CHAPTER 3

iptables: The Linux Firewall Administration Program

Chapter 2, "Packet-Filtering Concepts," covers the background ideas and concepts behind a packet-filtering firewall. Each built-in rule chain has its own default policy. Each rule can apply not only to an individual chain, but also to a specific network interface, message protocol type (such as TCP, UDP, or ICMP), and service port or ICMP message type number. Individual acceptance, denial, and rejection rules are defined for the `INPUT` chain and the `OUTPUT` chain, as well as for the `FORWARD` chain, which you'll learn about at the end of this chapter and in Chapter 6, "Packet Forwarding." The next chapter pulls those ideas together to demonstrate how to build a simple, single-system, custom-designed firewall for your site.

This chapter covers the iptables firewall administration program used to build a Netfilter firewall. For those of you who are familiar with or accustomed to the older ipfwadm and ipchains programs used with the IPFW technology, iptables will look very similar to those programs. However, it is much more feature-rich and flexible, and it is very different on subtle levels.

There is indeed a difference between iptables and Netfilter, though you'll often hear the terms used interchangeably. Netfilter is the Linux kernel-space program code to implement a firewall within the Linux kernel, either compiled directly into the kernel or included as a set of modules. On the other hand, iptables is the userland program used for administration of the Netfilter firewall. Throughout this text, I will refer to iptables as being inclusive of both Netfilter and iptables, unless otherwise noted.

Differences Between IPFW and Netfilter Firewall Mechanisms

Because iptables is so different from the previous ipchains, this book won't attempt to cover the older implementation.

The next section is written for the reader who is familiar with or is currently using ipchains. If iptables is your first introduction to Linux firewalling, you can skip ahead to the section "Netfilter Packet Traversal."

If you are converting from ipchains, you'll notice several minor differences in the iptables syntax, most notably that the input and output network interfaces are identified separately. iptables is highly modularized, and the individual modules must occasionally be loaded explicitly. Logging is a rule target rather than a command option. Connection state tracking can be maintained. Address and Port Translation are now logically separate functions from packet filtering. Full Source and Destination Address Translation are implemented. Masquerading is now a term used to refer to a specialized form of source address NAT. Port forwarding and Destination Address Translation are supported directly without the need for third-party software support such as ipmasqadm.

MASQUERADING IN EARLIER VERSIONS OF LINUX

For those of you who are new to Linux, Network Address Translation (NAT) is fully implemented in iptables. Before this, NAT was called masquerading in Linux. A simple, partial implementation of Source Address Translation, masquerading was used by site owners who had a single public IP address and who wanted other hosts on their private network to be capable of accessing the Internet. Outgoing packets from these internal hosts had their source address masqueraded to that of the public, routable IP address.

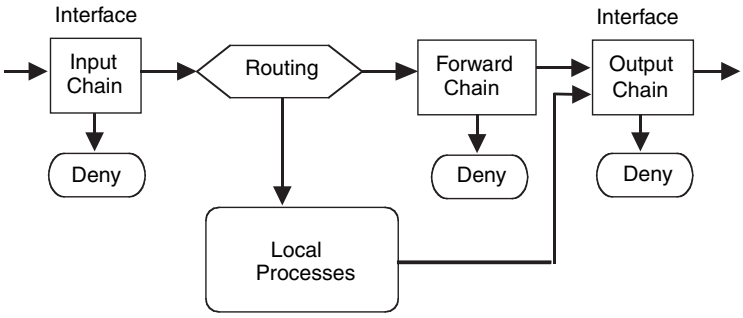
The most important difference is in how packets are routed or forwarded through the operating system, making for subtle differences in how the firewall rule set is constructed.

For ipchains users, understanding the differences in packet traversal that are discussed in the next two sections is very important. iptables and ipchains look very much alike on the surface, but they are very different in practice. It's very easy to write syntactically correct iptables rules that have a different effect from what a similar rule would have done in ipchains. It can be confusing. If you already know ipchains, you must keep the differences in mind.

IPFW Packet Traversal

Under IPFW (ipfwadm and ipchains), three built-in filter chains were used. All packets arriving on an interface were filtered against the input chain. If the packet was accepted, it was passed to the routing module. The routing function determined whether the packet was to be delivered locally or forwarded to another outgoing interface. IPFW packet flow is pictured in Figure 3.1.

FIGURE 3.1
IPFW packet traversal. (Figure based on "Linux IPCHAINS-HOWTO," by Rusty Russel, v1.0.8.)



If forwarded, the packet was filtered a second time against the forward chain. If the packet was accepted, it was passed to the output chain.

Both locally generated outgoing packets and forwarded packets were passed to the output chain. If the packet was accepted, it was sent out the interface.

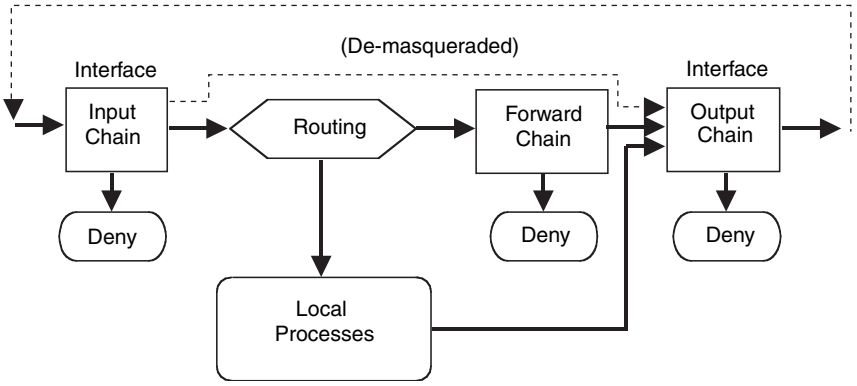
Received and sent local (loopback) packets passed through two filters. Forwarded packets passed through three filters.

The loopback path involved two chains. As shown in Figure 3.2, each loopback packet passed through the output filter before going "out" the loopback interface, where it was then delivered to the loopback's input interface. Then the input filter was applied.

Note that the loopback path demonstrates why people's X Window session hangs when starting a firewall script that either doesn't allow loopback traffic or fails before doing so when a deny by default policy is used.

In the case of response packets being demasqueraded before forwarding them on to the LAN, the input filters were applied. Rather than passing through the routing function, the packet was handed directly to the output filter chain. Thus, demasqueraded incoming packets were filtered twice. Outgoing masqueraded packets were filtered three times.

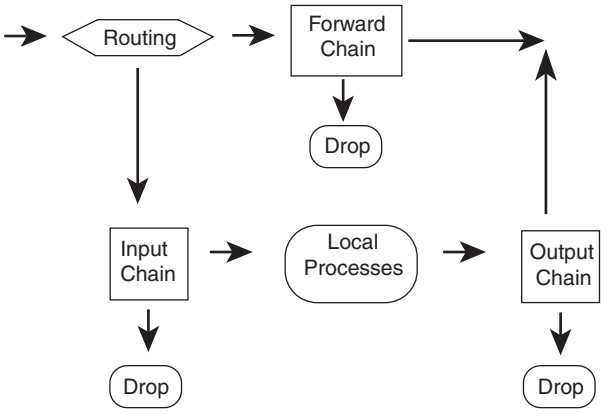
FIGURE 3.2
IPFW loopback and masqueraded packet traversal. (Figure based on "Linux IPCHAINS-HOWTO," by Rusty Russel, v1.0.8.)



Netfilter Packet Traversal

Under Netfilter (iptables), built-in `INPUT`, `OUTPUT`, and `FORWARD` filter chains are used. Incoming packets pass through the routing function, which determines whether to deliver the packet to the local host's input chain or on to the forward chain. Netfilter packet flow is pictured in Figure 3.3.

FIGURE 3.3
Netfilter packet traversal. (Figure based on "Linux 2.4 Packet Filtering HOWTO," by Rusty Russel, v1.0.1.)



If a locally destined packet is accepted by the `INPUT` chain's rules, the packet is delivered locally. If a remotely destined packet is accepted by the `FORWARD` chain's rules, the packet is sent out the appropriate interface.

