SIO Lab
Packet Filtering Firewalls

version 1.0

Authors: João Paulo Barraca, Hélder Gomes, André Zúquete, Vitor Cunha

Version log:

- 1.0: Initial version

1 Introduction

A firewall is a service allowing to filter the traffic flowing through a host. When using the Linux Operating System, the firewall rules can be implemented using the legacy `iptables` or the new `nftables` frontend. In both cases, the actual firewall is within the `netfilter` kernel framework, allowing detailed control over the packets exchanged. The `netfilter` kernel framework provides both stateless and statefull packet filtering.

2 Setup

In this guide, the objective is to explore the creation and exploitation of a Linux based firewall. For this purpose it is required to create two containers.

- The **FW** will have two network interfaces, one connected to the regular LXD network with Internet access and automatic configuration (DHCP), and another connected to an internal sandbox network (manually configured).
- The **Server** will have a single network interface connected to the sandbox network (manually configured), and will be using the FW as the default route.

```
# Create the sandbox network
vm:~$ lxc network create sandbox ipv4.address=none ipv6.address=none
# Disable IPv6 in the regular LXD network
vm:~$ lxc network set lxdbr0 ipv6.address=none

# Check if you have lxdbr0 and sanbox networks created
vm:~$ lxc network list
+---------+----------+---------+----------------+------+-------------+---------+---------+
|  NAME   |   TYPE   | MANAGED |      IPV4      | IPV6 | DESCRIPTION | USED BY |  STATE  |
+---------+----------+---------+----------------+------+-------------+---------+---------+
| ens33   | physical | NO      |                |      |             | 0       |         |
+---------+----------+---------+----------------+------+-------------+---------+---------+
| lxdbr0  | bridge   | YES     | 10.74.129.1/24 | none |             | 1       | CREATED |
+---------+----------+---------+----------------+------+-------------+---------+---------+
| sandbox | bridge   | YES     | none           | none |             | 1       | CREATED |
+---------+----------+---------+----------------+------+-------------+---------+---------+
```

```
# Create the firewall
vm:~$ lxc launch images:debian/bookworm fw
vm:~$ lxc network attach sandbox fw eth1
vm:~$ lxc shell fw
fw:~$ ip a add 192.168.1.1/30 dev eth1
fw:~$ ip link set dev eth1 up

# Install iptables
fw:~$ apt update
```

```
fw:~$ apt install iptables

# Install a DNS forwarder
fw:~$ apt purge systemd-resolved
fw:~$ apt install dnsmasq

# Install netfilter userspace logging
fw:~$ apt install ulogd2
fw:~$ exit

# Create the server container in the sandbox network
vm:~$ lxc launch images:debian/bookworm server -n sandbox
vm:~$ lxc shell server
server:~$ ip a add 192.168.1.2/30 dev eth0
server:~$ ip r add default via 192.168.1.1
server:~$ resolvectl dns eth0 192.168.1.1
server:~$ exit

# Check status
vm:~$ lxc list
+--------+---------+----------------------+------+-----------+-----------+
|  NAME  |  STATE  |         IPV4         | IPV6 |   TYPE    | SNAPSHOTS |
+--------+---------+----------------------+------+-----------+-----------+
| fw     | RUNNING | 192.168.1.1 (eth1)   |      | CONTAINER | 0         |
|        |         | 10.74.129.117 (eth0) |      |           |           |
+--------+---------+----------------------+------+-----------+-----------+
| server | RUNNING | 192.168.1.2 (eth0)   |      | CONTAINER | 0         |
+--------+---------+----------------------+------+-----------+-----------+

# Route the sandbox network
vm:~$ sudo ip r add 192.168.1.0/30 via <fw_eth0_ip>
```

After all containers are running, the FW should be able to `ping` the Server (address `192.168.1.2`). You VM shoud be able to ping both containers (addresses `192.168.1.1` and `192.168.1.2`). Despite having all routes configured, it is not expected for the Server to reach the remote networks successfully at this point (NAT is missing).

3 Default policies

For each `iptables` chain we should define a default policy, which represents the rule that is applied when no other rule is present. Like in any other security aspect we should choose a defensive approach, denying traffic from potentially malicious sources. In our case we will allow outgoing traffic, generated from the local machine.

For this purpose execute the following commands in the FW:

```
fw:~$ iptables -P INPUT DROP
fw:~$ iptables -P OUTPUT ACCEPT
fw:~$ iptables -P FORWARD DROP
```

You can check the correct application of the rules using the following command:

```
fw:~$ iptables -nvL
```

You can also check that everything is working if you try to `ping` the FW from the Host.

4 Establish basic connectivity

With the policies that are applied, no host can communicate with the external network. Not even the Firewall.

