**Policy Based Routing**

Juntao Zhong

ccnp (cisco certified network professional)

apr. 2019

NEWPORT HIGH SCHOOL, WA

## Purpose

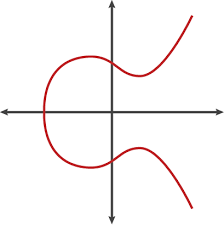
The purpose of this lab is to route or block different type of traffic by configurating the router. In this lab, I set up two identical web servers (different IP) which are both HTTP and HTTPS enabled. If we only have normal routing, both HTTP and HTTPS in both servers should be accessible. However, we can configure the router so that only HTTP in ServerA (HTTP server) is accessible while only HTTPS on ServerB (HTTPS server) is accessible.

## Backgroud:

Policy-based routing (**PBR**) tells routers to forward data packets based on the policies defined by network administer. The policies can be protocol type (HTTP or HTTPS), packet sizes, source/destination addresses, etc. In application, it’s a way to have customized policies to override automatic routing protocol decisions.

PBR works by selectively route/modify data packets based on access lists and other criteria (in this lab, we apply policies to a destination address, so we use extended access list).

**Apache** is the software we used to set up HTTP and HTTPS service. In this lab, both servers run Apache on Ubuntu OS. More than a third of the world’s active websites are ran on Apache.

Another part of the lab is setting an HTTPS server. HTTPS is secure because it uses SSL (Secure Sockets Layer), which uses asymmetric encryption. Asymmetric encryption means with a public encryption key, you can only encrypt the data, but not decrypt it. It’s a one-way street.

SSL algorithm is based on Elliptic Curve Cryptography (shown in the picture)—which if you input x, you can get an y-value, but it’s almost impossible to work back and get x if you are given y. The only way to decrypt the data (get the original x with y) is to a private key, which is only known locally in the server.

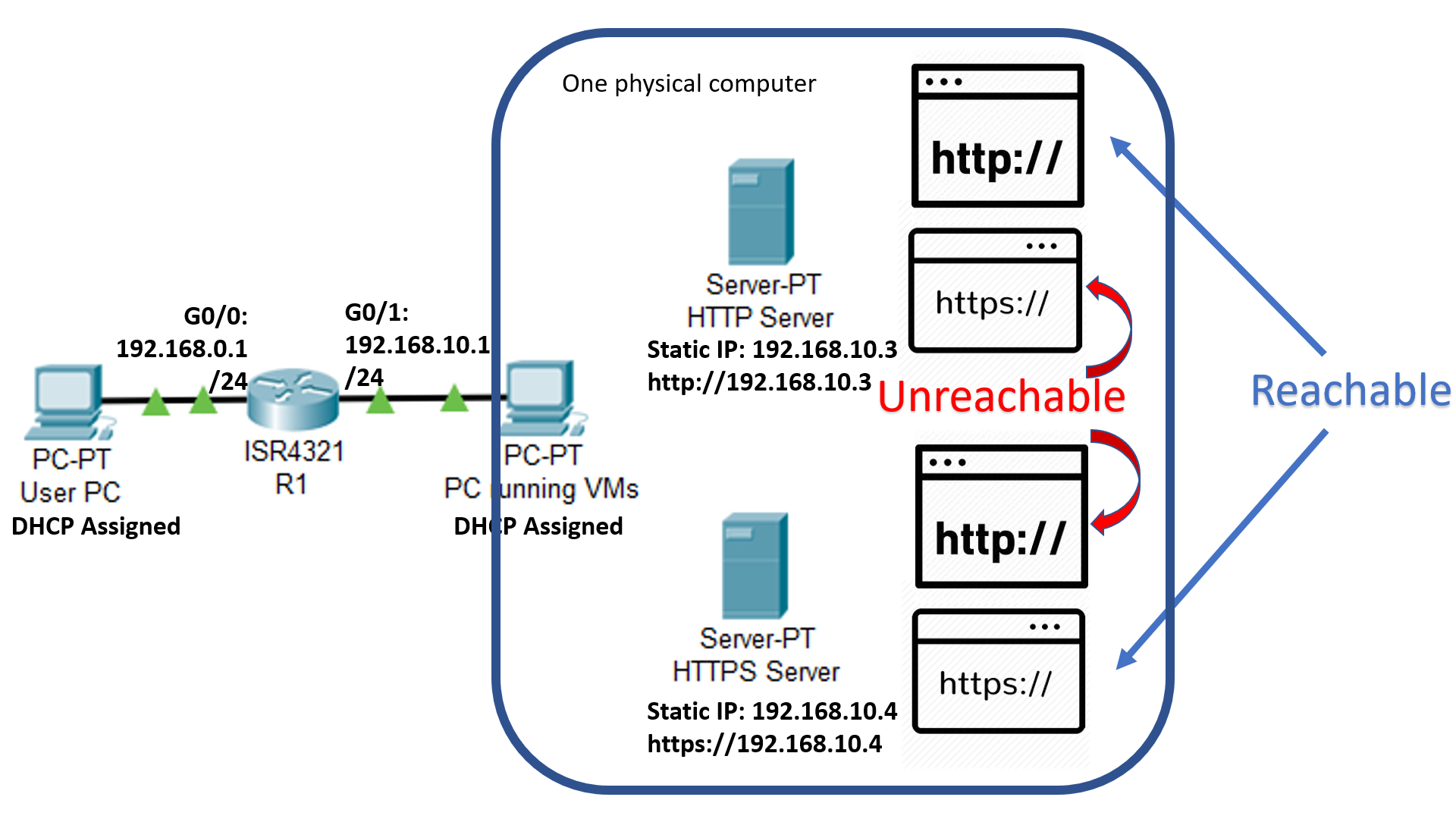
A public encryption key needs a certificate to operate. In a large enterprise, a certificate can cost up to $1500 from a trusted authority, who constantly mitigating attacks. However, in this small network, we will have Apache to use a **self-assigned SSL certificate** to run HTTPS.

