

Introduction à la sécurité TP 3.5 - Pare-feu Cisco

Maciej Korczynski & Simon Fernandez

Septembre 2020

Objectives

- Deploy a filtering system on the network to protect your server

This is a group work for a team of 3-4 people.

This lab focuses on access control functionalities of routers, turning them into a secure level 1 border between a company network and the exterior, like Internet, or to separate some internal zones from each other in big private networks. On this border, the traffic may be between a remote user on the internet accessing services in the internal network, or traffic from the internal network, reaching the internet. It can be analyzed and controlled based on some parameters written in access lists.

What you need

Access to FreeBSD machines, as root, that will represent the servers between which we try to control the traffic.

Basic Command Line Interface control of the Cisco routers (see the CISCO document for a quick tutorial on how to use them).

Make sure that the router does not contain previous configurations. Always reset the router before using it. To do this, connect to the router and use the `write erase` and then the `reload` commands (see the CISCO doc)

Some theory

Filtering

Basics

Only authorized senders will be allowed to send messages to machines on the network.

The router examines every packet and determines if it shall forward it or drop it, based on some parameters written in the access lists. For examples, the router can use:

- Physical address of the machine (MAC)
- Source or destination IP
- Protocol (IP, ICMP, TCP, UDP)
- Source or destination ports (for TCP and UDP)

Why filter?

- Source routing et IP spoofing
- Ban malicious websites, to limit intrusion risks
- Only allow some services, in some directions, for some users. For example, ban X11, rsh, NFS, tftp, finger.

Rules

- One ACL rule per interface (`access-group` command) per direction
- Direction can be outgoing traffic (`out`, the default) or incoming traffic (`in`)

- Lists are numbered between 1 and 99 (for regular lists) or 100 to 199 (for extended lists). See details in the following sections
- Allow (**permit**) or deny traffic (**deny**)
- The items on the list are executed one after the other. The analysis stops when one condition matches. If nothing matches, by default, the traffic is denied.
- The rule can specify the protocol to filter (**ip**, **udp**, **tcp**, **icmp**)
- The rule can specify the source and destination ports to filter (**eq/gt/lt/neq**)
- The rule can specify the IP addresses. Be careful, the masks work the opposite way as network mask. For example:

```
129.90.0.0      0.0.255.255      =>    129.90.X.X
129.90.5.3      0.0.0.0        =>    129.90.5.3
0.0.0.0         255.255.255.255    =>    X.X.X.X
```

In this case, the masks identify the bits to filter: - a 0 bit means to check if the value matches - a 1 bit means to not check the value

Access lists

Regular IP access lists

They are numbered between 1 and 99.

They use the source IP of the packets. The syntax is:

```
access-list <list_number> {deny | permit} <sourceIP> [<sourceMask>]
```

The keyword **any** can be used to specify an IP or a mask

Example: This list rejects all traffic coming from a 10.X.X.X IP (1), and accepts the rest (2)

```
(1) access-list 1 deny 10.0.0.0    0.255.255.255
(2) access-list 1 permit any
```

Extended IP access lists

They use source and destination IP, the protocol and the port or the service. They allow finer control over the traffic. They can also handle established traffic. The syntax is:

```
access-list <list_number> {deny | permit} <protocol>
      <sourceIP> <sourceMask>
      <destinationIP> <destinationMask>
      [portSpecifier] [port | service] [established] [log]
```

The keyword **host** is used to describe a single IP, and not use a mask. For example:

```
access-list 101 permit tcp any host 10.1.1.1 eq www
```

This allows all TCP traffic to the traditional HTTP ports (80, 443) from all sources, to the specific 10.1.1.1 web server

At each step, you can type **?** to get the list of possible values for a given field. For example, you can list the protocols with

```
access-list 101 permit ?
```

You can list the port operators with

```
access-list 101 permit tcp any any ?
```

You can list the most used TCP ports with

```
access-list 101 permit tcp any any eq ?
```

You can specify a range of ports, using **range** **x** **y**, or stack multiple ports on the same line, like

```
eq telnet eq ftp eq smtp
```

Example :

```
(1) access-list 101 permit tcp any any established
(2) access-list 101 permit tcp any host 144.254.1.3 eq www
(3) access-list 101 deny ip any any log
    ! Apply this to traffic going into the VLAN 1 interface
    interface fastEthernet 1
    ip access-group 101 in
```

This list accepts:

- (1) All traffic linked to sessions already allowed.
- (2) The HTTP traffic from the outside to the internal 144.254.1.3 webserver
- (3) The rest of the traffic is dropped and logged.

There are other kind of lists, but we will not see them in this lab session. Look at the CISCO documentation for more information.

Remarks:

- UDP and TCP ports are always destination ports. The source port is always chosen at random by the source.
- By default, if no rule matches, the packet is dropped
- Adding a rule using the terminal adds it to the end of the list. You cannot selectively drop a rule, you need to drop the whole table and start again, or save the configuration, edit it remotely and flash it back with a **tftp** server for example.
- The rule order is important. The first match will be used, the rest will not be read. So if you add a **deny all** rule, put it at the end.
- Clean the control list before modification, to make sure you have an empty list to start with. **no access-list 101** deletes list 101 for example.

Testing

Because we do not have the webserver on the FreeBSD machines, we will create small custom services on the different ports.

On the server side:

- Listen to messages on a port: **nc -l <port>**

On the client side:

- Try to create a TCP connection to the server: **telnet <serverIP> <port>**
- Once the connection is created, type a message, it should be printed on the server side

Security policies

There are two main ways to setup firewall rules:

- Explicit deny: allow everything and deny specific things. We need to always watch for new threats and blocklist them.
- Explicit allow: block everything and allow specific things. Slowly open for the specific services that you need.

Explicit allow is way better, even if it is harder to deploy. You need to know each and every service that will use the network, some users may complain when their service does not work, you have to manually allow the services, but no uncontrolled traffic will be allowed, so this is way safer.

What to do during the lab?

To save your configuration, you can deploy a TFTP server (see <http://doc.ubuntu-fr.org/tftpd>), to edit the access lists on the computer and download them to the router after. First try with the basic configuration, after the IP configuration of the different interfaces.

To do:

- Deploy and experiment the filtering rules described before.
- Is there an optimized ordering of the rules in a list?
- Are there limitations? Security breaches?

Bonus - Logging

`syslog` is a logging service on Unix machines and network components. Events generated by some applications are logged on a syslog server. To deploy logging:

First, tell the router to log everything more important than `debugging`:

```
no logging console
logging trap debugging
```

Tell the router how to call itself in the logs, so that the source of the logs can be understandable:

```
logging facility local7
```

Give the IP of the logging server:

```
logging a.b.c.d
```

For all access list rules that you want to monitor, add the `log` keyword at the end of the rule. Be careful, if you log everything, every single packet will be logged... This can be a LOT.

Then, we configure the logging server.

First, fill the `/etc/hosts` file with the IP and names of the network and router:

```
127.0.0.1    localhost
10.12.12.x   my_network
10.12.12.254 local7
```

Edit the `/etc/rsyslog.conf` configuration file: just add a line at the end of the file to identify the source of the log messages

```
local7.debug <tab> (no space) /var/log/cisco7.debug
```

Restart the `syslog` service (`service rsyslog restart`, or `/etc/init.d/rsyslog restart`).

Then, test the logging system by generating traffic that will match a rule with the `log` keyword