

IT-Sikkerhed, Assignment 4

ITS - 2021

Department of Computer Science
University of Copenhagen

M. Ali
M. Chleih

April 2, 2022

Review Questions

7.5 Why do many DoS attacks use packets with spoofed source addresses?

It is to prevent the attack from being traced back and also to prevent response packets from returning and overwhelming the source computer, since they would be scattered across the internet to various forged source addresses which in turn adds to the flood of traffic directed at the target system.

7.7 What is the difference between a DDoS attack and a classic DoS attack? Why are DDoS attacks considered more potent than classic DoS attacks?

In a classic DoS attack, only a single system is responsible for the attack, whereas a DDoS attack utilizes multiple systems to generate the attack. In a DDoS, the attacker is in control of zombie systems that collectively form a botnet. The botnet is either directly controlled by the attacker or indirectly through handlers. DDoS attacks are more potent since multiple systems are used to generate overwhelming traffic and it is much more difficult to respond to the attack.

9.1 List the different types of firewalls.

- Packet Filtering Firewall.
- Stateful Inspection Firewalls.
- Application-Level Gateway.
- Circuit-Level Gateway.

9.3 Which type of attacks is possible on a packet filtering firewall?

- Packet filtering firewalls can not prevent attacks that employ application-specific vulnerabilities or functions.

- Packet filtering firewalls are susceptible to attacks that target TCP/IP specification vulnerability such as network layer address spoofing. It is because the firewall can not detect if addressing information has been altered.
- IP address spoofing in which the intruder transmits packets from the outside with the source IP address of an internal host.
- Source routing attacks in which the attacker specifies the route that a packet should take to bypass security measures that do not analyze the source routing information.
- Tiny fragment attacks in which the TCP header information is fragmented into separate packet fragments. The attacker hopes that the filtering firewall examines only the first packet fragment circumventing filtering rules that depend on TCP header information which leads to subsequent packets passing through.

9.10 Why is it useful to have host-based firewalls?

- Filtering rules can be customized to the host environment with different filters for different applications.
- Protection is independent of an overarching firewall, adding an extra layer of security when used in conjunction with a stand-alone firewall.

22.10 Explain the transport and tunnel modes of ESP.

Transport Mode

Transport mode provides protection primarily for upper layer protocols. Transport mode protection extends to the payload of an IP packet. Transport mode is used for end-to-end communication between two hosts. When a host runs ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extension headers that are present, with the possible exception of the destination options header, which may be included in the protection.

Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the ESP fields are added to the IP packet, the entire packet plus security fields are treated as the payload of new outer IP packet with a new outer IP header. The entire original, inner, packet travels through a tunnel from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security.

23.6 What is the role of a CA in X.509?

A CA certificate is used only to sign other certificates. Otherwise, the certificate belongs to an "end-user", and may be used for verifying server or client identities, signing or encrypting e-mail or other content, signing executable code, or other uses in applications such as those we listed above.

23.10 How do most current X.509 implementations check the validity of signatures on a certificate?

Most current X.509 implementations used the "trust store" or a large list of CA's and their public keys.

Problems

7.4

DNS Response packet = 100 bytes or 800 bits.

DNS Response packet = 1000 bytes or 8000 bits.

DNS Response packet = 1500 bytes or 12000 bits.

For a packet size of **100 bytes**: $8mb / 800 = 10$ packets per second

For a packet size of **1000 bytes**: $8mb / 8000 = 1$ packet per second

For a packet size of **1500 bytes**: $8mb / 12000 = 666.66$ packets per second

9.11

22.2

a. Man-in-the-middle: This can be prevented by the use of a public key certificate to authenticate the correspondents.

b. Password sniffing: User data is encrypted in SSL.

c. IP spoofing: The one doing the spoofing must be in possession of the secret key as well as the forged IP address.

d. IP hijacking: Encryption prevents this kind of attack.

e. SYN flooding: Unfortunately SSL does not provide any protection against this sort of attack.