## Lab summary

First, I install Linux Ubuntu Operation System on both serves. Then, install Apache to run both HTTP and HTTPS server.

After checking the connectivity of both servers on both website protocols. I implemented Policy-Based Routing that allows only HTTP and blocks HTTPS on HTTP Server, and block HTTP and allows only HTTPS Server.

## Lab Diagram



## Lab commands

### Router Commands:

ip policy route-map [name]-----------------define route-map (policy routing group)

access-list 100 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.3 eq www

-------------extended access list 100, anything HTTP (port 80, www traffic uses TCP) from 192.168.0.0/24 network belongs to access-list 100

route-map [name] permit 10------------------define policy NO.1 in this policy routing group

match ip address 10-----------if incoming traffic match the criteria in access-list 100

set ip next-hop 192.168.10.3---------then, route traffic to next-hop 192.168.10.3

set default interface Null0------------don’t route anything that is not in ACL 100

route-map [name] permit 20------------------define policy NO.2 in this policy routing group

### Linux Commands:

Abstract:

**HTTP config:**

apt-get update

apt install apache2

service apache2 start

---From this point, the HTTP server is already running. The IP address of HTTP website is server address (which can be static address or DHCP address).

**HTTPS Config:**

a2enmod ssl---------------------------------------------------Apache enable SSL (Secure Socket Layer)

service apache2 restart

mkdir /etc/apache2/ssl--------------------------------------------Make directory /etc/apache/ssl

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt

------------------generate a ssl key using RSA encryption algorithm, and push it out to /etc/apache2/ssl/apache.key and /etc/apache2/ssl/apache.crt

nano /etc/apache2/sites-available/default-ssl.conf

find these two lines and edit them inside file:

SSLCertificateFile /etc/apache2/ssl/apache.crt

SSLCertificateKeyFile /etc/apache2/ssl/apache.key

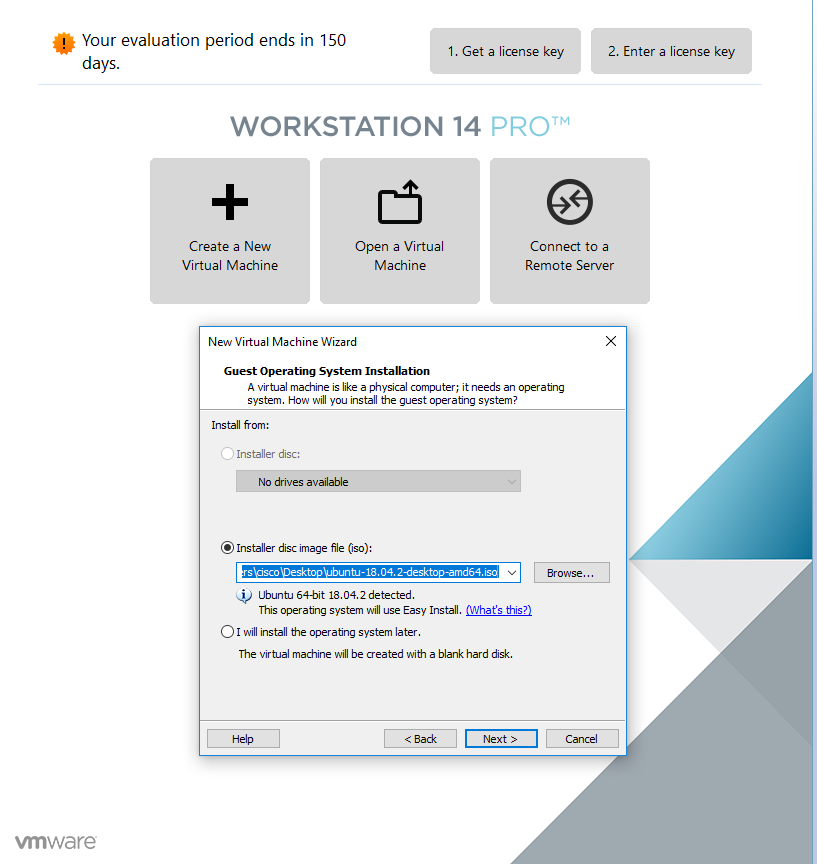
Save and exit /etc/apache2/sites-available/default-ssl.conf

