACCEPT
SEED-ROUTER:PES1UG24CS840:VINAY/
$iptables -P FORWARD DROP
SEED-ROUTER:PES1UG24CS840:VINAY/
$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
SEED-ROUTER:PES1UG24CS840:VINAY/
$
```

- This shows the router firewall configuration allowing **TCP traffic on port 23** (Telnet) only to internal host `192.168.60.5`.
- Any Telnet request from external hosts (`eth0`) to other internal hosts is blocked because the default FORWARD policy is set to **DROP**.
- Return traffic from `192.168.60.5` (source port 23) through `eth1` is allowed.
- This effectively limits Telnet access from outside to only the approved server.
- These rules enforce **controlled access** and protect the rest of the internal network.

```
HOSTA:PES1UG24CS840:VINAY/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
df5053df564f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content
at are
not required on a system that users do not log into.
```

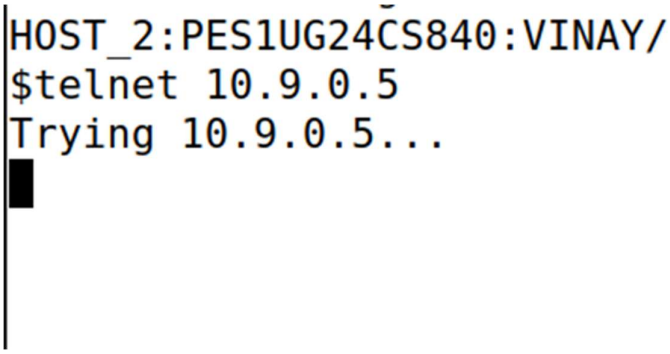
- The outside host (10.9.0.5) successfully connects via Telnet to `192.168.60.5`.
- The login prompt is displayed, confirming that the firewall allowed traffic on port 23.
- This proves that the **first rule (allow traffic to 192.168.60.5)** is working correctly.
- Successful access is limited only to the intended server.
- This validates objective 1 of the task.

```
HOSTA:PES1UG24CS840:VINAY/
$telnet 192.168.60.6
Trying 192.168.60.6...
^C
HOSTA:PES1UG24CS840:VINAY/
$telnet 192.168.60.7
Trying 192.168.60.7...
^C
HOSTA:PES1UG24CS840:VINAY/
$
```

- The outside host attempts Telnet connections to `192.168.60.6` and `192.168.60.7`.
- Both attempts hang with no response, proving that the firewall blocks traffic to these hosts.
- This confirms **no Telnet access to any host other than 192.168.60.5**.
- The default DROP policy ensures no accidental traffic passes through.
- This validates objective 2 of the task.

```
HOST_2:PES1UG24CS840:VINAY/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
df5053df564f login: ^CConnection closed by foreign host.
HOST_2:PES1UG24CS840:VINAY/
$telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4965bf45db6f login: ^Z^CConnection closed by foreign host.
HOST_2:PES1UG24CS840:VINAY/
$
```

- An internal host successfully connects to internal servers (192.168.60.5 and 192.168.60.7) via Telnet.
- Since both hosts are within the internal network, traffic bypasses the external filtering rules.
- The connection is established and then closed normally by the remote host.
- This confirms **internal hosts can communicate with each other** freely.
- This validates objective 3 of the task.



- The internal host tries to Telnet to `10.9.0.5` (external host).
- The connection does not succeed, showing the firewall blocked outgoing Telnet traffic.
- This confirms internal hosts **cannot access external servers**.
- The FORWARD chain DROP policy enforces this restriction.
- This validates objective 4 of the task.

Task 3: Connection Tracking and Stateful Firewall

```
HOSTA:PES1UG24CS840:VINAY/
$ 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=0.126 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.075 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.078 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.095 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.070 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.084 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.103 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.111 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.075 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.062 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.064 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.063 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.129 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.063 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.107 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.063 ms
```

- The outside host (`10.9.0.5`) is pinging the internal host (`192.168.60.5`).
- Continuous ICMP echo request and reply packets are successfully exchanged.
- This establishes an **ICMP connection entry** in the conntrack table on the router.
- Since ICMP doesn't have a session like TCP, conntrack simulates a "connection" state with a **timeout timer**.
- When the ping stops, the conntrack entry remains briefly before expiring automatically.

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 sr
c=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
icmp     1 2 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 src
=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
icmp     1 1 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 src
=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
icmp     1 0 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 src
=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
```

