Here we have a list of containers: please use this as reference for the container ids:

```
e8c74841a96d host1-192.168.60.5
3891448f10a5 host2-192.168.60.6
68a8ecc3eacf hostA-10.9.0.5
a9bc3a5e14f8 host3-192.168.60.7
454ee006bd8d seed-router
```

The following screenshot is from ip addr, we can see eth0 if for external and eth1 is for internal.

```
root@454ee006bd8d:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
64: eth0@if65: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default
    link/ether 02:42:0a:09:00:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.11/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
66: eth1@if67: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default
    link/ether 02:42:c0:a8:3c:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.60.11/24 brd 192.168.60.255 scope global eth1
       valid_lft forever preferred_lft forever
```

1. A) we can see here that we have made the filter.ko file.

```
[11/17/22]sysc4810@sysc4810-vm23:~/.../packet_filter$ make
make -C /lib/modules/5.15.0-47-generic/build M=/home/sysc4810/Documents/a3/code/
packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-47-generic'
  CC [M]  /home/sysc4810/Documents/a3/code/packet_filter/filter.o
  MODPOST /home/sysc4810/Documents/a3/code/packet_filter/Module.symvers
  CC [M]  /home/sysc4810/Documents/a3/code/packet_filter/filter.mod.o
  LD [M]  /home/sysc4810/Documents/a3/code/packet_filter/filter.ko
  BTF [M] /home/sysc4810/Documents/a3/code/packet_filter/filter.ko
Skipping BTF generation for /home/sysc4810/Documents/a3/code/packet_filter/filte
r.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-47-generic'
[11/17/22]sysc4810@sysc4810-vm23:~/.../packet_filter$ ls
Makefile        filter.c   filter.mod    filter.mod.o  modules.order
Module.symvers  filter.ko  filter.mod.c  filter.o
```

Then we load the kernel and generate udp packets.

```
[11/17/22]sysc4810@sysc4810-vm23:~/.../packet_filter$ sudo insmod filter.ko
[11/17/22]sysc4810@sysc4810-vm23:~/.../packet_filter$ lsmod | grep filter
filter                 16384  0
br_netfilter           32768  0
bridge                307200  1 br_netfilter
[11/17/22]sysc4810@sysc4810-vm23:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.18.1-1ubuntu1.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached

[11/17/22]sysc4810@sysc4810-vm23:~/.../packet_filter$ dmesg | tail -10
dmesg: read kernel buffer failed: Operation not permitted
```

b) Here we telnet our attacker host from our external host. It gets blocked.

```
root@68a8ecc3eacf:/# telnet 10.9.0.1
Trying 10.9.0.1...
telnet: Unable to connect to remote host: Connection timed out
```

Then we ping our attacker host from our external host and the ping is also blocked.

```
root@68a8ecc3eacf:/# 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 ---
229 packets transmitted, 0 received, 100% packet loss, time 233460ms
```

Then we telnet and ping our internal hosts from our external host and it goes through showing other IP addresses are not affected.

```
root@68a8ecc3eacf:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3891448f10a5 login: ^CConnection closed by foreign host.
root@68a8ecc3eacf:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e8c74841a96d login: ^CConnection closed by foreign host.
root@68a8ecc3eacf:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a9bc3a5e14f8 login:
```

```
root@68a8ecc3eacf:/# 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=0.479 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.283 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.263 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.324 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.336 ms
^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.263/0.337/0.479/0.075 ms
root@68a8ecc3eacf:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
64 bytes from 192.168.60.6: icmp_seq=1 ttl=63 time=0.564 ms
64 bytes from 192.168.60.6: icmp_seq=2 ttl=63 time=0.134 ms
64 bytes from 192.168.60.6: icmp_seq=3 ttl=63 time=0.300 ms
64 bytes from 192.168.60.6: icmp_seq=4 ttl=63 time=1.26 ms
64 bytes from 192.168.60.6: icmp_seq=5 ttl=63 time=0.437 ms
64 bytes from 192.168.60.6: icmp_seq=6 ttl=63 time=0.250 ms
^C64 bytes from 192.168.60.6: icmp_seq=7 ttl=63 time=0.236 ms
64 bytes from 192.168.60.6: icmp_seq=8 ttl=63 time=0.133 ms
^C
--- 192.168.60.6 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7105ms
rtt min/avg/max/mdev = 0.133/0.414/1.263/0.348 ms
root@68a8ecc3eacf:/# ping 192.168.60.7
PING 192.168.60.7 (192.168.60.7) 56(84) bytes of data.
64 bytes from 192.168.60.7: icmp_seq=1 ttl=63 time=2.25 ms
64 bytes from 192.168.60.7: icmp_seq=2 ttl=63 time=0.196 ms
64 bytes from 192.168.60.7: icmp_seq=3 ttl=63 time=0.216 ms
64 bytes from 192.168.60.7: icmp_seq=4 ttl=63 time=0.124 ms
64 bytes from 192.168.60.7: icmp_seq=5 ttl=63 time=0.150 ms
64 bytes from 192.168.60.7: icmp_seq=6 ttl=63 time=0.123 ms
```

