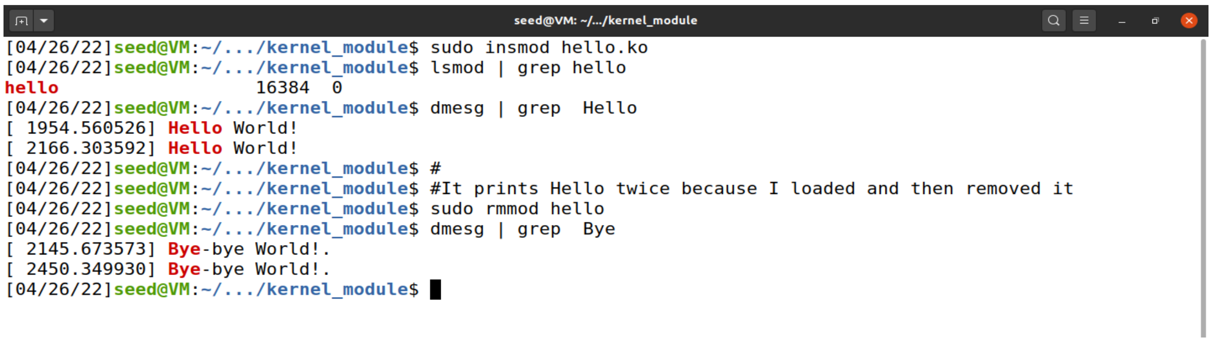
Report Firewall Exploration Lab - Fatjon Freskina

Task 1 - Implement a simple Firewall

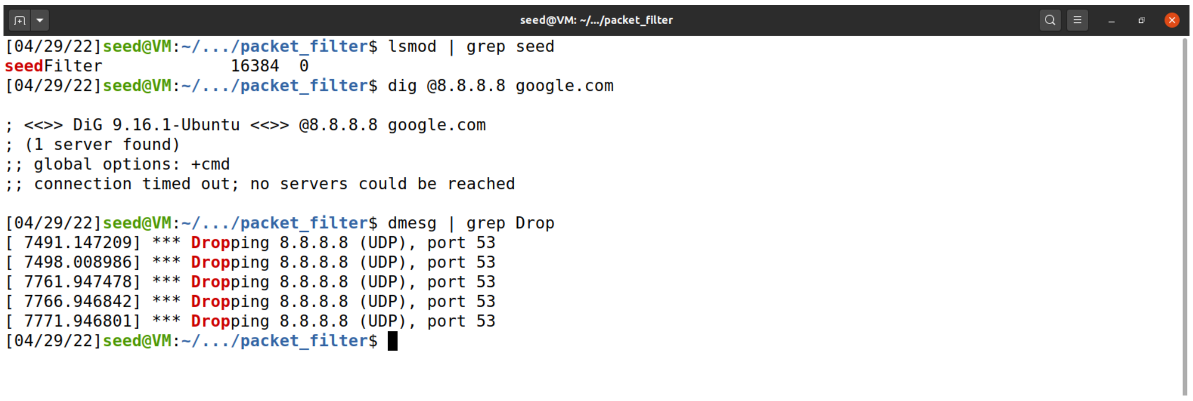
1.A I loaded and removed successfully the loadable kernel module:



```
[04/26/22]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[04/26/22]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[04/26/22]seed@VM:~/.../kernel_module$ dmesg | grep  Hello
[ 1954.560526] Hello World!
[ 2166.303592] Hello World!
[04/26/22]seed@VM:~/.../kernel_module$ #
[04/26/22]seed@VM:~/.../kernel_module$ #It prints Hello twice because I loaded and then removed it
[04/26/22]seed@VM:~/.../kernel_module$ sudo rmmod hello
[04/26/22]seed@VM:~/.../kernel_module$ dmesg | grep  Bye
[ 2145.673573] Bye-bye World!.
[ 2450.349930] Bye-bye World!.
[04/26/22]seed@VM:~/.../kernel_module$ 
```