a2ensite default-ssl.conf

service apache2 restart

## configuration of both servers and routers

### Linux APACHE Server Setup:

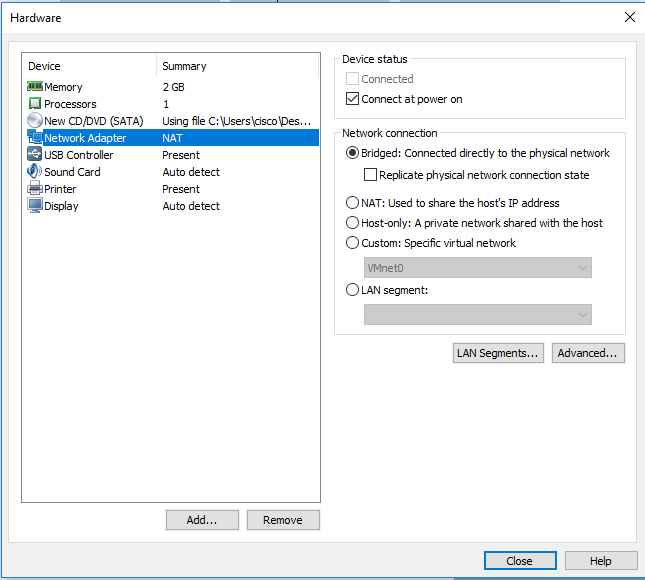


In vmware, load an ISO image of Ubuntu 18.04.2.

Other versions of Linux (such as Kali Linux) works as well, Apache is a universal application.

Run through the normal username and password registration.

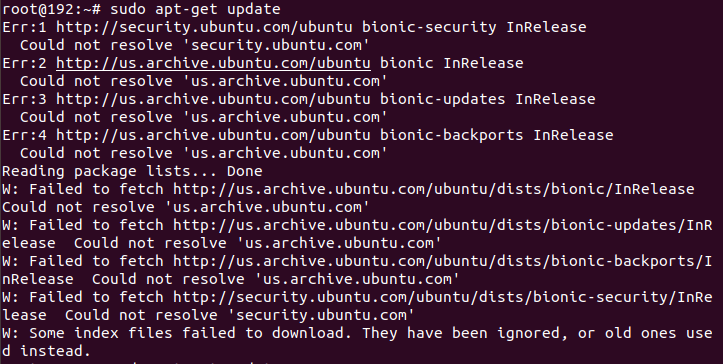
We have two servers, so we need to do this step twice.



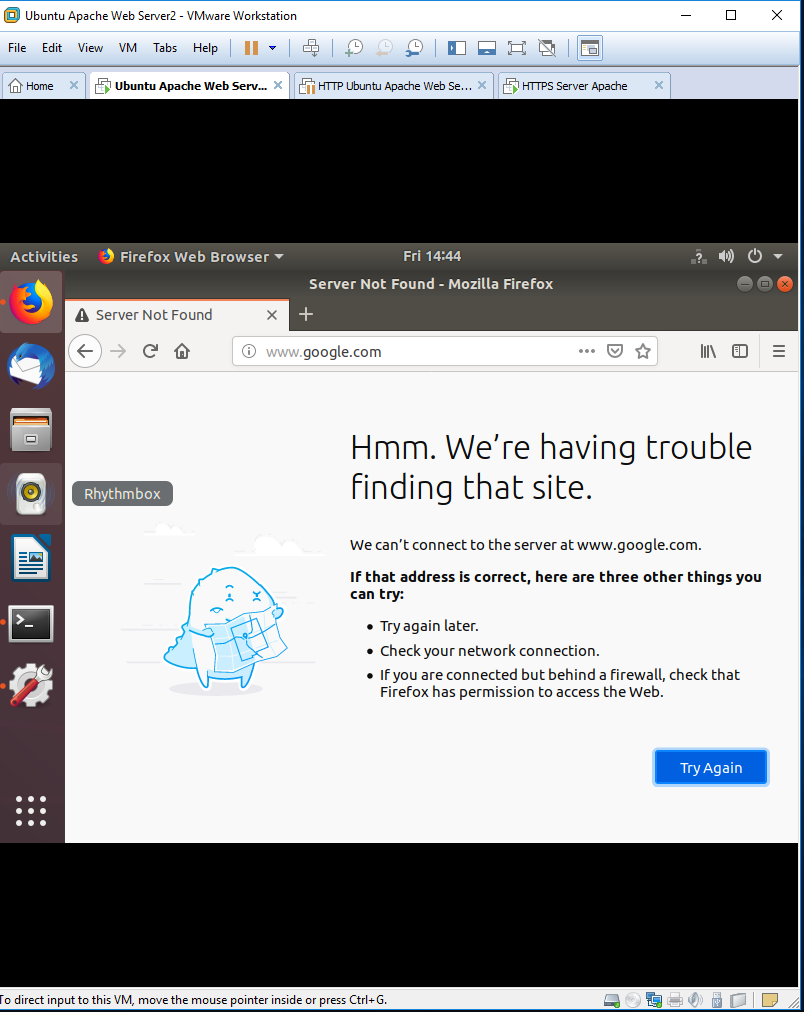
Since website servers need to have distinct IP address separated from physical PC and the other vm servers, we must use NAT for Network Adapter.

But, in my lab, my physical PC is connected to Internet with a wifi, using NAT means vm machine lost connections to outside Internet. This problem will be addressed later.

Then, I want to do a **sudo apt-get update**, which enables me to download applications like Apache. But the Error messages shows that the update failed.



