



---

# Firewall Exploration

---



March 2024  
Abdulfatah Abdillahi

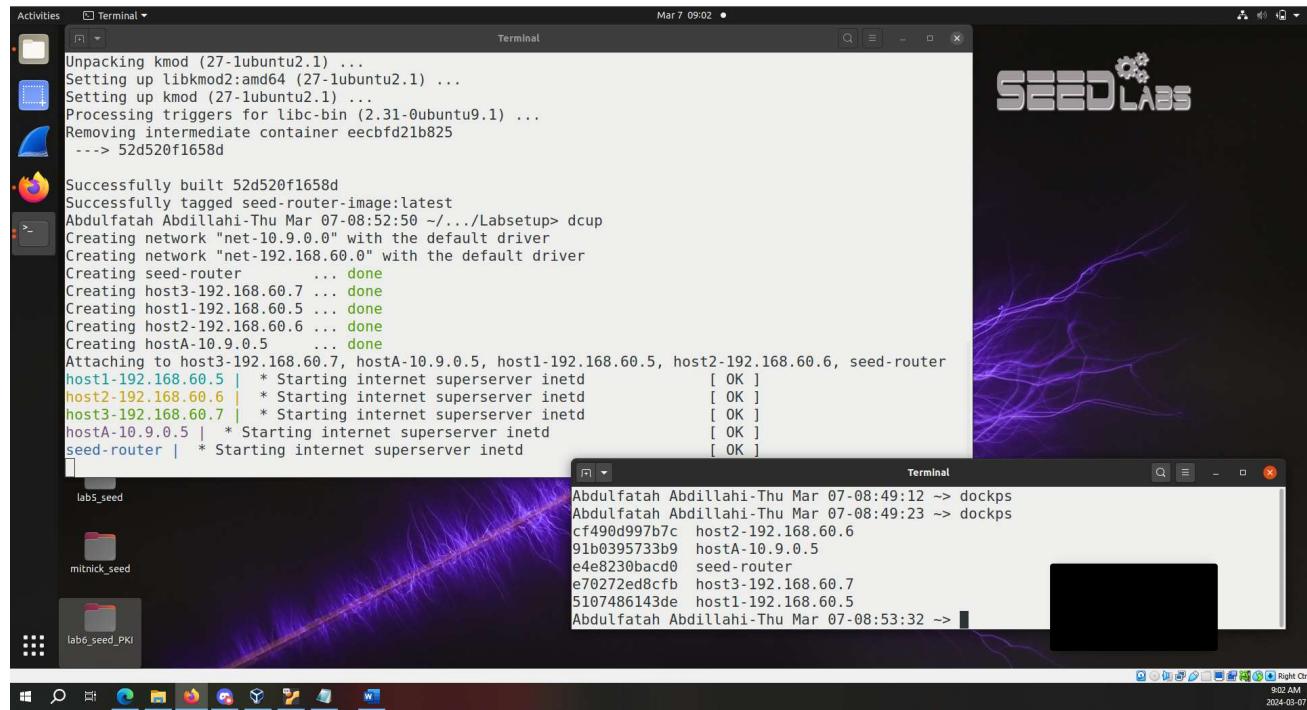
## Contents

SEED: Firewall Lab Environment .....	3
Task 1: Implementing a Simple Firewall.....	5
Task 2: Experimenting with Stateless Firewall Rules .....	19
Task 3: Connection Tracking and Stateful Firewall .....	25
Task 5: Load Balancing .....	32
References .....	40

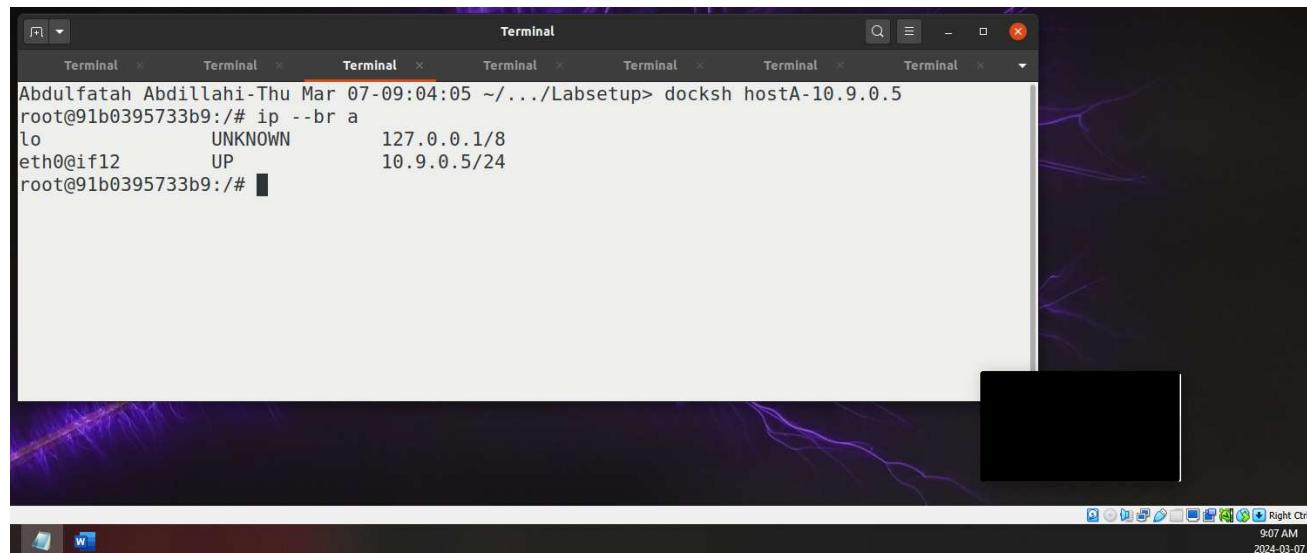
# SEED: Firewall Lab Environment

## Setting up the Environment

### Creating five docker containers



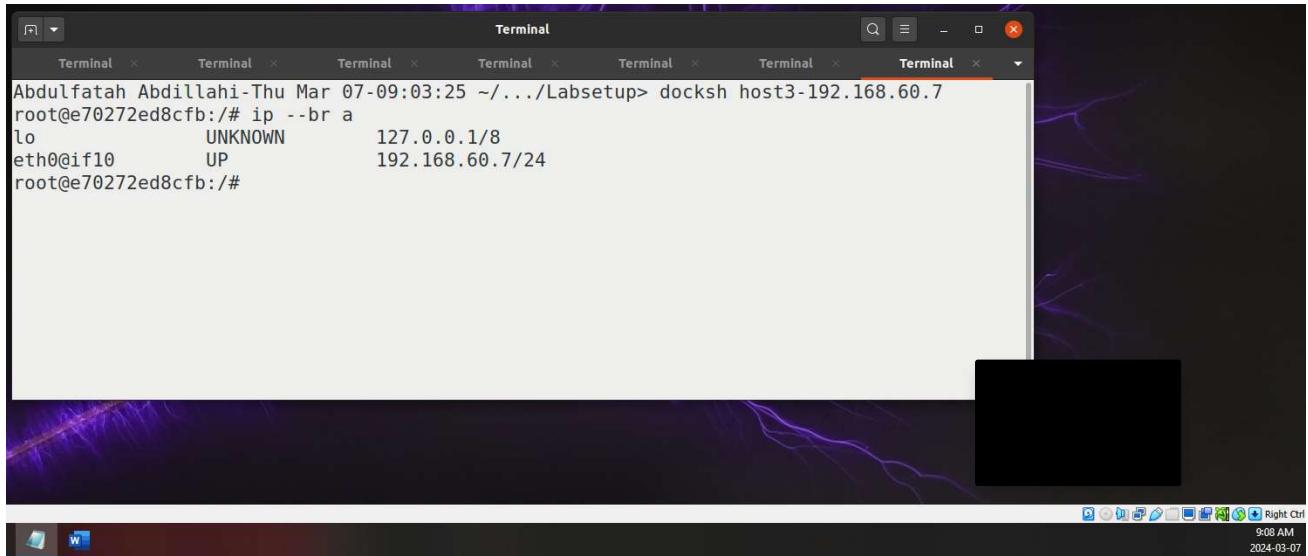
Checking the IP addresses for the container (host A, seed-router, host1, host2, and host3).



```
Terminal Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Thu Mar 07-09:04:56 ~/.../Labsetup> docksh seed-router
root@e4e8230bacd0:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if14   UP        10.9.0.11/24
eth1@if20   UP        192.168.60.11/24
root@e4e8230bacd0:/#
```

```
Terminal Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Thu Mar 07-09:03:24 ~/.../Labsetup> docksh host1-192.168.60.5
root@5107486143de:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if18   UP        192.168.60.5/24
root@5107486143de:/#
```

```
Terminal Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~/.../Labsetup> docksh host2-192.168.60.6
root@cf490d997b7c:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if16   UP        192.168.60.6/24
root@cf490d997b7c:/#
```



## Task 1: Implementing a Simple Firewall

### Task 1.A: Implement a Simple Kernel Module

Since this task is operating on the kernel level, I am not using the containers yet. I am instead doing this on the host.

Here, this is the structure of the provided kernel module *hello.c* as well as the *Makefile*.

```
hello.c
~/Desktop/Lab8_SEED_Firewall/LabSetup/Files/kernel_module
hello.c
Makefile
```

```
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3
4 int initialization(void)
5 {
6     printk(KERN_INFO "Hello World!\n");
7     return 0;
8 }
9
10 void cleanup(void)
11 {
12     printk(KERN_INFO "Bye-bye World!.\n");
13 }
14
15 module_init(initialization);
16 module_exit(cleanup);
17
18 MODULE_LICENSE("GPL");
```

The screenshot shows a code editor window with two tabs: "Makefile" and "hello.c". The "Makefile" tab contains the following content:

```
1 obj-m += hello.o
2
3 all:
4     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6 clean:
7     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
8
```

The "hello.c" tab is partially visible at the top. The status bar at the bottom right shows the date and time: "9:38 AM 2024-03-07".

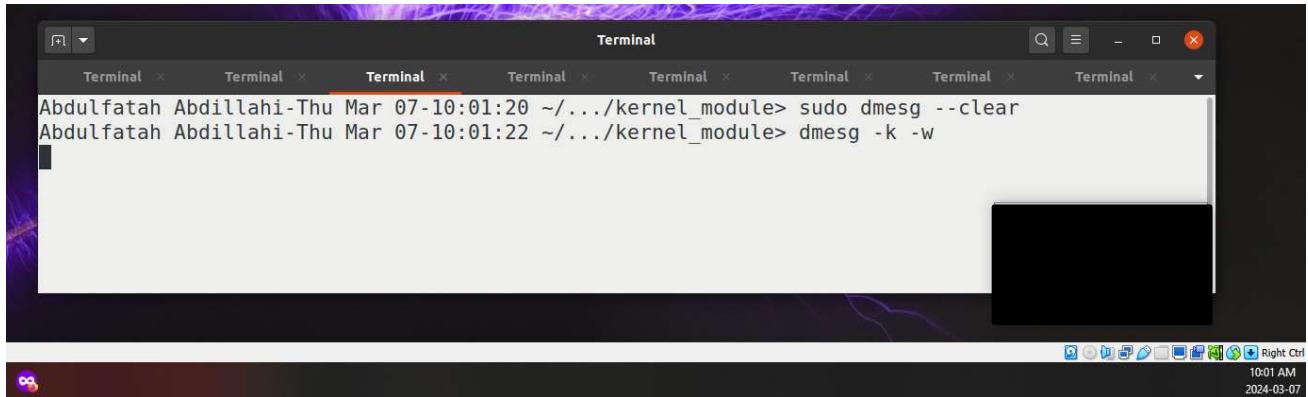
Here, you can see me running the `make` command to compile and build the kernel module `hello.ko`

The screenshot shows a terminal window with multiple tabs, all titled "Terminal". The terminal output is as follows:

```
Abdulfatah Abdillahi-Thu Mar 07-09:40:33 ~/.../Labsetup>
Abdulfatah Abdillahi-Thu Mar 07-09:40:33 ~/.../Labsetup> ls
docker-compose.yml  Files  router  volumes
Abdulfatah Abdillahi-Thu Mar 07-09:40:36 ~/.../Labsetup> cd Files/
Abdulfatah Abdillahi-Thu Mar 07-09:40:39 ~/.../Files> ks
^C
Abdulfatah Abdillahi-Thu Mar 07-09:40:41 ~/.../Files> ls
kernel_module  packet_filter
Abdulfatah Abdillahi-Thu Mar 07-09:40:42 ~/.../Files> cd kernel_module/
Abdulfatah Abdillahi-Thu Mar 07-09:40:58 ~/.../kernel_module> ls
hello.c  Makefile
Abdulfatah Abdillahi-Thu Mar 07-09:41:01 ~/.../kernel_module> make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
Abdulfatah Abdillahi-Thu Mar 07-09:41:42 ~/.../kernel_module> ls
hello.c  hello.ko  hello.mod  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
Abdulfatah Abdillahi-Thu Mar 07-09:43:31 ~/.../kernel_module>
```

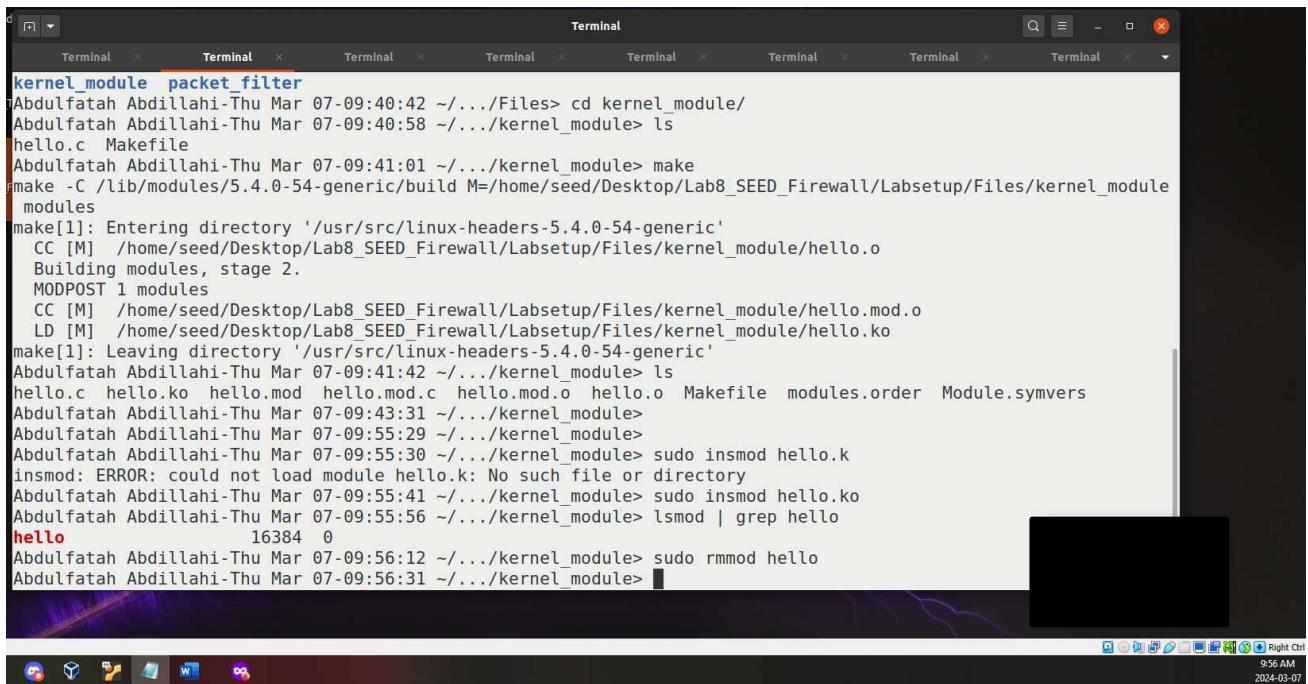
The status bar at the bottom right shows the date and time: "9:43 AM 2024-03-07".

