Chapter 14. iptables firewall

Table of Contents

This chapter introduces some simple firewall rules and how to configure them with **iptables**.

iptables is an application that allows a user to configure the firewall functionality built into the **Linux** kernel.

iptables tables

By default there are three **tables** in the kernel that contain sets of rules.

The **filter table** is used for packet filtering.

```
root@debian6~# iptables -t filter -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

The **nat table** is used for address translation.

```
root@debian6~# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

The **mangle table** can be used for special-purpose processing of packets.

Series of rules in each table are called a **chain**. We will discuss chains and the nat table later in this chapter.

starting and stopping iptables

The following screenshot shows how to stop and start **iptables** on Red Hat/Fedora/CentOS and compatible distributions.

```
[root@centos6 ~]# service iptables stop
[root@centos6 ~]# service iptables start
iptables: Applying firewall rules                          [  ok  ]
[root@centos6 ~]#
```

Debian and *buntu distributions do not have this script, but allow for an uninstall.

```
root@debian6~# aptitude purge iptables
```

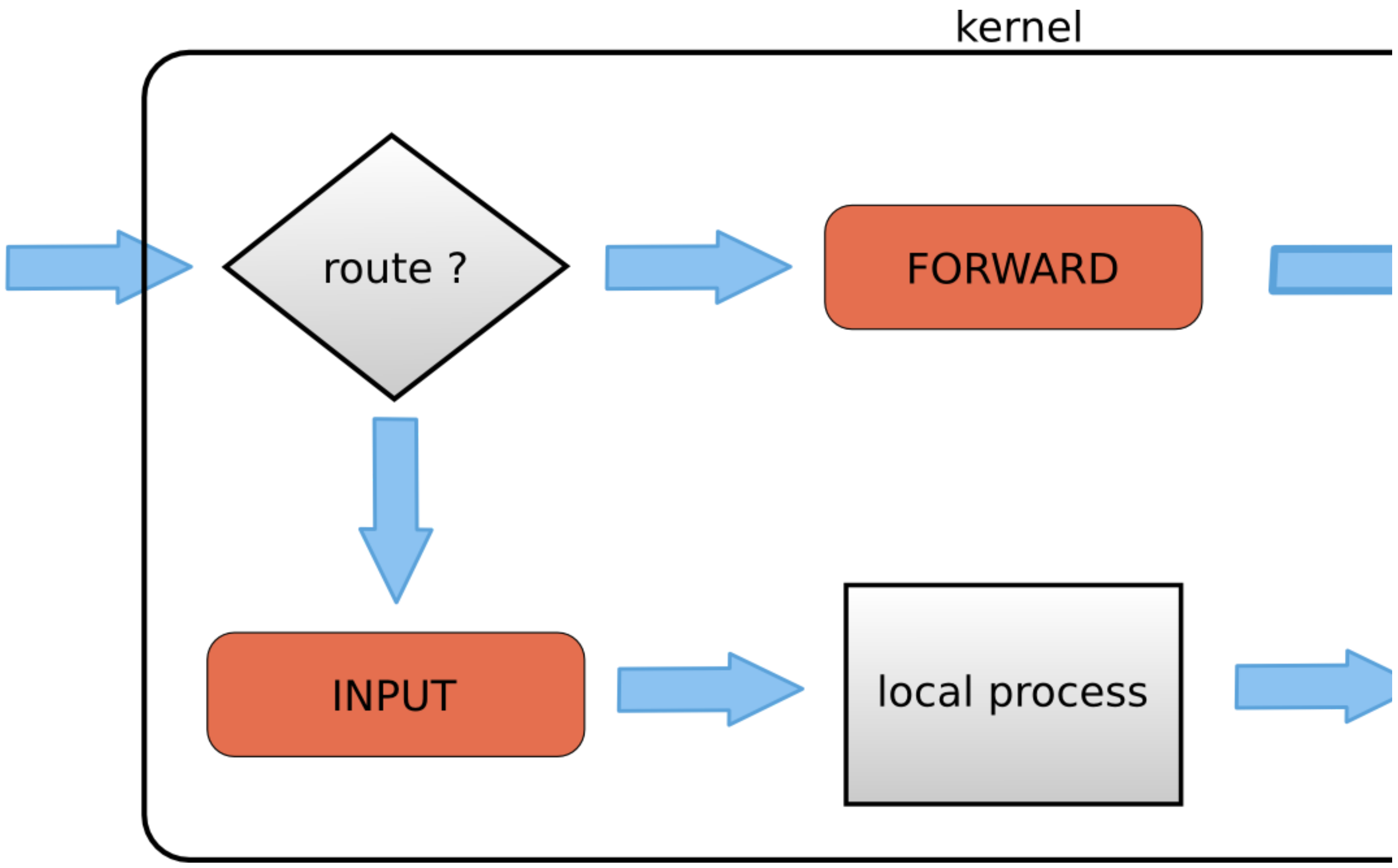
the filter table

about packet filtering

Packet filtering is a bit more than **packet forwarding**. While **packet forwarding** uses only a routing table to make decisions, **packet filtering** also uses a list of rules. The kernel will inspect packets and decide based on these rules what to do with each packet.

filter table

The filter table in **iptables** has three chains (sets of rules). The INPUT chain is used for any packet coming into the system. The OUTPUT chain is for any packet leaving the system. And the FORWARD chain is for packets that are forwarded (routed) through the system.



The screenshot below shows how to list the filter table and all its rules.

```
[root@RHEL5 ~]# iptables -t filter -nL
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[root@RHEL5 ~]#
```

As you can see, all three chains in the filter table are set to ACCEPT everything. ACCEPT is the default behaviour.

setting default rules

The default for the default rule is indeed to ACCEPT everything. This is not the most secure firewall.

A more secure setup would be to DROP everything. A package that is **dropped** will not continue in any chain, and no warning or error will be sent anywhere.