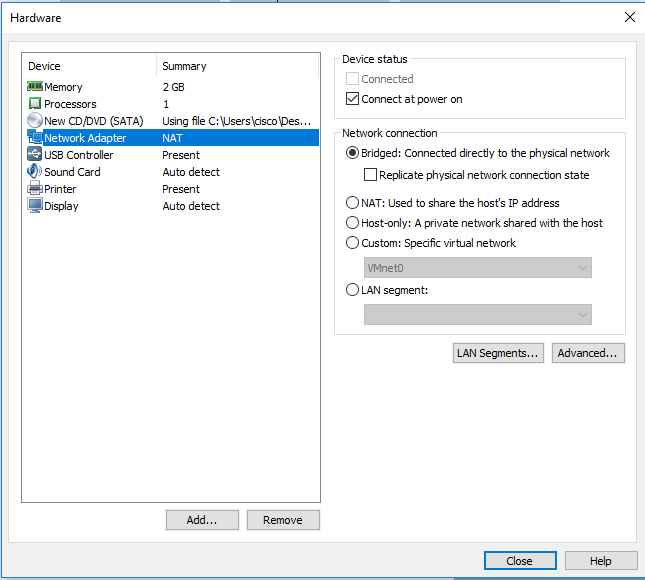
So, I check my connectivity—which shows that I am not connected.

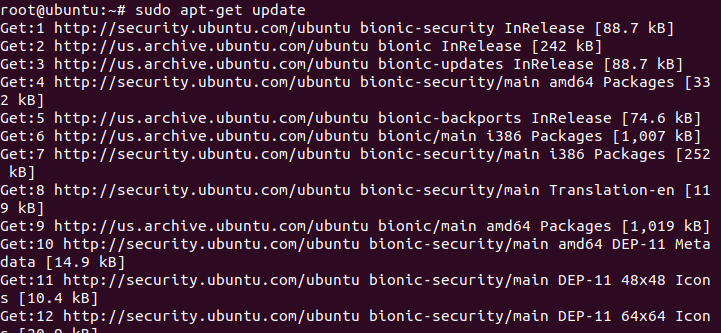


To get connectivity, first, we use bridged connection to share Internet connection with the physical PC.

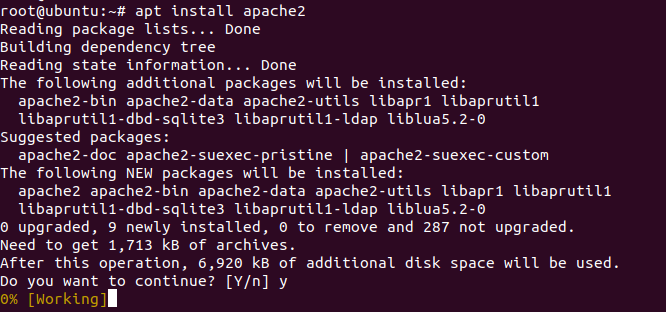
I’ll switch back to NAT connection after downloading all the necessary application packets.

In next image, I will try **sudo apt-get update** again, and this time, the update is successfully.





Then, issue **Apt install apache2** to install application Apache2. Give permission to use disk space, and Apache will be installed.

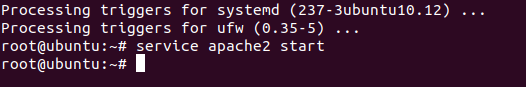


Then, start apache2 service.

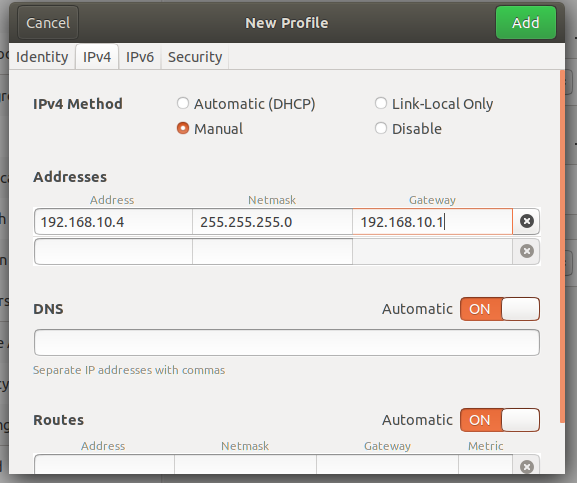
Be default, the firewall is not enabled. But if you are not sure about it, you can use

$ sudo ufw status ---check status

$ sudo ufw disable ---disable firewall



Apache2 is running from now on. We just need to switch back to NAT connection mode and configure a static IP address for our web server. DHCP assigned address also works, but I prefer static because static address is more reliable than DHCP.



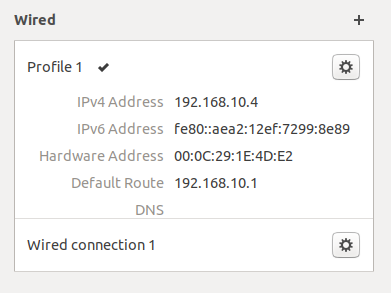
Since I’m using an Ubuntu GUI, I can just configure it through a graphic interface. Make sure to check it after configuring.

If you are using a CLI interface, do “nano /etc/network/interfaces”.  Edit the network adaptor options and save the configuration page.