- The `conntrack -L` command on the router shows a single ICMP entry.
- It contains **source (10.9.0.5)**, **destination (192.168.60.5)**, and timer values.
- Repeated execution of the command shows the **timer decreasing**, indicating how long the entry will stay active.
- This proves conntrack maintains a temporary state for ICMP.
- Once the timer reaches zero, the ICMP entry is **automatically removed**.

```
Host1:PES1UG24CS840:Vinay/
$ nc -lu 9090

hello
```

- The internal host (`192.168.60.5`) runs a UDP server using netcat (`nc -lu 9090`).
- This makes the host listen for UDP datagrams on port 9090.
- UDP, unlike TCP, is connectionless — no handshake occurs.
- It waits passively for messages from a client.
- This setup will be used to test conntrack's behavior for UDP traffic.

```
HOSTA:PES1UG24CS840:VINAY/
$nc -u 192.168.60.5 9090

hello
```

- The outside host (`10.9.0.5`) sends a message to the UDP server on `192.168.60.5:9090`.
- The message `hello` is successfully received by the server.
- This generates a **UDP entry in the conntrack table** with `UNREPLIED` state (because no response yet).
- UDP flows are tracked based on source/destination IP and port.
- Conntrack timer starts counting down after the first packet is seen

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
udp       17 20 src=10.9.0.5 dst=192.168.60.5 sport=42698 dport=909
0 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=42698
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
udp       17 18 src=10.9.0.5 dst=192.168.60.5 sport=42698 dport=909
0 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=42698
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
udp       17 17 src=10.9.0.5 dst=192.168.60.5 sport=42698 dport=909
0 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=42698
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
udp       17 16 src=10.9.0.5 dst=192.168.60.5 sport=42698 dport=909
0 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=42698
 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown
.
SEED_ROUTER:PES1UG24CS840:VINAY/
$conntrack -L
```

- The router shows a UDP entry with `UNREPLIED` status initially.
- It records source/destination IPs and ports (client's ephemeral port and server port 9090).
- The timer can be observed decreasing over time, similar to ICMP.
- If the server replied, the entry state would change from `UNREPLIED` to `ASSURED`.
- Once the timer reaches zero, the entry is **cleared automatically**

Task 3.B: Setting Up a Stateful Firewall

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn
 -m conntrack
iptables v1.8.4 (legacy): conntrack: At least one option is required
Try `iptables -h' or 'iptables --help' for more information.
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn
 -m conntrack --ctstate NEW -j ACCEPT
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW
 -j ACCEPT
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISH
ED -j ACCEPT
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -p tcp -j DROP
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -P FORWARD ACCEPT
SEED_ROUTER:PES1UG24CS840:VINAY/
$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 12 packets, 1008 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
```

- Firewall rules are added using the `iptables` **conntrack** module to enable state tracking.
- A rule allows new TCP connections to **192.168.60.5** on port **23 (Telnet)** from external hosts.
- Another rule allows packets that are part of an **existing or related** connection.
- A `DROP` rule blocks all other TCP packets that are not part of allowed connections.
- The final list of rules confirms that only specific and tracked connections are permitted through the router.

```
HOSTA:PES1UG24CS840:VINAY/
$telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
```

- The external host (10.9.0.5) is able to connect to **192.168.60.5** through Telnet.
- This shows that the firewall correctly allows access only to the specified internal system.
- The rule for new connections on port 23 works as expected.
- It proves that the firewall is functioning in a **stateful manner**, monitoring the connection state.

```
HOSTA:PES1UG24CS840:VINAY/
$telnet 192.168.60.6
Trying 192.168.60.6...
^C
HOSTA:PES1UG24CS840:VINAY/
$telnet 192.168.60.7
Trying 192.168.60.7...
^C
HOSTA:PES1UG24CS840:VINAY/
$
```