The below commands lock down a computer. Do not execute these commands inside a remote ssh shell.

```
root@debianpaul~# iptables -P INPUT DROP
root@debianpaul~# iptables -P OUTPUT DROP
root@debianpaul~# iptables -P FORWARD DROP
root@debianpaul~# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

changing policy rules

To start, let's set the default policy for all three chains to drop everything. Note that you might lose your connection when typing this over ssh ;-).

```
[root@RHEL5 ~]# iptables -P INPUT DROP
[root@RHEL5 ~]# iptables -P FORWARD DROP
[root@RHEL5 ~]# iptables -P OUTPUT DROP
```

Next, we allow the server to use its own loopback device (this allows the server to access its services running on localhost). We first append a rule to the INPUT chain to allow (ACCEPT) traffic from the lo (loopback) interface, then we do the same to allow packets to leave the system through the loopback interface.

```
[root@RHEL5 ~]# iptables -A INPUT -i lo -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o lo -j ACCEPT
```

Looking at the filter table again (omitting -t filter because it is the default table).

```
[root@RHEL5 ~]# iptables -nL
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
```

Allowing ssh over eth0

This example show how to add two rules to allow ssh access to your system from outside.

```
[root@RHEL5 ~]# iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
```

The filter table will look something like this screenshot (note that -v is added for more verbose output).

```
[root@RHEL5 ~]# iptables -nvL
Chain INPUT (policy DROP 7 packets, 609 bytes)
 pkts bytes target prot opt in    out   source     destination
    0     0 ACCEPT all  --  lo    *     0.0.0.0/0  0.0.0.0/0
    0     0 ACCEPT tcp  --  eth0  *     0.0.0.0/0  0.0.0.0/0  tcp dpt:22

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target prot opt in    out   source     destination

Chain OUTPUT (policy DROP 3 packets, 228 bytes)
 pkts bytes target prot opt in    out   source     destination
    0     0 ACCEPT all  --  *     lo    0.0.0.0/0  0.0.0.0/0
    0     0 ACCEPT tcp  --  *     eth0  0.0.0.0/0  0.0.0.0/0  tcp spt:22
[root@RHEL5 ~]#
```

Allowing access from a subnet

This example shows how to allow access from any computer in the 10.1.1.0/24 network, but only through eth1. There is no port (application) limitation here.

```
[root@RHEL5 ~]# iptables -A INPUT -i eth1 -s 10.1.1.0/24 -p tcp -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o eth1 -d 10.1.1.0/24 -p tcp -j ACCEPT
```

Together with the previous examples, the policy is expanding.