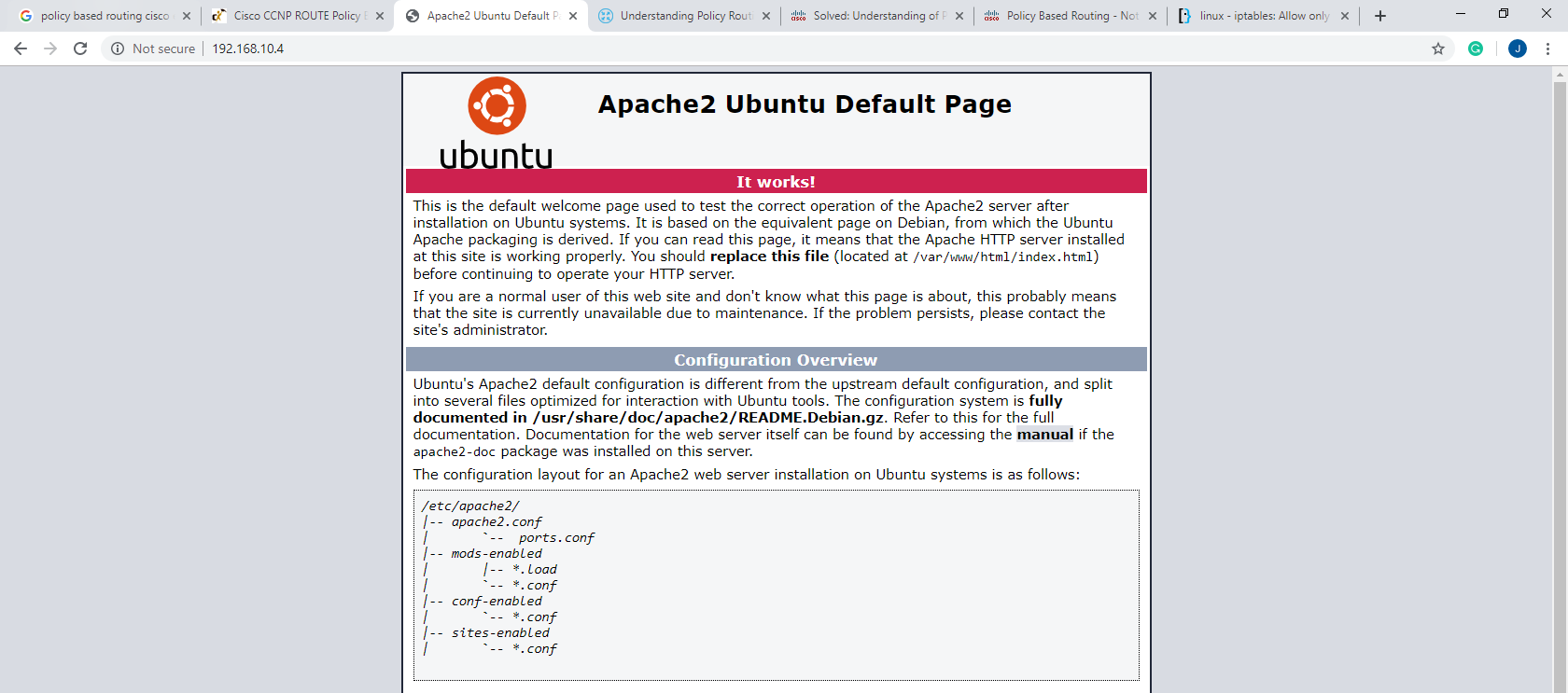
If server ip address are DHCP assigned, make sure to do (ens33 is the default network interface, change it to the interface id of your own).

“dhclient ens33 -v

route add default gw 192.168.10.1 ens33”

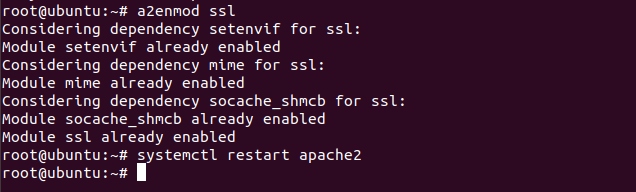


Double check that you have an address and server is connected through NAT. Go to a web browser on the physical PC that is running the vmware (Microsoft Edge won’t work). Type in server IP address. You can customize this page, but I’m not doing it in this lab.



### HTTPS SSL Setting:

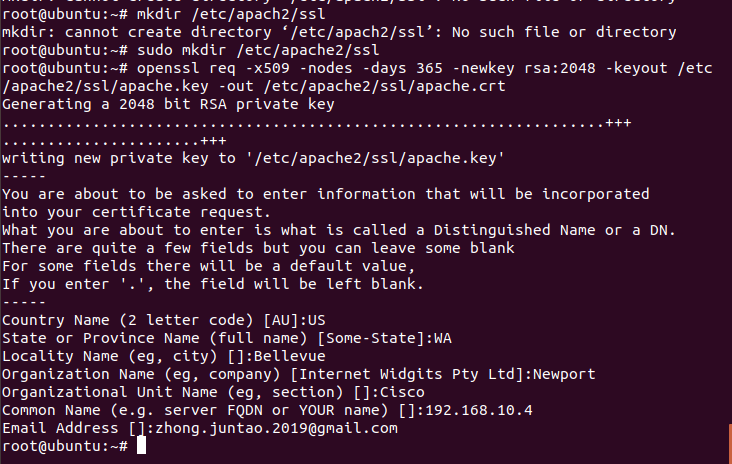
Type in **a2enmond ssl** to enable Secure Socket Layer. And restart Apache.