- The external host tries to connect to **192.168.60.6** and **192.168.60.7**, but both connections fail.
- This happens because there are no rules allowing these systems to accept external Telnet traffic.
- The `DROP` rule successfully blocks unauthorized access.
- This confirms that only one internal system is reachable from outside the network.

```
HOST_2:PES1UG24CS840:VINAY/
$ telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
```

- The internal host (192.168.60.6) successfully connects to **192.168.60.5**.
- Internal communication within the same LAN is allowed by the firewall.
- The rule for new internal connections on interface **eth1** allows this communication.
- This verifies that internal systems can freely connect with each other.

```
HOST_2:PES1UG24CS840:VINAY/
$ telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
```

- The internal host also connects successfully to **192.168.60.7**.
- This shows that there are no restrictions for internal-to-internal traffic.
- The firewall accepts new and established connections between internal systems.
- It confirms that local communication inside the LAN is fully functional.

```
HOST_2:PES1UG24CS840:VINAY/
$telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
```

- The internal host (192.168.60.6) is able to connect to the external system (10.9.0.5).
- This proves that **outgoing connections** from the internal network are allowed.
- The **RELATED,ESTABLISHED** rule permits the response packets to return to the sender.
- It shows that the firewall maintains proper two-way communication using connection tracking.

Task 4: Limiting Network Traffic

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limi
t-burst 5 -j ACCEPT
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -A FORWARD -s 10.9.0.5 -j DROP
SEED_ROUTER:PES1UG24CS840:VINAY/
$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
SEED_ROUTER:PES1UG24CS840:VINAY/
$
```

- The `iptables` rule with `-m limit` is used to restrict packets from source **10.9.0.5** to only **10 packets per minute** with a burst of **5 packets**.
- The second rule with `DROP` ensures that any extra packets beyond this limit are blocked.
- These two rules together prevent excessive traffic from a single source, which helps avoid flooding or DoS attacks.
- The output shows both `ACCEPT` (limited rate) and `DROP` entries in the FORWARD chain.

```
HOSTA:PES1UG24CS840:VINAY/
$ 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=0.358 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.263 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.188 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.407 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.227 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.337 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.233 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.215 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.357 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.217 ms
64 bytes from 192.168.60.5: icmp_seq=37 ttl=63 time=0.213 ms
64 bytes from 192.168.60.5: icmp_seq=43 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=48 ttl=63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=54 ttl=63 time=0.193 ms
64 bytes from 192.168.60.5: icmp_seq=60 ttl=63 time=0.316 ms
64 bytes from 192.168.60.5: icmp_seq=66 ttl=63 time=0.422 ms
64 bytes from 192.168.60.5: icmp_seq=72 ttl=63 time=0.204 ms
64 bytes from 192.168.60.5: icmp_seq=78 ttl=63 time=0.207 ms
^C
--- 192.168.60.5 ping statistics ---
83 packets transmitted, 18 received, 78.3133% packet loss, time 83
975ms
rtt min/avg/max/mdev = 0.106/0.253/0.422/0.089 ms
```

- The ping command from **10.9.0.5** to **192.168.60.5** shows heavy **packet loss (~78%)**.
- This happens because only a few packets are allowed due to the limit rule, and the rest are dropped.
- It confirms that the rate-limiting mechanism is working correctly.
- The packet loss clearly shows that the firewall is enforcing the limit of 10 packets per minute.

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$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
   18  1512 ACCEPT     all  --  *      *       10.9.0.5
  0.0.0.0/0            limit: avg 10/min burst 5
   65  5460 DROP       all  --  *      *       10.9.0.5
  0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       10.9.0.5
  0.0.0.0/0            limit: avg 10/min burst 5

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
  destination
SEED_ROUTER:PES1UG24CS840:VINAY/
```

- The firewall table now shows only the limit rule without the second drop rule.
- Packets from **10.9.0.5** are accepted according to the rate limit, and no explicit drop rule is added.
- The absence of a drop rule means that once the limit is exceeded, extra packets are not handled explicitly.
- This setup is less secure because the firewall does not block excess packets clearly.

```
HOSTA:PES1UG24CS840:VINAY/
$ 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=0.112 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.213 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.219 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.216 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.263 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.210 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.211 ms
```

- The ping command from **10.9.0.5** to **192.168.60.5** runs smoothly with no visible packet loss.
- This happens because, without the drop rule, packets beyond the limit are not blocked.
- As a result, all ICMP packets are accepted, ignoring the traffic control rule.
- This confirms that the second drop rule is necessary to enforce packet rate limiting properly.

