The netfilter framework in Linux 2.4

by

Harald Welte <laforge@gnumonks.org>

Contents

- Introduction
- PART I - Netfilter basics / concepts
- Part II - Packet filtering using iptables and netfilter
- Part III - NAT using iptables and netfilter
- Part IV - Packet mangling using iptables and netfilter
- Advanced netfilter concepts
- Current development and Future

Introduction

What is netfilter

- More than a firewall subsystem
- Generalized Framework (protocol independend)
  - Hooks in the Network stack
  - Multiple kernel modules can register with the hooks
  - Asynchronous packet handling in userspace

Traditional packet filtering / NAT / ... implemented on top of this framework

Introduction

Why did we need netfilter

- No infrastructure for passing packets to userspace
- Transparent proxying extremely difficult
- Packet filter rules depend on interface addresses
- Masquerading and packet filtering not implemented seperately
- Code too complex
- Neither modular nor extensible

Introduction

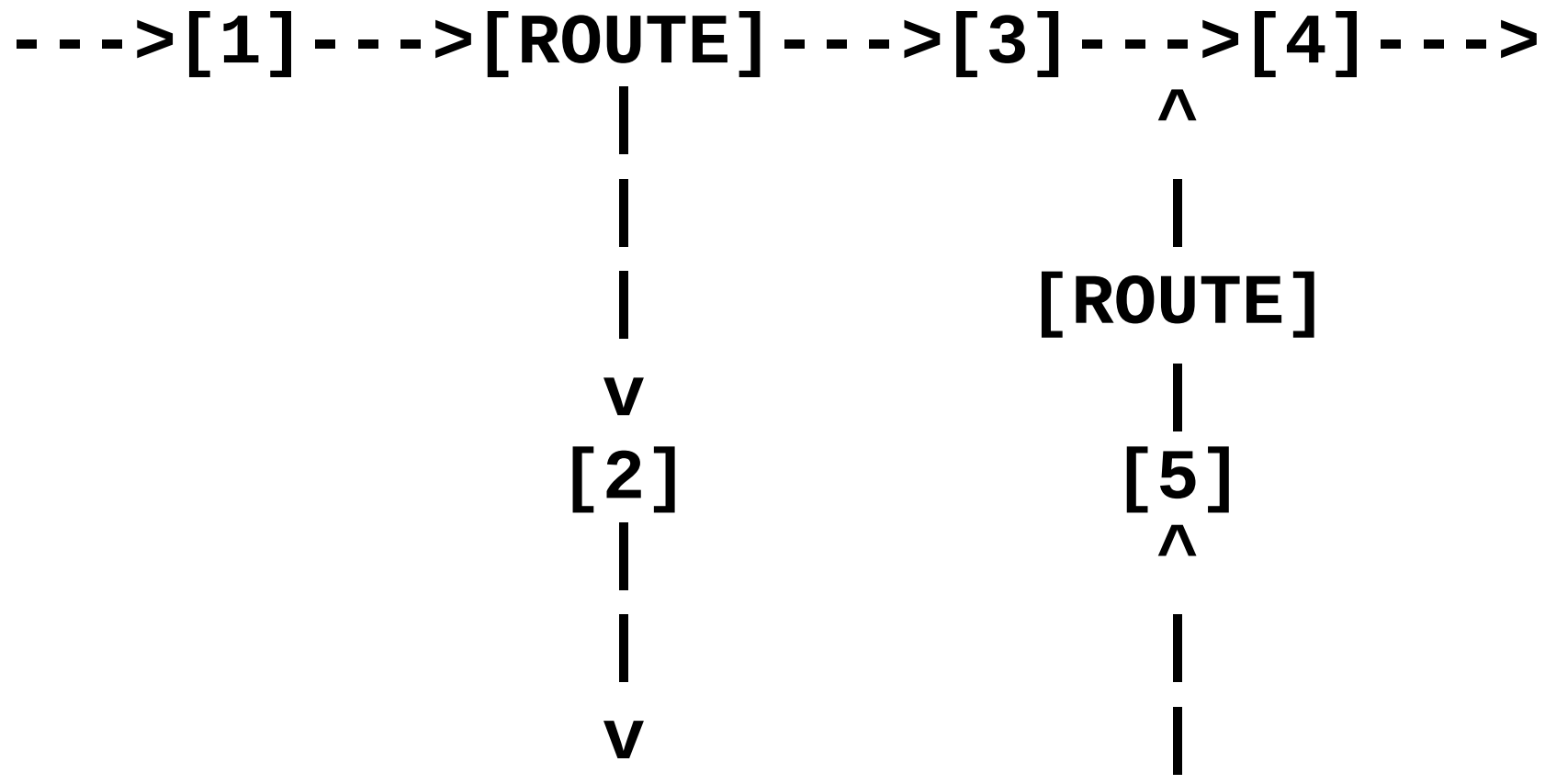
Authors of netfilter

- Paul 'Rusty' Russel
  - co-author of iptables in Linux 2.2
  - was paid by Watchguard for about one Year of development
  - now works for Linuxcare
- James Morris
  - userspace queuing (kernel, library and tools)
  - REJECT target
- Marc Boucher
  - NAT and packet filtering controlled by one comand
  - Mangle table
- Non-core team contributors
  - Philip Blundell (IPv6 integration)
  - Jozsef Kadlecsik (full TCP window tracking)
  - Harald Welte (IRC NAT helper, userspace logging)
- lots of people I probably forgot

see http://netfilter.kernelnotes.org/scoreboard.html

PART I - Netfilter basics

Netfilter architecture in IPv4