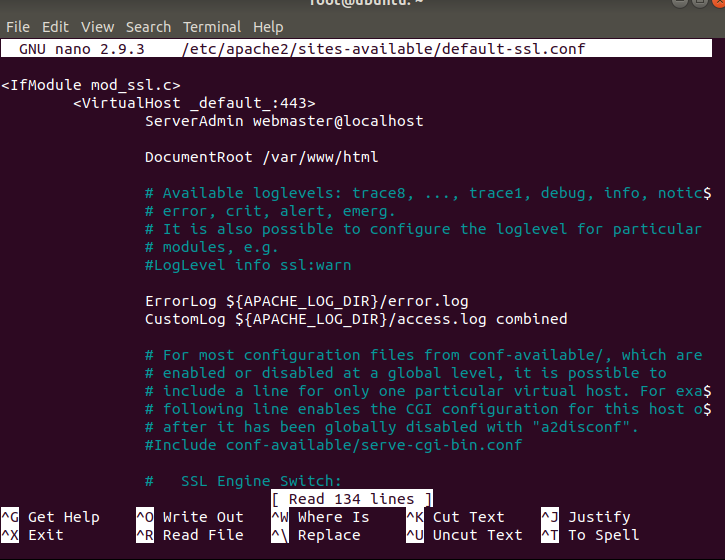
Then, we need to make a directory to store our self-assigned certificate. Type in **mkdir /etc/apache2/ssl** to make a directory in Apache2 folder.

Then, issue **Openssl req-x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out/etc/apache2/ssl/apache.crt.** Private key for un-decryption is export to “apache.key”, and the certification for public key is apache.crt.

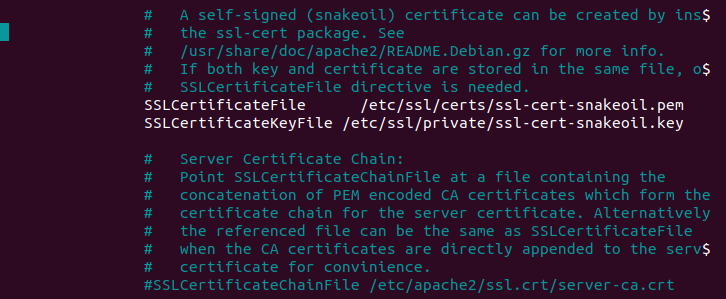
RSA 2048 defines the encryption algorithm to use. They will ask you several other personal questions for the certificate, the answer doesn’t really matter.



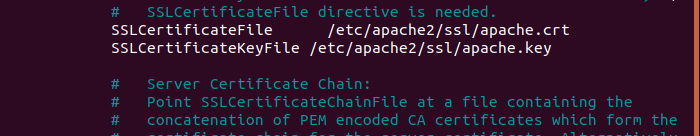
Then, enter **nano /etc/apache2/sites-available/default-ssl.conf.**



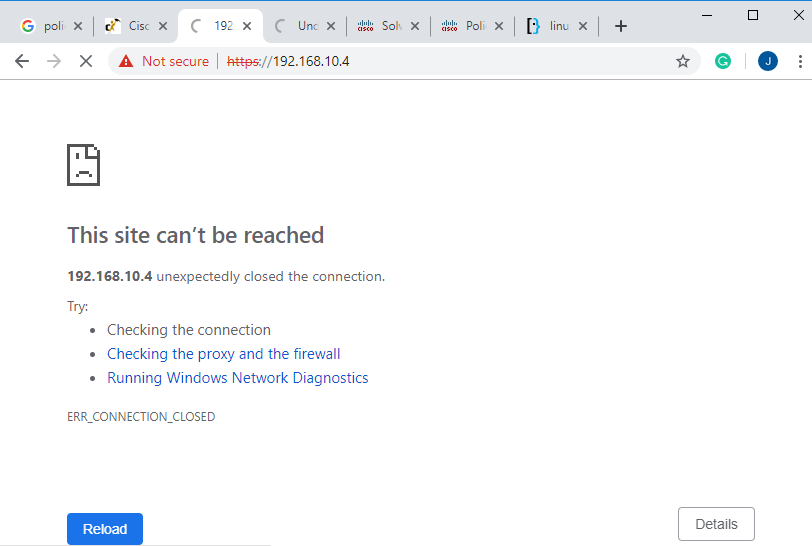
As you scroll down, you will find the code that directs to private key and ssl certification. You will need to change it to two files that we made just now.

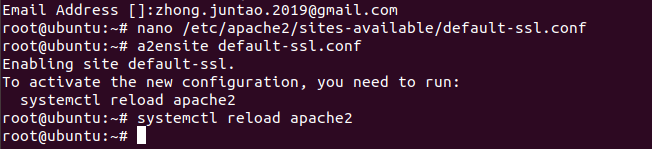


Yes, it’s changed.

