Internet Security

Firewall Lab

Name: Gaurav Upadhyay

Email: gsupadhy@syr.edu

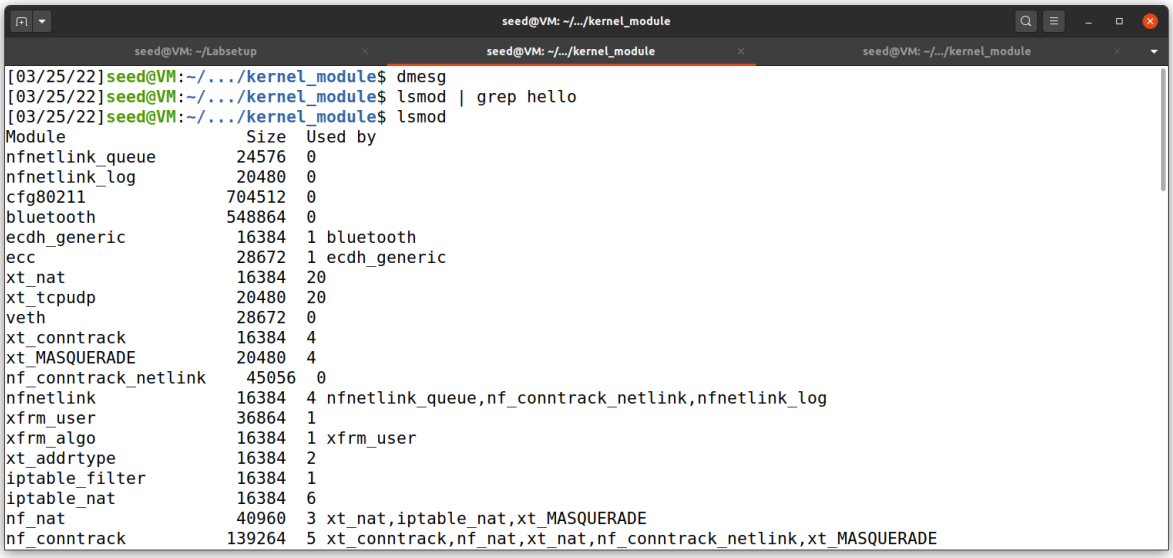
Task 1: Implementing a Simple Firewall

Task 1.A: Implement a Simple Kernel Module

First, we go into kernel module folder and compile the files into a loadable kernel module using the make command.

```
[03/25/22]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Labsetup/Files/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Labsetup/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Labsetup/Files/kernel_module/hello.mod.o
  LD [M]  /home/seed/Labsetup/Files/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/25/22]seed@VM:~/.../kernel_module$ ls
hello.c  hello.ko  hello.mod  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
```

We display the list of modules:



Now, we insert the following hello.ko module and list it below:

```
[03/25/22]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[03/25/22]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[03/25/22]seed@VM:~/.../kernel_module$
```

We make use of dmesg command to check the messages created by the module:

```
[03/25/22]seed@VM:~/.../kernel_module$ dmesg
[31362.376470] Hello World!
[03/25/22]seed@VM:~/.../kernel_module$
```

Now, we remove the hello.ko module as follows and check the message using dmesg:

```
[03/25/22]seed@VM:~/.../kernel_module$ sudo rmmod hello.ko
[03/25/22]seed@VM:~/.../kernel_module$
```

```
[03/25/22]seed@VM:~/.../kernel_module$ dmesg
[31362.376470] Hello World!
[31416.932698] Bye-bye World!.
[03/25/22]seed@VM:~/.../kernel_module$
```

Task 1.B: Implement a Simple Firewall Using Netfilter

1. I compiled the seedFileter.c file using the make command as follows:

```
[03/25/22]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Labsetup/Files/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Labsetup/Files/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Labsetup/Files/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/25/22]seed@VM:~/.../packet_filter$ ls
Makefile       Module.symvers  seedFilter.ko   seedFilter.mod.c  seedFilter.o
modules.order  seedFilter.c    seedFilter.mod  seedFilter.mod.o
```

We use the following command to generate UDP packets to 8.8.8.8 which is a Google's DNS server:

WE can see it is reachable before making the kernel module

Now we insert the kernel module and check for the message:

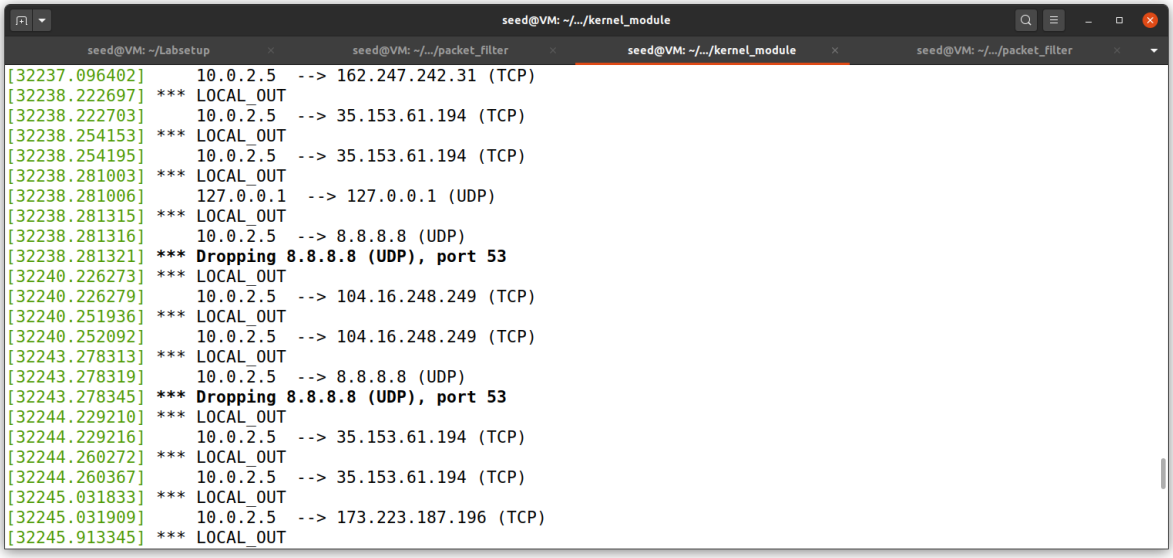
```
[03/25/22]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[03/25/22]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
[03/25/22]seed@VM:~/.../packet_filter$
```

```
[03/25/22]seed@VM:~/.../kernel_module$ dmesg -k -w
[31362.376470] Hello World!
[31416.932698] Bye-bye World!.
[32055.822650] Registering filters.
[32057.963648] *** LOCAL_OUT
[32057.963654]     10.0.2.5  --> 35.153.61.194 (TCP)
[32057.997076] *** LOCAL_OUT
[32057.997118]     10.0.2.5  --> 35.153.61.194 (TCP)
[32058.977144] *** LOCAL_OUT
[32058.977150]     10.0.2.5  --> 104.16.248.249 (TCP)
[32059.002778] *** LOCAL_OUT
```

