Student Name: Marco Ermini  
Email: mermini@asu.edu  
Submission Date: 29th May, 2021  
Class Name and Term: CSE548 Summer 2021

Project Report 1 - Packet Filter Firewall

# Project Overview

In this lab we are exploring how a packet filter firewall works by setting up an environment based on two Linux virtual machines – one working as a dual-homes Gateway client, which can access external networks on one interface – and another set up as a Client, which can only access external networks through the gateway. The gateway will be configured with the Linux iptables firewall and will also enable NAT for selected protocols.

The lab will also setup a web server on the Gateway with a test web page. We will test having full control of the network traffic, allowing only specific protocols for specific destinations at will, by modifying specific parameters in the firewall script.

All the files and configurations used for this lab have been uploaded on GitHub; references are provided throughout the text and in the Appendix A at the bottom of the File.

# Network Setup

Please find below the set-up of the virtual infrastructure as I have configured it in VirtualBox.

The peculiarity of my infrastructure is that I have not used DHCP and a default gateway for the internal network (10.0.2/24). This is because, even if I force the Client VM to use the Gateway VM as the default gateway, the Ubuntu machine will still detect the default gateway offered by VirtualBox on the same network (which is the DHCP server configured by VirtualBox), and the VM will configure itself to use that default gateway, bypassing the Gateway VM, and would end up using VirtualBox’ DHCP server on that subnet as a router to reach the internet.

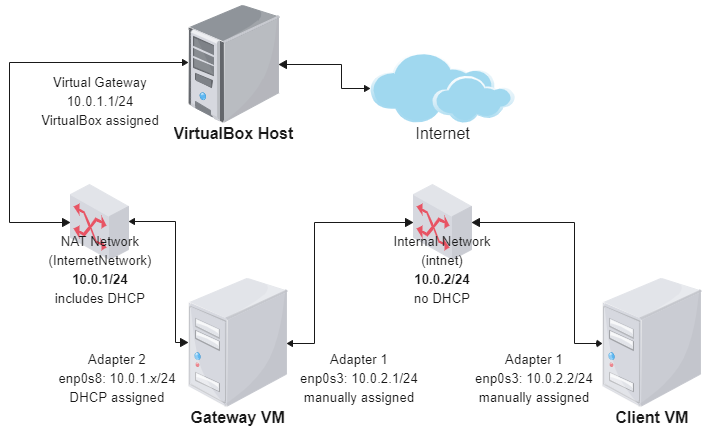


Figure 1 - Network Map of the Lab

Therefore, for the purpose of effectively executing the exercises in this lab and working with routing and iptables firewall rules, I have opted for a different set up, where the Gateway is forced in the path as the only possible gateway for the client. To achieve that, I have disabled VirtualBox’s DHCP in the internal network (10.0.2/24) and manually configured the IPs on the internal interfaces of the Gateway VM and the Client VM. I have also configured the Client VM to use the Gateway VM’s IP (10.0.2.1) as the default gateway.

By using this set up, the Client VM can only pass through the Gateway VM for external reachability. Note that because of the kind of configuration, there is no reason to mess up with any routing command on the client – **the correct routing is automatic**.

The overall initial configuration looks like the followings:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Gateway VM – Adapter 1** | **Gateway VM – Adapter 2** | **Client VM – Adapter 1** |
| **IP address** | 10.0.2.x/24 (DHCP assigned) | 10.0.1.1/24 | 10.0.2.2/24 |
| **Gateway** | On interface | Adapter 1 | 10.0.2.1 |
| **Initial Reachability** | Internet via VirtualBox host | Client VM IP | Gateway Adapter 2 IP |

The way in which Ubuntu’s 18.04 LTS versions configure network interfaces is through an application called ***NetPlan***. Its configuration scripts, which assume the form of YAML files, are located under /etc/netplan. By default, on that Ubuntu version, NetPlan is configured to use NetworkManager to handle the IP addresses. I have found it was easier to just override the default configuration (present on the file *01-network-manager-all.yaml*) and manually assign the IP address and configure the use of DHCP where required. The following configurations were used (**all the files are available on GitHub**, see Appendix A):

On the gateway:

root@ubuntu:/etc/netplan#**cat 02-gateway-networks.yaml**

*network:*

*renderer: networkd*

*version: 2*

*ethernets:*

*enp0s3:*

*addresses: [10.0.2.1/24]*

*#gateway4: 10.0.2.1*

*dhcp4: no*

*dhcp6: no*

*nameservers:*

*addresses: [208.67.222.222,208.67.220.222]*

*enp0s8:*

*addresses: []*

*#gateway4: 10.0.2.1*

*dhcp4: yes*

*dhcp6: no*

*nameservers:*

*addresses: [208.67.222.222,208.67.220.220]*

root@ubuntu:/etc/netplan#

On the client:

root@ubuntu:/etc/netplan#**cat 02-internal-network.yaml**

*network:*

*renderer: networkd*

*version: 2*

*ethernets:*

*enp0s3:*

*addresses: [10.0.2.2/24]*

*gateway4: 10.0.2.1*

*dhcp4: no*

*dhcp6: no*

*nameservers:*

*addresses: [208.67.222.222,208.67.220.220]*

root@ubuntu:/etc/netplan#

To enable the NetPlan configuration, it is necessary to issue the command “**sudo netplan apply**”.

On VirtualBox, the two internal networks were configured as it follows: via the Preferences menu, select “Network”.

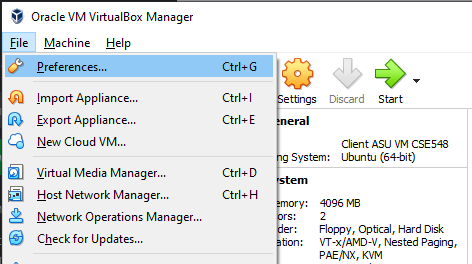
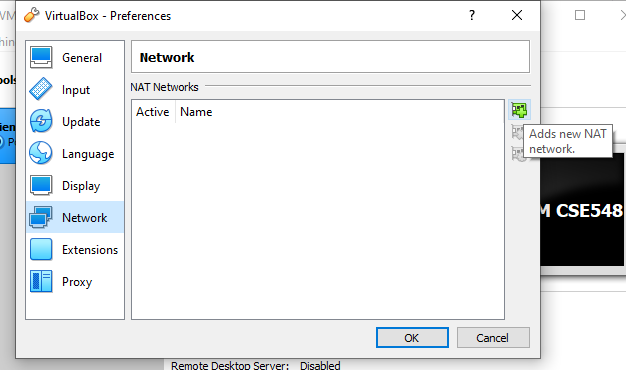


Figure 2 - VirtualBox Preferences

Figure 3 - VirtualBox Networks Configuration

Add a network with the “+” plus button, and configure the NatNetwork as following, then repeat for the InternetNetwork.

Please notice that DHCP is disabled for the NatNetwork (the internal).

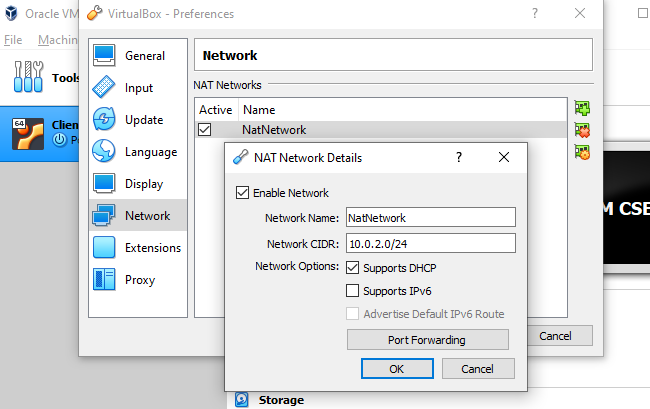
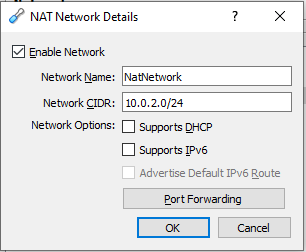


Figure 4 - InternetNetwork configuration

Figure 5 - NatNetwork Configuration

On the VM’s configuration, assign the interfaces on the correct networks. The followings are for the Gateway:

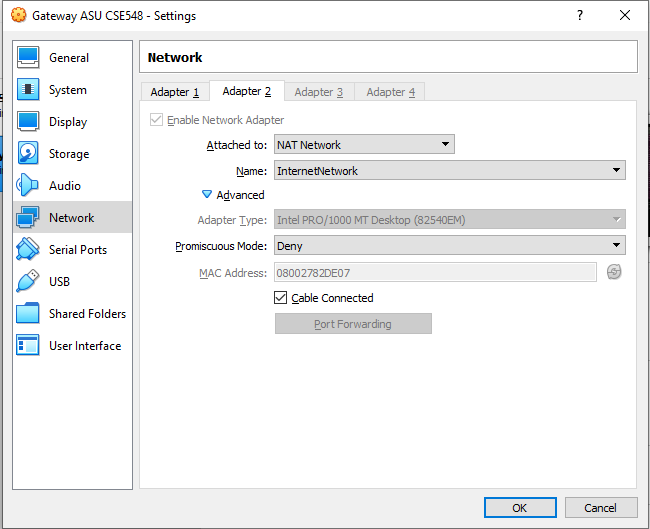
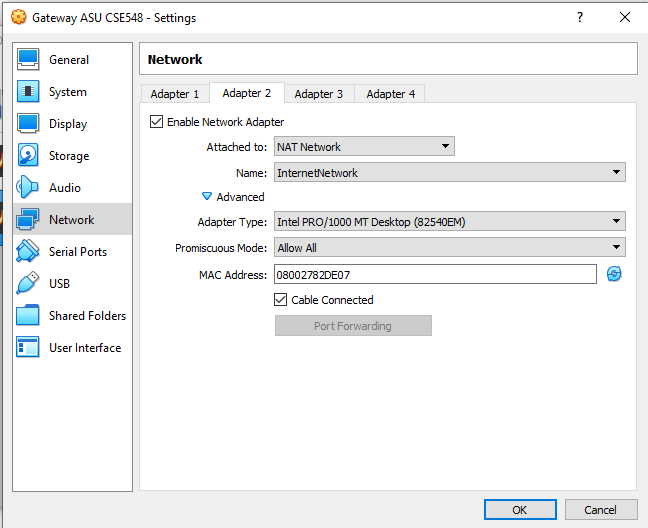


Figure 6 - Gateway’s Adapter 1 set up on VirtualBox

Figure 7 - Gateway’s Adapter 2 set up on VirtualBox

This is for the Client:

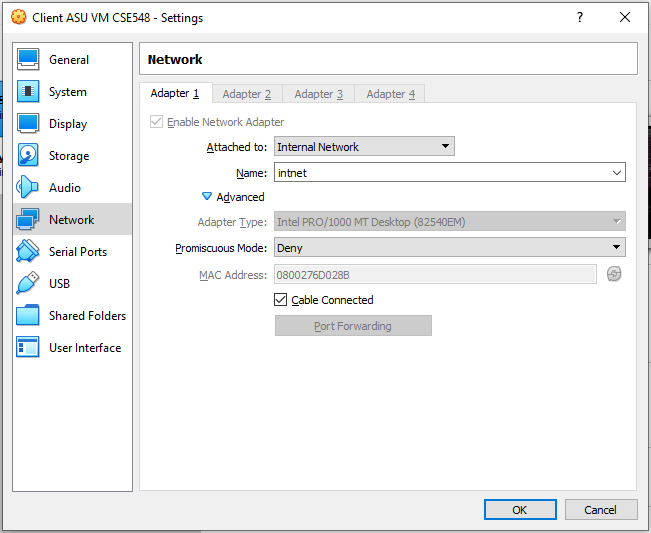


Figure 8 - Client's Adapter Configuration

On both systems, the command “**sudo service networking restart**” must be issues to ensure new network configurations are enabled.

At the end of the day, the following configurations should result when using Linux’s command line:

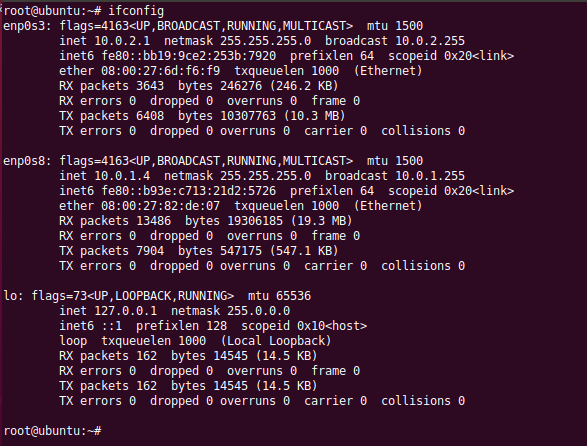
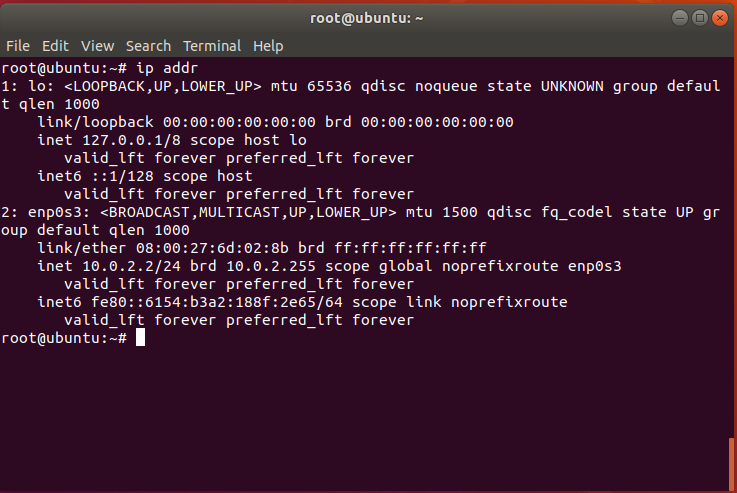


Figure 9 - Gateway VM's interfaces as they appear at the Linux console

Figure 10 - Client VM's interface as it appears at the Linux console

At the very beginning, there is nothing configured on the Gateway to forward traffic:

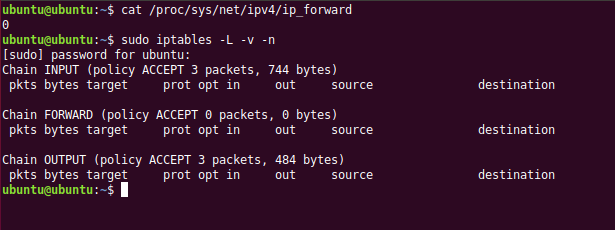


Figure 11 - Gateway VM's initial forward routing and iptable rules

Therefore, the Client VM is completely isolated in the InternalNetwork, and can only ping the Gateway’s VM interface.

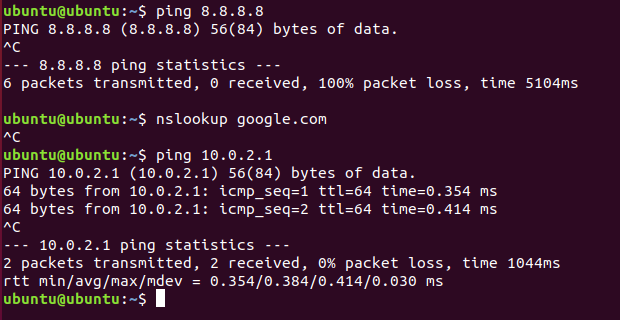


Figure 12 - Client's VM initial network situation

This is the client’s initial routing table:

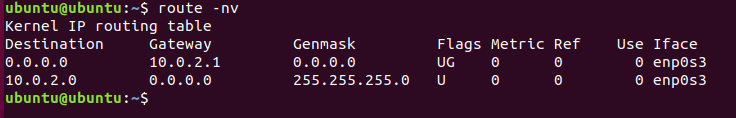


Figure 13 - Client's routing table

Conversely, the Gateway has no problems doing name resolution and reaching the internet by its default route:

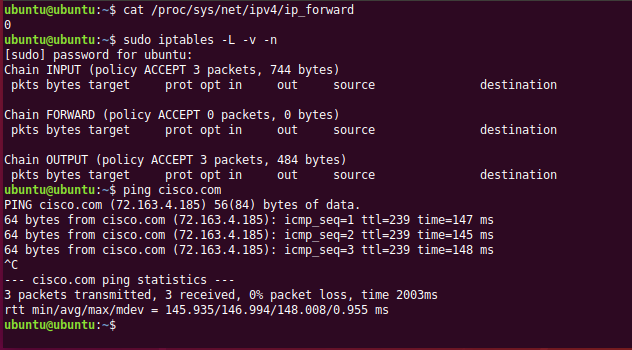


Figure 14 - Initial Gateway internet connection

This is gateways’ initial routing table.