Outgoing packets from local processes are passed to the `OUTPUT` chain's rules. If the packet is accepted, it is sent out the appropriate interface. Thus, each packet is filtered once (except for loopback packets, which are filtered twice).

Basic iptables Syntax

Firewalls built with Netfilter are built through the iptables firewall administration command. The iptables command implements the firewall policies that you create and manages the behavior of the firewall. Netfilter firewalls have three individual tables: filter, NAT, and mangle. Within these tables, firewalls are built through chains, with each individual link in the chain being an individual iptables command.

Within the default `filter` table there is a chain for input or data coming into the firewall, a chain for output or data leaving the firewall, a chain for forwarding or data being sent through the firewall, and other chains including chains named and configured by the user, commonly (and appropriately) called user-defined chains. The NAT and `mangle` tables have specialty chains that will be discussed later. For now, it's sufficient to know that the `filter` table is the default table for implementing a basic firewall, the NAT table is used to provide NAT and related functions, and the `mangle` table is used when the packet will be altered by the firewall.

iptables commands are issued with very specific syntax. Many times, the ordering of the options given to iptables makes the difference between a successful command and a syntax error. The commands issued to iptables fall through, so a command that allows certain packets that follows a command that denies those same packets will cause the data to be dropped by the firewall.

The basic syntax for an iptables command begins with the iptables command itself, followed by one or more options, a chain, a set of match criteria, and a target or disposition. The layout of the command largely depends on the action to be performed. Consider this syntax:

```
iptables <option> <chain> <matching criteria> <target>
```

In building a firewall, the option is usually `-A` to append a rule onto the end of the ruleset. Naturally, there are several options depending on the target and the operation being performed. This chapter covers most of those options.

As previously stated, the chain can be an input chain, an output chain, a forwarding chain, or a user-defined chain. In addition, the chain might also be a specialty chain contained in the NAT or `mangle` tables.

The matching criteria in an iptables command sets the conditions for the rule to be applied. For example, the matching criteria would be used to tell iptables that all TCP traffic destined for port `80` is allowed into the firewall.

Finally, the target sets the action to perform on a matching packet. The target can be something as simple as `DROP` to silently discard the packet, or it can send the matching packet to a user-defined chain, or it can perform any other configured action in iptables.

The following sections of this chapter show hands-on examples using iptables to implement real-world rules for various tasks. Some of the examples include syntax and options that haven't yet been introduced. If you get lost, refer to this section or the iptables man page for more information on the syntax being used.

iptables Features

iptables uses the concept of separate rule tables for different kinds of packet processing functionality. These rule tables are implemented as functionally separate table modules. The three primary modules are the rule `filter` table, the NAT `nat` table, and the specialized packet-handling `mangle` table. Each of these three table modules has its own associated module extensions that are dynamically loaded when first referenced, unless you've built them directly into the kernel.

The `filter` table is the default table. The other tables are specified by a command-line option. The basic `filter` table features include these:

- Chain-related operations on the three built-in chains (`INPUT`, `OUTPUT`, and `FORWARD`) and on user-defined chains
- Help
- Target disposition (`ACCEPT` or `DROP`)
- IP header field match operations for protocol, source and destination address, input and output interfaces, and fragment handling
- Match operations on the TCP, UDP, and ICMP header fields

The `filter` table has two kinds of feature extensions: *target* extensions and *match* extensions. The target extensions include the `REJECT` packet disposition, the `BALANCE` and `CLUSTERIP` targets, the `CLASSIFY` target, `CONNMARK`, `TRACE`, and the `LOG` and `ULOG` functionalities. The match extensions support matching on the following:

- The current connection state
- Port lists (supported by the multiport module)

- The hardware Ethernet MAC source address or physical device
- The type of address, link-layer packet type, or range of IP addresses
- Various parts of IPSec packets or the IPSec policy
- The ICMP type
- The length of the packet
- The time the packet arrived
- Every *n*th packet or random packets
- The packet sender's user, group, process, or process group ID
- The IP header Type of Service (TOS) field (possibly set by the `mangle` table)
- The TTL section of the IP header
- The iptables mark field (set by the `mangle` table)
- Rate-limited packet matching

The `mangle` table has two target extensions. The `MARK` module supports assigning a value to the packet's `mark` field that iptables maintains. The `TOS` module supports setting the value of the `TOS` field in the IP header.

UPCOMING FEATURES IN IPTABLES

iptables is being actively developed and enhanced. Based on the source code and build environment, it is clear that additional modules will be available by the time this book is published. Some other modules are not intended for release in the public distributions.

For example, there is an experimental `MIRROR` target. This target retransmits a packet after reversing the source and destination sections of the IP header.

The `nat` table has target extension modules for Source and Destination Address Translation and for Port Translation. These modules support these forms of NAT:

- SNAT—Source NAT.
- DNAT—Destination NAT.
- MASQUERADE—A specialized form of source NAT for connections that are assigned a temporary, changeable, dynamically assigned IP address (such as a phone dial-up connection).
- REDIRECT—A specialized form of destination NAT that redirects the packet to the local host, regardless of the address in the IP header's destination field.

All TCP state flags can be inspected, and filtering decisions can be made based on the results. iptables can check for stealth scans, for example.

TCP can optionally specify the maximum segment size that the sender is willing to accept in return. Filtering on this one, single TCP option is a very specialized case. The TTL section of the IP header can also be matched and is a specialized case as well.

TCP connection state and ongoing UDP exchange information can be maintained, allowing packet recognition on an ongoing basis rather than on a stateless, packet-by-packet basis. Accepting packets recognized as being part of an established connection allows bypassing the overhead of checking the rule list for each packet. When the initial connection is accepted, subsequent packets can be recognized and allowed.

Generally, the TOS field is of historical interest only. The TOS field is either ignored or used with the newer Differentiated Services definitions by intermediate routers. IP TOS filtering has uses for local packet prioritizing—routing and forwarding among local hosts and the local router.

Incoming packets can be filtered by the MAC source address. This has limited, specialized uses for local authentication because MAC addresses are passed only between adjacent hosts and routers.

Individual filter log messages can be prefixed with user-defined strings. Messages can be assigned kernel logging levels as defined for `/etc/syslog.conf`. This allows logging to be turned on and off, and for the log output files to be defined, in `/etc/syslog.conf`. In addition, there is a `ULOG` option that sends logging to a userspace daemon, `ulogd`, to enable further detail to be logged about the packet.

Packet matches can be limited to an initial burst rate, after which a limit is imposed by the number of allowed matches per second. If match limiting is enabled, the default is that, after an initial burst of five matched packets, a rate limit of three matches per hour is imposed. In other words, if the system were flooded with `ping` packets, for example, the first five `ping`s would match. After that, a single `ping` packet could be matched 20 minutes later, and another one could be matched 20 minutes after that, regardless of how many `echo-request`s were received. The disposition of the packets, whether logged or not, would depend on any subsequent rules regarding the packets.

The `REJECT` target can optionally specify which ICMP (or `RST` for TCP) error message to return. The IPv4 standard requires TCP to accept either `RST` or `ICMP` as an error indication, although `RST` is the default TCP behavior. `iptable`'s default is to return nothing (`DROP`) or else to return an ICMP error (`REJECT`).

Along with `REJECT`, another special-purpose target is `QUEUE`. Its purpose is to hand off the packet via the netlink device to a user-space program for handling. If there is no waiting program, the packet is dropped.

`RETURN` is another special-purpose target. Its purpose is to return from a user-defined chain before rule matching on that chain has completed.

Locally generated outgoing packets can be filtered based on the user, group, process, or process group ID of the program generating the packet. Thus, access to remote services can be authorized at the packet-filtering level on a per-user basis. This is a specialized option for multiuser, multipurpose hosts because firewall routers shouldn't have normal user accounts.

Matching can be performed on various pieces of IPSec header, including the SPIs (security parameter indices) of the AH (authentication header) and ESP (encapsulating security payload).

The type of packet, be it broadcast, unicast, or multicast, is another form of match. This is done at the link layer.

