The future of Linux packet filtering

by

Harald Welte <laforge@netfilter.org>
Future of Linux packet filtering

Contents

- Problems with current 2.4/2.5 netfilter/iptables
  - Solution to code replication
  - Solution for dynamic rulesets
  - Solution for API to GUI’s and other management programs

- Other current work
  - Optimizing Rule load time of large rulesets
  - Making netfilter/iptables compatible with zerocopy tcp

- HA for stateful firewalling
  - What’s special about firewalling HA
  - Poor man’s failover
  - Real state replication
Problems with 2.4.x netfilter/iptables

- Code replication between iptables/ip6tables/arptables
  - iptables was never meant for other protocols, but people did copy+paste 'ports'
  - Replication of
    - Core kernel code
    - Layer 3 independent matches (mac, interface, ...)
    - Userspace library (libiptc)
    - Userspace tool (iptables)
    - Userspace plugins (libipt_xxx.so)

- Doesn't suit the needs for dynamically changing rulesets
  - Dynamic rulesets becoming more common due (service selection, IDS)
  - A whole table is created in userspace and sent as blob to kernel
  - For every ruleset the table needs to be copied to userspace and back
  - Inside kernel consistency checks on whole table, loop detection

- Too extensible for writing any forward-compatible GUI
  - New extensions showing up all the time
  - A frontend would need to know about the options and use of a new extension
  - Thus frontends are always incomplete and out-of-date
  - No high-level API other than piping to iptables-restore
Future of Linux packet filtering

Reducing code replication

- code replication is a real problem: unclean, bugfixes missed
- we need layer 3 independent layer for
  - submitting rules to the kernel
  - traversing packet-rulesets supporting match/target modules
  - registering matches/targets
    - layer 3 specific (like matching ipv4 address)
    - layer 3 independent (like matching MAC address)

- solution
  - pkt_tables inside kernel
    - pkt_tables_ipv4 registers layer 3 handler with pkt_tables
    - pkt_tables_ipv6 registers layer 3 handler with pkt_tables
    - everybody registering a pkt_table (like iptable_filter) needs to specify the l3 protocol
  - libraries in userspace (see later)
Future of Linux packet filtering

Supporting dynamic rulesets

- atomic table-replacement turned out to be bad idea
- need new interface for sending individual rules to kernel
- policy routing has the same problem and good solution: rtnetlink
- solution: nfnetlink
  - multicast-netlink based packet-oriented socket between kernel and userspace
  - has extra benefit that other userspace processes get notified of rule changes
    [just like routing daemons]
- nfnetlink will be low-layer below all kernel/userspace communication
  - pkttnetlink [aka iptnetlink]
  - ctnetlink
  - ulog
  - ip_queue
Future of Linux packet filtering

Communication with other programs

whole set of libraries
- libnfnetlink for low-layer communication
- libpkttnetlink for rule modifications
  - will handle all plugins [which are currently part of iptables]
  - query functions about available matches/targets
  - query functions about parameters
  - query functions for help messages about specific match/parameter of a match
  - generic structure from which rules can be built
  - conversion functions to parse generic structure into in-kernel structure
  - conversion functions to parse kernel structure into generic structure
  - functions to convert generic structure in plain text

- libipq will stay API-compatible to current version
- libipulog will stay API-compatible to current version
- libiptc will go away [compatibility layer extremely difficult]
Future of Linux packet filtering

Optimizing rule load time

☐ Current situation
  ○ loading 10,000 rules in 1,000 chains takes about 4 minutes on a PIII 733Mhz
  ○ this is caused by two bottlenecks
    ▶ loop detection algorithm on kernel side inefficient
    ▶ a couple of $O^2$ complexity functions in libiptc

☐ Solution
  ○ efficient loop detection and mark_source_chains() algorithm (graph coloring)
  ○ current CVS libiptc with only one $O^2$ function: 2minutes37
  ○ whole reimplementation of libiptc needed for removing the last $O^2$ function
Optimizing the connection tracking code

- Conntrack hash function optimization
  - old hash function not good for even hash bucket count
  - hash function evaluation tool [cttest] available
  - other hash functions in development (already in 2.4.21)
  - introduce per-system randomness to prevent hash attack
  - code optimization (locking/timers/...)