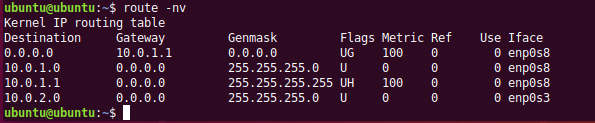


Figure 15 - Gateway's routing table

# Software

For this first lab, the following software has been used:

* Various network tools (specifically, tcpdump, ping, traceroute)
* Linux’s built in packet forwarding router
* Linux’s built-in packet inspection and NAT firewall, iptables
* Apache web server

# Project Description

The following tasks have been executed to obtain the required results for the lab, and they can be repeated step-by-step by pasting the commands listed below, provided that the initial configuration, as described in the “Network Setup” chapter, is reproduced exactly.

## Initial setup and connectivity testing

Please set up the VMs as described in the “Network Setup” page. The result must be so that the gateway can reach the internet, but the client cannot.

## Check the network setup on the Gateway/Server VM

From a terminal window on the gateway, please run the commands **in bold** to demonstrate internet reachability (please note that ping would run indefinitely, and you have to interrupt it with CTRL+C); the part *in italic* is the result that should be obtained:

ubuntu@ubuntu:~$ **ping cisco.com**

*PING cisco.com (72.163.4.185) 56(84) bytes of data.*

*64 bytes from cisco.com (72.163.4.185): icmp\_seq=1 ttl=239 time=147 ms*

*64 bytes from cisco.com (72.163.4.185): icmp\_seq=2 ttl=239 time=156 ms*

**^C**

*--- cisco.com ping statistics ---*

*3 packets transmitted, 3 received, 0% packet loss, time 2002ms*

*rtt min/avg/max/mdev = 144.485/149.644/156.627/5.141 ms*

ubuntu@ubuntu:~$ **dig google.com**

*; <<>> DiG 9.11.3-1ubuntu1.15-Ubuntu <<>> google.com*

*;; global options: +cmd*

*;; Got answer:*

*;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16376*

*;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1*

*;; OPT PSEUDOSECTION:*

*; EDNS: version: 0, flags:; udp: 65494*

*;; QUESTION SECTION:*

*;google.com. IN A*

*;; ANSWER SECTION:*

*google.com. 108 IN A 216.58.206.78*

*;; Query time: 0 msec*

*;; SERVER: 127.0.0.53#53(127.0.0.53)*

*;; WHEN: Sat May 29 15:24:17 MST 2021*

*;; MSG SIZE rcvd: 55*

ubuntu@ubuntu:~$

Please note that response times and latency will vary.

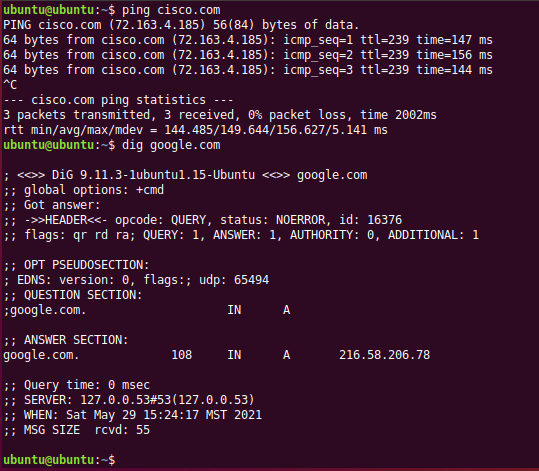


Figure 16 - Evidence of Gateway connectivity

After this test, verify that the Gateway has connectivity to the Client, issuing the command ping 10.0.2.2 (again, ping commands need to be manually interrupted with CTRL+C):

ubuntu@ubuntu:~$ **ping 10.0.2.2**

*PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.*

*64 bytes from 10.0.2.2: icmp\_seq=1 ttl=64 time=0.394 ms*

*64 bytes from 10.0.2.2: icmp\_seq=2 ttl=64 time=0.373 ms*

*64 bytes from 10.0.2.2: icmp\_seq=3 ttl=64 time=0.433 ms*

**^C**

*--- 10.0.2.2 ping statistics ---*

*3 packets transmitted, 3 received, 0% packet loss, time 2052ms*

*rtt min/avg/max/mdev = 0.373/0.400/0.433/0.024 ms*

ubuntu@ubuntu:~$

We have already verified on the Network Setup chapter, that there is no forwarding nor iptables rules enabled on the Gateway right now. No further demonstration is necessary, but at will, further commands can be run:

ubuntu@ubuntu:~$ **sudo iptables -L -v -n**

*[sudo] password for ubuntu:* **123456**

*Chain INPUT (policy ACCEPT 256 packets, 24309 bytes)*

*pkts bytes target prot opt in out source destination*

*Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)*

*pkts bytes target prot opt in out source destination*

*Chain OUTPUT (policy ACCEPT 435 packets, 30539 bytes)*

*pkts bytes target prot opt in out source destination*

ubuntu@ubuntu:~$ **route -nv**

*Kernel IP routing table*

*Destination Gateway Genmask Flags Metric Ref Use Iface*

*0.0.0.0 10.0.1.1 0.0.0.0 UG 100 0 0 enp0s8*

*10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 enp0s8*

*10.0.1.1 0.0.0.0 255.255.255.255 UH 100 0 0 enp0s8*

*10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 enp0s3*

ubuntu@ubuntu:~$ **ip addr**

*1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default 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*

*inet6 ::1/128 scope host*

*valid\_lft forever preferred\_lft forever*

*2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP group default qlen 1000*

*link/ether 08:00:27:6d:f6:f9 brd ff:ff:ff:ff:ff:ff*

*inet 10.0.2.1/24 brd 10.0.2.255 scope global enp0s3*

*valid\_lft forever preferred\_lft forever*

*inet6 fe80::bb19:9ce2:253b:7920/64 scope link noprefixroute*

*valid\_lft forever preferred\_lft forever*

*3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP group default qlen 1000*

*link/ether 08:00:27:82:de:07 brd ff:ff:ff:ff:ff:ff*

*inet 10.0.1.5/24 brd 10.0.1.255 scope global dynamic enp0s8*

*valid\_lft 514sec preferred\_lft 514sec*

*inet6 fe80::b93e:c713:21d2:5726/64 scope link noprefixroute*

*valid\_lft forever preferred\_lft forever*

ubuntu@ubuntu:~$

## Enable Packet forwarding on Gateway

To enable Linux kernel’s packet forwarding engine, it is necessary to flip a configuration parameter in a system table. This is done by one of the two commands:

$ **sudo sysctl -w net.ipv4.ip\_forward=1**

or

$ **echo 1 | sudo tee /proc/sys/net/ipv4/ip\_forward**

Using “cat” to print the value of the parameter will verify it is set correctly.

ubuntu@ubuntu:~$ **cat /proc/sys/net/ipv4/ip\_forward**

*1*

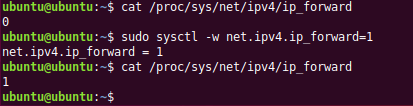


Figure 17 - Kernel routing on the Gateway

## Configure default routes

Because of our set up, there is no need to configure default routes – they are automatic. We may want to test that the Client is able to reach the Gateway.

