High speed firewalls with ipset

József Kadlecsik
<kadlec@blackhole.kfki.hu>
HAS Wigner Research Centre for Physics
Content

• Why we need an additional tool?
• Properties of ipset
• ipset and iptables examples
Special firewalls

• High number of rules
  – Fast evaluation algorithm
• Need to change the rules often
  – The rule storage method must support fast modifications
Special firewalls: iptables?

- High number of rules: slow
  - Linear evaluation
- Change rules: slow/inefficient
  - **All** rule must be sent back and forth between kernel-userspace for the sake of changing a **single** rule
  - `iptables-restore`
Special firewalls: nf-hipac?

- High number of rules: fast
  - Complex evaluation algorithm
- Change rules: fast
  - Just the change is sent
- Dead, last modification: 2005.
Special firewalls: ipset?

- High number of rules: fast
  - Simple evaluation algorithms
- Change rules: fast
  - Just the change is sent, simple storage methods
- From 2.6.39 it's in the vanilla kernel
Real comparisons I.

- Netfilter performance testing (2005)
  - Dual Intel Xeon 2.4GHz
  - 2GB DDR RAM, 200MHz
  - ServerWorks GC-LE chipset
  - Intel 82545EM Gbit Ethernet, 64bit, 66MHz
iptables, nf-hipac and ipset I.
iptables, nf-hipac and ipset II.
Real comparisons II.

- Mass blocking IP addresses with ipset (2012)
  http://daemonkeeper.net/781/mass-blocking-ip-addresses-with-ipset/
ipset and ippool

- Joakim Axelsson: ippool, bitmap type
- Joakim Axelsson, Patrick Schaaf and Martin Josefsson: modular ippool, bitmap and macipmap types
- ipset: rewritten ippool, more types
- ipset 6.x: ipset rewritten from the ground
ipset

- In-kernel data sets: IP addresses, protocol/port numbers, MAC addresses, interfaces and several combinations
- Different storage methods: *bitmap*, *hash*, *list*
- Program to handle the sets: *ipset*
- Use the sets from iptables: *set* match and *SET* target
ipset 6.x

• Kernel-userspace communication: netlink
  – Flexible protocol, TLV
  – save-restore: PAGE_SIZE chunks
• IPv6 support
• Syntax similar to ip
netlink

- Queue handling over BSD socket infrastructure
netlink II.

- Netlink subsystems:
  - Rtnetlink
  - Nfnetlink
    - Conntrack-tools
    - NFLOG
    - ...
    - ipset
  - Genetlink
ipset netlink protocol

- Formalized
- Needs a simple kernel netlink interface modification:
  - netlink.patch below 2.6.39
- Supports batch mode (create + add)
- Userspace netlink library:
  - libnfnetlink
  - libnl
  - libmnl
ipset: unified syntax

• Type: method:elem1[,elem2[,elem3]]
  – hash:ip,port,ip
• Elem: part1[,part2[,part3]]
  – 192.168.1.1,80,10.10.10.10
• Iptables match/target: dir[,dir[,dir]]
  – dst,dst,src
Keyword-based syntax

• create, add, del, test, ...
  – No need for the dashes: --create, --add, ...
• Abbreviation, one-letter commands
• Backward compatibility
Timeout

• Every type supports timeouts:
  - Type without timeout
  - Type with timeout

• Timeout value can be reset:

create test hash:ip timeout 10
add test 10.0.0.1 timeout 0
add --exist test 10.0.0.1 timeout 20
ipset save-restore

- Restore implemented as batch mode

ipset save > ipset.rules
ipset restore < ipset.rules
iptables set match

- Search a match in an set by iptables: `set match`

- `-m set <setname> src|dst[,src|dst]`

```
ipset create servers hash:ip,port
ipset add servers 192.168.0.1,tcp:25
ipset add servers 192.168.0.2,tcp:80
iptables -A FORWARD -m set \ 
   --match-set servers dst,dst \ 
   -m state --state NEW -j log-accept
```
SET target

- Add/delete elements from sets by an iptables rule
- `-j SET --add-set|--del-set <setname> src|dst[,...]`

```bash
ipset create scanners hash:ip
   --timeout $((60*60*24*7))
iptables -A FORWARD -d <honeypot> \
   -p tcp --dport 22 \
   -j SET --add-set scanners src
```
Swap

• A set referenced by a rule cannot be deleted – but can be swapped with another set:

  ipset create main-set ...
  iptables -A ... -m set --match-set main-set ...
  ipset create main-set-tmp
  ipset add main-set-tmp
  ...
  ipset swap main-set main-set-tmp
  ipset destroy main-set-tmp
Storage method: bitmap

- Continuous binary area, every bit can store and IPv4 address: 192.168.0.1, 192.168.0.2, 192.168.0.5
  
  
  
  
  
  
  
  
  

- Can be generalized to store same-sized networks, IPv4 and MAC pairs and TCP/UDP port numbers
bitmap:ip type

- Store IPv4 addresses from a range
  - One address is stored in one bit
- Max 65536 elements (/16)
bitmap:ip examples

• Store IP addresses:
  ipset create set1 bitmap:ip range 192.168.0.0-192.168.255.255
  [ipset create set1 bitmap:ip range 192.168.0.0/16]
  ipset add set1 192.168.0.1
  ipset add set1 192.168.0.2-192.168.0.15

• Store same-sized whole networks:
  ipset create set2 bitmap:ip range 0.0.0.0/0 netmask 16
  ipset add set2 10.1.0.0
  ipset add set2 10.7.0.1
bitmap:ip,mac type

- IPv4 and MAC pair
- Only source MAC
- First matching can fill out unspecified MAC

ipset create set3 bitmap:ip,mac range 192.168.0.0/16
ipset add set3 192.168.1.1,00:01:23:45:67:89
ipset add set3 192.168.2.3
bitmap:port type

- Just for fun

```
ipset create set4 bitmap:port range 0-1024
ipset add set4 22
```
Storage method: hash I.