1=NF_IP_PRE_ROUTING
2=NF_IP_LOCAL_IN
3=NF_IP_FORWARD
4=NF_IP_POST_ROUTING

PART I - Netfilter basics

Netfilter base

A Module registered to a hook has to return one of the following constants

- NF_ACCEPT
  - continue traversal as normal
- NF_DROP
  - drop the packet, do not continue
- NF_STOLEN
  - I’ve taken over the packet do not continue
- NF_QUEUE
  - enqueue packet to userspace
- NF_REPEAT
  - call this hook again

PART I - Netfilter basics

Packet selection using IP tables

Most modules registering for one of the netfilter hooks also use the generic IP tables infrastructure.

The three major parts of 2.4 advanced packet handling are implemented using IP tables

- Packet filtering table ’filter’
- NAT table ’nat’
- Packet mangling table ’mangle’

PART I - Netfilter basics

Connection tracking

Implemented seperately from NAT to Provide a basis for statefule filtering using the ’state’ match described later.

The generic conntrack infrastructure is extensible via

- protocol helpers (currently TCP/UDP/ICMP)
- application helpers (currently FTP and IRC-DCC)

Conntrack divides packets in the following four categories:

- NEW - would establish new connection
- ESTABLISHED - part of already established connection
- RELATED - is related to established connection
- INVALID - (multicast, errors...)

PART II - packet filtering

Overview

Packet filtering in IPv4 is implemented on top of three netfilter hooks

- NF_IP_LOCAL_IN (packets destined for the local host)
- NF_IP_FORWARD (packets forwarded by us)
- NF_IP_LOCAL_OUT (packets from the local host)

Each packet passes exactly one of the three hooks. Note that this is very different compared to the old 2.2 ipchains behaviour.