Task 5: Load Balancing

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mo
de nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:80
80
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mo
de nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:80
80
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mo
de nth --every 1 --packet 0 -j DNAT --to-destination 192.168.60.7:80
80
SEED_ROUTER:PES1UG24CS840:VINAY/
$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 4 packets, 212 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
SEED_ROUTER:PES1UG24CS840:VINAY/
$
```

- Three rules are added using the **statistic module** in `iptables` with **nth mode** to divide packets equally between three internal UDP servers.
- Packets received on port **8080** are distributed in a **round-robin** manner:
  - Every 3rd packet goes to **192.168.60.5:8080**,
  - Every 2nd packet goes to **192.168.60.6:8080**,

- Every packet goes to **192.168.60.7:8080**.

- This setup ensures that traffic is evenly balanced between all three servers.
- The command `iptables -L -n -v` confirms that the NAT table and PREROUTING rules were successfully applied.

```
HOSTA:PES1UG24CS840:VINAY/
$nc -u 10.9.0.11 8080
hello 1
hello 2
hello 3
```

- The external host **(10.9.0.11)** sends three UDP packets to port **8080** using `nc -u`.
- The messages sent are “hello 1”, “hello 2”, and “hello 3”.
- These messages are distributed across the three internal servers according to the round-robin load balancing rules.
- This verifies that the router is successfully forwarding packets to different servers based on the defined rule sequence.

```
Host1:PES1UG24CS840:Vinay/
$nc -luk 8080
hello 1
```

- Server 1, running on **192.168.60.5**, receives the first packet **“hello 1”**.
- This confirms that the first UDP datagram was redirected to this host as per the nth-mode configuration.
- The use of `nc -luk 8080` keeps the server running to receive multiple UDP messages.
- The load balancing rule successfully routes the initial packet here.

```
HOST_2:PES1UG24CS840:VINAY/
$ nc -luk 8080
hello 2
```

- Server 2, on **192.168.60.6**, receives the message **“hello 2”**.
- This shows that the second packet was directed to the second server based on the nth counting logic.
- The configuration ensures even packet distribution without manual intervention.
- The result demonstrates that the router handles UDP packet redirection efficiently.

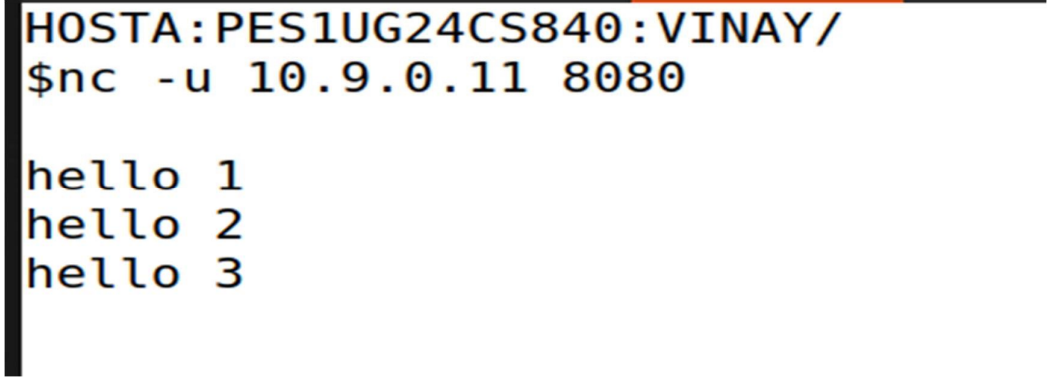
```
host_3:PES1UG24CS840:Vinay/
$nc -luk 8080
hello 3
```

- Server 3, on **192.168.60.7**, receives the message **"hello 3"**.
- This confirms that the third UDP packet was routed to the third server as part of the load-balancing setup.
- Each internal server gets one packet in a round-robin manner, proving balanced distribution.
- This behavior is achieved without using any external load balancer, relying purely on iptables.

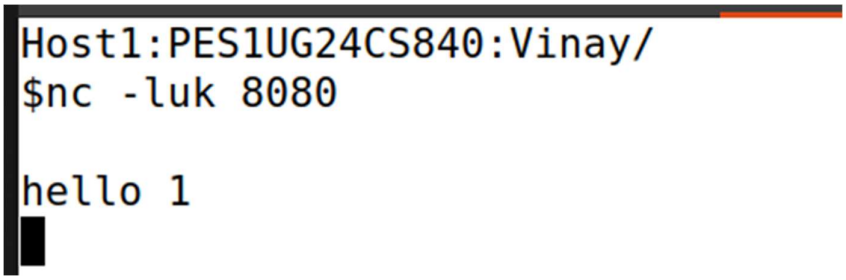
Using another mode:

