Network Security

Build a Linux Firewall Report

VMs Setup

As recommended for the assignment 3 VMs with Linux distribution of Debian 9 with the LXDE Window Manager from live CD was used. Each VM was given 1GB of RAM and 4GB of storage (just in case).

The VM specification:

- Kernel release: 4.9.0-4-686
- Kernel version: SMP Debian 4.9.65-3
- Operating system: Debian 9 GNU/Linux i686

As described two networks were used:

- client-net: 192.168.100.0/24
- server-net: 192.168.101.0/24

Client VM

1. Client has one network interface as described: 192.168.100.2/24
2. This was achieved by modifying /etc/network/interfaces configuration file which had the following structure after modification:

```
user@debian:~$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto enp0s3
iface enp0s3 inet static
    address 192.168.100.2
    netmask 255.255.255.0
    gateway 192.168.100.1
    network 192.168.100.0
```

3. /etc/hosts file was modified where two lines for name translation were added:

```
192.168.101.2   server
192.168.100.1   router
```

4. After the configuration client network setup looks like that.

```
user@debian:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.2  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::a00:27ff:fe7b:3fdc  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:7b:3f:dc  txqueuelen 1000  (Ethernet)
        RX packets 112567  bytes 64270064 (61.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 75494  bytes 5574292 (5.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Server VM

1. Server has one network interface as described: 192.168.101.2/24
2. This was achieved by modifying /etc/network/interfaces configuration file which had the following structure after modification:

```
user@debian:~$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto enp0s3
iface enp0s3 inet static
    address 192.168.101.2
    netmask 255.255.255.0
    gateway 192.168.101.1
    network 192.168.101.0
```

3. /etc/hosts file was modified where two lines for name translation were added:

   192.168.100.2 client 192.168.101.1 router

4. After the configuration client network setup looks like that.

```
user@debian:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.101.2  netmask 255.255.255.0  broadcast 192.168.101.255
        inet6 fe80::a00:27ff:fe7b:3fdc  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:7b:3f:dc  txqueuelen 1000  (Ethernet)
        RX packets 112486  bytes 64263735 (61.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 74955  bytes 5484664 (5.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Router VM

1. Router has two network interface as described:
   - 192.168.100.1/24, connected to client-net
   - 192.168.101.1/24, connected to server-net.
2. This was achieved by modifying /etc/network/interfaces configuration file which had the following structure after modification:

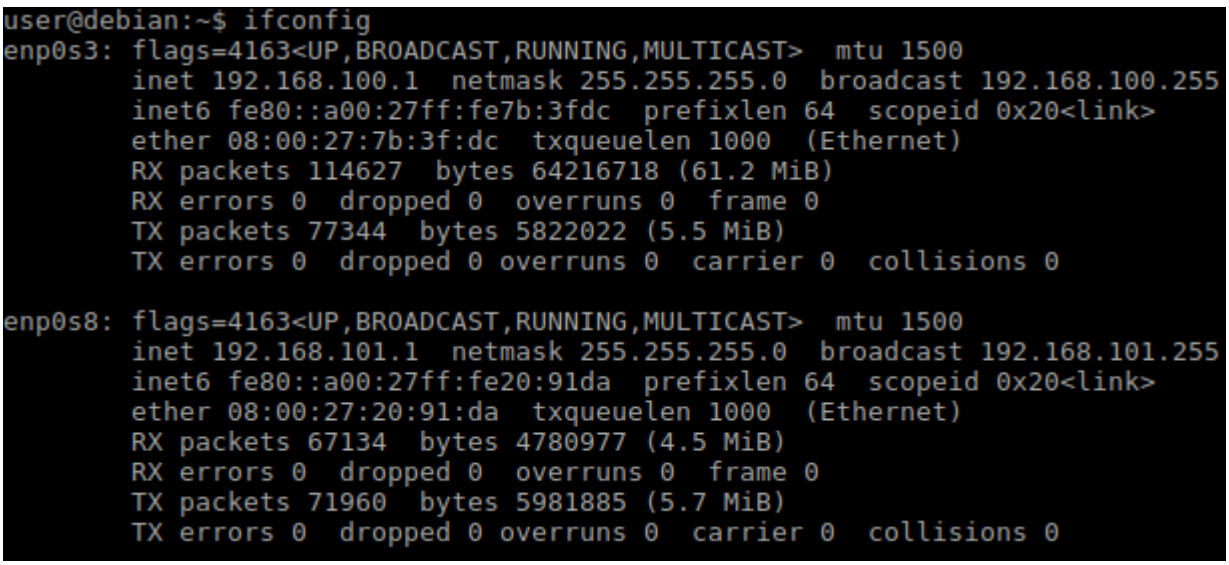
```
user@debian:~$ cat /etc/network/interfaces
auto lo
iface lo inet loopback
```

```
# client-net
auto enp0s3
iface enp0s3 inet static
    address 192.168.100.1
    netmask 255.255.255.0
    network 192.168.100.0

# server-net
auto enp0s8
iface enp0s8 inet static
    address 192.168.101.1
    netmask 255.255.255.0
    network 192.168.101.0
```

3. /etc/hosts file was modified where two lines for name translation were added:

   192.168.101.2 server 192.168.100.2 client

4. After the configuration router setup looks like that.



Tools Used

- telnet -tool for testing communication to another node on the network
- nc -tool for testing the communication to other nodes in the network and to test whether the connection times out to the specific ports.
- wget -for downloading/fetching the the index.html pages from other nodes on the network
- Apache2 -web server running on all the nodes to test the connections on port 80.
- ssh -for testing the connections on the port 22.
- wireshark -for capturing the traffic and testing the correctness of the firewall rules.
- nmap -for scanning the nodes on the network in order to verify the correctness of the firewall rules.
- iptables -creating, deleting and modifying firewall rules.

Main Testing

Server Part

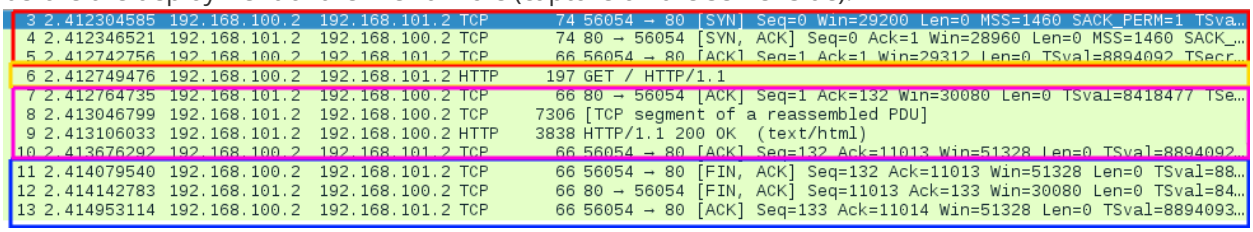
Before Firewall Deployment

1. nmap scan before the deployment of the rule:

```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 09:39 UTC
Nmap scan report for server (192.168.101.2)
Host is up (0.00029s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds
```

   open means that an application on the target machine is listening for connections/packets on that port. It clearly shows that two services (ssh on port 22 and http on port 80) are listening for incoming connections.

2. Wireshark capture of the client fetching the index page with thefollowing command: `wget server` before the deployment of the firewall rule (capture on the server side):



- In the red outline we see the TCP connection establishment (handshake).
- In yellow outline (packet 6) we see the HTTP GET request from the client to the server.
- In purple outline we see the payload being sent from the server to the client and all the packets being ACK(nowledged) in the packet 10.
- In the blue outline we see the finish of the TCP connection.

  From the capture we see that full connection between the client and the server was established, payload carried from the server to the client and graceful connection finish. This and Nmap scan clearly demonstrates that connection between the client and the server is fully working.

Part 2

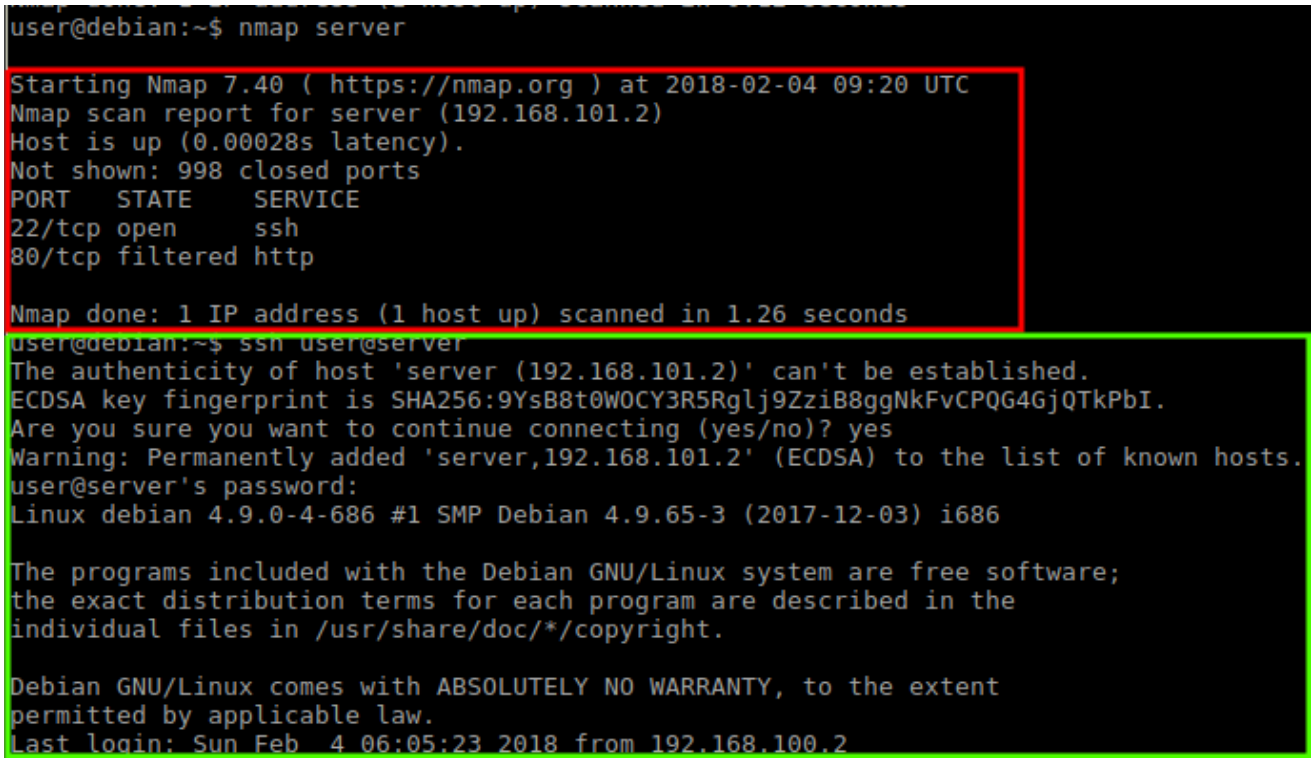
Blocking the access to port 80 so that fetching index page times out. Dropping all the traffic on port 80 so that no result is returned to the caller.

After deploying the firewall rules from the script **part2.sh** all the traffic to the port 80 was blocked. This was verified by the nmap scan and Wireshark capture on the server.