The following the blockTCP function for telnet commands that is called on hook3.

```
unsigned int blockTCP(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 (TCP), port %d\n", &(iph->daddr), port);
           return NF_DROP;
       }
   }
   return NF_ACCEPT;
}
```

The following is the blockICMP function for pings that is called on hook4.

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

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

The following is where we initialize hook3 and hook4. Both have the same netfilter of NF_INET_PRE_ROUTING because we want to check the packet right after the packet has been received on a network and before a routing decision is made.

```
hook3.hook = blockTCP;
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 = blockICMP;
hook4.hooknum = NF_INET_PRE_ROUTING;
hook4.pf = PF_INET;
hook4.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook4);
```

This shows the hooks being unregistered in removeFilter.

```
nf_unregister_net_hook(&init_net, &hook3);
nf_unregister_net_hook(&init_net, &hook4);
```

2. A)

   iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
   this line allows all incoming pings to go to the router.
   iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
   this line allows outgoing pings to go from the router.
   iptables -P OUTPUT DROP
   this line blocks all other packets going out.
   iptables -P INPUT DROP
   this line blocks all other packets going in.

   b) We can't telnet into the router as seen from the screenshot below since we have a rule to drop/block everything that isn't a ping.

```
root@68a8ecc3eacf:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

c) Yes, we can ping the router because we made a rule to allow pings to go to the router, it is an exception.

```
root@68a8ecc3eacf:/# 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=0.836 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.168 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.142 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.153 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3023ms
rtt min/avg/max/mdev = 0.142/0.324/0.836/0.295 ms
```

3. These were the rules created:
   iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP

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

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

   iptables -P FORWARD DROP

   The rules in order listed above mean, we block external ping requests to internal, accept internal pings going to external, and accept external pings to the router. And we block all other packets going either way.

```
root@454ee006bd8d:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reques
t -j DROP
root@454ee006bd8d:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-reques
t -j ACCEPT
root@454ee006bd8d:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-reques
t -j ACCEPT
root@454ee006bd8d:/# iptables -P FORWARD DROP
root@454ee006bd8d:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply
-j ACCEPT
```

   (Wrote one twice accidentally).

   This shows us pinging an internal host from an external host, it does not go through as we are pinging from an external host.

```
root@68a8ecc3eacf:/# 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 ---
50 packets transmitted, 0 received, 100% packet loss, time 50180ms
```

   This shows us pinging our router from our external host and it goes through.

```
root@68a8ecc3eacf:/# 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=0.428 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.210 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.121 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.303 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3075ms
rtt min/avg/max/mdev = 0.121/0.265/0.428/0.113 ms
```

   This shows us pinging our external host from out internal host 192.168.60.5 and it goes through.

```
root@e8c74841a96d:/# 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=2.76 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.178 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.126 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.437 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3051ms
rtt min/avg/max/mdev = 0.126/0.876/2.764/1.096 ms
```

This shows us using a different packet (telnet) and it does not go through as all other packets are blocking going anywhere from anywhere (in this example, telnet internal from an external host).

```
root@68a8ecc3eacf:/# telnet 192.168.60.5
Trying 192.168.60.5...
telnet: Unable to connect to remote host: Connection timed out
```

(In this example we telnet external from internal and it also does not go through).