On the Client, type the followings:

ubuntu@ubuntu:~$ **route -nv**

*Kernel IP routing table*

*Destination Gateway Genmask Flags Metric Ref Use Iface*

*0.0.0.0 10.0.2.1 0.0.0.0 UG 0 0 0 enp0s3*

*10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 enp0s3*

ubuntu@ubuntu:~$ **ping 10.0.2.1**

*PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.*

*64 bytes from 10.0.2.1: icmp\_seq=1 ttl=64 time=0.427 ms*

*64 bytes from 10.0.2.1: icmp\_seq=2 ttl=64 time=0.400 ms*

*64 bytes from 10.0.2.1: icmp\_seq=3 ttl=64 time=0.379 ms*

*^C*

*--- 10.0.2.1 ping statistics ---*

*3 packets transmitted, 3 received, 0% packet loss, time 2028ms*

*rtt min/avg/max/mdev = 0.379/0.402/0.427/0.019 ms*

ubuntu@ubuntu:~$

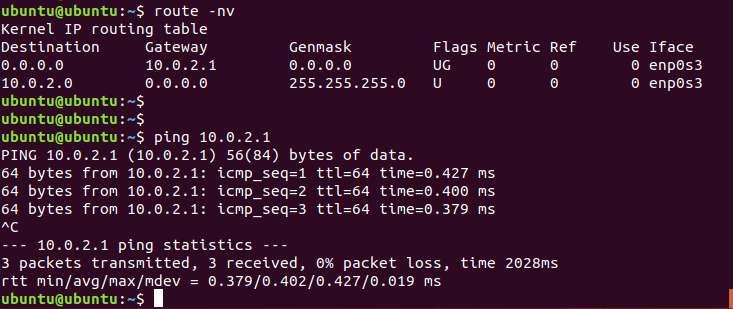


Figure 18 - Routing on the Client

However, trying to ping anything beyond the Gateway is hopeless at the moment, and it will fail.

ubuntu@ubuntu:~$ **ping 8.8.8.8**

*PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.*

(after circa four seconds…)

**^C**

*--- 8.8.8.8 ping statistics ---*

*5 packets transmitted, 0 received, 100% packet loss, time 4085ms*

ubuntu@ubuntu:~$

This is because the Gateway does not know yet how to process those packets – we will need NAT rules with iptables to accomplish that. Anyway, at the moment, routes are working correctly.

## Enable network traffic to Internet

To enable NAT rules on the Gateway and allow reachability of the Internet from the Client, type the followings:

ubuntu@ubuntu:~$ **sudo -i**

*[sudo] password for ubuntu:* **123456**

root@ubuntu:~# **iptables -P FORWARD ACCEPT**

root@ubuntu:~# **iptables -t nat -A POSTROUTING -o enp0s8 -j MASQUERADE**

root@ubuntu:~# **iptables -t nat -L POSTROUTING -n -v --line-number**

*Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)*

*num pkts bytes target prot opt in out source destination*

*1 0 0 MASQUERADE all -- \* enp0s8 0.0.0.0/0 0.0.0.0/0*

root@ubuntu:~#

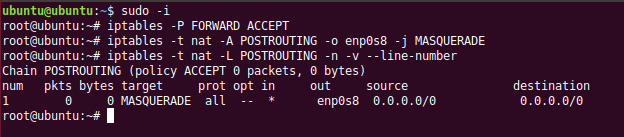


Figure 19 - Lazy NAT configuration on the Gateway

You should now be able to verify on the Client that you can reach certain Internet IP addresses (e.g. 8.8.8.8) and hosts which imply name resolution (e.g. google.com, cisco.com…) via simple “ping” commands.

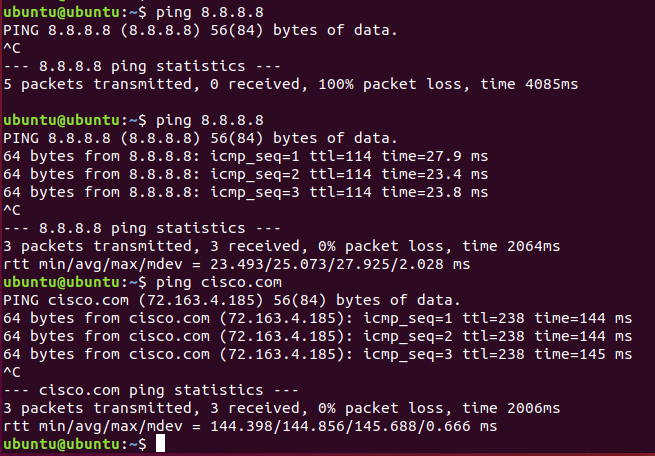


Figure 20 - Successfully pinging the Internet from the client

## Install Software in Linux

Now that the Client can reach the Internet, we can upgrade the software and install further packages. Please run “**sudo apt update**” at the terminal prompt, which will refresh the package list to the latest version:

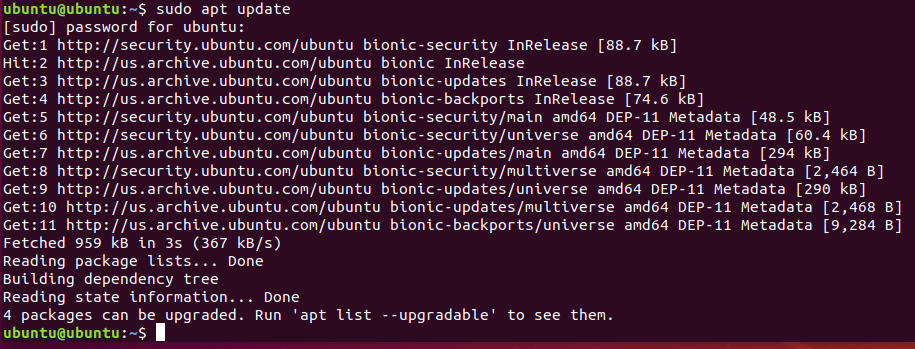


Figure 21 - apt update command

Please run now “**sudo apt -y dist-upgrade**” to update the already installed packages to the current version (output may vary):

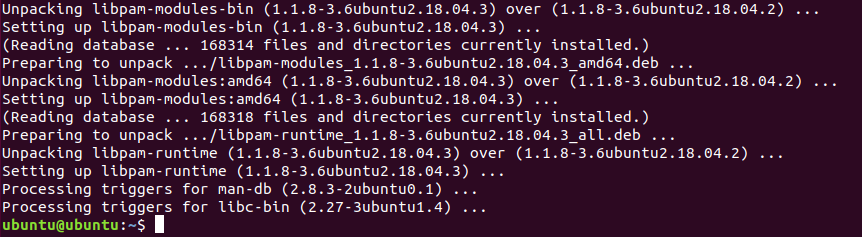


Figure 22 - apt dist-upgrade

Repeat the same operations on the Gateway, and in addition, install the “tcpdump” package (which will be needed soon) by typing “**sudo apt install tcpdump**”:

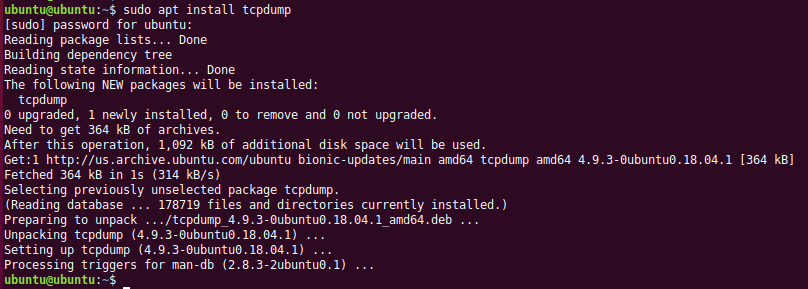


Figure 23 - Installation of tcpdump on the Gateway

## Install Apache2 on the Gateway

Please issue the command “**sudo apt install apache2**” on the Gateway to install apache2.

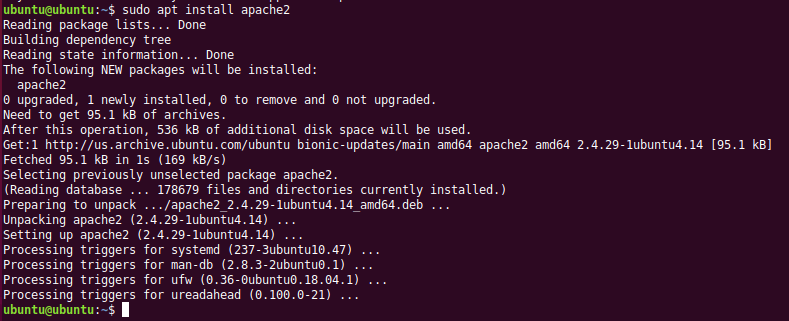


Figure 24 - Installation of Apache2 on the Gateway

We need now to be sure that the “ufw” application is stopped and disabled (will not reactivate on restart), by issuing the commands: “**sudo service ufw stop**” and “**sudo systemctl disable ufw**”.