Here, I'm opening a new terminal and clearing any previous messages using `dmesg --clear` and waiting for new kernel messages using `dmesg -k -w`.



```
Terminal Abdulfatah Abdillahi-Thu Mar 07-10:01:20 ~/.../kernel_module> sudo dmesg --clear  
Abdulfatah Abdillahi-Thu Mar 07-10:01:22 ~/.../kernel_module> dmesg -k -w
```

Then, I insert the hello module and make sure it is created under the modules list. If this is the case, I remove the module afterwards and check back on the other terminal to see if the messages were printed as expected. The messages “Hello World!” and “Bye-bye World!” were printed successfully.

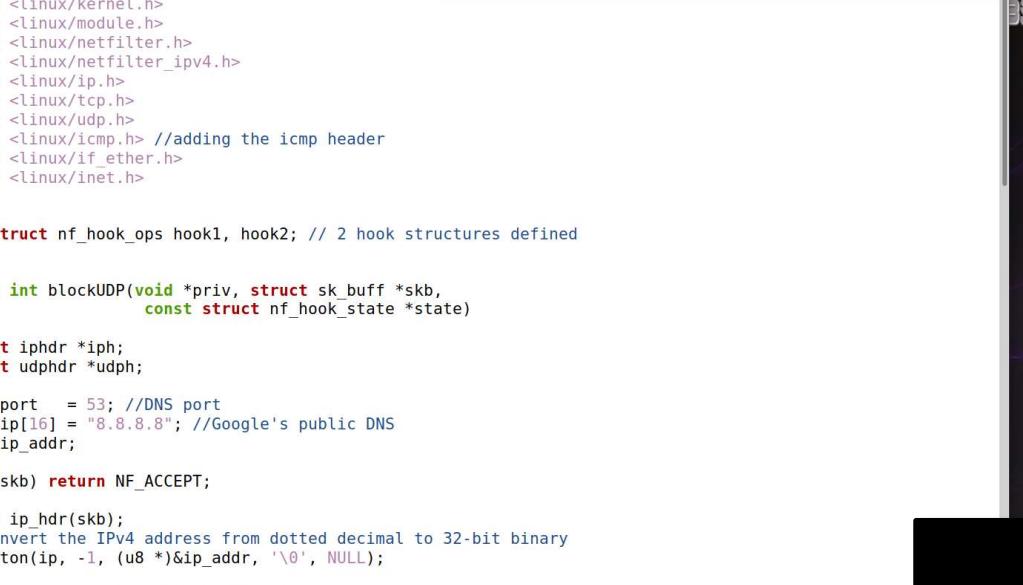


```
Terminal kernel_module packet_filter  
Abdulfatah Abdillahi-Thu Mar 07-09:40:42 ~/.../Files> cd kernel_module/  
Abdulfatah Abdillahi-Thu Mar 07-09:40:58 ~/.../kernel_module> ls  
hello.c Makefile  
Abdulfatah Abdillahi-Thu Mar 07-09:41:01 ~/.../kernel_module> make  
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module  
modules  
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'  
CC [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module/hello.o  
Building modules, stage 2.  
MODPOST 1 modules  
CC [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module/hello.mod.o  
LD [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/kernel_module/hello.ko  
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'  
Abdulfatah Abdillahi-Thu Mar 07-09:41:42 ~/.../kernel_module> ls  
hello.c hello.ko hello.mod hello.mod.c hello.mod.o hello.o Makefile modules.order Module.symvers  
Abdulfatah Abdillahi-Thu Mar 07-09:43:31 ~/.../kernel_module>  
Abdulfatah Abdillahi-Thu Mar 07-09:55:29 ~/.../kernel_module>  
Abdulfatah Abdillahi-Thu Mar 07-09:55:30 ~/.../kernel_module> sudo insmod hello.k  
insmod: ERROR: could not load module hello.k: No such file or directory  
Abdulfatah Abdillahi-Thu Mar 07-09:55:41 ~/.../kernel_module> sudo insmod hello.ko  
Abdulfatah Abdillahi-Thu Mar 07-09:55:56 ~/.../kernel_module> lsmod | grep hello  
hello 16384 0  
Abdulfatah Abdillahi-Thu Mar 07-09:56:12 ~/.../kernel_module> sudo rmmod hello  
Abdulfatah Abdillahi-Thu Mar 07-09:56:31 ~/.../kernel_module>
```

## **Task 1.B: Implement a Simple Firewall Using Netfilter**

1. Compile the sample 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.

Here, you can see the structure of the kernel module *seedFilter.c* as well as the *Makefile*.



```
#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> //adding the icmp header
#include <linux/if_ether.h>
#include <linux/inet.h>
11
12
13 static struct nf_hook_ops hook1, hook2; // 2 hook structures defined
14
15
16 unsigned int blockUDP(void *priv, struct sk_buff *skb,
17                       const struct nf_hook_state *state)
18 {
19     struct iphdr *iph;
20     struct udphdr *udph;
21
22     u16 port = 53; //DNS port
23     char ip[16] = "8.8.8.8"; //Google's public DNS
24     u32 ip_addr;
25
26     if (!skb) return NF_ACCEPT;
27
28     iph = ip_hdr(skb);
29     // Convert the IPv4 address from dotted decimal to 32-bit binary
30     in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
31
32     if (iph->protocol == IPPROTO_UDP) {
```

The screenshot shows a code editor window with two tabs: 'Makefile' and 'seedFilter.c'. The 'Makefile' tab contains the following content:

```
1 obj-m += seedFilter.o
2 all:
3     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
4
5 clean:
6     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
7
8 ins:
9     sudo dmesg -C
10    sudo insmod seedFilter.ko
11
12 rm:
13    sudo rmmod seedFilter
14
```

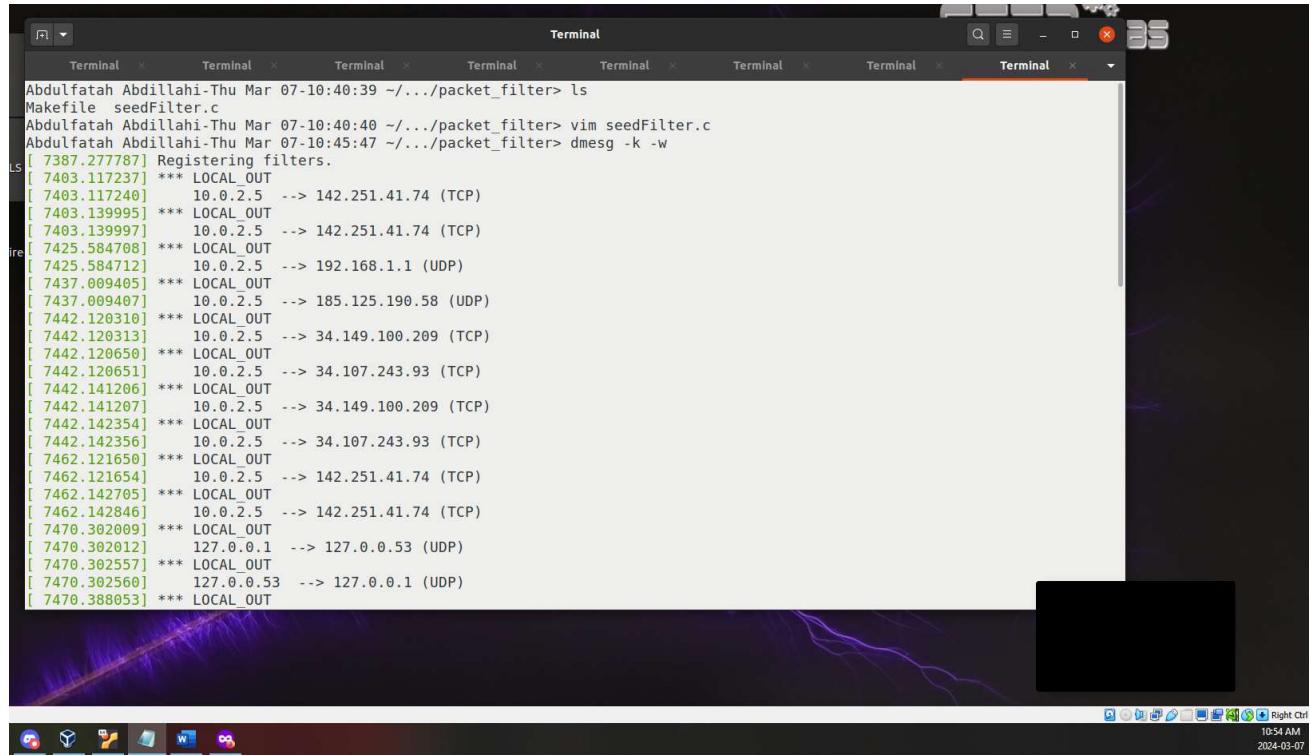
The 'seedFilter.c' tab is currently not visible.

Here, we generate our kernel module *seedFilter.ko* and insert it.

The screenshot shows a terminal window with multiple tabs, all titled 'Terminal'. The active tab displays the following command-line session:

```
Abdulfatah Abdillahi-Thu Mar 07-10:17:25 ~/.../Files> ls
kernel_module packet_filter
Abdulfatah Abdillahi-Thu Mar 07-10:17:25 ~/.../Files> cd packet_filter/
Abdulfatah Abdillahi-Thu Mar 07-10:17:29 ~/.../packet_filter> ls
Makefile seedFilter.c
Abdulfatah Abdillahi-Thu Mar 07-10:17:29 ~/.../packet_filter> make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
CC [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedFilter.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedFilter.mod.o
LD [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
Abdulfatah Abdillahi-Thu Mar 07-10:46:11 ~/.../packet_filter> ls
Makefile modules.order Module.symvers seedFilter.c seedFilter.ko seedFilter.mod seedFilter.mod.o
seedFilter.o seedFilter.o
Abdulfatah Abdillahi-Thu Mar 07-10:46:31 ~/.../packet_filter> sudo insmod seedFilter.ko
Abdulfatah Abdillahi-Thu Mar 07-10:50:27 ~/.../packet_filter> lsmod | grep seed
seedFilter      16384  0
Abdulfatah Abdillahi-Thu Mar 07-10:50:39 ~/.../packet_filter>
```

We can see this also being reflected from our kernel messages and print outs based on the source code (e.g. IP source, IP destination and protocol).



```
Abdulfatah Abdillahi-Thu Mar 07-10:40:39 ~/.../packet_filter> ls
Makefile seedFilter.c
Abdulfatah Abdillahi-Thu Mar 07-10:40:40 ~/.../packet_filter> vim seedFilter.c
Abdulfatah Abdillahi-Thu Mar 07-10:45:47 ~/.../packet_filter> dmesg -k -w
[ 7387.277787] Registering filters.
[ 7403.117237] *** LOCAL_OUT
[ 7403.117240] 10.0.2.5 --> 142.251.41.74 (TCP)
[ 7403.139995] *** LOCAL_OUT
[ 7403.139997] 10.0.2.5 --> 142.251.41.74 (TCP)
[ 7425.584708] *** LOCAL_OUT
[ 7425.584712] 10.0.2.5 --> 192.168.1.1 (UDP)
[ 7437.009405] *** LOCAL_OUT
[ 7437.009407] 10.0.2.5 --> 185.125.190.58 (UDP)
[ 7442.120310] *** LOCAL_OUT
[ 7442.120313] 10.0.2.5 --> 34.149.100.209 (TCP)
[ 7442.120650] *** LOCAL_OUT
[ 7442.120651] 10.0.2.5 --> 34.107.243.93 (TCP)
[ 7442.141206] *** LOCAL_OUT
[ 7442.141207] 10.0.2.5 --> 34.149.100.209 (TCP)
[ 7442.142354] *** LOCAL_OUT
[ 7442.142356] 10.0.2.5 --> 34.107.243.93 (TCP)
[ 7462.121650] *** LOCAL_OUT
[ 7462.121654] 10.0.2.5 --> 142.251.41.74 (TCP)
[ 7462.142705] *** LOCAL_OUT
[ 7462.142846] 10.0.2.5 --> 142.251.41.74 (TCP)
[ 7470.302009] *** LOCAL_OUT
[ 7470.302012] 127.0.0.1 --> 127.0.0.53 (UDP)
[ 7470.302557] *** LOCAL_OUT
[ 7470.302560] 127.0.0.53 --> 127.0.0.1 (UDP)
[ 7470.388053] *** LOCAL_OUT
```

Here, you can see that running the dig command did not work as expected. And this was reflected in the kernel messages.



```
Abdulfatah Abdillahi-Thu Mar 07-10:50:27 ~/.../packet_filter> lsmod | grep seed
seedFilter      16384  0
Abdulfatah Abdillahi-Thu Mar 07-10:50:39 ~/.../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

Abdulfatah Abdillahi-Thu Mar 07-10:59:38 ~/.../packet_filter>
```

```

[ 7770.379945] *** LOCAL_OUT
[ 7770.379946] 127.0.0.53 --> 127.0.0.1 (UDP)
[ 7815.489567] *** LOCAL_OUT
[ 7815.489573] 10.0.2.5 --> 192.168.1.1 (UDP)
[ 7815.516248] *** LOCAL_OUT
[ 7815.516249] 10.0.2.5 --> 185.125.190.97 (TCP)
[ 7815.627883] *** LOCAL_OUT
[ 7815.627887] 10.0.2.5 --> 185.125.190.97 (TCP)
[ 7815.628178] *** LOCAL_OUT
[ 7815.628180] 10.0.2.5 --> 185.125.190.97 (TCP)
[ 7815.740563] *** LOCAL_OUT
[ 7815.740700] 10.0.2.5 --> 185.125.190.97 (TCP)
[ 7815.741590] *** LOCAL_OUT
[ 7815.741591] 10.0.2.5 --> 185.125.190.97 (TCP)
[ 7821.071367] *** LOCAL_OUT
[ 7821.071371] 10.0.2.5 --> 10.0.2.3 (UDP)
[ 7923.289547] *** LOCAL_OUT
[ 7923.289549] 127.0.0.1 --> 127.0.0.1 (UDP)
[ 7923.289986] *** LOCAL_OUT
[ 7923.289989] 10.0.2.5 --> 8.8.8.8 (UDP)
[ 7923.289914] *** Dropping 8.8.8.8 (UDP), port 53
[ 7924.897843] *** LOCAL_OUT
[ 7924.897846] 10.0.2.5 --> 34.107.243.93 (TCP)
[ 7928.289883] *** LOCAL_OUT
[ 7928.289887] 10.0.2.5 --> 8.8.8.8 (UDP)
[ 7928.289910] *** Dropping 8.8.8.8 (UDP), port 53
[ 7933.289282] *** LOCAL_OUT
[ 7933.289284] 10.0.2.5 --> 8.8.8.8 (UDP)
[ 7933.289314] *** Dropping 8.8.8.8 (UDP), port 53

```

Then we removed the module and saw the respective kernel message on the terminal.

```

Abdulfatah Abdillahi-Thu Mar 07-10:50:39 ~/.../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

Abdulfatah Abdillahi-Thu Mar 07-10:59:38 ~/.../packet_filter> sudo rmmod seedFilter
Abdulfatah Abdillahi-Thu Mar 07-11:03:18 ~/.../packet_filter>

[ 8115.721931] 10.0.2.5 --> 185.125.190.98 (TCP)
[ 8115.722333] *** LOCAL_OUT
[ 8115.722335] 10.0.2.5 --> 185.125.190.98 (TCP)
[ 8121.079677] *** LOCAL_OUT
[ 8121.079682] 10.0.2.5 --> 10.0.2.3 (UDP)
[ 8158.561586] The filters are being removed.