```
root@e8c74841a96d:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

4. These were the rules created:

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

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

   iptables -P FORWARD DROP

```
root@454ee006bd8d:/# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
root@454ee006bd8d:/# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
root@454ee006bd8d:/# iptables -P FORWARD DROP
```

The rules in order listed above mean

- we accept external TCP packets to the specific internal host where the destination is 192.168.60.5 if it also has a destination port of 23.
- we accept TCP packets coming from the internal host if their source is 192.168.60.5 and has a source port is 23.
- we block all other packets going either way.

This screenshot shows we can telnet to our specific internal server of 192.168.60.5. from our external host 10.9.0.5.

```
root@68a8ecc3eacf:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e8c74841a96d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-47-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.
```

This shows that we can't telnet other internal hosts from our external host.

```
root@68a8ecc3eacf:/# telnet 192.168.60.6
Trying 192.168.60.6...
telnet: Unable to connect to remote host: Connection timed out
```

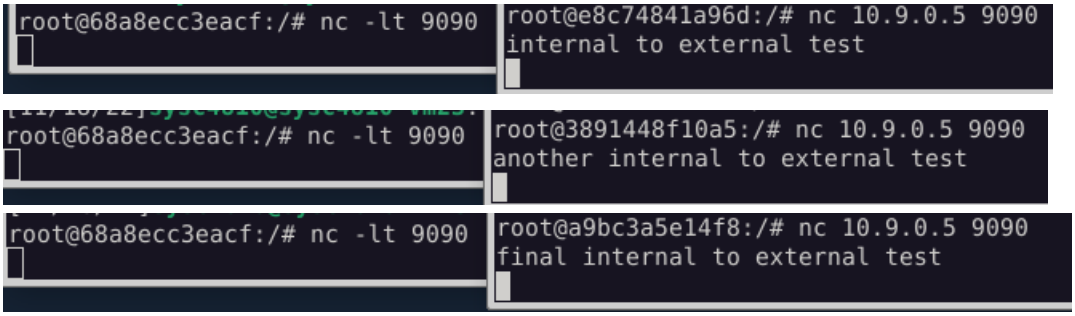
```
root@68a8ecc3eacf:/# telnet 192.168.60.7
Trying 192.168.60.7...
telnet: Unable to connect to remote host: Connection timed out
```

This shows we can telnet other internal hosts from another internal host. (192.168.60.5 to 192.168.60.6 and 192.168.60.5 to 192.168.60.7)

```
root@e8c74841a96d:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3891448f10a5 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-47-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.
```

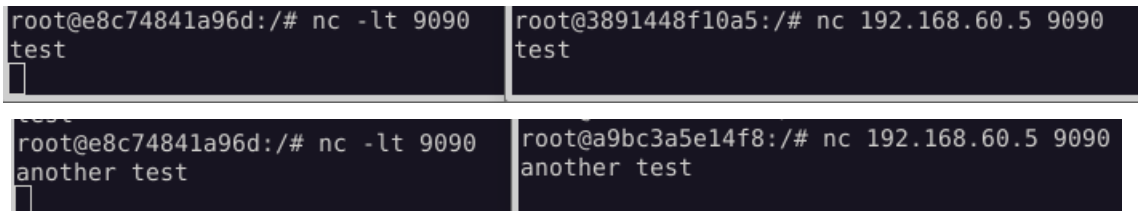
```
root@e8c74841a96d:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a9bc3a5e14f8 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-47-generic x86_64)

 * 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.
```

This shows that an external server can't be accessed by an internal host (left is server right is accessing host). We see the message does not go through.