```
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mo
de random --probability 0.3333 -j DNAT --to-destination 192.168.60.5
:8080
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mo
de random --probability 0.5 -j DNAT --to-destination 192.168.60.6:80
80
SEED_ROUTER:PES1UG24CS840:VINAY/
$iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mo
de random --probability 1 -j DNAT --to-destination 192.168.60.7:8080
SEED_ROUTER:PES1UG24CS840:VINAY/
$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
SEED_ROUTER:PES1UG24CS840:VINAY/
$
```

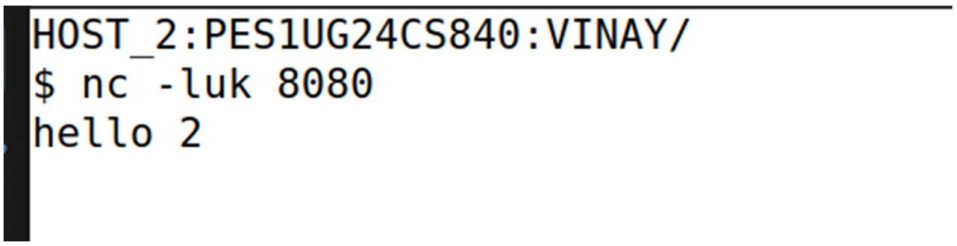
- The `iptables` rules use the **statistic module** in **random mode** to distribute incoming UDP traffic on port **8080**.
- Each packet is assigned to one of the three servers based on a given **probability value**.
- The probabilities 0.3333, 0.5, and 1 are used for **192.168.60.5**, **192.168.60.6**, and **192.168.60.7** respectively.
- The `DNAT` target redirects traffic from the external host to one of these internal servers randomly.
- The output confirms that the NAT rules are successfully added for the PREROUTING chain.



- The external host (**10.9.0.11**) sends three messages ("hello 1", "hello 2", and "hello 3") using the `nc -u` command on port **8080**.
- Each message is treated as a separate UDP packet and forwarded randomly to one of the internal servers.
- The randomness means that messages will not always follow a fixed pattern.
- The router decides dynamically which internal server receives each packet based on probability.



- The first internal server (**192.168.60.5**) receives the message **"hello 1"**.
- This shows that the first UDP packet was forwarded to this server by chance according to the random probability rule.
- The `nc -luk 8080` command keeps the UDP listener active to receive data from multiple sources.
- The message confirms that the server successfully received one of the random packets.



- The second server (**192.168.60.6**) receives the message **"hello 2"**.
- This shows that the next UDP packet was routed to a different host.
- The random probability setting ensures that packets are distributed unpredictably among the servers.
- This helps in spreading network load even if the packet arrival pattern changes.

```
host_3:PES1UG24CS840:Vinay/
$nc -luk 8080
hello 3
```

- The third server (**192.168.60.7**) receives the message **"hello 3"**.
- This confirms that the third message was delivered to the third server based on random selection.
- Unlike nth mode, where packets are divided sequentially, here the selection is purely probabilistic.
- It demonstrates how random mode helps in **non-deterministic load balancing**.

Code Snippets:

Kernel Module Code (hello.c):

```c
#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");
```

- Simple kernel module that prints messages when loaded and removed.
- initialization() → runs on module insert (insmod), prints "Hello World!".
- cleanup() → runs on module remove (rmmod), prints "Bye-bye World!".
- printk() → used instead of printf() for kernel logging.
- module_init() and module_exit() register load and unload functions.
- MODULE_LICENSE("GPL") avoids kernel warnings.

Makefile:

```
obj-m += hello.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- `obj-m += hello.o` → builds `hello.ko` module.
- `make -C /lib/modules/$(uname -r)/build M=$(PWD) modules` → compiles the module using current kernel headers.
- `clean:` removes build files.