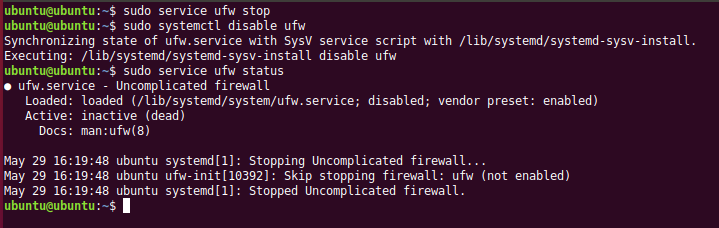


Figure 25 - Disabling of ufw

## Setup Apache2

The configuration file “/etc/apache2/sites-available/test-and-demo.conf” needs to be created and be equal to the listing below. It can be accomplished by using the “sudo” command with the favorite Linux editor, such as vim or nano.

If preferred, it can be downloaded from this link on GitHub: <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/test-and-demo.conf>

*<VirtualHost \*:80>*

*ServerName test-and-demo.com*

*ServerAlias \*.test-and-demo.com*

*ServerAdmin mermini@asu.edu*

*DocumentRoot /var/www/test/html*

*ErrorLog ${APACHE\_LOG\_DIR}/error.log*

*CustomLog ${APACHE\_LOG\_DIR}/access.log combined*

*</VirtualHost>*

The next configuration file to change is “/etc/apache2/ports.conf”, which needs to be set as the followings:

*Listen 127.0.0.1:80*

*Listen 10.0.2.1:80*

If preferred, the file ports.conf can be downloaded from here: <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/ports.conf>

While still on the Gateway, please add now the hostname “www.test-and-demo.com” on the /etc/hosts file by typing:

ubuntu@ubuntu:~$ **sudo -i**

*[sudo] password for ubuntu:* **123456**

root@ubuntu:~# **echo "127.0.0.1 www.test-and-demo.com" >> /etc/hosts**

root@ubuntu:~# **cat /etc/hosts**

*127.0.0.1 localhost*

*127.0.1.1 ubuntu*

*# The following lines are desirable for IPv6 capable hosts*

*::1 ip6-localhost ip6-loopback*

*fe00::0 ip6-localnet*

*ff00::0 ip6-mcastprefix*

*ff02::1 ip6-allnodes*

*ff02::2 ip6-allrouters*

*127.0.0.1 www.test-and-demo.com*

root@ubuntu:~#

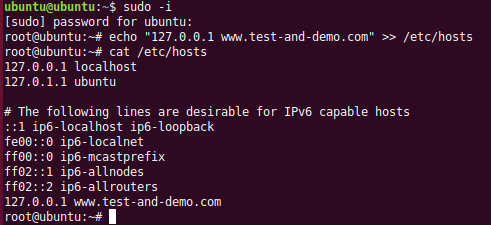


Figure 26 - Adding the host name to /etc/hosts

We have now to create the directory “/var/www/test/html” by issuing the following commands while still being the “root” user:

root@ubuntu:~# **mkdir /var/www/test**

root@ubuntu:~# **mkdir /var/www/test/html**

root@ubuntu:~# **cd /var/www/test/html**

root@ubuntu:~#

Create now a file called “index.html” to be positioned in the current folder and fill it with the content below.

**<html>**

**<head>**

**<title>ASU rocks!</title>**

**</head>**

**<body>**

**<p>This website is running on an Ubuntu Server from an ASU lab!</p>**

**<p>Contact the admin at <a href="mailto:mermini@asu.edu">mermini@asu.edu</a> for questions</p>**

**</body>**

**</html>**

If preferred, the file can be download from GitHub at this link: <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/index.html>

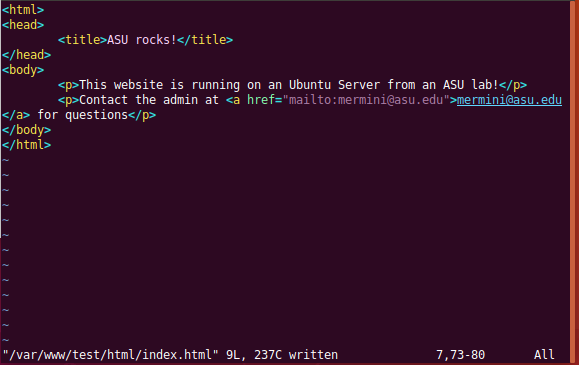


Figure 27 - the index.html of our test web site

After this is accomplished, please enable the new Apache2 VirtualHost and restart Apache2 with the following commands:

ubuntu@ubuntu:~$ **sudo a2ensite test-and-demo**

ubuntu@ubuntu:~$ **sudo systemctl restart apache2.service**

## Enable Reachability of Apache2 from the Gateway

At this point, on your Gateway you should be able to open a Firefox browser tab, point it to “www.test-and-demo.com” and see the default Apache2 Debian page appear.

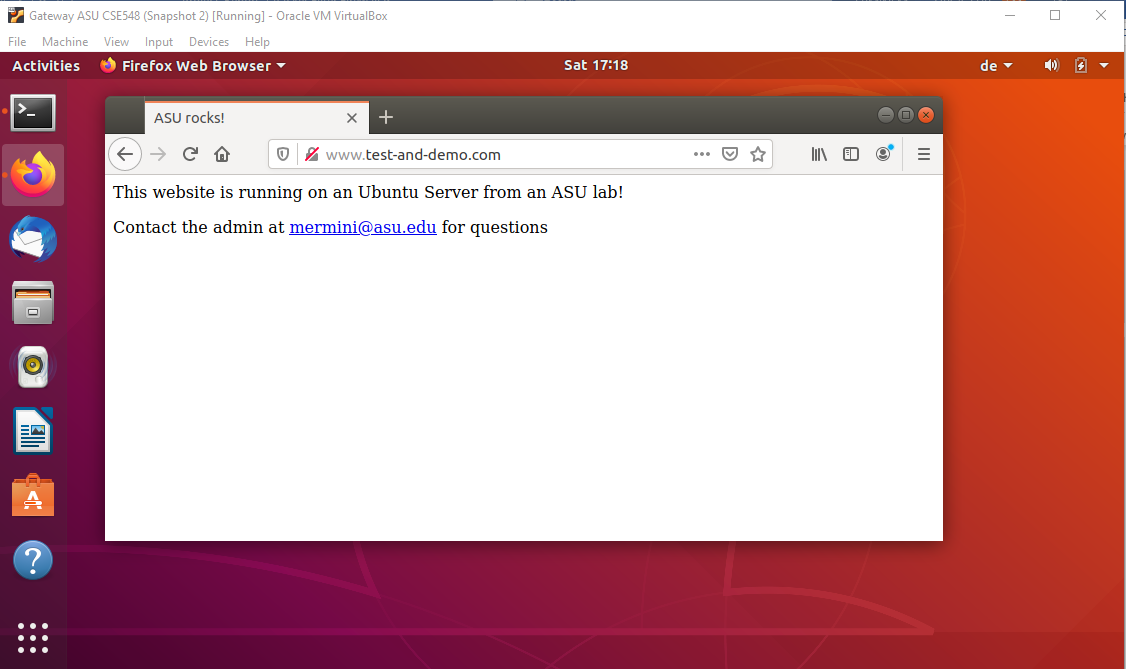


Figure 28 - Firefox browser testing out web site on the Gateway

## Enable Reachability of Apache2 from the Client

To enable reachability from the Client, we only need to perform similar commands affecting */etc/hosts*.

ubuntu@ubuntu:~$ **sudo -i**

*[sudo] password for ubuntu:* **123456**

root@ubuntu:~# **echo "10.0.2.1 www.test-and-demo.com" >> /etc/hosts**

root@ubuntu:~# **cat /etc/hosts**

*127.0.0.1 localhost*

*127.0.1.1 ubuntu*

*# The following lines are desirable for IPv6 capable hosts*

*::1 ip6-localhost ip6-loopback*

*fe00::0 ip6-localnet*

*ff00::0 ip6-mcastprefix*

*ff02::1 ip6-allnodes*

*ff02::2 ip6-allrouters*

*10.0.2.1 www.test-and-demo.com*

root@ubuntu:~#

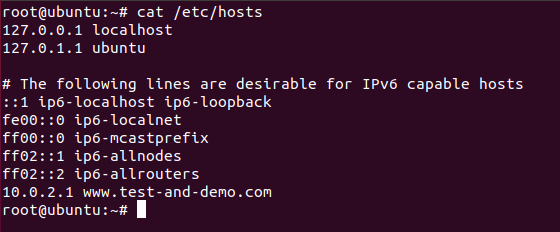


Figure 29 - /etc/hosts file on the Client

It will be then possible also from the Client to visualize the testing web page on the Gateway web server, by just opening Firefox and pointing it towards “www.test-and-demo.com”.