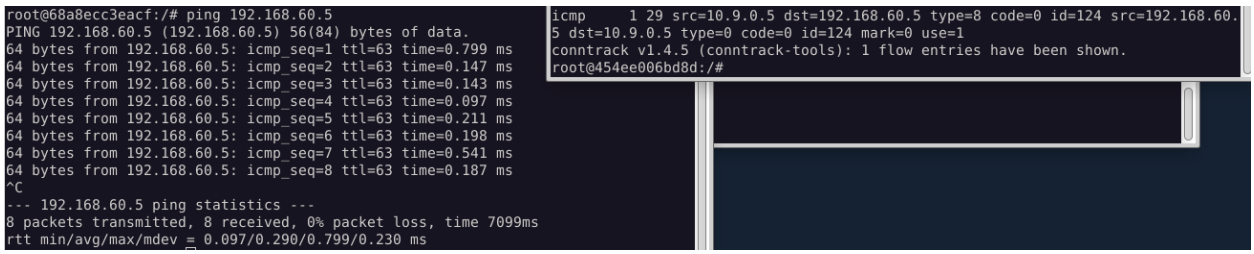


This shows that an internal server can't be accessed by an external host. We see the message does not go through.

This shows that an internal server can be accessed by another internal server. We see the message does go through.

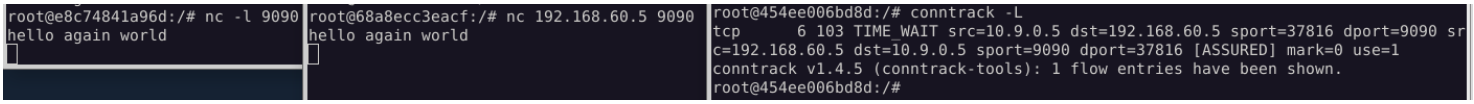


5. a) We can see from the right terminal when we run conntrack -L, the ICMP connection is there and we can see from that (the third value) that the ICMP connection state is to be kept for 29 seconds.



b) We see that the internal server running (left terminal) can be accessed from the external host (middle terminal) where we see the text reflected in the server for UDP. In the right terminal we check the UDP connection and see that the UDP connection state is to be kept for 10 seconds.

c) We see that the internal server running (left terminal) can be accessed from the external host (middle terminal) where we see the text reflected in the server for TCP. In the right terminal we check the TCP connection and see that the TCP connection state is to be kept for 103 seconds.



6. The rules used for this were:

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

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

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

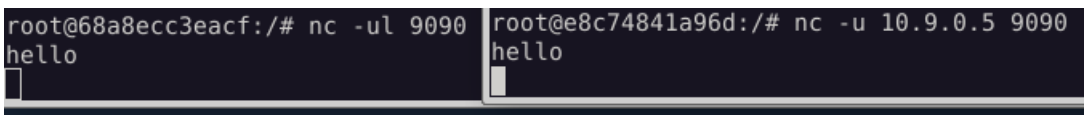
iptables -P FORWARD DROP

```
root@454ee006bd8 iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 -
-syn -m conntrack --ctstate NEW -j ACCEPT
root@454ee006bd8d:/# iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --cts
tate NEW -j ACCEPT
root@454ee006bd8d:/# 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@454ee006bd8d:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,E
STABLISHED -j ACCEPT
```

The rules in order listed above mean

- we accept external TCP packets to the specific internal host where the destination is 192.168.60.5 if it also has a destination port of 23.
- we accept TCP packets coming from the internal host.
- We make TCP an established and related connection.
- we block all other packets going either way.

This shows that we can access a server on an external host (left side) from an internal host (right side). We can see the message go through.



This shows our internal host can telnet external host.

