Spring 2014
Computer Networks
CMPE323

Laboratory Experiment No. 8: Introduction to Firewalls

Aims and Objectives:

- Introducing stateless network firewalls,
- and stateful firewalls.

Materials Required:

- Network firewalls,
- PCs with Ethernet adapters,
- and straight-through/crossover/rollover cables.

Change Log:

- 22-4-2014: original document – mkhonji.
- 23-4-2014: typo fixes – mkhonji.

1 Introduction

Essentially, network firewalls are devices that reside in the pathways of communication networks and serve as filtering devices that decide which packets are allowed to pass.

Figure 1 shows an example of a case where a firewall is used to filter packets in order to achieve the following scenario:

- An internal network that has strict/regulated access to the Internet. Nodes in the internal network are usually regulated by network firewalls such that the nodes can only access Internet resources if the nodes have initiated the connection. In other words, nodes from the Internet cannot send packets to nodes in the internal network unless the packets were responses to requests that were originated by the nodes in the internal network.

- A demilitarized zone (DMZ) that is directly accessible from the Internet. Unlike nodes in the internal network, nodes in the DMZ can be accessed directly by nodes in the Internet without having DMZ nodes initiate the request first.

- And an external network which is in this case the Internet.

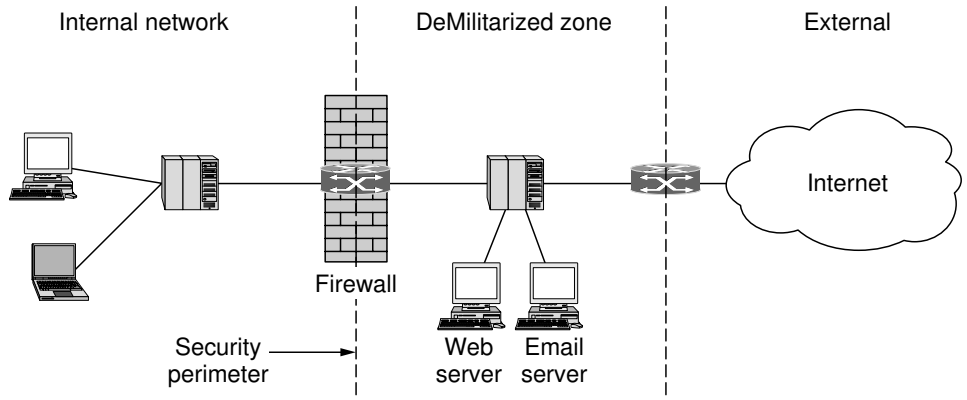

Figure 1: A firewall use case — Source: Computer Networks 5^{th} Edition, 2010, by Andrew S. Tanenbaum and David J. Wetherall, page 818.

The following types of firewalls are common:

- Stateless firewalls — such firewalls are among the oldest and simplest firewall types. They simply analyze packets and if certain patterns are matched, they decide whether to drop or permit them. The patterns that such firewalls analyze are often limited to layer 3 and layer 4 protocol headers (e.g. IP, TCP or UDP headers).

- Stateful firewalls — similar to the stateful firewalls, except that they maintain a states table. Such states table allows the firewall to identify whether a packet is an initial packet or a response to a previous request.

- Application-layer firewalls — similar to the previous types of firewalls, with the exception of being able to not only analyze layer 3 or layer 4 data, but the application layer data as well. For example, by analyzing the application-layer data, it can identify the application (e.g. FTP) even if it uses the port number of another application (e.g. TCP 80, which is for HTTP).

In this lab, we will only configure stateless and stateful firewalls using Cisco routers due to their availability in our lab. This is possible as our Cisco routers are not only routers, but also provide packet filtering capabilities (which is the functionality of a firewall).

2 Lab Preparation

1. Erase the configuration of `R1`[1] and reboot `PC1` and `PC2`.
2. Physically connect and configure IP addresses of the router[2] and the PCs[3] as depicted in Figure 2.

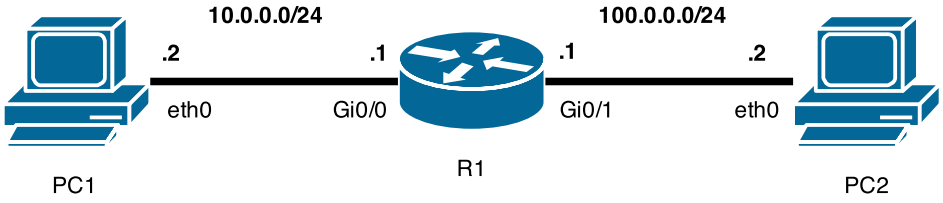

Figure 2: Lab topology.