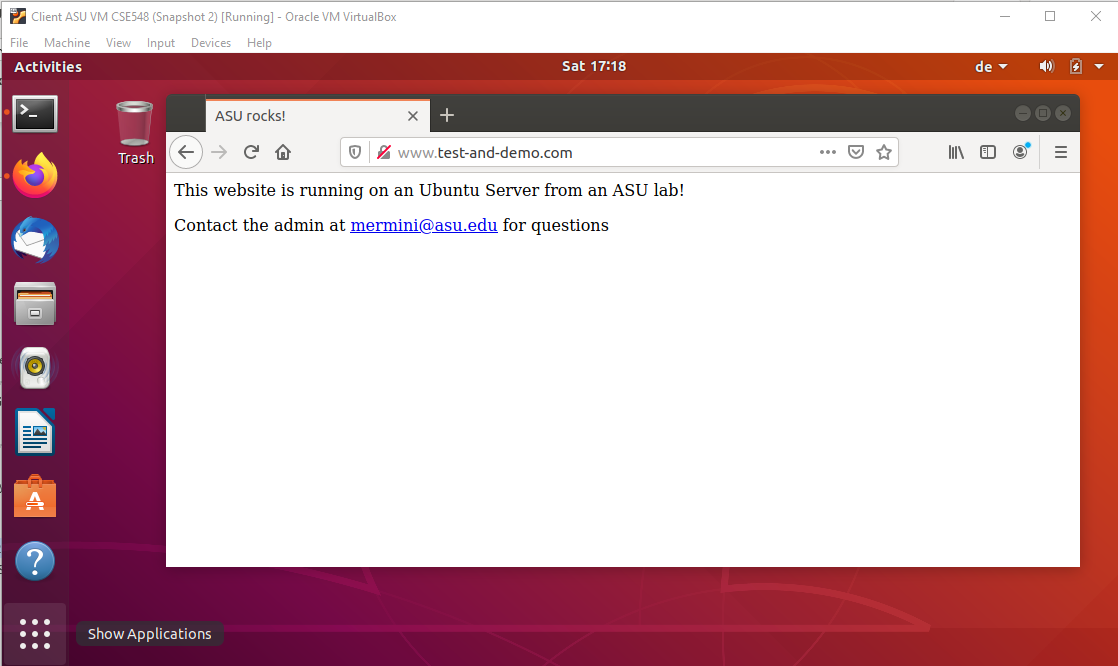


Figure 30 - Firefox browser testing out web site from the Client

## Set up of the Packet filter firewall

Setting up the stateless packet filter packet firewall requires running a script, which is provided here in the Appendix A or in GitHub at the address: <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/rc.sh>

We need to use the file rc.sh as the “root” user, therefore, if we are not it, please run “**sudo -i**”. After that, the file needs to be made executable with the command “**chmod +x rc.sh**”, or in alternative it can be run in this way: “**. ./rc.sh**”. In any case, it should run displaying verbose output of iptables rules being set, but with no errors.

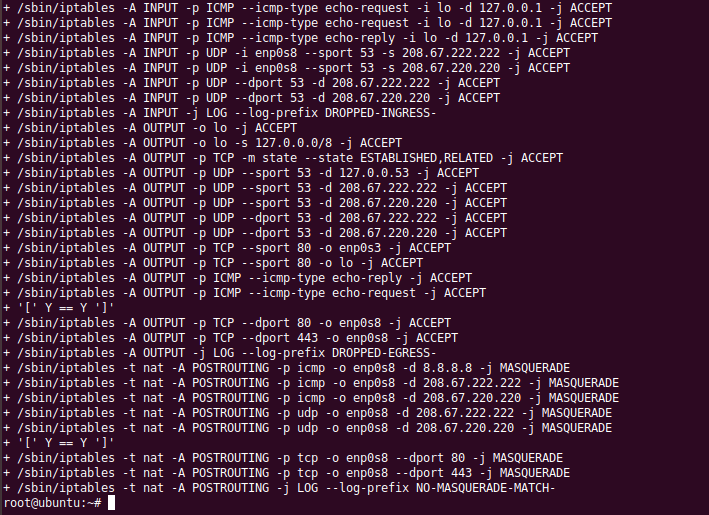


Figure 31 - One run of the rc.sh script

After the script is run, the Gateway will be configured to filter certain ports, blocking others, and propagating others via NAT.

This is the default configuration set up in the script:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Source** | **Destination** | **Protocol** | **Rule** | **Comment** |
| Client | Gateway | TCP/80 | Allowed | Reach the Web Server on the Gateway |
| Client | Internet | TCP/80 | Allowed | Reach Web for updates |
| Client | Internet | TCP/443 | Allowed | Reach Web for updates |
| Client | DNS servers | ICMP | Allowed | Ping DNS servers |
| Client | DNS servers | DNS | Allowed | Allow DNS resolution |
| Client | Gateway | ICMP | Allowed | Ping the Gateway |
| Client | 8.8.8.8 | ICMP | Allowed | Ping Google DNS |
| Client | \* | \* | Deny | Deny the rest |
| Gateway | Loopback | \* | Allowed | Allow loopback |
| Gateway | Client | ICMP | Allowed | Ping the Client |
| Gateway | Internet | TCP/80 | Allowed | Reach Web for updates |
| Gateway | Internet | TCP/443 | Allowed | Reach Web for updates |
| Gateway | DNS servers | ICMP | Allowed | Ping DNS servers |
| Gateway | DNS servers | DNS | Allowed | Allow DNS resolution |
| Gateway | 8.8.8.8 | ICMP | Allowed | Ping Google DNS |
| Gateway | \* | \* | Deny | Deny the rest |

The scripts also set up NAT for the allowed protocols for the Client.

As we can see from the screenshot, the client can ping the DNS servers, but although it can resolve the name cisco.com via DNS, it cannot ping it. It can anyway browse the web, while also being still able to reach the web server on the Gateway.

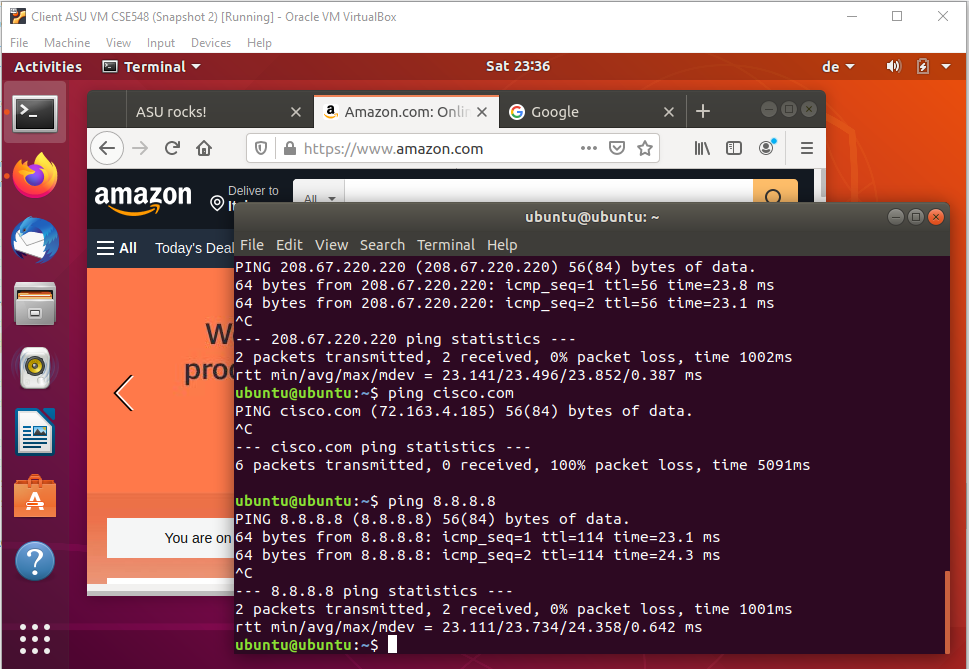


Figure 32 - Testing connectivity from the Client

## Sniffing traffic on the Gateway

One interesting aspect of this configuration is that the Gateway is in the position to “sniff” all the traffic of the Client.