```

**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 will each of the hook function be invoked.**

Here, we use the seed filter as a template to make a copy (*seedPrinting.c*) and use that copy for this part instead. This shows the Makefile pointing to the new copy of the code.



```
*Makefile      seedFilter.c      seedPrinting.c
1#obj-m += seedFilter.o
2 obj-m += seedPrinting.o # points to the new copy of the file
3 all:
4     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6 clean:
7     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
8
9 ins:
10    sudo dmesg -C
11    sudo insmod seedFilter.ko
12
13 rm:
14    sudo rmmod seedFilter
15
```

Makefile Tab Width: 8 Ln 1, Col 23 INS

11:47 AM 2024-03-07

Here, you can see the modifications I made to the copy to successfully complete this task.



```
*seedPrinting.c      seedFilter.c      *seedPrinting.c
1#include <linux/kernel.h>
2#include <linux/module.h>
3#include <linux/netfilter.h>
4#include <linux/netfilter_ipv4.h>
5#include <linux/ip.h>
6#include <linux/tcp.h>
7#include <linux/udp.h>
8#include <linux/icmp.h> //adding the icmp header
9#include <linux/if_ether.h>
10#include <linux/inet.h>
11
12
13 static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5; // Ensure we have a total of 5 hooks
14
15
16 unsigned int blockUDP(void *priv, struct sk_buff *skb,
17                       const struct nf_hook_state *state)
18 {
19     struct iphdr *iph;
20     struct udphdr *udph;
21
22     u16 port = 53; //DNS port
23     char ip[16] = "8.8.8.8"; //Google's public DNS
24     u32 ip_addr;
25
26     if (!skb) return NF_ACCEPT;
27
28     iph = ip_hdr(skb);
29     // Convert the IPv4 address from dotted decimal to 32-bit binary
30     in4_pton(ip, -1, (u8 *)ip_addr, '\0', NULL);
31
```

C Tab Width: 8 Ln 111, Col 26 INS

12:04 PM 2024-03-07

```

75 int registerFilter(void) {
76     printk(KERN_INFO "seedPrinting: Registering filters.\n");
77     //NF_INET_PRE_ROUTING
78     hook1.hook = printInfo;
79     hook1.hooknum = NF_INET_PRE_ROUTING; //adjust hooknum perimeter to reflect the required function
80     hook1(pf = PF_INET;
81     hook1.priority = NF_IP_PRI_FIRST;
82     nf_register_net_hook(&init_net, &hook1);
83     //NF_INET_LOCAL_IN
84     hook2.hook = printInfo;
85     hook2.hooknum = NF_INET_LOCAL_IN;
86     hook2(pf = PF_INET;
87     hook2.priority = NF_IP_PRI_FIRST;
88     nf_register_net_hook(&init_net, &hook2);
89     //NF_INET_FORWARD
90     hook3.hook = printInfo;
91     hook3.hooknum = NF_INET_FORWARD;
92     hook3(pf = PF_INET;
93     hook3.priority = NF_IP_PRI_FIRST;
94     nf_register_net_hook(&init_net, &hook3);
95     //NF_INET_LOCAL_OUT
96     hook4.hook = printInfo;
97     hook4.hooknum = NF_INET_LOCAL_OUT;
98     hook4(pf = PF_INET;
99     hook4.priority = NF_IP_PRI_FIRST;
100    nf_register_net_hook(&init_net, &hook4);
101    //NF_INET_POST_ROUTING
102    hook5.hook = printInfo;
103    hook5.hooknum = NF_INET_POST_ROUTING;
104    hook5(pf = PF_INET;
105    hook5.priority = NF_IP_PRI_FIRST;
106    nf_register_net_hook(&init_net, &hook5);

108     return 0;
109 }
110 // serves as module exit
111 void removeFilter(void) {
112     printk(KERN_INFO "seedPrinting: [The filters are being removed.\n"); //ensures we unregister all the net hooks
113     nf_unregister_net_hook(&init_net, &hook1);
114     nf_unregister_net_hook(&init_net, &hook2);
115     nf_unregister_net_hook(&init_net, &hook3);
116     nf_unregister_net_hook(&init_net, &hook4);
117     nf_unregister_net_hook(&init_net, &hook5);
118 }
119
120 module_init(registerFilter);
121 module_exit(removeFilter);
122
123 MODULE_LICENSE("GPL");
124

```

```

108     return 0;
109 }
110 // serves as module exit
111 void removeFilter(void) {
112     printk(KERN_INFO "seedPrinting: [The filters are being removed.\n"); //ensures we unregister all the net hooks
113     nf_unregister_net_hook(&init_net, &hook1);
114     nf_unregister_net_hook(&init_net, &hook2);
115     nf_unregister_net_hook(&init_net, &hook3);
116     nf_unregister_net_hook(&init_net, &hook4);
117     nf_unregister_net_hook(&init_net, &hook5);
118 }
119
120 module_init(registerFilter);
121 module_exit(removeFilter);
122
123 MODULE_LICENSE("GPL");
124

```

Then I generate the new kernel module (*seedPrinting.ko*) and insert it. This also reflected on the kernel messages.

```

Abdulfatah Abdillahi-Thu Mar 07-12:22:00 ~.../packet_filter> make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedPrinting.o
Building modules, stage 2.
MODPOST 1 modules
  CC [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedPrinting.mod.o
  LD [M] /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedPrinting.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
Abdulfatah Abdillahi-Thu Mar 07-12:23:31 ~.../packet_filter> ls
Makefile      seedFilter.c  seedFilter.mod.c  seedPrinting.c  seedPrinting.mod.c
modules.order  seedFilter.ko  seedFilter.mod.o  seedPrinting.ko  seedPrinting.mod.o
Module.symvers seedFilter.mod  seedFilter.o   seedPrinting.mod  seedPrinting.o
Abdulfatah Abdillahi-Thu Mar 07-12:23:33 ~.../packet_filter> sudo insmod seedPrinting.ko
Abdulfatah Abdillahi-Thu Mar 07-12:23:59 ~.../packet_filter>

```

```

Terminal x Terminal x
[ 8115.722335] 10.0.2.5 --> 185.125.190.98 (TCP)
[ 8121.079677] *** LOCAL_OUT
[ 8121.079682] 10.0.2.5 --> 10.0.2.3 (UDP)
[ 8158.561586] The filters are being removed.
[12998.998095] seedPrinting: Registering filters. ↩
[13030.559884] *** PRE_ROUTING
[13030.560222] 34.107.243.93 --> 10.0.2.5 (TCP)
[13030.560528] *** LOCAL_IN
[13030.560813] 34.107.243.93 --> 10.0.2.5 (TCP)
[13030.561192] *** LOCAL_OUT
[13030.561472] 10.0.2.5 --> 34.107.243.93 (TCP)
[13030.561729] *** POST_ROUTING
[13030.561979] 10.0.2.5 --> 34.107.243.93 (TCP)
[13030.562354] *** LOCAL_OUT
[13030.562358] 10.0.2.5 --> 34.107.243.93 (TCP)
[13030.562367] *** POST_ROUTING
[13030.562368] 10.0.2.5 --> 34.107.243.93 (TCP)
[13030.710851] *** PRE_ROUTING
[13030.711322] 34.107.243.93 --> 10.0.2.5 (TCP)
[13030.711722] *** LOCAL_IN
[13030.712222] 34.107.243.93 --> 10.0.2.5 (TCP)
[13125.559029] *** LOCAL_OUT
[13125.559033] 10.0.2.5 --> 192.168.1.1 (UDP)
[13125.559061] *** POST_ROUTING
[13125.559063] 10.0.2.5 --> 192.168.1.1 (UDP)
[13125.582012] *** PRE_ROUTING
[13125.582015] 192.168.1.1 --> 10.0.2.5 (UDP)
[13125.582026] *** LOCAL_IN
[13125.582027] 192.168.1.1 --> 10.0.2.5 (UDP)

```

To verify that the hooks will work as expected we will run the same dig command as above from our machine and check the kernel messages. As you can see this was successful.

```

Abdulfatah Abdillahi-Thu Mar 07-12:32:34 ~/.../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: 22662
;; 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.    4143    IN      A      93.184.216.34

;; Query time: 24 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Mar 07 12:32:42 EST 2024
;; MSG SIZE  rcvd: 60

Abdulfatah Abdillahi-Thu Mar 07-12:32:42 ~/.../packet_filter>

```

```

[13521.783376] *** LOCAL_OUT
[13521.783377] 127.0.0.1 --> 127.0.0.1 (loop)
[13521.783380] *** POST_ROUTING
[13521.783381] 10.0.2.5 --> 8.8.8.8 (UDP)
[13521.805093] *** PRE_ROUTING
[13521.805199] 8.8.8.8 --> 10.0.2.5 (UDP)
[13521.805305] *** LOCAL_IN
[13521.805422] 8.8.8.8 --> 10.0.2.5 (UDP)

```

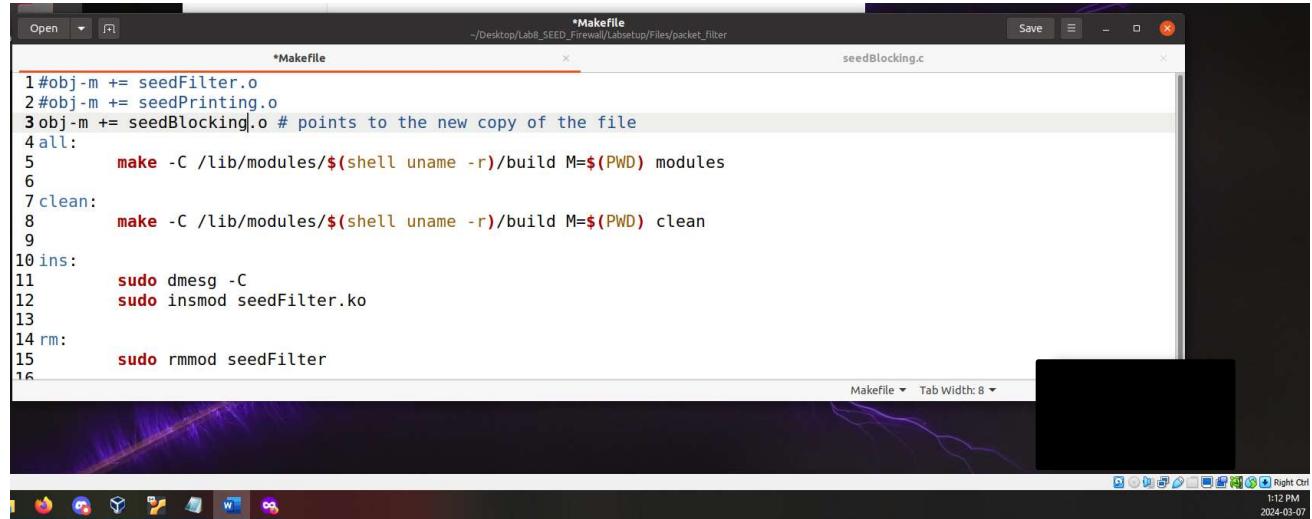
Based on the above screenshots I can conclude that the ***PER\_ROUTIN*** hook is invoked as soon as a packet enters the networking stack and before any routing decision is made. While the ***LOCAL\_IN*** hook is invoked anytime a packet is destined for the local system. The ***FORWARD*** hook is invoked

if the packet is being forwarded to another destination. The ***LOCAL\_OUT*** hook is invoked anytime a packet originates from our local system. The ***POST\_ROUTING*** hook is invoked after a decision has been made about the packet and just before it leaves the network interface.

**3. Implement two more hooks to achieve the following: (1) preventing other computers to ping the VM, and (2) preventing other computers to telnet 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.**

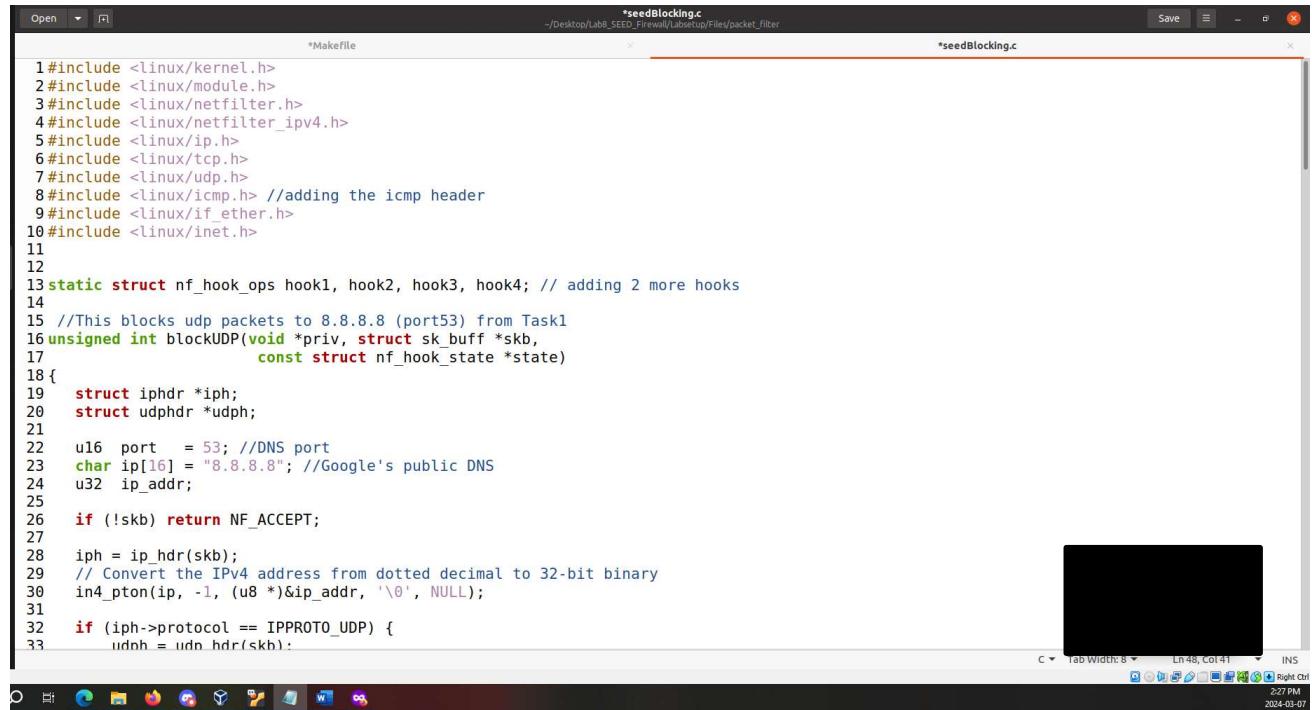
Here, again, I will make another copy of the *seedFilter.c* and work on that copy (*seedBlocking.c*) for this part.