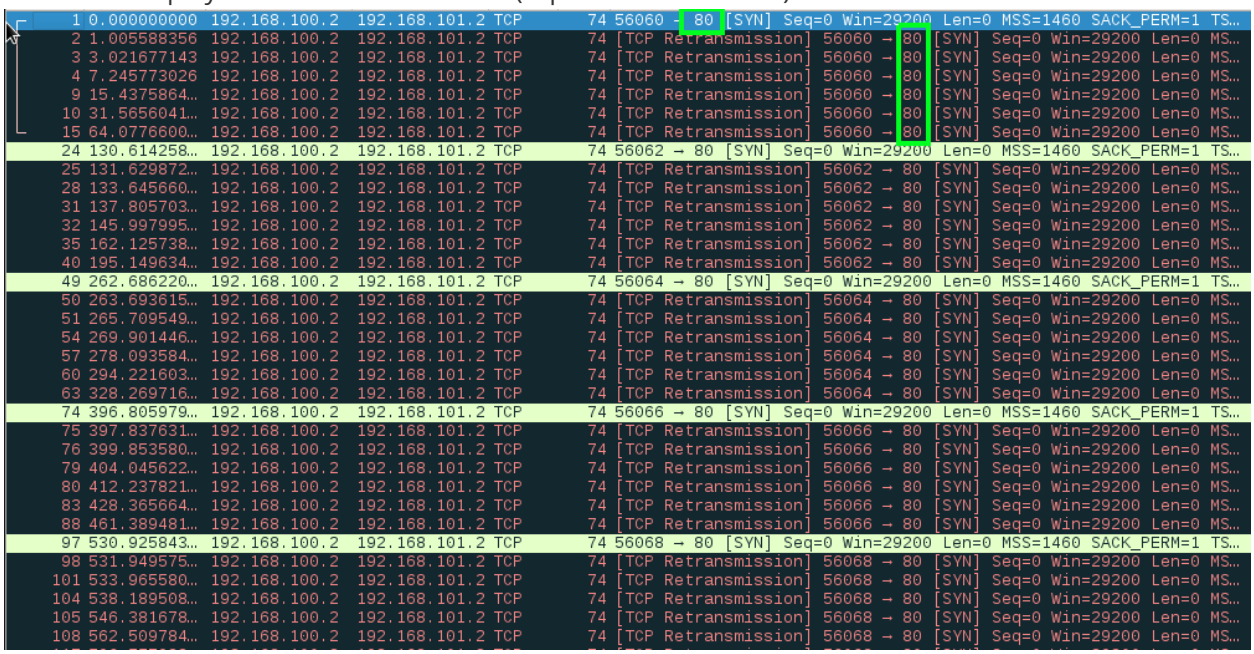
1. nmap scan after deployment of **part2.sh** script:

```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 09:39 UTC
Nmap scan report for server (192.168.101.2)
Host is up (0.00028s latency).
Not shown: 998 closed ports
PORT   STATE    SERVICE
22/tcp open     ssh
80/tcp filtered http

Nmap done: 1 IP address (1 host up) scanned in 1.24 seconds
```

We see that the connection to the port 80 is **filtered** which means that Nmap a firewall, filter, or other network obstacle is blocking the port so that Nmap cannot tell whether it is open or closed. This happens due to the fact that all the connections to the port 80 are dropped rather than rejected and caller does not receive any feedback about the connection to port 80 while in the reject case it would send a notification to the caller and would allow to clearly identify the information about the port. It is also clear from picture that ssh connection is still open and this was verified by connecting to the server via ssh. This can be seen in the image below where in red we see the nmap scan and then in the green ssh connection happening from the client to the server.



2. Wireshark capture of the client fetching the index page with the following command: `wget server` before the deployment of the firewall rule (capture on the server):



In the capture we see the client trying to establish the TCP connection to the server by sending a SYN packet on the port 80 (green outline). However, because the firewall DROPs all the packets, the server

does not send anything back to the client. We can see the retransmission happening from the client (source IP is always from the client (192.168.100.2)).

3. We can see the failed `wget server` operation on the client (as expected client times out which means the server is dropping the packets it should):

```
user@debian:~$ wget server
--2018-02-04 08:54:56--  http://server/
Resolving server (server)... 192.168.101.2
Connecting to server (server)|192.168.101.2|:80... failed: Connection timed out.
Retrying.

--2018-02-04 08:57:06--  (try: 2)  http://server/
Connecting to server (server)|192.168.101.2|:80... failed: Connection timed out.
Retrying.

--2018-02-04 08:59:18--  (try: 3)  http://server/
Connecting to server (server)|192.168.101.2|:80... failed: Connection timed out.
Retrying.

--2018-02-04 09:01:33--  (try: 4)  http://server/
Connecting to server (server)|192.168.101.2|:80... failed: Connection timed out.
Retrying.
```

4. Therefore, it is verified that the firewall is blocking what it should block (i.e. connections to port 80) and allowing all the other connections (such as SSH to port 22) to be established. Also, because Nmap returns filtered to the single port 80 it means that all the other ports rejects the connections by sending back the [RST,ACK] packet while only port 80 connections completely dropped the packets.

Part 3

Blocking the access to all the ports except for port 22 on the server so that ssh still works but connections to all the other ports time out. Basically all the INPUT traffic is dropped except for port 22.

After deploying the firewall rules from the script **part3.sh** all the traffic was blocked except for port 22 and all the SSH connections were available. This was verified by the Nmap, nc scan and Wireshark capture on the server.

1. Nmap scan after deployment of **part3.sh** script:

```
user@debian:~$ nmap -Pn server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 10:49 UTC
Nmap scan report for server (192.168.101.2)
Host is up (0.00048s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 4.34 seconds
```

   This time we use Nmap with **-Pn** option because the server drops all the packets and normally Nmap would only perform heavy port scanning when the host found is up. Therefore if we simply run Nmap without any options we get following behavior because all the ports drops the packets:

```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 09:51 UTC
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.05 seconds
```

From the advanced **-Pn** version we see that only a single port 22 for SSH connections is open. This means that all the other ports drops or rejects the packets. The standard version of Nmap, however, shows that none of the ping requests were returned back to the caller what implies that all of the ports refused connection and DROPed the packets as required.

2. SSH connection from the client to the server after deployment of **part3.sh**:

```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 09:51 UTC
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.05 seconds
user@debian:~$ ssh user@server
user@server's password:
Linux debian 4.9.0-4-686 #1 SMP Debian 4.9.65-3 (2017-12-03) i686

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Feb  4 11:39:13 2018 from 192.168.100.2
```

After deployment of part3.sh we can again see that standard Nmap fails (red outline), however, the SSH connection is successfully established (green outline).

This can be confirmed by the wireshark capture of the connection on the server:

1	0.000000000	192.168.100.2	192.168.101.2	TCP	74	49982 → 22 [SYN] Seq=0 Win=29200 Len=0 M
2	0.000046189	192.168.101.2	192.168.100.2	TCP	74	22 → 49982 [SYN, ACK] Seq=0 Ack=1 Win=28
3	0.000495310	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=1 Ack=1 Win=29312 L
4	0.000711097	192.168.100.2	192.168.101.2	SSHv2	106	Client: Protocol (SSH-2.0-OpenSSH_7.4p1
5	0.000723467	192.168.101.2	192.168.100.2	TCP	66	22 → 49982 [ACK] Seq=1 Ack=41 Win=29056
6	0.006337390	192.168.101.2	192.168.100.2	SSHv2	105	Server: Protocol (SSH-2.0-OpenSSH_7.4p1
7	0.007671133	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=41 Ack=40 Win=29312
8	0.007695114	192.168.101.2	192.168.100.2	SSHv2	1146	Server: Key Exchange Init
9	0.008049439	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=41 Ack=1120 Win=321
10	0.008792841	192.168.100.2	192.168.101.2	SSHv2	1498	Client: Key Exchange Init
11	0.052245025	192.168.101.2	192.168.100.2	TCP	66	22 → 49982 [ACK] Seq=1120 Ack=1473 Win=3
12	0.052746879	192.168.100.2	192.168.101.2	SSHv2	114	Client: Diffie-Hellman Key Exchange Init
13	0.052762372	192.168.101.2	192.168.100.2	TCP	66	22 → 49982 [ACK] Seq=1120 Ack=1521 Win=3
14	0.058198067	192.168.101.2	192.168.100.2	SSHv2	486	Server: Diffie-Hellman Key Exchange Repl
15	0.064619769	192.168.100.2	192.168.101.2	SSHv2	82	Client: New Keys
16	0.108300287	192.168.101.2	192.168.100.2	TCP	66	22 → 49982 [ACK] Seq=1540 Ack=1537 Win=3
17	0.108906258	192.168.100.2	192.168.101.2	SSHv2	110	Client: Encrypted packet (len=44)
18	0.108924245	192.168.101.2	192.168.100.2	TCP	66	22 → 49982 [ACK] Seq=1540 Ack=1581 Win=3
19	0.109017415	192.168.101.2	192.168.100.2	SSHv2	110	Server: Encrypted packet (len=44)
20	0.109425413	192.168.100.2	192.168.101.2	SSHv2	126	Client: Encrypted packet (len=60)
21	0.111223651	192.168.101.2	192.168.100.2	SSHv2	118	Server: Encrypted packet (len=52)
22	0.154136211	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=1641 Ack=1636 Win=3
23	4.040184583	192.168.100.2	192.168.101.2	SSHv2	214	Client: Encrypted packet (len=148)
24	4.057884694	192.168.101.2	192.168.100.2	SSHv2	94	Server: Encrypted packet (len=28)
25	4.058310018	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=1789 Ack=1664 Win=3
26	4.058375830	192.168.100.2	192.168.101.2	SSHv2	178	Client: Encrypted packet (len=112)
27	4.066017554	192.168.101.2	192.168.100.2	SSHv2	566	Server: Encrypted packet (len=500)
28	4.110175263	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=1901 Ack=2164 Win=3
29	4.110214599	192.168.101.2	192.168.100.2	SSHv2	110	Server: Encrypted packet (len=44)
30	4.110623249	192.168.100.2	192.168.101.2	TCP	66	49982 → 22 [ACK] Seq=1901 Ack=2208 Win=3

It is clearly visible that connection is established and Diffie-Helman handshake is made from both and and the information passed between to hosts. This again confirms that SSH connection is still working with the firewall rules.

3. Wireshark capture of the telnet connection:

1	0.000000000	192.168.100.2	192.168.101.2	TCP	74	57208 → 1234 [SYN] Seq=0 Win=29200 Len=0
2	1.017918859	192.168.100.2	192.168.101.2	TCP	74	[TCP Retransmission] 57208 → 1234 [SYN]
3	3.034081265	192.168.100.2	192.168.101.2	TCP	74	[TCP Retransmission] 57208 → 1234 [SYN]
4	7.161849329	192.168.100.2	192.168.101.2	TCP	74	[TCP Retransmission] 57208 → 1234 [SYN]
5	15.3537643…	192.168.100.2	192.168.101.2	TCP	74	[TCP Retransmission] 57208 → 1234 [SYN]
8	31.4821055…	192.168.100.2	192.168.101.2	TCP	74	[TCP Retransmission] 57208 → 1234 [SYN]
9	64.2501829…	192.168.100.2	192.168.101.2	TCP	74	[TCP Retransmission] 57208 → 1234 [SYN]

We see the TCP SYN packets being send from the client to the server, however, because the packets are being dropped by the firewall on the server, the server does not send any ACKs and RST back to the client.