Seed Labs

Task 2.A

After executing the given commands:

```
seed@VM: ~/Desktop
root@f44d13dc0fd7:~# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@f44d13dc0fd7:~# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@f44d13dc0fd7:~# iptables -P OUTPUT DROP
root@f44d13dc0fd7:~# iptables -P INPUT DROP
root@f44d13dc0fd7:~#
```

We were able to ping the router but not access the router with telnet as expected.

```
seed@VM: ~/Desktop
[10/09/21]seed@VM:~/Desktop$ dockps
606c2d9b567f  hostA-10.9.0.5
5ae0761218bb  host2-192.168.60.6
a19208ff74f0  host1-192.168.60.5
f44d13dc0fd7  seed-router
d1697e275656  host3-192.168.60.7
69405a5d4685  mysql-10.9.0.6
[10/09/21]seed@VM:~/Desktop$ docksh 60
root@606c2d9b567f:~# ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.054 ms
^C
--- seed-router ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2077ms
rtt min/avg/max/mdev = 0.054/0.060/0.068/0.005 ms
root@606c2d9b567f:~# telnet seed-router
Trying 10.9.0.11...
```

The first 2 commands specify the rule for ping. We allow echo-request for the INPUT chain and echo-reply for the OUTPUT chain. The last 2 rules sets the default rule to drop all other requests/connections.

Task 2.B

Before moving on, here is the name for the different docker containers for reference:

```
606c2d9b567f hostA-10.9.0.5
5ae0761218bb host2-192.168.60.6
a19208ff74f0 host1-192.168.60.5
f44d13dc0fd7 seed-router
d1697e275656 host3-192.168.60.7
69405a5d4685 mysql-10.9.0.6
```

The command for rule 1:

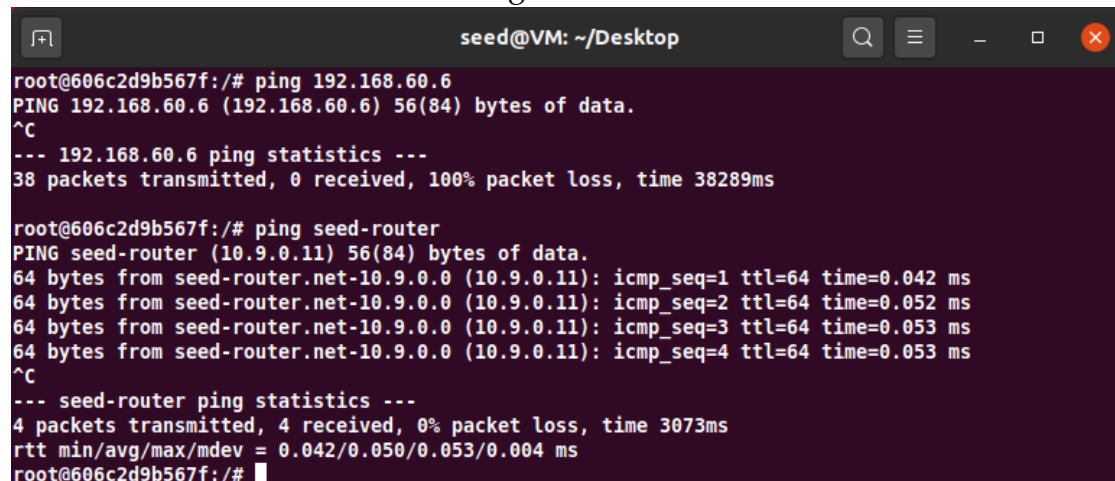
```
iptables -A FORWARD -s 10.9.0.0/24 -d 192.168.60.0/24 -p icmp - -icmp-type echo-request -j DROP
```

The command for rule 4:

```
iptables -P FORWARD DROP
```

Both of these commands were executed in the router container.

