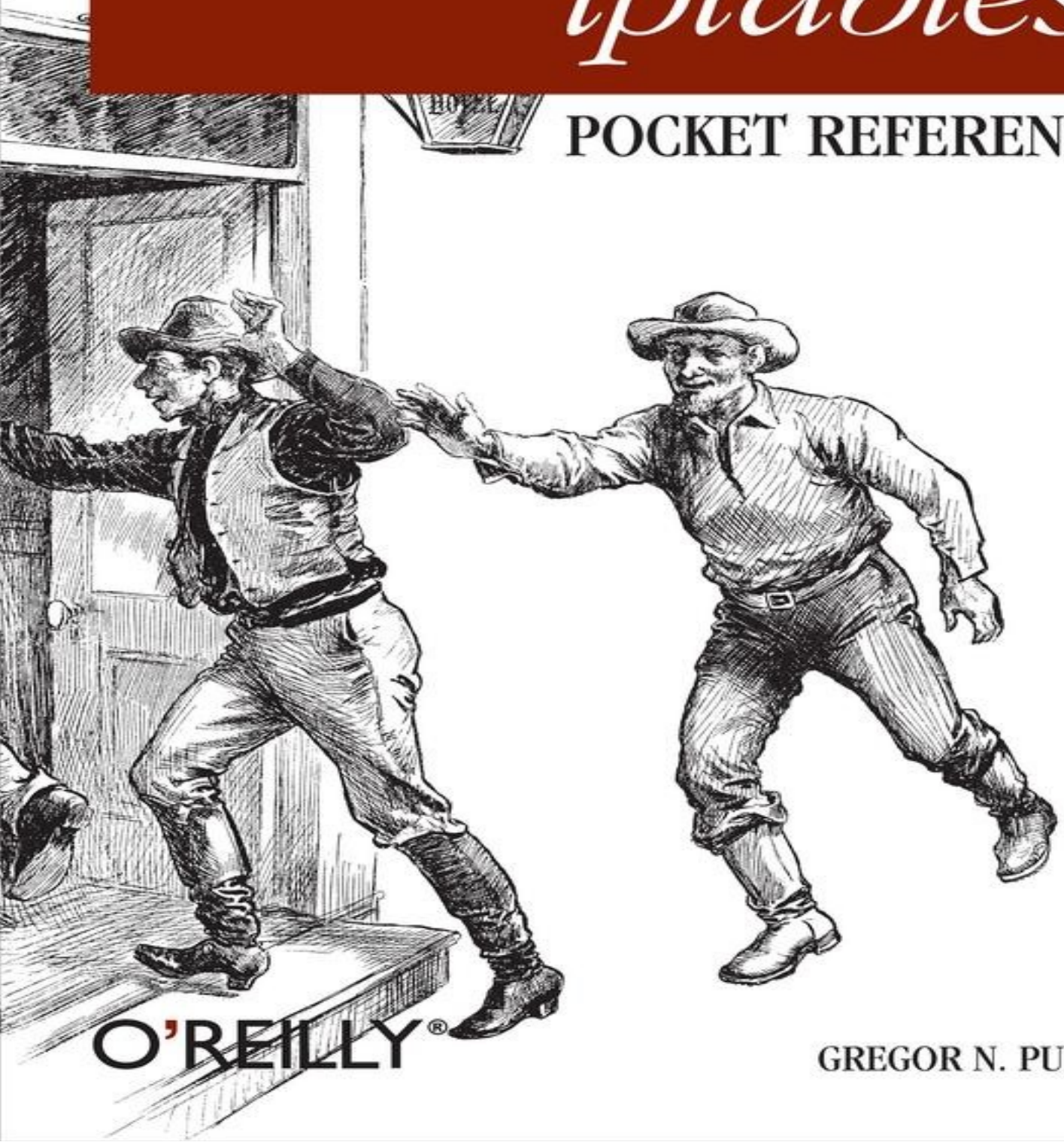
FIREWALLS, NAT & ACCOUNTING

LINUX *iptables*

POCKET REFERENCE



O'REILLY®

GREGOR N. PURDY

Linux iptables Pocket Reference

Gregor N. Purdy

Editor

Andy Oram

O'Reilly Media

Dedication

This book is dedicated to the memory of my brother W. Scott Purdy (1969–1995)

Chapter 1. Linux iptables Pocket Reference

Introduction

The Linux kernel's network packet processing subsystem is called Netfilter, and **iptables** is the command used to configure it. This book covers the **iptables** user-space utilities Version 1.2.7a, whi uses the Netfilter framework in the Linux kernel version 2.4 and also covers most of what's in 2.6. Because Netfilter and **iptables** are tightly coupled, I will use "**iptables**" to refer to either or both of them throughout this book.

The **iptables** architecture groups network packet processing rules into tables by function (packet filtering, network address translation, and other packet mangling), each of which have chains (sequences) of processing rules. Rules consist of matches (used to determine which packets the rule will apply to) and targets (that determine what will be done with the matching packets).

iptables operates at OSI Layer 3 (Network). For OSI Layer 2 (Link), there are other technologies suc as **ebtables** (Ethernet Bridge Tables). See http://ebtables.sourceforge.net/ for more information.

