Policy-Based Routing

-Tanishk Singh

Purpose

The motive behind performing the *Policy-Based Routing* Lab was to restrict and permit data in our LAN on the basis of the concept that the lab is named on. We would have to orchestrate the lab by merging the *access-lists* to the *route-maps* to direct data. On top of all that, the lab would educate us with the basic idea of setting up *HTTPS* and *HTTP* web services on the Linux machines as we would determine the permission or denial of access of those two services from an end host in a different network, using policy-based routing.

Background Information

Before we dive into the definition and methodology of policy-based routing, I would like to explain the terms that we would handle during the course of this lab. Let us begin with Access Control Lists (ACLs). Access-lists or Access Control lists are organized using a few commands on the nodes. They provide a means to filter packets by allowing a user to permit or deny IP packets from crossing specified interfaces. There are two common types of ACLs used in the world: *Standard ACLs* and *Extended ACLs.* Standard ACLs rages from 1 through 99, and they judge the traffic just through the IP addresses mentioned in the command. The Extended ACLs however, ranges from 100 to 199, and they do not only judge the traffic solely on the IP addresses, but also filter the traffic through networking protocols given in the command. For example, Standard ACLs can either deny or permit all the traffic originating from a host 192.168.1.2 to reach a network 192.168.2.0, whereas, Extended ACLs can deny or permit the access to the 192.168.2.0 from the host 192.168.1.2, based on a specific protocol (like https). The ACLs, when configured in the global-configuration mode of the router, must then be bound to an interface. Instead of binding it to an interface, we would bind the ACLs to route-maps.

Route maps are very similar to the ACLs, but they have a few enhanced capabilities: -

1. Route maps can modify certain fields in the packet.
2. Route maps can forward packets in a specified manner.
3. Route maps can Filter and modify the attributes of a route.

Route maps can have a sequence of statements. Each entry in a route map statement contains a combination of match and set commands. The route map statements also have a permit or deny action. For PBR, the match command defines the criteria for matching the packets based on the defined policy. The set command defines the action to be taken on the matched packets. The action could either be modifying or forwarding the packet, bypassing the normal routing based on the IP routing table. We would use Extended ACLs and Route maps to filter HTTP and HTTPS traffic.

HTTP stands for *Hypertext Transfer Protocol* and it allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a computer network and is widely used on the Internet. In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS), or, formerly, its predecessor, Secure Sockets Layer (SSL). The protocol is therefore also often referred to as HTTP over TLS, or HTTP over SSL. The principal motivation for HTTPS is authentication of the accessed website and protection of the privacy and integrity of the exchanged data while in transit.

Policy-Based Routing is configured using the ACLs and route maps, and it provides a network engineer with the ability to dictate the routing behavior based on several different criteria other than destination network, including source or destination network, source or destination address, source or destination port, protocol, packet size, and packet classification among others.

Lab Summary

We commenced the lab with successfully creating a topology as per our teacher’s instructions. We used VMWare to install two Linux servers on a PC. The first step to configuring any data related network lab is to assign IP Addresses to the end hosts and the nodes. We were supposed to operate DHCP, due to which, we created DHCP pools on the router. After ensuring that the PCs have received an address from the router, we began installing HTTP and HTTPS on both servers. We, then, began configuring the router, where we configured the ACLs and bound them to a route map. After configuring the router, we made sure if the requirements for a successful completion of the lab, which was to make an HTTP connection and not being able to establish an HTTPS connection on a server and vice-versa on another server from the isolated end host.

Topology and IP Addressing Scheme

A screenshot of a video game

Description generated with high confidence

Configurations

Step 1: -

A close up of a logo

Description generated with high confidence

Open the application, VMWare, and create two new Linux 18.04 virtual machines. Use automatic installation. After the installation is complete, set the IP address to automatic, and bridge the connection of the VMs with the end PC.

Step 2: -

Configure the IP Addressing in the router according to the topology and start configuring DHCP. Create 2 DHCP pools for the two networks in the topology, using the command: ip dhcp pool LOCAL-HOST and ip dhcp pool HTTP/HTTPS, under the global config mode. Under the DHCP config mode, then, enter the network 192.168.1.0 255.255.255.0 or network 192.168.2.0 255.255.255.0 command, depending on which DHCP pool the network engineer is configuring. State the default gateway for the hosts obtaining the IP address from the router by entering the command, default-router 192.168.1.1 or default-router 192.168.2.1.

Step 3: -

Move onto the Linux PCs and open the terminal to install HTTP and HTTPS. Installing HTTP is a prerequisite for installing HTTPS. HTTP installation requires just two commands to be executed on the terminal: sudo apt-get update and sudo apt-get install apache2. The second command installs *apache* on the VM. Apache is a free and open-source cross-platform web server software, released under the terms of Apache License 2.0.