1.B

After loading the seedFilter kernel module, I tested the filter trying to dig 8.8.8.8. The hook function blockUDP is successfully invoked and the packet is dropped since it specifies the conditions (ip header of outgoing packet = 8.8.8.8 and UDP port is 53). A message in the buffer of the kernel appears when looking for dropped packets with dmesg.



```
[04/29/22]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
[04/29/22]seed@VM:~/.../packet_filter$ dig @8.8.8.8 google.com

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

[04/29/22]seed@VM:~/.../packet_filter$ dmesg | grep Drop
[ 7491.147209] *** Dropping 8.8.8.8 (UDP), port 53
[ 7498.008986] *** Dropping 8.8.8.8 (UDP), port 53
[ 7761.947478] *** Dropping 8.8.8.8 (UDP), port 53
[ 7766.946842] *** Dropping 8.8.8.8 (UDP), port 53
[ 7771.946801] *** Dropping 8.8.8.8 (UDP), port 53
[04/29/22]seed@VM:~/.../packet_filter$ 
```

The PRE_ROUTING condition is triggered when packets come in: having passed simple sanity checks they are passed to the netfilter hook.

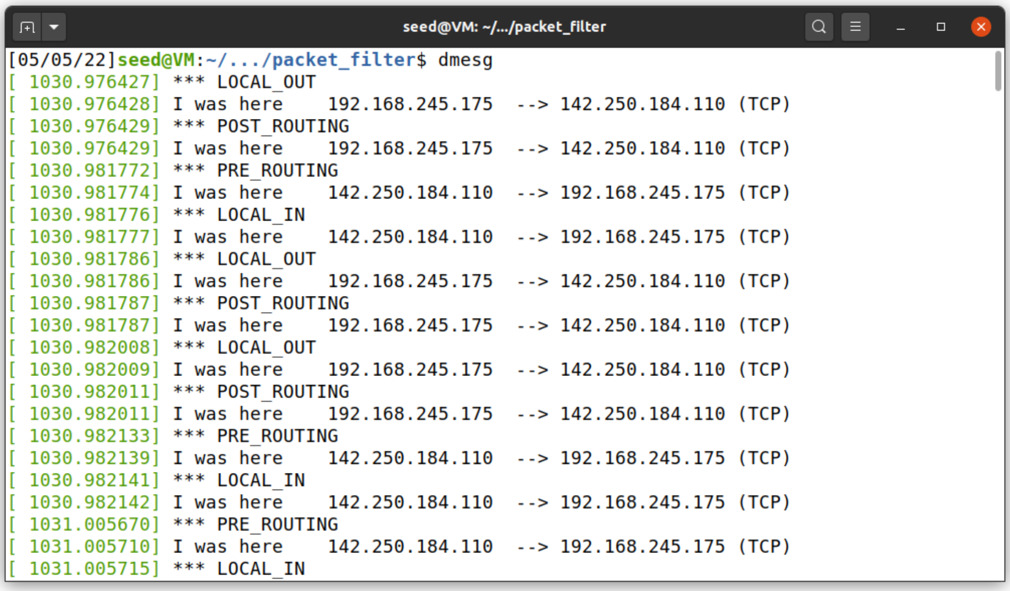
Once the packet has been filtered at pre_routing, it enters the routing code, which decides whether the packet is destined for another interface or a local process. If it is destined to a local process, the LOCAL_IN is triggered and there is another filtering (which needs to be defined).

The FORWARD hook is triggered instead if the packet is destined to another interface.

LOCAL_OUT is activated when packets created locally need to be routed, but before routing the hook performs some filtering.

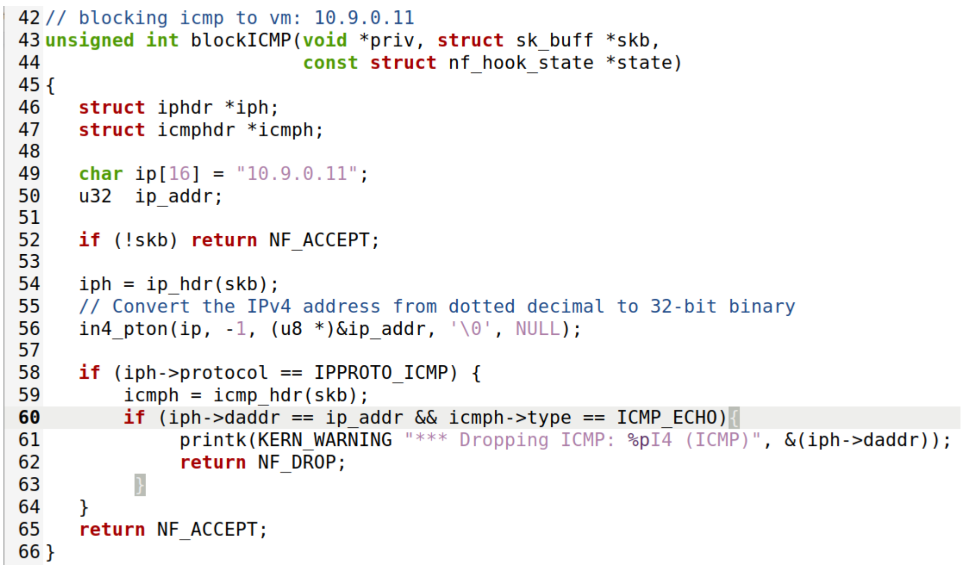
Finally POST_ROUTING is the last hook that a packet that needs to be put on the wire has to pass.

