5/15/19

Policy based routing

Jeffrey Xu

CCNP 6-7

**Purpose:**

The purpose of this lab was to set restrictions on access to different web pages by using access control lists and binding them to interfaces. By doing this, it enables us to allow certain people and deny certain people on to websites just like a normal ACL does, but we can put restrictions such as time, labels and what protocols we want to be allowed on that interface.

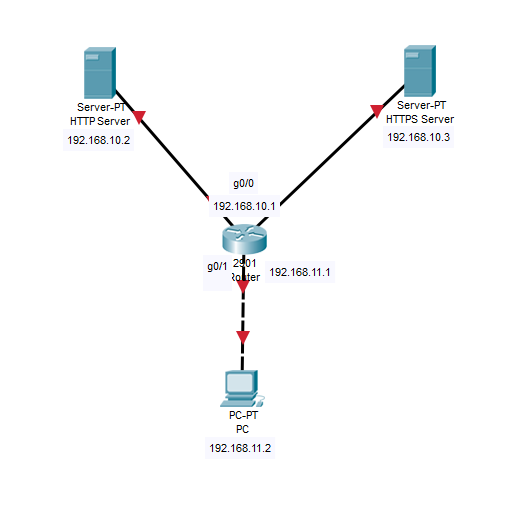
**Background:**

Policy based routing can be called an extension for normal and extended access lists. We can apply ones we have previously made and add more features to it. For example, there are some people who want to download a “large file” off of the internet through the company’s routers, but because you don’t want to people to do that, you can set an ACL and deny anything that is above a certain file size. There are also many other restrictions you can add onto an access list to allow and deny different types of traffics. This allows an access list to use these protocols in the same direction based on policies. This could also be described as a type of QoS that allows restricted access to be fully implemented. Policy based routing also allows priority data to be transferred just like QoS.

**Lab setup:**

This lab included 1 PC, 1 2901 routers, and 2 VMware machines running Ubuntu desktop with http and https apache2 servers running on both of them. We needed to set up the router to allow only http access for one of the Ubuntu desktops and only https on the other one even though they were both running both servers.

Topology



Configurations:

**Router#show run**

Building configuration...

hostname Router

ip dhcp excluded-address 192.168.10.1

ip dhcp excluded-address 192.168.10.4 192.168.10.255

ip dhcp pool Bot

network 192.168.11.0 255.255.255.0

default-router 192.168.11.1

ip dhcp pool Top

network 192.168.10.0 255.255.255.0

default-router 192.168.11.1

lease infinite

interface GigabitEthernet0/0

ip address 192.168.10.1 255.255.255.0

duplex auto

speed auto

interface GigabitEthernet0/1

ip address 192.168.11.1 255.255.255.0

ip policy route-map Routing

duplex auto

speed auto

access-list 100 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.2 eq www

access-list 101 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.3 eq 443

access-list 102 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.2 eq 443

access-list 103 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.3 eq www

route-map Routing permit 10

match ip address 100

set ip next-hop 192.168.10.2

route-map Routing permit 20

match ip address 101

set ip next-hop 192.168.10.3

route-map Routing permit 30

match ip address 102

set interface Null0

route-map Routing permit 40

match ip address 103

set interface Null0

end

**Router#show ip dhcp binding**

Bindings from all pools not associated with VRF:

IP address Client-ID/ Lease expiration Type

Hardware address/

User name

192.168.10.2 000c.29ea.75fb Infinite Automatic

192.168.10.3 000c.2903.d98c Infinite Automatic

192.168.11.2 014c.cc6a.388e.e3 Apr 25 2019 07:20 PM Automatic

**Router#show route-map**

route-map Routing, permit, sequence 10

Match clauses:

ip address (access-lists): 100

Set clauses:

ip next-hop 192.168.10.2

Policy routing matches: 12 packets, 1107 bytes

route-map Routing, permit, sequence 20

Match clauses:

ip address (access-lists): 101

Set clauses:

ip next-hop 192.168.10.3

Policy routing matches: 18 packets, 2081 bytes

route-map Routing, permit, sequence 30

Match clauses:

ip address (access-lists): 102

Set clauses:

interface Null0

Policy routing matches: 9 packets, 594 bytes

route-map Routing, permit, sequence 40

Match clauses:

ip address (access-lists): 103

Set clauses:

interface Null0

Policy routing matches: 6 packets, 396 bytes

**Router#show ip access-lists**

Extended IP access list 100

10 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.2 eq www (12 matches)