Now we again try to reach the Google's DNS server and we can see that it is unreachable which means our firewall is working as specified.

```
[03/25/22]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

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

This can be shown when the UDP packets are retried to picked three times in total before getting dropped:



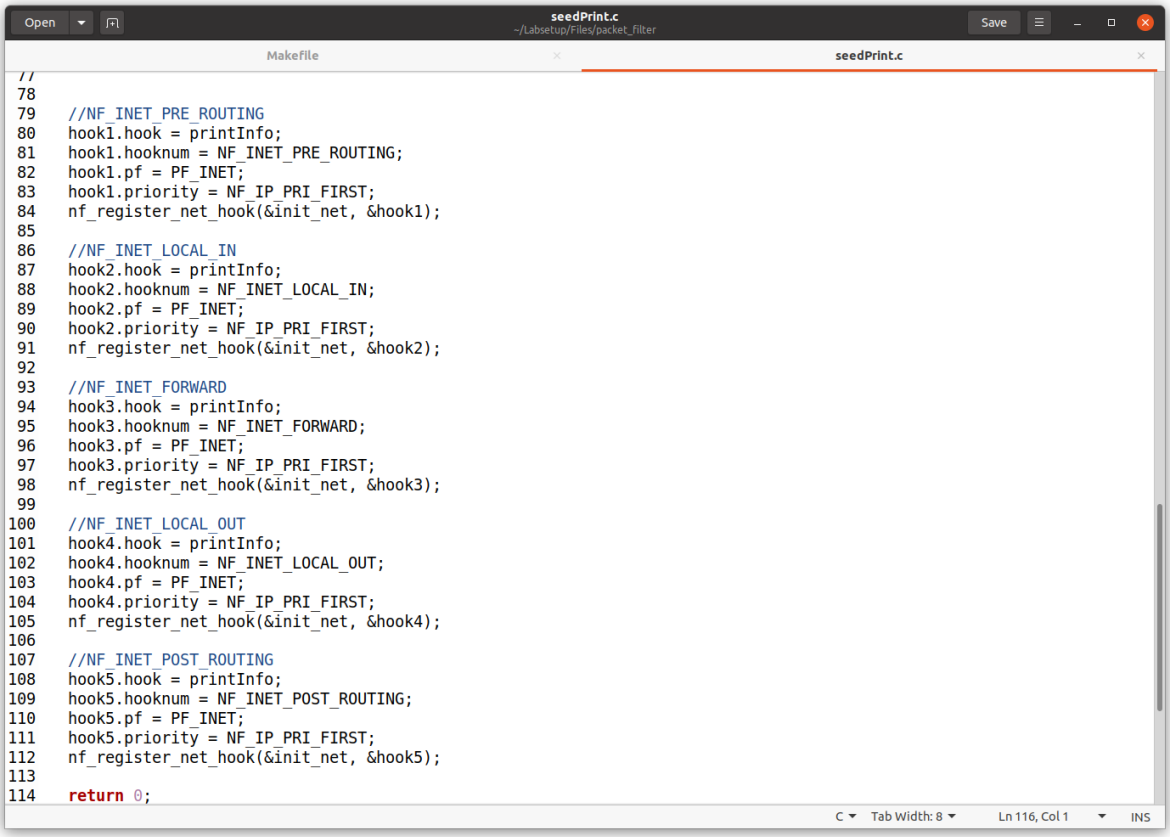
2. For this task we create a new file named seedPrint and add its executable kernel to the Maekefile:

```
#obj-m += seedFilter.o
obj-m += seedPrint.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 seedFilter.ko

rm:
	sudo rmmod seedFilter
```

Now we make changes in the code to add more hooks as follows:

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

For the following added hooks in their corresponding hook functions as follows:



Once you exit this module, we also need to unregister the hooks:

```
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);
}
```

After sabong the file, we compile the fine into a kernel module using make command and insert the seedPrint kernel module as follows:

```
[03/25/22]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Labsetup/Files/packet_filter/seedPrint.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Labsetup/Files/packet_filter/seedPrint.mod.o
  LD [M]  /home/seed/Labsetup/Files/packet_filter/seedPrint.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/25/22]seed@VM:~/.../packet_filter$ sudo insmod seedPrint.ko
```

We get the following messages when the kernel is registered:

```
[32294.403002] *** LOCAL_OUT
[32294.403003]     10.0.2.5  --> 104.16.248.249 (TCP)
[32294.757307] The filters are being removed.
[33241.535455] seedPrint file: Registering filters.
```

Again we make use of the dig command to check the generated UDP packets and the different functions being invoked:

We get the following messages on the dmesg live commands:

```
[33699.436583] *** LOCAL_OUT
[33699.436584]     10.0.2.5  --> 8.8.8.8 (UDP)
[33699.436591] *** POST_ROUTING
[33699.436592]     10.0.2.5  --> 8.8.8.8 (UDP)
[33699.460435] *** PRE_ROUTING
[33699.460461]     8.8.8.8  --> 10.0.2.5 (UDP)
[33699.460476] *** LOCAL_IN
[33699.460482]     8.8.8.8  --> 10.0.2.5 (UDP)
```

We can see that the LOCAL_OUT, LOCAL_IN, POST_ROUTING and PRE_ROUTING functions were invoked as the UDP packets were generated except the FORWARD function. Now we remove the module:

```
[03/25/22]seed@VM:~/.../packet_filter$ sudo rmmod seedPrint.ko
[03/25/22]seed@VM:~/.../packet_filter$
```

It gives us the following output while being unregistered:

```
[33787.344595]     10.0.2.5  --> 104.16.248.249 (TCP)
[33790.063378] seedPrint file: The filters are being removed.
```

3. For this task we will create another new file named seedBlock as follows:

```
[03/27/22]seed@VM:~/.../packet_filter$ make clean
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Labsetup/Files/packet_filter clean
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CLEAN   /home/seed/Labsetup/Files/packet_filter/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/27/22]seed@VM:~/.../packet_filter$ ls
Makefile  seedFilter.c  seedPrint.c
[03/27/22]seed@VM:~/.../packet_filter$ cp seedFilter.c seedBlock.c
[03/27/22]seed@VM:~/.../packet_filter$ ls
Makefile  seedBlock.c  seedFilter.c  seedPrint.c
[03/27/22]seed@VM:~/.../packet_filter$ gedit seedBlock.c
[03/27/22]seed@VM:~/.../packet_filter$ ls
Makefile  seedBlock.c  seedFilter.c  seedPrint.c
```

In this new file, we will have two separate functions:

1. preventing other computers to ping the VM, and
2. preventing other computers to telnet into the VM.

They are implemented as follows:

Blocking ping to VM:

```c
//blocking to 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;
   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 (UDP)\n", &(iph->daddr));
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

Blocking ping to 8.8.8.8:53

```c
//blocking 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;
   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;
}
```

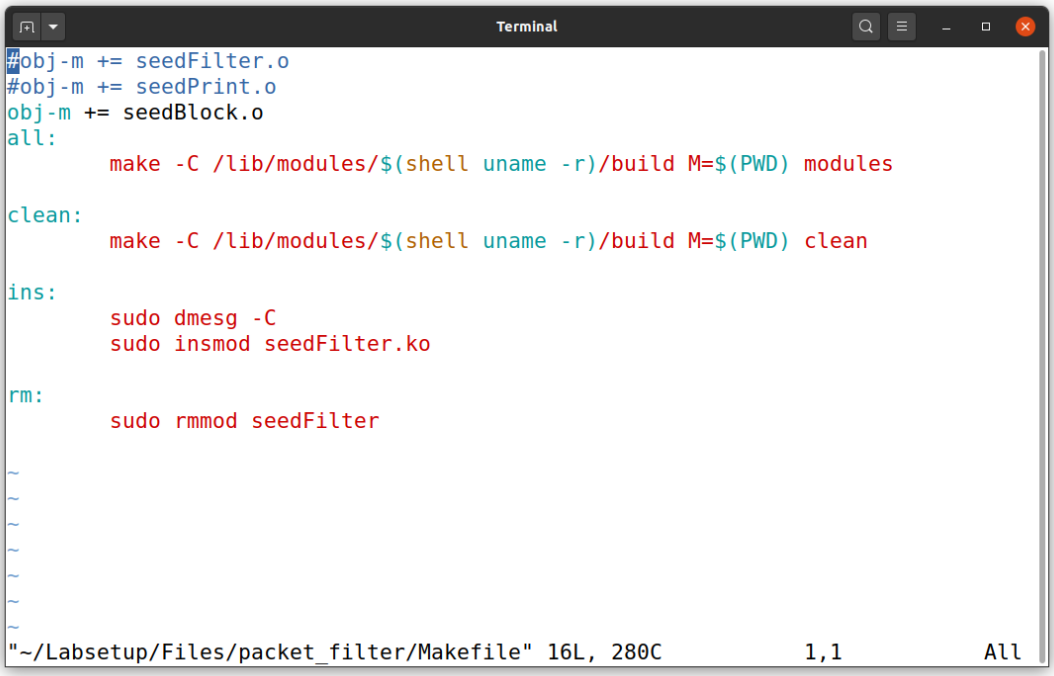
Blocking ping to Telnet to VM

```
//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;
   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 (UDP), port %d\n", &(iph->daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

Now, we register our hooks to the functions we created above along with unregistering them:

```
    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);
}
```

We also make the changes in our makefile to add the kernel module object file to be created:



Now we make our module and insert the kernel module as follows:

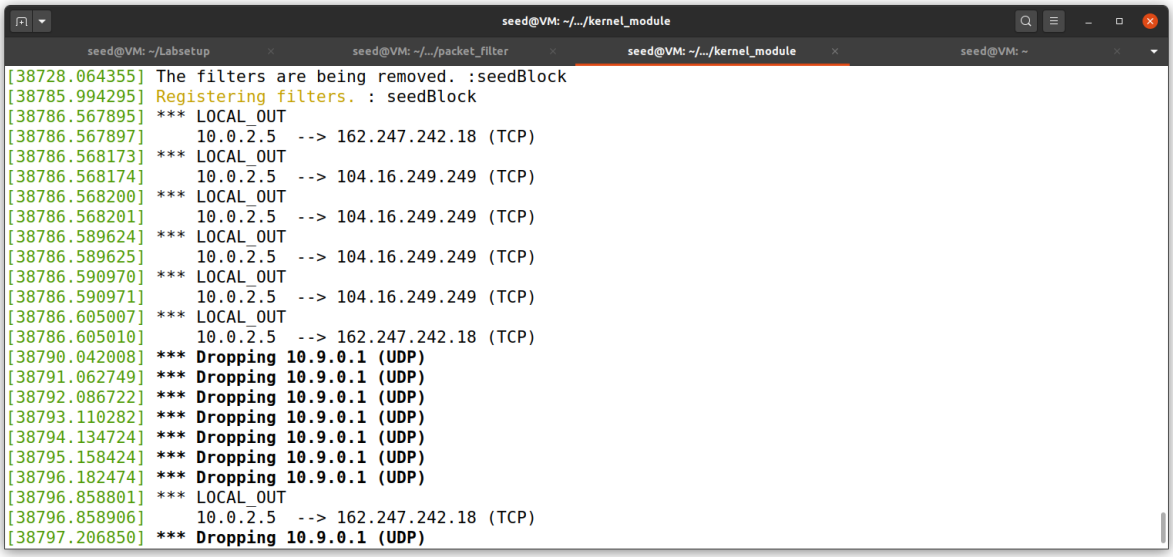
```
[03/27/22]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Labsetup/Files/packet_filter/seedBlock.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Labsetup/Files/packet_filter/seedBlock.mod.o
  LD [M]  /home/seed/Labsetup/Files/packet_filter/seedBlock.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/27/22]seed@VM:~/.../packet_filter$ ls
Makefile       Module.symvers  seedBlock.ko   seedBlock.mod.c  seedBlock.o   seedPrint.c
modules.order  seedBlock.c     seedBlock.mod  seedBlock.mod.o  seedFilter.c
[03/27/22]seed@VM:~/.../packet_filter$ sudo insmod seedBlock.ko
[03/27/22]seed@VM:~/.../packet_filter$
```

We can see the message being displayed using dmesg which shows that the filters are registered using the changes we made:

```
[38664.995938] Registering filters. : seedBlock
[38666.564189] *** LOCAL_OUT
[38666.564196]     10.0.2.5  --> 162.247.242.18 (TCP)
```

Now we try to ping into the VM and can see that the UDP packets are getting dropped:

```
root@60bdc5ea58f6:/# 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 ---
12 packets transmitted, 0 received, 100% packet loss, time 11265ms
```



We try to telnet into the VM and can see that it is not successful:

```
root@60bdc5ea58f6:/# telnet 10.9.0.1
Trying 10.9.0.1...
```

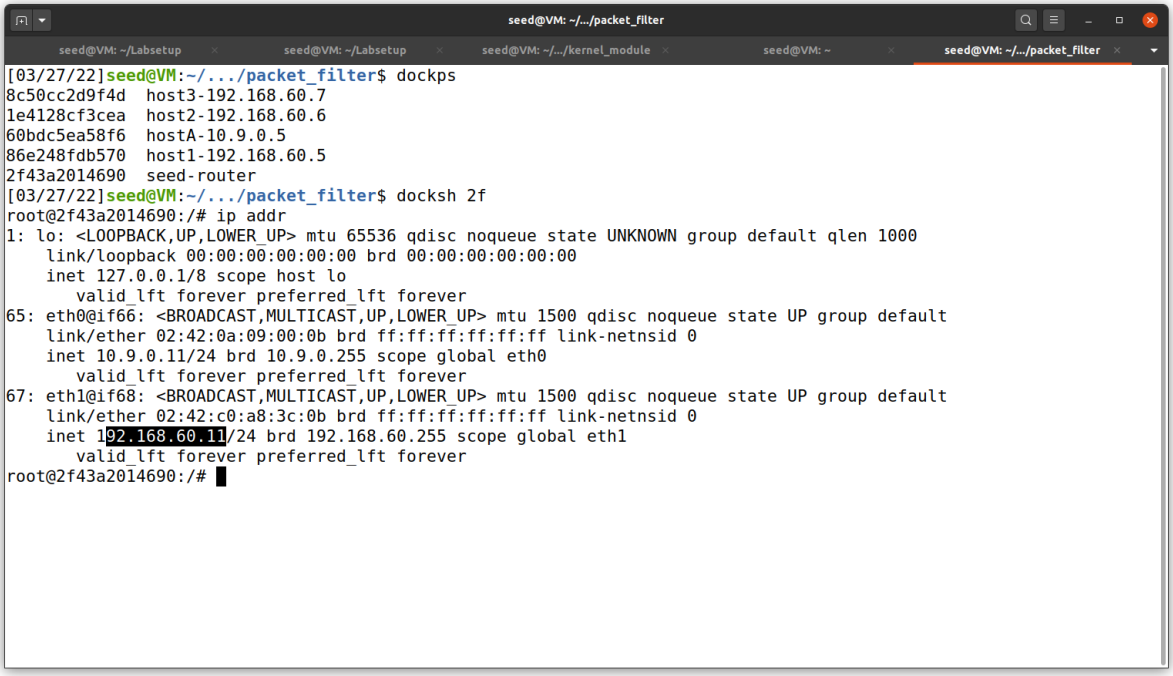
We can see the output as the UDP packets are being dropped:

```
[38927.640063] *** Dropping 10.9.0.1 (UDP), port 23
[38927.943659] *** LOCAL_OUT
[38927.943677]     10.0.2.5  --> 162.247.242.18 (TCP)
[38928.662957] *** Dropping 10.9.0.1 (UDP), port 23
[38929.392303] *** LOCAL_OUT
[38929.392307]     10.0.2.5  --> 35.184.35.160 (TCP)
[38929.439872] *** LOCAL_OUT
[38929.439875]     10.0.2.5  --> 35.184.35.160 (TCP)
[38930.679700] *** Dropping 10.9.0.1 (UDP), port 23
[38934.839301] *** Dropping 10.9.0.1 (UDP), port 23
[38935.354556] *** LOCAL_OUT
[38935.354561]     10.0.2.5  --> 3.15.76.60 (TCP)
[38938.167703] *** LOCAL_OUT
[38938.167768]     10.0.2.5  --> 162.247.242.18 (TCP)
```

Task 2: Experimenting with Stateless Firewall Rules

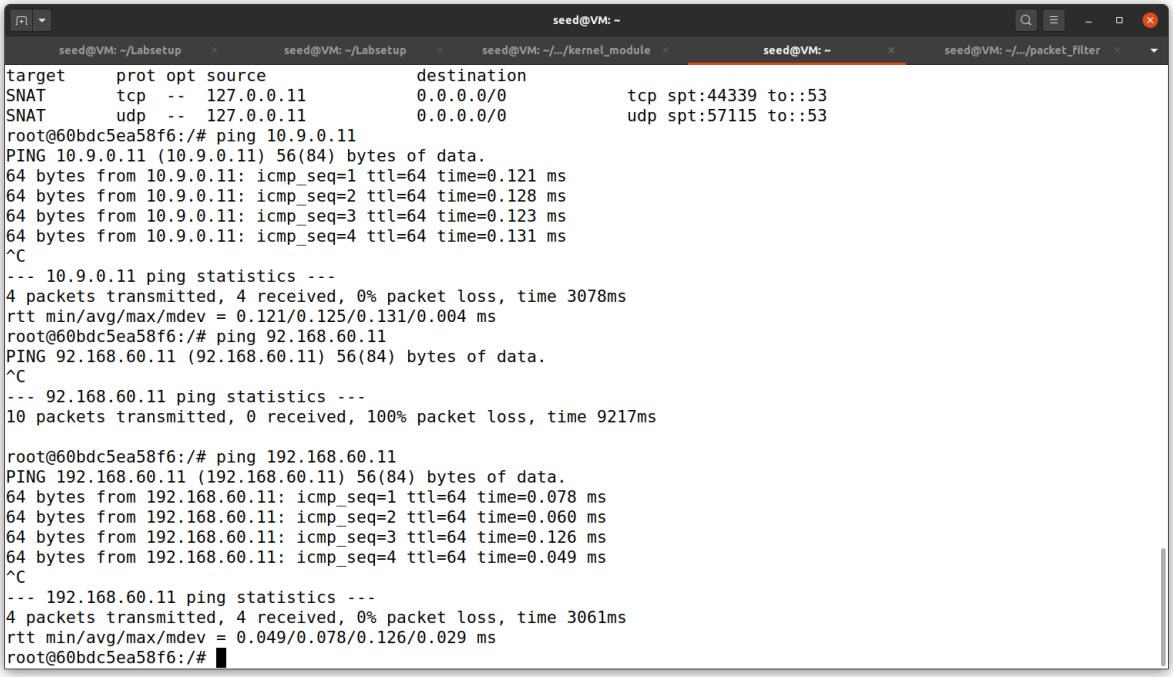
Task 2.A: Protecting the Router

For this task, we need to open the router container. The container information is as follows:

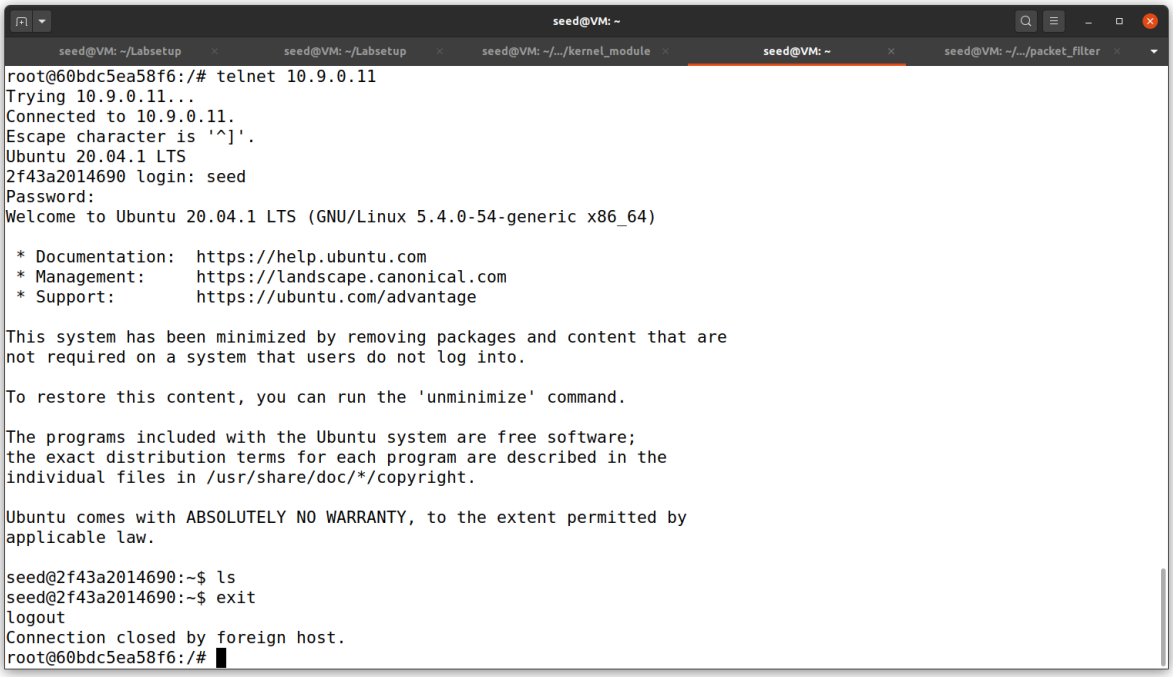


We can see that the two interfaces have two IPs. Eth0 has ip 10.9.0.11 while Eth1 has ip 192.168.60.11.

First, we will try to ping to interfaces eth0 and eth1 which can be seen as successful:



We can also use telnet into 10.9.0.11 which is also successful:



Before we run the rules, we check the current status of iptables:

```
root@2f43a2014690:/# 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
```

We set up the rules for preventing outside machines from accessing to router machine except ping as follows and we check the iptables status:

```
root@2f43a2014690:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@2f43a2014690:/# iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@2f43a2014690:/# iptables -P OUTPUT DROP
root@2f43a2014690:/# iptables -P INPUT DROP
root@2f43a2014690:/# 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
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
root@2f43a2014690:/#
```

1. iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

The following command accepts the echo requests as ping requests from ping and block all other machines.

2. iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

For this command, we also accept the ping reply to the user.

3. iptables -P OUTPUT DROP
4. iptables -P INPUT DROP

The following two commands are set to default for Output and Input which means other protocols will not have any access to the router.

Now, we test the rules by pinging and telnet which are denied as needed.

```
root@60bdc5ea58f6:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
^C
--- 192.168.60.11 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3073ms

root@60bdc5ea58f6:/# telnet 10.9.0.11
Trying 10.9.0.11...
^C
root@60bdc5ea58f6:/#
```

Before moving on to the next task, we restore the filter table to its original state:

```
root@2f43a2014690:/# iptables -F
root@2f43a2014690:/# iptables -P OUTPUT ACCEPT
root@2f43a2014690:/# iptables -P INPUT ACCEPT
root@2f43a2014690:/# 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@2f43a2014690:/#
```

Task 2.B: Protecting the Internal Network

Here, we check the status of iptables before setting the rule for dropping ICMP echo requests and the table status after they are applied:

```
root@2f43a2014690:/# 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@2f43a2014690:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
root@2f43a2014690:/# iptables  -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@2f43a2014690:/#
```

For the given rule, we can see that the outside hosts will not be able to ping the inside hosts because there will be a drop of the packets. Now we will test for the conditions:

The outside host can ping the router:

```
root@60bdc5ea58f6:/# 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=0.079 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.140 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.129 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3077ms
rtt min/avg/max/mdev = 0.058/0.101/0.140/0.034 ms
```

The outside host cannot ping the internal hosts:

```
root@60bdc5ea58f6:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4094ms

root@60bdc5ea58f6:/#
```

The internal host can ping the outside hosts:

```
root@86e248fdb570:/# 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=0.117 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.174 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.148 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.136 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.174 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.173 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.068 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=63 time=0.159 ms
^C
--- 10.9.0.5 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8172ms
rtt min/avg/max/mdev = 0.068/0.146/0.174/0.033 ms
```

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

For the following task, we need to generate a rule which accepts the packets sent outside from the internal host and vice versa as follows:

```
root@2f43a2014690:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
root@2f43a2014690:/# iptables  -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@2f43a2014690:/#
```

```
root@2f43a2014690:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
root@2f43a2014690:/# iptables  -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@2f43a2014690:/#
```

We can see the iptables as follows:

```
root@2f43a2014690:/# 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
    5   420 DROP       icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0           icmptype 8
    0     0 ACCEPT     icmp --  eth1   *       0.0.0.0/0            0.0.0.0/0           icmptype 8
    0     0 ACCEPT     icmp --  eth0   *       0.0.0.0/0            0.0.0.0/0           icmptype 0

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@2f43a2014690:/#
```

Now we test by pinging the external hosts with internal hosts and vice versa:

From the external host to internal host, it doesn't work:

```
root@60bdc5ea58f6:/# 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 ---
21 packets transmitted, 0 received, 100% packet loss, time 20483ms
```

From the internal host to external host, it works:

```
root@86e248fdb570:/# 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=0.117 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.174 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.148 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.136 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.174 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.173 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.068 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=63 time=0.159 ms
^C
--- 10.9.0.5 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8172ms
rtt min/avg/max/mdev = 0.068/0.146/0.174/0.033 ms
```

For other connection requests we make use of telnet:

```
root@60bdc5ea58f6:/# telnet 192.168.60.5
Trying 192.168.60.5...
^C
root@60bdc5ea58f6:/#
```

```
root@86e248fdb570:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@86e248fdb570:/#
```

We can see that the telnet from internal to external and vice versa both doesn't work at all which means our firewall is working as expected.

Before moving on to the next task, we restore the filter table to its original state:

```
root@2f43a2014690:/# iptables -F
root@2f43a2014690:/# iptables -P FORWARD  ACCEPT
root@2f43a2014690:/# 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
root@2f43a2014690:/#
```

Task 2.C: Protecting Internal Servers

For the following task, we write the following rules:

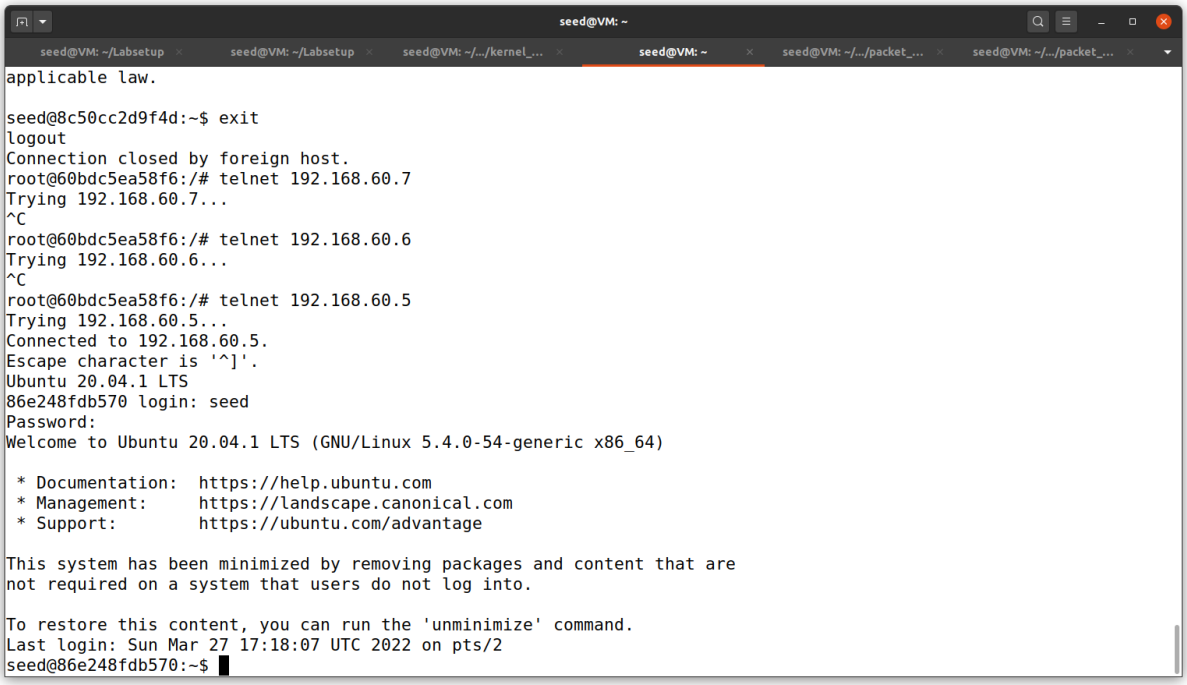
```
root@2f43a2014690:/# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
root@2f43a2014690:/# iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 --sport 23 -j ACCEPT
root@2f43a2014690:/# iptables -P FORWARD DROP
root@2f43a2014690:/# iptables  -L -n -v
Chain INPUT (policy ACCEPT 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
root@2f43a2014690:/#
```

Eth0 accepts the requests from outside to inside on the dest IP 192.168.60.5 and dport 23 only while Eth1 does the same but from inside to outside.

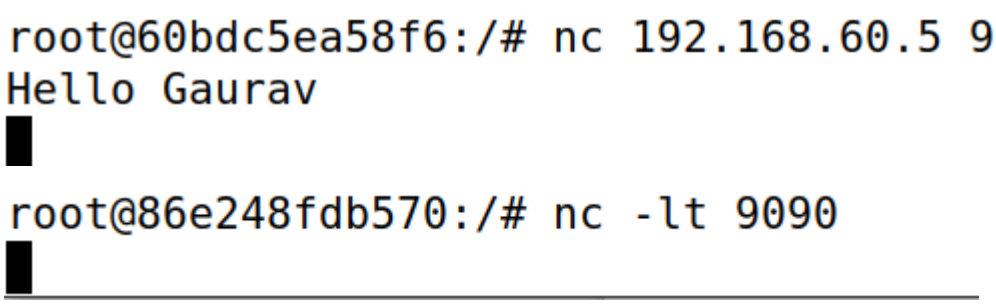
The third rule specifies that we drop any other packets not intended to the IP or port specified in the above rules.

Now, we test for the multiple conditions:

Here we try to telnet from external server to internal host such as 192.168.60.6 or 192.168.60.7 which does not work but only to 192.168.60.5 will work as we have specified in our rules.

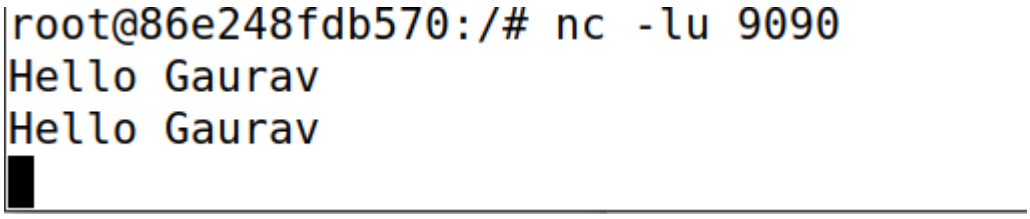


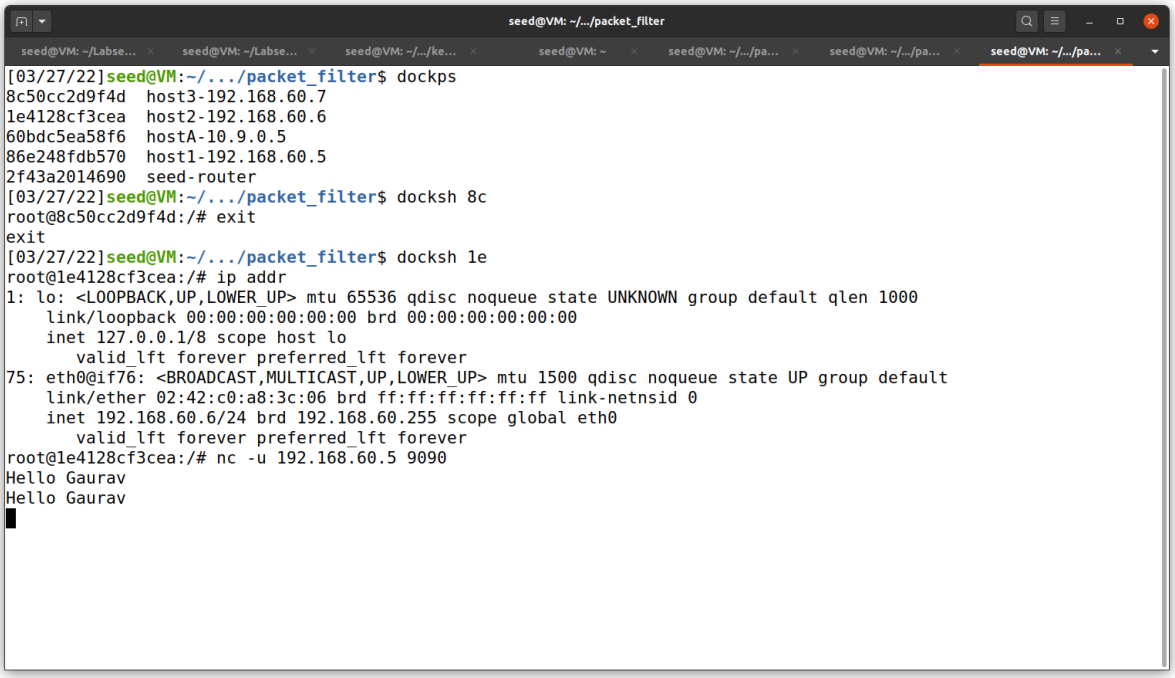
We try external host to connect to any internal host on a port number which also does not work as per intended:



We didn’t receive any connection.

Now we try to connect from an internal host to access the internal server and we see that it connects successfully:





Now, the internal host try to connect to external server which is not possible:

```
root@1e4128cf3cea:/# nc -t 10.9.0.5 9090
Hello gaurav!!

root@60bdc5ea58f6:/# nc -ltv 9090
Listening on 0.0.0.0 9090
```

Before moving on to the next task, we restore the filter table to its original state:

```
root@2f43a2014690:/# iptables -F
root@2f43a2014690:/# iptables -P FORWARD  ACCEPT
root@2f43a2014690:/# 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
root@2f43a2014690:/#
```

Task 3: Connection Tracking and Stateful Firewall

Task 3.A: Experiment with the Connection Tracking

ICMP experiment:

For the ICMP connection type we ping the 192.168.60.5 in the background to run continuously by suppressing it as follows:

```
root@b90e26a223f2:/# ping 192.168.60.5 &>/dev/null &
[1] 47
```

On the server router, we can check for the connection tracking information using the following command:

For the ICMP, the connection stays for about 5 to 6 seconds for each packet sent.

