Policy Based Routing

CCNP Lab 9

2018-2019

Nipun Chhajed

Cisco CCNP - Hoffman and Mason - Periods 6 and 7

Policy Based Routing Lab 9

Purpose

There main reason for accomplishing this lab was to learn how to restrict and permit data in a Local Area Network. To do this successfully, we had to use access-lists combined with route-maps to direct different data to their respective destinations. Along with that, this lab teaches us how to use Apache to set up HTTP and HTTPS web services on linux machines. These were the “destinations” for the data, as certain users could not access one or the other.

Background Information

To understand the inner workings of policy-based routing, it is essential to first understand how Access Control Lists (ACLs) work. More commonly known as, “access-lists”, ACLs allow for filtering of packets in a network. Using access lists, one can permit or deny data packets from travelling through certain interfaces, and can permit or deny packets from reaching certain destinations. The two types of ACLs that are most commonly used are: extended and standard ACLs. Standard ACLs have a range of 1 through 99 and solely permit and deny packets based on their IP addresses. Extended ACLs, on the other hand, range from 100 to 199, and make decisions based on IP addresses and the networking protocols attached to the packets. A standard ACL could, for example, deny any packet that has the ip address 192.168.1.1 from reaching the host. This is, in most cases, too restrictive, because certain packets that come from this ip address, may be using other protocols that the user does want to allow to reach the destination. This is

where extended ACLs come in handy. The user can use one to look for packets with the IP address 192.168.1.1, but they can permit certain packets, with the same IP address, that use the HTTP protocol for example and deny the rest. Usually, when ACLs are created, they are configured only for one interface to use. However, using route maps, one can configure ACLs to not only act on one interface but also: change different fields in certain packets, transport packets in specific ways, and change different aspects of routes. Route maps contain combinations of “match” and “set” commands. “Match” commands create a criteria for the packets to be matched with, based on the policy. “Set” commands define what is going to be done with the matched packets. Possible actions that could be taken with the packets include: modifying the packet, forwarding the packet, or letting the packet bypass its normal route. In essence, Policy Based Routing allows networking managers to customize and micromanage how their networks manage different types of traffic, not only based on a packet’s ip address, but also on other factors such as protocols and packet size, to name a few.

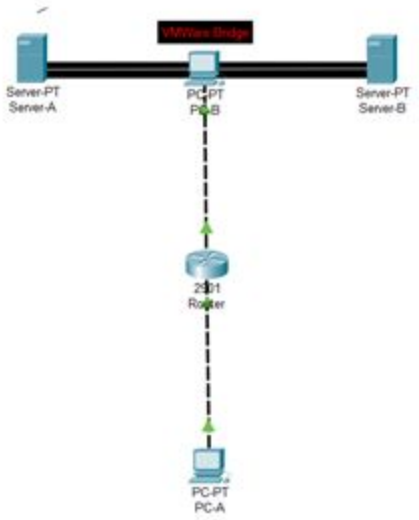
Lab Summary

We set up two Linux virtual machines running Apache web servers. Both web servers were configured to be accessed through HTTP and HTTPS. The VMs were also automatically assigned IP addresses from the router using DHCP. We then started the Policy Based Routing part of the lab, where we set up multiple ACLs and Route Maps, to make it so that one host could access both HTTP and HTTP web servers, while the other one was only allowed to access the HTTP web server.

Lab Commands

ip dhcp pool [POOL NAME]	Creates a DHCP pool with the user specified name
network [network address] [wildcard mask] [area-id]	Creates the network from which DHCP will distribute addresses to the hosts
default-router [ip address]	Sets the default gateway for the DHCP pool so that the hosts have a default gateway to get the addresses from
access-list [number] [permit/deny] [protocol] host [host ip address] host [destination ip address] eq [protocol]	Creates an access list. If number is greater than 100, then it is an extended access list.
route map [name]	Generates a route map for PBR
match ip address [ip or list number]	Connects the route map to the specified access list or ip address
set interface [interface]	Connects the desired interface to the route map
ip policy route-map [name]	Sets the route map to the interface desired

Network Diagram



Device	IP Address
R1	
G0/0	192.168.1.1
/24	
G0/1	192.168.2.1
/24	
PC-A	DHCP
PC-B	DHCP
Server-A	DHCP
Server-B	DHCP