SeedBlock.c

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


static struct nf_hook_ops hook1, hook2, hook3, hook4;

// blocking ping to vm - 10.9.0.1
unsigned int blockICMP(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct icmphdr *icmph;

   //u16  port   = 53; // DNS
   char ip[16] = "10.9.0.1";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_ICMP) {
       icmph = icmp_hdr(skb);
       if (iph->daddr == ip_addr && icmph->type == ICMP_ECHO){
```

```
            printk(KERN_WARNING "*** Dropping %pI4 (ICMP) \n", &(iph->daddr));
            return NF_DROP;
        }
    }
    return NF_ACCEPT;
}

// blocking udp to 8.8.8.8 : 53
unsigned int blockUDP(void *priv, struct sk_buff *skb,
                      const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct udphdr *udph;

    u16  port   = 53; // DNS
    char ip[16] = "8.8.8.8";
    u32  ip_addr;

    if (!skb) return NF_ACCEPT;

    iph = ip_hdr(skb);
    // Convert the IPv4 address from dotted decimal to 32-bit binary
    in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

    if (iph->protocol == IPPROTO_UDP) {
        udph = udp_hdr(skb);
        if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %d\n", &(iph-
>daddr), port);
            return NF_DROP;
        }
    }
    return NF_ACCEPT;
}

// blocking telnet to 10.9.0.1 : 23
unsigned int blockTelnet(void *priv, struct sk_buff *skb,
                      const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    u16  port   = 23; // DNS
    char ip[16] = "10.9.0.1";
    u32  ip_addr;

    if (!skb) return NF_ACCEPT;

    iph = ip_hdr(skb);
    // Convert the IPv4 address from dotted decimal to 32-bit binary
    in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

    if (iph->protocol == IPPROTO_TCP) {
        tcph = tcp_hdr(skb);
        if (iph->daddr == ip_addr && ntohs(tcph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (TCP), port %d\n", &(iph-
>daddr), port);
            return NF_DROP;
        }
```

```
    }
    return NF_ACCEPT;
}


unsigned int printInfo(void *priv, struct sk_buff *skb,
                const struct nf_hook_state *state)
{
    struct iphdr *iph;
    char *hook;
    char *protocol;

    switch (state->hook){
      case NF_INET_LOCAL_IN:     hook = "LOCAL_IN";     break;
      case NF_INET_LOCAL_OUT:    hook = "LOCAL_OUT";    break;
      case NF_INET_PRE_ROUTING:  hook = "PRE_ROUTING";  break;
      case NF_INET_POST_ROUTING: hook = "POST_ROUTING"; break;
      case NF_INET_FORWARD:      hook = "FORWARD";      break;
      default:                   hook = "IMPOSSIBLE";   break;
    }
    printk(KERN_INFO "*** %s\n", hook); // Print out the hook info

    iph = ip_hdr(skb);
    switch (iph->protocol){
      case IPPROTO_UDP:  protocol = "UDP";   break;
      case IPPROTO_TCP:  protocol = "TCP";   break;
      case IPPROTO_ICMP: protocol = "ICMP";  break;
      default:           protocol = "OTHER"; break;

    }
    // Print out the IP addresses and protocol
    printk(KERN_INFO "    %pI4  --> %pI4 (%s)\n",
                    &(iph->saddr), &(iph->daddr), protocol);

    return NF_ACCEPT;
}


int registerFilter(void) {
    printk(KERN_INFO "Registering filters.\n");

    hook1.hook = printInfo;
    hook1.hooknum = NF_INET_LOCAL_OUT;
    hook1.pf = PF_INET;
    hook1.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook1);

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

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

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

    return 0;
}

void removeFilter(void) {
    printk(KERN_INFO "The filters are being removed.\n");
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

- This program is a **Netfilter-based firewall kernel module**.
- It registers multiple hooks (`hook1`–`hook4`) to inspect and drop specific packets before they reach the network layer.

