# iptables

From ArchWiki

Iptables is a powerful firewall built into the Linux kernel and is part of the netfilter project. It can be configured directly, or by using one of the many frontends and GUIs. iptables is used for IPv4 and ip6tables is used for IPv6.

nftables (http://netfilter.org/projects/nftables/) is slated for release with Linux kernel 3.13 (http://www.phoronix.com/scan.php?page=news_item&px=MTQ5MDU), and will be ultimately replacing iptables as the main Linux firewall utility. For now, a howto is available here (https://home.regit.org/netfilter-en/nftables-quick-howto/).

**Related articles**

Firewalls

Simple Stateful Firewall

Sysctl#TCP/IP stack hardening

Sshguard

Fail2ban

## Contents



## Installation

The stock Arch Linux kernel is compiled with iptables support. You will only need to install the userland utilities, which are provided by the package `iptables (https://www.archlinux.org/packages/?name=iptables)` in the official repositories.

## Basic concepts

## Tables

iptables contains five *tables*, which are areas where a chain of rules can apply:

1. `raw` filters packets before any of the other table. It is used mainly for configuring exemptions from connection tracking in combination with the `NOTRACK` target.
2. `filter` is the default table (if no `-t` option is passed).
3. `nat` is used for network address translation (e.g. port forwarding). Because of limitations in iptables, filtering should not be done here.
4. `mangle` is used for specialized packet alteration (see Mangled packet).
5. `security` is used for Mandatory Access Control networking rules.

## Chains

Tables contain *chains*, which are lists of rules for packets that are followed in order. The default table `filter` contains three built-in chains: `INPUT`, `OUTPUT` and `FORWARD`.

1. Inbound traffic addressed to the machine itself hits the `INPUT` chain.
2. Outbound, locally-generated traffic hits the `OUTPUT` chain.
3. Routed traffic which should not be delivered locally hits the `FORWARD` chain.

See `man 8 iptables` for a description of built-in chains in other tables.

User-defined chains can be added to make rulesets more efficient.

Built-in chains have a default target, which is used if no rules are hit. Neither built-in nor user-defined chains can be a default target.

## Rules

The packet filtering is based on *rules*, which are specified by multiple *matches* (conditions the packet must satisfy so that the rule can be applied), and one *target* (action taken when the packet matches all condition). While individual conditions are usually very simple, the full rule specification can be very complex.

Targets are specified using the `-j` or `--jump` option. Targets can be either user-defined chains, one of the special built-in targets, or a target extension. Built-in targets are `ACCEPT`, `DROP`, `QUEUE` and `RETURN`, target extensions are for example `REJECT` and `LOG`. If the target is a built-in target, the fate of the packet is decided immediately and processing of the packet in current table is stopped. If the target is a user-defined chain and the packet passes successfully through this second chain, it will move to the next rule in the original chain. Target extensions can be either *terminating* (as built-in targets) or *non-terminating* (as user-defined chains), see `man 8 iptables-extensions` for details.

## Modules

There are many modules which can be used to extend iptables such as connlimit, conntrack, limit and recent. These modules add extra functionality to allow complex filtering rules.

# Configuration

## From the command line

### Showing the current rules

You can check the current ruleset and the number of hits per rule by using the command:

```
# iptables -nvL
```

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

Chain OUTPUT (policy ACCEPT 0K packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

If the output looks like the above, then there are no rules. Nothing is blocked.

To show the line numbers when listing rules, append `--line-numbers` to that input. This is useful when deleting and adding individual rules.

### Editing rules

Rules can be added either by appending a rule to a chain or inserting them at a specific position on the chain. We will explore both methods here.

First of all, our computer is not a router (unless, of course, it is a router). We want to change the default policy on the `FORWARD` chain from `ACCEPT` to `DROP`.

```
# iptables -P FORWARD DROP
```

**Warning:** The rest of this section is meant to teach the syntax and concepts behind iptables rules. It is not intended as a means for securing servers. For improving the security of your system, see Simple Stateful Firewall for a minimally secure iptables configuration and Security for hardening Arch Linux in general.

The Dropbox LAN sync feature broadcasts packets every 30 seconds (https://isc.sans.edu/port.html?port=17500) to all computers it can see. If we happen to be on a LAN with Dropbox clients and do not use this feature, then we might wish to reject those packets.

```
# iptables -A INPUT -p tcp --dport 17500 -j REJECT --reject-with icmp-port-unreachable
```

```
# iptables -nvL --line-numbers
```

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 REJECT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:17500 reje
```

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
```

**Note:** We use `REJECT` rather than `DROP` here, because RFC 1122 3.3.8 (https://tools.ietf.org/html/rfc1122#page-69) requires hosts return ICMP errors whenever possible, instead of dropping packets. In reality, it is best to `REJECT` packets from hosts who should know about your server's existence, and `DROP` packets from hosts who should not even know your server exists, or those who appear "up to something".