These are the modifications I made to the *makeFile* and the new copy of the source code.



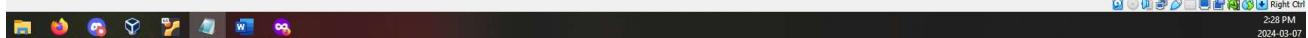
```
*Makefile
-/Desktop/Lab8_SEED_Firewall/Lab8setup/Files/packet_filter
*seedBlocking.c

1#obj-m += seedFilter.o
2#obj-m += seedPrinting.o
3obj-m += seedBlocking.o # points to the new copy of the file
4all:
5    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
6
7clean:
8    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
9
10ins:
11    sudo dmesg -C
12    sudo insmod seedFilter.ko
13
14rm:
15    sudo rmmod seedFilter
16
```

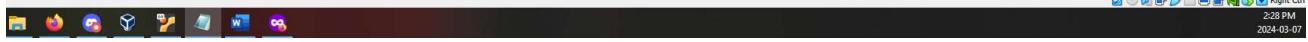


```
*seedBlocking.c
-/Desktop/Lab8_SEED_Firewall/Lab8setup/Files/packet_filter
*seedBlocking.c

1#include <linux/kernel.h>
2#include <linux/module.h>
3#include <linux/netfilter.h>
4#include <linux/netfilter_ipv4.h>
5#include <linux/ip.h>
6#include <linux/tcp.h>
7#include <linux/udp.h>
8#include <linux/icmp.h> //adding the icmp header
9#include <linux/if_ether.h>
10#include <linux/inet.h>
11
12
13static struct nf_hook_ops hook1, hook2, hook3, hook4; // adding 2 more hooks
14
15//This blocks udp packets to 8.8.8.8 (port53) from Task1
16unsigned int blockUDP(void *priv, struct sk_buff *skb,
17                      const struct nf_hook_state *state)
18{
19    struct iphdr *iph;
20    struct udphdr *udph;
21
22    u16 port = 53; //DNS port
23    char ip[16] = "8.8.8.8"; //Google's public DNS
24    u32 ip_addr;
25
26    if (!skb) return NF_ACCEPT;
27
28    iph = ip_hdr(skb);
29    // Convert the IPv4 address from dotted decimal to 32-bit binary
30    in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
31
32    if (iph->protocol == IPPROTO_UDP) {
33        udph = udp_hdr(skb);
```



```
*seedBlocking.c
~/Desktop/Lab8_SEED_Firewall/LabSetup/Files/packet_Filter
Save ×
*seedBlocking.c
41 // This blocks icmp to our vm (10.9.0.1)
42 unsigned int blockICMP(void *priv, struct sk_buff *skb,
43                         const struct nf_hook_state *state)
44 {
45     struct iphdr *iph;
46     struct icmphdr *icmph;
47
48     //u16 port = 53; //No port needed for ICMP
49     char ip[16] = "10.9.0.1"; //Host vm IP address
50     u32 ip_addr;
51
52     if (!skb) return NF_ACCEPT;
53
54     iph = ip_hdr(skb);
55     // Convert the IPv4 address from dotted decimal to 32-bit binary
56     in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
57
58     if (iph->protocol == IPPROTO_ICMP) {
59         icmph = icmp_hdr(skb);
60         if (iph->daddr == ip_addr && icmph->type == ICMP_ECHO){
61             printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n", &(iph->daddr));
62             return NF_DROP;
63         }
64     }
65     return NF_ACCEPT;
66 }
67
68 // This blocks telnet (port 23) to our vm (10.9.0.1)
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95 unsigned int printInfo(void *priv, struct sk_buff *skb,
```



```
*seedBlocking.c
~/Desktop/Lab8_SEED_Firewall/LabSetup/Files/packet_Filter
Save ×
*seedBlocking.c
68 // This blocks telnet (port 23) to our vm (10.9.0.1)
69 unsigned int blockTelnet(void *priv, struct sk_buff *skb,
70                         const struct nf_hook_state *state)
71 {
72     struct iphdr *iph;
73     struct tcphdr *tcph;
74
75     u16 port = 23; //Telnet port
76     char ip[16] = "10.9.0.1"; //Host vm IP address
77     u32 ip_addr;
78
79     if (!skb) return NF_ACCEPT;
80
81     iph = ip_hdr(skb);
82     // Convert the IPv4 address from dotted decimal to 32-bit binary
83     in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
84
85     if (iph->protocol == IPPROTO_TCP) {
86         tcph = tcp_hdr(skb);
87         if (iph->daddr == ip_addr && ntohs(tcph->dest) == port){
88             printk(KERN_WARNING "*** Dropping %pI4 (TCP), port %d\n", &(iph->daddr), port);
89             return NF_DROP;
90         }
91     }
92     return NF_ACCEPT;
93 }
94
95 unsigned int printInfo(void *priv, struct sk_buff *skb,
```

```
seedBlocking.c
128 int registerFilter(void) {
129     printk(KERN_INFO "seedBlocking: Registering filters.\n");
130
131     hook1.hook = printInfo;
132     hook1.hooknum = NF_INET_LOCAL_OUT;
133     hook1.pf = PF_INET;
134     hook1.priority = NF_IP_PRI_FIRST;
135     nf_register_net_hook(&init_net, &hook1);
136
137     hook2.hook = blockUDP;
138     hook2.hooknum = NF_INET_POST_ROUTING;
139     hook2.pf = PF_INET;
140     hook2.priority = NF_IP_PRI_FIRST;
141     nf_register_net_hook(&init_net, &hook2);
142
143     hook3.hook = blockICMP; //new hook blocking ICMP
144     hook3.hooknum = NF_INET_PRE_ROUTING;
145     hook3.pf = PF_INET;
146     hook3.priority = NF_IP_PRI_FIRST;
147     nf_register_net_hook(&init_net, &hook3);
148
149     hook4.hook = blockTelnet; //new hook blocking Telnet
150     hook4.hooknum = NF_INET_PRE_ROUTING;
151     hook4.pf = PF_INET;
152     hook4.priority = NF_IP_PRI_FIRST;
153     nf_register_net_hook(&init_net, &hook4);
154
155     return 0;
156 }
```

```
seedBlocking.c
157 // serves as module_exit
158 void removeFilter(void) {
159     printk(KERN_INFO "seedBlocking: The filters are being removed.\n");
160     nf_unregister_net_hook(&init_net, &hook1);
161     nf_unregister_net_hook(&init_net, &hook2);
162     nf_unregister_net_hook(&init_net, &hook3); //adding 2 new hook to unregister
163     nf_unregister_net_hook(&init_net, &hook4);
164 }
165
166 module_init(registerFilter);
167 module_exit(removeFilter);
168
169 MODULE_LICENSE("GPL");
170
```

Now, based on these modifications I will generate the new module kernel module (seedBlocking.ko) and insert it.

```

Terminal Terminal Terminal Terminal Terminal Terminal Terminal Terminal Terminal
Terminal Abdillahi-Thu Mar 07-02:35:17 ~/.../packet_filter>
Terminal Abdillahi-Thu Mar 07-02:35:17 ~/.../packet_filter> make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedBlocking.o
Building modules, stage 2.
MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedBlocking.mod.o
  LD [M]  /home/seed/Desktop/Lab8_SEED_Firewall/Labsetup/Files/packet_filter/seedBlocking.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
Abdillahi-Thu Mar 07-02:35:30 ~/.../packet_filter> ls
Makefile Module.symvers seedBlocking.ko seedBlocking.mod.c seedBlocking.o seedPrinting.c
modules.order seedBlocking.c seedBlocking.mod seedBlocking.mod.o seedFilter.c
Abdillahi-Thu Mar 07-02:35:34 ~/.../packet_filter> sudo insmod seedBlocking.ko
Abdillahi-Thu Mar 07-02:35:52 ~/.../packet_filter>

```

The screenshot shows a Linux desktop environment with a dark theme. In the top panel, there are seven terminal windows labeled "Terminal". The terminal content shows the process of building a kernel module named "seedBlocking.ko". The module is successfully built and inserted into the kernel. Below the terminals is a desktop background featuring a purple lightning bolt pattern. At the bottom is a dock with various icons, and the system tray shows the date and time as "2024-03-07".

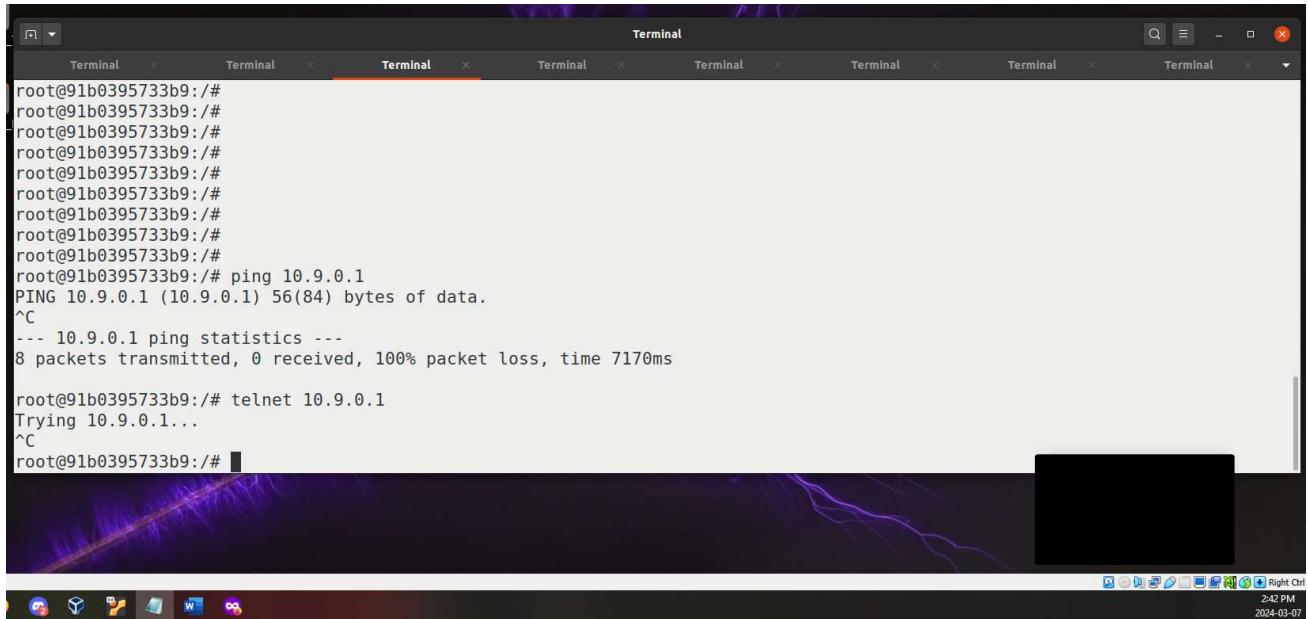
```

[20911.851844] seedBlocking: Registering filters.
[20925.495761] *** LOCAL_OUT
[20925.495763] 10.0.2.5 -> 192.168.1.1 (UDP)
[20959.267039] *** LOCAL_OUT
[20959.267042] 10.0.2.5 -> 34.149.100.209 (TCP)
[20959.289173] *** LOCAL_OUT
[20959.289177] 10.0.2.5 -> 34.149.100.209 (TCP)
[20970.193417] *** LOCAL_OUT
[20970.193420] 127.0.0.1 -> 127.0.0.53 (UDP)
[20970.194021] *** LOCAL_OUT
[20970.194023] 127.0.0.53 -> 127.0.0.1 (UDP)
[20970.287912] *** LOCAL_OUT
[20970.287915] 127.0.0.1 -> 127.0.0.53 (UDP)
[20970.288245] *** LOCAL_OUT
[20970.288247] 127.0.0.53 -> 127.0.0.1 (UDP)

```

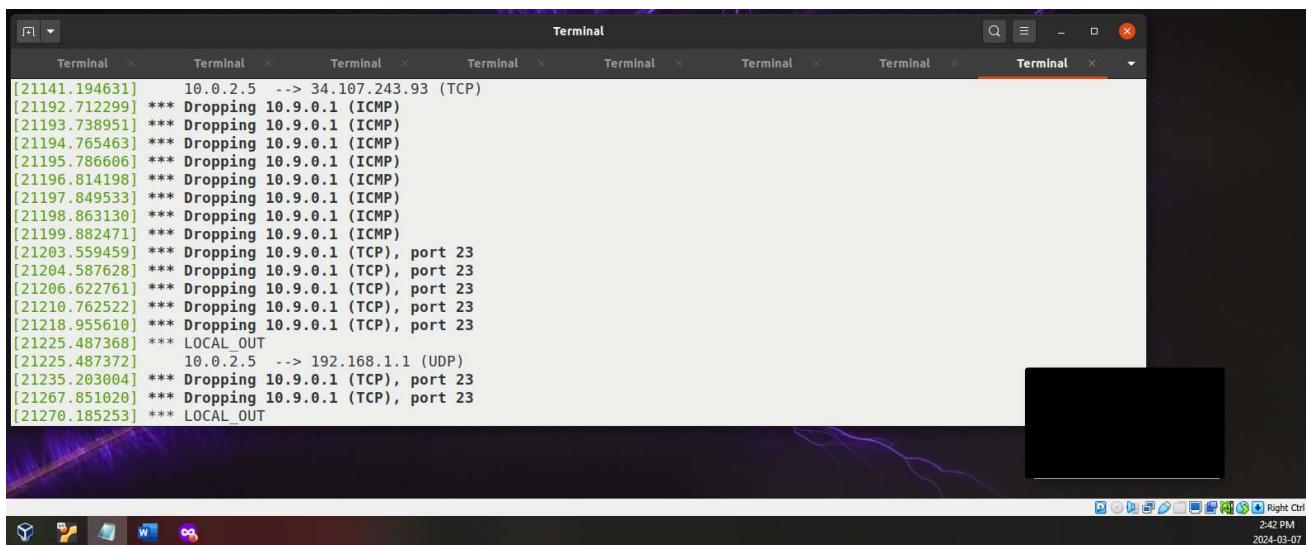
This screenshot shows a single terminal window in the foreground displaying kernel messages related to the "seedBlocking" module. The log entries show various network filter registrations. The background is a dark desktop environment with a purple lightning bolt pattern. The system tray at the bottom right shows the date and time as "2024-03-07".