Here it is a screenshot of *dmesg* after loading the kernel module



```
[05/05/22]seed@VM:~/.../packet_filter$ dmesg
[ 1030.976427] *** LOCAL_OUT
[ 1030.976428] I was here    192.168.245.175  --> 142.250.184.110 (TCP)
[ 1030.976429] *** POST_ROUTING
[ 1030.976429] I was here    192.168.245.175  --> 142.250.184.110 (TCP)
[ 1030.981772] *** PRE_ROUTING
[ 1030.981774] I was here    142.250.184.110  --> 192.168.245.175 (TCP)
[ 1030.981776] *** LOCAL_IN
[ 1030.981777] I was here    142.250.184.110  --> 192.168.245.175 (TCP)
[ 1030.981786] *** LOCAL_OUT
[ 1030.981786] I was here    192.168.245.175  --> 142.250.184.110 (TCP)
[ 1030.981787] *** POST_ROUTING
[ 1030.981787] I was here    192.168.245.175  --> 142.250.184.110 (TCP)
[ 1030.982008] *** LOCAL_OUT
[ 1030.982009] I was here    192.168.245.175  --> 142.250.184.110 (TCP)
[ 1030.982011] *** POST_ROUTING
[ 1030.982011] I was here    192.168.245.175  --> 142.250.184.110 (TCP)
[ 1030.982133] *** PRE_ROUTING
[ 1030.982139] I was here    142.250.184.110  --> 192.168.245.175 (TCP)
[ 1030.982141] *** LOCAL_IN
[ 1030.982142] I was here    142.250.184.110  --> 192.168.245.175 (TCP)
[ 1031.005670] *** PRE_ROUTING
[ 1031.005710] I was here    142.250.184.110  --> 192.168.245.175 (TCP)
[ 1031.005715] *** LOCAL_IN
```

In order to block icmp messages I wrote this function:

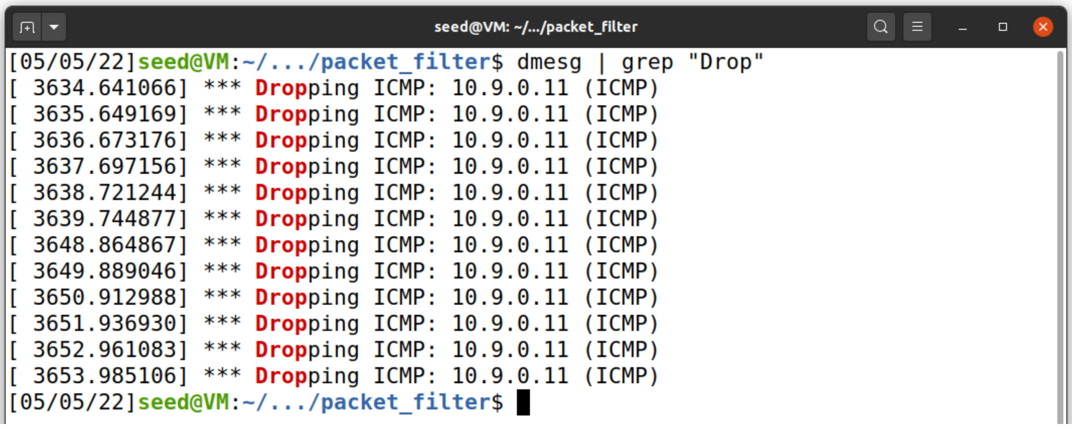


```c
// blocking icmp to vm: 10.9.0.11
unsigned int blockICMP(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct icmphdr *icmph;

   char ip[16] = "10.9.0.11";
   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 ICMP: %pI4 (ICMP)", &(iph->daddr));
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

The function basically drops the packet if it is an ICMP_ECHO message towards ip address 10.9.0.11.
Then, when I registered the hook, I “attached” this function to the hook, invoking it with the pre_routing hook number:

```
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);
```

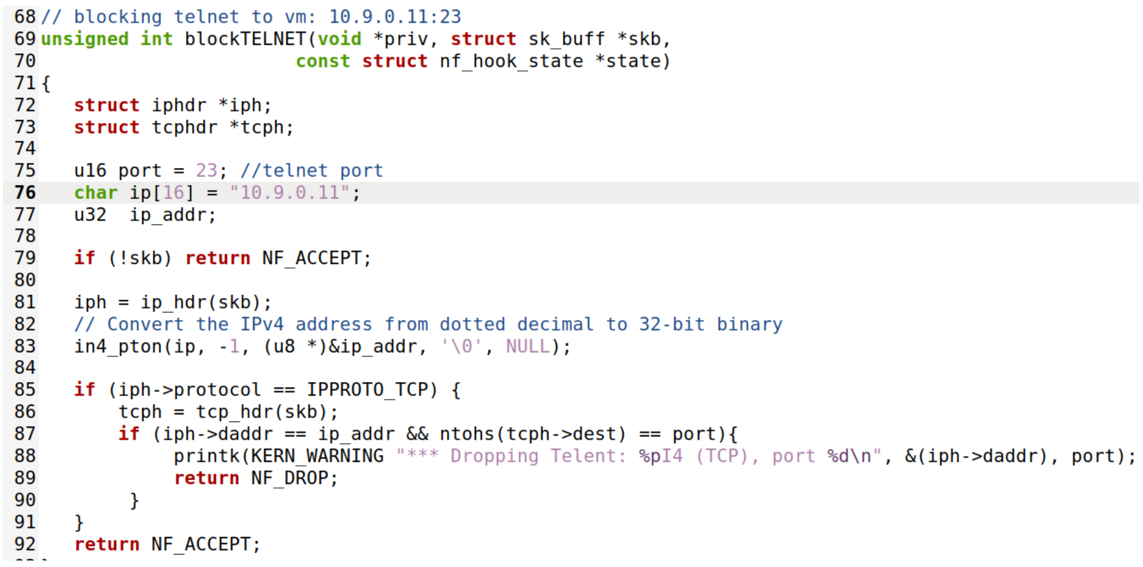
When I try to ping the router from another virtual machine, the kernel drops the packet successfully:



```
[05/05/22]seed@VM:~/.../packet_filter$ dmesg | grep "Drop"
[ 3634.641066] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3635.649169] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3636.673176] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3637.697156] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3638.721244] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3639.744877] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3648.864867] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3649.889046] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3650.912988] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3651.936930] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3652.961083] *** Dropping ICMP: 10.9.0.11 (ICMP)
[ 3653.985106] *** Dropping ICMP: 10.9.0.11 (ICMP)
[05/05/22]seed@VM:~/.../packet_filter$
```

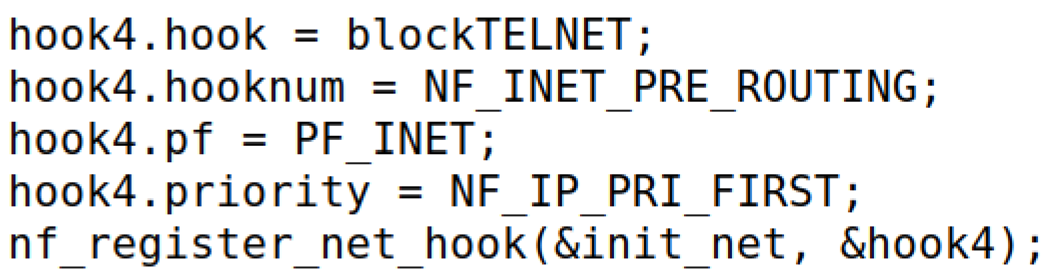
1.B (iv)

Function defined in the .c file in order to block telnet connections to the router:



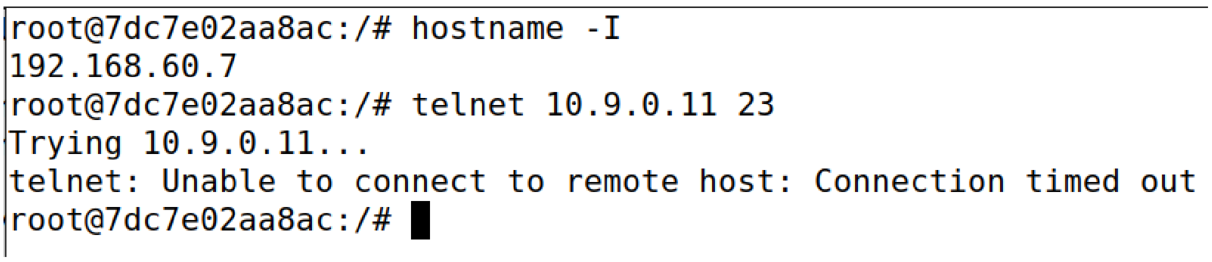
```
// blocking telnet to vm: 10.9.0.11:23
unsigned int blockTELNET(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct tcphdr *tcph;