The call from the client to random port 1234 can be seen below which verifies that the connection times out:

```
user@debian:~$ telnet server 1234
Trying 192.168.101.2...
telnet: Unable to connect to remote host: Connection timed out
user@debian:~$
```

4. Additional testing with netcat: I performed additional testing with the netcat to make sure that all the connections times out and the port 22 notifies that it is still open. I produced the following simple script:

```
# check if the connection times out or not # print all the open connections
function connect(){
    # -w 2 flag sets the timeout for the connection to 2 seconds
    echo "QUIT" | nc -w 2 server "$1" > /dev/null 2>&1
    if [ $? -eq 0 ]; then
        echo "port $1 is open"
    fi
}

# run the connection to all of the ports
for i in {1..65535} ; do
    connect "$i" &
done

sleep 10
```

After the run the following results were received which further confirms that only port 22 is open and other ports DROPs the connections which eventually times out:

```
user@debian:~$ ./test_part3.sh
port 22 is open
user@debian:~$
```

5. To sum up, we got the following behavior: all the connections to the server are dropped and none of the feedback is returned to the caller except for the connections to the port 22 which allows to establish the SSH connection.

Router Part

Part 4

Because the description of this part was quite vague, I have tried to come up with the most sensible solution. I used a firewall on the router to filter out the traffic coming from the client-net to the server-net which meant that firewall on the router will be dealing with FORWARD packets rather than INPUT packets. I split this part into 2:

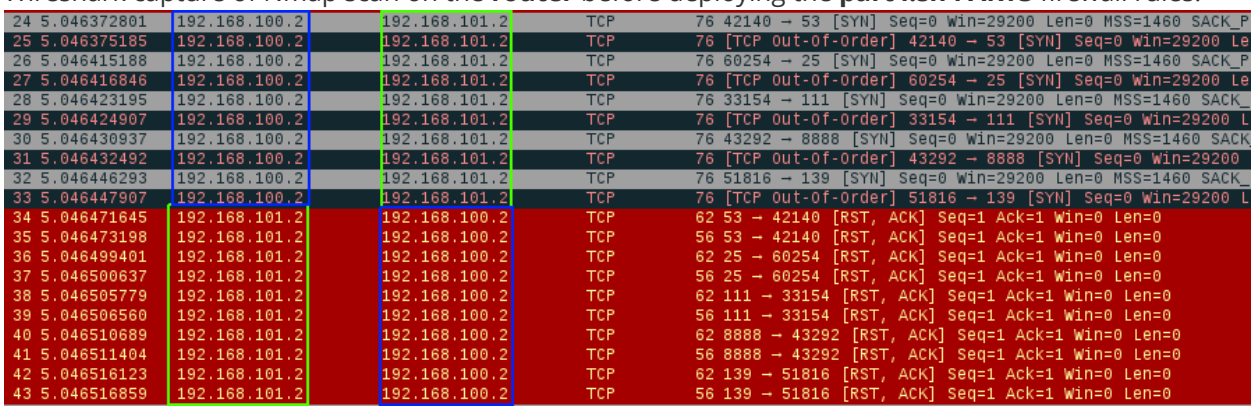
- Running the part 2 description by: `sudo ./part4.sh PART2` with **PART2** flag set.
- Running the part 3 description by: `sudo ./part4.sh PART3` with **PART3** flag set.

Part 4 (PART3 testing)

Blocking the access to all ports except for destination port 22 from the **client-net** to the **server-net** in the router so that only SSH connection passes the firewall on the router to the server-net. All the connections from the client-net which are not for port 22 are dropped. Server is still able to send all the packets out of the server-net to the client-net. In the end I added `-d 192.168.101.2` to the rule (`/sbin/iptables -A FORWARD -i "$client_net" -o "$server_net" -d 192.168.101.2 -p tcp --dport 22 -j ACCEPT`) accepting SSH connections from the client-net to the server-net which are only meant for the server itself. I consider this as a sensible solution as in the real world we would have a server on the network on which we would like to enable SSH, however, for other nodes on the same network we might not want to have access to port 22 open. If it is not the case only the part `-d 192.168.101.2` should be removed.

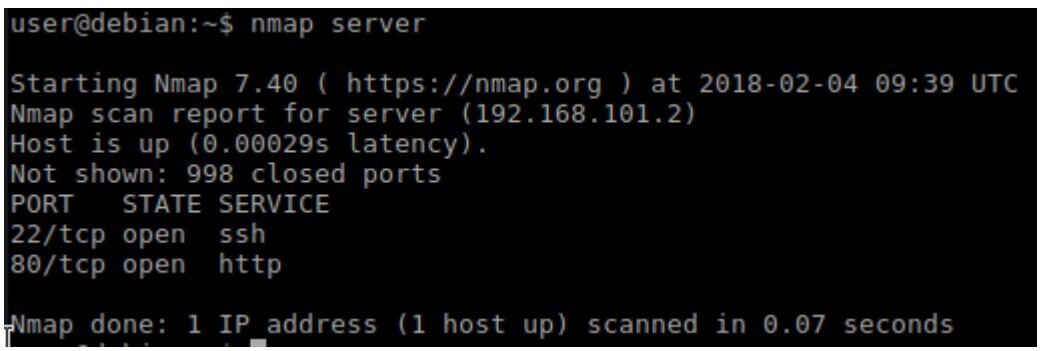
After deploying the firewall rules from the script **part4.sh PART3** all the traffic was blocked except for port 22 and all the SSH connections were available. This was verified by the Nmap, nc scan and Wireshark capture on both the router and the server.

1. Wireshark capture of Nmap scan on the **router** before deploying the **part4.sh PART3** firewall rules:



24 5.046372801	192.168.100.2	192.168.101.2	TCP	76 42140 → 53 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P
25 5.046375185	192.168.100.2	192.168.101.2	TCP	76 [TCP Out-Of-Order] 42140 → 53 [SYN] Seq=0 Win=29200 Le
26 5.046415188	192.168.100.2	192.168.101.2	TCP	76 60254 → 25 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P
27 5.046416846	192.168.100.2	192.168.101.2	TCP	76 [TCP Out-Of-Order] 60254 → 25 [SYN] Seq=0 Win=29200 Le
28 5.046423195	192.168.100.2	192.168.101.2	TCP	76 33154 → 111 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_
29 5.046424907	192.168.100.2	192.168.101.2	TCP	76 [TCP Out-Of-Order] 33154 → 111 [SYN] Seq=0 Win=29200 L
30 5.046430937	192.168.100.2	192.168.101.2	TCP	76 43292 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK
31 5.046432492	192.168.100.2	192.168.101.2	TCP	76 [TCP Out-Of-Order] 43292 → 8888 [SYN] Seq=0 Win=29200
32 5.046446293	192.168.100.2	192.168.101.2	TCP	76 51816 → 139 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_
33 5.046447907	192.168.100.2	192.168.101.2	TCP	76 [TCP Out-Of-Order] 51816 → 139 [SYN] Seq=0 Win=29200 L
34 5.046471645	192.168.101.2	192.168.100.2	TCP	62 53 → 42140 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
35 5.046473198	192.168.101.2	192.168.100.2	TCP	56 53 → 42140 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36 5.046499401	192.168.101.2	192.168.100.2	TCP	62 25 → 60254 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37 5.046500637	192.168.101.2	192.168.100.2	TCP	56 25 → 60254 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
38 5.046505779	192.168.101.2	192.168.100.2	TCP	62 111 → 33154 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
39 5.046506560	192.168.101.2	192.168.100.2	TCP	56 111 → 33154 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
40 5.046510689	192.168.101.2	192.168.100.2	TCP	62 8888 → 43292 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
41 5.046511404	192.168.101.2	192.168.100.2	TCP	56 8888 → 43292 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
42 5.046516123	192.168.101.2	192.168.100.2	TCP	62 139 → 51816 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
43 5.046516859	192.168.101.2	192.168.100.2	TCP	56 139 → 51816 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

   The scan clearly demonstrates that packets are flowing from and to the both ends. Client (blue outline) (192.168.100.2) sends multiple SYN requests to the server (green outline) (192.168.101.2). Later server responds to those packets by sending packet with RST,ACK flags set. Therefore, the communication between the server and the client are allowed to pass arbitrary packets. Therefore, with nmap scan we get the full scan of the server:



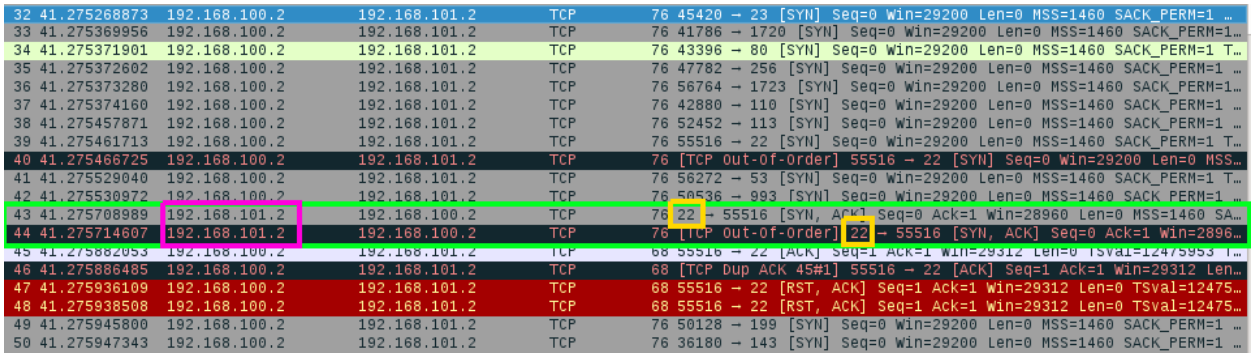
```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 09:39 UTC
Nmap scan report for server (192.168.101.2)
Host is up (0.00029s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds
```

2. Nmap scan on the **client** after deploying the **part4.sh PART3** firewall rules:

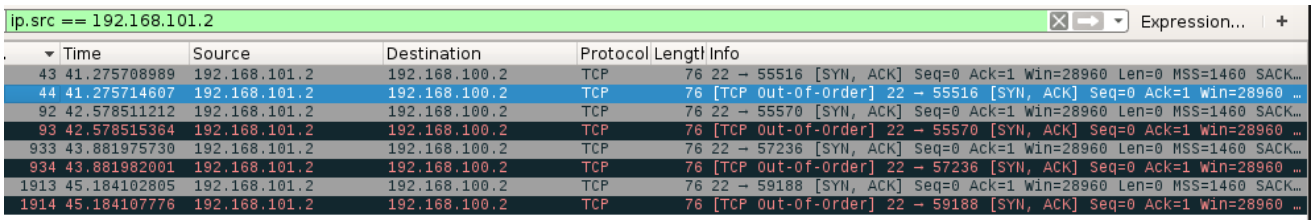
```
user@debian:~$ nmap -Pn server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 12:40 UTC
Nmap scan report for server (192.168.101.2)
Host is up (0.00063s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 6.54 seconds
```

   After deploying the rules it is clear that for the client only a SSH connection is open (red outline) because all the other ports are filtered as seen in the green outline meaning that no response to those requests have been returned and they were dropped by the router firewall.