Next, we attempt to ping and telnet, and we can see that both ICMP and telnet requests have timed out (as expected). This is also reflected in the kernel messages.



```
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/# 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 ---
8 packets transmitted, 0 received, 100% packet loss, time 7170ms

root@91b0395733b9:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@91b0395733b9:/#
```



```
[21141.194631] 10.0.2.5 --> 34.107.243.93 (TCP)
[21192.712299] *** Dropping 10.9.0.1 (ICMP)
[21193.738951] *** Dropping 10.9.0.1 (ICMP)
[21194.765463] *** Dropping 10.9.0.1 (ICMP)
[21195.786606] *** Dropping 10.9.0.1 (ICMP)
[21196.814198] *** Dropping 10.9.0.1 (ICMP)
[21197.849533] *** Dropping 10.9.0.1 (ICMP)
[21198.863130] *** Dropping 10.9.0.1 (ICMP)
[21199.882471] *** Dropping 10.9.0.1 (ICMP)
[21203.559459] *** Dropping 10.9.0.1 (TCP), port 23
[21204.587628] *** Dropping 10.9.0.1 (TCP), port 23
[21206.622761] *** Dropping 10.9.0.1 (TCP), port 23
[21210.762522] *** Dropping 10.9.0.1 (TCP), port 23
[21218.955610] *** Dropping 10.9.0.1 (TCP), port 23
[21225.487368] *** LOCAL_OUT
[21225.487372] 10.0.2.5 --> 192.168.1.1 (UDP)
[21235.203004] *** Dropping 10.9.0.1 (TCP), port 23
[21267.851620] *** Dropping 10.9.0.1 (TCP), port 23
[21270.185253] *** LOCAL_OUT
```

## Task 2: Experimenting with Stateless Firewall Rules

### Task 2.A: Protecting the Router

The below figure shows the rules I placed on the router. The firewall is set to default drop meaning it drops all traffic except for the authorized ones. In this case, we only allowed ICMP echo-request and echo-reply (aka ping).

```

root@e4e8230bacd0:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@e4e8230bacd0:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@e4e8230bacd0:/# iptables -P OUTPUT DROP
root@e4e8230bacd0:/# iptables -P INPUT DROP
root@e4e8230bacd0:/# 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
root@e4e8230bacd0:/#

```

This figure shows me attempting to ping the router from hostA (which worked as expected) and attempting to telnet to the router which did not work.

```

root@91b0395733b9:/#
root@91b0395733b9:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.220 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.056 ms
^C
--- 192.168.60.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.056/0.138/0.220/0.082 ms
root@91b0395733b9:/# telnet 192.168.60.11
Trying 192.168.60.11...
^C
root@91b0395733b9:/#

```

Then before moving on, I restored the filter table to its original state in the router.

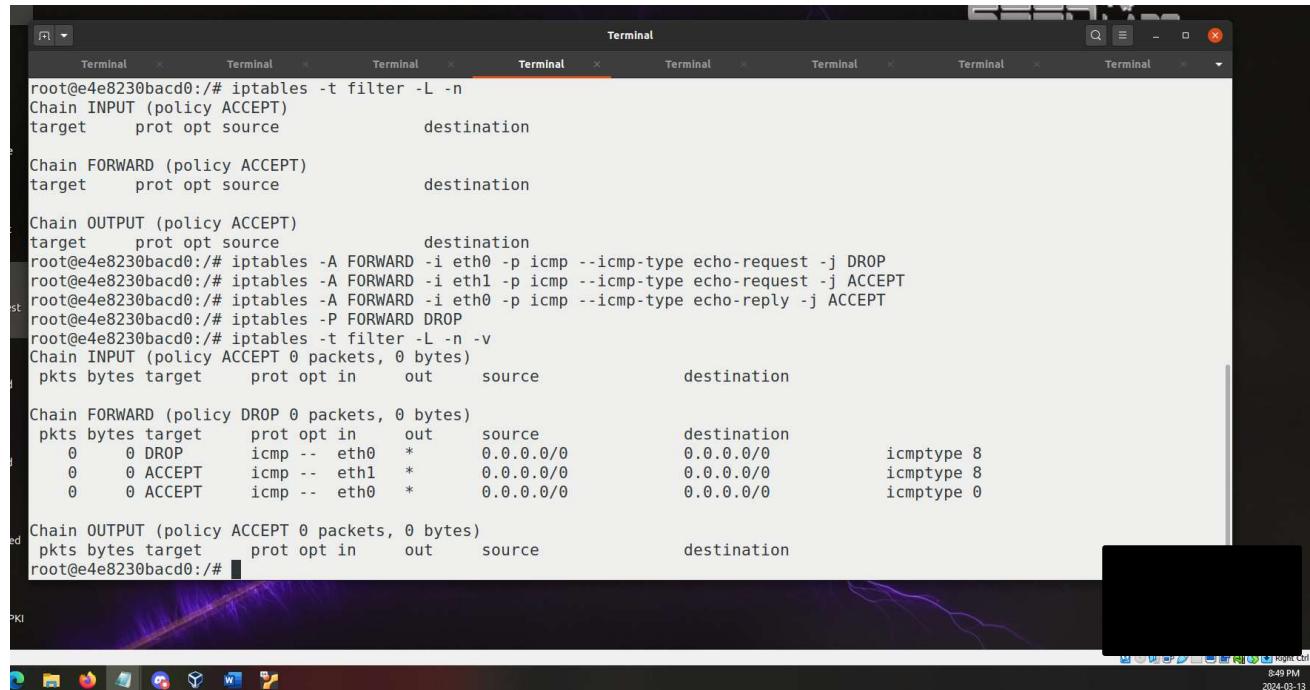
```

root@e4e8230bacd0:/# iptables -P INPUT DROP
root@e4e8230bacd0:/# 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
root@e4e8230bacd0:/# iptables -F
root@e4e8230bacd0:/# iptables -P OUTPUT ACCEPT
root@e4e8230bacd0:/# iptables -P INPUT ACCEPT
root@e4e8230bacd0:/#

```

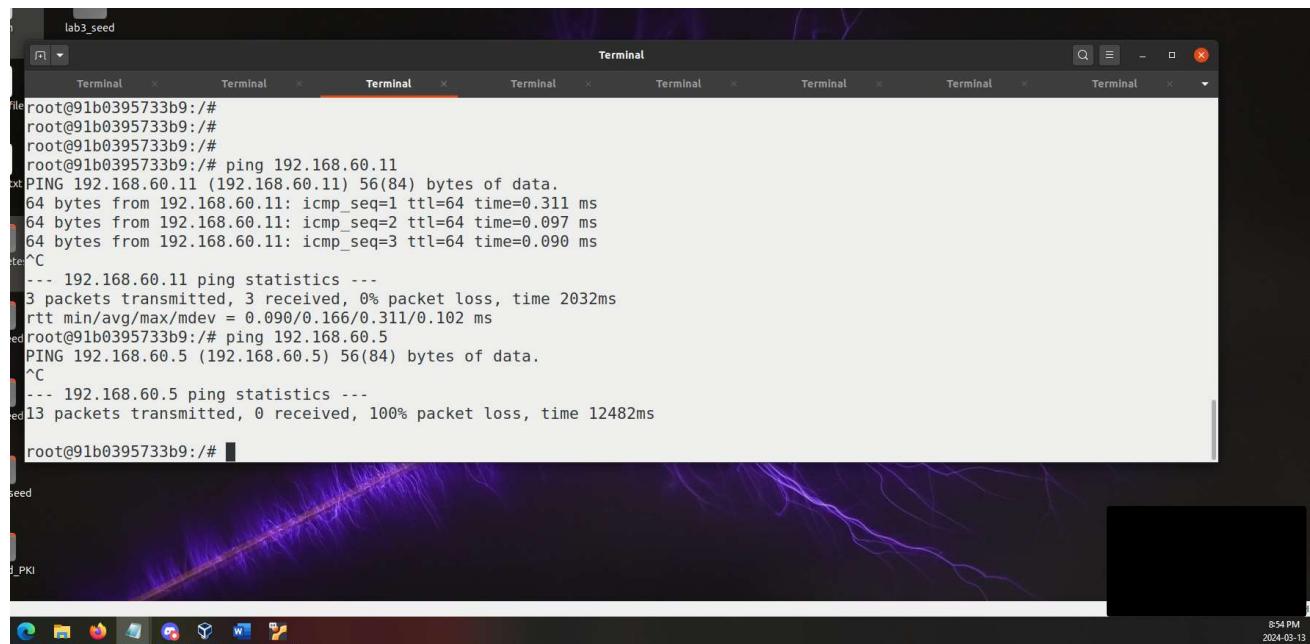
## Task 2.B: Protecting the Internal Network

The below screenshot shows the rules I put in place to enforce the following restrictions: Outside hosts cannot ping internal hosts, Outside hosts can ping the router, Internal hosts can ping outside hosts, and all other packets between the internal and external networks should be blocked. Please note that *eth0* is the interface facing the outside network, while *eth1* is the interface facing the inside network. And *icmptype 8* represents echo-request while *icmptype 0* represents echo-reply.



```
root@e4e8230bacd0:/# 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
root@e4e8230bacd0:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
root@e4e8230bacd0:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
root@e4e8230bacd0:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
root@e4e8230bacd0:/# iptables -P FORWARD DROP
root@e4e8230bacd0:/# iptables -t filter -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
      0     0   ACCEPT    icmp  --  eth1   *       0.0.0.0/0        0.0.0.0/0
      0     0   ACCEPT    icmp  --  eth0   *       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
root@e4e8230bacd0:/#
```

Here, you can see that outside host can ping the router but not internal hosts.



```
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
 64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.311 ms
 64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.097 ms
 64 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.090 ms
^C
--- 192.168.60.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2032ms
rtt min/avg/max/mdev = 0.090/0.166/0.311/0.102 ms
root@91b0395733b9:/# 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 ---
13 packets transmitted, 0 received, 100% packet loss, time 12482ms
root@91b0395733b9:/#
```

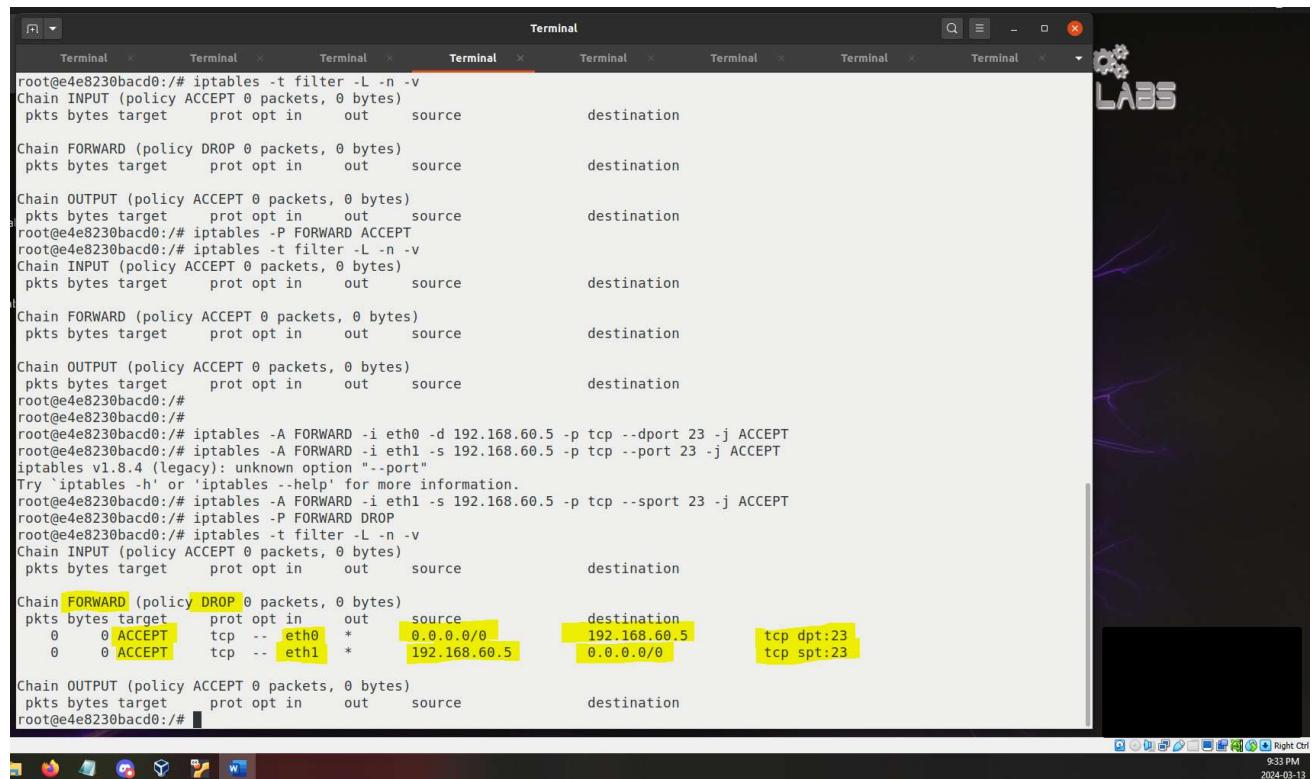
Here, you can see that internal hosts can ping outside hosts. You can also see that all other packet types (e.g. telnet) are blocked between the internal and external networks.

Then proceeded to clean up the rules before moving to the next part.

```
root@e4e8230bacd0:/# iptables -t filter -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out    source          destination
ext      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
ext      pkts bytes target     prot opt in     out    source          destination
0       0   DROP     icmp -- eth0   *      0.0.0.0/0      0.0.0.0/0      icmp type 8
0       0   ACCEPT   icmp -- eth1   *      0.0.0.0/0      0.0.0.0/0      icmp type 8
0       0   ACCEPT   icmp -- eth0   *      0.0.0.0/0      0.0.0.0/0      icmp type 0
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out    source          destination
root@e4e8230bacd0:/#
root@e4e8230bacd0:/# iptables -F
root@e4e8230bacd0:/# iptables -P OUTPUT ACCEPT
root@e4e8230bacd0:/# iptables -P INPUT ACCEPT
root@e4e8230bacd0:/#
```

## Task 2.C: Protecting Internal Servers

Here, you can see the rule I put in place to enforce the following restrictions on the internal servers:  
Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts, Outside hosts cannot access other internal servers, Internal hosts can access all the internal servers, Internal hosts cannot access external servers. As before that *eth0* is the interface facing the outside network, while *eth1* is the interface facing the inside network.



The screenshot shows a terminal window with multiple tabs, each labeled "Terminal". The active tab displays the output of several iptables commands. The output includes:

```
root@e4e8230bacd0:/# iptables -t filter -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
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in     out      source          destination
root@e4e8230bacd0:/# iptables -P FORWARD ACCEPT
root@e4e8230bacd0:/# iptables -t filter -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
root@e4e8230bacd0:/#
root@e4e8230bacd0:/# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
root@e4e8230bacd0:/# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --port 23 -j ACCEPT
iptables v1.8.4 (legacy): unknown option "--port"
Try `iptables -h` or `iptables -help` for more information.
root@e4e8230bacd0:/# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
root@e4e8230bacd0:/# iptables -P FORWARD DROP
root@e4e8230bacd0:/# iptables -t filter -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
root@e4e8230bacd0:/#
```

The terminal window has a dark theme with a purple gradient background. A "LABS" logo is visible in the top right corner. The taskbar at the bottom shows various application icons, and the system tray indicates the date and time as "2024-03-13 9:33 PM".