An Example Command

Here is a sample **iptables** command:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80
  -j DNAT --to-destination 192.168.1.3:8080
```

Table 1-1 shows what this sample **iptables** command means.

Table 1-1. Decomposed example iptables command arguments

Component	Description
`-t nat`	Operate on the `nat` table...
`-A PREROUTING`	... by appending the following rule to its `PREROUTING` chain.
`-i eth1`	Match packets coming in on the `eth1` network interface...
`-p tcp`	... that use the `tcp` (TCP/IP) protocol
`--dport 80`	... and are intended for local port `80`.
`-j DNAT`	Jump to the `DNAT` target...
`--to-destination 192.168.1.3:8080`	... and change the destination address to `192.168.1.3` and destination port to `8080`.

Concepts

iptables defines five "hook points" in the kernel's packet processing pathways: `PREROUTING`, `INPUT`, `FORWARD`, `POSTROUTING` and `OUTPUT`. Built-in chains are attached to these hook points; you can add a sequence of rules for each hook point. Each rule represents an opportunity to affect or monitor packet flow.

Tip ✎

It is common to refer to "the `PREROUTING` chain of the `nat` table," which implies that chains belong to tables. However chains and tables are only partially correlated, and neither really "belongs" to the other. *Chains* represent hook points in the packet flow, and *tables* represent the types of processing that can occur. Figure 1-1 through Figure 1-3 show all the legal combinations, and the order in which they are encountered by packets flowing through the system.

Figure 1-1 shows how packets traverse the system for network address translation. These are the chains for the `nat` table.

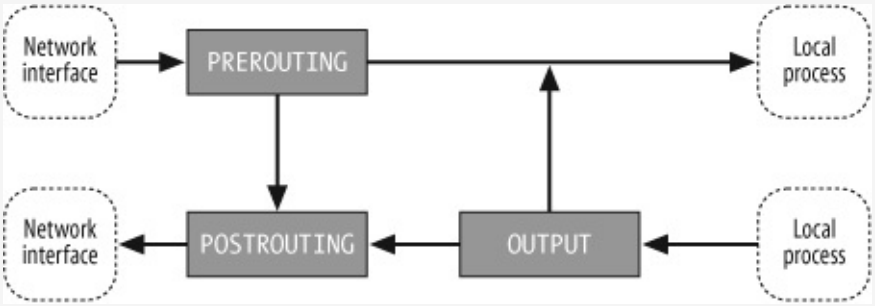


Figure 1-1. Network packet flow and hook points for NAT

Figure 1-2 shows how packets traverse the system for packet filtering. These are the chains for the `filter` table.

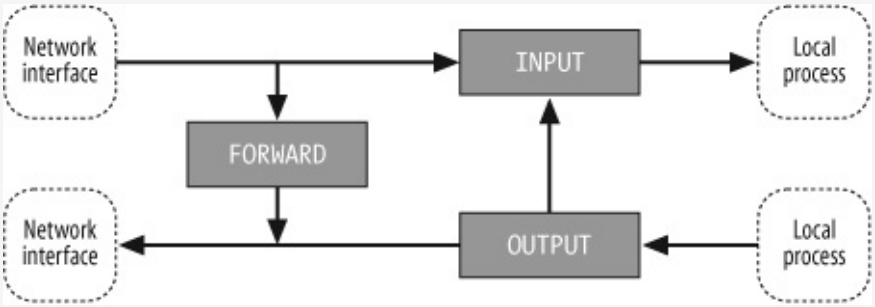


Figure 1-2. Network packet flow and hook points for filtering

Figure 1-3 shows how packets traverse the system for packet mangling. These are the chains for the `mangle` table.

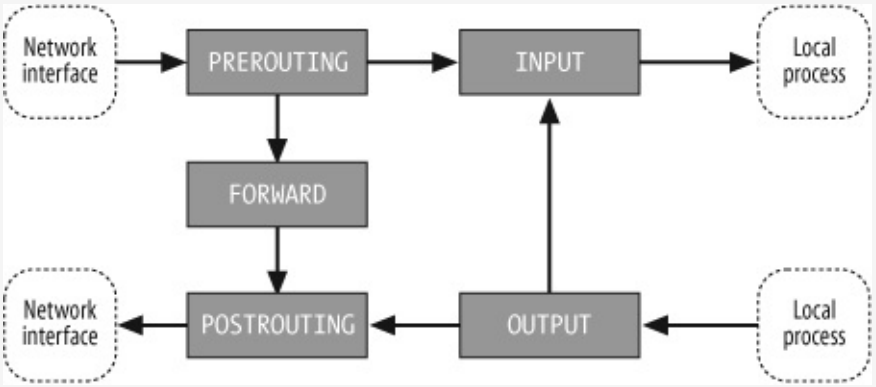


Figure 1-3. Network packet flow and hook points for mangling

Table 1-2 shows the five hook points and describes the points in the packet flow where you can specify processing.

Table 1-2. Hook points

Hook	Allows you to process packets...
`FORWARD`	... that flow through a gateway computer, coming in one interface and going right back out another.
`INPUT`	... just before they are delivered to a local process.
`OUTPUT`	... just after they are generated by a local process.
`POSTROUTING`	... just before they leave a network interface.
`PREROUTING`	... just as they arrive from a network interface (after dropping any packets resulting from the interface being in promiscuous mode and after checksum validation).

Tip ✎

For the curious, the hook points are defined in the kernel header file */usr/include/linux/netfilter_ipv4.h* with names like `NF_IP_FORWARD`, `NF_IP_LOCAL_{IN,OUT}`, and `NF_IP_{PRE,POST}_ROUTING`.

Your choice of chain will be based on where in the packet lifecycle you need to apply your rules. For example, if you want to filter outgoing packets, it is best to do so in the `OUTPUT` chain because the `POSTROUTING` chain is not associated with the `filter` table.

Tables

iptables comes with three built-in tables: `filter`, `mangle`, and `nat`. Each is preconfigured with chains corresponding to one or more of the hook points described in Table 1-2 and shown in Figure 1-1 through Figure 1-3. The three built-in tables are described in Table 1-3.

Table 1-3. Built-in tables

Table	Description

`nat`	Used with connection tracking to redirect connections for network address translation; typically based on source or destination addresses. Its built-in chains are: `OUTPUT`, `POSTROUTING`, and `PREROUTING`.
`filter`	Used to set policies for the type of traffic allowed into, through, and out of the computer. Unless you refer to a different table explicitly, **iptables** operate on chains within this table by default. Its built-in chains are: `FORWARD`, `INPUT`, and `OUTPUT`.
`mangle`	Used for specialized packet alteration, such as stripping off IP options (as with the `IPV4OPTSSTRIP` target extension). Its built in chains are: `FORWARD`, `INPUT`, `OUTPUT`, `POSTROUTING`, and `PREROUTING`.

iptables arranges for the appropriate chains in these tables to be traversed by network packets based on the source and destination, and in the order depicted in Figure 1-1 through Figure 1-3 and detailed in Table 1-4 through Table 1-7.

> **Tip** ✎
>
> The default table is the `filter` table; if you do not specify an explicit table in an **iptables** command, `filter` is assumed.

Chains

By default, each table has chains, which are initially empty, for some or all of the hook points. See Table 1-2 for a list of hook points and Table 1-3 for a list of built-in chains for each table.

In addition, you can create your own custom chains to organize your rules.

A chain's *policy* determines the fate of packets that reach the end of the chain without otherwise being sent to a specific target. Only the built-in targets (see Table 1-8) `ACCEPT` and `DROP` can be used as the policy for a built-in chain, and the default is `ACCEPT`. All user-defined chains have an implicit policy of `RETURN` that cannot be changed.

If you want a more complicated policy for a built-in chain or a policy other than `RETURN` for a user-defined chain, you can add a rule to the end of the chain that matches all packets, with any target you like. You can set the chain's policy to `DROP` in case you make a mistake in your catch-all rule or wish to filter out traffic while you make modifications to your catch-all rule (by deleting it and re-adding it with changes).

Packet flow

Packets traverse chains, and are presented to the chains' rules one at a time in order. If the packet does not match the rule's criteria, the packet moves to the next rule in the chain. If a packet reaches the last rule in a chain and still does not match, the chain's policy (essentially the chain's default target; see the previous section Chains section for more information) is applied to it.

Based on the flow depicted in Figure 1-1 through Figure 1-3, the order in which packets are presented to the built-in tables and chains is shown in Table 1-4 through Table 1-7.

Table 1-4. Packet flows from one network interface to another (forwarding)

Table	Chain
mangle	PREROUTING
nat	PREROUTING
mangle	FORWARD
filter	FORWARD
mangle	POSTROUTING
nat	POSTROUTING

Table 1-5. Packet flows from a network interface to a local process (input)

Table	Chain
mangle	PREROUTING
nat	PREROUTING
mangle	INPUT
filter	INPUT

Table 1-6. Packet flows from a local process to a network interface (output)

Table	Chain
mangle	OUTPUT
nat	OUTPUT
filter	OUTPUT
mangle	POSTROUTING
nat	POSTROUTING

Table 1-7. Packet flows from a local process to another local process (local)

Table	Chain
mangle	OUTPUT
nat	OUTPUT
filter	OUTPUT
filter	INPUT

mangle	INPUT

Rules

An **iptables** rule consists of one or more match criteria that determine which network packets it affects (all match options must be satisfied for the rule to match a packet) and a target specification that determines how the network packets will be affected.

The system maintains packet and byte counters for every rule. Every time a packet reaches a rule and matches the rule's criteria, the packet counter is incremented, and the byte counter is increased by the size of the matching packet.

Both the match and the target portion of the rule are optional. If there are no match criteria, all packets are considered to match. If there is no target specification, nothing is done to the packets (processing proceeds as if the rule did not exist—except that the packet and byte counters are updated). You can add such a null rule to the `FORWARD` chain of the `filter` table with the command:

```
iptables -t filter -A FORWARD
```

Matches

There are a variety of matches available for use with **iptables**, although some are available only for kernels with certain features enabled. Generic Internet Protocol (IP) matches (such as protocol, source, or destination address) are applicable to any IP packet (described in the reference section ip (Internet Protocol IPv4) matches, even though the IP matches are available without referencing any match extension).

In addition to the generic matches, **iptables** includes many specialized matches available through dynamically loaded extensions (use the **iptables** `-m` or `--match` option to inform **iptables** you want to use one of these extensions).

There is one match extension for dealing with a networking layer below the IP layer. The `mac` match extension matches based on Ethernet media access controller (MAC) addresses.

Targets

Targets are used to specify the action to take when a rule matches a packet and also to specify chain policies. Four targets are built into **iptables**, and extension modules provide others. Table 1-8 describes the built-in targets.

Table 1-8. Built-in targets

Target	Description
`ACCEPT`	Let the packet through to the next stage of processing. Stop traversing the current chain, and start at the next stage shown in Figure 1-1 through Figure 1-3 (and Table 1-4 through Table 1-7).
`DROP`	Discontinue processing the packet completely. Do not check it against any other rules, chains, or tables. If you want to provide some feedback to the sender, use the `REJECT` target extension.

QUEUE	Send the packet to userspace (i.e. code not in the kernel). See the *libipq* manpage for more information.
RETURN	From a rule in a user-defined chain, discontinue processing this chain, and resume traversing the calling chain at the rule following the one that had this chain as its target. From a rule in a built-in chain, discontinue processing the packet and apply the chain's policy to it. See the previous section Chains for more information about chain policies.

Applications

The following list provides a brief overview of packet processing techniques and some of their applications:

Packet filtering

Packet filtering is the most basic type of network packet processing. Packet filtering involves examining packets at various points as they move through the kernel's networking code and making decisions about how the packets should be handled (accepted into the next stage of processing, dropped completely without a reply, rejected with a reply, and so on).

Accounting

Accounting involves using byte and/or packet counters associated with packet matching criteria t monitor network traffic volumes.

Connection tracking

Connection tracking provides additional information that can match related packets in ways that are otherwise impossible. For example, FTP (file transfer protocol) sessions can involve two separate connections: one for control and one for data transfer. Connection tracking for FTP monitors the control connection and uses knowledge of the FTP protocol to extract enough information from the control interactions to identify the data connections when they are created. This tracking information is then made available for use by packet processing rules.

Packet mangling

Packet mangling involves making changes to packet header fields (such as network addresses and port numbers) or payloads.

Network address translation (NAT)

Network address translation is a type of packet mangling that involves overwriting the source and/or destination addresses and/or port numbers. Connection tracking information is used to mangle related packets in specific ways. The term "Source NAT" (or just S-NAT or SNAT) refers to NAT involving changes to the source address and/or port, and "Destination NAT" (or just D-NAT or DNAT) refers to NAT involving changes to the destination address and/or port.

Masquerading

Masquerading is a special type of SNAT in which one computer rewrites packets to make them appear to come from itself. The computer's IP address used is determined automatically, and if it changes, old connections are destroyed appropriately. Masquerading is commonly used to share a Internet connection with a dynamic IP address among a network of computers.

Port Forwarding

Port forwarding is a type of DNAT in which one computer (such as a firewall) acts as a proxy for one or more other computers. The firewall accepts packets addressed to itself from the outside network, but rewrites them to appear to be addressed to other computers on the inside network before sending them on to their new destinations. In addition, related reply packets from the insid computers are rewritten to appear to be from the firewall and sent back to the appropriate outside computer.

Port forwarding is commonly used to provide publicly accessible network services (such as web o email servers) by computers other than the firewall, without requiring more than one public IP address. To the outside world, it appears that the services are being provided by the proxy machine, and to the actual server, it appears that all requests are coming from the proxy machine.

Load balancing

Load balancing involves distributing connections across a group of servers so that higher total throughput can be achieved. One way to implement simple load balancing is to set up port forwarding so that the destination address is selected in a round-robin fashion from a list of possible destinations.

Configuring iptables

The procedures for configuring **iptables** vary by distribution. This section provides both generic and Red Hat-specific information on **iptables** configuration.

Persistent rules

On recent Red Hat systems, you can find the **iptables** rules stored in */etc/sysconfig/iptables*. You can determine which runlevels have **iptables** enabled by running the command:

```
chkconfig --list iptables
```

You can enable **iptables** for runlevels 3, 4, and 5 by running the command:

```
chkconfig --levels 345 iptables on
```

You can start **iptables** manually by running:

```
service iptables start
```

You can stop it with:

```
service iptables stop
```

Other configuration files

The kernel's general networking and **iptables** behavior can be monitored and controlled by a number of pseudofiles in the */proc* filesystem. Table 1-9 lists the most prominent ones.

Table 1-9. iptables configuration and information files

Path	Purpose
	Contains settings for configurations in the */proc/sys* directory that are applied at boot time. Fo

/etc/sysctl.conf	example, */proc/sys/net/ipv4/ip_forward* can be set to `1` at boot time by adding an entry `net.ipv4.ip_forward = 1` to this file.
/proc/net/ip_conntrack	Dumps the contents of the connection tracking structures if you read it.
/proc/sys/net/ipv4/ip_conntrack_max	Controls the size of the connection tracking table in the kernel. The default value is calculate based on the amount of RAM in your computer. You may need to increase it if you are getting "`ip_conntrack: table full, dropping packet`" errors in your log files. See also the entry for */etc/sysctl.conf* in this table.
/proc/sys/net/ipv4/ip_forward	You need to set this to `1` for the host to act as a gateway (forwarding packets among the networks connected to its interfaces). See also the entry for */etc/sysctl.conf* in this table.

Compiling your own kernel

On Red Hat machines, you can determine the kernel you are currently running by looking at the outp of the `uname -r` command, which will print a message such as this:

```
2.4.20-20.9
```

Using your kernel version and your machine type, which can be determined by consulting the output of `uname -a` (see the manpage for **uname** for more information), you can find the most appropriate configuration file to use to build your new kernel in a file named something like this (we'll use `i636` for this example): */usr/src/linux-2.4.20-20.9/configs/kernel-2.4.20-i686.config*.

The **iptables** configuration settings are found in entries with names like `CONFIG_IP_NF_*`.

The following configuration options must be selected, at a minimum:

- `CONFIG_PACKET` (direct communication with network interfaces)
- `CONFIG_NETFILTER` (the basic kernel support required by **iptables**)
- `CONFIG_IP_NF_CONNTRACK` (required for NAT and masquerading)
- `CONFIG_IP_NF_FILTER` (adds the `filter` table)
- `CONFIG_IP_NF_IPTABLES` (the basic support for user space **iptables** utility)
- `CONFIG_IP_NF_MANGLE` (adds the `mangle` table)
- `CONFIG_IP_NF_NAT` (adds the `nat` table)

Warning ☠

You might be tempted to turn on `CONFIG_NET_FASTROUTE`, since fast routing sounds pretty attractive for a firewall computer. Don't do that; fast routing bypasses Netfilter's hooks.

The following configuration options provide compatibility layers with older firewalling technologies

- `CONFIG_IP_NF_COMPAT_IPCHAINS`
- `CONFIG_IP_NF_COMPAT_IPFWADM`

Tip ✎

There is a repository of Kernel patches that add features to Netfilter called "patch-o-matic." You can find out more about this repository by visiting the Netfilter web site at http://www.netfilter.org/ and reading the Netfilter Extensions HOWTO at http://www.netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO.html. Patch-o-matic is distributed separately from **iptables** and can be found at: ftp://ftp.netfilter.org/pub/patch-o-matic/.

You should exercise extreme caution when patching your kernel, especially if doing so with experimental Netfilter extensions. Some combinations don't even compile, and others might compile but fail to run. Always test your newly built kernels in a noncritical setting.

Connection Tracking

iptables associates packets with the logical connections they belong to (it even considers certain UDP communication patterns to imply connections even though UDP is a connectionless protocol). In order to do this, it tracks the progress of connections through their lifecycle, and this tracking information is made available through the `conntrack` match extension.

Although the underlying TCP connection state model is more complicated, the connection tracking logic assigns one of the states in Table 1-10 to each connection at any point in time.

Table 1-10. Connection tracking states

State	Description
`ESTABLISHED`	The connection has already seen packets going in both directions. See also `SEEN_REPLY` status.
`INVALID`	The packet doesn't belong to any tracked connections.
`NEW`	The packet is starting a new connection or is part of a connection that hasn't yet seen packets in both directions.
`RELATED`	The packet is starting a new connection, but the new connection is related to an existing connection (such as the data connection for an FTP transfer).

The connection tracking logic maintains three bits of status information associated with each connection. Table 1-11 contains a list of these status codes as they are named in the `conntrack` match extension (the `--ctstatus` option).

Table 1-11. Connection tracking statuses

Status	Description
`ASSURED`	For TCP connections, indicates the TCP connection setup has been completed. For UDP connections, indicates it looks like a UDP stream to the kernel.
`EXPECTED`	Indicates the connection was expected.

SEEN_REPLY	Indicates that packets have gone in both directions. See also ESTABLISHED state.

The **iptables** connection tracking logic allows plug-in modules to help identify new connections that are related to existing connections. You need to use these plug-ins if you want to make multiconnection protocols work right across your gateway/firewall. Table 1-12 shows the main connection tracking "helper" modules.

To use these, you need to run the **modprobe** command to install the kernel module. See also the `helper` match extension.

Table 1-12. Connection tracking helper modules

Helper	Protocol
ip_conntrack_amanda	Amanda backup protocol (requires CONFIG_IP_NF_AMANDA kernel config)
ip_conntrack_ftp	File Transfer Protocol (requires CONFIG_IP_NF_FTP kernel config)
ip_conntrack_irc	Internet Relay Chat (requires CONFIG_IP_NF_IRC kernel config)
ip_conntrack_tftp	Trivial File Transfer Protocol (requires CONFIG_IP_NF_TFTP kenel config)

Accounting

The kernel automatically tracks packet and byte counts for each rule. This information can be used to do accounting on network usage.

For example, if you add the following four rules to a machine serving as an Internet gateway (assuming two network interfaces: `eth0` for the internal network, and `eth1` for the Internet connection), the kernel tracks the number of packets and bytes exchanged with the outside world.

```
iptables -A FORWARD -i eth1
iptables -A FORWARD -o eth1
iptables -A INPUT -i eth1
iptables -A OUTPUT -o eth1
```

After running these commands, `iptables -L -v` shows (note the counts for `INPUT` and `OUTPUT` the nonzero counts indicate that some traffic had already traversed the chains by the time we displayed the counts):

```
Chain INPUT (policy ACCEPT 27 packets, 1728 bytes)
 pkts bytes target prot opt in   out source   destination
    3   192        all  --  eth1 any  anywhere anywhere

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target prot opt in   out  source   destination
    0     0        all  --  eth1 any  anywhere anywhere
    0     0        all  --  any  eth1 anywhere anywhere