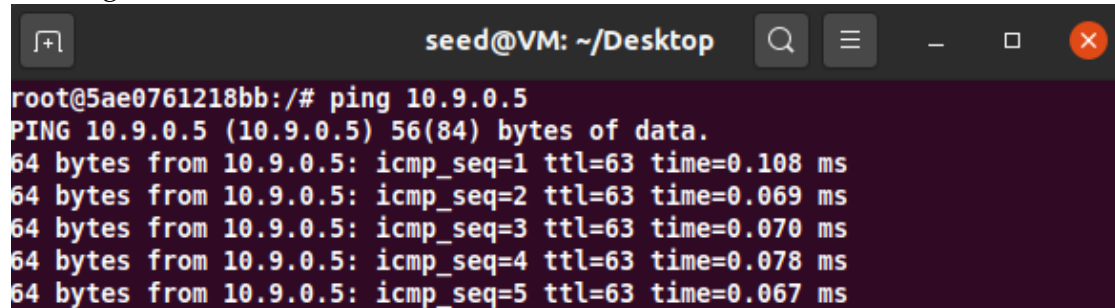
Below is a screenshot demonstrating the behaviour for rule 1 and 2:



```
seed@VM: ~/Desktop
root@606c2d9b567f:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
^C
--- 192.168.60.6 ping statistics ---
38 packets transmitted, 0 received, 100% packet loss, time 38289ms

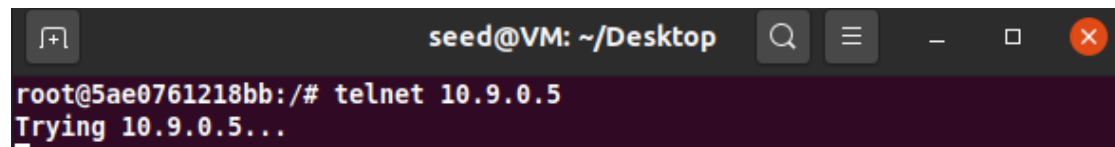
root@606c2d9b567f:/# ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.042 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=4 ttl=64 time=0.053 ms
^C
--- seed-router ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.042/0.050/0.053/0.004 ms
root@606c2d9b567f:/#
```

Below is a screenshot demonstrating the behaviour for rule 3. The container starting with 5ae is the internal host with the IP 192.168.60.6



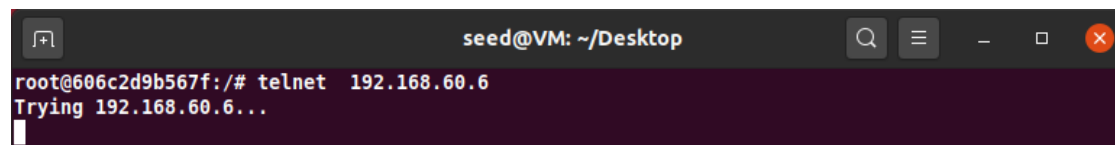
```
seed@VM: ~/Desktop
root@5ae0761218bb:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.108 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.069 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.070 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.078 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.067 ms
```

The two screenshots below demonstrate the behaviour for rule 4.
Internal network to external network:



```
seed@VM: ~/Desktop
root@5ae0761218bb:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

External network to internal network:



```
seed@VM: ~/Desktop
root@606c2d9b567f:/# telnet 192.168.60.6
Trying 192.168.60.6...
```

Task 2.C

We have assumed that what is meant by "access" in the rules, is whether or not we can use telnet.

The commands for rule 1:

```
iptables -A FORWARD -s 10.9.0.0/24 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
and
```

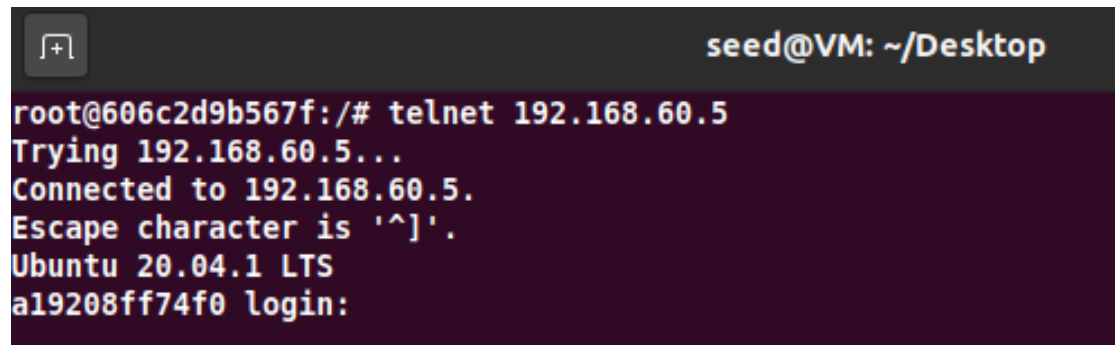
```
iptables -A FORWARD -s 10.9.0.0/24 -d 192.168.60.0/24 -p tcp --dport 23 -j DROP
```

The commands for rule 4:

```
iptables -A FORWARD -s 192.168.60.0/24 -d 10.9.0.0/24 -p tcp --dport 23 -j DROP
```

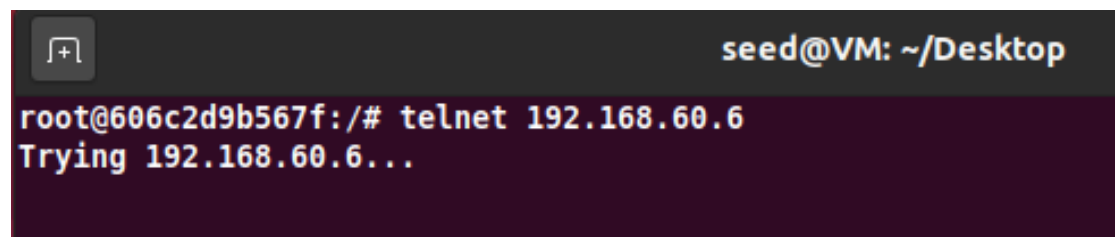

Below are screenshots for rule 1 and 2.

Accessing the server on 192.168.60.5 from the external network:

A terminal window titled 'seed@VM: ~/Desktop' showing a telnet session. The user 'root' on host '606c2d9b567f' initiates a telnet connection to '192.168.60.5'. The connection is successful, and the remote host identifies itself as 'Ubuntu 20.04.1 LTS'. The login prompt 'a19208ff74f0 login:' is displayed.

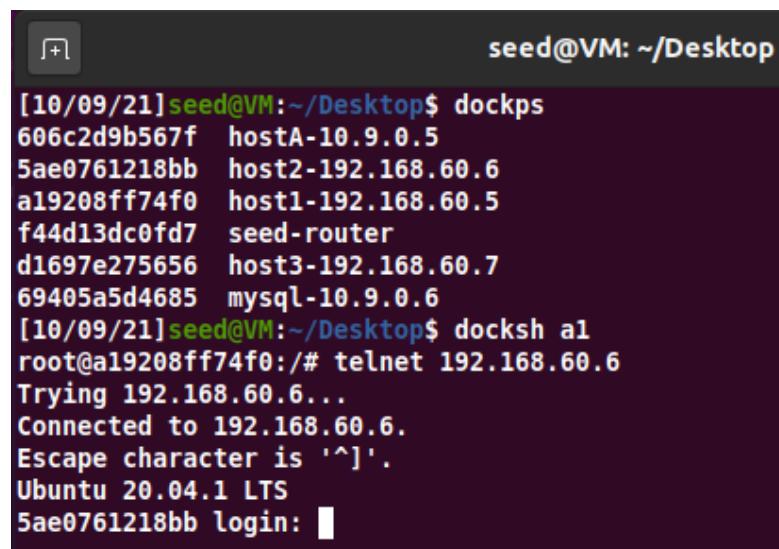
```
seed@VM: ~/Desktop
root@606c2d9b567f:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a19208ff74f0 login:
```

Accessing other internal servers from the external network:

A terminal window titled 'seed@VM: ~/Desktop' showing a telnet session. The user 'root' on host '606c2d9b567f' initiates a telnet connection to '192.168.60.6'. The connection attempt is shown, but the output is truncated.

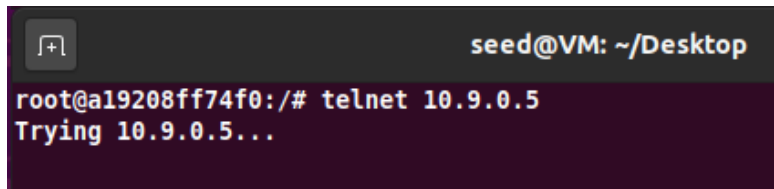
```
seed@VM: ~/Desktop
root@606c2d9b567f:/# telnet 192.168.60.6
Trying 192.168.60.6...
```

Screenshot for rule 3. We are accessing host2 from host1:

A terminal window titled 'seed@VM: ~/Desktop' showing a sequence of commands. First, 'dockps' is used to list containers, showing mappings for hostA, host2, host1, seed-router, host3, and mysql. Then, 'docksh a1' is used to enter a shell on container 'a1'. Finally, a telnet connection is initiated from 'root@606c2d9b567f' to '192.168.60.6', which is successful, showing the 'Ubuntu 20.04.1 LTS' login prompt.