Testing the rules, we see that outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts (e.g. 192.168.60.6, 192.168.60.7).

```
root@91b0395733b9:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5107486143de login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.

Last login: Thu Mar 14 01:39:39 UTC 2024 on pts/2
seed@5107486143de:~$ exit
logout
Connection closed by foreign host.
root@91b0395733b9:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@91b0395733b9:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@91b0395733b9:/#
```

We can also see that all internal hosts can access other internal hosts. In this example, internal host1 (192.168.60.5), can access host2 (192.168.60.6) and host3 (192.168.60.7) but not external hosts like hostA (10.9.0.5).

```
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cf490d997b7c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@cf490d997b7c:~$ exit
logout
Connection closed by foreign host.
root@5107486143de:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
```

```
Connection closed by foreign host.
root@5107486143de:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e70272ed8cfb login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@e70272ed8cfb:~$ exit
logout
Connection closed by foreign host.
root@5107486143de:/#
```

9:48 PM  
2024-03-13

Finally, we can see that internal hosts (e.g. host1) cannot access the outside hosts.

```
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@5107486143de:/#
```

9:48 PM  
2024-03-13

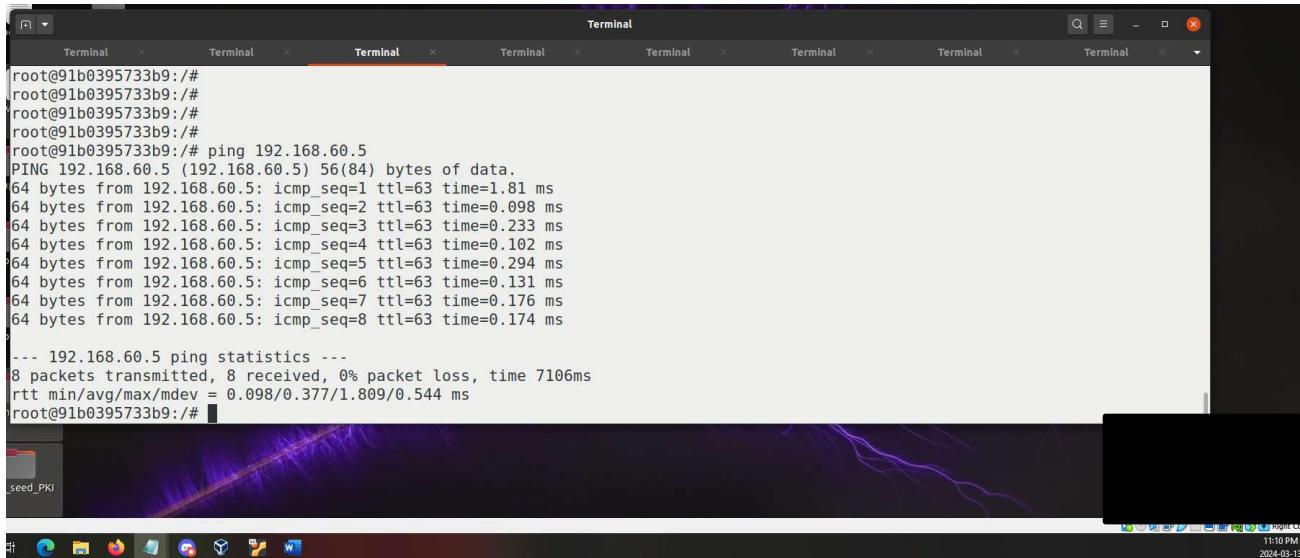
Cleaning rules for next task.

```
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target  prot opt in     out      source          destination
root@e4e8230bacd0:/#
root@e4e8230bacd0:/# iptables -F
root@e4e8230bacd0:/# iptables -P FORWARD ACCEPT
root@e4e8230bacd0:/#
```

9:48 PM  
2024-03-13

### Task 3: Connection Tracking and Stateful Firewall

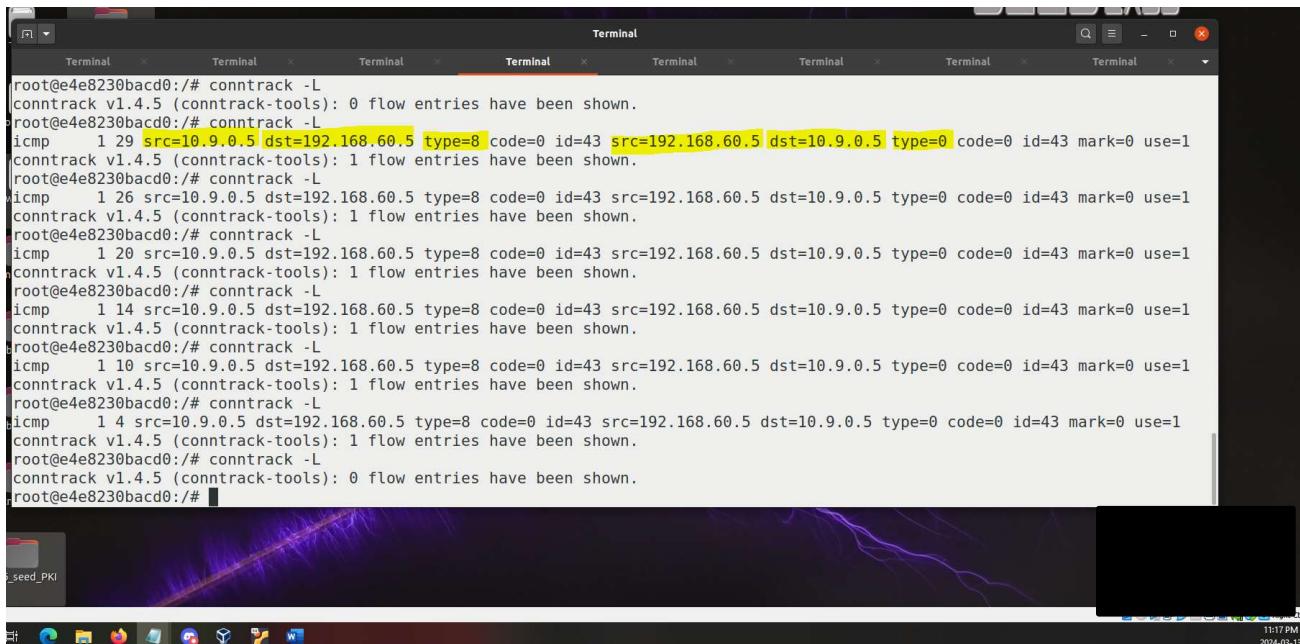
Here, you see me attempting the first experiment (ICMP experiment) by pinging 192.168.60.5 from 10.9.0.5.



```
root@91b0395733b9:/# 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=1.81 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.098 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.233 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.102 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.294 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.131 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.176 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.174 ms

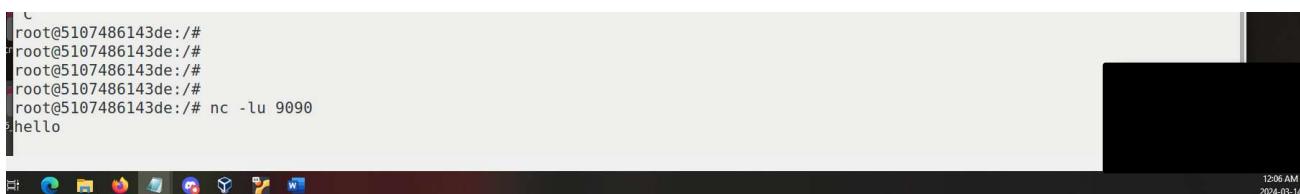
--- 192.168.60.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7106ms
rtt min/avg/max/mdev = 0.098/0.377/1.809/0.544 ms
root@91b0395733b9:/#
```

By running `conntrack -L` we can see a list of currently tracked connection which only shows the ping command I was running at the time with an ICMP echo-request (`type=8`) from 10.9.0.5 and an ICMP (`type=0`) echo-reply from 192.168.60.5. The ICMP connection state was kept for about 5 or 6 seconds.



```
root@e4e8230bacd0:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
icmp 1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
icmp 1 26 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
icmp 1 20 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
icmp 1 14 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
icmp 1 10 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
icmp 1 4 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@e4e8230bacd0:/#
```

Here, you see me attempting the second experiment (UDP experiment) by running a netcat UDP server on host1 (192.168.60.5 port 9090) and connecting to the server from host A (10.9.0.5).



```
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/# nc -l -u 9090
hello
root@5107486143de:/#
```

```
root@91b0395733b9:/# nc -u 192.168.60.5 9090
hello
```

12:06 AM  
2024-03-14

By running **conntrack -L** we can see a list of currently tracked connection which only shows the UDP packet I sent by typing in the netcat window on hostA. The UDP connection state was kept for about 11 or 12 seconds.

```
root@e4e8230bacd0:/# conntrack -L
udp 17 26 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 24 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 22 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 19 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 14 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 12 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 9 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 7 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 4 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
udp 17 0 src=10.9.0.5 dst=192.168.60.5 sport=47271 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=47271 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@e4e8230bacd0:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

12:09 AM  
2024-03-14

At last, you can see me attempting the third experiment (TCP experiment) by running a netcat TCP server on host1 (192.168.60.5 port 9090) and connecting to the server from host A (10.9.0.5).

```
root@5107486143de:/# nc -l 9090
how are you?
```

```
root@91b0395733b9:/# nc 192.168.60.5 9090
how are you?
```

12:18 AM  
2024-03-14

12:18 AM  
2024-03-14

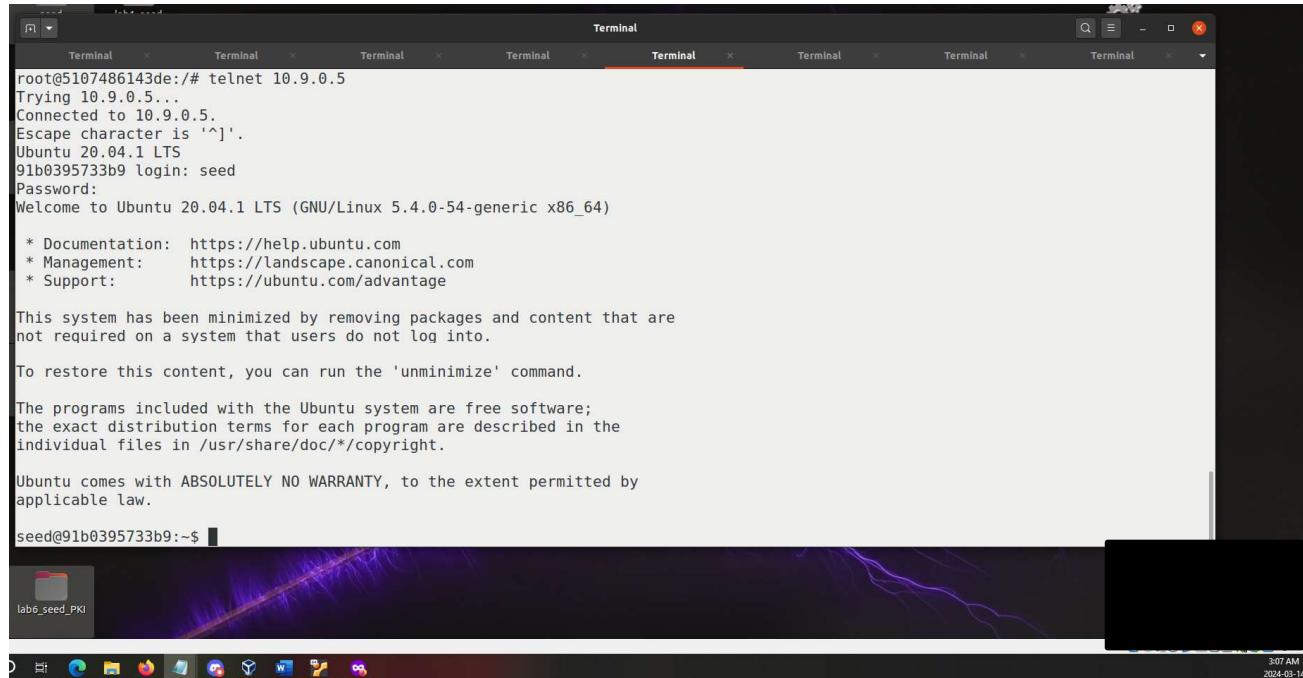
By running **conntrack -L** we can see a list of currently tracked connection which only shows the TCP packet I sent by typing in the netcat window on hostA. The TCP connection state was kept for about 113 seconds after the connection was terminated. You can see this shown on the highlighted yellow portion of the output as the **TIME\_WAIT** feature drops from 113 to zero.

### Task 3.B: Setting Up a Stateful Firewall

In this section we are rewriting the rules from Task 2.C using the connection tracking mechanism. However, unlike last time, we will now add a rule allowing internal hosts to visit any external servers. The first rule allows TCP traffic on port 23 from the outside network (eth0) to host1 (192.168.60.5 on port 23) on the inside network. It has the SYN flag set to allow for connections to be initiated. The second rule allows incoming TCP SYN packets from the inside network (eth1) to visit all hosts on the outside network. The third rule allows incoming TCP packets related to established connections. The fourth rule drops all other TCP traffic not matching the above rules. The last rule sets the default policy for FORWARD chain to ACCEPT.

Then some tests were conducted to test these rules. First, we see if hosts on the inside network

(host1) could telnet to hosts on the outside network (hostA). This was successful.



```
root@5107486143de:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
91b0395733b9 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

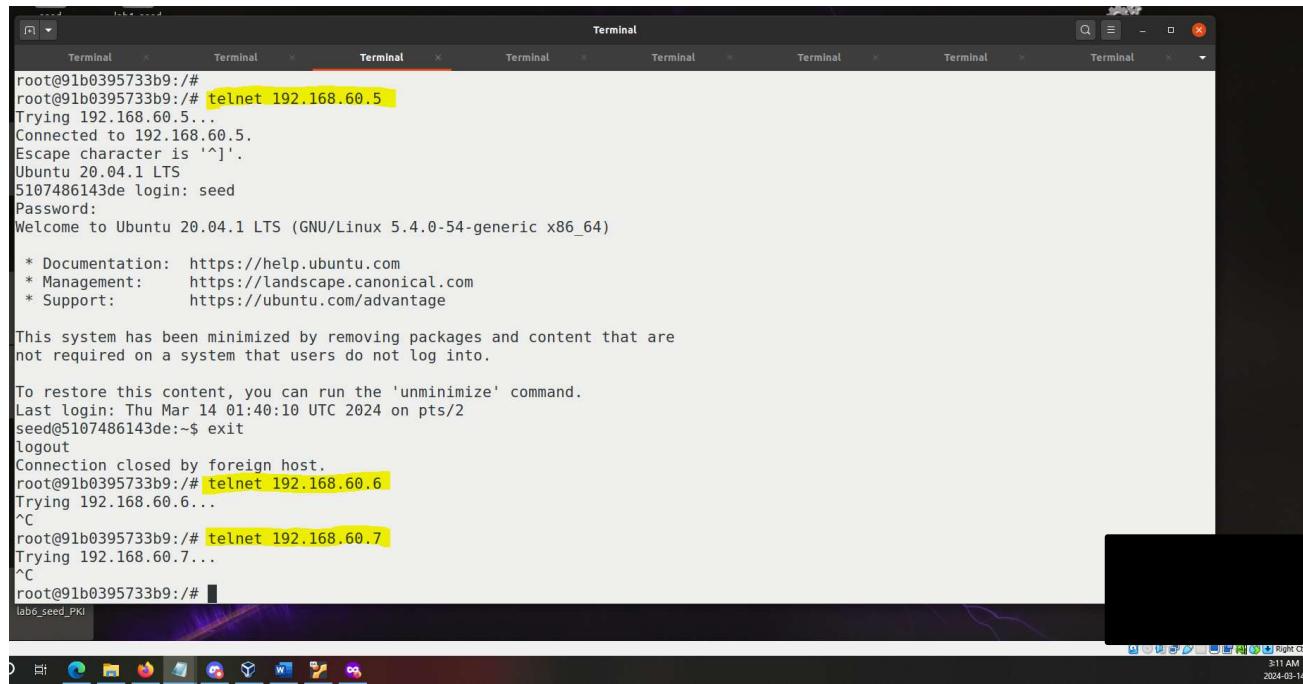
To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@91b0395733b9:~$
```

Then we see if outside hosts can connect to inside hosts – specifically only 192.168.60.5:23 and nothing else. This was successful.