If we wish to allow the `ping` command to be used, we can add a rule for this purpose. If a packet matches this rule, the default policy will not be applied to it and the packet will be handled according to the rule. The following commands insert rules that allow for `ICMP` requests from your VM to the FW to be accepted:

```
fw:~$ iptables -A INPUT -p icmp --icmp echo-request -j ACCEPT
```

Try to `ping` the firewall from your host VM.

What happens if you try to `ping` your host VM from the FW container? Check with tcpdump or Wireshark which packets are exchanged and describe what happens.

Can you create a rule that allows for the correct operation of the `ping` command? Take in consideration that while the availability of this protocol is not mandatory, blocking all `ICMP` packets can have negative consequences.

Note: You can whitelist all traffic originating from your machine easily using the statefull firewalling capabilities. Therefore, if a connection is already established or is know to be related to another connection started from your machine (e.g., FTP data transfer), allow it.

```
fw:~$ iptables -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

The command above uses `-I` (insert) and not `-A` (append). Rules are processed in a sequential manner, so you want to control if the rule is applied at the end of all rules or in any other order. INSERT allows you to put the rule in any line number. If no line number is given, the default is to place it at the fist rule. If you did some mistake, you can replace the `-I` or `-A` by a `-D` to delete a rule.

5 Filter internal traffic

Up to this moment the Server has no connectivity to the outside world, because all packets will be blocked by the rules active in the FW.

In order to allow this connectivity, the FW must enable the following functionality: (i) forward IP traffic, (ii) do not block the forwarded traffic, (iii) apply a Network (Port) Address Translation (NAT) mechanism to the traffic coming from the internal network.

Allowing the traffic to be forwarded requires the creation of a rule matching the traffic from the Server. Considering that the internal interface of the FW is named `eth1`, configured with the address `192.168.1.1`, and the external interface is named `eth0`, the following rule can be used:

```
fw:~$ iptables -A FORWARD -i eth1 -s 192.168.1.0/30 -j ACCEPT
fw:~$ iptables -A FORWARD -o eth1 -d 192.168.1.0/30 -j ACCEPT
```

Check that the Server can access a service in the external network. Use Wireshark in the FW in order to analyze what is happening.

Finally, enabling NAT mechanism will allow the FW to hide the Server from the outside world, while providing connectivity. Without these mechanisms, packets will reach the destination but the destination will be unable to send replies due to the lack of routes.

The Linux OS can do NAT by issuing a rule stating that all packets from the Server network should be masqueraded using the FW external IP address.

```
fw:~$ iptables -t nat -A POSTROUTING -o eth0 -s 192.168.1.0/30 -j MASQUERADE
```

Check that the Server can access a service in the external network.

6 Filter traffic for specific hosts

Frequently, it is required to block the access to a specific host in order to comply with the security policy of the domain. As an example, consider that users should be allowed to access social networks, such as Facebook. A rule blocking access to a host has the following shape:

```
fw:~$ iptables -I FORWARD -d <ip_address> -j DROP
```

Using the `host` command, find all address of `www.facebook.com` and insert the appropriate rules. Then, validate the effectiveness of the setup.

Note that all your firewall's incoming traffic is dropped by default. This holds true for DNS requests. Since when you make a request to `www.facebook.com` (instead of its IP address), your server will make a DNS request to the firewall, which will lead the firewall to make a request to your host, we need to add some new rules:

```
# Allow DNS requests to the firewall
fw:~$ iptables -A INPUT -p udp --dport 53 -m state --state NEW -j ACCEPT
# Allow DNS responses from the host to the firewall
fw:~$ iptables -A INPUT -p udp --sport 53 -m state --state ESTABLISHED -j ACCEPT
```

7 Using DROP vs. REJECT

In the previous situation the decision was to silently drop all packets. However, other decisions can be used, such as REJECT. The different decisions will influence how the packet is handled.

To check this, insert rules to block `www.google.pt` using the REJECT decision.

Access both services and compare the results. Wireshark can also help diagnosing the behavior of the firewall.

8 Traffic logs

In this section we shall test a different decision target which creates records of the network activity. They may be useful to identify the communication endpoints, detecting unusual patterns or communication activity. It should be noticed that this will only allow detection after the communication actually took place, and will not allow blocking the offending exchange.

One possible approach is to configure `iptables` so that some packets, both from successful connections and rejected connections are logged to a registry.

Considering the next command, if the destination address (`ip_address`) is the same as the one used in the previews rules, this will log all dropped packets. Using `-A` would not work in this case, as the rule would never be reached.

```
fw:~$ iptables -I FORWARD -d <ip_address> -j NFLOG --nflog-prefix "DROP "
```

You can see the logs in `/var/log/ulog/syslogemu.log`.

In order to log all successful TCP connections, we will need to add rules that apply to every new packets from TCP connections that were not dropped. In this case, the rules must be added to the middle of the table, after the rules that drop packets, and before the rules that accept traffic. The following command achieves this when using the appropriate value of `<N>`.