Chain OUTPUT (policy ACCEPT 21 packets, 2744 bytes)
 pkts bytes target prot opt in   out  source   destination
    3   192        all  --  any  eth1 anywhere anywhere
```

See the discussion of the `-c`, `-n`, `-t`, and `-x` options in Table 1-14, and the `-L` and `-Z` options in Table 1-15 to learn more about the **iptables** options applicable to accounting applications.

Network Address Translation (NAT)

NAT is the modification of the addresses and/or ports of network packets as they pass through a computer. The computer performing NAT on the packets could be the source or destination of the packets, or it could be one of the computers on the route between the source and destination.

> **Warning** ☠
>
> Network address translation requires connection tracking, and connection tracking only works when the computer sees all the packets. So, if your firewall setup involves more than one computer, take care not to break connection tracking.

NAT can be used to perform a variety of useful functions based on the manipulations of addresses an ports. These functions can be grouped based on which addresses (source or destination) are being manipulated.

The `nat` built-in table is intended specifically for use in NAT applications.

The **iptables** NAT logic allows plug-in modules to help handle packets for protocols that embed addresses within the data being exchanged. Without the helper module, the packets would be modifie to go to different hosts, but the application data being exchanged would still use the pre-NAT addresses, keeping the application from working.

To use these, you need to run the **modprobe** command to install the kernel module. Table 1-13 lists the NAT helper modules.

Table 1-13. NAT helper modules

Helper	Protocol
ip_nat_amanda	Amanda backup protocol (requires `CONFIG_IP_NF_NAT_AMANDA` kernel config)
ip_nat_ftp	File Transfer Protocol (requires `CONFIG_IP_NF_NAT_FTP` kernel config)
ip_nat_irc	Internet Relay Chat (requires `CONFIG_IP_NF_NAT_IRC` kernel config)
ip_nat_snmp_basic	Simple Network Management Protocol (requires `CONFIG_IP_NF_NAT_SNMP_BASIC` kernel config)
ip_nat_tftp	Trivial File Transfer Protocol (requires `CONFIG_IP_NF_NAT_TFTP` kernel config)

If you want certain packets to bypass NAT, you can write rules that match the packets you are interested in and jump to the special target `ACCEPT`. You need to have such rules before your other NAT rules.

```
iptables -t nat -i eth1 ... -j ACCEPT
```

Source NAT and Masquerading

Source NAT (SNAT) is used to share a single Internet connection among computers on a network. Th computer attached to the Internet acts as a gateway and uses SNAT (along with connection tracking) to rewrite packets for connections between the Internet and the internal network. The source address of outbound packets is replaced with the static IP address of the gateway's Internet connection. When outside computers respond, they will set the destination address to the IP address of the gateway's Internet connection, and the gateway will intercept those packets, change their destination addresses the correct inside computer, and forward them to the internal network.

Since SNAT entails modifying the source addresses and/or ports of packets just before they leave the kernel, it is performed through the `POSTROUTING` chain of the `nat` table.

There are two ways of accomplishing SNAT with **iptables**. The `SNAT` target extension is intended fo situations where the gateway computer has a static IP address, and the `MASQUERADE` target extensio is intended for situations where the gateway computer has a dynamic IP address. The `MASQUERADE` target extension provides additional logic that deals with the possibility that the network interface could go off line and come back up again with a different address. Additional overhead is involved in this logic, so if you have a static IP address, you should use the `SNAT` target extension instead.

You can set up SNAT on the `eth1` interface by putting a simple rule on the `POSTROUTING` chain of the `nat` table:

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT
```

The corresponding command for masquerading is:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

Destination NAT

Destination NAT (DNAT) exposes specific services on an internal network to the outside world without linking the internal computers directly to the Internet. And as long as there is no more than one service to be exposed on any given port, only one Internet connection (public IP address) is required. The gateway computer redirects connections to the specified ports to the designated interna computers and ports and arranges for return traffic to go back to the original address outside the network.

Since DNAT entails modifying the destination addresses and/or ports of packets just before they are either routed to local processes or forwarded to other computers, it is performed through the `PREROUTING` chain of the `nat` table.

For example, to forward inbound connections coming in on a gateway's port 80 (HTTP) to an interna web server running on port 8080 of 192.168.1.3, you could use a rule like this:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80
  -j DNAT --to-destination 192.168.1.3:8080
```

Transparent Proxying

Transparent proxying is a way to intercept specific outgoing connections and redirect them to a computer that will service them in the place of the original destination computer. This technique allows you to set up proxies for services without having to configure each computer on the internal network. Since all traffic to the outside world goes through the gateway, all connections to the outsid world on the given port will be proxied transparently.

If you have an HTTP proxy (such as Squid) configured to run as a transparent proxy on your firewall computer and listen on port 8888, you can add one rule to redirect outbound HTTP traffic to the HTT proxy:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80
  -j REDIRECT --to-port 8888
```

It is more complicated to transparently proxy to a service running on a different host. You can find details on making this work for Squid in Daniel Kiracofe's "Transparent Proxy with Linux and Squid mini-HOWTO," available online at The Linux Documentation Project's web site (http://www.tldp.org/HOWTO/TransparentProxy.html).

Load Distribution and Balancing

You can distribute load across a number of participating hosts using the `nth` match extension and th `DNAT` target extension.

Load balancing is a refinement of load distribution that entails using load statistics for the target hos to advise the choice of target for packets in order to keep the participating hosts close to equally loaded.

Stateless and Stateful Firewalls

A *firewall* is a gateway computer that restricts the flow of network traffic among the networks it connects.

Stateless firewalls use simple rules that do not require connection or other state tracking, such as matches on combinations of source and destination addresses and ports for certain protocols.

Stateful firewalls allow more advanced packet processing that involve tracking connections and othe state, such as keeping track of recent activity by host or connection (such as the `iplimit`, `limit`, and `recent` match extensions).

iptables supports both types of firewall rules (but see the warning in the section Network Address Translation (NAT)).

Tools of the Trade

There are many networking tools that can come in handy while troubleshooting your firewall or othe network functionality. Table 1-14 provides links for a few of the most common ones.

Table 1-14. Tools of the trade

Tool	Description

ethereal	Network protocol analyzer. http://www.ethereal.com/
Nessus	Remote security scanner. http://www.nessus.org/intro.html
nmap	Network mapper. http://www.insecure.org/nmap/
ntop	Network traffic probe. http://ntop.ethereal.com/ntop.html
ping	Send ICMP ECHO_REQUEST to specific hosts.
tcpdump	Packet capture and dumping. http://www-nrg.ee.lbl.gov/
traceroute	Print the route packets take to a specific host. http://www-nrg.ee.lbl.gov/

iptables Command Reference

Most of the options for the **iptables** command can be grouped into subcommands and rule match criteria. Table 1-15 describes the other options.