```
root@91b0395733b9:/#
root@91b0395733b9:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5107486143de login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Thu Mar 14 01:40:10 UTC 2024 on pts/2
seed@5107486143de:~$ exit
logout
Connection closed by foreign host.
root@91b0395733b9:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@91b0395733b9:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@91b0395733b9:/#
```

We can also see that internal hosts can visit **any** outside hosts (e.g. netcat). In this example, I setup a netcat listener on hostA (10.9.0.5) and attempt to connect to it from host1. This was successful as expected per the firewall rules. We can see the hi! message reflected on both ends.

The screenshot shows two terminal windows side-by-side. Both windows have a title bar 'lab6\_seed\_PKI' and a timestamp '2024-03-14'. The left window (host1) has the command 'nc -l 9090' running. The right window (host2) has the command 'nc 10.9.0.5 9090' running. In the host1 window, the message 'Hi!' is visible. In the host2 window, the message 'root@91b0395733b9:~\$' is visible.

```
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~# nc 10.9.0.5 9090  
Hi!  
lab6_seed_PKI  
root@91b0395733b9:~#  
root@91b0395733b9:~#  
root@91b0395733b9:~#  
root@91b0395733b9:~# nc -l 9090  
Hi!  
lab6_seed_PKI
```

If we try to do it the other way around with the netcat server being on the inside network, then it wouldn't work as per the firewall rules.

The screenshot shows two terminal windows side-by-side. Both windows have a title bar 'lab6\_seed\_PKI' and a timestamp '2024-03-14'. The left window (host1) has the command 'nc -l 9090' running. The right window (host2) has the command 'nc 192.168.60.5 9090' running. In the host1 window, the message 'hello' is visible. In the host2 window, the message 'root@91b0395733b9:~\$' is visible.

```
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~#  
root@5107486143de:~# nc -l 9090  
hello  
lab6_seed_PKI  
root@91b0395733b9:~#  
root@91b0395733b9:~#  
root@91b0395733b9:~#  
root@91b0395733b9:~#  
root@91b0395733b9:~# nc 192.168.60.5 9090  
hello  
lab6_seed_PKI
```

As you can see the hello message was not reflected on the netcat window of host1.

**Based on these two sets of rules, compare these two different approaches, and explain the advantage and disadvantage of each approach.**

With connection tracking the disadvantage is consumption because there may be additional consumption associated with connection tracking, especially in high-traffic environments, potentially impacting performance. Without this the advantage would be that it consumes less resources.

With stateful firewall the disadvantage is potential security risks because depending solely on stateful inspection may introduce security risks, as it relies on the assumption that all packets related to established connections are legitimate. The advantage might be that stateful firewalls are generally more efficient in terms of processing packets since they do not require tracking individual connections for every rule.

#### Task 4: Limiting Network Traffic

As per the instructions, you can see me here adding the first rule to the router container.

```

root@e4e8230bacd0:#
root@e4e8230bacd0:#
root@e4e8230bacd0:/# iptables -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
root@e4e8230bacd0:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
root@e4e8230bacd0:#

```

Then I test the rule by pinging 192.168.60.5 from 10.9.0.5 and seeing if the rule works as expected. You can see that the rule did not work as expected.

```

root@91b0395733b9:#
root@91b0395733b9:#
root@91b0395733b9:/# 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=1.22 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.130 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.180 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.121 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.157 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.125 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.127 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.133 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.167 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.135 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.126 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.166 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.132 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.131 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.131 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.133 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.133 ms
^C
--- 192.168.60.5 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16376ms
rtt min/avg/max/mdev = 0.121/0.202/1.219/0.254 ms
root@91b0395733b9:#

```

Now we will see if adding the second rule will make a difference. This rule ensures that packets not matching the first rule will be dropped by the second rule.

```

root@e4e8230bacd0:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
root@e4e8230bacd0:#
root@e4e8230bacd0:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@e4e8230bacd0:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target    prot opt source          destination
Chain FORWARD (policy ACCEPT)
target    prot opt source          destination
ACCEPT    all  --  10.9.0.5      0.0.0.0/0           limit: avg 10/min burst 5
DROP     all  --  10.9.0.5      0.0.0.0/0
Chain OUTPUT (policy ACCEPT)
target    prot opt source          destination
root@e4e8230bacd0:#

```

Then I test the rule by pinging the same way once again. As you can see this time we got the expected result. The limit was set to 10 packets per minute which is equivalent to about 1 packet every six seconds. And this is exactly what we see highlighted in yellow as it roughly takes 6 seconds before the next packet comes back.

```

64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.133 ms
^C
--- 192.168.60.5 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16376ms
rtt min/avg/max/mdev = 0.121/0.202/1.219/0.254 ms
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/#
root@91b0395733b9:/# 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=0.450 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.143 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.124 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.122 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.184 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.128 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.505 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.128 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.132 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.141 ms
64 bytes from 192.168.60.5: icmp_seq=37 ttl=63 time=0.133 ms
64 bytes from 192.168.60.5: icmp_seq=43 ttl=63 time=0.134 ms
64 bytes from 192.168.60.5: icmp_seq=48 ttl=63 time=0.125 ms
^C
--- 192.168.60.5 ping statistics ---
49 packets transmitted, 13 received, 73.4694% packet loss, time 49150ms
rtt min/avg/max/mdev = 0.122/0.188/0.505/0.124 ms
root@91b0395733b9:/# lab6_seed_PKI

```

Right Ctrl  
4:05 PM  
2024-03-14

Based on our observations we can conclude that the first rule only works as expected when combined with the second rule. This is because while the rule is in place, packets are not forced to adhere to them. However, when the second rule is incorporated, packets have no choice but adhere to the first rule if they intend to cross to the internal network.

## Task 5: Load Balancing

**Using the nth mode (round-robin).**

As per the instructions, we will first start the netcat server on host 1 (192.168.60.5), host 2, (192.168.60.6), and host 3 (192.168.60.7).

```

root@5107486143de:/#
root@5107486143de:/# ip --br a
lo          UNKNOWN      127.0.0.1/8
eth0@if18   UP          192.168.60.5/24
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/# nc -luk 8080
lab6_seed_PKI

```

Right Ctrl  
4:57 PM  
2024-03-14

```

Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~/.../Labsetup> docksh host2-192.168.60.6
root@cf490d997b7c:/# ip --br a
lo          UNKNOWN      127.0.0.1/8
eth0@if16   UP          192.168.60.6/24
root@cf490d997b7c:/# nc -luk 8080

```

Right Ctrl  
4:58 PM  
2024-03-14

Then we add the following rule. This time we use the nat table because we need to change the target and destination ports of the IP packets which requires us to use network address translation (NAT). This rule will apply to all UDP packets going to port 8080 where for every three packets, we pick the packet 0, change its destination IP address and port number to 192.168.60.5 and 8080, respectively.

```
root@4e8230bacd0:/# iptables -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
root@4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-d
estination 192.168.60.5:8080
root@4e8230bacd0:/#
```

After setting this rule, we sent a UDP packet to the router's 8080 port from host A (10.9.0.5).

```
root@91b0395733b9:/#  
root@91b0395733b9:/#  
root@91b0395733b9:/# echo hello | nc -u 10.9.0.11 8080  
^C  
root@91b0395733b9:/#  
  
_seed_PKI
```

We then saw that it was sent to it was only sent to host1(192.168.60.5) but not to host 2, or 3. This is because according to the rules it forwards every third UDP packet (starting from packet 0) with a destination port of 8080 to the IP address 192.168.60.5 on port 8080.

```
root@5107486143de:/#  
root@5107486143de:/# ip -br a  
blob UNKNOWN 127.0.0.1/8  
eth0@if18 UP 192.168.60.5/24  
root@5107486143de:/#  
root@5107486143de:/#  
root@5107486143de:/# nc -luk 8080  
hello  
  
5_seed_PKI
```

```
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~/.../Labsetup> docksh host2-192.168.60.6
root@cf490d997b7c:/# ip -br a
lo      UNKNOWN      127.0.0.1/8
eth0@if16    UP      192.168.60.6/24
root@cf490d997b7c:/# nc -luk 8080
```

Now we will add more rules such that the three internal hosts get an equal number of packets. With every *third* UDP packet with a destination port of 8080 to the IP address *192.168.60.5 on port 8080*, every *second* UDP packet with a destination port of 8080 to the IP address *192.168.60.6 on port 8080*, and *every* UDP packet with a destination port of 8080 to the IP address *192.168.60.7 on port 8080*.

```
root@e4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-d  
estination 192.168.60.5:8080  
root@e4e8230bacd0:/#  
root@e4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-d  
estination 192.168.60.6:8080  
root@e4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 1 --packet 0 -j DNAT --to-d  
estination 192.168.60.7:8080
```

```
root@e4e8230bacd0:/# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target    prot opt source          destination
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 3 to:192.168.60.5:8080
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 2 to:192.168.60.6:8080
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 1 to:192.168.60.7:8080

Chain INPUT (policy ACCEPT)
target    prot opt source          destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source          destination
c) DOCKER_OUTPUT all  --  0.0.0.0/0      127.0.0.11

Chain POSTROUTING (policy ACCEPT)
target    prot opt source          destination
c) DOCKER_POSTROUTING all  --  0.0.0.0/0      127.0.0.11

Chain DOCKER_OUTPUT (1 references)
target    prot opt source          destination
DNAT      tcp  --  0.0.0.0/0      127.0.0.11      tcp dpt:53 to:127.0.0.11:32925
DNAT      udp  --  0.0.0.0/0      127.0.0.11      udp dpt:53 to:127.0.0.11:36675

Chain DOCKER_POSTROUTING (1 references)
target    prot opt source          destination
m) target    prot opt source          destination
SNAT      tcp  --  127.0.0.11     0.0.0.0/0      tcp spt:32925 to::53
SNAT      udp  --  127.0.0.11     0.0.0.0/0      udp spt:36675 to::53
```

Now, we will test these rules by once again sending a UDP packet to the router's 8080 port from host A (10.9.0.5). The first packet seems to have arrived to host1 but not at host 2 and .

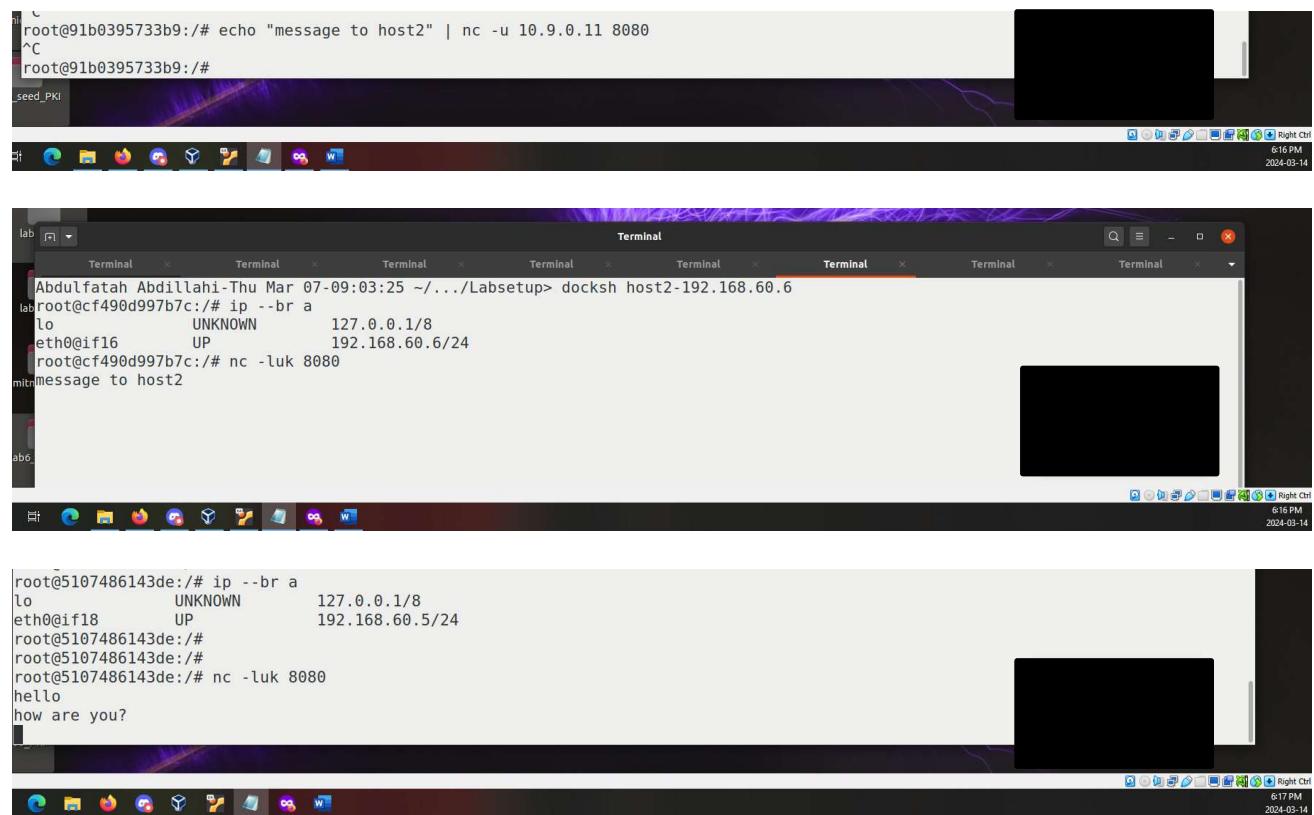


```
root@91b0395733b9:/# echo "how are you?" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/#
o6_seed_PKI
```

```
root@5107486143de:/# nc -luk 8080
hello
how are you?
```

The screenshot shows two separate Linux desktop environments. The top window is titled 'o6\_seed\_PKI' and the bottom window is also titled 'o6\_seed\_PKI'. Both windows show terminal sessions. The top terminal has sent a UDP packet to host1 (10.9.0.11) on port 8080 and received a response. The bottom terminal is listening on port 8080 and has received the same message.

The second packet has arrived at host2 but not host 1 and 3.

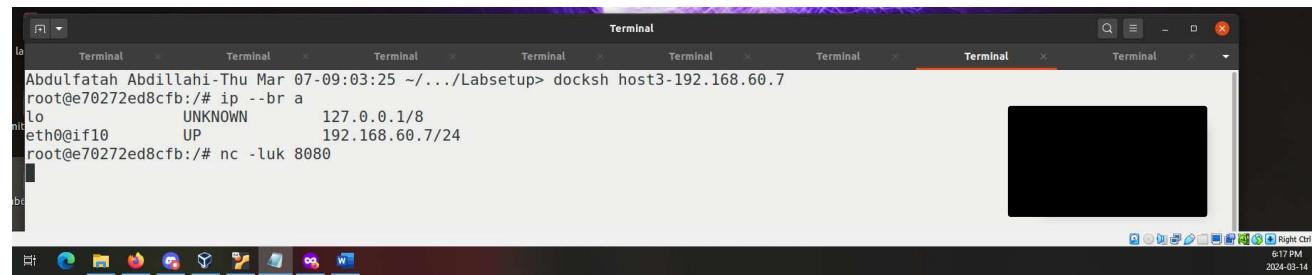


```
root@91b0395733b9:/# echo "message to host2" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/#
seed_PKI
```

```
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~/.../Labsetup> docksh host2-192.168.60.6
labroot@cf490d997b7c:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if16   UP        192.168.60.6/24
root@cf490d997b7c:/# nc -luk 8080
mit message to host2
```