A range of ports as well as a range of addresses are also valid matches with iptables. The type of address is another valid match as well. Related to type matching is the ICMP packet type. Recall that there are a number of valid types of ICMP packet types. Iptables can match against these types.

The length of the packet is a valid match, as is the time a packet arrived. This time matching is interesting. Using the time matches, you could configure the firewall to reject certain traffic after business hours or allow it only during certain times of day.

A good match for auditing, a random packet match is also available with iptables. Using this match, you can capture every *n*th packet and log it. This would be a method for auditing the firewall rules without logging too much information.

NAT Table Features

There are three general forms of NAT:

- *Traditional, unidirectional outbound NAT*—Used for networks using private addresses.
  - *Basic NAT*—Address Translation only. Usually used to map local private source addresses to one of a block of public addresses.
  - *NAPT (Network Address Port Translation)*—Usually used to map local private source addresses to a single public address (for example, Linux masquerading).
- *Bidirectional NAT*—Two-way address translation allows both outbound and inbound connections. A use of this is bidirectional address mapping between IPv4 and IPv6 address spaces.

- *Twice NAT*—Two-way Source and Destination Address Translation allows both outbound and inbound connections. Twice NAT can be used when the source and destination networks' address spaces collide. This could be the result of one site mistakenly using public addresses assigned to someone else. Twice NAT also can be used as a convenience when a site was renumbered or assigned to a new public address block and the site administrator didn't want to administer the new address assignments locally at that time.

iptables NAT supports source (SNAT) and destination NAT (DNAT). The NAT table allows for modifying a packet's source address or destination address and port. It has three built-in chains:

- The `PREROUTING` chain specifies destination changes to incoming packets before passing the packet to the routing function (DNAT). Changes to the destination address can be to the local host (transparent proxying, port redirection) or to a different host for host forwarding (`ipmasqadm` functionality, port forwarding in Linux parlance) or load sharing.
- The `OUTPUT` chain specifies destination changes to locally generated outgoing packets before the routing decision has been made (DNAT, REDIRECT). This is usually done to transparently redirect an outgoing packet to a local proxy, but it can also be used to port-forward to a different host.
- The `POSTROUTING` chain specifies source changes to outgoing packets being routed through the box (SNAT, MASQUERADE). The changes are applied after the routing decision has been made.

MASQUERADING IN IPTABLES

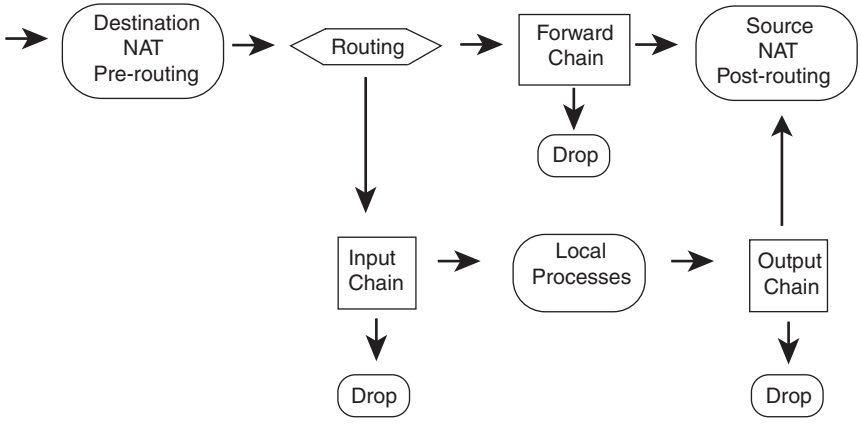
In iptables, masquerading is a specialized case of source NAT in the sense that the masqueraded connection state is forgotten immediately if the connection is lost. It's intended for use with connections (for example, dial-up) in which the IP address is assigned temporarily. If the user reconnected immediately, he would probably be assigned a different IP address than he had during the previous connection. (This is often not the case with many cable-modem and ADSL service providers. Often, after a connection loss, the same IP address is assigned upon reconnection.)

With regular SNAT, connection state is maintained for the duration of a timeout period. If a connection were reestablished quickly enough, any current network-related programs could continue undisturbed because the IP address hasn't changed, and interrupted TCP traffic would be retransmitted.

The distinction between MASQUERADE and SNAT is an attempt to avoid a situation that occurred in previous Linux NAT/MASQUERADE implementations. When a dial-up connection was lost and the user reconnected immediately, he was assigned a new IP address. The new address couldn't be used immediately because the old IP address and NAT information were still in memory until the timeout period expired.

Figure 3.4 shows the NAT chains in relation to the routing function and `INPUT`, `OUTPUT`, and `FORWARD` chains.

FIGURE 3.4
NAT packet traversal. (Figure based on "Linux 2.4 Packet Filtering HOWTO," v1.0.1, and "Linux 2.4 NAT HOWTO," v1.0.1.)



Note that, for outgoing packets, the routing function is implied between the local process and the `OUTPUT` chain. Static routing is used to determine which interface the packet will go out on, before the `OUTPUT` chain's filter rules are applied.

`mangle` Table Features

The `mangle` table allows *marking*, or associating a Netfilter-maintained value, with the packet, as well as making changes to the packet before sending the packet on to its destination. The `mangle` table has five built-in chains:

- The `PREROUTING` chain specifies changes to incoming packets as they arrive at an interface, before any routing or local delivery decision has been made.
- The `INPUT` chain specifies changes to packets as they are processed, but after the `PREROUTING` chain is traversed.

- The `POSTROUTING` chain specifies changes to packets as they are exiting the firewall, after the `OUTPUT` chain is traversed.
- The `FORWARD` chain specifies changes to packets that are forwarded through the firewall.
- The `OUTPUT` chain specifies changes to locally generated outgoing packets.

For the `TOS` field, the local Linux router can be configured to honor the `TOS` flags set by the `mangle` table or as set by the local hosts.

Little information is available about packet marking in the iptables documentation, beyond that it's used by the Linux Quality of Service implementation and that it's intended as a communication flag between iptables modules.

The preceding sections provided an overview of the features available in iptables and the general structure and functionality of the individual table modules. The following sections present the syntax used to invoke these features.

iptables Syntax

As presented earlier, iptables uses the concept of separate rule tables for different packet processing functionality. Nondefault tables are specified by a command-line option. Three tables are available:

- `filter`—The `filter` table is the default table. It contains the actual firewall filtering rules. The built-in chains include these:
  - `INPUT`
  - `OUTPUT`
  - `FORWARD`
- `nat`—The `nat` table contains the rules for Source and Destination Address and Port Translation. These rules are functionally distinct from the firewall filter rules. The built-in chains include these:
  - `PREROUTING` (DNAT/REDIRECT)
  - `OUTPUT` (DNAT/REDIRECT)
  - `POSTROUTING` (SNAT/MASQUERADE)
- `mangle`—The `mangle` table contains rules for setting specialized packet-routing flags. These flags are then inspected later by rules in the `filter` table. The built-in chains include these:

- PREROUTING (routed packets)
- INPUT (packets arriving at the firewall but after the PREROUTING chain)
- FORWARD (changes packets being routed through the firewall)
- POSTROUTING (changes packets just before they leave the firewall, after the OUTPUT chain)
- OUTPUT (locally generated packets)

SYNTAX FORMAT CONVENTIONS

The conventions used to present command-line syntax options are fairly standard in the computer world. For those of you who are new to Linux or to computer documentation in general, Table 3.1 shows the conventions used in the upcoming syntax descriptions.

TABLE 3.1
Conventions Representing Command-Line Syntax Options

ELEMENT	DESCRIPTION
|	A bar or pipe symbol separates alternate syntax options. For example, most of the iptables commands have both a short and a long form, such as -L and --list, and so they would be listed as alternate options because you would use one or the other of -L or --list.
<value>	Angle brackets indicate a user-supplied value, such as a string or numeric value.
[ ]	Square brackets indicate that the enclosed command, option, or value is optional. For example, most match operators can take a negation operator, !, which matches anything other than the value specified in the match. The negation operator is usually placed between the match operator and the value to be matched.
<value>:<value>	A colon indicates a range of values. The two values define the minimum and maximum values within the range. Because ranges themselves are optional, the convention is more often presented as <value>[:<value>].