Table 1-15. iptables miscellaneous options

Option	Description
`-c` *`packets bytes`*	When combined with the `-A`, `-I`, or `-R` subcommand, sets the packet counter to *`packets`* and the byte counter to *`bytes`* for the new or modified rule.
`--exact`	Synonym for `-x`.
`-h`	Displays information on **iptables** usage. If it appears after `-m` *`match`* or `-j` *`target`*, then any additional help related to the extension *`match`* or *`target`* (respectively) is also displayed.
`--help`	Synonym for `-h`.
`-j` *`target`* [*`options`*]	Determines what to do with packets matching this rule. The *`target`* can be the name of a user-defined chain, one of the built-in targets, or an **iptables** extension (in which case there may be additional *`options`*).
`--jump`	Synonym for `-j`.
`--line-numbers`	When combined with the `-L` subcommand, displays numbers for the rules in each chain, so you can refer to the rules by index when inserting rules into (via `-I`) or deleting rules from (via `-D`) a chain.
`-m` *`match`* [*`options`*]	Invoke extended *`match`*, possibly with additional *`options`*.
`--match`	Synonym for `-m`.
`-M` *`cmd`*	Used to load an **iptables** module (with new targets or match extensions) when appending, inserting, or replacing rules.
`--modprobe=`*`cmd`*	Synonym for `-M`.
`-n`	Displays numeric addresses and ports instead of looking up and displaying domain names for the IP addresses and displaying service names for the port numbers. This can be especially useful if your DNS service is slow or down.
`--numeric`	Synonym for `-n`.
`--set-counters`	Synonym for `-c`.
`-t table`	Performs the specified subcommand on *`table`*. If this option is not used, the subcommand operates on the `filter` table by default.
`--table`	Synonym for `-t`.

`-v`	Produces verbose output.
`--verbose`	Synonym for `-v`.
`-x`	Displays exact numbers for packet and byte counters, rather than the default abbreviated format with metric suffixes (K, M, or G).

Getting help

iptables provides some online help. You can get basic information via these commands:

```
iptables -h
iptables -m match -h
iptables -j TARGET -h
man iptables
```

> **Warning** ☠
>
> Sometimes there are contradictions among these sources of information.

The iptables Subcommands

Each **iptables** command can contain one subcommand, which performs an operation on a particular table (and, in some cases, chain). Table 1-16 lists the options that are used to specify the subcommand.

> **Warning** ☠
>
> The manpage for the **iptables** command in the 1.2.7a release shows a `-C` option in the synopsis section, but the option does not exist

Table 1-16. iptables subcommand options

Option	Description
`-A` *`chain rule`*	Appends *`rule`* to *`chain`*.
`--append`	Synonym for `-A`.
`-D` *`chain`* [*`index`* | *`rule`*]	Deletes the rule at position *`index`* or matching *`rule`* from *`chain`*.
`--delete`	Synonym for `-D`.
`--delete-chain`	Synonym for `-X`.

`-E` *chain newchain*	Renames *chain* to *newchain*.
`-F` [*chain*]	Flushes (deletes) all rules from *chain* (or from all chains if no chain is given).
`--flush`	Synonym for `-F`.
`-I` *chain [index] rule*	Inserts *rule* into *chain*, at the front of the chain, or at position *index*.
`--insert`	Synonym for `-I`.
`-L` [*chain*]	Lists the rules for *chain* (or for all chains if no chain is given).
`--list`	Synonym for `-L`.
`-N` *chain*	Creates a new user-defined *chain*.
`--new-chain`	Synonym for `-N`. Commonly abbreviated `--new`.
`-P` *chain target*	Sets the default policy of the built-in *chain* to *target*. Applies to built-in chains and targets only.
`--policy`	Synonym for `-P`.
`-R` *chain index rule*	Replaces the rule at position *index* of *chain* with the new *rule*.
`--rename-chain`	Synonym for `-E`.
`--replace`	Synonym for `-R`.
`-V`	Displays the version of **iptables**.
`--version`	Synonym for `-V`.
`-X` [*chain*]	Deletes the user-defined *chain* (or all user-defined chains if none is specified).
`-Z` *chain*	Zeros the packet and byte counters for *chain* (or for all chains if no chain is specified).
`--zero`	Synonym for `-Z`.

iptables Matches and Targets

iptables has a small number of built-in matches and targets, and a set of extensions that are loaded if they are referenced. The matches for IP are considered built-in, and the others are considered match extensions (even though the `icmp`, `tcp`, and `udp` match extensions are automatically loaded when th corresponding protocols are referenced with the `-p` built-in Internet Protocol match option).

This section describes all of the built-in and extension matches and targets included in **iptables** version 1.2.7a.

> **Tip ✎**
>
> Some options can have their senses inverted by inserting an exclamation point surrounded by spaces, immediately before the option. The options that allow this are annotated with `[!]`. Only the noninverted sense is described in the sections that follow since the inverted sense can be inferred from the description.

Internet Protocol (IPv4) matches

The built-in IP matches are listed in the later section ip (Internet Protocol IPv4) matches in order to keep with the encyclopedic format of this section.

ACCEPT target

This built-in target discontinues processing of the current chain and goes to the next table and chain i the standard flow (see Figure 1-1 through Figure 1-3 and Table 1-4 through Table 1-7).

