Security
Firewall – IP Tables

TECHNICAL NOTES

Introduction

Prior to iptables, the predominant software packages for creating Linux firewalls were 'IPChains' in Linux 2.2 and ipfwadm in Linux 2.0, which in turn was based on BSD's ipfw. Both ipchains and ipfwadm alter the networking code so they could manipulate packets, as there was no general packet-control framework until netfilter. Iptables preserves the basic ideas introduced into Linux with ipfwadm: lists of rules which each specified what to match within a packet, and what to do with such a packet. ipchains added the concept of *chains* of rules, and iptables extended this further into *tables*: one table was consulted when deciding whether to NAT a packet, and another consulted when deciding how to filter a packet. In addition, the three points at which filtering is done in a packet's journey were altered, so any packet only passes through one filtering point.

Whereas ipchains and ipfwadm combine packet filtering and NAT (particularly three specific kinds of NAT*masquerading*, *port forwarding* and *redirection*), netfilter makes it possible to separate packet operations into three parts: packet filtering, connection tracking, and Network Address Translation. Each part connects to the netfilter hooks at different points to access packets. The connection tracking and NAT subsystems are more general and more powerful than the stunted versions within ipchains and ipfwadm.

This split allowed iptables, in turn, to use the information which the connection tracking layer had determined about a packet: this information was previously tied to NAT. This makes iptables superior to ipchains because it has the ability to monitor the state of a connection and redirect, modify or stop data packets based on the state of the connection, not just on the source, destination or data content of the packet. A firewall using iptables this way is said to be a Stateful Firewall versus ipchains, which can only create a 'Stateless Firewall' (except in very limited cases). It can be said that ipchains is not aware of the full context from which a data packet arises, whereas iptables is, and therefore iptables can make better decisions on the fate of packets and connections.

Iptables, the NAT subsystem and the connection tracking subsystem are also extensible, and many extensions are included in the base iptables package, such as the iptables extension which allows querying of the connection state mentioned above. Additional extensions are distributed alongside the iptables utility, as patches to the kernel Source Code along with a patch tool called *patch-o-matic*.

Operational Summary

The netfilter framework allows the System Administrator to define rules for how to deal with network packets. Rules are grouped into *chains*—each chain is an ordered list of rules. Chains are grouped into *tables*—each table is associated with a different kind of packet processing.

Each rule contains a specification of which packets match it and a *target* that specifies what to do with the packet if it is matched by that rule. Every network packet arriving at or leaving from the computer traverses at least one chain, and each rule on that chain attempts to match the packet. If the rule matches the packet, the traversal stops, and the rule's target dictates what to do with the packet.

If a packet reaches the end of a predefined chain without being matched by any rule on the chain, the chain's *policy* target dictates what to do with the packet. If a packet reaches the end of a user-defined chain without being matched by any rule on the chain or the user-defined chain is empty, traversal continues on the calling chain (implicit target RETURN). Only predefined chains have policies.

Rules in iptables are grouped into chains. A chain is a set of rules for IP packets, determining what to do with them. Each rule can possibly dump the packet out of the chain (short-circuit), and further chains are not considered.

A chain may contain a link to another chain - if either the packet passes through that entire chain or matches a RETURN target rule it will continue in the first chain. There is no limit to how nested chains can be. There are three basic chains (INPUT, OUTPUT, and FORWARD), and the user can create as many as desired. A rule can merely be a pointer to a chain.

Tables

There are three built-in tables, each of which contains some predefined chains. It is possible for extension modules to create new tables. The administrator can create and delete user-defined chains within any table. Initially, all chains are empty and have a policy target that allows all packets to pass without being blocked or altered in any fashion.

Filter table — This table is responsible for filtering (blocking or permitting a packet to proceed). Every packet passes through the filter table. It contains the following predefined chains, and any packet will pass through one of them:

INPUT chain — All packets destined for this system go through this chain (hence sometimes referred to as *LOCAL_INPUT*)

OUTPUT chain — All packets created by this system go through this chain (aka. *LOCAL_OUTPUT*)

FORWARD chain — All packets merely passing through the system (being routed go through this chain.