Extended IP access list 101

10 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.3 eq 443 (18 matches)

Extended IP access list 102

10 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.2 eq 443 (18 matches)

Extended IP access list 103

10 permit tcp 192.168.11.0 0.0.0.255 host 192.168.10.3 eq www (12 matches)

**Linux Commands:**

apt-get update

apt install apache2

service apache2 start

a2enmod ssl

service apache2 restart

mkdir /etc/apache2/ssl

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

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

\*\*inside, change stuffs

SSLCertificateFile /etc/apache2/ssl/apache.crt

SSLCertificateKeyFile /etc/apache2/ssl/apache.key

\*\*

a2ensite default-ssl.conf

service apache2 restart

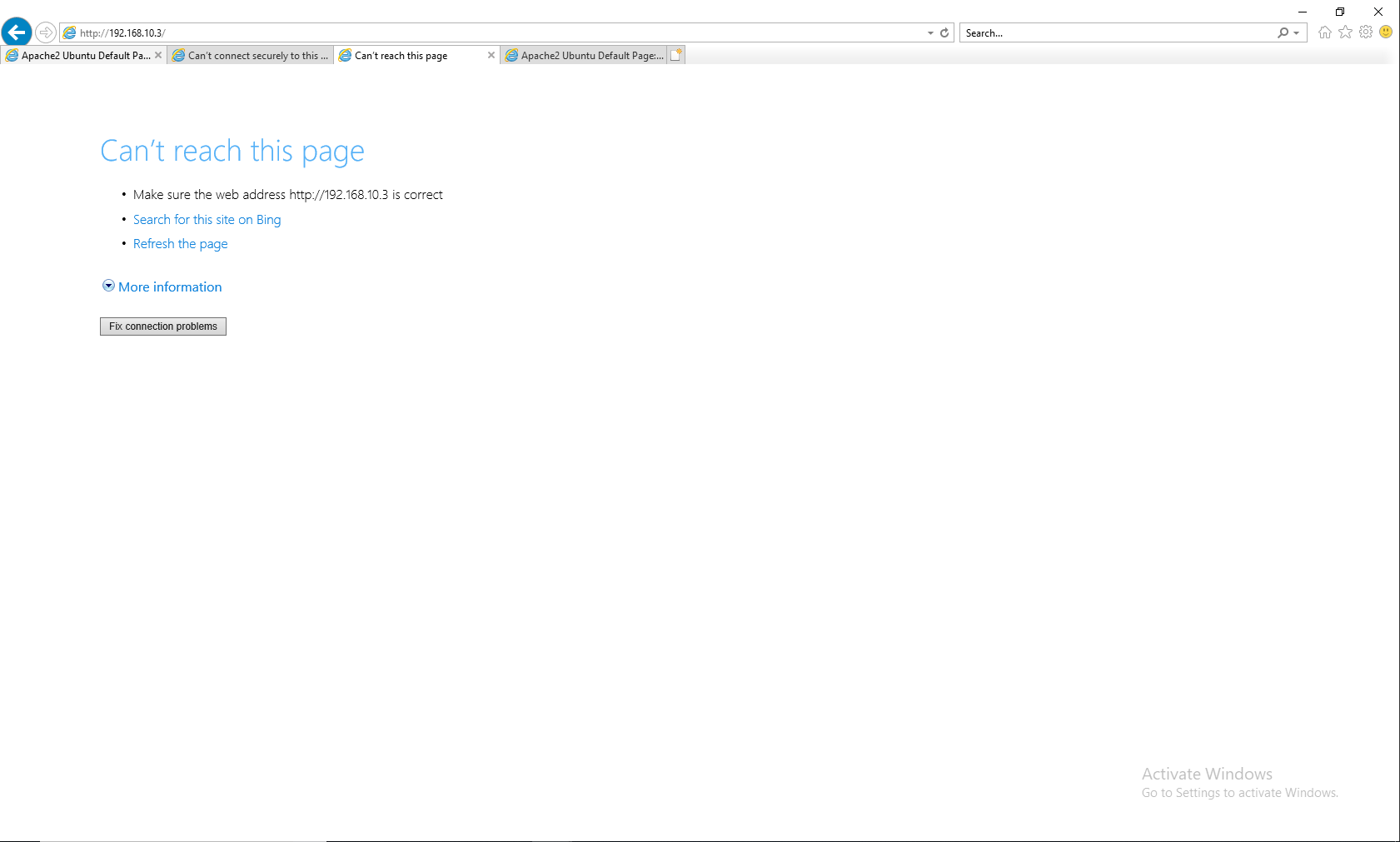
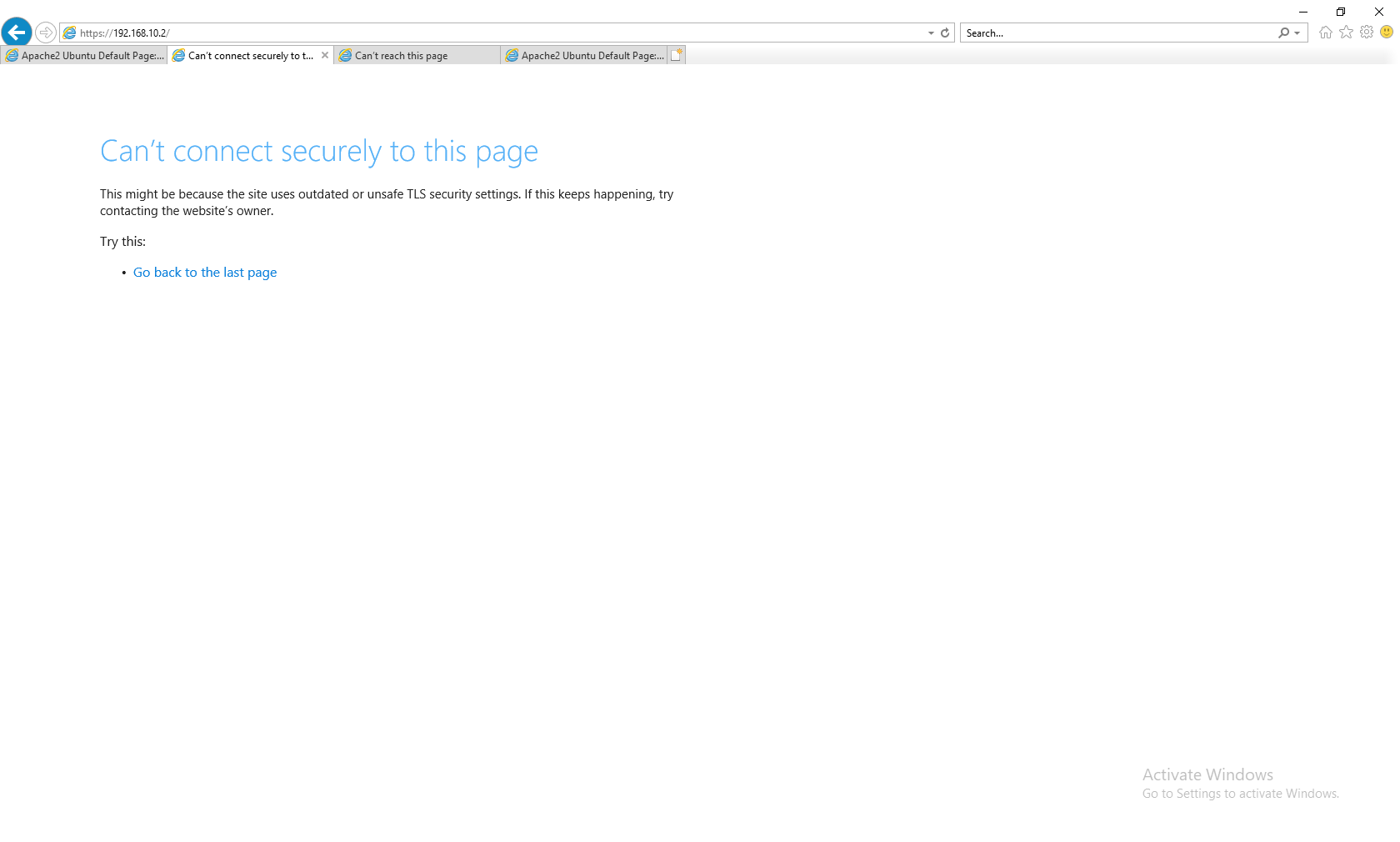
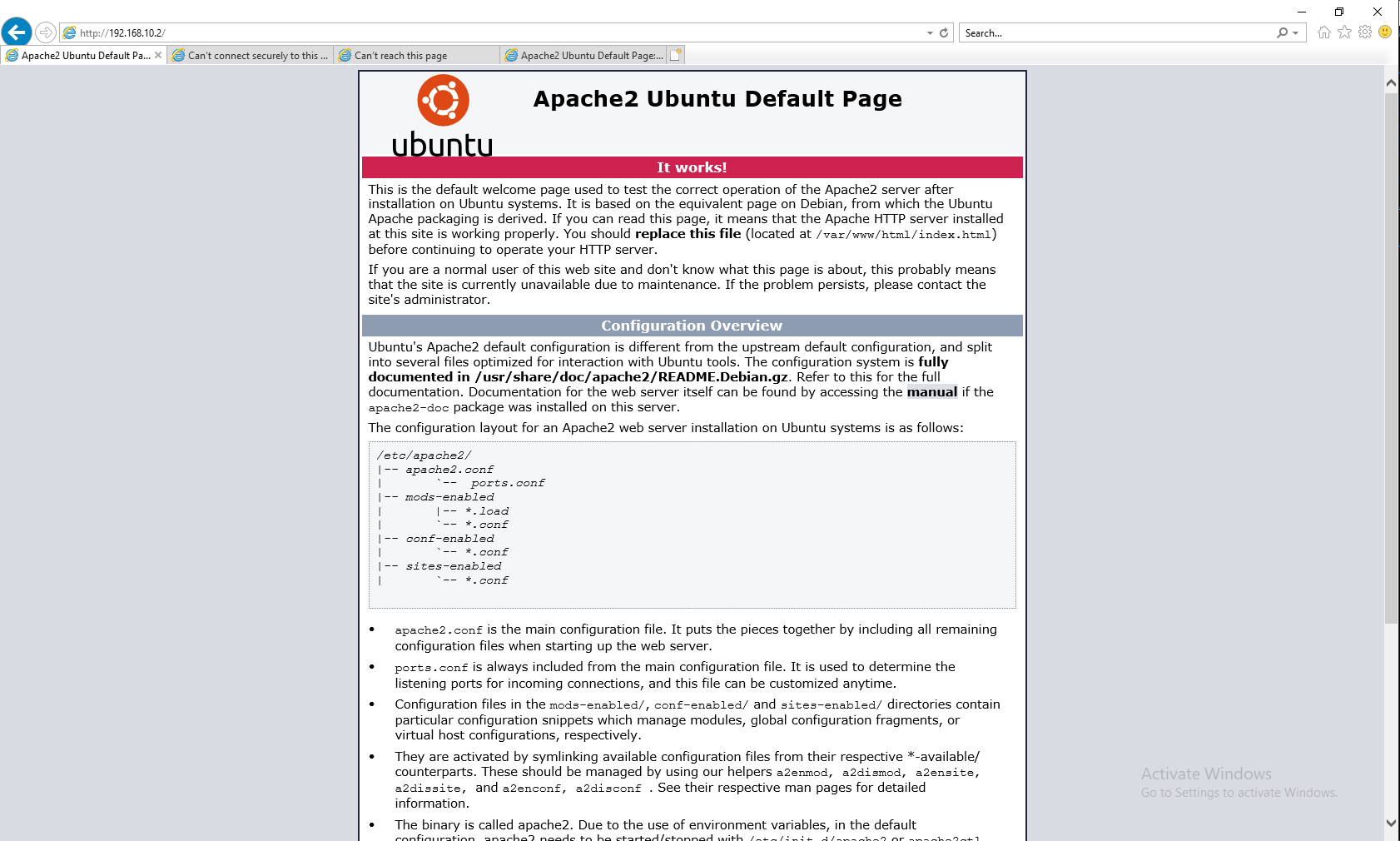
dhclient ens33 -v