Only this target and the `DROP` target can be used as the policy for a built-in chain.

ah match

Match extension for the IPSec protocol's Authentication Header (AH) Security Parameters Index (SP field. The destination address and the SPI together define the Security Association, or SA for the packet. Used in conjunction with the `-p ah` (or `-p ipv6-auth` or `-p 51`) protocol specification option. Table 1-17 describes the single option to this match.

> **Tip ✎**
>
> This match is available only if your kernel has been configured with `CONFIG_IP_NF_MATCH_AH_ESP` enabled.

Table 1-17. ah match options

Option	Description
`--ahspi [!] min[:max]`	Match the value (if only *min* is given) or inclusive range (if both *min* and *max* are given) for the SPI field of the AH.

For example:

```
iptables -A INPUT -p ah -m ah --ahspi 500 -j DROP
```

See the book *IPv6 Essentials*, by Silvia Hagen (O'Reilly) for more information on the IPv6 protocol. See also `esp` match.

connmark Match

Match based on the packet's connection mark. Table 1-18 describes the single option to this match.

Table 1-18. connmark match options

Option	Description
`--mark` *`value`*`[/`*`mask`*`]`	Match if the packet's connection mark is equal to *`value`* after applying *`mask`*.

See also the `CONNMARK` target extension.

CONNMARK target

Set the packet's connection mark. Table 1-19 describes the options to this target.

Table 1-19. CONNMARK target options

Option	Description
`--set-mark` *`value`*	Set the packet's connection mark to the integer *`value`*.
`--save-mark`	Save the packet's mark into the connection.
`--restore-mark`	Restore the packet's mark from the connection.

See also the `connmark` match extension.

conntrack match

Match based on information maintained by the connection tracking machinery. Table 1-20 describes the options to this match.

Tip ✎

This match is available only if your kernel has been configured with `CONFIG_IP_NF_MATCH_CONNTRACK` enabled.

Table 1-20. conntrack match options

Option	Description
`[!] --ctexpire` *`min`*`[:`*`max`*`]`	Match the value (if only *`min`* is given) or inclusive range (if both *`min`* and *`max`* are given) for the connection's remaining lifetime (in seconds).
`--ctorigdst [!]`	Match the original destination address (before NAT).

addr[/*mask*]	
--ctorigsrc [!] *addr*[/*mask*]	Match based on the original source address (before NAT).
[!] --ctproto *proto*	Match the given protocol. The *proto* argument can be a protocol number or name. See also Table 1-37.
--ctrepldst [!] *addr*[/*mask*]	Match the replacement destination address (after NAT).
--ctreplsrc [!] *addr*[/*mask*]	Match the replacement source address (after NAT).
[!] --ctstate *states*	Match any of the given connection tracking states. The *states* argument is a comma-separated list of connection tracking states (see Table 1-10) or SNAT or DNAT.
[!] --ctstatus *statuses*	Match any of the given connection tracking statuses. The *statuses* argument is a comma-separated list of connection tracking statuses (see Table 1-11). The special value NONE may be used to indicate that none of the status bits should b set.

DNAT target

Perform Destination Network Address Translation (DNAT) by modifying the destination addresses and/or ports of packets. If multiple destination addresses are specified, connections are distributed across those addresses. Connection tracking information ensures that packets for each connection go to the same host and port. Table 1-21 describes the options to this target.

Table 1-21. DNAT target options

Option	Description
--to-destination *a1*[-*a2*] [:*p1*-*p2*]	*a1* and *a2* are used to specify a range of destination addresses. *p1* and *p2* are used to specify a range of ports (for TCP or UDP protocols).

The DNAT target extension is available only on the PREROUTING and OUTPUT chains of the nat table.

For example, to forward packets coming in on interface eth0 for port 80 to an internal web server listening on IP address 192.168.1.80:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80
  -j DNAT --to-destination 192.168.1.80
```

> **Warning** ☠
>
> When doing this kind of DNAT, it is important to separate internal and external DNS so that internal hosts use the inside address of the web server directly.

See also:

- The `REDIRECT` target extension for simple redirection to ports on the local machine.
- The `SNAT` target extension for source NAT.
- The `nth` match extension for an alternative way of implementing load distribution.

DROP target

This built-in target causes the kernel to discontinue processing in the current chain without continuin processing elsewhere and without providing rejection notices to the sender.

Only the `DROP` target and the `ACCEPT` target can be used as the policy for a built-in chain.

See also the `REJECT` target extension, which will send an ICMP reply to the sender.

dscp match

Use this match to identify packets with particular Differentiated Services Codepoint (DSCP) values i their IPv4 headers. The DSCP field is a reinterpretation of the TOS byte of the IPv4 header. Table 1-22 describes the options to this match.

> **Tip** ✎
>
> This match is available only if your kernel has been configured with `CONFIG_IP_NF_MATCH_DSCP` enabled.

Table 1-22. dscp match options

Option	Description
`--dscp` *value*	Match if the packet's DSCP field equals *value*, which can be specified in decimal or hexadecimal notation (such as `0x0e`).
`--dscp-class` *name*	Match if the packet's DSCP field value corresponds to DSCP class *name*. The names are `AF[1-3][1-4]`, `BE`, `CS[0-7]`, and `EF`. See Table 1-23 for descriptions of the classes, and Table 1-24 fo the corresponding DSCP values.

At most, one of these options may be specified for any rule.

Table 1-23 provides descriptions of the classes, and Table 1-24 shows the corresponding DSCP value

Table 1-23. Differentiated Services classes