```
root@b90e26a223f2:/# conntrack -L
icmp     1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=35 src=192.168.60.5 dst=192.168.60.11 type=0 code
=0 id=35 mark=0 use=1
icmp     1 30 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=47 src=192.168.60.5 dst=192.168.60.11 type=0 code
=0 id=47 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
root@b90e26a223f2:/#
```

UDP experiment:

For this experiment, we need to create a netcat UDP server as follows:

```
[03/27/22]seed@VM:~/Labsetup$ docksh host1-192.168.60.5
root@d6b23c67b5ed:/# nc -lu 9090
hello Gaurav
```

```
[03/27/22]seed@VM:~/Labsetup$ docksh hostA-10.9.0.5
root@6418b95c2eca:/# nc -u 192.168.60.5 9090
hello Gaurav
```

From observations, we can say that the UDP connection stays for around 6-7 seconds for each packet connection tracking. This is done using the conntrack -L command:

```
root@b90e26a223f2:/# conntrack -L
icmp     1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=35 src=192.168.60.5 dst=192.168
.60.11 type=0 code=0 id=35 mark=0 use=1
udp      17 12 src=10.9.0.5 dst=192.168.60.5 sport=58061 dport=9090 [UNREPLIED] src=192.168.60.5
dst=10.9.0.5 sport=9090 dport=58061 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
root@b90e26a223f2:/# conntrack -L
icmp     1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=35 src=192.168.60.5 dst=192.168
.60.11 type=0 code=0 id=35 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@b90e26a223f2:/#
```

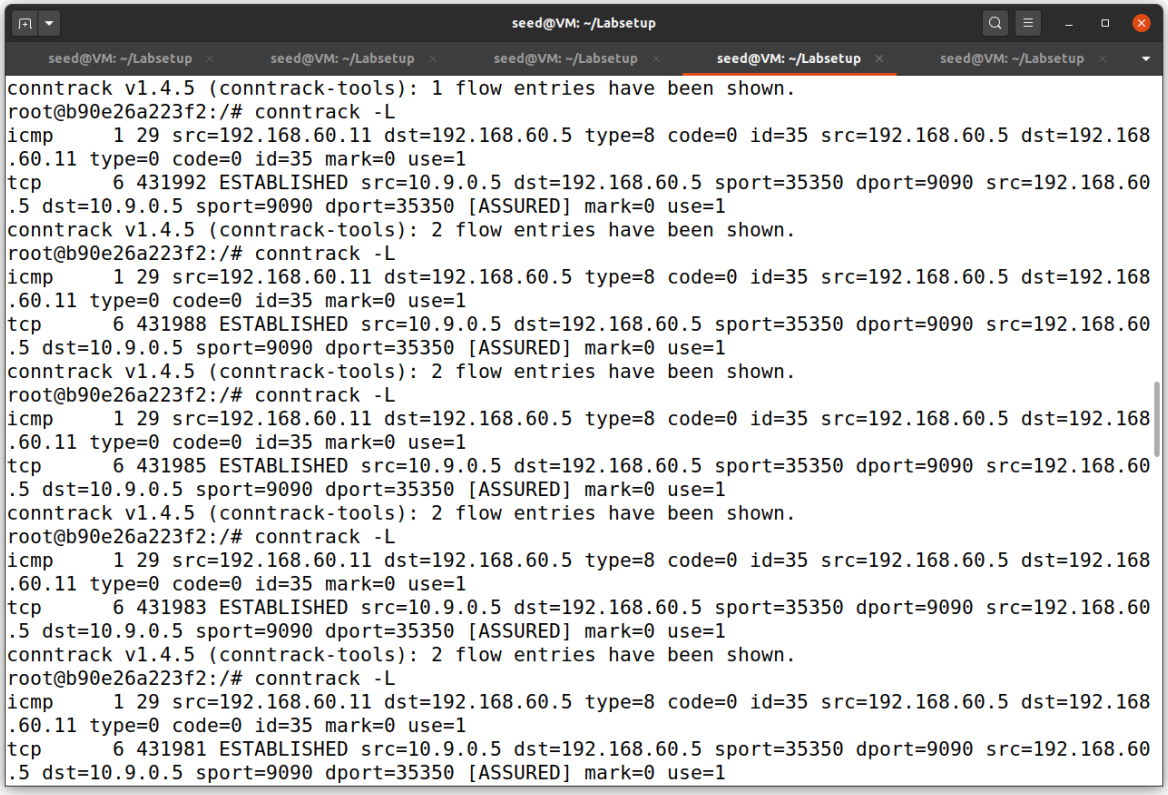
TCP experiment:

For this experiment, we create a TCP server on netcat:

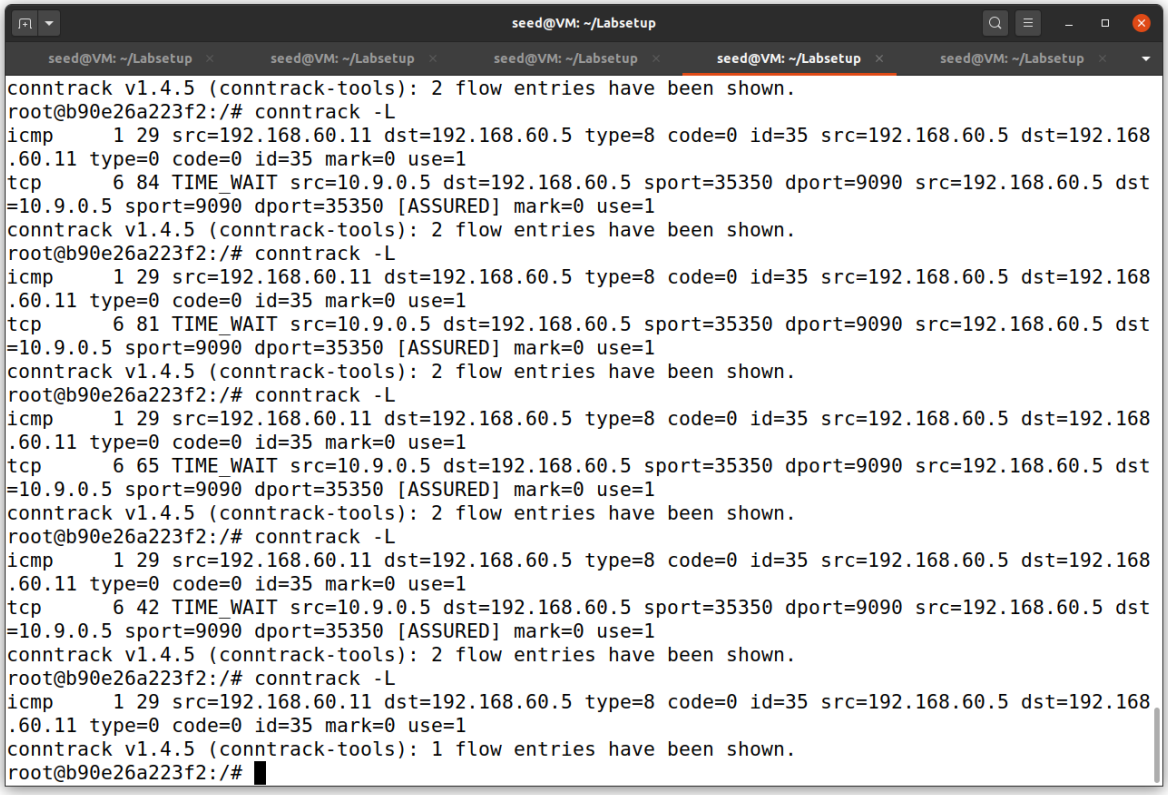
```
root@6418b95c2eca:/# nc 192.168.60.5 9090
Hello TCP Gaurav
^C
root@6418b95c2eca:/#
```