3. Nmap scan captured with Wireshark on the **router** after deploying the **part4.sh PART3** firewall rules. This was the capture of the scan above (step 2):



   Now all the requests from client are being dropped on the router firewall. This can be seen by exploring the source address which is almost always the one of the client (192.168.100.2) with a single exception. Because router allows connections from client to server on port 22 only. These are the packets that gets through the routers firewall to the server. When the server receives those it sends the SYN,ACK back to the client. This is seen in the green outline where server's IP is in purple and the ports from which the server responds to the client are 22 (yellow outline). This verifies the fact that the server receives ping to the port 22 only. We can make sure that this is the case by applying filter to the router of the same capture to make sure that responses from the server are only made from the port 22:



This fact was also verified by running the Wireshark capture on the **server** as well during the Nmap scan:

	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.2	192.168.101.2	TCP	74	59552 → 22 [SYN] Seq=0 Win=29200
2	0.000058410	192.168.101.2	192.168.100.2	TCP	74	22 → 59552 [SYN, ACK] Seq=0 Ack=1
3	0.000314115	192.168.100.2	192.168.101.2	TCP	66	59552 → 22 [ACK] Seq=1 Ack=1 Win=
4	0.000334028	192.168.100.2	192.168.101.2	TCP	66	59552 → 22 [RST, ACK] Seq=1 Ack=1
5	1.302165348	192.168.100.2	192.168.101.2	TCP	74	60354 → 22 [SYN] Seq=0 Win=29200
6	1.302235358	192.168.101.2	192.168.100.2	TCP	74	22 → 60354 [SYN, ACK] Seq=0 Ack=1
7	1.302674808	192.168.100.2	192.168.101.2	TCP	66	60354 → 22 [ACK] Seq=1 Ack=1 Win=
8	1.302851451	192.168.100.2	192.168.101.2	TCP	66	60354 → 22 [RST, ACK] Seq=1 Ack=1
9	2.604774188	192.168.100.2	192.168.101.2	TCP	74	34074 → 22 [SYN] Seq=0 Win=29200
10	2.604816003	192.168.101.2	192.168.100.2	TCP	74	22 → 34074 [SYN, ACK] Seq=0 Ack=1
11	2.605240964	192.168.100.2	192.168.101.2	TCP	66	34074 → 22 [ACK] Seq=1 Ack=1 Win=
12	2.605263332	192.168.100.2	192.168.101.2	TCP	66	34074 → 22 [RST, ACK] Seq=1 Ack=1

We can see that SYN requests on the server were received to the port 22 only. This confirms the correct firewall behavior on the router because only requests to port 22 are passed throw by the router.

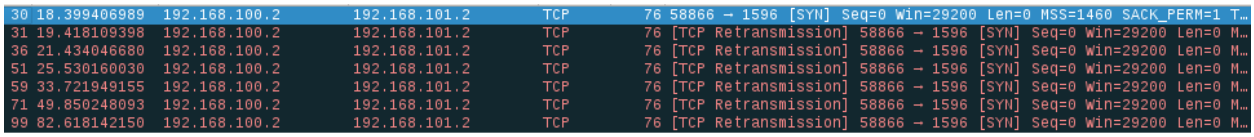
4. After deployment of **part4.sh PART3** firewall rules the SSH connections were passing from client to the server through the router. There was no difference in the Wireshark captures before and after the rules were deployed. The capture on the **router** of the SSH between the server and the client happening can be seen below:

12	8.282934431	192.168.100.2	192.168.101.2	TCP	76 39374 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T…
13	8.282949180	192.168.100.2	192.168.101.2	TCP	76 [TCP Out-Of-Order] 39374 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS…
14	8.283259767	192.168.101.2	192.168.100.2	TCP	76 22 → 39374 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SA…
15	8.283266169	192.168.101.2	192.168.100.2	TCP	76 [TCP Out-Of-Order] 22 → 39374 [SYN, ACK] Seq=0 Ack=1 Win=2896…
16	8.283472442	192.168.100.2	192.168.101.2	TCP	68 39374 → 22 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=12592817 T…
17	8.283478364	192.168.100.2	192.168.101.2	TCP	68 [TCP Dup ACK 16#1] 39374 → 22 [ACK] Seq=1 Ack=1 Win=29312 Len…
18	8.283813558	192.168.100.2	192.168.101.2	SSHv2	108 Client: Protocol (SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u2)
19	8.283819496	192.168.100.2	192.168.101.2	TCP	108 [TCP Retransmission] 39374 → 22 [PSH, ACK] Seq=1 Ack=1 Win=29…
20	8.284019001	192.168.101.2	192.168.100.2	TCP	68 22 → 39374 [ACK] Seq=1 Ack=41 Win=29056 Len=0 TSval=14672695 …
21	8.284023481	192.168.101.2	192.168.100.2	TCP	68 [TCP Dup ACK 20#1] 22 → 39374 [ACK] Seq=1 Ack=41 Win=29056 Le…
22	8.291420941	192.168.101.2	192.168.100.2	SSHv2	107 Server: Protocol (SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u2)
23	8.291431040	192.168.101.2	192.168.100.2	TCP	107 [TCP Retransmission] 22 → 39374 [PSH, ACK] Seq=1 Ack=41 Win=2…
24	8.291675979	192.168.100.2	192.168.101.2	TCP	68 39374 → 22 [ACK] Seq=41 Ack=40 Win=29312 Len=0 TSval=12592819…
25	8.291680896	192.168.100.2	192.168.101.2	TCP	68 [TCP Dup ACK 24#1] 39374 → 22 [ACK] Seq=41 Ack=40 Win=29312 L…
26	8.292017078	192.168.101.2	192.168.100.2	SSHv2	1148 Server: Key Exchange Init
27	8.292021540	192.168.101.2	192.168.100.2	TCP	1148 [TCP Retransmission] 22 → 39374 [PSH, ACK] Seq=40 Ack=41 Win=…
28	8.292158366	192.168.100.2	192.168.101.2	TCP	68 39374 → 22 [ACK] Seq=41 Ack=1120 Win=32128 Len=0 TSval=125928…
29	8.292162884	192.168.100.2	192.168.101.2	TCP	68 [TCP Dup ACK 28#1] 39374 → 22 [ACK] Seq=41 Ack=1120 Win=32128…
30	8.292750140	192.168.100.2	192.168.101.2	SSHv2	1500 Client: Key Exchange Init
31	8.292754562	192.168.100.2	192.168.101.2	TCP	1500 [TCP Retransmission] 39374 → 22 [PSH, ACK] Seq=41 Ack=1120 Wi…
32	8.336928217	192.168.101.2	192.168.100.2	TCP	68 22 → 39374 [ACK] Seq=1120 Ack=1473 Win=31872 Len=0 TSval=1467…
33	8.336943013	192.168.101.2	192.168.100.2	TCP	68 [TCP Dup ACK 32#1] 22 → 39374 [ACK] Seq=1120 Ack=1473 Win=318…

We see packets flowing to both ends and payload being carried through what confirms that port 22 is still open for the connections coming from the client-net. The capture of the same connection done on the **server** can be seen below verifying the fact of SSH still working with firewall rules being deployed:

1	0.000000000	192.168.100.2	192.168.101.2	TCP	74 39374 → 22 [SYN] Seq=0 Win=2
2	0.000049062	192.168.101.2	192.168.100.2	TCP	74 22 → 39374 [SYN, ACK] Seq=0 A
3	0.000504921	192.168.100.2	192.168.101.2	TCP	66 39374 → 22 [ACK] Seq=1 Ack=1 Win=
4	0.000796634	192.168.100.2	192.168.101.2	SSHv2	106 Client: Protocol (SSH-2.0-OpenSSH
5	0.000810178	192.168.101.2	192.168.100.2	TCP	66 22 → 39374 [ACK] Seq=1 Ack=41 Win
6	0.008192912	192.168.101.2	192.168.100.2	SSHv2	105 Server: Protocol (SSH-2.0-OpenSSH
7	0.008662131	192.168.100.2	192.168.101.2	TCP	66 39374 → 22 [ACK] Seq=41 Ack=40 Wi
8	0.008831916	192.168.101.2	192.168.100.2	SSHv2	1146 Server: Key Exchange Init
9	0.009159556	192.168.100.2	192.168.101.2	TCP	66 39374 → 22 [ACK] Seq=41 Ack=1120
10	0.009762663	192.168.100.2	192.168.101.2	SSHv2	1498 Client: Key Exchange Init
11	0.053689946	192.168.101.2	192.168.100.2	TCP	66 22 → 39374 [ACK] Seq=1120 Ack=147
12	0.054227793	192.168.100.2	192.168.101.2	SSHv2	114 Client: Diffie-Hellman Key Exchan
13	0.054243437	192.168.101.2	192.168.100.2	TCP	66 22 → 39374 [ACK] Seq=1120 Ack=152
14	0.059481034	192.168.101.2	192.168.100.2	SSHv2	486 Server: Diffie-Hellman Key Exchan
15	0.065756910	192.168.100.2	192.168.101.2	SSHv2	82 Client: New Keys
16	0.109916722	192.168.101.2	192.168.100.2	TCP	66 22 → 39374 [ACK] Seq=1540 Ack=153
17	0.110477970	192.168.100.2	192.168.101.2	SSHv2	110 Client: Encrypted packet (len=44)
18	0.110497576	192.168.101.2	192.168.100.2	TCP	66 22 → 39374 [ACK] Seq=1540 Ack=158
19	0.110592838	192.168.101.2	192.168.100.2	SSHv2	110 Server: Encrypted packet (len=44)
20	0.110974757	192.168.100.2	192.168.101.2	SSHv2	126 Client: Encrypted packet (len=60)
21	0.112746091	192.168.101.2	192.168.100.2	SSHv2	118 Server: Encrypted packet (len=52)
22	0.155688515	192.168.100.2	192.168.101.2	TCP	66 39374 → 22 [ACK] Seq=1641 Ack=163
23	2.509278705	192.168.100.2	192.168.101.2	SSHv2	214 Client: Encrypted packet (len=148
24	2.527484504	192.168.101.2	192.168.100.2	SSHv2	94 Server: Encrypted packet (len=28)
25	2.527963899	192.168.100.2	192.168.101.2	TCP	66 39374 → 22 [ACK] Seq=1789 Ack=166
26	2.528087251	192.168.100.2	192.168.101.2	SSHv2	178 Client: Encrypted packet (len=112

5. Wireshark capture of the telnet connection to the random port (not port 22) on the **router** confirms that packets are not going through the firewall on the router and that **router** block what it should block:



30	18.399406989	192.168.100.2	192.168.101.2	TCP	76 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 T…
31	19.418109398	192.168.100.2	192.168.101.2	TCP	76 [TCP Retransmission] 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 M…
36	21.434046680	192.168.100.2	192.168.101.2	TCP	76 [TCP Retransmission] 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 M…
51	25.530160030	192.168.100.2	192.168.101.2	TCP	76 [TCP Retransmission] 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 M…
59	33.721949155	192.168.100.2	192.168.101.2	TCP	76 [TCP Retransmission] 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 M…
71	49.850248093	192.168.100.2	192.168.101.2	TCP	76 [TCP Retransmission] 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 M…
99	82.618142150	192.168.100.2	192.168.101.2	TCP	76 [TCP Retransmission] 58866 → 1596 [SYN] Seq=0 Win=29200 Len=0 M…

We see the TCP SYN packets being sent from the client to the server, however, because the packets are being dropped by the firewall on the **router**, the server does not receive anything and that was confirmed by the capture on the server which did not capture any traffic. This confirms that router firewall blocked that traffic.