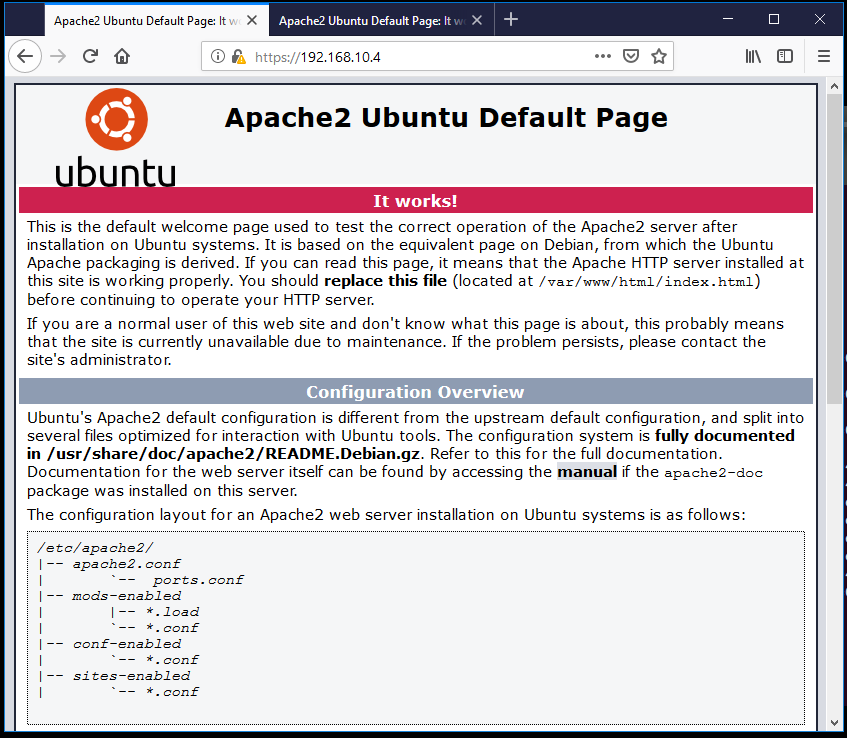


At this point, the HTTPS is still not running. You need to reload Apache for it to run.





It’s running now! The lock symbol means HTTPS secure connection; however, this HTTPS use a self-assigned certificate, so it has a triangle next to it.



### Router configuration:

Router(config)# hostname Router

ip dhcp excluded-address 192.168.0.1 192.168.0.2

ip dhcp excluded-address 192.168.10.0 192.168.10.11

ip dhcp pool Users

network 192.168.0.0 255.255.255.0

default-router 192.168.0.1

ip dhcp pool Servers

network 192.168.10.0 255.255.255.0

default-router 192.168.10.1

lease infinite

interface GigabitEthernet0/0

ip address 192.168.0.1 255.255.255.0

ip policy route-map Routing

no shutdown

interface GigabitEthernet0/1

ip address 192.168.10.1 255.255.255.0

no shutdown

access-list 100 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.3 eq www

access-list 101 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.3 eq 443

access-list 102 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.4 eq www

access-list 103 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.4 eq 443

route-map Routing permit 10

match ip address 100

set ip next-hop 192.168.10.3

set default interface Null0

route-map Routing permit 20

match ip address 101

set interface Null0

route-map Routing permit 30

match ip address 102

set interface Null0

route-map Routing permit 40

match ip address 103

set ip next-hop 192.168.10.4

set default interface Null0

With this configuration, router will route HTTP traffic to 192.168.0.3 and block the HTTPS traffic to it. And it will route HTTPS to 192.168.0.4, and block HTTP traffic to it, as expected in the lab diagram.

After going trying both HTTPS and HTTP on both routers several time, router will gather some network data—which we can explore using show comments.

**Router#show route-map**

route-map Routing, permit, sequence 10

Match clauses:

ip address (access-lists): 100

Set clauses:

ip next-hop 192.168.10.3

Nexthop tracking current: 0.0.0.0

192.168.10.3, fib\_nh:0,oce:0,status:0

default interface Null0

Default Interface tracking current: NULL

Null0, adj\_lh:0,oce:0,status:0

Policy routing matches: 33 packets, 3080 bytes

route-map Routing, permit, sequence 20

Match clauses:

ip address (access-lists): 101

Set clauses:

interface Null0

Interface tracking current: NULL

Null0, adj\_lh:0,oce:0,status:0

Policy routing matches: 18 packets, 1188 bytes

route-map Routing, permit, sequence 30

Match clauses:

ip address (access-lists): 102

Set clauses:

interface Null0

Interface tracking current: NULL

Null0, adj\_lh:0,oce:0,status:0

Policy routing matches: 48 packets, 3168 bytes

route-map Routing, permit, sequence 40

Match clauses:

ip address (access-lists): 103

Set clauses:

ip next-hop 192.168.10.4

Nexthop tracking current: 0.0.0.0

192.168.10.4, fib\_nh:0,oce:0,status:0

default interface Null0

Default Interface tracking current: NULL

Null0, adj\_lh:0,oce:0,status:0

Policy routing matches: 106 packets, 15941 bytes

**Router#show ip access-lists**

Extended IP access list 100

10 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.3 eq www (32 matches)

Extended IP access list 101

10 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.3 eq 443 (36 matches)

Extended IP access list 102

10 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.4 eq www (96 matches)

Extended IP access list 103

10 permit tcp 192.168.0.0 0.0.0.255 host 192.168.10.4 eq 443 (106 matches)