```
fw:~$ iptables -I FORWARD <N> -d <ip_address> -p tcp -m state --state NEW -j NFLOG --nflog-prefix "TCP NEW "
```

To list all rules and find the appropriate place to inject the previous rule, you can use the following command:

```
fw:~$ iptables -L
```

As a curiosity, if we wanted to apply a rule to the remaining TCP packets we could use the following match:

```
fw:~$ iptables -I FORWARD -m state -p tcp --state RELATED,ESTABLISHED -j NFLOG --nflog-prefix "TCP CONTENT "
```

9 Filter traffic from specific services

Frequently it is required to allow only a set of services over well known ports, blocking all remaining traffic. One practical example would be to allow only DNS and HTTPS traffic, which can be implemented using the following rules.

```
fw:~$ iptables -I FORWARD -s 192.168.1.0/30 -p udp ! --dport 53 -j DROP
fw:~$ iptables -I FORWARD -s 192.168.1.0/30 -p tcp ! --dport 443 -j DROP
```

Check the command is applied correctly by accessing a page that uses HTTP (port 80), and a page that uses HTTPS (port 443). Remove the previous block using the following commands:

```
fw:~$ iptables -D FORWARD -s 192.168.1.0/30 -p udp ! --dport 53 -j DROP
fw:~$ iptables -D FORWARD -s 192.168.1.0/30 -p tcp ! --dport 443 -j DROP
```

10 Forward traffic from the outside to internal services

Because we are using NAT, it is not possible for external hosts to access services provided by internal servers (in the Server). Additional rules can be added in order to implement the adequate *port forwarding* mechanisms.

Let us start by install the SSH server in the server container:

```
# Install openSSH server
server:~$ apt install openssh-server

# Set the root password
server:~$ passwd

# Edit /etc/ssh/sshd_config to allow password logins
PermitRootLogin yes
PasswordAuthentication yes

# Restart SSH server
systemctl restart ssh
```

Assuming that the SSH server is already running and configured properly, we can create a rule that forwards connections to this server. Specifically we will forward TCP packets, to port 22 of the external interface of the FW, to port 22 of the Server.

This can be implemented by the following command:

```
fw:~$ iptables -t nat -A PREROUTING -p tcp --dport 22 -d X.X.X.X -j DNAT --to 192.168.1.2
```

where `x.x.x.x` represents the external IP address of the FW.

You can use your VM to check if the service is available. Test if you can SSH from your VM to the Server using the newly set root credentials.

!! Warning !!: Do not use the root account in the wild like it is configured here. We are deliberately creating insecure services to exemplify how defense mechanisms work in different scenarios.

11 Save and restore `iptables` rules

The rules added to `iptables` are always temporary, and will be cleared if the host reboots, or the `-F` argument is specified to the command.

Therefore, it is important to save rules to a file, restore these rules at a later time.

The following commands will save the rules to a file named `/etc/iptables.save`, clear the tables, list the content (they should be empty), and then restore the rules:

```
fw:~$ iptables-save > /etc/iptables.save
fw:~$ iptables -F
fw:~$ iptables -t nat -F
fw:~$ iptables -L
fw:~$ iptables-restore < /etc/iptables.save
```

You can edit the file `/etc/iptables.save` and see how rules are saved. If required, you can also edit the rules.

12 Dynamic Host Firewall

While the rules in the FW are able to apply several restrictions to traffic, they are unable to react to all attacks targeting the services exposed. One important situation that must be addressed is the exposition of the SSH service. Once a server exposes this port to the Internet, it will receive tens or hundreds of login attempts per hour. The attempts will come from human attackers, but also from programs doing large scale service discovery, login attempts with dictionary attacks. It is very important to block offending addresses, after they reach a determined number of failed connection attempts. Because this type of behavior depends on the software running on the Server, it is more practical to deploy such filtering at the Server, in the form of a dynamic set of rules.

For this purpose, install the `fail2ban` package. Then, edit `/etc/fail2ban/jails.conf` and enable several jails there related to `pam` and `ssh`.

Restart the `fail2ban` service, list the rules that are automatically created, and the jails:

```
fw:~$ service fail2ban restart
fw:~$ iptables -L
fw:~$ fail2ban-client status
fw:~$ fail2ban-client status ssh
```

Now try to access the SSH service multiple times, failing the username or password. After some tries you should be blocked. List the existing rules and the status of the jails and you should see your IP address listed.

13 Acknowledgements

Authored by João Paulo Barraca, André Zúquete, Hélder Gomes, and Vitor Cunha

14 Bibliography

- Network Address Translation (NAT)
- iptables
- Fail2ban:w :q :q!