```
root@d6b23c67b5ed:/# nc -l 9090
Hello TCP Gaurav
^C
root@d6b23c67b5ed:/#
```

For this connection, the connection state stays for almost around one to two minutes before being timed out.



WE can see the time being decreased every time indicating the time remaining for the connection before getting timed out.



Task 3.B: Setting Up a Stateful Firewall

We rewrite the rules for the following task as follows:

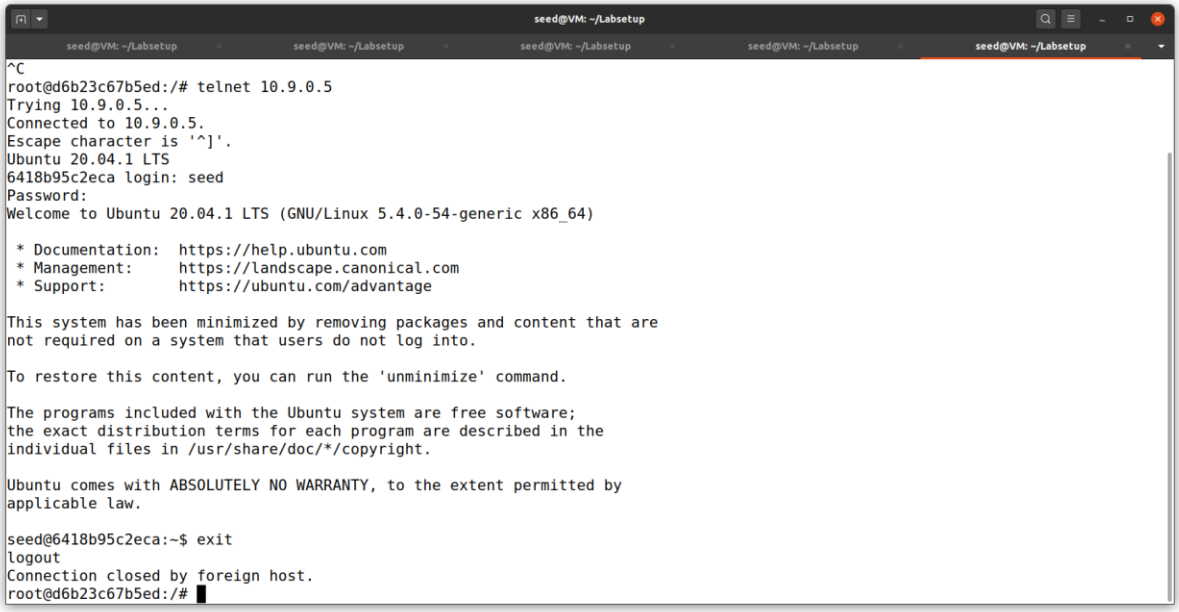
```
root@b90e26a223f2:/# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@b90e26a223f2:/# iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --ctstate NEW -j ACCEPT
root@b90e26a223f2:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED, ESTABLISHED -j ACCEPT
iptables v1.8.4 (legacy): Bad ctstate ""
Try `iptables -h' or 'iptables --help' for more information.
root@b90e26a223f2:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPTroot@b90e26a223f2:/# iptable
s -A FORWARD -p tcp -j DROP
root@b90e26a223f2:/# iptables -p FORWARD ACCEPT
iptables v1.8.4 (legacy): unknown protocol "forward" specified
Try `iptables -h' or 'iptables --help' for more information.
root@b90e26a223f2:/# iptables -P FORWARD ACCEPT
```

First, we will accept the new sync packet used to establish a connection to TCP. WE want to accept the TCP packets belonging to the established and related connections no matter from which connection as long as they follow the established and related rule. Any other connection will be dropped.

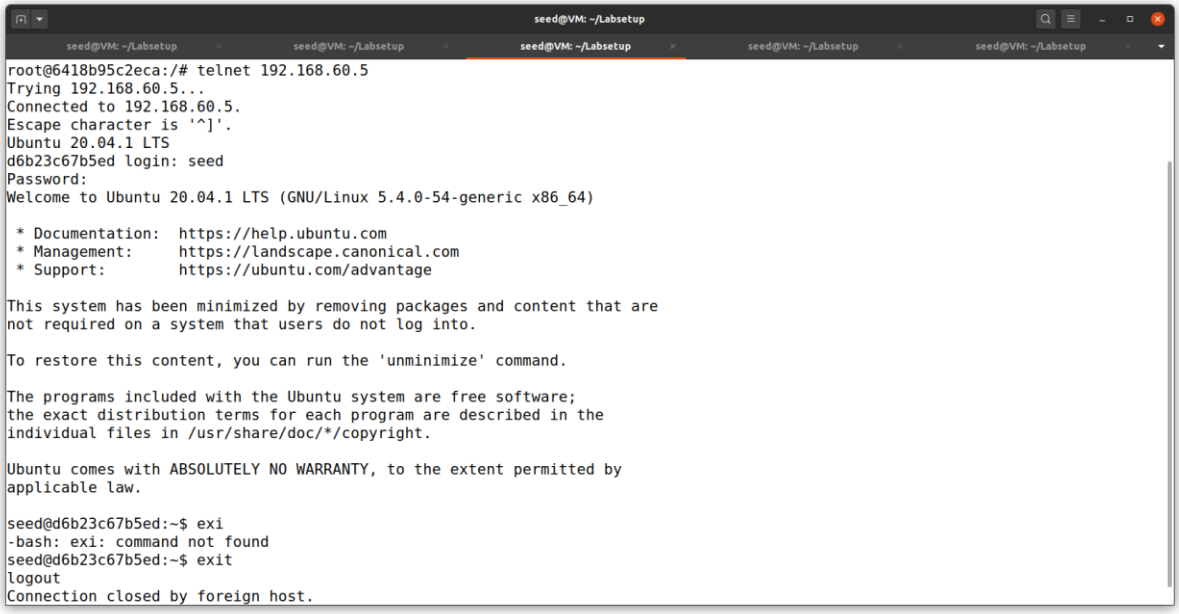
We can check the state of our iptables as follows:

```
root@b90e26a223f2:/# iptables -L -n -v
Chain INPUT (policy ACCEPT 52 packets, 4368 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

Chain OUTPUT (policy ACCEPT 52 packets, 4368 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@b90e26a223f2:/#
```

First, we try telnet to 10.9.0.5 external host. This is successfully connected.



For the outside host, it can telnet into an internal host successfully:



WE can see that in the internal network connection to host 60.6 and host 60.7 is not allowed as specified in the rules.

```
root@6418b95c2eca:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@6418b95c2eca:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@6418b95c2eca:/#
```

On the outside host, we try to connect on the internal server through the netcap TCP server, which is not allowed:

```
root@6418b95c2eca:/# nc 192.168.60.5 9090
Hello TCP connection
Hello??
```

```
root@d6b23c67b5ed:/# nc -l 9090
```

Before moving on to the next task, we restore the filter table to its original state:

```
root@b90e26a223f2:/# iptables -F
root@b90e26a223f2:/# iptables -L -n -v
Chain INPUT (policy ACCEPT 3 packets, 252 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 3 packets, 252 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Based on the set of rules with the connection tracking mechanism used and without the mechanism used, the disadvantage of the connection-based mechanism is it consumes more resources as the connection is need to be kept open which is not the case for the without connection tracking mechanism.

However, for the mechanism without connection tracking, the rules are not as strict which will not be as safe as compared to the connection tracking one. This is not the case with the connection tracking based as the rules are strict as we have seen above such that TCP connections reply only to those with established and related connections.

Task 4: Limiting Network Traffic

We run the following commands on router, and then ping 192.168.60.5 from 10.9.0.5.

```
root@b90e26a223f2:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
```

We can observe that the connection is established but the firewall is not able to limit the number of packets that can pass through the firewall.

```
root@d6b23c67b5ed:/# 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=0.161 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.171 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.171 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.136 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.163 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.190 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.166 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=10 ttl=63 time=0.166 ms
64 bytes from 10.9.0.5: icmp_seq=11 ttl=63 time=0.169 ms
64 bytes from 10.9.0.5: icmp_seq=12 ttl=63 time=0.170 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=63 time=0.158 ms
64 bytes from 10.9.0.5: icmp_seq=14 ttl=63 time=0.171 ms
64 bytes from 10.9.0.5: icmp_seq=15 ttl=63 time=0.956 ms
64 bytes from 10.9.0.5: icmp_seq=16 ttl=63 time=0.169 ms
64 bytes from 10.9.0.5: icmp_seq=17 ttl=63 time=0.169 ms
64 bytes from 10.9.0.5: icmp_seq=18 ttl=63 time=0.084 ms
64 bytes from 10.9.0.5: icmp_seq=19 ttl=63 time=0.091 ms
64 bytes from 10.9.0.5: icmp_seq=20 ttl=63 time=0.169 ms
64 bytes from 10.9.0.5: icmp_seq=21 ttl=63 time=0.160 ms
64 bytes from 10.9.0.5: icmp_seq=22 ttl=63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=23 ttl=63 time=0.066 ms
64 bytes from 10.9.0.5: icmp_seq=24 ttl=63 time=0.173 ms
64 bytes from 10.9.0.5: icmp_seq=25 ttl=63 time=0.145 ms
64 bytes from 10.9.0.5: icmp_seq=26 ttl=63 time=0.169 ms
```

We add another rule for which the packets not dropped from the first rule will be dropped from the second rule which does not match the first rule:

```
root@b90e26a223f2:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@b90e26a223f2:/#
```

Now we ping again and get the following results:

```
root@d6b23c67b5ed:/# 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=0.117 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.170 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.173 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.171 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.141 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=63 time=0.159 ms
64 bytes from 10.9.0.5: icmp_seq=19 ttl=63 time=0.107 ms
64 bytes from 10.9.0.5: icmp_seq=25 ttl=63 time=0.066 ms
64 bytes from 10.9.0.5: icmp_seq=31 ttl=63 time=0.170 ms
64 bytes from 10.9.0.5: icmp_seq=37 ttl=63 time=0.174 ms
64 bytes from 10.9.0.5: icmp_seq=43 ttl=63 time=0.170 ms
64 bytes from 10.9.0.5: icmp_seq=48 ttl=63 time=0.247 ms
64 bytes from 10.9.0.5: icmp_seq=54 ttl=63 time=0.172 ms
64 bytes from 10.9.0.5: icmp_seq=60 ttl=63 time=0.166 ms
64 bytes from 10.9.0.5: icmp_seq=66 ttl=63 time=0.169 ms
^C
--- 10.9.0.5 ping statistics ---
67 packets transmitted, 16 received, 76.1194% packet loss, time 67583ms
rtt min/avg/max/mdev = 0.066/0.158/0.247/0.037 ms
```

There is a time condition on the connection which is 10/minute which means it takes 6 seconds approx. for each packet to be sent. This shows that the second rule is required.