- Hashing: map a number space to a smaller number space with a method, which is
  - Fast
  - Deterministic
  - Uniform: every hash value has got the same probability
- Collision handling: typically with linked lists
Storage method: hash II.

Input | Hash function | Hash with linked lists
--- | --- | ---
10.0.0.2 |  | 0
172.16.0.5 |  | 2
172.16.0.5 |  | 1
192.168.0.1 |  | 172.16.0.5

Hash elem: Next
Prev
10.0.0.2

Hash elem: Next
Prev
192.168.0.1
Hash family

- Data stored in a `hashsize` sized hash
  - Size is the power of two
  - Array hash: data block for four elements, max three times increased
  - Hash size duplicated and rehashing when “full”

- Hash max size indirectly limited by the `maxelem` parameter

- Supports IPv4 or IPv6 addresses

Open Software Days 2013, Copenhagen
hash:ip type

- Random IP addresses
- Store IP addresses

```
ipset create set5 hash:ip hashsize 1024
ipset add set5 10.1.1.1
ipset add set5 192.168.52.3
```

- Store same-sized netblocks:

```
ipset create set6 hash:ip family inet6 netmask 64
ipset add set6 2001:2001:2001::
ipset add set6 2001:2001:abcd::
```
Hash and netblocks

- Possible stored netblocks:
  - IPv4: /1 - /32
  - IPv6: /1 - /128

- Exceptions: “nomatch”

- Speed linearly grows with the number of different sizes of the netblocks
hash:net type

- Store different sized netblocks in a set

```bash
ipset create set7 hash:net
ipset add set7 192.168.1.0/24
Ipset add set7 192.168.1.0/30 nomatch
ipset add set7 10.1.8.0/21
```
Store networks

- **Same sized nets from a single network:**
  
  ipset create set1 bitmap:ip range 192.168.0.0/16 netmask 24
  ipset add set1 192.168.1.0/24
  ipset add set1 192.168.2.0/24

- **Same sized nets from different networks:**
  
  ipset create set2 hash:ip netmask 24
  ipset add set2 10.1.1.0/24
  ipset add set2 192.168.2.0/24

- **Different sized nets: hash:net**
  
  ipset create set3 hash:net
  ipset add set3 192.168.1.0/24
  ipset add set3 10.1.8.0/21
Hash type and port

- We store both the protocol and port
  - TCP, UDP, SCTP, UDPLite, ICMP, ICMPv6
  - default TCP
  - ICMP and ICMPv6: type/code
  - Any other protocol
hash:ip,port type

- Store given services

  ipset create set8 hash:ip,port
  ipset add set8 192.168.1.1,icmp:ping
  ipset add set8 192.168.1.1,22
  ipset add set8 192.168.1.1,80
  ipset add set8 192.168.1.4,25
  ipset add set8 192.168.1.254,udp:53
  ipset add set8 192.168.1.254,tcp:53
hash:ip,port,ip type

- Services with restrictions

  `ipset create set9 hash:ip,port,ip`

  `ipset add set9 192.168.1.1,22,10.1.1.1`

  `ipset add set9 192.168.1.1,80,10.1.1.2`

  `ipset add set9 192.168.1.4,25,10.1.1.1`
hash:ip,port,net types

- Services with restrictions for a subnet:

  ipset create set10 hash:ip,port,net
  ipset add set10 192.168.1.1,22,10.1.1.0/24
  ipset add set10 192.168.1.1,80,10.2.0.0/16
  ipset add set10 192.168.1.4,25,10.1.1.0/24
hash:net,port type

● Networks and port numbers together:

```bash
ipset create set7 hash:net,port
ipset add set7 192.168.1.0/24,tcp:80
ipset add set7 10.1.8.0/21,icmp:ping
```
From most to least specific

- hash:ip,port,ip
- hash:ip,port,net
- hash:ip,port
- hash:net,port
- hash:ip
- hash:net
hash:net,iface type

- Networks and interfaces: eggress ingress filtering

  ipset create set7 hash:net,iface
  ipset add set7 192.168.1.0/24,eth1
  ipset add set7 10.1.8.0/21,eth0
list: set type

• Set “union”

• First match in a subset wins

ipset create set13 list: set size 6
ipset add set13 set1
ipset add set13 set2
ipset add set13 set3
Firewall built by ipset I.

# All internal clients who may access the Internet
# (Could have a bitmap:ip,mac type if the clients are on the same
# LAN or hash:ip,port with port restrictions):
ipset create clients bitmap:ip range 192.168.0.0/16
ipset add clients 192.168.10.1 
...
# All servers with the servcies
ipset create servers hash:ip,port
ipset add servers 192.168.0.1,22
ipset add servers 192.168.0.2,25
...
Firewall built by ipset II.

# Stateful rules
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# Define logging chains
...

# Egress/ingress filtering
iptables -A FORWARD -s 192.168.0.0/16 -i ext-iface -j log-drop
iptables -A FORWARD ! -s 192.168.0.0/16 -i int-iface -j log-drop
Firewall built by ipset III.

# The rule for the servers
iptables -A FORWARD -m set --match-set servers dst,dst \    -m state --state NEW -J log-accept
# The rule for the clients
iptables -A FORWARD -m set --match-set clients src \    -m state --state NEW -J log-accept
# Otherwise we log and drop everything else
iptables -A FORWARD -j log-drop
Ipset and tc

- **tc** supports matching in sets, thanks to Florian Westphal
New features in next version

• Optional packet and byte counters for the elements, including the list of sets
• set match is extended with packet and byte counter matching
Thanks!