route add default gw 192.168.10.1 ens33

\*Please note the “route add….” command varies depending on the specific VM

|  |  |
| --- | --- |
|  | This page is the start of the creation process for a new VM. This lab only requires a Typical machine. |
|  | Install the live server Ubuntu ISO for the creation of a Linux machine with the necessary image. |
|  | Basic naming/registration information with any names and passwords within certain parameters (no capital letters, etc.) |
|  | Naming the Virtual machine and storage location |
|  | For this lab, we do not need more than 20 GB of disk capacity. |
|  | Finish the VM wizard process |
|  | Basic startup information being displayed in the console |
|  | Selecting a language |
|  | Setting location |
|  | For this lab, we only need a simple installation of Ubunut |
|  | Press Done, by default network connections are set by DHCP which is what we need. –Through trial and error, a static set IP is much more simple unless you know what you are doing. |
|  | We do not need an indirect proxy to access the internet |
|  | An alternative mirror for Ubuntu is also not used in this lab (irrelevant to PBR) |
|  | Use the entire disk for the Ubuntu server, as we have already allocated space for the VM in the wizard |
|  | Select the already allocated disk |
|  | Press Done after reading the basic summary information |
|  | Confirm destructive action, as this will begin the installation (at the cost of now unnecessary software) |
|  | Set basic naming and password information. SSH identity is not relevant to PBR |
|  | We do not need any external snaps for our server environment for PBR |
|  | Installing process…. |
|  | Press reboot upon the finished install |
|  | Startup screen…. |
|  | More startup screen…. |
|  | Even more startup screen…. |
|  | More startup screen!....... |
|  | Log into the Linux VM. All other commands can now be implemented in the device (shown above) for PBR |

The screenshots below demonstrate that one VM indicated by its IP address is able to receive and send HTTP packets but not HTTPS while the other VM (also indicated by IP) can handle HTTPS but not HTTP. This is shown by the Apache page or by a “Cannot connect” page.



**Problems:**   
During the lab, the only problem I had was with the IP addresses and DHCP. When I set up DNCP on my router, it successfully gave both of my servers an IP address and IP connectivity was established between certain devices. But the problem DHCP gave me was that it wouldn’t be able to ping across to a different network. For example, my PC on a different network could ping the default gateway of my servers network, but couldn’t ping the servers themselves. At the same time, the router could ping everything. This got me confused so I just switched to a static set IP address by modifying –sudo nano /etc/netplan/50(tab) and it would let me set a static IP address and it fixed everything.

**Conclusion:**

This lab is a good extension of what we learned about Access lists back in CCNA. It also gave us a real world implementation on how to use certain commands to block which ones. Overall this lab was a simple lab able to be finished in a 2-3 days at max.