The future of Linux packet filtering targeted for kernel 2.6

by

Harald Welte <laforge@gnumonks.org>

Contents

- Problems with current 2.4.x netfilter/iptables
  - Solution to code replication
  - Solution for dynamic rulesets
  - Solution for API to GUI's and other management programs
- HA for stateful firewalling
  - What's special about firewalling HA
  - Poor man's failover
  - Real state replication
- Other current work

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 becomming 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  - thus frontends are always incomplete and out-of-date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

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)

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-orinented 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

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 avaliable 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 functiosn to perse 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]

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

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)
  - application helpers currently (FTP,IRC,H.323,talk,SNMP)
- 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

Common structures

- struct ip_conntrack_tuple, representing unidirectional flow
  - layer 3 src + dst
  - layer 4 protocol
  - layer 4 src + dst

- connetions represented as struct ip_conntrack
  - original tuple
  - reply tuple
  - timeout
  - l4 state private data
  - app helper
  - app helper private data
  - expected connections

Connection Tracking Subsystem

Flow of events for new packet

- packet enters NF_IP_PRE_ROUTING
  - tuple is derived from packet
  - lookup conntrack hash table with hash(tuple) -> fails
  - new ip_conntrack is allocated
    - fill in original and reply == inverted(original) tuple
    - initialize timer
    - assign app helper if applicable
    - see if we've been expected -> fails
    - call layer 4 helper 'new' function
- ...
- packet enters NF_IP_POST_ROUTING
  - do hashtable lookup for packet -> fails
  - place struct ip_conntrack in hashtable

Connection Tracking Subsystem

Flow of events for packet part of existing connection

- packet enters NF_IP_PRE_ROUTING
  - tuple is derived from packet
  - lookup conntrack hash table with hash(tuple)
  - assosiate conntrack entry with skb->nfct
  - call l4 protocol helper 'packet' function
    - do l4 state tracking
    - update timeouts as needed [i.e. TCP TIME_WAIT,...]
- ...

- packet enters NF_IP_POST_ROUTING
  - do hashtable lookup for packet -> succeds
  - do nothing else

Poor man's failover

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 conntrac 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

Other current work

- optimizing the conntrack code
  - hash function optimization
    - current hash function not good for even hash bucket count
    - other hash functions in development
    - hash function evaluation tool [cttest] avaliable
    - introduce per-system randomness to prevent hash attack
  - code optimization (locking/timers/...)

- getting our work submitted into the mainstream kernel
  - turns out to be more difficult
    - e.g. newnat api now waiting for three months

- discussions about multiple targets/actions per rule
  - technical implementation easy
  - however, not everybody convinced that it fits into the concept

- using tc for firewalling
  - Jamal Hadi Selim uses iptables targets from within TC
  - leads to discussion of generic classification engine API in kernel

Thanks

- The slides and the an according paper of this presentation are available at http://www.gnumonks.org/
- 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 parts of my netfilter work