```
[root@RHEL5 ~]# iptables -nvL
Chain INPUT (policy DROP 7 packets, 609 bytes)
 pkts bytes target prot opt in    out   source      destination
    0     0 ACCEPT all  --  lo    *     0.0.0.0/0   0.0.0.0/0
    0     0 ACCEPT tcp  --  eth0  *     0.0.0.0/0   0.0.0.0/0  tcp dpt:22
    0     0 ACCEPT tcp  --  eth1  *     10.1.1.0/24 0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target prot opt in    out   source      destination

Chain OUTPUT (policy DROP 3 packets, 228 bytes)
 pkts bytes target prot opt in    out   source      destination
    0     0 ACCEPT all  --  *     lo    0.0.0.0/0   0.0.0.0/0
    0     0 ACCEPT tcp  --  *     eth0  0.0.0.0/0   0.0.0.0/0  tcp spt:22
    0     0 ACCEPT tcp  --  *     eth1  0.0.0.0/0   10.1.1.0/24
```

iptables save

Use **iptables save** to automatically implement these rules when the firewall is (re)started.

```
[root@RHEL5 ~]# /etc/init.d/iptables save
Saving firewall rules to /etc/sysconfig/iptables:          [  OK  ]
[root@RHEL5 ~]#
```

scripting example

You can write a simple script for these rules. Below is an example script that implements the firewall rules that you saw before in this chapter.

```
#!/bin/bash
# first cleanup everything
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X

# default drop
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# allow loopback device
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# allow ssh over eth0 from outside to system
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT

# allow any traffic from 10.1.1.0/24 to system
iptables -A INPUT -i eth1 -s 10.1.1.0/24 -p tcp -j ACCEPT
iptables -A OUTPUT -o eth1 -d 10.1.1.0/24 -p tcp -j ACCEPT
```

Allowing ICMP(ping)

When you enable iptables, you will get an **'Operation not permitted'** message when trying to ping other hosts.

```
[root@RHEL5 ~# ping 192.168.187.130
PING 192.168.187.130 (192.168.187.130) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

The screenshot below shows you how to setup iptables to allow a ping from or to your machine.

```
[root@RHEL5 ~]# iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -p icmp --icmp-type any -j ACCEPT
```

The previous two lines do not allow other computers to route ping messages through your router, because it only handles INPUT and OUTPUT. For routing of ping, you will need to enable it on the FORWARD chain. The following command enables routing of icmp messages between networks.

```
[root@RHEL5 ~]# iptables -A FORWARD -p icmp --icmp-type any -j ACCEPT
```

practice: packet filtering

1. Make sure you can ssh to your router-system when iptables is active.

2. Make sure you can ping to your router-system when iptables is active.

3. Define one of your networks as 'internal' and the other as 'external'. Configure the router to allow visits to a website (http) to go from the internal network to the external network (but not in the other direction).

4. Make sure the internal network can ssh to the external, but not the other way around.

solution: packet filtering

A possible solution, where leftnet is the internal and rightnet is the external network.

```
#!/bin/bash

# first cleanup everything
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X

# default drop
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# allow loopback device
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# question 1: allow ssh over eth0
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT

# question 2: Allow icmp(ping) anywhere
iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type any -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type any -j ACCEPT

# question 3: allow http from internal(leftnet) to external(rightnet)
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 80 -j ACCEPT
```

```
# question 4: allow ssh from internal(leftnet) to external(rightnet)
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 22 -j ACCEPT

# allow http from external(rightnet) to internal(leftnet)
# iptables -A FORWARD -i eth2 -o eth1 -p tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth2 -p tcp --sport 80 -j ACCEPT

# allow rpcinfo over eth0 from outside to system
# iptables -A INPUT -i eth2 -p tcp --dport 111 -j ACCEPT
# iptables -A OUTPUT -o eth2 -p tcp --sport 111 -j ACCEPT
```