   u16 port = 23; //telnet port
   char ip[16] = "10.9.0.11";
   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 Telent: %pI4 (TCP), port %d\n", &(iph->daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
```

Registration of the filter:



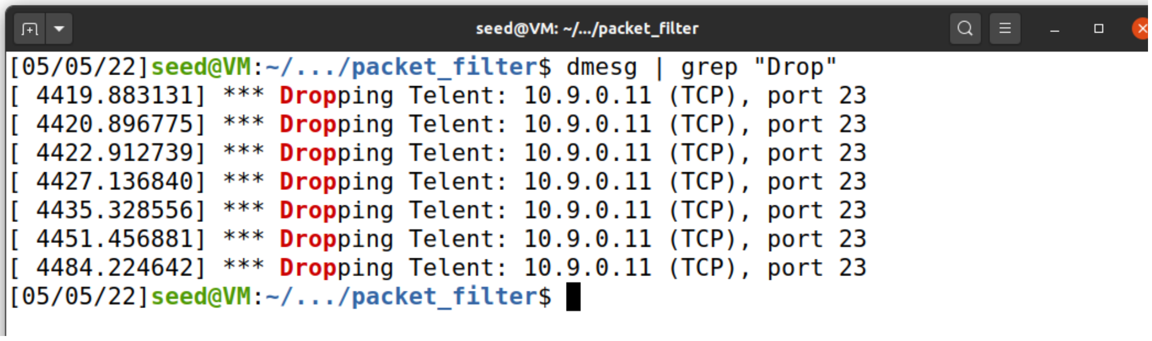
```
hook4.hook = blockTELNET;
hook4.hooknum = NF_INET_PRE_ROUTING;
hook4.pf = PF_INET;
hook4.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook4);
```

Proof of the work:



```
root@7dc7e02aa8ac:/# hostname -I
192.168.60.7
root@7dc7e02aa8ac:/# telnet 10.9.0.11 23
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
root@7dc7e02aa8ac:/# 
```

And dmesg output from the host machine:



```
[05/05/22]seed@VM:~/.../packet_filter$ dmesg | grep "Drop"
[ 4419.883131] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[ 4420.896775] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[ 4422.912739] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[ 4427.136840] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[ 4435.328556] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[ 4451.456881] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[ 4484.224642] *** Dropping Telent: 10.9.0.11 (TCP), port 23
[05/05/22]seed@VM:~/.../packet_filter$ 
```

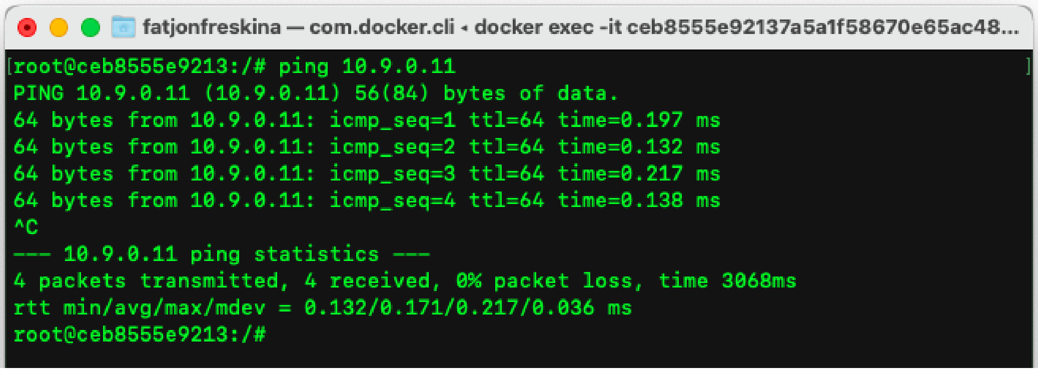
TASK 2: Experimenting with stateless firewall rules

2.A Protecting the router

I added these rules to the seed router's firewall. I will also try to explain the syntax.
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
Append a new rule in the filter (default) table, chain INPUT, protocol icmp -> Accept.
Now ping requests or generally icmp messages are accepted by the router.
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
Same as before, in this case outgoing icmp packets are not blocked by the firewall.
iptables -P OUTPUT DROP # Set default rule for OUTPUT
Add rule to the filter table, chain output. All outgoing packets are dropped (applyed after the previous ones)

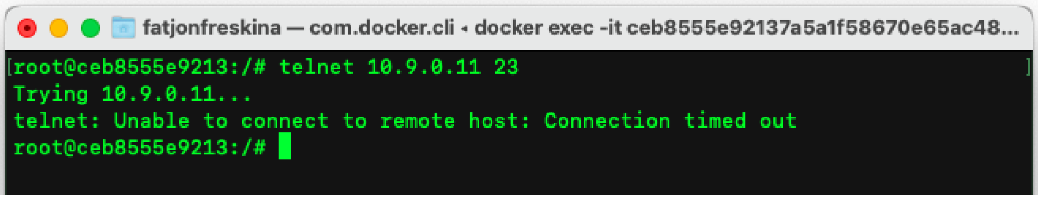
iptables -P INPUT DROP # Set default rule for INPUT
Same as the previous one, for incoming packets.

As expected ping works, since it uses the icmp protocol:


fatjonfreskina — com.docker.cli ◂ docker exec -it ceb8555e92137a5a1f58670e65ac48...

```
[root@ceb8555e9213:/# 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=0.197 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.217 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.138 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.132/0.171/0.217/0.036 ms
root@ceb8555e9213:/#
```

Telnet on the other hand is not successful since it does not use the icmp protocol, and by default incoming connections not based on this protocol are dropped:


fatjonfreskina — com.docker.cli ◂ docker exec -it ceb8555e92137a5a1f58670e65ac48...

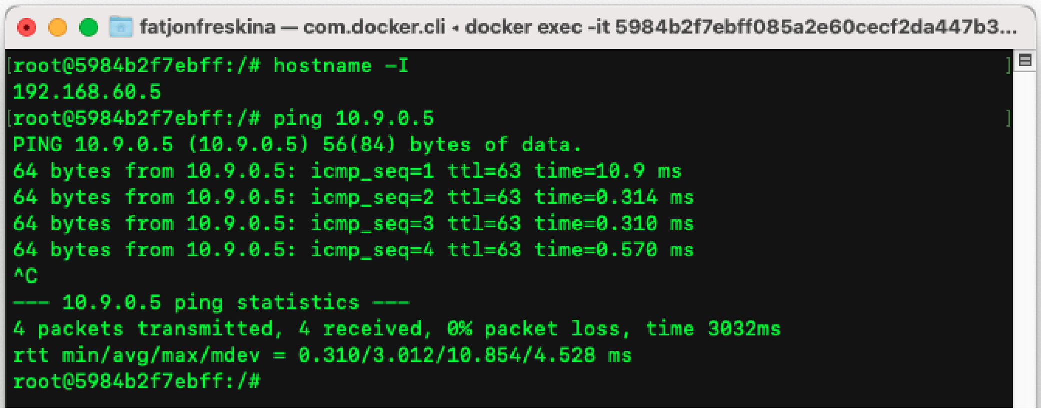
```
[root@ceb8555e9213:/# telnet 10.9.0.11 23
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
root@ceb8555e9213:/#
```

2.B Protecting the Internal Network

Commands I used to setup the firewall in the router:

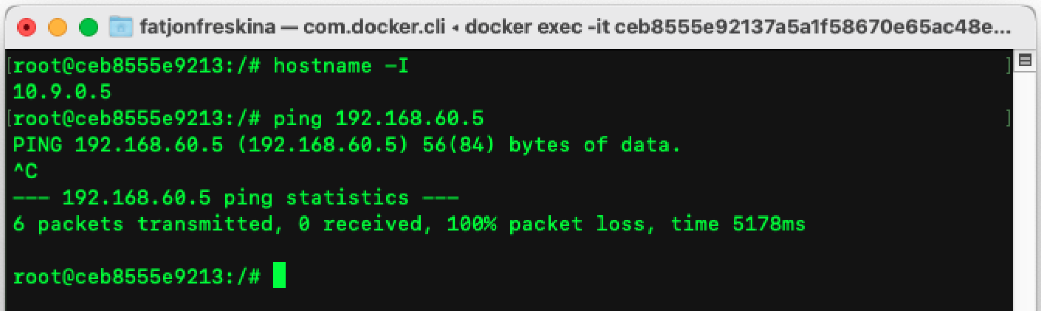
iptables -A FORWARD -p icmp --icmp-type echo-request -d 10.9.0.0/24 -j ACCEPT
This appends in the filter table, forward chain, a new rule: accepting icmp echo request packets toward 10.9.0.0/24 which is the “external” network.
iptables -A FORWARD -p icmp --icmp-type echo-reply -d 192.168.60.0/24 -j ACCEPT
Again filter table (default if not specified), forward chain: accepting icmp echo reply packets towards the internal network 192.168.60.0/23.
iptables -P FORWARD DROP
Finally here I am setting the default policy on FORWARD on drop, so that if a packet is not accepted by the two rules above, it will be dropped.

Here we can see how ping worked from the inside towards the outside:


fatjonfreskina — com.docker.cli ◂ docker exec -it 5984b2f7ebff085a2e60cecf2da447b3...

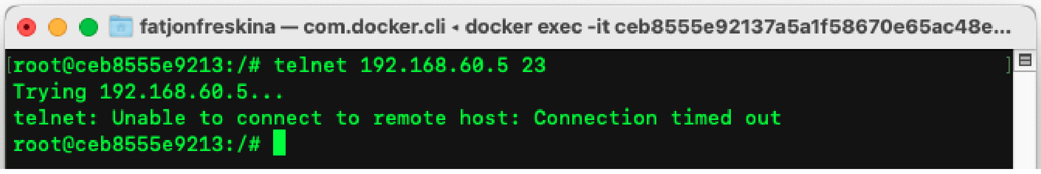
```
[root@5984b2f7ebff:/# hostname -I
192.168.60.5
[root@5984b2f7ebff:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=10.9 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.314 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.310 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.570 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3032ms
rtt min/avg/max/mdev = 0.310/3.012/10.854/4.528 ms
root@5984b2f7ebff:/#
```

Here instead, I tested ping from the outside towards the internal network:


fatjonfreskina — com.docker.cli ◂ docker exec -it ceb8555e92137a5a1f58670e65ac48e...