Please run the command “sudo tcpdump -i enp0s3” on a Terminal on the Gateway, and then move to the Client and execute an allowed network command – for instance, ping 8.8.8.8.

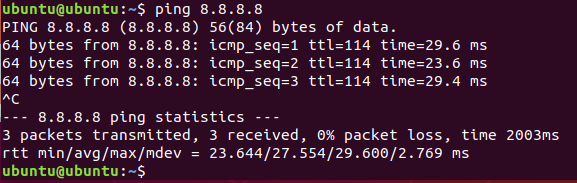


Figure 33 - Pinging Google DNS from the Client

All the traffic generated is captured in the tcpdump session on the Gateway, as visible.

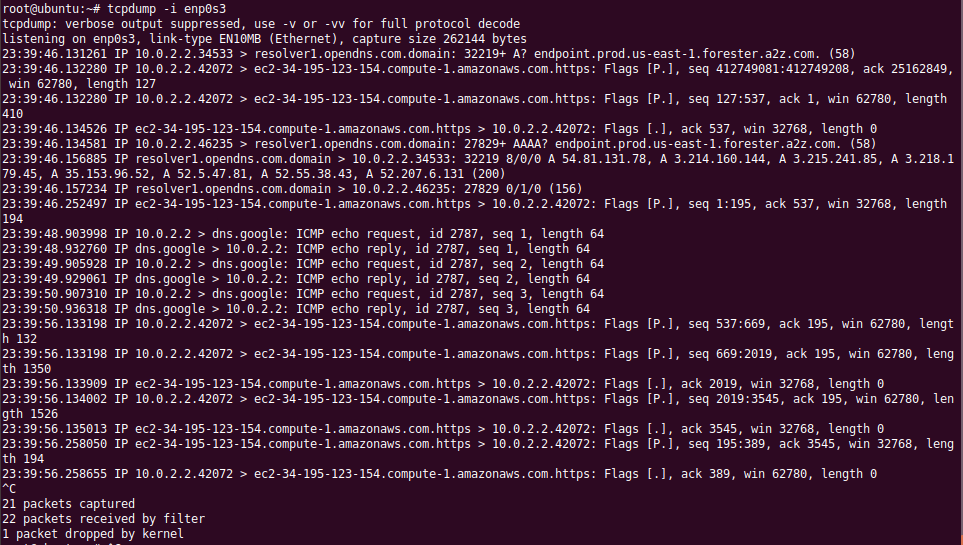


Figure 34 - tcpdump run from the Gateway

Using a software with a graphical interface such as Wireshark may provide a more rewarding user experience.

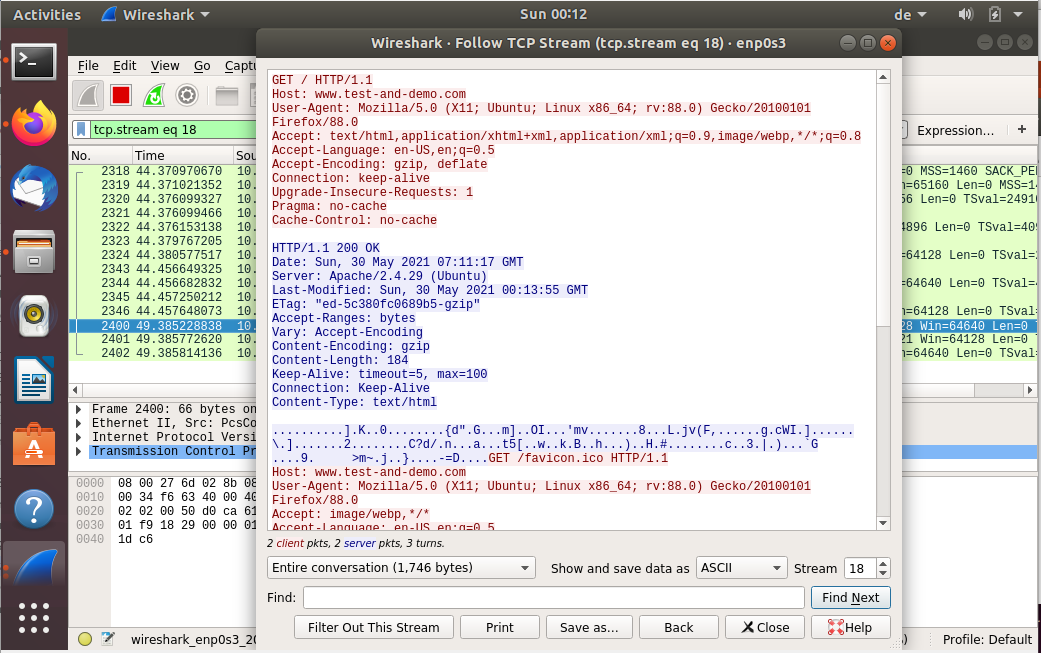


Figure 35 - Running Wireshark on the Gateway

## Changing Allowed Protocols on the Gateway

On the Gateway we have the possibility to decide which protocols are allowed through for the clients, and what is NATted and what not.

A simple way to test it is to edit the rc.sh firewall script and change two values: at the lines 45 and 48, change the values from “Y” to any other value (e.g. “N”).

Change from:

*# Allow web access to the Client (for Ubuntu updates)*

*export Client\_Allowed\_Web="****Y****"*

*# Allow web access to the Gateway (for Ubuntu updates)*

*export GW\_Allowed\_Web="****Y****"*

To:

*# Allow web access to the Client (for Ubuntu updates)*

*export Client\_Allowed\_Web="****N****"*

*# Allow web access to the Gateway (for Ubuntu updates)*

*export GW\_Allowed\_Web="****N****"*

After the rc.sh file is saved and it is run again, the client and the gateway will lose the ability to browse the web, while all the other services – name resolution, ping, etc. – will keep working as usual (including the test web page hosted on Apache2 on the gateway).