`filter` Table Commands

The `filter` table commands are provided by the `ip_tables` module. The functionality is enabled by loading the module, which is done automatically with the first invocation of the iptables command, or it could be compiled into the kernel itself, which means you don't need to worry about modules being loaded at all.

`filter` TABLE OPERATIONS ON ENTIRE CHAINS

Table 3.2 shows the iptables operations on entire chains.

TABLE 3.2
iptables Operations on Entire Chains

OPTION	DESCRIPTION
`-N | --new-chain <chain>`	Creates a user-defined chain.
`-F | --flush [<chain>]`	Flushes the chain, or all chains if none is specified.
`-X | --delete-chain [<chain>]`	Deletes the user-defined chain, or all chains if none is specified.
`-P | --policy <chain> <policy>`	Defines the default policy for one of the built-in chains, `INPUT`, `OUTPUT`, or `FORWARD`. The policy is either `ACCEPT` or `DROP`.
`-L | --list [<chain>]`	Lists the rules in the chain, or all chains if none is specified.
`-Z | --zero`	Resets the packet and byte counters associated with each chain.
`-h | <some command> -h`	Lists the iptables commands and options, or if preceded by an iptables command, lists the syntax and options for that command.
`--modprobe=<command>`	Use `<command>` to load the necessary module(s) when adding or inserting a rule into a chain.
`-E | --rename-chain <old chain> <new chain>`	Renames the user-defined chain `<old chain>` to the user-defined chain `<new chain>`.

The `-h` help command is obviously not an operation on a chain nor is `--modprobe=<command>`, but I didn't know where else to list the command.

The list command takes additional options, as shown in Table 3.3.

TABLE 3.3
Options to the List Chain Command

OPTION	DESCRIPTION
`-L -n | --numeric`	Lists the IP addresses and port numbers numerically, rather than by name
`-L -v | --verbose`	Lists additional information about each rule, such as the byte and packet counters, rule options, and relevant network interface
`-L -x | --exact`	Lists the exact values of the counter, rather than the rounded-off values
`-L -line-numbers`	Lists the rule's position within its chain

`filter` TABLE OPERATIONS ON A RULE

The most frequently used commands to create or delete rules within a chain are shown in Table 3.4.

TABLE 3.4
Chain Commands on Individual Rules

COMMAND	DESCRIPTION
`-A | --append <chain>`	Appends a rule to the end of a chain
`-I | --insert <chain>`	Inserts a rule at the beginning of the chain
`-R | --replace <chain> <rule number> <rule specification>`	Replaces a rule in the chain
`-D | --delete <chain> <rule number>`	Deletes the rule at position `rule number` within a chain

BASIC `filter` TABLE MATCH OPERATIONS

The basic filter match operations supported in the default iptables `filter` table are listed in Table 3.5.

TABLE 3.5
`filter` Table Rule Operations

OPTION	DESCRIPTION
`-i | --in-interface [!] [<interface>]`	For incoming packets on either the `INPUT` or the `FORWARD` chains, or their user-defined subchains, specifies the interface name that the rule applies to. If no interface is specified, all interfaces are implied.
`-o | --out-interface [!] [<interface>]`	For outgoing packets on either the `OUTPUT` or the `FORWARD` chains, or their user-defined subchains, specifies the interface name that the rule applies to. If no interface is specified, all interfaces are implied.
`-p | --protocol [!] [<protocol>]`	Specifies the IP protocol that the rule applies to. The built-in protocols are `tcp`, `udp`, `icmp`, and `all`. The protocol value can be either the name or the numeric value, as listed in `/etc/protocols`.
`-s | --source | --src [!] <address>[</mask>]`	Specifies the host or network source address in the IP header.
`-d | --destination | --dst [!] <address>[</mask>]`	Specifies the host or network destination address in the IP header.
`-j | --jump <target>`	Specifies the target disposition for the packet if it matches the rule. The default targets include the built-in targets, an extension, or a user-defined chain.
`[!] -f | --fragment`	Specifies second and additional fragmented packets. The negated version of this specifies unfragmented packets.
`-c | --set-counters <packets> <bytes>`	Initializes the packet and byte counters.

RULE TARGETS ARE OPTIONAL

If the packet matches a rule that doesn't have a target disposition, the packet counters are updated, but list traversal continues.

`tcp filter` TABLE MATCH OPERATIONS

TCP header match options are listed in Table 3.6.

TABLE 3.6
`tcp filter` Table Match Operations

`-p tcp` OPTION	DESCRIPTION
`--source-port | --sport [[!] <port>[:<port>]]`	This command specifies the source ports.
`--destination-port | --dport [!] <port>[:<port>]`	This command specifies the destination ports.
`--tcp-flags [!] <mask>[,<mask>] <set>[,<set>]`	This command tests the bits in the mask list, out of which the following bits must be set in order to match.
`[!] -syn`	The SYN flag must be set as an initial connection request.
`--tcp-option [!] <number>`	The only legal `tcp` option is the maximum packet size that the sending host is willing to accept.

`udp filter` TABLE MATCH OPERATIONS

UDP header match options are listed in Table 3.7.

TABLE 3.7
`udp filter` Table Match Operations

`-p udp` OPTION	DESCRIPTION
`--source-port | --sport [!] <port>[:<port>]`	Specifies the source ports
`--destination-port | --dport [!] <port>[:<port>]`	Specifies the destination ports

`icmp filter` TABLE MATCH OPERATIONS

ICMP header match options are listed in Table 3.8.

TABLE 3.8
`icmp filter` Table Match Operations

MATCH	DESCRIPTION
`--icmp-type [!] <type>`	Specifies the ICMP type name or number. The ICMP type is used in place of a source port.

The major supported ICMP type names and numeric values are the following:

- `echo-reply` (0)
- `destination-unreachable` (3)
  - `network-unreachable`
  - `host-unreachable`
  - `protocol-unreachable`
  - `port-unreachable`
  - `fragmentation-needed`
  - `network-unknown`
  - `host-unknown`
  - `network-prohibited`
  - `host-prohibited`
- `source-quench` (4)
- `redirect` (5)
- `echo-request` (8)
- `time-exceeded` (10)
- `parameter-problem` (11)

ADDITIONAL ICMP SUPPORT

iptables supports a number of additional, less common or router-specific ICMP message types and subtypes. To see the entire list, use the following `iptables` help command:

```
iptables -p icmp -h
```

`filter` Table Target Extensions

The `filter` table target extensions include logging functionality and the capability to reject a packet rather than dropping it.

Table 3.9 lists the options available to the `LOG` target. Table 3.10 lists the single option available to the `REJECT` target.

TABLE 3.9
LOG Target Extension

-j LOG **OPTION**	**DESCRIPTION**
--log-level <syslog level>	Log level is either the numeric or the symbolic login priority, as listed in /usr/include/sys/syslog.h. These are the same log levels used in /etc/syslog.conf. The levels are emerg (0), alert (1), crit (2), err (3), warn (4), notice (5), info (6), memerg (0), alert (1), crit (2), err (3), warn (4), notice (5), info (6), and debut (7).
--log-prefix <"descriptive string">	The prefix is a quoted string that will be printed at the start of the log message for the rule.
--log-ip-options	This command includes any IP header options in the log output.
--log-tcp-sequence	This command includes the TCP packet's sequence number in the log output.
--log-tcp-option	This command includes any TCP header options in the log output.

TABLE 3.10
REJECT Target Extension

-j REJECT **OPTION**	**DESCRIPTION**
--reject-with <ICMP type 3>	By default, a rejected packet results in an ICMP type 3 icmp-port-unreachable message being returned to the sender. Other type 3 error messages can be returned instead, including icmp-net-unreachable, icmp-host-unreachable, icmp-proto-unreachable, icmp-net-prohibited, and icmp-host-prohibited.
--reject-with tcp-reset	Incoming TCP packets can be rejected with the more standard TCP RST message, rather than an ICMP error message.
--reject-with echo-reply	ping echo-request messages can be rejected with a faked echo-reply message. That is, the firewall generates the reply, but the request is not forwarded to the target host.