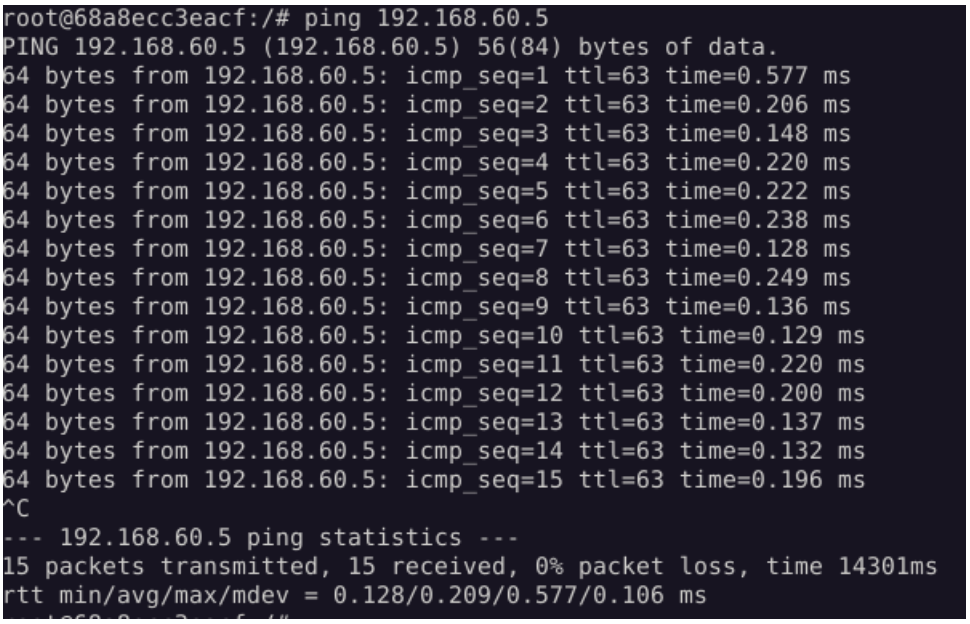
```
root@e8c74841a96d:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
68a8ecc3eacf login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-47-generic x86_64)

 * 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@68a8ecc3eacf:~$
```

With conntrack you consume more resources but each packet can be handled consistently together. Without it we consume less resources but the packets are harder to manage.

7. a) From this we can see the ping (from external pinging internal) is occurring every 6 seconds after seq=7. This is with both commands. This shows the limit was applied.

```
root@68a8ecc3eacf:/# 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=2.21 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.133 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.146 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.136 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.134 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.267 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.129 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.173 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.267 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.257 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.278 ms
64 bytes from 192.168.60.5: icmp_seq=37 ttl=63 time=0.120 ms
64 bytes from 192.168.60.5: icmp_seq=43 ttl=63 time=0.119 ms
64 bytes from 192.168.60.5: icmp_seq=48 ttl=63 time=0.223 ms
64 bytes from 192.168.60.5: icmp_seq=54 ttl=63 time=0.139 ms
64 bytes from 192.168.60.5: icmp_seq=60 ttl=63 time=0.186 ms
64 bytes from 192.168.60.5: icmp_seq=66 ttl=63 time=0.170 ms
64 bytes from 192.168.60.5: icmp_seq=72 ttl=63 time=0.150 ms
64 bytes from 192.168.60.5: icmp_seq=78 ttl=63 time=0.523 ms
64 bytes from 192.168.60.5: icmp_seq=84 ttl=63 time=0.170 ms
64 bytes from 192.168.60.5: icmp_seq=89 ttl=63 time=0.527 ms
64 bytes from 192.168.60.5: icmp_seq=95 ttl=63 time=0.118 ms
64 bytes from 192.168.60.5: icmp_seq=101 ttl=63 time=0.365 ms
^C
--- 192.168.60.5 ping statistics ---
101 packets transmitted, 23 received, 77.2277% packet loss, time 102370ms
rtt min/avg/max/mdev = 0.118/0.301/2.207/0.422 ms
```

b) From this we can see the ping (from external pinging internal) is occurring every second. This is with only the first command.



The second rule is needed as the packets were not being stopped with only the first command. We needed the default in place to enforce the first rule.

8. From our experiments we can conclude that a loadable kernel module runs the risk of crashing the system kernel if hooks are not unregistered properly. This could be an easy mistake to make even if kernels are easy to work with. We did end up finding a better solution in the form of iptables with connection tracking mechanism, conntrack. Although it uses more resources, we can see from it and examine in the firewall the state of a connection's attempt. This will allow for better packet managing. These were seen from problems 6 and 7.

   It is also recommended to have a default rule such as *iptables -P FORWARD DROP* so we can block unnecessary packets and have a fail-safe default.

   If you find something fishy you can limit the network traffic via:
   iptables -A FORWARD -s <IP> -m limit --limit 10/minute \ --limit-burst 6 -j ACCEPT
   iptables -A FORWARD -s <IP> -j DROP

   This is an example of 6 second time burts, but it can be adapted to however long you would like to limit it to, as seen in problem 7 a.