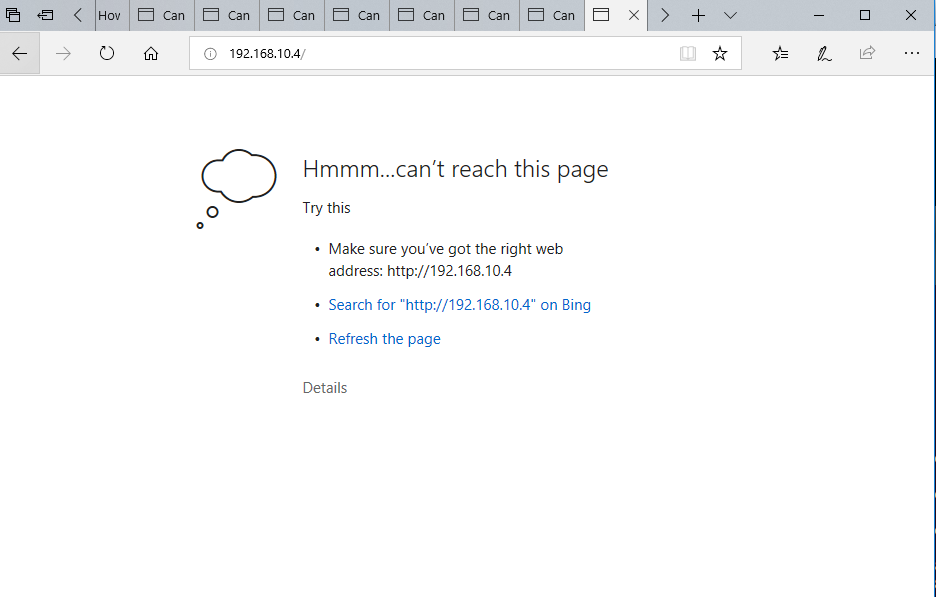
There should be matches whether the route forwards or blocks the packets—router must categorize the packet (with access-list) before making decision to forward it.

## Problems encountered

**Microsoft Edge don’t trust local developing Apache website:**

This problem haunted me for a while, as I tried desperately to troubleshoot my Apache server. It turned out that my Apache servers work perfectly fine, but just Microsoft Edge don’t trust the less-secure local servers (it doesn’t trust self-assigned certificate by default, which makes sure that client is not talking to the fake server).

The solution is very easy: use Chrome or Firefox instead.



**Server Linux VM can’t connect to Internet and download Apache under NAT mode:**

Apache needs to be download online, so server much connect to the Internet first. Thus, we can bridge the network to download necessary files, then use NAT connections.

**Router routes both all traffic between two networks without blocking it:**

Since both networks are directly connected to the router, not routing protocol is necessary the communication in two. If router routes everything across, then Policy-Based Routing is not working.

I suspected that I placed the extended ACL in the wrong interface; and I find the problem:

**Router(config-if)#do debug ip policy**

Policy routing debugging is on

Router(config-if)#

\*Apr 26 15:04:56.527: IP: s=192.168.10.4 (GigabitEthernet0/1), d=192.168.0.3, le n 52, **FIB policy rejected(no match)** - normal forwarding

\*Apr 26 15:04:56.527: IP: s=192.168.10.4 (GigabitEthernet0/1), d=192.168.0.3, le n 40, **FIB policy rejected(no match)** - normal forwarding

\*Apr 26 15:04:56.535: IP: s=192.168.10.4 (GigabitEthernet0/1), d=192.168.0.3, le n 1464, **FIB policy rejected(no match)** - normal forwarding

… and a lot of **FIB policy rejected(no match)…**

Theatrically every webpage traffic in this network should have a match, but there is no match. It’s because that I put command **ip policy route-map Routing** on the wrong interface (G0/1). The right interface should be one facing users, not servers. It makes sense though: traffic must be decided where to go first, before it arrives at the destination interface.

So, I changed it, and the debug message shows.

Debug messages continue:

\*Apr 26 15:27:58.815: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4, len 52, **FIB policy match**

\*Apr 26 15:27:58.815: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4, len 52, PBR Counted

\*Apr 26 15:27:58.815: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4, len 52, policy match

\*Apr 26 15:27:58.815: IP: route map Routing, item 30, permit

\*Apr 26 15:27:58.815: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4 (Null0), len 52, **policy routed**

\*Apr 26 15:27:58.815: IP: GigabitEthernet0/0 to Null0 192.168.10.4

\*Apr 26 15:27:58.819: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4, len 52, FIB policy match

\*Apr 26 15:27:58.819: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4, len 52, PBR Counted

\*Apr 26 15:27:58.815: IP: s=192.168.0.3 (GigabitEthernet0/0), d=192.168.10.4 (Null0), len 52, policy routed

\*Apr 26 15:27:58.815: IP: **GigabitEthernet0/0 to Null0** 192.168.10.4

All the traffic has matches, and the debug message record the decisions made by the router, such as “policy routed” (pass) or “to Null0” (blocked). At this point, the HTTP and HTTPS servers work as expected.

## Summary

In this lab, I install and configure two Apache servers based on Linux Ubuntu. Besides the basic HTTP, I allow servers to self-assign certificate to run HTTPS. Then, I configure PBR, which blocks and sends traffic through access list.

I also expand my networking knowledge by learning concepts such as asymmetric encryption, ssl, and self-assigned certificates.