The call from the client to random port 1596 can be seen below which verifies that the connection times out:

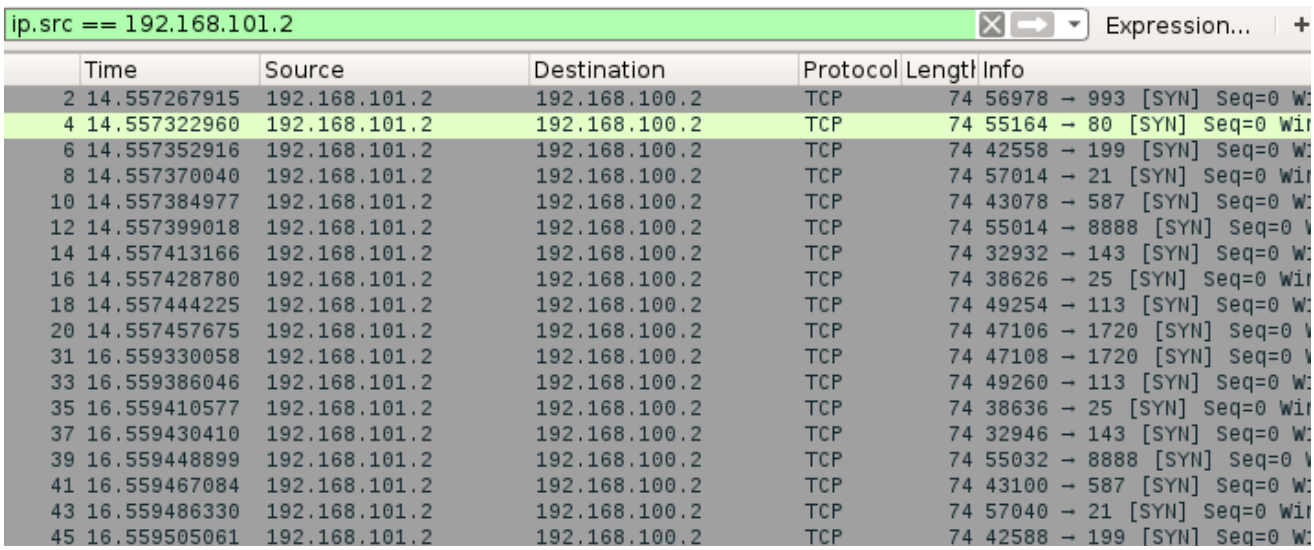
```
user@debian:~$ telnet server 1596
Trying 192.168.101.2...
telnet: Unable to connect to remote host: Connection timed out
user@debian:~$
```

6. I ran same netcat script as in the part3 to confirm that all the connections time out except for one on port 22 :

```
user@debian:~$ ./test_part3.sh
port 22 is open
user@debian:~$
```

7. Confirming that router allows all the packets sent by the server to go through to the client-net.

   I ran wireshark on the **client** to confirm that all the traffic from the server reaches the the client and firewall on the router is not dropping any of those packets. I ran `nmap client` and client captured all the packets sent from the server:



ip.src == 192.168.101.2 — Expression… +

	Time	Source	Destination	Protocol	Length	Info
2	14.557267915	192.168.101.2	192.168.100.2	TCP	74	56978 → 993 [SYN] Seq=0 W…
4	14.557322960	192.168.101.2	192.168.100.2	TCP	74	55164 → 80 [SYN] Seq=0 Win…
6	14.557352916	192.168.101.2	192.168.100.2	TCP	74	42558 → 199 [SYN] Seq=0 W…
8	14.557370040	192.168.101.2	192.168.100.2	TCP	74	57014 → 21 [SYN] Seq=0 Win…
10	14.557384977	192.168.101.2	192.168.100.2	TCP	74	43078 → 587 [SYN] Seq=0 W…
12	14.557399018	192.168.101.2	192.168.100.2	TCP	74	55014 → 8888 [SYN] Seq=0 …
14	14.557413166	192.168.101.2	192.168.100.2	TCP	74	32932 → 143 [SYN] Seq=0 W…
16	14.557428780	192.168.101.2	192.168.100.2	TCP	74	38626 → 25 [SYN] Seq=0 Win…
18	14.557444225	192.168.101.2	192.168.100.2	TCP	74	49254 → 113 [SYN] Seq=0 W…
20	14.557457675	192.168.101.2	192.168.100.2	TCP	74	47106 → 1720 [SYN] Seq=0 …
31	16.559330058	192.168.101.2	192.168.100.2	TCP	74	47108 → 1720 [SYN] Seq=0 …
33	16.559386046	192.168.101.2	192.168.100.2	TCP	74	49260 → 113 [SYN] Seq=0 W…
35	16.559410577	192.168.101.2	192.168.100.2	TCP	74	38636 → 25 [SYN] Seq=0 Win…
37	16.559430410	192.168.101.2	192.168.100.2	TCP	74	32946 → 143 [SYN] Seq=0 W…
39	16.559448899	192.168.101.2	192.168.100.2	TCP	74	55032 → 8888 [SYN] Seq=0 …
41	16.559467084	192.168.101.2	192.168.100.2	TCP	74	43100 → 587 [SYN] Seq=0 W…
43	16.559486330	192.168.101.2	192.168.100.2	TCP	74	57040 → 21 [SYN] Seq=0 Win…
45	16.559505061	192.168.101.2	192.168.100.2	TCP	74	42588 → 199 [SYN] Seq=0 W…

This confirms that firewall was passing what it should pass and did not block any outgoing traffic from the server.

8. To sum up, we got the following behavior: all the connections to the server are dropped and none of the feedback is returned to the caller except for the connections to the port 22 which allows to establish the SSH connection. This is the same behavior as with the part 3, however, now all the connections are being dropped by the router rather than the server. The server does not receive any

traffic that it is suppose to reach which is SSH on port 22. Also, we verified that traffic from the server is able to leave the server-net freely without any constraints.

Part 4 (PART2 testing)

Blocking the access to the port 80 from the **client-net** to the **server-net** in the router so that fetching index page times out and the packets to port 80 are dropped on the router firewall and does not pass through the router to the server-net. In the end I added `-d 192.168.101.2` to the rule (`/sbin/iptables -A FORWARD -i "$client_net" -o "$server_net" -d 192.168.101.2 -p tcp --dport 80 -j DROP`) dropping the packets to port 80 only to the server and not the entire server-net. I consider this as a sensible solution as in the real world we might have multiple servers on the server-net which would have their port 80 exposed to the outside world and would not like traffic to them being filtered. If it is not the case only the part `-d 192.168.101.2` should be removed which filters port 80 for the entire server-net.

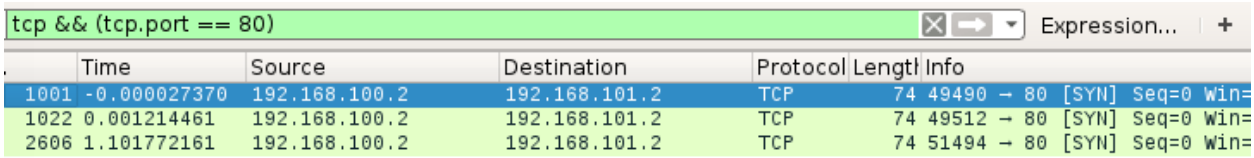
In this part only captures after deployment of **sudo ./part4.sh PART2** are going to be explained as the behavior before the capture was described in the section above (Part 4 (PART3 testing)).

After deploying the firewall rules from the script **part4.sh PART2** all the traffic to the port 80 to the server was blocked. This was verified by the nmap scan and Wireshark capture on the server and the router.

1. Nmap scan after the deployment **part4.sh PART2** firewall rules on the router (scan done from the client):

```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 09:39 UTC
Nmap scan report for server (192.168.101.2)
Host is up (0.00028s latency).
Not shown: 998 closed ports
PORT   STATE    SERVICE
22/tcp open     ssh
80/tcp filtered http

Nmap done: 1 IP address (1 host up) scanned in 1.24 seconds
```

   Which is the same behavior as in the Part 2 which was expected. We see that the connection to the port 80 is **filtered** meaning that no rules came back to the client. This should be further verified by running a wireshark on the router to see if the traffic from port 80 from the server is sent back to the client or is it dropped on the router an no results are returned. The results of Wireshark capture on the router can be seen below:



   The filter was applied to the Wireshark capture because responses for each nmap ping were sent from the server to the client and it was hard to verify if the port 80 is being blocked (it verified the fact that packets to other ports are freely passed through by the router). After filtering `tcp.port == 80` we see that packets flow only from the client (source `192.168.100.2`) to the server (dest `192.168.101.2`) and no response is sent back from the server. This was further verified by running the capture on the server and applying the same `tcp.port == 80` filter, however, no packets were seen in the capture which proves that the firewall on the router was blocking access to the port 80 (and allowing all the others).

Part 5

Blocking all the packets sent through the router from the interface connected to the **server-net** other than the server-net itself (192.168.101.0/24) and blocking all the packets sent through the router from the interface connected to the **client-net** other than the client-net itself (192.168.100.0/24). This includes blocking all the private networks specified in RFC1918 (with exception described before) :

```
10.0.0.0        -   10.255.255.255  (10/8 prefix)
172.16.0.0      -   172.31.255.255  (172.16/12 prefix)
192.168.0.0     -   192.168.255.255 (192.168/16 prefix)
```

The behavior of the nmap scan and Wireshark capture before the firewall rules were deployed was described in the Part 4. Therefore, only the testing after will be described in this part.

IP Spoofing

In this part for testing it was required to fake IPs of the two nodes in the network (client and server). The tool used for faking IP addresses is described in detailed here: https://sandilands.info/sgordon/address-spoofing-with-iptables-in-linux In a nutshell we can use the following command to fake the outgoing IP address:

```
$ sudo iptables -t nat -A POSTROUTING -j SNAT --to-source <fake_ip_number>
```

This was necessary in order to confirm the following functionality of the firewall on the router:

5.a) Firewall was blocking all the traffic from the interface connected to the client-net which is not address contained within 192.168.100.0/24.

5.b) Firewall was blocking all the traffic from the interface connected to the server-net which is not address contained within 192.168.101.0/24.

5.c) Firewall was allowing the traffic from the interface connected to the client-net which IP address was within range 192.168.100.0/24.

5.d) Firewall was allowing the traffic from the interface connected to the server-net which IP address was within range 192.168.101.0/24.