Now, say we change our mind about Dropbox and decide to install it on our computer. We also want to LAN sync, but only with one particular IP on our network. So we should use `-R` to replace our old rule. Where `10.0.0.85` is our other IP:

```
# iptables -R INPUT 1 -p tcp --dport 17500 ! -s 10.0.0.85 -j REJECT --reject-with icmp-port-unreachable
```

```
# iptables -nvL --line-numbers
```

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 REJECT     tcp  --  *      *      !10.0.0.85            0.0.0.0/0            tcp dpt:17500 reje

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
```

We have now replaced our original rule with one that allows `10.0.0.85` to access port `17500` on our computer. But now we realize that this is not scalable. If our friendly Dropbox user is attempting to access port `17500` on our device, we should allow him immediately, not test him against any firewall rules that might come afterwards!

So we write a new rule to allow our trusted user immediately. Using `-I` to insert the new rule before our old one:

```
# iptables -I INPUT 1 -p tcp --dport 17500 -s 10.0.0.85 -j ACCEPT -m comment --comment "Friendly Dropbox"
```

```
# iptables -nvL --line-numbers
```

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 ACCEPT     tcp  --  *      *       10.0.0.85            0.0.0.0/0            tcp dpt:17500 /* F
2        0     0 REJECT     tcp  --  *      *      !10.0.0.85            0.0.0.0/0            tcp dpt:17500 reje

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
```

And replace our second rule with one that rejects everything on port `17500`:

```
# iptables -R INPUT 2 -p tcp --dport 17500 -j REJECT --reject-with icmp-port-unreachable
```

Our final rule list now looks like this:

```
# iptables -nvL --line-numbers

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 ACCEPT     tcp  --  *      *       10.0.0.85            0.0.0.0/0            tcp dpt:17500 /* F
2        0     0 REJECT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:17500 reje

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
```

### Resetting rules

You can flush and reset iptables to default using these commands:

```
# iptables -F
# iptables -X
# iptables -t nat -F
# iptables -t nat -X
# iptables -t mangle -F
# iptables -t mangle -X
# iptables -t raw -F
# iptables -t raw -X
# iptables -t security -F
# iptables -t security -X
# iptables -P INPUT ACCEPT
# iptables -P FORWARD ACCEPT
# iptables -P OUTPUT ACCEPT
```

The `-F` command with no arguments flushes all the chains in its current table. Similarly, `-X` deletes all empty non-default chains in a table.

Individual chains may be flushed or deleted by following `-F` and `-X` with a `[chain]` argument.

## Configuration file

Iptables rules are by default stored in `/etc/iptables/iptables.rules`. This file is read by `iptables.service`:

```
# systemctl enable iptables.service
# systemctl start iptables.service
```

Iptables rules for ipv6 are by default stored in `/etc/iptables/ip6tables.rules`, this file is read by `ip6tables.service`. You can start it the same way as above.

After adding rules via command-line, the configuration file is not changed automatically - you have to save it manually:

```
# iptables-save > /etc/iptables/iptables.rules
```

If you edit the configuration file manually, you have to reload it:

```
# systemctl reload iptables
```

### Guides

- Simple stateful firewall
- Router

## Logging

The `LOG` target can be used to log packets that hit a rule. Unlike other targets like `ACCEPT` or `DROP`, the packet will continue moving through the chain after hitting a `LOG` target. This means that in order to enable logging for all dropped packets, you would have to add a duplicate `LOG` rule before each DROP rule. Since this reduces efficiency and makes things less simple, a `logdrop` chain can be created instead.

Create the chain with:

```
# iptables -N logdrop
```

Then define it:

```
## /etc/iptables/iptables.rules

*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]

... other user defined chains ..

## logdrop chain
:logdrop - [0:0]

-A logdrop -m limit --limit 5/m --limit-burst 10 -j LOG
-A logdrop -j DROP

... rules ...

## log AND drop packets that hit this rule:
-A INPUT -m state --state INVALID -j logdrop

... more rules ...
```

### Limiting log rate

The limit module should be used to prevent your iptables log from growing too large or causing needless hard drive writes. Without limiting, an attacker could fill your drive (or at least your `/var` partition) by causing writes to the iptables log.

`-m limit` is used to call on the limit module. You can then use `--limit` to set an average rate and `--limit-burst` to set an initial burst rate. Example:

```
-A logdrop -m limit --limit 5/m --limit-burst 10 -j LOG
```

This appends a rule to the `logdrop` chain which will log all packets that pass through it. The first 10 packets will the be logged, and from then on only 5 packets per minute will be logged. The "limit burst" is restored by one every time the "limit rate" is not broken.

### syslog-ng

Assuming you are using syslog-ng, you can control where iptables' log output goes this way:

```
filter f_everything { level(debug..emerg) and not facility(auth, authpriv); };
```

to

```
filter f_everything { level(debug..emerg) and not facility(auth, authpriv) and not filter(f_iptables); };
```

This will stop logging iptables output to `/var/log/everything.log`.

If you also want iptables to log to a different file than `/var/log/iptables.log`, you can simply change the file value of destination `d_iptables` here (still in `syslog-ng.conf`)

```
destination d_iptables { file("/var/log/iptables.log"); };
```

### ulogd

ulogd (http://www.netfilter.org/projects/ulogd/index.html) is a specialized userspace packet logging daemon for netfilter that can replace the default `LOG` target. The package `ulogd (https://www.archlinux.org/packages/?name=ulogd)` is available in the `[community]` repository.

## See also

See the Wikipedia article on this subject for more information: **iptables**

- Port Knocking
- Official iptables web site (http://www.netfilter.org/projects/iptables/index.html)
- iptables Tutorial 1.2.2 (http://www.frozentux.net/iptables-tutorial/iptables-tutorial.html) by Oskar Andreasson
- iptables Debian (http://wiki.debian.org/iptables) Debian wiki

Retrieved from "https://wiki.archlinux.org/index.php?title=Iptables&oldid=283899"
Category: Firewalls