Nat table — This table is responsible for setting up the rules for rewriting packet addresses or ports. The first packet in any connection passes through this table: any verdicts here determine how all packets in that connection will be rewritten. It contains the following predefined chains:

PREROUTING chain — Incoming packets pass through this chain before the local routing table is consulted, primarily for DNAT (destination-NAT).

POSTROUTING chain — Outgoing packets pass through this chain after the routing decision has been made, primarily for SNAT (source- NAT)

OUTPUT chain — Allows limited DNAT on locally-generated packets

Mangle table — This table is responsible for adjusting packet options, such as 'Quality of Service' . All packets pass through this table. Because it is designed for advanced effects, it contains all the possible predefined chains:

PREROUTING chain — All packets entering the system in any way, before routing decides whether the packet is to be forwarded (FORWARD chain) or is destined locally (INPUT chain).

INPUT chain — All packets destined for this system go through this chain

FORWARD chain — All packets merely passing through the system go through this chain.

OUTPUT chain — All packets created by this system go through this chain

POSTROUTING chain — All packets leaving the system go through this chain.

In addition to the built-in chains, the user can create any number of user-defined chains within each table, which allows them to group rules logically.

Each chain contains a list of rules. When a packet is sent to a chain, it is compared against each rule in the chain in order. The rule specifies what properties the packet must have for the rule to match, such as the port number or IP Address. If the rule does not match then processing continues with the next rule. If, however, the rule does match the packet, then the rule's *target* instructions are followed (and further processing of the chain is usually aborted). Some packet properties can only be examined in certain chains (for example, the outgoing network interface is not valid in the INPUT chain). Some targets can only be used in certain chains, and/or certain tables (for example, the *SNAT* target can only be used in the POSTROUTING chain of the nat table).

Rule targets

The target of a rule can be the name of a user-defined chain or one of the built-in targets `ACCEPT`, `DROP`, `QUEUE`, or `RETURN`. When a target is the name of a user-defined chain, the packet is diverted to that chain for processing (much like a Subroutine call in a programming language). If the packet makes it through the user-defined chain without being acted upon by one of the rules in that chain, processing of the packet resumes where it left off in the current chain. These inter-chain calls can be nested to an arbitrary depth.

The following built-in targets exist:

ACCEPT

This target causes netfilter to accept the packet. What this means depends on which chain is doing the accepting. For instance, a packet that is accepted on the INPUT chain is allowed to be received by the host, a packet that is accepted on the OUTPUT chain is allowed to leave the host, and a packet that is accepted on the FORWARD chain is allowed to be routed through the host.

DROP

This target causes netfilter to drop the packet without any further processing. The packet simply disappears without any indication of the fact that it was dropped being given to the sending host or application. This frequently appears to the sender as a communication timeout, which can cause confusion (though dropping undesirable inbound packets is often considered a good security policy, because it gives no indication to a potential attacker that your host even exists).

QUEUE

This target causes the packet to be sent to a queue in user space. An application can use the libipq library, which also is part of the netfilter/iptables project, to alter the packet. If there is no application that reads the queue, this target is equal to DROP.

RETURN

According to the official netfilter documentation, this target has the same effect of falling off the end of a chain: for a rule in a built-in chain, the policy of the chain is executed. For a rule in a user-defined chain, the traversal continues at the previous chain, just after the rule which jumped to this chain.