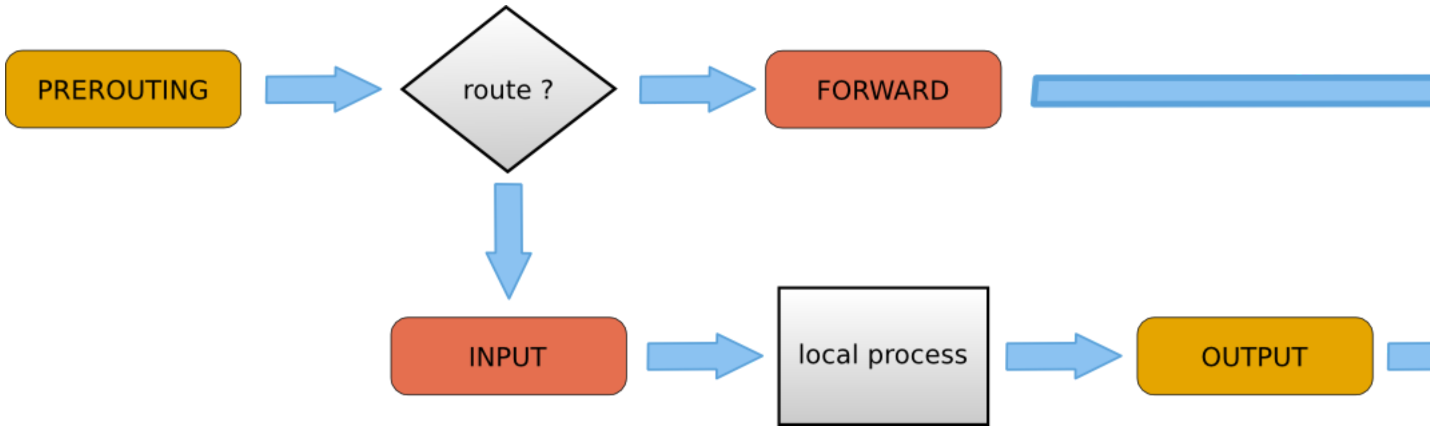
network address translation

about NAT

A NAT device is a router that is also changing the source and/or target ip-address in packets. It is typically used to connect multiple computers in a private address range with the (public) internet. A NAT can hide private addresses from the internet.

NAT was developed to mitigate the use of real ip addresses, to allow private address ranges to reach the internet and back, and to not disclose details about internal networks to the outside.

The nat table in iptables adds two new chains. PREROUTING allows altering of packets before they reach the INPUT chain. POSTROUTING allows altering packets after they exit the OUTPUT chain.



Use **iptables -t nat -nvL** to look at the NAT table. The screenshot below shows an empty NAT table.

```
[root@RHEL5 ~]# iptables -t nat -nL
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[root@RHEL5 ~]#
```

SNAT (Source NAT)

The goal of source nat is to change the source address inside a packet before it leaves the system (e.g. to the internet). The destination will return the packet to the NAT-device. This means our NAT-device will need to keep a table in memory of all the packets it changed, so it can deliver the packet to the original source (e.g. in the private network).

Because SNAT is about packets leaving the system, it uses the POSTROUTING chain.

Here is an example SNAT rule. The rule says that packets coming from 10.1.1.0/24 network and exiting via eth1 will get the source ip-address set to 11.12.13.14. (Note that this is a one line command!)

```
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j SNAT \
--to-source 11.12.13.14
```

Of course there must exist a proper iptables filter setup to allow the packet to traverse from one network to the other.

SNAT example setup

This example script uses a typical nat setup. The internal (eth0) network has access via SNAT to external (eth1) webservers (port 80).

```
#!/bin/bash
#
# iptables script for simple classic nat websurfing
# eth0 is internal network, eth1 is internet
#
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -s 10.1.1.0/24 -p tcp \
--dport 80 -j ACCEPT
```

```
iptables -A FORWARD -i eth1 -o eth0 -d 10.1.1.0/24 -p tcp \
--sport 80 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j SNAT \
--to-source 11.12.13.14
echo 1 > /proc/sys/net/ipv4/ip_forward
```

IP masquerading

IP masquerading is very similar to SNAT, but is meant for dynamic interfaces. Typical example are broadband 'router/modems' connected to the internet and receiving a different ip-address from the isp, each time they are cold-booted.

The only change needed to convert the SNAT script to a masquerading is one line.

```
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j MASQUERADE
```

DNAT (Destination NAT)

DNAT is typically used to allow packets from the internet to be redirected to an internal server (in your DMZ) and in a private address range that is inaccessible directly form the internet.

This example script allows internet users to reach your internal (192.168.1.99) server via ssh (port 22).

```
#!/bin/bash
#
# iptables script for DNAT
# eth0 is internal network, eth1 is internet
#
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -s 10.1.1.0/24 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 22 -j ACCEPT
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 22 \
-j DNAT --to-destination 10.1.1.99
echo 1 > /proc/sys/net/ipv4/ip_forward
```