Step 4: -

The network engineer must then install the SSL module on the Linux machine. SSL support actually comes standard in the Ubuntu 14.04 Apache package. We simply need to enable it to take advantage of SSL on our system. One can enable the module by typing: sudo a2enmod ssl. After the engineer has enabled SSL, he/she will have to restart the web server for the change to be recognized by using this command: sudo service apache2 restart

Step 5: -

This step involves with the action to creating a self-signed SSL Certificate. The engineer will have to begin with creating a subdirectory within Apache's configuration hierarchy to place the certificate files that he/she will be making, using the command: sudo mkdir /etc/apache2/ssl. Now that a location has been decided to place the key and certificate, the network engineer cane create them both in one step by entering: sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt. Let us break it sown and know what this command means: -

* openssl: This is the basic command line tool provided by OpenSSL to create and manage certificates, keys, signing requests, etc.
* req: This specifies a subcommand for X.509 certificate signing request (CSR) management. X.509 is a public key infrastructure standard that SSL adheres to for its key and certificate managment. Since we are wanting to create a new X.509 certificate, this is what we want.
* -x509: This option specifies that we want to make a self-signed certificate file instead of generating a certificate request.
* -nodes: This option tells OpenSSL that we do not wish to secure our key file with a passphrase. Having a password protected key file would get in the way of Apache starting automatically as we would have to enter the password every time the service restarts.
* -days 365: This specifies that the certificate we are creating will be valid for one year.
* -newkey rsa:2048: This option will create the certificate request and a new private key at the same time. This is necessary since we didn't create a private key in advance. The rsa:2048 tells OpenSSL to generate an RSA key that is 2048 bits long.
* -keyout: This parameter names the output file for the private key file that is being created.
* -out: This option names the output file for the certificate that we are generating.

When the engineer enters the command, he/she will be asked a series of questions. The most important item that is requested is the line that reads "Common Name (e.g. server FQDN or YOUR name)". One should enter the domain name you want to associate with the certificate, or the server's public IP address if you do not have a domain name.

Step 6: -

Now that the engineer has their key and certificate, they can configure to use these files in a virtual host file. They will have to base this configuration on the default-ssl.conf file that contains some default SSL configuration. Open the file with root privileges, using this command: sudo nano /etc/apache2/sites-available/default-ssl.conf.

With the comments removed, the file looks a little like this: -

The engineer must make the following changes in the file in order to progress to the next step: -



The user must save and exit the file after completing this part of the configuration.

Step 7: -

After having configured the SSL-enabled virtual host, the engineer will have to enable it by typing this command: sudo a2ensite default-ssl.conf. Afterwards, the engineer will have to restart Apache to load the new virtual host by using the command: sudo service apache2 restart. At this point, the HTTP and HTTPS server should be configured on one of the two virtual machines. Follow the same procedure on the other virtual machine to have completely configured the servers on the other Linux machine.

Step 8: -

Head back to the router to begin configuring the access-list and after that the route maps. Unlike the usual use of access-lists to permit or deny certain types of data, the engineer must permit all the data so that it does not defeat the purpose of configuring PBR on the router. The engineer must permit the actual data that he/she wants to permit using an access-list and permit the data that he/she want to deny using another access-list. The configuration would look something like this: -

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

Step 9: -

Now comes the part to configure route maps. There will be a total of two route maps configured, but with the same name, but a different sequence number. By default, the route maps are set to sequence 10. One must be matched with the access-list 101 and set on interface G0/1, which is the destination interface. The other route map must be matched with access-list 102 and set on interface Null0. The route maps specify that the any traffic originating with access-list 101 will transfer to interface G0/1 and any traffic originating from access-list 102 will transfer to interface Null0, which does not exist, due to which the packets get dropped. At the end, the policy route map must be bound to the source interface. The configuration should seem a little like this: -

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

The engineer should be done with the Lab at this point.

Router configuration: -

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

Problems: -

This lab was easier than a few of the previous labs that we have had to complete. However, like every other lab, we did face a few problems while performing this lab, even though those problems were minor. The first problem that we faced was during the installation of the HTTPS server on our Linux VM. We performed a typing error during the modification of the SSL certificate file. That problem was however, identified and fixed when we reviewed our screenshots. Another problem that we faced was that the PC on the network 192.168.1.0 was able to connect to both the web services on both the Linux machines. That problem however, disappeared the following day. I suspect that it initially happened because on of the host in the topology had a different IP address than what we thought it was because of DHCP configurations.

Conclusions: -

The lab was successfully completed and the motive of working on this lab was fulfilled. We learnt a new way to filter traffic on our network. Normal routing is only concerned about destination-based forwarding, while PBR gives us more control. We can use PBR to direct traffic down links that are reserved for specific types of traffic or levels of priority. We can use PBR to direct traffic based on the source, not the destination, to steer specific customer traffic down links that match their service level agreement.