There are many extension targets available. Some of the most common ones are:

REJECT
This target has the same effect as 'DROP' except that it sends an error packet back to the original sender. It is mainly used in the INPUT or FORWARD chains of the filter table. The type of packet can be controlled thorough the '--reject-with' parameter. A rejection packet can explicitly state that the connection has been filtered (an ICMP connection-administratively-filtered packet), though most users prefer that the packet will simply state that the computer does not accept that type of connection (such packet will be a tcp-reset packet for denied TCP connections, an icmp-port-unreachable for denied udp sessions or an icmp-protocol-unreachable for non-tcp non-udp packets). If the '--reject-with' parameter hasn't been specified, the default rejection packet is always icmp-port-unreachable.

LOG
This target logs the packet. This can be used in any chain in any table, and is often used for debugging (such as to see which packets are being dropped).

ULOG
This target logs the packet but not like the LOG target. The LOG target sends information to the Kernel log but ULOG Multicasts the packets matching this rule through a Netlink Socket so that userspace programs can receive this packets by connecting to the socket

DNAT
This target causes the packet's destination address (and optionally port) to be rewritten for Network Address Translation. The '--to-destination' flag must be supplied to indicate the destination to use. This is only valid in the OUTPUT and PREROUTING chains within the nat table. This decision is remembered for all future packets which belong to the same connection, and replies will have their source address and port changed back to the original (ie. the reverse of this packet).

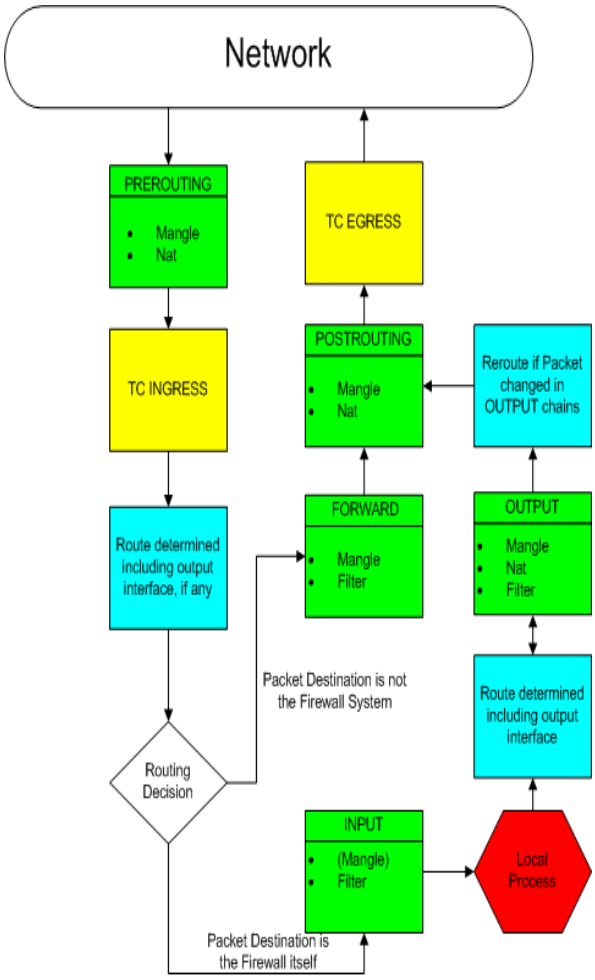
SNAT
This target causes the packet's source address (and optionally port) to be rewritten for Network Address Translation. The '--to-source' flag must be supplied to indicate the source to use. This is only valid in the POSTROUTING chain within the nat table, and like DNAT, is remembered for all other packets belonging to the same connection.

MASQUERADE
This is a special, restricted form of SNAT for dynamic IP addresses, such as most ISPs (Internet Service Providers) provide for Modems or DSL users. Rather than change the SNAT rule every time the IP address changes, this calculates the source IP address to NAT to by looking at the IP address of the outgoing interface when a packet matches this rule. In addition, it remembers which connections used MASQUERADE, and if the interface address changes (such as reconnecting to the ISP), all connections NATted to the old address are forgotten.