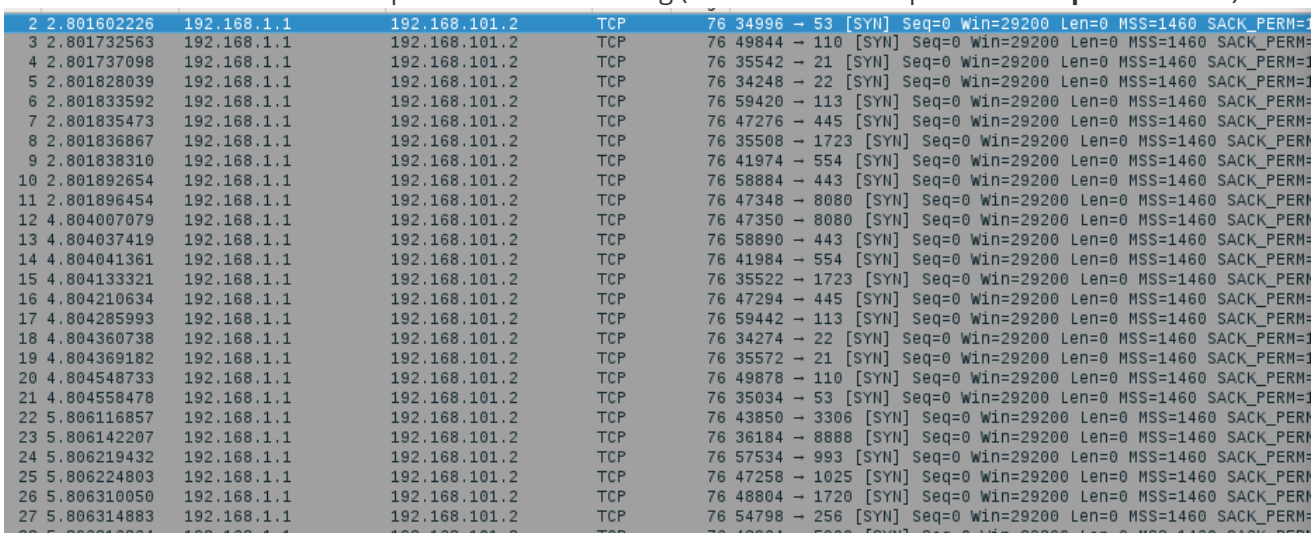
Part 5a

To confirm that firewall was blocking all the traffic from the interface connected to the client-net other than the 192.168.100.0/24. The client address was faked to be **192.168.1.1** which is out of range of possible addresses. When running a simple `nmap server` or `nmap -Pn server` on the client no results were returned:

```
user@debian:~$ nmap server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 16:07 UTC
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.04 seconds
user@debian:~$
```

```
user@debian:~$ nmap -Pn server

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 15:56 UTC
Nmap scan report for server (192.168.101.2)
Host is up.
All 1000 scanned ports on server (192.168.101.2) are filtered

Nmap done: 1 IP address (1 host up) scanned in 201.37 seconds
```

This might be due to 2 reasons: firewall dropped the packets or because we used fake IP (192.168.1.1) it did not reach the client back. So further explorations were required during which the Wireshark was run on the router and server. The router capture was the following (note that it was capture of **nmap -Pn** version):



2	2.801602226	192.168.1.1	192.168.101.2	TCP	76 34996 → 53 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
3	2.801732563	192.168.1.1	192.168.101.2	TCP	76 49844 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
4	2.801737098	192.168.1.1	192.168.101.2	TCP	76 35542 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
5	2.801828039	192.168.1.1	192.168.101.2	TCP	76 34248 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
6	2.801833592	192.168.1.1	192.168.101.2	TCP	76 59420 → 113 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
7	2.801835473	192.168.1.1	192.168.101.2	TCP	76 47276 → 445 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
8	2.801836867	192.168.1.1	192.168.101.2	TCP	76 35508 → 1723 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
9	2.801838310	192.168.1.1	192.168.101.2	TCP	76 41974 → 554 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
10	2.801892654	192.168.1.1	192.168.101.2	TCP	76 58884 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
11	2.801896454	192.168.1.1	192.168.101.2	TCP	76 47348 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
12	4.804007079	192.168.1.1	192.168.101.2	TCP	76 47350 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
13	4.804037419	192.168.1.1	192.168.101.2	TCP	76 58890 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
14	4.804041361	192.168.1.1	192.168.101.2	TCP	76 41984 → 554 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
15	4.804133321	192.168.1.1	192.168.101.2	TCP	76 35522 → 1723 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
16	4.804210634	192.168.1.1	192.168.101.2	TCP	76 47294 → 445 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
17	4.804285993	192.168.1.1	192.168.101.2	TCP	76 59442 → 113 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
18	4.804360738	192.168.1.1	192.168.101.2	TCP	76 34274 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
19	4.804369182	192.168.1.1	192.168.101.2	TCP	76 35572 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
20	4.804548733	192.168.1.1	192.168.101.2	TCP	76 49878 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
21	4.804558478	192.168.1.1	192.168.101.2	TCP	76 35034 → 53 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
22	5.806116857	192.168.1.1	192.168.101.2	TCP	76 43850 → 3306 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
23	5.806142207	192.168.1.1	192.168.101.2	TCP	76 36184 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
24	5.806219432	192.168.1.1	192.168.101.2	TCP	76 57534 → 993 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
25	5.806224803	192.168.1.1	192.168.101.2	TCP	76 47258 → 1025 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
26	5.806310050	192.168.1.1	192.168.101.2	TCP	76 48804 → 1720 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
27	5.806314883	192.168.1.1	192.168.101.2	TCP	76 54798 → 256 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=

Which clearly demonstrates that packets were flowing to the one side only, and no results from the server were received. The Wireshark capture on the server did not receive any packets at all which confirmed that they first of all reached the router and then they were dropped by the router and were not passed any further which confirm the correct behavior of the router firewall.

Part 5b

Same was done to confirm that firewall was blocking all the traffic from the interface connected to the server-net other than the 192.168.101.0/24. The server address was faked to be **192.168.9.9** which is out of range of possible addresses. When running a simple `nmap client` or `nmap -Pn client` on the server no results were returned which is the same as with Part 5a:

```
user@debian:~$ sudo iptables -t nat -A POSTROUTING -j SNAT --to-source 192.168.9.9
user@debian:~$ nmap client

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 19:20 UTC
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.04 seconds
user@debian:~$ nmap -Pn client

Starting Nmap 7.40 ( https://nmap.org ) at 2018-02-04 19:21 UTC
Nmap scan report for client (192.168.100.2)
Host is up.
All 1000 scanned ports on client (192.168.100.2) are filtered

Nmap done: 1 IP address (1 host up) scanned in 201.37 seconds
```

The correct behavior was confirmed by the router and client captures which was the following (this time it is simple nmap version without any flags):

1	0.000000000	192.168.9.9	192.168.100.2	TCP	76 60150 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
2	0.000034910	192.168.9.9	192.168.100.2	TCP	76 35106 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
3	2.002119403	192.168.9.9	192.168.100.2	TCP	76 35108 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
4	2.002145852	192.168.9.9	192.168.100.2	TCP	76 60156 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460

Client capture was empty because no packets reached it as expected.

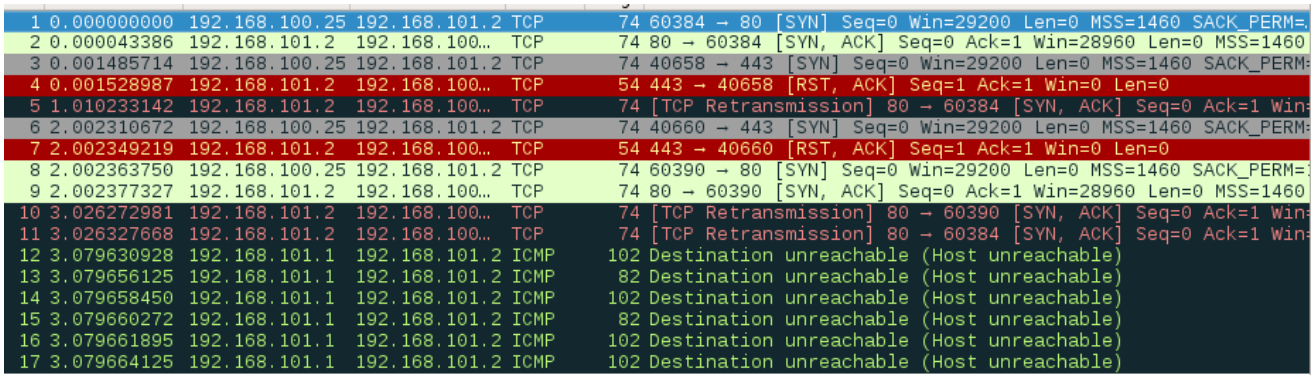
This is the same results as with the part 5a and verifies the fact that the router firewall is dropping the packets from incorrect IPs.

Part 5c

To confirm that firewall was allowing the traffic from the interface connected to the client-net which IP address was within range 192.168.100.0/24, the client address was faked to be **192.168.100.25** which is within the range of possible addresses. When running a simple `nmap server` or `nmap -Pn server` on the client no results were returned as before but this time it was due to the fact that no node on the client-net could be detected as having the IP address of **192.168.100.25**. The correct behavior of the router firewall was confirmed by the server and the router traffic captures with Wireshark:

12	19.095781919	192.168.100.25	192.168.101.2	TCP	76 60384 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
13	19.095815017	192.168.100.25	192.168.101.2	TCP	76 [TCP Out-Of-Order] 60384 → 80 [SYN] Seq=0 Win=29200 Len=0
14	19.096101633	192.168.101.2	192.168.100.25	TCP	76 80 → 60384 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=146
16	19.097319076	192.168.100.25	192.168.101.2	TCP	76 40658 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PER
17	19.097341732	192.168.100.25	192.168.101.2	TCP	76 [TCP Out-Of-Order] 40658 → 443 [SYN] Seq=0 Win=29200 Len=
18	19.097590427	192.168.101.2	192.168.100.25	TCP	62 443 → 40658 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
25	20.106339409	192.168.101.2	192.168.100.25	TCP	76 [TCP Retransmission] 80 → 60384 [SYN, ACK] Seq=0 Ack=1 Wi
30	21.098085959	192.168.100.25	192.168.101.2	TCP	76 40660 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PER
31	21.098116805	192.168.100.25	192.168.101.2	TCP	76 [TCP Out-Of-Order] 40660 → 443 [SYN] Seq=0 Win=29200 Len=
32	21.098179474	192.168.100.25	192.168.101.2	TCP	76 60390 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
33	21.098185039	192.168.100.25	192.168.101.2	TCP	76 [TCP Out-Of-Order] 60390 → 80 [SYN] Seq=0 Win=29200 Len=0
34	21.098415968	192.168.101.2	192.168.100.25	TCP	62 443 → 40660 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
35	21.098426607	192.168.101.2	192.168.100.25	TCP	76 80 → 60390 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=146
38	22.122371158	192.168.101.2	192.168.100.25	TCP	76 [TCP Retransmission] 80 → 60390 [SYN, ACK] Seq=0 Ack=1 Wi
39	22.122393613	192.168.101.2	192.168.100.25	TCP	76 [TCP Retransmission] 80 → 60384 [SYN, ACK] Seq=0 Ack=1 Wi
40	22.175462731	192.168.101.1	192.168.101.2	ICMP	104 Destination unreachable (Host unreachable)
41	22.175542293	192.168.101.1	192.168.101.2	ICMP	84 Destination unreachable (Host unreachable)
42	22.175551495	192.168.101.1	192.168.101.2	ICMP	104 Destination unreachable (Host unreachable)
43	22.175558297	192.168.101.1	192.168.101.2	ICMP	84 Destination unreachable (Host unreachable)
44	22.175565057	192.168.101.1	192.168.101.2	ICMP	104 Destination unreachable (Host unreachable)
45	22.175571190	192.168.101.1	192.168.101.2	ICMP	104 Destination unreachable (Host unreachable)