THE ULOG TABLE TARGET EXTENSION

Related to the LOG target is the ULOG target, which sends the log message to a userspace program for logging. Behind the scenes for ULOG, the packet gets multicast by the kernel through a netlink socket of your choosing (the default is socket 1). The userspace daemon would then

read the message from the socket and do with it what it pleases. The ULOG target is typically used to provide more extensive logging than is possible with the standard LOG target.

As with the LOG target, processing continues after matches on a ULOG targeted rule. The ULOG target has four configuration options, as described in Table 3.11.

TABLE 3.11
ULOG Target Extension

OPTION	DESCRIPTION
`--ulog-nlgroup <group>`	Defines the netlink group that will receive the packet. The default group is 1.
`--ulog-prefix <prefix>`	Messages will be prefixed by this value, up to 32 characters in length.
`--ulog-cprange <size>`	The size in bytes to send to the netlink socket. The default is 0, which sends the entire packet.
`--ulog-qthreshold <size>`	The size in packets to queue within the kernel. The default is 1, which means that one packet is sent per message to the netlink socket.

`filter` Table Match Extensions

The `filter` table match extensions provide access to the fields in the TCP, UDP, and ICMP headers, as well as the match features available in iptables, such as maintaining connection state, port lists, access to the hardware MAC source address, and access to the IP TOS field.

MATCH SYNTAX

The match extensions require the `-m` or `--match` command to load the module, followed by any relevant match options.

`multiport filter` TABLE MATCH EXTENSION

`multiport` port lists can include up to 15 ports per list. Whitespace isn't allowed. There can be no blank spaces between the commas and the port values. Port ranges cannot be interspersed in the list. Also, the `-m multiport` command must exactly follow the `-p <protocol>` specifier.

Table 3.12 lists the options available to the `multiport` match extension.

TABLE 3.12
`multiport` Match Extension

`m | --match multiport` **OPTION**	**DESCRIPTION**
`--source-port <port> [,<port>]`	Specifies the source port(s).
`--destination-port <port> [,<port>]`	Specifies the destination port(s).
`--port <port>[,<port>]`	Source and destination ports are equal, and they match a port in the list.

The `multiport` syntax can be a bit tricky. Some examples and cautions are included here. The following rule blocks incoming packets arriving on interface `eth0` destined for the UDP ports associated with NetBIOS and SMB, common ports that are exploited on Microsoft Windows computers and targets for worms:

```
iptables -A INPUT -i eth0 -p udp\
         -m multiport --destination-port 135,136,137,138,139 -j DROP
```

The next rule blocks outgoing connection requests sent through the `eth0` interface to high ports associated with the TCP services NFS, `socks`, and `squid`:

```
iptables -A OUTPUT -o eth0 -p tcp\
         -m multiport --destination-port 2049,1080,3128 --syn -j REJECT
```

What is important to note in this example is that the `multiport` command must exactly follow the protocol specification. A syntax error would have resulted if the `--syn` were placed between the `-p tcp` and the `-m multiport`.

To show a similar example of `--syn` placement, the following is correct:

```
iptables -A INPUT -i <interface> -p tcp \
         -m multiport --source-port 80,443 ! --syn -j ACCEPT
```

However, this causes a syntax error:

```
iptables -A INPUT -i <interface> -p tcp ! --syn \
         -m multiport --source-port 80,443 -j ACCEPT
```

Furthermore, the placement of source and destination parameters is not obvious. The following two variations are correct:

```
iptables -A INPUT -i <interface> -p tcp -m multiport \
              --source-port 80,443  \
         ! --syn -d $IPADDR --dport 1024:65535 -j ACCEPT
```

and

```
iptables -A INPUT -i <interface> -p tcp -m multiport \
              --source-port 80,443  \
         -d $IPADDR ! --syn --dport 1024:65535 -j ACCEPT
```

However, this causes a syntax error:

```
iptables -A INPUT -i <interface> -p tcp -m multiport \
              --source-port 80,443  \
              -d $IPADDR --dport 1024:65535 ! --syn -j ACCEPT
```

This module has some surprising syntax side effects. Either of the two preceding correct rules produces a syntax error if the reference to the `SYN` flag is removed:

```
iptables -A INPUT -i <interface> -p tcp -m multiport \
              --source-port 80,443  \
         -d $IPADDR --dport 1024:65535 -j ACCEPT
```

The following pair of rules, however, does not:

```
iptables -A OUTPUT -o <interface> \
        -p tcp -m multiport --destination-port 80,443 \
        ! --syn -s $IPADDR --sport 1024:65535 -j ACCEPT

iptables -A OUTPUT -o <interface> \
        -p tcp -m multiport --destination-port 80,443 \
       --syn -s $IPADDR --sport 1024:65535 -j ACCEPT
```

Note that the `--destination-port` argument to the `multiport` module is not the same as the `--destination-port` or `--dport` argument to the module that performs matching for the `-p tcp` arguments.

`limit filter` TABLE MATCH EXTENSION

Rate-limited matching is useful for choking back the number of log messages that would be generated during a flood of logged packets.

Table 3.13 lists the options available to the `limit` match extension.

TABLE 3.13
`limit` Match Extension

`-m | --match limit` OPTION	DESCRIPTION
`--limit <rate>`	Maximum number of packets to match within the given time frame
`--limit-burst <number>`	Maximum number of initial packets to match before applying the limit

The burst rate defines the number of initial matches to be accepted. The default value is five matches. When the limit has been reached, further matches are limited to the rate limit. The

default limit is three matches per hour. Optional time frame specifiers include `/second`, `/minute`, `/hour`, and `/day`.

In other words, by default, when the initial burst rate of five matches is reached within the time limit, at most three more packets will match over the next hour, one every 20 minutes, regardless of how many packets are received. If a match doesn't occur within the rate limit, the burst is recharged by one.

It's easier to demonstrate rate-limited matching than it is to describe it in words. The following rule will limit logging of incoming `ping` message matches to one per second when an initial five `echo-requests` are received within a given second:

```
iptables -A INPUT -i eth0 \
         -p icmp --icmp-type echo-request \
         -m limit --limit 1/second -j LOG
```

It's also possible to do rate-limited packet acceptance. The following two rules, in combination, will limit acceptance of incoming `ping` messages to one per second when an initial five `echo-requests` are received within a given second:

```
iptables -A INPUT -i eth0 \
         -p icmp --icmp-type echo-request \
         -m limit --limit 1/second -j ACCEPT

iptables -A INPUT -i eth0 \
         -p icmp --icmp-type echo-request -j DROP
```

The next rule limits the number of log messages generated in response to dropped ICMP `redirect` messages. When an initial five messages have been logged within a 20-minute time frame, at most three more log messages will be generated over the next hour, one every 20 minutes:

```
iptables -A INPUT -i eth0 \
         -p icmp --icmp-type redirect \
         -m limit -j LOG
```

The assumption in the final example is that the packet and any additional unmatched `redirect` packets are silently dropped by the default `DROP` policy for the `INPUT` chain.

`dstlimit` filter TABLE MATCH EXTENSION

The `dstlimit` match extension enables rate limiting on a per-destination basis, whether per IP address or per port. Note the difference between the `dstlimit` match extension and the `limit` match extension, which has one limit for packets of a certain type.

Table 3.14 lists the options for the `dstlimit` match extension.