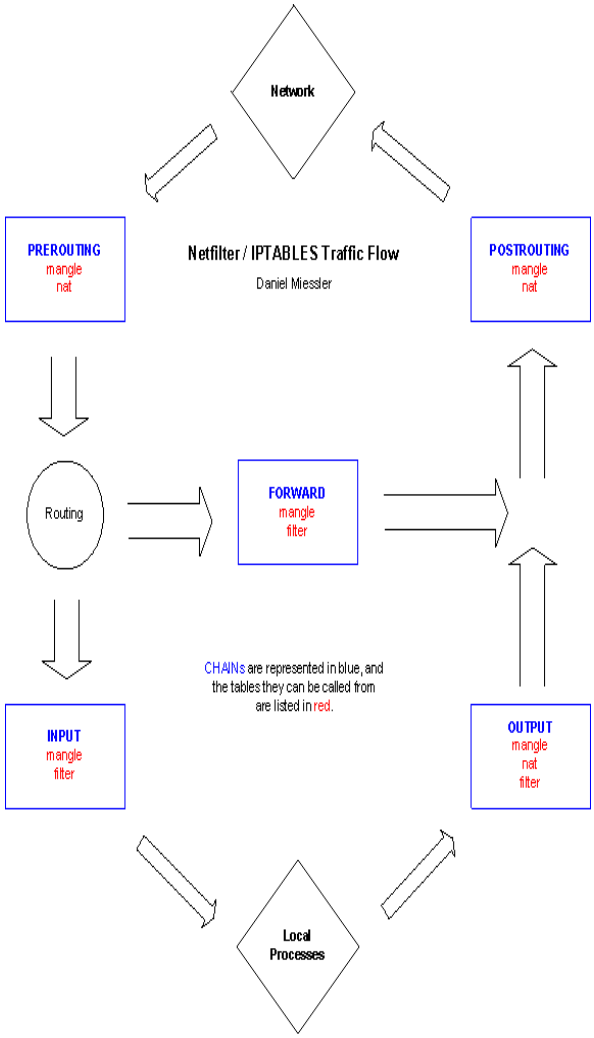
TECHNICAL NOTES

Diagrams

To better understand how a packet traverses the kernel netfilter tables/chains you may find these diagrams useful:



Netfilter Packet Flow



Connection tracking

One of the important features built on top of the netfilter framework is connection tracking. Connection tracking allows the kernel to keep track of all logical network connections or sessions, and thereby relate all of the packets which may make up that connection. NAT relies on this information to translate all related packets in the same way, and iptables can use this information to act as a stateful firewall.

Connection tracking classifies each packet as being in one of four states: NEW (trying to create a new connection), ESTABLISHED (part of an already-existing connection), RELATED (related to, but not actually part of an existing connection) or INVALID (not part of an existing connection, and unable to create a new connection). A normal example would be that the first packet the firewall sees will be classified NEW, the reply would be classified ESTABLISHED and an ICMP error would be RELATED. An ICMP error packet which did not match any known connection would be INVALID.

The connection state is completely independent of any *TCP state*. If the host answers with a SYN ACK packet to

acknowledge a new incoming TCP connection, the TCP connection itself is not yet established but the tracked connection is - this packet will match the state ESTABLISHED.

A tracked connection of a *stateless protocol* like UDP nevertheless has a connection state.

Furthermore, through the use of plugin modules, connection tracking can be given knowledge of application layer protocols and thus understand that two or more distinct connections are "related". For example, consider the FTP protocol. A control connection is established, but whenever data is transferred, a separate connection is established to transfer it. When the ip_conntrack_ftp module is loaded, the first packet of an FTP data connection will be classified `RELATED` instead of `NEW`, as it is logically part of an existing connection.

iptables can use the connection tracking information to make packet filtering rules more powerful and easier to manage. The "state" extension allows iptables rules to examine the connection tracking classification for a packet. For example, one rule might allow NEW packets only from inside the firewall to outside, but allow RELATED and ESTABLISHED in either direction. This allows normal reply packets from the outside (ESTABLISHED), but does not allow new connections to come from the outside to the inside. However, if an FTP data connection needs to come from outside the firewall to the inside, it will be allowed, because the packet will be correctly classified as RELATED to the FTP control connection, rather than a NEW connection.