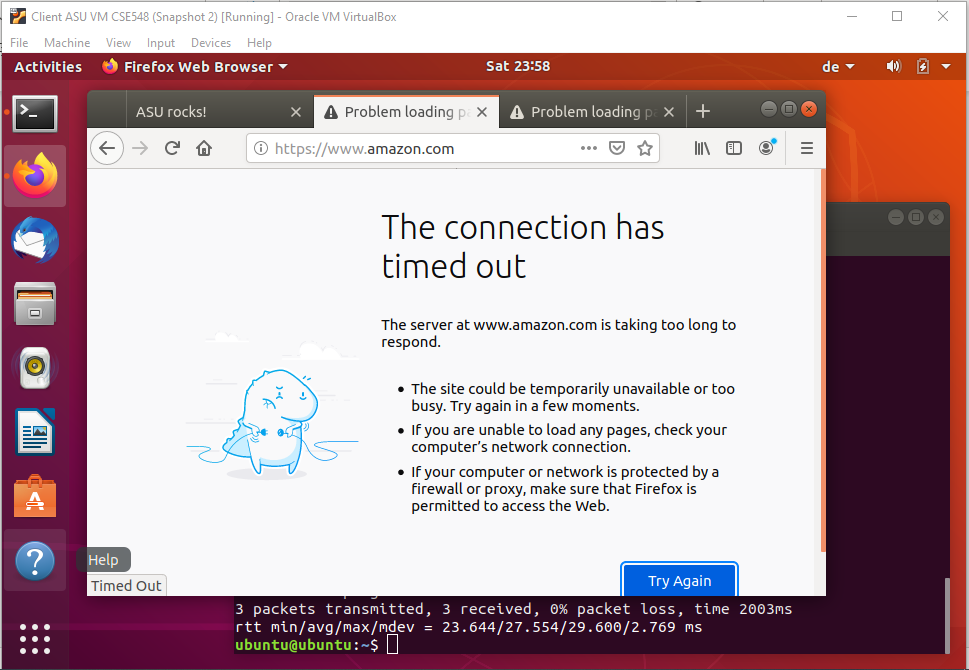


Figure 36 - Internet web sites timing out on the Client

It is worth noticing that the web sites will timeout after circa one minute from when the page is being requested. This is because on the iptables rules on the Gateway we have not specified a “reject” or sent a TCP Reset back to the client – we have simply dropped the packet and ignored it. The browser and the client operative system do not know if the Gateway is not allowing the traffic, or simply the Internet is very slow, or the destination web site is unreachable for any other reason. If desired, protocols can be explicitly rejected by setting the appropriate configuration on iptables.

# Conclusion

The following are the most interesting tricks I have learned:

1. By choosing to run the internal network without DHCP and assigning manually the addressed on the internal network, we have maximized the control and we are operating like we are placed in a dual-homed host architecture. It would be possible to enable the IPS system with Snort, for instance.
2. NetPlan and systemd don’t like each other’s. If you typed the wrong IP for the DNS server in NetPlan, after you correct it, it won’t be refreshed. You need to reboot your Linux box to fix this issue!
3. You can put logging rules at the end of every iptables chains to record a log when the pattern matching is not happening. In this way you can troubleshoot your iptables mistakes (this is what allowed me to figure out I was still using the wrong IP address for the DNS servers).

One of these rules can be like the following:

*$IPTABLES -A INPUT -j LOG --log-prefix DROPPED-INGRESS-*

This rule should be placed as the last of the INPUT chain. While troubleshooting, it is possible to open the /var/log/messages file with “**tail -f /var/log/syslog**”:

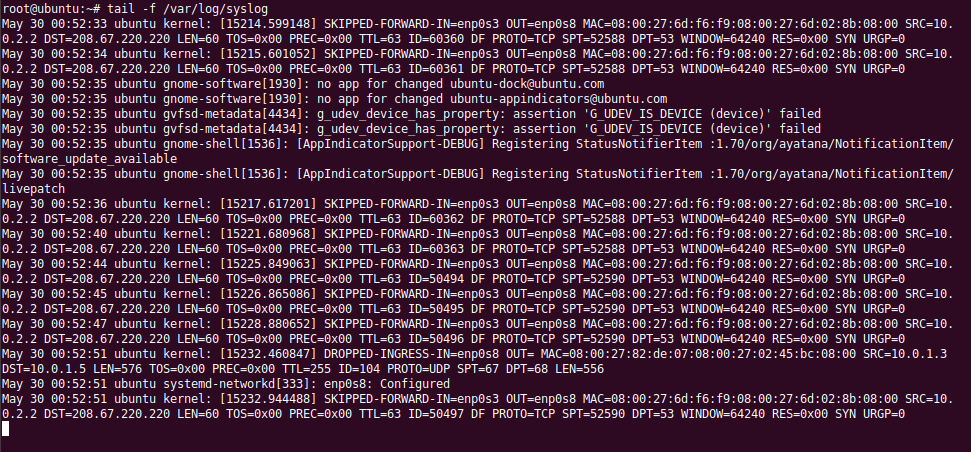


Figure 37 - Example of monitoring for rule drops via SYSLOG.

# Appendix A: Files for the Lab

Please find the list of files created for this lab and mentioned throughout this document, plus their GitHub link for download.

|  |  |
| --- | --- |
| rc.sh | <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/rc.sh> |
| index.html | <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/index.html> |
| test-and-demo.conf | <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/test-and-demo.conf> |
| ports.conf | <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/ports.conf> |
| 02-gateway-networks.yaml | <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/02-gateway-networks.yaml> |
| 02-internal-network.yaml | <https://github.com/markoer73/CSE-548/blob/main/Project%201%20-%20Packet%20Filter%20Firewall/02-internal-network.yaml> |

# References

* Linux NAT Tutorial: <https://www.karlrupp.net/en/computer/nat_tutorial>
* Ubuntu “Basic Iptables HOWTO”: <https://help.ubuntu.com/community/IptablesHowTo>
* “Iptables Tutorial: Ultimate Guide to Linux Firewall”: <https://phoenixnap.com/kb/iptables-tutorial-linux-firewall>

VIII. Table of Figures

[Figure 1 - Network Map of the Lab 1](#_Toc73261405)

[Figure 2 - VirtualBox Preferences 3](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261406)

[Figure 3 - VirtualBox Networks Configuration 3](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261407)

[Figure 4 - InternetNetwork configuration 3](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261408)

[Figure 5 - NatNetwork Configuration 3](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261409)

[Figure 6 - Gateway’s Adapter 1 set up on VirtualBox 3](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261410)

[Figure 7 - Gateway’s Adapter 2 set up on VirtualBox 3](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261411)

[Figure 8 - Client's Adapter Configuration 4](#_Toc73261412)

[Figure 9 - Gateway VM's interfaces as they appear at the Linux console 4](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261413)

[Figure 10 - Client VM's interface as it appears at the Linux console 4](file:///C:\Users\DJSM0911\Documents\Personal\Training\ASU%20MasterTrack\CSE%20548%20-%20Advanced%20Computer%20Network%20Security\GitHub\CSE-548\Project%201%20-%20Packet%20Filter%20Firewall\Project-Report-1%20Packet%20Filter%20Firewall.docx#_Toc73261414)

[Figure 11 - Gateway VM's initial forward routing and iptable rules 4](#_Toc73261415)

[Figure 12 - Client's VM initial network situation 5](#_Toc73261416)

[Figure 13 - Client's routing table 5](#_Toc73261417)

[Figure 14 - Initial Gateway internet connection 5](#_Toc73261418)

[Figure 15 - Gateway's routing table 5](#_Toc73261419)

[Figure 16 - Evidence of Gateway connectivity 7](#_Toc73261420)

[Figure 17 - Kernel routing on the Gateway 8](#_Toc73261421)

[Figure 18 - Routing on the Client 9](#_Toc73261422)

[Figure 19 - Lazy NAT configuration on the Gateway 10](#_Toc73261423)

[Figure 20 - Successfully pinging the Internet from the client 10](#_Toc73261424)

[Figure 21 - apt update command 10](#_Toc73261425)

[Figure 22 - apt dist-upgrade 11](#_Toc73261426)

[Figure 23 - Installation of tcpdump on the Gateway 11](#_Toc73261427)

[Figure 24 - Installation of Apache2 on the Gateway 11](#_Toc73261428)

[Figure 25 - Disabling of ufw 12](#_Toc73261429)

[Figure 26 - Adding the host name to /etc/hosts 13](#_Toc73261430)

[Figure 27 - the index.html of our test web site 13](#_Toc73261431)

[Figure 28 - Firefox browser testing out web site on the Gateway 14](#_Toc73261432)

[Figure 29 - /etc/hosts file on the Client 15](#_Toc73261433)

[Figure 30 - Firefox browser testing out web site from the Client 15](#_Toc73261434)

[Figure 31 - One run of the rc.sh script 16](#_Toc73261435)

[Figure 32 - Testing connectivity from the Client 17](#_Toc73261436)

[Figure 33 - Pinging Google DNS from the Client 17](#_Toc73261437)

[Figure 34 - tcpdump run from the Gateway 18](#_Toc73261438)

[Figure 35 - Running Wireshark on the Gateway 18](#_Toc73261439)

[Figure 36 - Internet web sites timing out on the Client 19](#_Toc73261440)

[Figure 37 - Example of monitoring for rule drops via SYSLOG. 20](#_Toc73261441)