Configurations

Router Configuration:

Router# show run

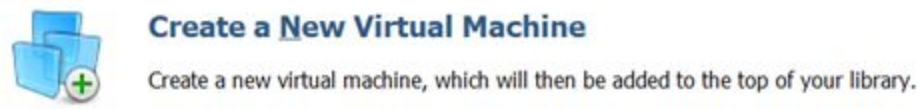
```
hostname Router
boot-start-marker
boot-end-marker
no aaa new-model
memory-size iomem 10
ip cef
ip dhcp excluded-address 192.168.1.1
ip dhcp excluded-address 192.168.2.1
ip dhcp pool LOCAL-HOST
 network 192.168.1.0 255.255.255.0
 default-router 192.168.1.1
 domain-name ccnp-lab.com
ip dhcp pool HTTP/HTTPS
 network 192.168.2.0 255.255.255.0
```

```
 default-router 192.168.2.1
 domain-name ccnp-lab.com
no ipv6 cef
multilink bundle-name authenticated
voice-card 0
license udi pid CISCO2901/K9 sn FTX15208074
license accept end user agreement
license boot module c2900 technology-package securityk9
license boot module c2900 technology-package uck9
vtp domain cisco
vtp mode transparent
redundancy
interface Embedded-Service-Engine0/0
 no ip address
 shutdown
interface GigabitEthernet0/0
 ip address 192.168.1.1 255.255.255.0
 ip policy route-map POLICY
 duplex auto
 speed auto
interface GigabitEthernet0/1
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto
interface Serial0/0/0
 no ip address
 shutdown
 clock rate 2000000
interface Serial0/0/1
 no ip address
 shutdown
 clock rate 2000000
interface GigabitEthernet0/1/0
 no ip address
 shutdown
 duplex auto
 speed auto
ip forward-protocol nd
no ip http server
```

```
no ip http secure-server
access-list 101 permit tcp host 192.168.1.2 host 192.168.2.3 eq 443
access-list 101 permit tcp host 192.168.1.2 host 192.168.2.4 eq www
access-list 102 permit tcp host 192.168.1.2 host 192.168.2.3 eq www
access-list 102 permit tcp host 192.168.1.2 host 192.168.2.4 eq 443
route-map POLICY permit 10
 match ip address 101
 set interface GigabitEthernet0/1
route-map POLICY permit 20
 match ip address 102
 set interface Null0
control-plane
mgcp profile default
gatekeeper
 shutdown
line con 0
line aux 0
line 2
 no activation-character
 no exec
 transport preferred none
 transport output lat pad telnet rlogin lapb-ta mop udptn v120 ssh
 stopbits 1
line vty 0 4
 login
 transport input all
scheduler allocate 20000 1000
end
```

Configuration Screenshots

Create VMs:



Using VMWare, create two new Linux 18.04 virtual machines. Select automatic installation. Once it's done installing, bridge the connection between the PCs and the VMs and set them to collect and IP address automatically.

Router Configuration Specifics:

Configure the routers for DHCP. First create 2 DHCP pools called "LOCAL-HOST" and "HTTP/HTTPS". Then you add the networks that correspond to each pool. Finally, make sure to set the default gateway for the hosts that will need the ip addresses. **AFTER CONFIGURING THE VMs:** Move back to the router to configure the access lists and route maps. For PBR, no data should be denied. Any data you want to permit and deny should all be denied. For this lab we used the following command setup for the access lists:

access-list 101 permit tcp host 192.168.1.2 host 192.168.2.3 eq 443
access-list 101 permit tcp host 192.168.1.2 host 192.168.2.4 eq www
access-list 102 permit tcp host 192.168.1.2 host 192.168.2.3 eq www
access-list 102 permit tcp host 192.168.1.2 host 192.168.2.4 eq 443

Now, for route map configurations. Both will have the same name but with different sequence numbers. They are by default set to number 10. One of them needs to be connected to access-list 101 and interface G0/1 - the destination interface. The other one will be connected to access-list 102 and interface Null0 as destination. The configuration is shown here:

route map POLICY
match ip address 101
set interface G0/1
exit
route map POLICY 20
match ip address 102

set interface Null0

exit

interface G0/0

ip policy route-map POLICY

All router commands not shown are in the command table above.