IP Tables.
iptables is a user space application program that allows a system administrator to configure the netfilter tables, chains, and rules (described above). Because iptables requires elevated privileges to operate, it must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /sbin/iptables. The detailed syntax of the iptables command is documented in its man page, which can be displayed by typing the command "man iptables"

Common options

In each of the iptables invocation forms shown below, the following common options are available:

-t *table* - Makes the command apply to the specified *table*. When this option is omitted, the command applies to the *filter* table by default.

-v - Produces verbose output.

-n - Produces numeric output (i.e., port numbers instead of service names, and IP addresses instead of domain names)

--line-numbers - When listing rules, add line numbers to the beginning of each rule, corresponding to that rule's position in its chain.

Rule-specifications

Most iptables command forms require you to provide a rule-specification, which is used to match a particular subset of the network packet traffic being processed by a chain. The rule-specification also includes a target that specifies what to do with packets that are matched by the rule. The following options are used (frequently in combination with each other) to create a rule-specification.

```
-j target
--jump target
```

Specifies the target of a rule. The target is either the name of a user-defined chain (created using the -N option), one of the built-in targets, ACCEPT, DROP, QUEUE, or RETURN, or an extension target, such as REJECT, LOG, DNAT, or SNAT. If this option is omitted in a rule, then matching the rule will have no effect on a packet's fate, but the counters on the rule will be incremented.

```
-i [!] in-interface
--in-interface [!] in-interface
```

Name of an interface via which a packet is going to be received (only for packets entering the INPUT, FORWARD and PREROUTING chains). When the '!' argument is used before the interface name, the sense is inverted. If the interface name ends in a '+', then any interface which begins with this name will match. If this option is omitted, any interface name will match.

```
-o [!] out-interface
--out-interface [!] out-interface
```

Name of an interface via which a packet is going to be sent (for packets entering the FORWARD, OUTPUT and POSTROUTING chains). When the '!' argument is used before the interface name, the sense is inverted. If the interface name ends in a '+', then any interface which begins with this name will match. If this option is omitted, any interface name will match.

```
-p [!] protocol
--protocol [!] protocol
```

Matches packets of the specified protocol name. If '!' precedes the protocol name, this matches all packets that are not of the specified protocol. Valid protocol names are icmp, udp, tcp... A list of all the valid protocols could be found in the file /etc/protocols.

```
-s [!] source[/prefix]
--source [!] source[/prefix]
```

Matches IP packets coming from the specified source address. The source address can be an IP address, an IP address with associated network prefix, or a hostname. If '!' precedes the source, this matches all packets that are not coming from the specified source.

```
-d [!] destination[/prefix]
--destination [!] destination[/prefix]
```

Matches IP packets going to the specified destination address. The destination address can be an IP address, an IP address with associated network prefix, or a hostname. If '!' precedes the destination, this matches all packets that are not going to the specified destination.

```
--destination-port [!] [port:port
--dport [!] [port:port
```

Matches TCP or UDP packets (depending on the argument to the -p option) destined for the specified port or the range of ports (when the `port:port` form is used). If '!' precedes the port specification, this matches all TCP or UDP packets not destined for the specified port or port range.

```
--source-port [!] [port:port
--sport [!] [port:port
```

Matches TCP or UDP packets (depending on the argument to the -p option) coming from the specified port or the range of ports (when the *port*:*port* form is used). If '!' precedes the port specification, this matches all TCP or UDP packets not coming from the specified port or port range.

```
--tcp-flags [!] mask comp
```