```
root@5107486143de:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if18   UP        192.168.60.5/24
root@5107486143de:/#
root@5107486143de:/# nc -luk 8080
hello
how are you?
```

The screenshot shows three Linux desktop environments. The top window is titled 'lab' and contains multiple terminal tabs. The middle window is titled 'ab6' and the bottom window is also titled 'ab6'. All three windows show terminal sessions. The middle terminal has sent a UDP packet to host2 (192.168.60.6) on port 8080 and received a response. The bottom terminal is listening on port 8080 and has received the same message.

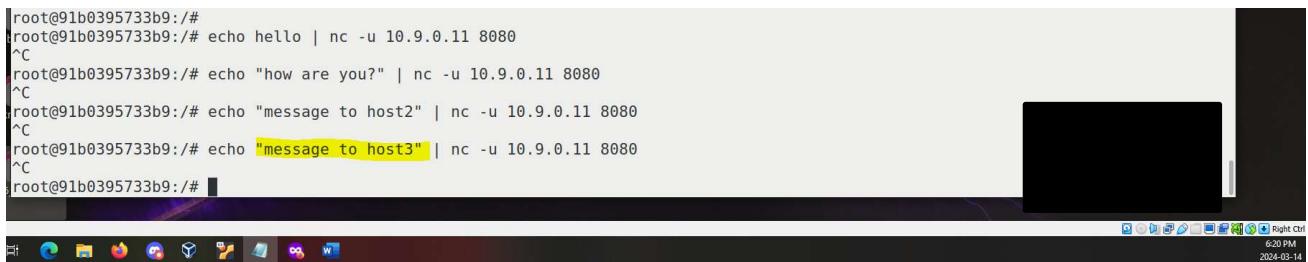


```
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~/.../Labsetup> docksh host3-192.168.60.7
root@e70272ed8cfb:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if10   UP        192.168.60.7/24
root@e70272ed8cfb:/# nc -luk 8080
```

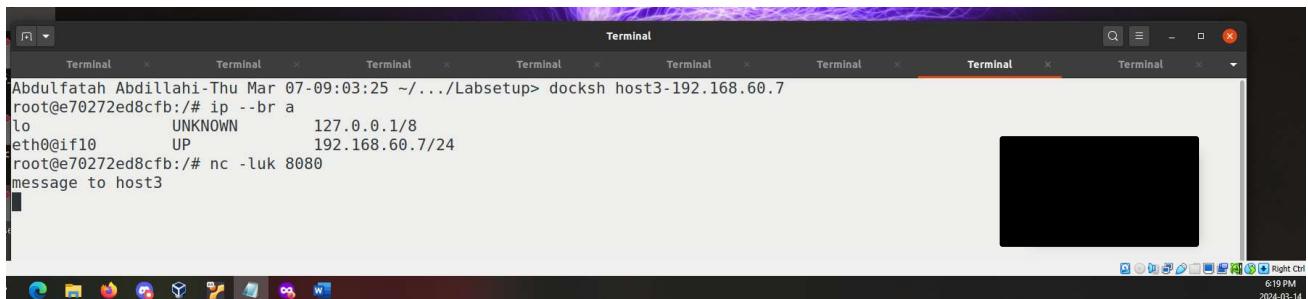
The screenshot shows three Linux desktop environments. The top window is titled 'la' and contains multiple terminal tabs. The middle window is titled 'bs' and the bottom window is also titled 'bs'. All three windows show terminal sessions. The middle terminal has sent a UDP packet to host3 (192.168.60.7) on port 8080 and received a response. The bottom terminal is listening on port 8080 and has received the same message.

The third packet has arrived at host3 but not host 1 and 2.

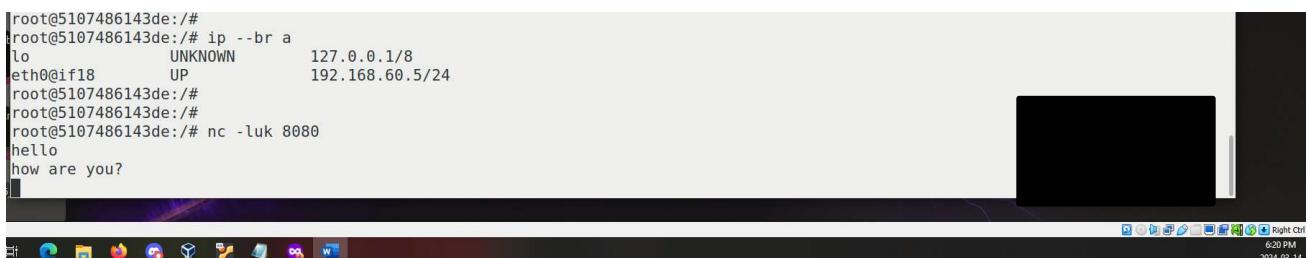
```
root@91b0395733b9:/#  
root@91b0395733b9:/# echo hello | nc -u 10.9.0.11 8080  
^C  
root@91b0395733b9:/# echo "how are you?" | nc -u 10.9.0.11 8080  
^C  
root@91b0395733b9:/# echo "message to host2" | nc -u 10.9.0.11 8080  
^C  
root@91b0395733b9:/# echo "message to host3" | nc -u 10.9.0.11 8080  
^C  
root@91b0395733b9:/#
```



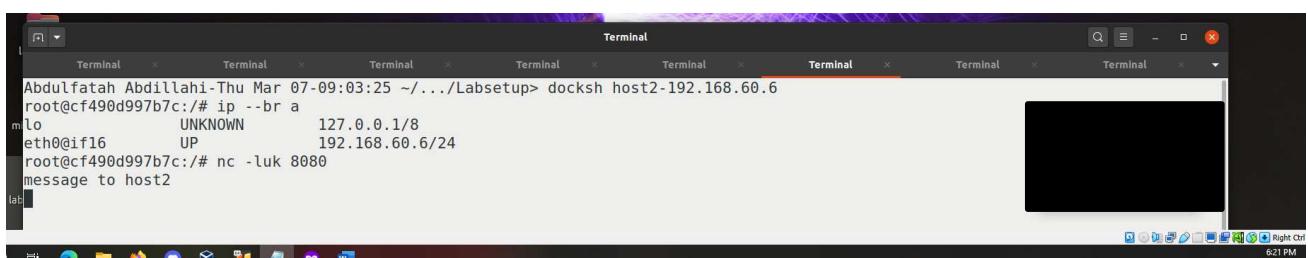
```
Terminal  
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~.../Labsetup> docksh host3-192.168.60.7  
root@e70272ed8cfb:/# ip --br a  
lo UNKNOWN 127.0.0.1/8  
eth0@if10 UP 192.168.60.7/24  
root@e70272ed8cfb:/# nc -luk 8080  
message to host3
```



```
root@5107486143de:/#  
root@5107486143de:/# ip --br a  
lo UNKNOWN 127.0.0.1/8  
eth0@if18 UP 192.168.60.5/24  
root@5107486143de:/#  
root@5107486143de:/# nc -luk 8080  
hello  
how are you?
```



```
Terminal  
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~.../Labsetup> docksh host2-192.168.60.6  
root@cf490d997b7c:/# ip --br a  
lo UNKNOWN 127.0.0.1/8  
eth0@if16 UP 192.168.60.6/24  
root@cf490d997b7c:/# nc -luk 8080  
message to host2
```

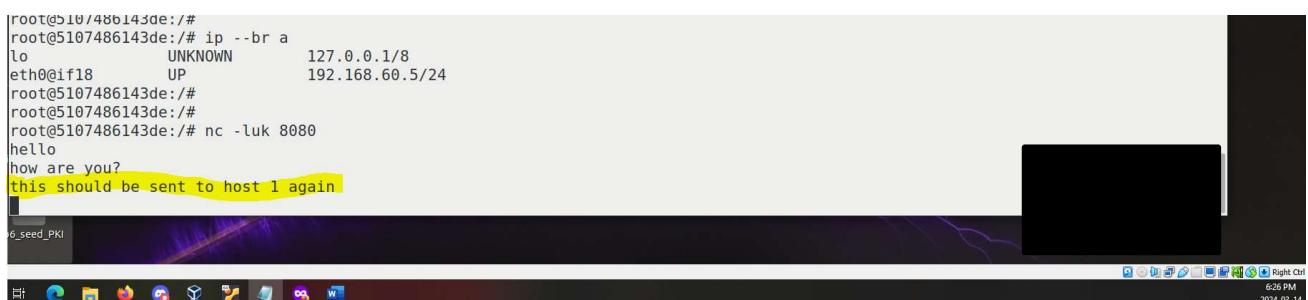


Finally, just to confirm, if everything is working fine then the next UDP packet should be sent to host1. We can see host 1 received the message but host 2 and 3 did not.

```
root@91b0395733b9:/# echo "this should be sent to host 1 again" | nc -u 10.9.0.11 8080  
^C  
root@91b0395733b9:/#
```



```
root@5107486143de:/#  
root@5107486143de:/# ip --br a  
lo UNKNOWN 127.0.0.1/8  
eth0@if18 UP 192.168.60.5/24  
root@5107486143de:/#  
root@5107486143de:/# nc -luk 8080  
hello  
how are you?  
this should be sent to host 1 again
```



**Clean the rules – must be careful when deleting rules here since the Docker network also uses the nat table.**

```
root@e4e8230bacd0:/#  
root@e4e8230bacd0://#  
root@e4e8230bacd0://# iptables -t nat -L -n --line-numbers  
Chain PREROUTING (policy ACCEPT)  
num target prot opt source destination  
1 DNAT    udp  --  0.0.0.0/0    0.0.0.0/0      udp dpt:8080 statistic mode nt  
2 DNAT    udp  --  0.0.0.0/0    0.0.0.0/0      udp dpt:8080 statistic mode nt  
3 DNAT    udp  --  0.0.0.0/0    0.0.0.0/0      udp dpt:8080 statistic mode nt  
ed_PKI
```

```
root@e4e8230bacd0:/# iptables -t nat -D PREROUTING 3
root@e4e8230bacd0:/# iptables -t nat -D PREROUTING 2
root@e4e8230bacd0:/# iptables -t nat -D PREROUTING 1
root@e4e8230bacd0:/# iptables -t nat -L -n --line-numbers
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source          destination
id

Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination

seed_PKI
```

### **Using the random mode.**

In this section we will reapply the rules to achieve load balancing in a different mode such that it will select a matching packet with the probability  $\langle P \rangle$ . You will notice that unlike in the previous rules, the below rules uses mode random along with different probabilities. For example, in the first rule since we want every third packet to be dispatched to host 1, we set the probability to  $(1/3) 0.3333$ . Now, one-third is dispatched so we only have 2 left so the probability for the second rule is  $(1/2) 0.5$  and this will be sent to host 2. Now, we have only 1 packet left to be distributed to host 3 so this time the probability is  $(1/1) 1$ .

```

root@e4e8230bacd0:/#
root@e4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.3333 -j DNAT --to-d o-destination 192.168.60.5:8080
root@e4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-d estimation 192.168.60.6:8080
root@e4e8230bacd0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1 -j DNAT --to-d estimation 192.168.60.7:8080
root@e4e8230bacd0:/#

```

ed\_PKI

Right Ctrl  
6:58 PM  
2024-03-14

```

Terminal Terminal
root@e4e8230bacd0:/# iptables -t nat -L -n --line-numbers
Chain PREROUTING (policy ACCEPT)
num target prot opt source destination
1 DNAT    udp  --  0.0.0.0/0  0.0.0.0/0      udp dpt:8080 statistic mode random probability 0.33330000006 to:
192.168.60.5:8080
2 DNAT    udp  --  0.0.0.0/0  0.0.0.0/0      udp dpt:8080 statistic mode random probability 0.50000000000 to:
192.168.60.6:8080
3 DNAT    udp  --  0.0.0.0/0  0.0.0.0/0      udp dpt:8080 statistic mode random probability 1.00000000000 to:
192.168.60.7:8080

Chain INPUT (policy ACCEPT)
num target prot opt source destination

Chain OUTPUT (policy ACCEPT)
num target prot opt source destination
1 DOCKER_OUTPUT all  --  0.0.0.0/0  127.0.0.11

Chain POSTROUTING (policy ACCEPT)
num target prot opt source destination
1 DOCKER_POSTROUTING all  --  0.0.0.0/0  127.0.0.11

Chain DOCKER_OUTPUT (1 references)
num target prot opt source destination
1 DNAT    tcp  --  0.0.0.0/0  127.0.0.11      tcp dpt:53 to::127.0.0.11:32925
2 DNAT    udp  --  0.0.0.0/0  127.0.0.11      udp dpt:53 to::127.0.0.11:36675

Chain DOCKER_POSTROUTING (1 references)
num target prot opt source destination
1 SNAT    tcp  --  127.0.0.11  0.0.0.0/0      tcp spt:32925 to:::53
2 SNAT    udp  --  127.0.0.11  0.0.0.0/0      udp spt:36675 to:::53
root@e4e8230bacd0:/#

```

Now to test we will send some UDP packets again using netcat. The first message “*test1*” was received by host 2 but not host 1 and host 3. The same thing happened with the second message “*test2*”.

```

root@91b0395733b9:/#
root@91b0395733b9:/# echo "test1" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/#

```

ed\_PKI

Right Ctrl  
7:15 PM  
2024-03-14

```

Terminal Terminal
Abdulfatah Abdillahi-Thu Mar 07-09:03:25 ~/.../Labsetup> docksh host2-192.168.60.6
root@cf490d997b7c:/# ip -br a
lo           UNKNOWN      127.0.0.1/8
eth0@if16   UP          192.168.60.6/24
root@cf490d997b7c:/# nc -luk 8080
message to host2
test1
root@cf490d997b7c:/# 

```

ed\_PKI

Right Ctrl  
7:25 PM  
2024-03-14

```

root@91b0395733b9:/# echo "test1" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/# echo "test2" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/#

```

ed\_PKI

Right Ctrl  
7:28 PM  
2024-03-14

The third message “*test3*” was received by host 3 but not host 1 and host 2.

```
root@91b0395733b9:/# echo "test3" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/#
```

The fourth message “*test4*” was received by host 2 again but not host 1 and host 3. The fifth message “*test5*” however, was **received by host1**.

```
root@91b0395733b9:/# echo "test1" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/# echo "test2" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/# echo "test3" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/# echo "test4" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/# echo "test5" | nc -u 10.9.0.11 8080
^C
root@91b0395733b9:/#
```

```
root@5107486143de:/# ip --br a
lo          UNKNOWN      127.0.0.1/8
eth0@if18    UP          192.168.60.5/24
root@5107486143de:/#
root@5107486143de:/#
root@5107486143de:/# nc -luk 8080
hello
how are you?
this should be sent to host 1 again
test5
```

Based on the observations, and since the mode is set to random, you will notice that the pattern is not visible in the short term as each internal server is not getting the exact same number of packets, but this should instead be visible in the long term (100 packets or more).

## References

[https://seedsecuritylabs.org/Labs\\_20.04/Files/Firewall/Firewall.pdf](https://seedsecuritylabs.org/Labs_20.04/Files/Firewall/Firewall.pdf)

[https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc\\_make.html](https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html)

<https://www.youtube.com/watch?v=OlBrvLm7m1w>

<https://youtu.be/LEgwowsmSPg>

<https://youtu.be/NAdJojxEIU>

<https://youtu.be/-CraNvj48J0>

Du, W. (2022). Internet Security: A Hands-on Approach (Third edition). Independent.