TABLE 3.14
`dstlimit` Match Extension

OPTION	DEFINITION
`--dstlimit <average>`	Maximum average match rate in packets per second.
`--dstlimit-mode <mode>`	Defines the limit to be per IP (`dstip`), per IP and port tuple (`dstip-dstport`), per source IP and destination IP tuple (`srcip-dstip`), or per source IP and destination IP and destination port tuple (`srcipdstip-dstport`).
`--dstlimit-name <name>`	Specifies the name for the file to be placed in `/proc/net/ipt_dstlimit/`.
`[--dstlimit-burst <burst>]`	Specifies the number of packets that should be matched when received as a burst of packets. The default is `5`.
`[--dstlimit-htable-size <size>]`	Defines the number of buckets in the hashtable.
`[--dstlimit-htable-max <entries>]`	Defines the limit for the number of entries in the hashtable.
`[--dstlimit-htable-gcinterval <interval>]`	Defines the length of time between cleanup of the hashtable. The value for `<interval>` is in milliseconds, with the default being 1000 ms.
`[--dstlimit-htable-expire <time>]`	Defines the amount of time before an idle entry is purged from the hashtable. The value is in milliseconds, with the default being 10000 ms.

`state filter` TABLE MATCH EXTENSION

Static filters look at traffic on a packet-by-packet basis alone. Each packet's particular combination of source and destination addresses and ports, the transport protocol, and the current TCP state flag combination is examined without reference to any ongoing context. ICMP messages are treated as unrelated, out-of-band IP Layer 3 events.

The state extension provides additional monitoring and recording technology to augment the stateless, static packet-filter technology. State information is recorded when a TCP connection or UDP exchange is initiated. Subsequent packets are examined not only based on the static tuple information, but also within the context of the ongoing exchange. In other words, some of the contextual knowledge usually associated with the upper TCP Transport layer, or the UDP Application layer, is brought down to the filter layer.

After the exchange is initiated and accepted, subsequent packets are identified as part of the established exchange. Associated ICMP messages are identified as being related to a particular exchange.

(In computer terminology, a collection of values or attributes that together uniquely identify an event or object is called a *tuple*. A UDP or TCP packet is uniquely identified by the tuple combination of its protocol, UDP or TCP, the source and destination addresses, and the source and destination ports.)

For session monitoring, the advantages of maintaining state information are less obvious for TCP because TCP maintains state information by definition. For UDP, the immediate advantage is the capability to distinguish responses from other datagrams. In the case of an outgoing DNS request, which represents a new UDP exchange, the concept of an established session allows an incoming UDP response datagram from the host and port the original message was sent to, within a certain time-limited window. Incoming UDP datagrams from other hosts or ports are not allowed. They are not part of the established state for this particular exchange. When applied to TCP and UDP, ICMP error messages are accepted if the error message is related to the particular session.

In considering packet flow performance and firewall complexity, the advantages are more obvious for TCP flows. Flows are primarily a firewall performance and optimization technology. The main goal of flows is to allow bypassing the firewall inspection path for a packet. Much faster TCP packet handling is obtained in some cases because the remaining firewall filters can be skipped if the TCP packet is immediately recognized as part of an allowed, ongoing connection. For TCP connections, flow state can be a major win in terms of filtering performance. Also, standard TCP application protocol rules can be collapsed into a single initial allow rule. The number of filter rules is reduced (theoretically, but not necessarily in practice, as you'll see later in the book).

The main disadvantage is that maintaining a state table requires more memory than standard firewall rules alone. Routers with 70,000 simultaneous connections, for example, would require tremendous amounts of memory to maintain state table entries for each connection. State maintenance is often done in hardware for performance reasons, where associative table lookups can be done simultaneously or in parallel. Whether implemented in hardware or software, state engines must be capable of reverting a packet to the traditional path if memory isn't available for the state table entry.

Also, table creation, lookup, and teardown take time in software. The additional processing overhead is a loss in many cases. State maintenance is a win for ongoing exchanges such as an FTP transfer or a UDP streaming multimedia session. Both types of data flow represent potentially large numbers of packets (and filter rule match tests). State maintenance is not a firewall performance win for a simple DNS or NTP client/server exchange, however. State buildup and

teardown can easily require as much processing—and more memory—than simply traversing the filter rules for these packets.

The advantages are also questionable for firewalls that filter primarily web traffic. Web client/server exchanges tend to be brief and ephemeral.

Telnet and SSH sessions are in a gray area. On heavily trafficked routers with many such sessions, the state maintenance overhead may be a win by bypassing the firewall inspection. For fairly quiescent sessions, however, it's likely that the connection state entry will timeout and be thrown away. The state table entry will be re-created when the next packet comes along, after it has passed the traditional firewall rules.

Table 3.15 lists the options available to the `state` match extension.

TABLE 3.15
`state` Match Extension

`-m | --match state` **OPTION**	**DESCRIPTION**
`--state <state>[,<state>]`	Matches if the connection state is one in the list. Legal values are `NEW`, `ESTABLISHED`, `RELATED`, or `INVALID`.

TCP connection state and ongoing UDP exchange information can be maintained, allowing network exchanges to be filtered as `NEW`, `ESTABLISHED`, `RELATED`, or `INVALID`:

- `NEW` is equivalent to the initial TCP `SYN` request, or to the first UDP packet.
- `ESTABLISHED` refers to the ongoing TCP `ACK` messages after the connection is initiated, to subsequent UDP datagrams exchanged between the same hosts and ports, and to ICMP `echo-reply` messages sent in response to a previous `echo-request`.
- `RELATED` currently refers only to ICMP error messages. FTP secondary connections are managed by the additional FTP connection tracking support module. With the addition of that module, the meaning of `RELATED` is extended to include the secondary FTP connection.
- An example of an `INVALID` packet is an incoming ICMP error message that wasn't a response to a current session, or an `echo-reply` that wasn't a response to a previous `echo-request`.

Ideally, using the `ESTABLISHED` match allows the firewall rule pair for a service to be collapsed into a single rule that allows the first request packet. For example, using the `ESTABLISHED` match, a web client rule requires allowing only the initial outgoing `SYN` request. A DNS client request requires only the rule allowing the initial UDP outgoing request packet.

With a deny-by-default input policy, connection tracking can be used (theoretically) to replace all protocol-specific filters with two general rules that allow incoming and outgoing packets that are part of an established connection, or packets related to the connection. Application-specific rules are required for the initial packet alone.

Although such a firewall setup might very well work for a small or residential site in most cases, it is unlikely to perform adequately for a larger site or a firewall that handles many connections simultaneously. The reason goes back to the case of state table entry timeouts, in which a state entry for a quiescent connection is replaced because of table size and memory constraints. The next packet that would have been accepted by the deleted state entry requires a rule to allow the packet, and the state table entry must be rebuilt.

A simple example of this is a rule pair for a local DNS server operating as a cache-and-forward name server. A DNS forwarding name server uses server-to-server communication. DNS traffic is exchanged between source and destination ports `53` on both hosts. The UDP client/server relationship can be made explicit. The following rules explicitly allow outgoing (`NEW`) requests, incoming (`ESTABLISHED`) responses, and any (`RELATED`) ICMP error messages:

```
iptables -A INPUT -m state \
        --state ESTABLISHED,RELATED -j ACCEPT

iptables -A OUTPUT --out-interface <interface> -p udp \
         -s $IPADDR --source-port 53 -d $NAME_SERVER --destination-port 53 \
         -m state --state NEW,RELATED -j ACCEPT
```

DNS uses a simple query-and-response protocol. But what about an application that can maintain an ongoing connection for extended periods, such as an FTP control session or a telnet or SSH session? If the state table entry is cleared out prematurely for some reason, future packets won't have a state entry to be matched against to be identified as part of an `ESTABLISHED` exchange.

The following rules for an SSH connection allow for that possibility:

```
iptables -A INPUT -m state \
        --state ESTABLISHED,RELATED -j ACCEPT

iptables -A OUTPUT -m state \
        --state ESTABLISHED,RELATED -j ACCEPT

iptables -A OUTPUT --out-interface <interface> -p tcp \
         -s $IPADDR --source-port $UNPRIVPORTS \
         -d $REMOTE_SSH_SERVER --destination-port 22 \
         -m state --state NEW, -j ACCEPT
```

```
iptables -A OUTPUT --out-interface <interface> -p tcp ! --syn \
         -s $IPADDR --source-port $UNPRIVPORTS \
         -d $REMOTE_SSH_SERVER --destination-port 22 \
         -j ACCEPT

iptables -A INPUT --in-interface <interface> -p tcp ! --syn \
         -s $REMOTE_SSH_SERVER --source-port 22 \
         -d $IPADDR --destination-port $UNPRIVPORTS \
         -j ACCEPT
```

`mac filter` TABLE MATCH EXTENSION

Table 3.16 lists the options available to the `mac` match extension.