This Wireshark capture on the router clearly shows that the server responded to the requests sent from the client. This can be seen in packets 14, 18, 25 and so on where the source address is one of the server's (192.168.101.2) and the destination is the faked one sent from the client (192.169.100.25). Later we see the ICMP packets in green font (packets 40-45) which explains why the client did not receive any answer from the server as the server did not discover any node in the client-net with the faked ip. Similar capture was seen on the server:



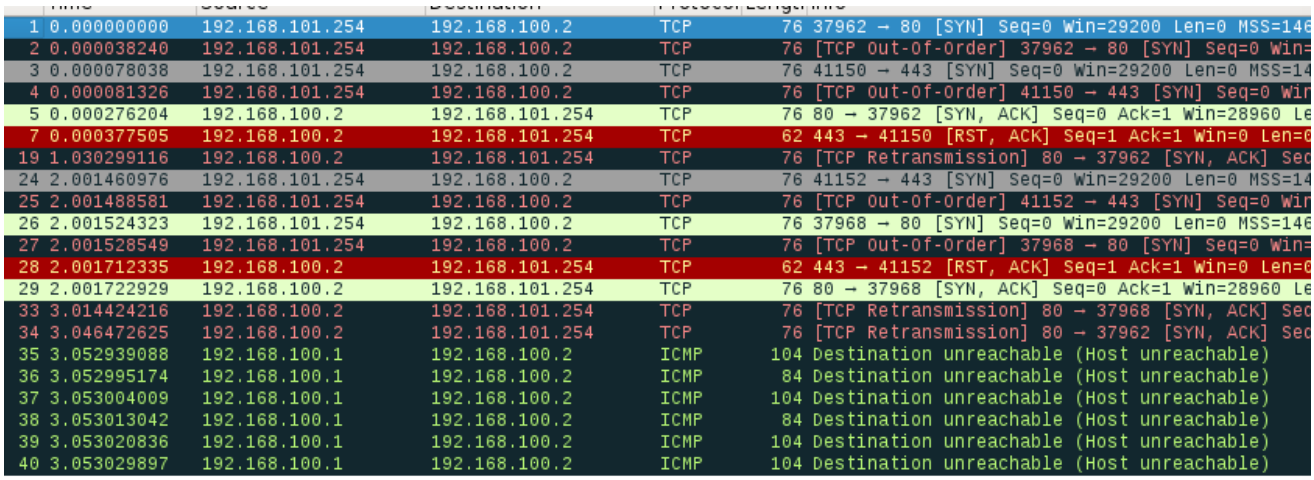
1	0.000000000	192.168.100.25	192.168.101.2	TCP	74 60384 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
2	0.000043386	192.168.101.2	192.168.100…	TCP	74 80 → 60384 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
3	0.001485714	192.168.100.25	192.168.101.2	TCP	74 40658 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
4	0.001528987	192.168.101.2	192.168.100…	TCP	54 443 → 40658 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.010233142	192.168.101.2	192.168.100…	TCP	74 [TCP Retransmission] 80 → 60384 [SYN, ACK] Seq=0 Ack=1 Win=
6	2.002310672	192.168.100.25	192.168.101.2	TCP	74 40660 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
7	2.002349219	192.168.101.2	192.168.100…	TCP	54 443 → 40660 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
8	2.002363750	192.168.100.25	192.168.101.2	TCP	74 60390 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
9	2.002377327	192.168.101.2	192.168.100…	TCP	74 80 → 60390 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
10	3.026272981	192.168.101.2	192.168.100…	TCP	74 [TCP Retransmission] 80 → 60390 [SYN, ACK] Seq=0 Ack=1 Win=
11	3.026327668	192.168.101.2	192.168.100…	TCP	74 [TCP Retransmission] 80 → 60384 [SYN, ACK] Seq=0 Ack=1 Win=
12	3.079630928	192.168.101.1	192.168.101.2	ICMP	102 Destination unreachable (Host unreachable)
13	3.079656125	192.168.101.1	192.168.101.2	ICMP	82 Destination unreachable (Host unreachable)
14	3.079658450	192.168.101.1	192.168.101.2	ICMP	102 Destination unreachable (Host unreachable)
15	3.079660272	192.168.101.1	192.168.101.2	ICMP	82 Destination unreachable (Host unreachable)
16	3.079661895	192.168.101.1	192.168.101.2	ICMP	102 Destination unreachable (Host unreachable)
17	3.079664125	192.168.101.1	192.168.101.2	ICMP	102 Destination unreachable (Host unreachable)

This shows that router firewall allowed the packets flow as expected and they reached the server confirming that the firewall was passing what it should pass.

Part 5d

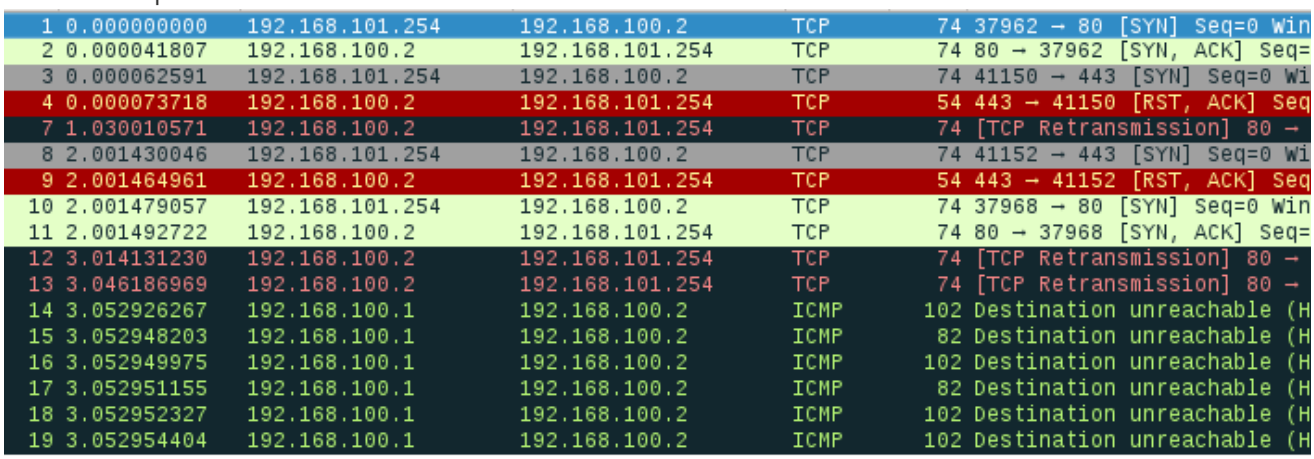
To confirm that firewall was allowing the traffic from the interface connected to the server-net which IP address was within range 192.168.101.0/24, the client address was faked to be **192.168.101.254** which is within the range of possible addresses. When running a simple `nmap client` or `nmap -Pn client` on the server no results were returned as the same as with Part 5c. Wireshark captures were very similar to the ones in Part 5c and further confirms the correct rules of the firewall:

The router capture:



1	0.000000000	192.168.101.254	192.168.100.2	TCP	76 37962 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=146
2	0.000038240	192.168.101.254	192.168.100.2	TCP	76 [TCP Out-Of-order] 37962 → 80 [SYN] Seq=0 Win=
3	0.000078038	192.168.101.254	192.168.100.2	TCP	76 41150 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=14
4	0.000081326	192.168.101.254	192.168.100.2	TCP	76 [TCP Out-Of-order] 41150 → 443 [SYN] Seq=0 Wir
5	0.000276204	192.168.100.2	192.168.101.254	TCP	76 80 → 37962 [SYN, ACK] Seq=0 Ack=1 Win=28960 Le
7	0.000377505	192.168.100.2	192.168.101.254	TCP	62 443 → 41150 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	1.030299116	192.168.100.2	192.168.101.254	TCP	76 [TCP Retransmission] 80 → 37962 [SYN, ACK] Seq
24	2.001460976	192.168.101.254	192.168.100.2	TCP	76 41152 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=14
25	2.001488581	192.168.101.254	192.168.100.2	TCP	76 [TCP Out-Of-order] 41152 → 443 [SYN] Seq=0 Wir
26	2.001524323	192.168.101.254	192.168.100.2	TCP	76 37968 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=146
27	2.001528549	192.168.101.254	192.168.100.2	TCP	76 [TCP Out-Of-order] 37968 → 80 [SYN] Seq=0 Win=
28	2.001712335	192.168.100.2	192.168.101.254	TCP	62 443 → 41152 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
29	2.001722929	192.168.100.2	192.168.101.254	TCP	76 80 → 37968 [SYN, ACK] Seq=0 Ack=1 Win=28960 Le
33	3.014424216	192.168.100.2	192.168.101.254	TCP	76 [TCP Retransmission] 80 → 37968 [SYN, ACK] Seq
34	3.046472625	192.168.100.2	192.168.101.254	TCP	76 [TCP Retransmission] 80 → 37962 [SYN, ACK] Seq
35	3.052939088	192.168.100.1	192.168.100.2	ICMP	104 Destination unreachable (Host unreachable)
36	3.052995174	192.168.100.1	192.168.100.2	ICMP	84 Destination unreachable (Host unreachable)
37	3.053004009	192.168.100.1	192.168.100.2	ICMP	104 Destination unreachable (Host unreachable)
38	3.053013042	192.168.100.1	192.168.100.2	ICMP	84 Destination unreachable (Host unreachable)
39	3.053020836	192.168.100.1	192.168.100.2	ICMP	104 Destination unreachable (Host unreachable)
40	3.053029897	192.168.100.1	192.168.100.2	ICMP	104 Destination unreachable (Host unreachable)

The client capture:



1	0.000000000	192.168.101.254	192.168.100.2	TCP	74 37962 → 80 [SYN] Seq=0 Win
2	0.000041807	192.168.100.2	192.168.101.254	TCP	74 80 → 37962 [SYN, ACK] Seq=
3	0.000062591	192.168.101.254	192.168.100.2	TCP	74 41150 → 443 [SYN] Seq=0 Wi
4	0.000073718	192.168.100.2	192.168.101.254	TCP	54 443 → 41150 [RST, ACK] Seq
7	1.030010571	192.168.100.2	192.168.101.254	TCP	74 [TCP Retransmission] 80 →
8	2.001430046	192.168.101.254	192.168.100.2	TCP	74 41152 → 443 [SYN] Seq=0 Wi
9	2.001464961	192.168.100.2	192.168.101.254	TCP	54 443 → 41152 [RST, ACK] Seq
10	2.001479057	192.168.101.254	192.168.100.2	TCP	74 37968 → 80 [SYN] Seq=0 Win
11	2.001492722	192.168.100.2	192.168.101.254	TCP	74 80 → 37968 [SYN, ACK] Seq=
12	3.014131230	192.168.100.2	192.168.101.254	TCP	74 [TCP Retransmission] 80 →
13	3.046186969	192.168.100.2	192.168.101.254	TCP	74 [TCP Retransmission] 80 →
14	3.052926267	192.168.100.1	192.168.100.2	ICMP	102 Destination unreachable (H
15	3.052948203	192.168.100.1	192.168.100.2	ICMP	82 Destination unreachable (H
16	3.052949975	192.168.100.1	192.168.100.2	ICMP	102 Destination unreachable (H
17	3.052951155	192.168.100.1	192.168.100.2	ICMP	82 Destination unreachable (H
18	3.052952327	192.168.100.1	192.168.100.2	ICMP	102 Destination unreachable (H
19	3.052954404	192.168.100.1	192.168.100.2	ICMP	102 Destination unreachable (H

Overview

To sum up, the testing of Part 5 verified the fact that firewall rules deployed on the router allows the flow of the packets from the client-net and server-net, but drops any other packets in order to secure against retransmission attacks.