On each of the three hooks is a chain (INPUT, FORWARD, OUTPUT) implemented on top of the IP table "filter"

PART II - packet filtering

Managing chains and tables

Each rule in a chain consists out of

- match (which packet match this rule)
- target (what to do if the rule is matched)

matches and targets can either be builtin or implemented as kernel modules

The userspace tool iptables is very flexible

- handles all different kinds of IP tables
- supports a plugin/shlib interface for target / match specific options

PART II - packet filtering

Basic iptables commands

To build a complete iptable command, we must specify

- which table to work with
- which chain in this table to use
- an operation (insert, add, delete, modify)
- a match
- a target

The syntax is

```
iptables -t table -Operation chain -j target match(es)
```

Example:

```
iptables -t filter -A INPUT -j ACCEPT -p tcp --dport smtp
```

PART II - packet filtering

Targets

Builtin Targets to be used in filter table

- ACCEPT accept the packet
- DROP silently drop the packet
- QUEUE enqueue packet to userspace
- RETURN return to previous (calling) chain
- foobar user defined chain

Targets implemented as loadable modules

- REJECT drop the packet but inform sender
- MIRROR change source/destination IP and resend
- LOG log via syslog
- ULOG log via userspace

PART II - packet filtering

Matches

Basic matches

- -p protocol (tcp/udp/icmp/...)
- -s source address (ip/mask)
- -d destination address (ip/mask)
- -i incoming interface
- -o outgoint interface

Match extensions

- --dport destination port
- --sport source port
- --state (ESTABLISHED/RELATED/NEW/INVALID)
- --mac-source source MAC address
- --mark nfmark
- --tos
- --limit rate limiting (n packets per timeframe)

PART III - NAT

Overview

- Previous Linux Kernels only implemented one special case of NAT: Masquerading
- Netfilter enables Linux to do any kind of NAT.
- All matches from packet filtering are available for the nat tables, too
- We divide NAT into ’source NAT’ and ’destination NAT’
  - SNAT changes the packet’s source whille passing NF_IP_POST_ROUTING
  - DNAT changes the packet’s destination while passing NF_IP_PRE_ROUTING
  - MASQUERADE is a special case of SNAT
  - REDIRECT is a special case of DNAT

PART III - NAT

Source NAT

- SNAT Example:

```
iptables -t nat -A POSTROUTING -j SNAT --to-source 1.2.3.4
```

Masquerading does almost the same as SNAT, but if the outgoing interfaces address changes (in case we have a dialup with dynamic ip), the new address is used.

- MASQUERADE Example:

```
iptables -t nat -A POSTROUTING -j MASQUERADE -o ppp0
```

PART III - NAT

Destination NAT

- DNAT example:

```
iptables -t nat -A PREROUTING -j DNAT --to-destination 1.2.3.4:8080 -p tcp
--dport 80 -i eth1
```

REDIRECT is a special case of DNAT, which alters the destination to the address of the incoming interface.

- REDIRECT example:

```
iptables -t nat -A PREROUTING -j REDIRECT --to-port 3128 -i eth1 -p tcp
--dport 80
```

PART IV - Packet mangling

Packet mangling enables us to change certain parts of a packet based on rules in IP tables.
Of course, we again have all the matches available, as described in the packet filtering section.

Currently, the supported packet mangling targets are:

- TOS - manipulate the TOS bits
- MARK - change the nfmark field of the skb

Simple example:

```
iptables -t mangle -A PREROUTING -j MARK --set-mark 10 -p tcp --dport 80
```

Advanced Netfilter concepts

- Connection tracking
- Userspace logging
  - flexible replacement for old syslog-based logging
  - packets to userspace via multicast netlink sockets
  - easy-to-use library (libipulog)
  - plugin-extensible userspace logging daemon already available
- Queuing
  - reliable asynchronous packet handling
  - packets to userspace via unicast netlink socket
  - easy-to-use library (libipq)
  - experimental queue multiplex daemon (ipqmpd)

Current Development and Future

Netfilter (although it proved very stable) is still work in progress.

Areas of current development

- provide an unique libnfnetlink (like librtnetlink)
- infrastructure for conntrack/nat helpers in userspace
- full TCP sequence number tracking
- multicast support
- more flexible matches (MAXCONN, ...)
- more NAT modules (RPC, SNMP, SMB, ...)
- more IPv6 matches / targets