Main functions

1. **blockICMP()** – Blocks ping (ICMP echo) requests sent to IP `10.9.0.1`.
2. **blockUDP()** – Blocks UDP packets sent to Google DNS (`8.8.8.8`) on port `53`.
3. **blockTelnet()** – Blocks Telnet (TCP port `23`) packets to `10.9.0.1`.
4. **printInfo()** – Prints packet details (source IP, destination IP, and protocol).

How it works

- The module hooks into Netfilter chains like **PRE_ROUTING**, **POST_ROUTING**, and **LOCAL_OUT**.
- Each hook checks the packet type and destination; if it matches a blocked rule, it returns `NF_DROP`.
- Otherwise, it returns `NF_ACCEPT`.
- On `insmod`, hooks are registered; on `rmmod`, they are unregistered.

Makefile:

```
obj-m += seedBlock.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

ins:
    sudo dmesg -C
    sudo insmod seedBlock.ko

rm:
    sudo rmmod seedBlock
```

- **obj-m** – Builds the kernel module `seedBlock.ko`.
- **all** – Compiles the module using the system's kernel headers.
- **clean** – Removes all build files.
- **ins / rm** – Handy commands to insert or remove the module, clearing kernel logs before/after testing.

SeedFilter.c

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


static struct nf_hook_ops hook1, hook2;


unsigned int blockUDP(void *priv, struct sk_buff *skb,
                      const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct udphdr *udph;

   u16  port   = 53; // DNS
   char ip[16] = "8.8.8.8";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;
```

```
   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_UDP) {
       udph = udp_hdr(skb);
       if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %d\n", &(iph-
>daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}

unsigned int printInfo(void *priv, struct sk_buff *skb,
                const struct nf_hook_state *state)
{
   struct iphdr *iph;
   char *hook;
   char *protocol;

   switch (state->hook){
     case NF_INET_LOCAL_IN:     hook = "LOCAL_IN";     break;
     case NF_INET_LOCAL_OUT:    hook = "LOCAL_OUT";    break;
     case NF_INET_PRE_ROUTING:  hook = "PRE_ROUTING";  break;
     case NF_INET_POST_ROUTING: hook = "POST_ROUTING"; break;
     case NF_INET_FORWARD:      hook = "FORWARD";      break;
     default:                   hook = "IMPOSSIBLE";   break;
   }
   printk(KERN_INFO "*** %s\n", hook); // Print out the hook info

   iph = ip_hdr(skb);
   switch (iph->protocol){
     case IPPROTO_UDP:  protocol = "UDP";   break;
     case IPPROTO_TCP:  protocol = "TCP";   break;
     case IPPROTO_ICMP: protocol = "ICMP";  break;
     default:           protocol = "OTHER"; break;

   }
   // Print out the IP addresses and protocol
   printk(KERN_INFO "    %pI4  --> %pI4 (%s)\n",
                  &(iph->saddr), &(iph->daddr), protocol);

   return NF_ACCEPT;
}


int registerFilter(void) {
   printk(KERN_INFO "Registering filters.\n");

   hook1.hook = printInfo;
   hook1.hooknum = NF_INET_LOCAL_OUT;
   hook1.pf = PF_INET;
   hook1.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook1);