```
seed@VM: ~/Desktop
[10/09/21]seed@VM:~/Desktop$ dockps
606c2d9b567f  hostA-10.9.0.5
5ae0761218bb  host2-192.168.60.6
a19208ff74f0  host1-192.168.60.5
f44d13dc0fd7  seed-router
d1697e275656  host3-192.168.60.7
69405a5d4685  mysql-10.9.0.6
[10/09/21]seed@VM:~/Desktop$ docksh a1
root@606c2d9b567f:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5ae0761218bb login: 
```

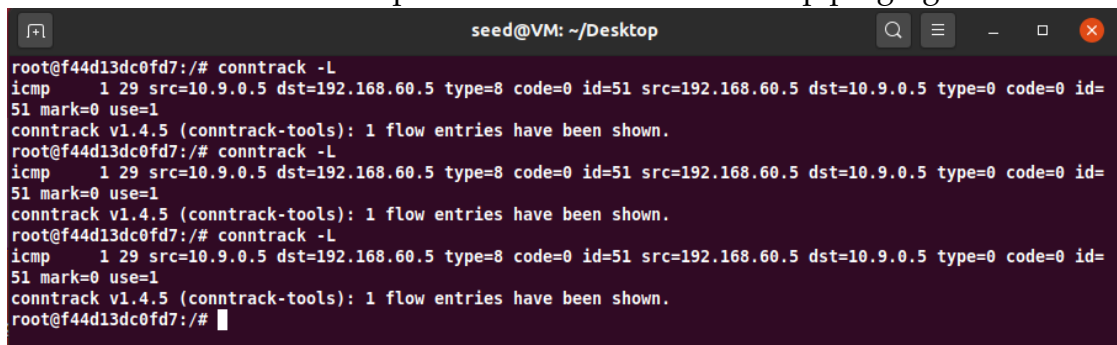
Screenshot for rule 4.