Matches TCP packets having certain TCP protocol flags set or unset. The first argument specified the flags to be examined in each TCP packet, written as a comma-separated list (no spaces allowed). The second argument is a comma-separated list of flags which must be set within those that are examined. The flags are: SYN, ACK, FIN, RST, URG, PSH, ALL, and NONE. Hence, the option "--tcp-flags SYN,ACK,FIN,RST SYN" will only match packets with the SYN flag set and the ACK, FIN and RST flags unset.

```
[!] --syn
```

Matches TCP packets having the SYN flag set and the ACK,RST and FIN flags unset. Such packets are used to initiate TCP connections. Blocking such packets on the INPUT chain will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. This option can be combined with others, such as --source to block or allow inbound TCP connections only from certain hosts or networks. This option is equivalent to "--tcp-flags SYN,RST,ACK SYN". If the '!' flag precedes the --syn, the sense of the option is inverted.

This section is under construction at time of writing.

TECHNICAL NOTES

Invocation

The iptables command has the following invocation forms. Items in braces, {...|...|...}, are required, but only one of the items separated by '|' can be entered. Items in brackets, [...], are optional.

```
iptables { -A | --append | -D | --delete } chain rule-specification [
options ]
```

This form of the command adds (-A or --append) or deletes (-D or --delete) a rule from the specified chain. For example to add a rule to the INPUT chain in the *filter* table (the default table when option -t is not specified) to drop all UDP packets, use this command:

```
iptables -A INPUT -p udp -j DROP
```

To delete the rule added by the above command, use this command:

```
iptables -D INPUT -p udp -j DROP
```

The above command actually deletes the first rule on the INPUT chain that matches the rule-specification "-p udp -j DROP". If there are multiple identical rules on the chain, only the first matching rule is deleted.

```
iptables { -R | --replace | -I | --insert } chain rulenum rule-
specification [ options ]
```

This form of the command replaces (-R or --replace) an existing rule or inserts (-I or --insert) a new rule in the specified chain. For instance, to replace the fourth rule in the INPUT chain with a rule that drops all ICMP packets, use this command:

```
iptables -R INPUT 4 -p icmp -j DROP
```

To insert a new rule in the second slot in the OUTPUT chain that drops all TCP traffic going to port 80 on any host, use this command:

```
iptables -I OUTPUT 2 -p tcp --dport 80 -j DROP
```

```
iptables { -D | --delete } chain rulenum [ options ]
```

This form of the command deletes a rule at the specified numeric index in the specified chain. Rules are numbers starting with 1. For example, to delete the third rule from the FORWARD chain, use this command:

```
iptables -D FORWARD 3
```

```
iptables { -L | --list | -F | --flush | -Z | --zero } [ chain ] [ options ]
```

This form of the command is used to list the rules in a chain (-L or --list), flush (i.e., delete) all rules from a chain (-F or --flush), or zero the byte and packet counters for a chain (-Z or --zero). If no chain is specified, the operation is performed on all chains.
For example, to list the rules in the OUTPUT chain, use this command:

```
iptables -L OUTPUT
```

To flush all chains, use this command:

```
iptables -F
```

To zero the byte and packet counters for the PREROUTING chain in the *nat* table, use this command:

```
iptables -t nat -Z PREROUTING
```

```
iptables { -N | --new-chain } chain iptables { -X | --delete-chain } [ chain ]
```

This form of the command is used to create (-N or --new-chain) a new user-defined chain or to delete (-X or --delete-chain) an existing user-defined chain. If no chain is specified with the -X or --delete-chain options, all user-defined chains are deleted. It is not possible to delete built-in chains, such as the INPUT or OUTPUT chains in the *filter* table.

```
iptables { -P | --policy } chain target
```

This form of the command is used to set the policy target for a chain. For instance, to set the policy target for the INPUT chain to DROP, use this command:

```
iptables -P INPUT DROP
iptables { -E | --rename-chain } old-chain-name new-chain-name
```

This form of the command is used to rename a user-defined chain.