HA for netfilter/iptables
Future of Linux packet filtering

**netfilter and zero-copy TCP**

- **Current situation (2.4.x)**
  - skb_linearize() at each netfilter hook effectively prevents zero-copy TCP to work if netfilter/iptables is enabled
  - this is a big performance loss on stand-alone servers which filter packets locally

- **Solution**
  - remove skb_linearize() from conntrack, nat and ip_tables core
  - all iptables extensions and conntrack/nat helpers have to use skb_copy_bits() if they want to access data beyond layer 4 header
Introduction

What is special about firewall failover?

- Nothing, in case of the stateless packet filter
  - Common IP takeover solutions can be used
    - VRRP
    - Hartbeat

- Distribution of packet filtering ruleset no problem
  - can be done manually
  - or implemented with simple userspace process

- Problems arise with stateful packet filters
  - Connection state only on active node
  - NAT mappings only on active node
HA for netfilter/iptables

Poor man’s failover

**principle**
- let every node do it’s own tracking rather than replicating state

**two possible implementations**
- connect every node to shared media (i.e. real ethernet)
  - forwarding only turned on on active node
  - slave nodes use promiscuous mode to sniff packets
- copy all traffic to slave nodes
  - active master needs to copy all traffic to other nodes
  - disadvantage: high load, sync traffic == payload traffic
  - IMHO stupid way of solving the problem

**advantages**
- very easy implementation
  - only addition of sniffing mode to conntrack needed
  - existing means of address takeover can be used
- same load on active master and slave nodes
- no additional load on active master

**disadvantages**
- can only be used with real shared media (no switches, ...)
- can not be used with NAT

**remaining problem**
- no initial state sync after reboot of slave node!
Real state replication

Parts needed
- state replication protocol
  - multicast based
  - sequence numbers for detection of packet loss
  - NACK-based retransmission
  - no security, since private ethernet segment to be used
- event interface on active node
  - calling out to callback function at all state changes
- exported interface to manipulate conntrack hash table
- kernel thread for sending conntrack state protocol messages
  - registers with event interface
  - creates and accumulates state replication packets
  - sends them via in-kernel sockets api
- kernel thread for receiving conntrack state replication messages
  - receives state replication packets via in-kernel sockets
  - uses conntrack hashtable manipulation interface
Real state replication

Flow of events in chronological order:

- on active node, inside the network RX softirq
  - connection tracking code is analyzing a forwarded packet
  - connection tracking gathers some new state information
  - connection tracking updates local connection tracking database
  - connection tracking sends event message to event API

- on active node, inside the conntrack-sync kernel thread
  - conntrack sync daemon receives event through event API
  - conntrack sync daemon aggregates multiple event messages into a state replication protocol message, removing possible redundancy
  - conntrack sync daemon generates state replication protocol message
  - conntrack sync daemon sends state replication protocol message

- on slave node(s), inside network RX softirq
  - connection tracking code ignores packets coming from the interface attached to the private conntrack sync network
  - state replication protocol messages is appended to socket receive queue of conntrack-sync kernel thread

- on slave node(s), inside conntrack-sync kernel thread
  - conntrack sync daemon receives state replication message
  - conntrack sync daemon creates/updates conntrack entry
Neccessary changes to kernel

Neccessary changes to current conntrack core

- event generation (callback functions) for all state changes

- conntrack hashtable manipulation API
  - is needed (and already implemented) for ’ctnetlink’ API

- conntrack exemptions
  - needed to _not_ track conntrack state replication packets
  - is needed for other cases as well
  - currently being developed by Jozsef Kadlecsik
Thanks

- The slides of this presentation are available at http://www.gnumonks.org/

- Visit the netfilter homepage http://www.netfilter.org/

Thanks to

- the BBS people, Z-Netz, FIDO, ...
  - for heavily increasing my computer usage in 1992

- KNF
  - 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 most of my current netfilter work