Before moving on to the next task, we restore the filter table to its original state:

```
root@b90e26a223f2:/# iptables -F
root@b90e26a223f2:/# iptables -L -n -v
Chain INPUT (policy ACCEPT 2 packets, 168 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 2 packets, 168 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@b90e26a223f2:/#
```

Task 5: Load Balancing

We first start the server on each of the hosts: 192.168.60.5, 192.168.60.6, and 192.168.60.7. In the router, we set up the following rule:

```
root@b90e26a223f2:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to
-destination 192.168.60.5:8080
root@b90e26a223f2:/#
```

It dispatches the packets coming to 8080 to the inner hosts.

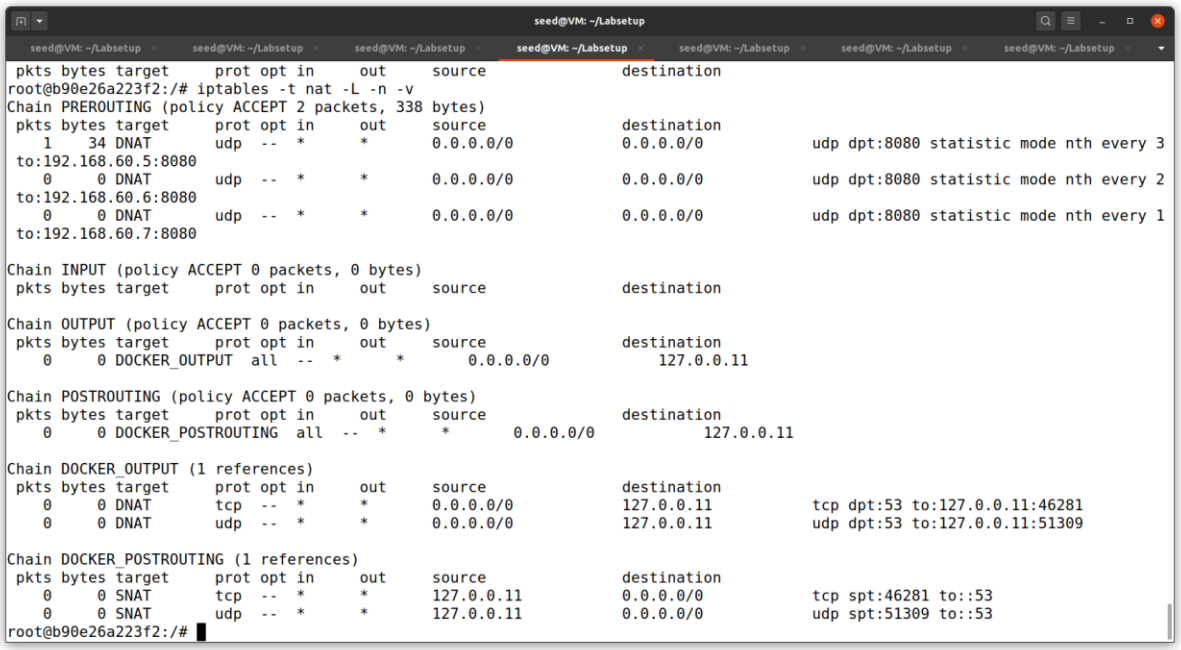
From the external host we run the following command to send a UDP packet to the router's 8080 port, we will see that one out of three packets gets to 192.168.60.5

```
root@6418b95c2eca:/# echo hello | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/#
```

```
root@d6b23c67b5ed:/# nc -luk 8080
hello
```

Here, we add more rules so that all the three internal hosts get the equal number of packets:

```
root@b90e26a223f2:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to
-destination 192.168.60.6:8080
root@b90e26a223f2:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 1 --packet 0 -j DNAT --to
-destination 192.168.60.7:8080
```



Now, we check by sending the message repeatedly while we can see that all the internal hosts gets the equal number of packets as follows:

```
root@6418b95c2eca:/# echo hello | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "hello gaurav hello gaurav hello gaurav" | nc -u 10.9.0.11 8080
^C
```

```
[03/27/22]seed@VM:~/Labsetup$ docksh host2-192.168.60.6
root@2707552e5f5b:/# nc -luk 8080
hello gaurav hello gaurav hello gaurav
```

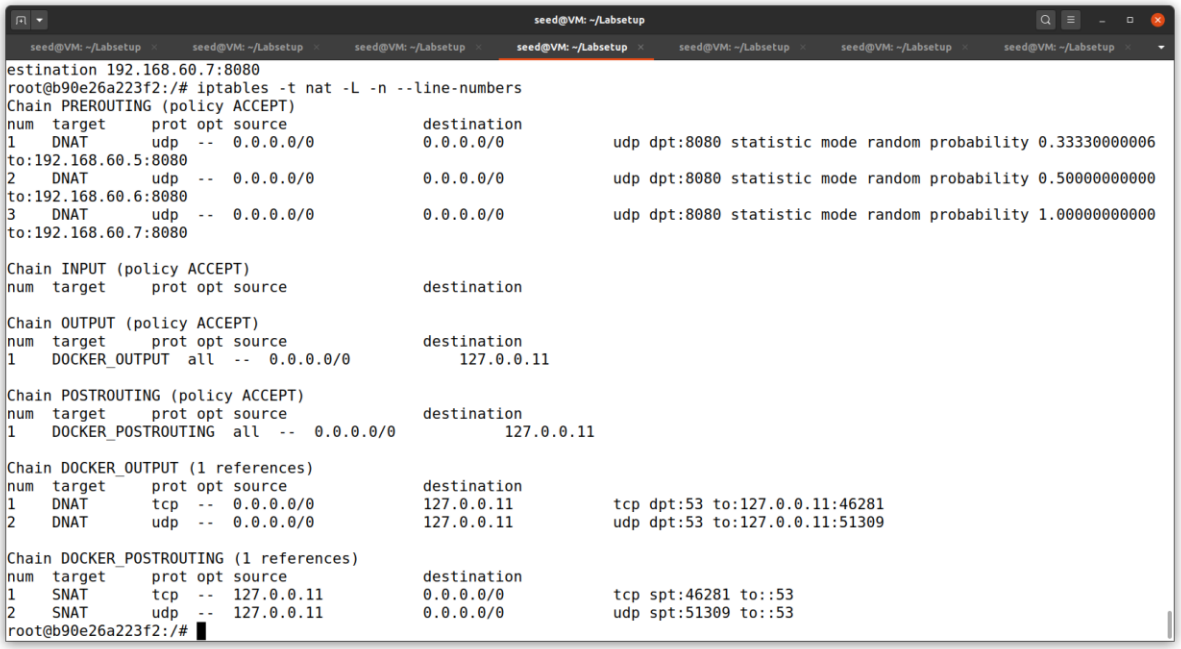
```
root@6418b95c2eca:/# echo hello | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "hello gaurav hello gaurav hello gaurav" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "hello gaurav hello gaurav hello gaurav" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/#
```

```
[03/27/22]seed@VM:~/Labsetup$ docksh host3-192.168.60.7
root@590976ecdd88:/# nc -luk 8080
hello gaurav hello gaurav hello gaurav
```

Using the random mode:

WE modify the rules based on the random mode for achieving load balancing with random probabilities assigned to the packets send to different hosts:

```
root@b90e26a223f2:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.3333 -j DNAT --to-destination 192.168.60.5:8080
root@b90e26a223f2:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-destination 192.168.60.6:8080
root@b90e26a223f2:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1 -j DNAT --to-destination 192.168.60.7:8080
```



We now test the command from the external host as follows:

```
root@6418b95c2eca:/# echo "NEW Hello 1" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 2" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 1" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 2" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 3" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 4" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 5" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 6" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 7" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/# echo "NEW Hello 8" | nc -u 10.9.0.11 8080
^C
root@6418b95c2eca:/#
```

We can see that based on the probabilities the packets are send to different internal hosts randomly:

```
root@d6b23c67b5ed:/# nc -luk 8080
NEW Hello 1
NEW Hello 6
NEW Hello 7
```

```
root@2707552e5f5b:/# nc -luk 8080
NEW Hello 2
NEW Hello 4
NEW Hello 5
```

```
root@590976ecdd88:/# nc -luk 8080
NEW Hello 3
NEW Hello 8
```