Instructions to install Apache HTTP on VM:

Now, on the Linux VMs, we need to install HTTP and HTTPS. HTTP is usually installed before HTTPS. To install HTTP, open the terminal on the Linux VM and type the following commands, one after another: **sudo apt-get update** and then **sudo apt-get install apache2**. These two commands install apache, which is a web server that allows you to run HTTP and HTTPS.

Instructions to install HTTPS on VM:

Next, to install HTTPS, one must take these steps:

1. First, if for some reason, SSL wasn't installed along with the Apache installation, then install SSL. It should be installed with Apache.
2. With SSL installed, we must enable the module by typing: **sudo a2enmod ssl**.
3. To see this change in effect, Apache must be restarted by typing the command: **sudo service apache2 restart**.
4. Next we must create a self-signed SSL certificate. Begin by creating a subdirectory within Apache's file system hierarchy by using this command: **sudo mkdir /etc/apache2/ssl**.
5. Then we must create the key and certificate, which can be done with this one command: **sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt**. You will be prompted with questions. The most important one to answer is "Common Name (e.g. server FQDN or

YOUR name)". Here you either enter your domain name, or the server's public ip address. Here is the breakdown of the command shown above:

a. **"req"** - This is a subcommand for X.509 certificate signing request management. X.509 is a public key that allows for self-signing certificates, and we would like to create a X.509 certificate.
b. **"-x509"** - This allows us to create a self-signed certificate rather than create a certificate request.
c. **"-nodes"** - This allows us to not have a password for our Apache server. This will save time, as we won't have to enter the password every time we start the apache server.
d. **"-days 365"** - This validates the certificate for one year.
e. **"-newkey rsa:2048"** - This will create a private key for the certificate, specifically an RSA key that is 2048 bits long.
f. **"-keyout"** - This names the output file for the private key file being generated.
g. **"-out"** - This is the name of the output file for the certificate we are generating.

6. After creating the key and certificate, it is time to set these files up for the purpose of HTTPS. This will be done on the "default-ssl.conf" file, which will already have some default SSL configurations. We open the file by using this command: **sudo nano /etc/apache2/sites-available/default-ssl.conf**.
7. Next we will edit the file. After opening it, it should look something like this:

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                        SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                        SSLOptions +StdEnvVars
        </Directory>
        BrowserMatch "MSIE [2-6]" \
                        nokeepalive ssl-unclean-shutdown \
                        downgrade-1.0 force-response-1.0
        BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
    </VirtualHost>
</IfModule>
```

We must make the following changes to the lines “SSLCertificateFile” and “SSLCertificateKeyFile”:

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLCertificateFile /etc/apache2/ssl/apache.crt
        SSLCertificateKeyFile /etc/apache2/ssl/apache.key
<FilesMatch "\.(cgi|shtml|phtml|php)$">
                        SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                        SSLOptions +StdEnvVars
        </Directory>
        BrowserMatch "MSIE [2-6]" \
                        nokeepalive ssl-unclean-shutdown \
                        downgrade-1.0 force-response-1.0
        BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
    </VirtualHost>
</IfModule>
```

After making these changes, save and exit the file.

8. After configuring the SSL certificate, we must enable it using the command: **sudo a2ensite default-ssl.conf**. Finally, to see the effect of

HTTPS in action, restart the apache service using the command: **sudo apache2 service restart**.

Problems

We faced many minor problems during this lab. The one that occurred the most though was the installation of HTTPS on the Linux VMs. We didn't realize that we had to create a certificate and key for HTTPs to work, but after some research, we figured it out, and got HTTPS to work. Another problem was wrong ACL statements, because at first, we didn't realize we had to permit everything, which is very important, because route maps will basically handle the "denying" of the packets.

Conclusion

In conclusion, we successfully completed policy based routing, and learned a new way to route and filter traffic on the network. Before, we only knew how to use ACLs to deny and permit certain types of traffic, but now with route maps, we have even more flexibility in creating the type of routing we want to do. Using PBR, we can explicitly direct traffic to where we want it to go, which normal ACLs and basic routing, doesn't give us access to.