```
[root@ceb8555e9213:/# hostname -I
10.9.0.5
[root@ceb8555e9213:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5178ms

root@ceb8555e9213:/#
```

Finally, to test a non-icmp packet I tried to telnet from the outside the internal network:


fatjonfreskina — com.docker.cli ◂ docker exec -it ceb8555e92137a5a1f58670e65ac48e...

```
[root@ceb8555e9213:/# telnet 192.168.60.5 23
Trying 192.168.60.5...
telnet: Unable to connect to remote host: Connection timed out
root@ceb8555e9213:/#
```

Note on my solution: this is correct only if considering as outside hosts the ones in net. 10.9.0.0/24. If I were to provide a wider solution I would write rules considering the interfaces and not the IP dest of the packets. For example, I should append:

Iptables -A FORWARD -icmp –icmp-type echo-request -i eth1 -j ACCEPT
Iptables -A FORWARD -icmp –icmp-type echo-request -o eth0 -j ACCEPT
Accept and forward icmp echo-request packets coming from interface eth0 (linked to internal hosts) and directed towards eth1 (the interface pointing to the external hosts)

Iptables -A FORWARD -icmp –icmp-type echo-reply -o eth1 -j ACCEPT
Iptables -A FORWARD -icmp –icmp-type echo-reply -i eth0 -j ACCEPT

Accept and forward icmp echo reply packets coming from the outside towards the internal hosts.
Else: drop everything with the drop policy.

Task 2.C: Protecting Internal Server

We basically need two commands to solve this part:

```
iptables -t filter -A FORWARD -d 192.168.60.0/24 ! 192.168.60.5 -p tcp --sport 1024:65535
-dport 23 -i eth0 -j REJECT &&
```

```
iptables -P FORWARD DROP
```

These two commands allow traffic only for the telnet connection on port 23 from external network to the host 192.168.60.5 and dropping all the others with destination host in the subnetwork 192.168.60.0/24.