3. Add default routes[4] to `PC1` and `PC2`.
4. Run Wireshark on all the PCs.
5. Using the `ncat`[5] command run three applications on `PC2` that listen on TCP ports as follows:
   - Application 1: listen on IP 100.0.0.2 and TCP port 100.
   - Application 2: listen on IP 100.0.0.2 and TCP port 200.
6. Using the `ncat`[6] command run three applications on `PC2` that listen on UDP ports as follows:
   - Application 1: listen on IP 100.0.0.2 and UDP port 100.
   - Application 2: listen on IP 100.0.0.2 and UDP port 200.

[1] `enable`, `erase startup-config`, `reload`, and make sure to answer `no` to all yes/no questions while hitting *enter* for all `confirm` prompts.

[2] `enable`, `configure terminal`, `interface Gi0/0`, `ip address <ip_address> <subnet_mask>`, `no shutdown`.

[3] `ifconfig eth0 <ip_address>/<netmask_bits>`

[4] `route add -net 0.0.0.0/0 gw <nexthop_ip>`

[5] `ncat -l -p <port>`

[6] `ncat -l -u -p <port>`

3 Lab experiments

3.1 A stateless firewall

[33 points]

Note: when done, show your work to the lab engineer for grading purposes.

Configure `R1` such that it behaves as a firewall that *only* allows `PC1` to connect to `PC2`'s TCP 100 and UDP 100. In other words, `PC1` can send to `PC2`'s TCP 100 and UDP 100 ports, but `PC2` cannot send to `PC1` (thus the uni-directional aspect).

Connect to `R1` using the console port and perform the following configurations:

1. Create an Access-Control List (ACL) with the name `pc1_to_pc2`[7] that matches packets packets with:

   - Source IP: 10.0.0.2.
   - Destination IP: 100.0.0.2.
   - Source TCP or UDP port: any.
   - Destination TCP or UDP port: 100.

   In order to configure the ACL above, perform the following tasks[8].

   (a) `enable`.

   (b) `config terminal`.

   (c) `ip access-list extended pc1_to_pc2`.

   (d) `permit tcp 10.0.0.2 0.0.0.0 100.0.0.2 0.0.0.0 eq 100`.

   (e) `permit udp 10.0.0.2 0.0.0.0 100.0.0.2 0.0.0.0 eq 100`.

   (f) `deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255. /* this is not mandatory as there is an implicit deny, however it's recommend to add this line for additional clarify as people sometimes forget the implicit deny */`

   (g) `end`.

2. Using the same logic, create *another* Access-Control List (ACL) with the name `pc2_to_pc1` that matches packets packets with:

   - Source IP: 100.0.0.2.
   - Destination IP: 10.0.0.2.

[7] Note that the name can be any arbitrary name, however choosing meaningful names is recommended.

[8] Note that `0.0.0.0` is called a *wild-card mask* which is different than a subnet mask in two ways: 1) matches don't have to be continuous, and 2) a bit of 0 corresponds to a match instead of 1. For example `0.0.0.0 254.254.254.254` matches all IP addresses with even octets, while `1.1.1.1 254.254.254.254` matches all odd IP addresses. Note how the wild-card bits are not continuous and how 0 denotes to a match instead of a 1. The reason why a wild-card mask uses a bit of 0 for matching instead of 1 as in subnet masks is purely historical in the Cisco IOS. More modern designs (e.g. iptables in Linux) don't have this distinction and both use 1s for a match.

- Source TCP or UDP port: 100.
- Destination TCP or UDP port[9]: 49152–65535.

Note that we need to permit a huge range of possible destination port numbers because the client may choose any port among the range of dynamic port numbers 49152–65535. This is a limitation of stateless firewalls. Stateful firewalls (next section) do not have this limitation.

3. Verify the correctness of the contents of your ACLs by the command `show ip access-list`.

4. So far you have created ACLs `pc1_to_pc2` and `pc2_to_pc1`, which are practically useless until you apply them. In this step, we will apply them.

   (a) `enable`.

   (b) `config terminal`.

   (c) `interface gi0/0`.

   (d) `ip access-group pc1_to_pc2 in`.

   (e) `exit`.

   (f) `interface gi0/1`.

   (g) `ip access-group pc2_to_pc1 in`.

   (h) `exit`.

   Note the direction `in`. Generally, it is recommended to drop the packets as early as possible (thus we drop them at the nearest interface to the source in this scenario). However, this is just a recommendation. You can also set it to `out` if you change the interfaces accordingly.

Using PC1 perform the following tests:

- Using PC1, try to connect to PC2's TCP 100. Can you connect and send some data? Show the lab engineer your analysis (using Wireshark) and explain why. In order to perform this test, use the command: `ncat 100.0.0.2 100`.

- Using PC1, try to connect to PC2's UDP 100. Can you connect and send some data? Show the lab engineer your analysis (using Wireshark) and explain why. In order to perform this test, use the command: `ncat -u 100.0.0.2 100`.