```
seed@VM: ~/Desktop
root@a19208ff74f0:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

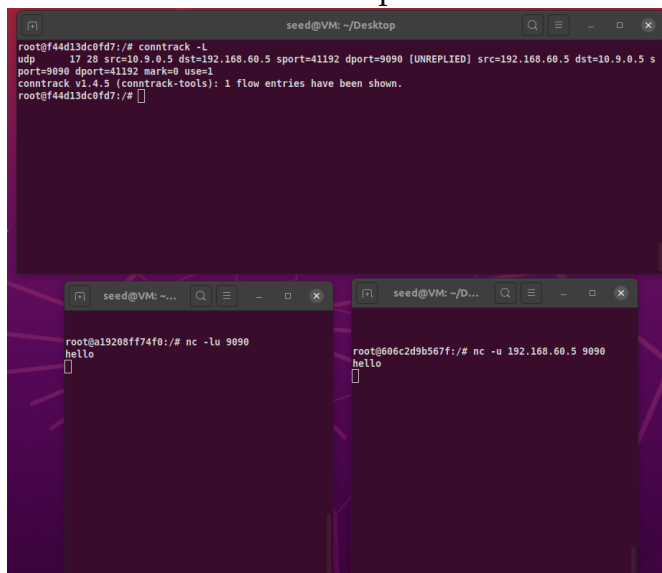
Task 3.A

- ICMP: The connection is kept for 30 seconds after we stop pinging.



```
seed@VM: ~/Desktop
root@f44d13dc0fd7:/# conntrack -L
icmp    1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:/# conntrack -L
icmp    1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:/# conntrack -L
icmp    1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=51 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=51 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:/#
```

- UDP: The connection is kept for 28 seconds after we end the connection.

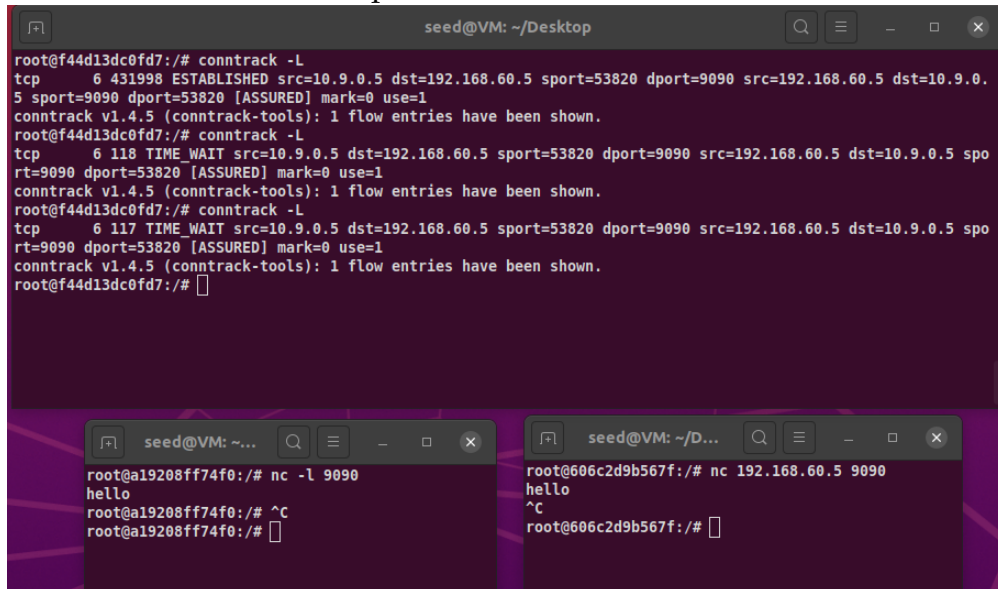


```
seed@VM: ~/Desktop
root@f44d13dc0fd7:/# conntrack -L
udp     17 28 src=10.9.0.5 dst=192.168.60.5 sport=41192 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=41192 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:/#
```

```
seed@VM: ~/Desktop
root@a19208ff74f0:/# nc -lu 9090
hello
```

```
seed@VM: ~/Desktop
root@606c2d9b567f:/# nc -u 192.168.60.5 9090
hello
```

