netfilter/iptables training

Nov 05/06/07, 2007
Day 1

by

Harald Welte <laforge@netfilter.org>

Contents

- Day 1

- Introduction
- Highly Scalable Linux Network Stack
- Netfilter Hooks
- Packet selection based on IP Tables
- The Connection Tracking Subsystem
- The NAT Subsystem
- Packet Mangling

Introduction

Who is speaking to you?

- an independent Free Software developer
- who earns his living off Free Software since 1997
- who is one of the authors of the Linux kernel firewall system called netfilter/iptables
- [who can claim to be the first to have enforced the GNU GPL in court]

Introduction

Linux and Networking

- Linux is a true child of the Internet
- Early adopters: ISP’s, Universities
- Lots of work went into a highly scalable network stack
- Not only for client/server, but also for routers
- Features unheared of in other OS’s

Introduction

Did you know, that a stock 2.6.x linux kernel can provide

- □ a stateful packet filter ?
- □ fully symmetric NA(P)T ?
- □ policy routing ?
- □ QoS / traffic shaping ?
- □ IPv6 firewalling ?
- □ packet filtering, NA(P)T on a bridge ?
- □ layer 2 (mac) address translation ?
- □ packet forwarding rates of up to 2.1Mpps ?

Introduction

Why did we need netfilter/iptables?
Because ipchains...

- ○ has no infrastructure for passing packets to userspace
- ○ makes transparent proxying extremely difficult
- ○ has interface address dependent Packet filter rules
- ○ has Masquerading implemented as part of packet filtering
- ○ code is too complex and intermixed with core ipv4 stack
- ○ is neither modular nor extensible
- ○ only barely supports one special case of NAT (masquerading)
- ○ has only stateless packet filtering

Introduction

Who’s behind netfilter/iptables

- □ The core team
  - ○ Paul ’Rusty’ Russel
    - ▷ co-author of iptables in Linux 2.2
  - ○ James Morris
  - ○ Marc Boucher
  - ○ Harald Welte
  - ○ Jozsef Kadlecsik
  - ○ Martin Josefsson
  - ○ Patrick McHardy

Netfilter Hooks

- □ What is netfilter?
  - ○ System of callback functions within network stack
  - ○ Callback function to be called for every packet traversing certain point (hook) within network stack
  - ○ Protocol independent framework
  - ○ Hooks in layer 3 stacks (IPv4, IPv6, DECnet, ARP)
  - ○ Multiple kernel modules can register with each of the hooks

Traditional packet filtering, NAT, ... is implemented on top of this framework

Can be used for other stuff interfacing with the core network stack, like DECnet routing daemon.

Netfilter Hooks

Netfilter architecture in IPv4

```
in --->[1]--->[ROUTE]--->[3]--->[4]---> out
              |            ^
              |            |
              |         [ROUTE]
              v            |
             [2]          [5]
              |            ^
              |            |
              v            |
1=NF_IP_PRE_ROUTING
2=NF_IP_LOCAL_IN
3=NF_IP_FORWARD
4=NF_IP_POST_ROUTING
5=NF_IP_LOCAL_OUT
```

Netfilter Hooks

Netfilter Hooks

- Any kernel module may register a callback function at any of the hooks
- The module 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

IP tables

- Packet selection using IP tables
  - The kernel provides generic IP tables support
  - Each kernel module may create it’s own IP table
  - The four major parts of the firewalling subsystem are implemented using IP tables
    - Packet filtering table ’filter’
    - NAT table ’nat’
    - Packet mangling table ’mangle’
    - The ’raw’ table for conntrack exemptions

IP Tables

- Managing chains and tables
  - An IP table consists out of multiple chains
  - A chain consists out of a list of rules
  - Every single rule in a chain consists out of
    - match[es] (rule executed if all matches true)
    - target (what to do if the rule is matched)
    - implicit packet and byte counter

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

- The userspace tool iptables is used to control IP tables
  - handles all different kinds of IP tables
  - supports a plugin/shlib interface for target/match specific options