   hook2.hook = blockUDP;
```

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

    return 0;
}

void removeFilter(void) {
    printk(KERN_INFO "The filters are being removed.\n");
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

- The **seedFilter.c** kernel module uses **Netfilter hooks** to monitor and control network packets in the Linux kernel.
- It registers two hooks — one (`printInfo`) to display packet details such as source IP, destination IP, and protocol, and another (`blockUDP`) to block specific traffic.
  The `blockUDP()` function drops all UDP packets destined for **8.8.8.8** on port **53 (DNS)** by returning `NF_DROP`.
- All other packets are accepted using `NF_ACCEPT`.
- The module prints network flow information using `printk()` for debugging and analysis.
  When loaded (`insmod`), it registers both hooks, and when removed (`rmmod`), it unregisters them.
- This module demonstrates a basic **kernel-level firewall** that filters outgoing packets using Netfilter's **POST_ROUTING** and **LOCAL_OUT** hooks.

SeedPrint.c

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


static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5;


unsigned int blockUDP(void *priv, struct sk_buff *skb,
                      const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct udphdr *udph;
```

```
   u16  port   = 53; // DNS
   char ip[16] = "8.8.8.8";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_UDP) {
       udph = udp_hdr(skb);
       if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %d\n", &(iph-
>daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}

unsigned int printInfo(void *priv, struct sk_buff *skb,
                const struct nf_hook_state *state)
{
   struct iphdr *iph;
   char *hook;
   char *protocol;

   switch (state->hook){
     case NF_INET_LOCAL_IN:     hook = "LOCAL_IN";     break;
     case NF_INET_LOCAL_OUT:    hook = "LOCAL_OUT";    break;
     case NF_INET_PRE_ROUTING:  hook = "PRE_ROUTING";  break;
     case NF_INET_POST_ROUTING: hook = "POST_ROUTING"; break;
     case NF_INET_FORWARD:      hook = "FORWARD";      break;
     default:                   hook = "IMPOSSIBLE";   break;
   }
   printk(KERN_INFO "*** %s\n", hook); // Print out the hook info

   iph = ip_hdr(skb);
   switch (iph->protocol){
     case IPPROTO_UDP:  protocol = "UDP";   break;
     case IPPROTO_TCP:  protocol = "TCP";   break;
     case IPPROTO_ICMP: protocol = "ICMP";  break;
     default:           protocol = "OTHER"; break;

   }
   // Print out the IP addresses and protocol
   printk(KERN_INFO "    %pI4  --> %pI4 (%s)\n",
                  &(iph->saddr), &(iph->daddr), protocol);

   return NF_ACCEPT;
}


int registerFilter(void) {
   printk(KERN_INFO "Registering filters.\n");

// NF_INET_PRE_ROUTING
   hook1.hook = printInfo;
```

```
    hook1.hooknum = NF_INET_PRE_ROUTING;
    hook1.pf = PF_INET;
    hook1.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook1);

// NF_INET_LOCAL_IN
    hook2.hook = printInfo;
    hook2.hooknum = NF_INET_LOCAL_IN;
    hook2.pf = PF_INET;
    hook2.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook2);

// NF_INET_FORWARD
    hook3.hook = printInfo;
    hook3.hooknum = NF_INET_FORWARD;
    hook3.pf = PF_INET;
    hook3.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook3);

// NF_INET_LOCAL_OUT
    hook4.hook = printInfo;
    hook4.hooknum = NF_INET_LOCAL_OUT;
    hook4.pf = PF_INET;
    hook4.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook4);

// NF_INET_POST_ROUTING
    hook5.hook = printInfo;
    hook5.hooknum = NF_INET_POST_ROUTING;
    hook5.pf = PF_INET;
    hook5.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook5);

    return 0;
}

void removeFilter(void) {
    printk(KERN_INFO "The filters are being removed.\n");
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
    nf_unregister_net_hook(&init_net, &hook5);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

- This kernel module demonstrates how to use **Netfilter hooks** to monitor all five stages of packet flow and block specific traffic.
- It defines five hooks (`hook1`–`hook5`) for the chains **PRE_ROUTING**, **LOCAL_IN**, **FORWARD**, **LOCAL_OUT**, and **POST_ROUTING**.
- Each hook runs the `printInfo()` function, which prints details like the hook name, source IP, destination IP, and protocol type (TCP, UDP, ICMP).
- Additionally, the `blockUDP()` function checks for UDP packets destined to **8.8.8.8:53** and drops them by returning `NF_DROP`.

- All other packets are accepted using `NF_ACCEPT`.
- When the module is inserted, it registers all hooks; when removed, it unregisters them.

  Overall, it acts as a **packet logger and filter**, helping to trace how packets move through different Netfilter chains while blocking DNS traffic to Google's DNS server.