- Using PC1, try to connect to PC2's TCP 200. Can you connect and send some data? Show the lab engineer your analysis (using Wireshark) and explain why. In order to perform this test, use the command: `ncat 100.0.0.2 100`.

[9]Use the argument `range 49152 65535` instead of `eq`.

- Using PC1, try to connect to PC2's UDP 200. Can you connect and send some data? Show the lab engineer your analysis (using Wireshark) and explain why. In order to perform this test, use the command: `ncat -u 100.0.0.2 100`.

Using PC2 perform the following tests:

- Using PC2, try to connect to PC1's TCP 100. Can you connect and send some data? Show the lab engineer your analysis (using Wireshark) and explain why. In order to perform this test, use the command: `ncat -p 100 100.0.0.2 100`.
- Using PC2, try to connect to PC1's UDP 100. Can you connect and send some data? Show the lab engineer your analysis (using Wireshark) and explain why. In order to perform this test, use the command: `ncat -p 100 -u 10.0.0.2 100`.

3.2 A notable problem with stateless firewalls

[33 points]

Note: when done, show your work to the lab engineer for grading purposes.

One obvious problem with the stateless firewall earlier was that we had to allow the full dynamic ports range 49152–65535 as the firewall couldn't do anything intelligent to figure it out in a more restrictive manner. So what the stateless firewall did: it allow the full dynamic range.

This creates a security problem as PC2 can connect to PC1 *even* if PC1 did not initiate the connection. For example, perform the following tasks:

- Run a server on PC1 on TCP port 50,000 using the command `ncat -l -p 50000`.
- From PC2, connect to PC1's server using the command: `ncat -p 100 10.0.0.2 50000`

Then answer the following question:

- Does the connection succeed?
- How can you block access from PC2 to any listening port on PC1 without affecting PC2's ability to respond to PC1's requests to TCP/UDP ports 100 on PC2?

3.3 A stateful firewall

[34 points]

Note: when done, show your work to the lab engineer for grading purposes.

This section solves the problem earlier by allowing the firewall to track connections and therefore set initial requests and responses apart from each other.

In order to achieve stateful packet filtering, perform the following tasks:

1. Modify the ACL `pc1_to_pc2` as follows:

(a) Remove all of the rules in the ACL [10]

(b) Add the updated rules with the added argument `reflect connection_states`, where `connection_states` is an arbitrary name that you can set (in this case we set it to connection_states) as follows:

i. `permit tcp 10.0.0.2 0.0.0.0 100.0.0.2 0.0.0.0 eq 100 reflect connection_states`

ii. `permit udp 10.0.0.2 0.0.0.0 100.0.0.2 0.0.0.0 eq 100 reflect connection_states`

iii. `deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255`

2. Modify the ACL `pc2_to_pc1` as follows:

(a) Remove all of the rules in the ACL `pc2_to_pc1`.

(b) Add the following rules

i. `evaluate connection_states`

ii. `deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255`
`/* not mandatory as there is already an implicit deny, but adding it is recommended for clarity only */`

3. Verify the correctness of the contents of your ACLs by the command `show ip access-list`.

Using PC1 perform the following tests:

- Using PC1, try to connect to PC2's TCP 100 using the command: `ncat 100.0.0.2 100`. Does it succeed?
- View the `connection_states` list by `show ip access-list`. What are the added entries? Explain why PC2 is able to respond to PC1.
- Repeat the same for all the other ports: UDP 100, TCP 200, and UDP 200. Then state which ones succeed and why.

Finally, perform the following test:

- Run a server on PC1 on TCP port 50,000 using the command `ncat -l -p 50000`.
- From PC2, connect to PC1's server using the command: `ncat -p 100 10.0.0.2 50000`

Then answer the following question:

- Does the connection succeed?
- Explain why.

[10] `enable`, `config terminal`, `ip access-list extended pc1_to_pc2`, `no <rule or rule_id`.

3.4 Extra questions (not graded)

3.4.1 Question 1

The protocol HTTP uses TCP port 80 according to IANA's. This is why when you type `http://google.com` in your web browser, your web browser will automatically assume that the destination port number is 80.

The question is, how can you block all HTTP traffic when knowing that there are some servers that run on non-standard ports that are possibly assigned to something else?

For example, suppose the servers:

- A HTTP server that listens on the IP address 1.1.1.1, and the TCP port 80.
- A HTTP server that listens on the IP address 2.2.2.2, and the TCP port 22.
- An SSH server that listens on the IP address 3.3.3.3, and the TCP port 22.

3.4.2 Question 2

What if there were hundreds of thousands of HTTP servers that use non-standard ports, how would you match their packets in a way that your ACL remains small in size.