- TCP: The connection is kept for 118 seconds after it ends.



```
seed@VM: ~/Desktop
root@f44d13dc0fd7:~# conntrack -L
tcp        6 431998 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=53820 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=53820 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:~# conntrack -L
tcp        6 118 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=53820 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=53820 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:~# conntrack -L
tcp        6 117 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=53820 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=53820 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f44d13dc0fd7:~#
```

```
seed@VM: ~...
root@a19208ff74f0:~# nc -l 9090
hello
root@a19208ff74f0:~# ^C
root@a19208ff74f0:~#
```

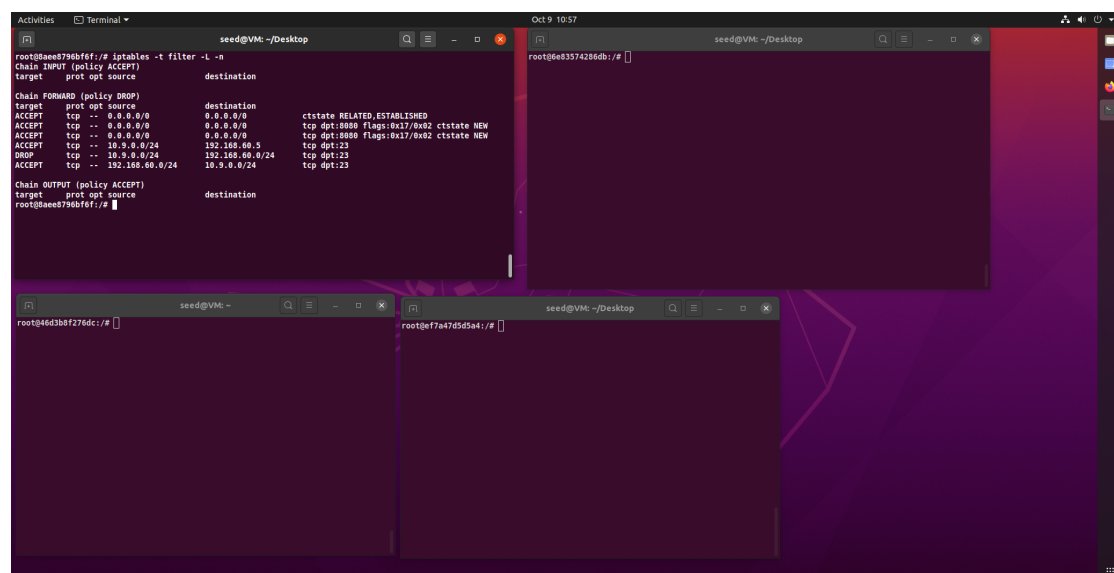
```
seed@VM: ~/D...
root@606c2d9b567f:~# nc 192.168.60.5 9090
hello
^C
root@606c2d9b567f:~#
```

Task 3.B

We have used the same commands as in 2.C with the only new one being

```
iptables -A FORWARD -s 192.168.60.0/24 -d 10.9.0.0/24 -p tcp --dport 23 -j ACCEPT
```

Here is the filter table with all of the rules:



```
seed@VM: ~/Desktop
root@baee879dbf6f:~# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target    prot opt source               destination

Chain FORWARD (policy DROP)
target    prot opt source               destination
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0             ctstate RELATED,ESTABLISHED
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0             tcp dpt:8080 flags:0x17/0x02 ctstate NEW
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0             tcp dpt:8080 flags:0x17/0x02 ctstate NEW
ACCEPT    tcp  --  10.9.0.0/24           192.168.60.5          tcp dpt:23
DROP      tcp  --  10.9.0.0/24           192.168.60.0/24       tcp dpt:23
ACCEPT    tcp  --  192.168.60.0/24       10.9.0.0/24           tcp dpt:23

Chain OUTPUT (policy ACCEPT)
target    prot opt source               destination

root@baee879dbf6f:~#
```

The rules work as expected, but we are unsure of the difference between the two approaches, therefore we are unfortunately unable to provide input.