IP Tables

Basic iptables commands

- □ To build a complete iptables command, we must specify
  - ○ which table to work with
  - ○ which chain in this table to use
  - ○ an operation (insert, add, delete, modify)
  - ○ one or more matches (optional)
  - ○ 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
```

IP Tables

Matches

- ○ Basic matches
  - ▹ -p protocol (tcp/udp/icmp/...)
  - ▹ -s source address (ip/mask)
  - ▹ -d destination address (ip/mask)
  - ▹ -i incoming interface
  - ▹ -o outgoing interface

IP Tables

- addrtype match
  - matches source/destionation address type
  - types are UNICAST/LOCAL/BROADCAST/ANYCAST/MULTICAST/...
- ah match
  - matches IPSEC AH SPI (range)
- comment match
  - always matches, allows user to place comment in rule
- connmark match
  - connection marking, see later
- conntrack match
  - more extended version of ’state’
  - match on timeout, fine-grained state, original tuples
- dscp match
  - matches DSCP codepoint (formerly-known as TOS bits)

IP Tables

- ecn match
  - matches ECN bits of tcp and ip header
- esp match
  - matches IPSEC ESP SPI (range)
- hashlimit match
  - dynamic limiting
- helper match
  - allows matching of conntrack helper name
- iprange match
  - match on arbitrary IP address ranges (not a mask)

IP Tables

- length match
  - match on packet length
- limit
  - static rate limiting
- mac
  - match on source mac address
- mark
  - match on nfmark (fwmark)
- multiport
  - match on multiple ports

IP Tables

- owner
  - match on socket owner (uid, gid, pid, sid, command name)
- physdev
  - match underlying device in case of bridge
- pkttype
  - match link-layer packet type (unicast,broadcast,multicast)
- realm
  - match routing realm
- recent
  - see special section below
- tcpmss
  - match on TCP maximum segment size

IP Tables

Targets

- very dependent on the particular table

- Table specific targets will be discussed later

- Generic Targets, always available
    - ACCEPT accept packet within chain
    - DROP silently drop packet
    - QUEUE enqueue packet to userspace
    - LOG log packet via syslog
    - ULOG log packet via ulogd
    - RETURN return to previous (calling) chain
    - foobar jump to user defined chain

Packet Filtering

Overview

- Implemented as ’filter’ table
- Registers with three netfilter hooks
    - NF_IP_LOCAL_IN (packets destined for the local host)
    - NF_IP_FORWARD (packets forwarded by local host)
    - NF_IP_LOCAL_OUT (packets from the local host)

Each of the three hooks has attached one chain (INPUT, FORWARD, OUTPUT)

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

Packet Filtering

Targets available within ’filter’ table

- □ 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

Connection Tracking Subsystem

- □ Connection tracking...
  - ○ implemented seperately from NAT
  - ○ enables stateful filtering
  - ○ implementation
    - ▷ hooks into NF_IP_PRE_ROUTING to track packets
    - ▷ hooks into NF_IP_POST_ROUTING and NF_IP_LOCAL_IN to see if packet passed filtering rules
    - ▷ protocol modules (currently TCP/UDP/ICMP/SCTP)
    - ▷ application helpers currently (FTP,IRC,H.323,talk,SNMP)

Connection Tracking Subsystem

- □ Connection tracking...
  - ○ 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...)
  - ○ does _NOT_ filter packets itself
  - ○ can be utilized by iptables using the ’state’ match
  - ○ is used by NAT Subsystem

Connection Tracking Subsystem

- □ State tracking for TCP is obvious
  - ○ TCP inherently stateful
  - ○ Two TCP state machines on each end have well-defined behaviour
  - ○ Passive tracking of state machines
  - ○ In more recent 2.6.x kernels, tracking of TCP window (seq/ack)
  - ○ Max idle timeout of fully-established session: 5 days

Connection Tracking Subsystem

- State tracking for UDP: How is this possible?
  - UDP itself not stateful at all
  - However, higher-level protocols mostly match request-reply
  - First packet (request) is assumed to be NEW
  - First matching reply packet is assumed to confirm connection
  - Further packets in either direction refresh timeout
  - Timeouts: 30sec unreplied, 180sec confirmed

Connection Tracking Subsystem

- State tracking on ICMP: What’s that?
  - ICMP Errors (e.g. host/net unreachable, ttl exceeded)
    - They can always be categorized as RELATED to other connections
  - ICMP request/reply (ECHO REQUEST, INFO REQUEST)
    - can be treated like UDP request/reply case

Connection Tracking Subsystem

- State tracking on SCTP: What’s SCTP?
  - Streaming Control Transfer Protocol
  - Linux has SCTP in the network stack, so why should the packet filter not support it?
  - Pretty much like TCP in most cases
  - Doesn’t support more advanced features such as failover of an endpoint

Connection Tracking Subsystem

- State tracking on other protocols
  - ’generic’ protocol: no layer-4 tuple information
  - ’gre’ helper in patch-o-matic

- State tracking of higher-layer protocols
  - implemented as ’connection tracking helpers’
  - currently in-kernel: amanda, ftp, irc, tftp
  - currently in patch-o-matic: pptp, h.323, sip, quake, ...
  - have to be explicitly loaded (ip_conntrack_*.[k]o)
  - work by issuing so-called "expectations"

Connection Tracking Subsystem

- Exemptions to connection tracking
  - Usually connection tracking is called first in PRE_ROUTING
  - Sometimes, filtering is preferred before this conntrack lookup
    - Therefore, the "raw" table was introduced
  - In some rare cases, one might want to not track certain packets
    - The NOTRACK can be used in the "raw" table

Connection Tracking Subsystem

- Configuration / Tuning
  - module parameter "hashsize"
    - number of hash table buckets
  - /proc/sys/net/ipv4/ip_conntrack_max
    - maximum number of tracked connections
  - /proc/sys/net/ipv4/ip_conntrack_buckets (read-only)
    - number of hash table buckets
  - /proc/net/ip_conntrack
    - list of connections
  - /proc/net/ip_conntrack_expect
    - list of pending expectations

Connection Tracking Subsystem

- □ Configuration / Tuning
  - ○ /proc/sys/net/ip_conntrack_log_invalid
    - ▹ log invalid packets?
  - ○ /proc/sys/net/ip_conntrack_tcp_be_liberal
    - ▹ basically disables window tracking, if "1"
  - ○ /proc/sys/net/ip_conntrack_tcp_loose
    - ▹ how many packets required until sync in case of pickup
    - ▹ if set to zero, disables pickup
  - ○ /proc/sys/net/ip_conntrack_tcp_max_retrans
    - ▹ maximum number of retransmitted packets without seeing a n ACK
  - ○ /proc/sys/net/ip_conntrack_*timeout*
    - ▹ timeout values of respective protocol states

Network Address Translation

- □ Network Address Translation
  - ○ Previous Linux Kernels only implemented one special case of NAT: Masquerading
  - ○ Linux 2.4.x / 2.6.x can do any kind of NAT.
  - ○ NAT subsystem implemented on top of netfilter, iptables and conntrack
  - ○ Following targets available within ’nat’ Table
    - ▹ 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
    - ▹ SAME
    - ▹ NETMAP

Network Address Translation

- Source NAT
  - SNAT Example:

```
iptables -t nat -A POSTROUTING -j SNAT --to-source 1.2.3.4 -s 10.0.0.0/8
```

  - MASQUERADE Example:

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

- 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 example

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

Packet Mangling

- Purpose of ’mangle’ table
  - packet manipulation except address manipulation

- Integration with netfilter
  - ’mangle’ table hooks in all five netfilter hooks
  - priority: after conntrack

Simple example:

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

Packet Mangling

- Targets specific to the ’mangle’ table:
  - DSCP
    - manipulate DSCP field
  - ECN
    - manipulate ECN bits
  - IPV4OPTSSTRIP
    - strip IPv4 options
  - MARK
    - change the nfmark field of the skb
  - TCPMSS
    - set TCP MSS option
  - TOS
    - manipulate the TOS bits
  - TTL
    - set / increase / decrease TTL field
  - CLASSIFY
    - classify packet (for tc/iproute)
  - CONNMARK
    - set mark of connection

The raw Table

- Purpose of ’raw’ table
  - to allow for filtering rules _before_ conntrack
- Targets specific to the ’raw’ table:
  - NOTRACK
    - don’t do connection tracking

- The table can also be useful for flood protection rules that happen before traversing the (computational) expensive connection tracking subsystem.

Advanced Netfilter concepts

- 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 (ulogd)
  - Can even be used to directly log into MySQL

- Queuing
  - reliable asynchronous packet handling
  - packets to userspace via unicast netlink socket
  - easy-to-use library (libipq)
  - provides Perl bindings
  - experimental queue multiplex daemon (ipqmpd)

Advanced Netfilter concepts

- Firewalling on a Bridge (ebtables + iptables)
  - totally transparent to layer 2 and above
  - no attack vector since firewall has no IP address
  - even possible to do NAT on the bridge
  - or even NAT of MAC addresses
- □

- ipset - Faster matching
  - iptables are a linear list of rules
  - ipset represents a ’group’ scheme
  - Implements different data types for different applications
    - hash table (for random addresses)
    - bitmask (for let’s say a /24 network)

Advanced Netfilter concepts

- ipv6 packet filtering
  - ip6tables almost identical to iptables
  - no connection tracking in mainline yet, but patches exist
  - ip6_conntrack
    - initial copy+paste ’port’ by USAGI
    - was not accepted because of code duplication
  - nf_conntrack
    - generalized connection tracking, supports ipv4 and ipv6
    - mutually exclusive with ip_conntrack
    - as of now, no ipv4 nat on to of nf_conntrack

Thanks

- Thanks to
  - the BBS scene, Z-Netz, FIDO, ...
    - for heavily increasing my computer usage in 1992
  - KNF (http://www.franken.de/)
    - for bringing me in touch with the internet as early as 1994
    - for providing a playground for technical people
    - for telling me about the existance of Linux!
  - Alan Cox, Alexey Kuznetsov, David Miller, Andi Kleen
    - for implementing (one of?) the world’s best TCP/IP stacks
  - Paul ’Rusty’ Russell
    - for starting the netfilter/iptables project
    - for trusting me to maintain it today
  - Astaro AG
    - for sponsoring parts of my netfilter work