TABLE 3.16
`mac` Match Extension

`-m | --match mac` OPTION	DESCRIPTION
`--mac-source [!] <address>`	Matches the Layer 2 Ethernet hardware source address, specified as `xx:xx:xx:xx:xx:xx`, in the incoming Ethernet frame

Remember that MAC addresses do not cross router borders (or network segments). Also remember that only source addresses can be specified. The `mac` extension can be used only on an `in-interface`, such as the `INPUT`, `PREROUTING`, and `FORWARD` chains.

The following rule allows incoming SSH connections from a single local host:

```
iptables -A INPUT -i <local interface> -p tcp \
         -m mac --mac-source xx:xx:xx:xx:xx:xx \
         --source-port 1024:65535 \
         -d <IPADDR> --dport 22 -j ACCEPT
```

`owner filter` TABLE MATCH EXTENSION

Table 3.17 lists the options available to the `owner` match extension.

TABLE 3.17
`owner` Match Extension

`-m | --match owner` OPTION	DESCRIPTION
`--uid-owner <userid>`	Matches on the creator's UID
`--gid-owner <groupid>`	Matches on the creator's GID
`--pid-owner <processid>`	Matches on the creator's PID
`--sid-owner <sessionid>`	Matches on the creator's SID or PPID
`--cmd-owner <name>`	Matches on a packet created by a process with command name `name`

The match refers to the packet's creator. The extension can be used on the `OUTPUT` chain only.

These match options don't make much sense on a firewall router; they make more sense on an end host.

So, let's say that you have a firewall gateway with a monitor, perhaps, but no keyboard. Administration is done from a local, multiuser host. A single user account is allowed to log in to the firewall from this host. On the multiuser host, administrative access to the firewall could be locally filtered as shown here:

```
iptables -A OUTPUT -o eth0 -p tcp \
         -s <IPADDR> --sport 1024:65535 \
         -d <fw IPADDR> --dport 22 \
         -m owner --uid-owner <admin userid> \
         --gid-owner <admin groupid> -j ACCEPT
```

`mark filter` TABLE MATCH EXTENSION

Table 3.18 lists the options available to the `mark` match extension.

TABLE 3.18
`mark` Match Extension

`-m | -match mark` OPTION	DESCRIPTION
`--mark <value>[/<mask>]`	Matches packets having the Netfilter-assigned `mark` value

The `mark` value and the mask are unsigned long values. If a mask is specified, the value and the mask are `AND`ed together.

In the example, assume that an incoming telnet client packet between a specific source and destination had been marked previously:

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp \
         -s <some src address> --sport 1024:65535 \
         -d <some destination address> --dport 23 \
         -m mark --mark 0x00010070 \
         -j ACCEPT
```

The `mark` value being tested for here was set at some earlier point in the packet processing. The `mark` value is a flag indicating that this packet is to be handled differently from other packets.

`tos filter` TABLE MATCH EXTENSION

Table 3.19 lists the options available to the `tos` match extension.

TABLE 3.19
`tos` Match Extension

`-m | --match tos` **OPTION**	**DESCRIPTION**
`--tos <value>`	Matches on the IP TOS setting

The `tos` value can be one of either the string or numeric values:

- `minimize-delay, 16, 0x10`
- `maximize-throughput, 8, 0x08`
- `maximize-reliability, 4, 0x04`
- `minimize-cost, 2, 0x02`
- `normal-service, 0, 0x00`

THE VALUE OF TOS BITS

The TOS bits are of historical interest only. Linux does support their use locally, and various Linux firewall documents refer to the bits and their uses. Nevertheless, the fact remains that the TOS bits are not used or examined generally.

The `TOS` field has been redefined as the Differentiated Services (DS) field for use by the Differentiated Services Control Protocol (DSCP).

For more information on Differentiated Services, see these sources:

- RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers"
- RFC 2475, "An Architecture for Differentiated Services"
- RFC 2990, "Next Steps for the IP QoS Architecture"
- RFC 3168, "The Addition of Explicit Congestion Notification (ECN) to IP"
- RFC 3260, "New Terminology and Clarifications for Diffserv"

`unclean filter` TABLE MATCH EXTENSION

The specific packet-validity checks performed by the `unclean` module are not documented. The module is considered to be experimental, and the iptables authors recommend against its use for now.

The following line shows the `unclean` module syntax. The module takes no arguments:

```
-m | --match unclean
```

The `unclean` extension might be "blessed" by the time this book is published. In the meantime, the module lends itself to an example of the `LOG` options:

```
iptables -A INPUT -p ! tcp -m unclean \
         -j LOG --log-prefix "UNCLEAN packet: " \
         --log-ip-options

iptables -A INPUT -p tcp -m unclean \
         -j LOG --log-prefix "UNCLEAN TCP: " \
         --log-ip-options \
         --log-tcp-sequence --log-tcp-options

iptables -A INPUT -m unclean -j DROP
```

`addrtype` `filter` TABLE MATCH EXTENSION

The `addrtype` match extension is used to match packets based on the type of address used, such as unicast, broadcast, and multicast. The types of addresses include those listed in Table 3.20.

TABLE 3.20
Address Types Used with the `addrtype` Match

NAME	DESCRIPTION
ANYCAST	An anycast packet
BLACKHOLE	A blackhole address
BROADCAST	A broadcast address
LOCAL	A local address
MULTICAST	A multicast address
PROHIBIT	A prohibited address
UNICAST	A unicast address
UNREACHABLE	An unreachable address
UNSPEC	An unspecified address

Two commands are used with the `addrtype` match, as listed in Table 3.21.

TABLE 3.21
`addrtype` Match Commands

OPTION	DESCRIPTION
`--src-type <type>`	Matches for addresses with a source of type `<type>`.
`--dst-type <type>`	Matches for addresses with a destination of type `<type>`.

`iprange filter` TABLE MATCH

Sometimes defining a range of IP addresses using CIDR notation is insufficient for your needs. For example, if you need to limit a certain range of IPs that don't fall on a subnet boundary or cross that boundary by only a couple addresses, the `iprange` match type will do the job.

Using the `iprange` match, you specify an arbitrary range of IP addresses for the match to take effect. The `iprange` match can also be negated. Table 3.22 lists the commands for the `iprange` match.

TABLE 3.22
`iprange` Match Commands

COMMAND	DESCRIPTION
`[!] --src-range <ip address-ip address>`	Specifies (or negates) the range of IP addresses to match. The range is given with a single hyphen and no spaces.
`[!] --dst-range <ip address-ip address>`	Specifies (or negates) the range of IP addresses to match. The range is given with a single hyphen and no spaces.

`length filter` TABLE MATCH

The `length filter` table match examines the length of the packet. If the packet's length matches the value given or optionally falls within the range given, the rule is invoked. Table 3.23 lists the one and only command related to the `length` match.

TABLE 3.23
`length` Match Command

COMMAND	DESCRIPTION
`--length <length>[:<length>]`	Matches a packet of `<length>` or within the range `<length:length>`

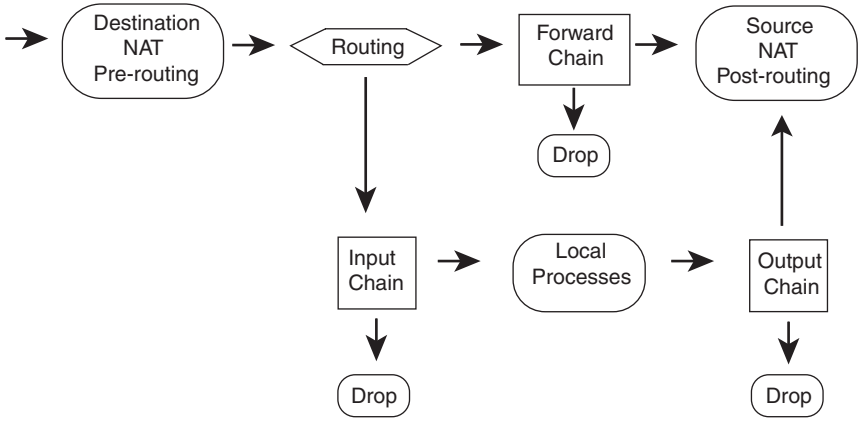
NAT Table Target Extensions

As mentioned earlier, iptables supports four general kinds of NAT: source NAT (SNAT); destination NAT (DNAT); masquerading (MASQUERADE), which is a specialized case of the SNAT implementation; and local port direction (REDIRECT) to the local host. As part of the NAT table, each of these targets is available when a rule specifies the `nat` table by using the `-t nat` table specifier.

SNAT NAT TABLE TARGET EXTENSION

Source Address and Port Translation (NAPT) is the kind of NAT people are most commonly familiar with. As shown in Figure 3.5, Source Address Translation is done after the routing decision is made. SNAT is a legal target only in the `POSTROUTING` chain. Because SNAT is applied immediately before the packet is sent out, only an outgoing interface can be specified.

FIGURE 3.5
NAT packet traversal.



Some documents refer to this form of source NAT (the most common form) as NAPT, to acknowledge the port number modification. The other form of traditional, unidirectional NAT is basic NAT, which doesn't touch the source port. That form is used when you are translating between the private LAN and a pool of public addresses.

NAPT is used when you have a single public address. The source port is changed to a free port on the firewall/NAT machine because it's translating for any number of internal computers, and the port that the internal machine is using might already be in use by the NAT machine. When the responses come back, the port is all that the NAT machine has to

determine that the packet is really meant for an internal computer rather than itself and then to determine which internal computer the packet is meant for.

The general syntax for `SNAT` is as follows:

```
iptables -t nat -A POSTROUTING --out-interface <interface>  ...  \
         -j SNAT --to-source <address>[-<address>][:<port>-<port>]
```

The source address can be mapped to a range of possible IP addresses, if more than one is available.

The source port can be mapped to a specific range of source ports on the router.

MASQUERADE NAT TABLE TARGET EXTENSION

Source Address Translation has been implemented in two different ways in iptables, as SNAT and as MASQUERADE. The difference is that the MASQUERADE target extension is intended for use with connections on interfaces with dynamically assigned IP addresses, particularly in the case in which the connection is temporary and the IP address assignment is likely to be different at each new connection. As discussed previously, in the section "NAT Table Features," MASQUERADE can be useful for phone dial-up connections in particular.

Because masquerading is a specialized case of SNAT, it is likewise a legal target only in the `POSTROUTING` chain, and the rule can refer to the outgoing interface only. Unlike the more generalized SNAT, MASQUERADE does not take an argument specifying the source address to apply to the packet. The IP address of the outgoing interface is used automatically.

The general syntax for `MASQUERADE` is as follows:

```
iptables -t nat -A POSTROUTING --out-interface <interface>  ...  \
         -j MASQUERADE [--to-ports <port>[-<port>]]
```

The source port can be mapped to a specific range of source ports on the router.

DNAT NAT TABLE TARGET EXTENSION

Destination Address and Port Translation is a highly specialized form of NAT. A residential or small business site is most likely to find this feature useful if its public IP address is dynamically assigned or if the site has a single IP address, and the site administrator wants to forward incoming connections to internal servers that aren't publicly visible. In other words, the DNAT features can be used to replace the previously required third-party port-forwarding software, such as `ipmasqadm`.

Referring back to Figure 3.5, Destination Address and Port Translation is done before the routing decision is made. DNAT is a legal target in the `PREROUTING` and `OUTPUT` chains. On the

PREROUTING chain, DNAT can be a target when the incoming interface is specified. On the OUTPUT chain, DNAT can be a target when the outgoing interface is specified.

The general syntax for DNAT is as follows:

```
iptables -t nat -A PREROUTING --in-interface <interface>  ...  \
         -j DNAT --to-destination <address>[-<address>][:<port>-<port>]
iptables -t nat -A OUTPUT --out-interface <interface>  ...  \
         -j DNAT --to-destination <address>[-<address>][:<port>-<port>]
```

The destination address can be mapped to a range of possible IP addresses, if more than one is available.

The destination port can be mapped to a specific range of alternate ports on the destination host.

REDIRECT NAT TABLE TARGET EXTENSION

Port redirection is a specialized case of DNAT. The packet is redirected to a port on the local host. Incoming packets that would otherwise be forwarded on are redirected to the incoming interface's INPUT chain. Outgoing packets generated by the local host are redirected to a port on the local host's loopback interface.

REDIRECT is simply an alias, a convenience, for the specialized case of redirecting a packet to *this* host. It offers no additional functional value. DNAT could just as easily be used to cause the same effect.

REDIRECT is likewise a legal target only in the PREROUTING and OUTPUT chains. On the PREROUTING chain, REDIRECT can be a target when the incoming interface is specified. On the OUTPUT chain, REDIRECT can be a target when the outgoing interface is specified.

The general syntax for REDIRECT is as follows:

```
iptables -t nat -A PREROUTING --in-interface <interface>  ...  \
         -j REDIRECT [--to-ports <port>[-<port>]]
iptables -t nat -A OUTPUT --out-interface <interface>  ...  \
         -j REDIRECT [--to-ports <port>[-<port>]]
```

The destination port can be mapped to a different port or to a specific range of alternate ports on the local host.

BALANCE NAT TABLE TARGET EXTENSION

The BALANCE target enables a round-robin method of sending connections to more than one target host. The BALANCE target uses a range of addresses for this purpose and thus provides a rudimentary load-balancing.

The general syntax for BALANCE is as follows:

```
iptables -t nat -A PREROUTING -p tcp -j BALANCE \
         --to-destination <ip address>-<ip address>
```

The CLUSTERIP target also provides some of these same options.

mangle Table Commands

The mangle table targets and extensions apply to the OUTPUT and PREROUTING chains. Remember, the filter table is implied by default. To use the mangle table features, you must specify the mangle table with the -t mangle directive.

mark mangle TABLE TARGET EXTENSION

Table 3.24 lists the target extensions available to the mangle table.

TABLE 3.24
mangle Target Extensions

-t mangle OPTION	DESCRIPTION
-j MARK --set-mark <value>	Sets the value of the Netfilter mark value for this packet
-j TOS --set-tos <value>	Sets the TOS value in the IP header

There are two mangle table target extensions: MARK and TOS. MARK contains the functionality to set the unsigned long mark value for the packet maintained by the iptables mangle table.

An example of usage follows:

```
iptables -t mangle -A PREROUTING --in-interface eth0 -p tcp \
         -s <some src address> --sport 1024:65535 \
         -d <some destination address> --dport 23 \
         -j MARK --set-mark 0x00010070
```

TOS contains the functionality to set the TOS bits in the IP header.

An example of usage follows:

```
iptables -t mangle -A OUTPUT ...  -j TOS --set-tos <tos>
```

The possible tos values are the same values available in the filter table's TOS match extension module.

Summary

This chapter covered the majority of features available in iptables—certainly, the features most commonly used. I've tried to give a general sense of the differences between Netfilter and IPFW, if for no other reason than to give you a "heads up" for the implementation differences that will appear in the following chapters. The modular implementation divisions of three separate major tables—`filter`, `mangle`, and `nat`—was presented. Within each of these major divisions, features were further broken down into modules that provide target extensions and modules that provide match extensions.

Chapter 4, "Building and Installing a Standalone Firewall," goes through a simple, standalone firewall example. Basic antispoofing, denial of service, and other fundamental rules are presented. The purpose of the chapter isn't to present a general firewall for people to cut and paste for practical use, as much as to demonstrate the syntax presented in this chapter in a functional way.

Subsequent chapters are more specific. User-defined chains, firewall optimization, LAN, NAT, and multihomed hosts are covered separately, as are larger local network architectures.