Part 6

Because Part 6 was based on the Part 4 which contained two different version of the firewall, the same was done with the Part 6:

- Running the part 2 description by: `sudo ./part6.sh PART2` with **PART2** flag set.
- Running the part 3 description by: `sudo ./part6.sh PART3` with **PART3** flag set.

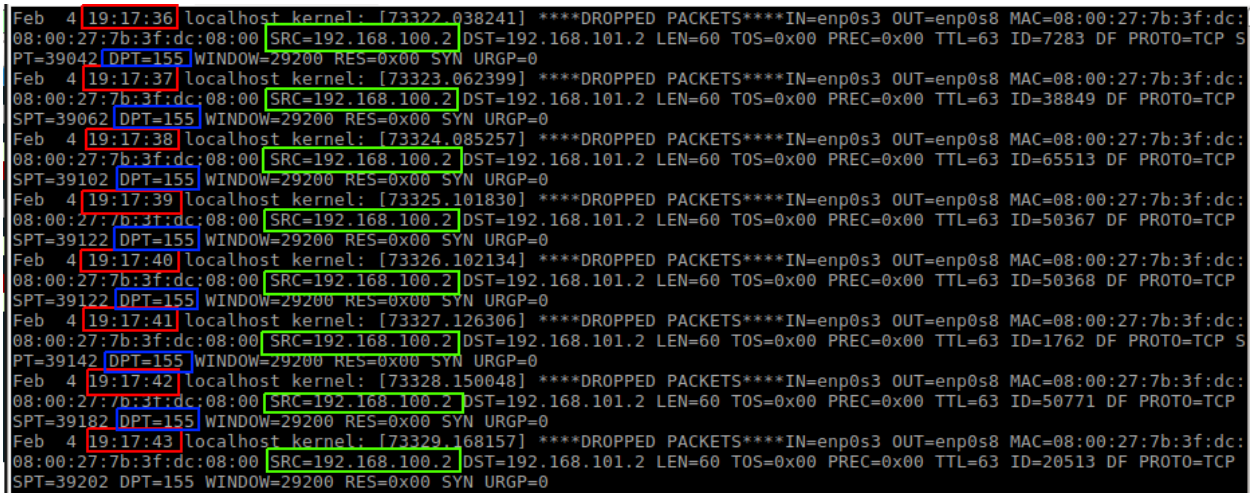
Part 6 (PART3)

Logging all the dropped packets omitting the SSH connections to the port 22. To confirm that logging is working correctly and only one logging line is generated per second per source IP/destination port pair two tests were done:

- For the first test I ran a burst of 400 connections from the client to the server and checked what is the rate at which lines are being generated. The test was the following:

```
#!/bin/bash

for i in {1..400} ; do
    echo "QUIT" | nc -w 5 server 155 &
    sleep 0.1
done
```

  This test allowed to verify at what rate lines are being generated. And the `/var/log/kern/log` confirmed that log is being generated at exactly 1 second intervals. This can be seen in the image below:

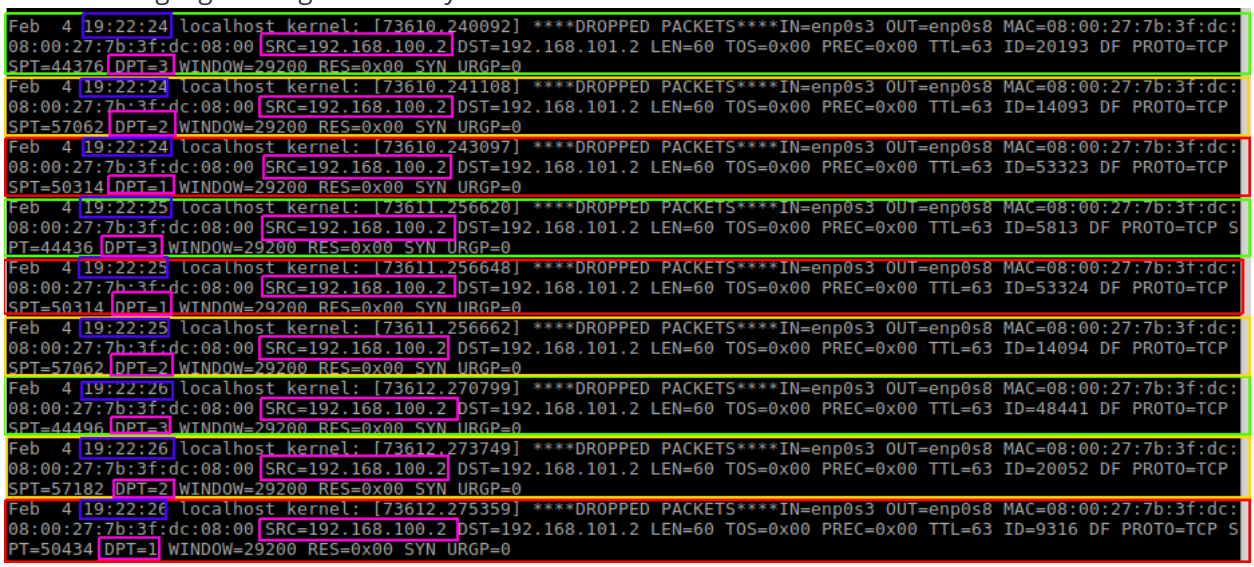


  In the red outline we can see the time when the log is generated at all the logs are exactly 1 second apart from each other. Green outline shows the source IP and blue outline shows destination port to confirm the fact that 1/sec logs are generated per source IP and destination port pair.

- For another test I ran a burst of 100 connections from the client to 3 different ports on the server in parallel to confirm that ports are generated per source IP and destination port pair. The test was the following:

```
#!/bin/bash

function test(){
    for i in {1..100} ; do
        echo "QUIT" | nc -w 5 "$1" &
        sleep 0.1
    done
}

for i in {1..3} ; do test "$1" & done
```

This following logs were generated by the kernel:



We see 3 different groups of packets:

- green outline -source IP-**192.168.100.2** and **destination port 3** (both in pink)
- yellow outline -source IP-**192.168.100.2** and **destination port 2** (both in pink)
- red outline -source IP-**192.168.100.2** and **destination port 1** (both in pink)

We see that first three logs were generated at exactly the same time `19:22:24` (seen in blue outline) and also we that first three logs are from different groups described above which means that logs are being generated based on the src ip and dest port. This is further repeated every second onward. Thus, the testing confirms that logging is generated 1/sec per src ip/dest port pair.

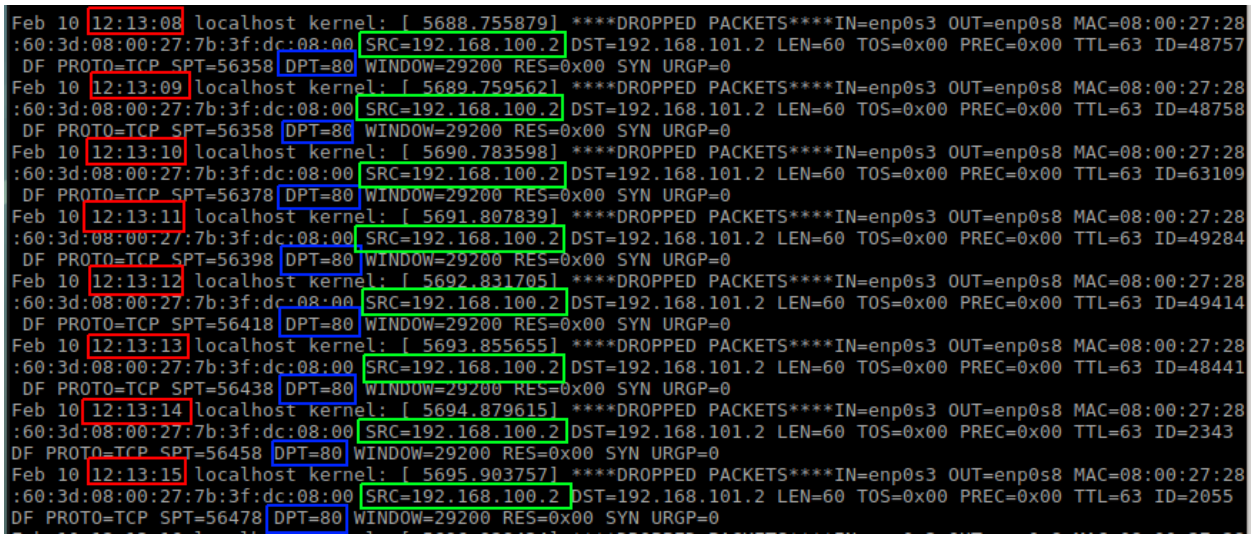
Part 6 (PART2)

The similar testing was done for this part. Logging all the dropped packets to the port 80 dedicated to the server. To confirm that logging is working correctly and only one logging line is generated per second per source IP/destination port pair only single testing was done because there is no use to test connections to different ports as only the ones to port 80 are logged.

- For the first test I ran a burst of 400 connections to the port 80 of the server from the client and checked what is the rate at which lines are being generated. The test was the following:

```
#!/bin/bash

for i in {1..400} ; do
    echo "QUIT" | nc -w 5 server 80 &
    sleep 0.1
done
```

This test allowed to verify at what rate lines are being generated. And the `/var/log/kern/log` confirmed that log is being generated at exactly 1 second intervals. This can be seen in the image below:



In the red outline we can see the time when the log is generated at all the logs are exactly 1 second apart from each other. Green outline shows the source IP and blue outline shows destination port which is 80 to confirm the fact that 1/sec logs are generated per source IP and destination port pair.

Thus, the testing confirms that logging is generated 1/sec per src ip/dest port pair.

Assignment Overvie w

Personally, I really enjoyed doing the assignment as it requires quite a lot of setup and research how to make the networks working correctly. Also, it gives a good sense of how firewalls actually work and that everyone can play with and configure their own routers, laptops rather than thinking of it as a magic which can only be touched by gods. The practicality and freedom of the assignments is really great and that's why I really enjoyed Networks module in the same way as Networks Security. :)