# nftables – the ip(6)tables successor

## a new packet filtering/mangling framework

Networking Services Team, Red Hat

Florian Westphal
4096R/AD5FF600 fw@strlen.de
80A9 20C5 B203 E069 F586
AE9F 7091 A8D9 AD5F F600

February 2016

redhat.

# What is nftables?

- New packet classification framework based on lessons learnt.
- nftables was presented at Netfilter Workshop 2008 (Paris, France) and released in March 2009 by Patrick McHardy
- available since Linux 3.13 (January 2014)

redhat.

# What is being replaced?

- iptables, ip6tables, arptables, ebtables
  - Userspace tools
  - Kernel implementation of ruleset evaluation
- Re-uses existing netfilter hooks, conntrack, NAT, ...
- Userspace tools like conntrack(d) and ulogd remain unchanged as well

redhat.

# WHY is it being replaced?

Address iptables architectural design problems

- from Kernel: Avoid code duplication:
  - Four families forked from original iptables
  - Very similar extensions to match protocol fields and metadata.
- from Userspace:
  - unified tool
  - proper library for 3rd party software

redhat.

# Netfilter Kernel Architecture (1)

Hooks: particular locations in kernel where packet processing can be influenced, e.g. incoming packets to local stack:

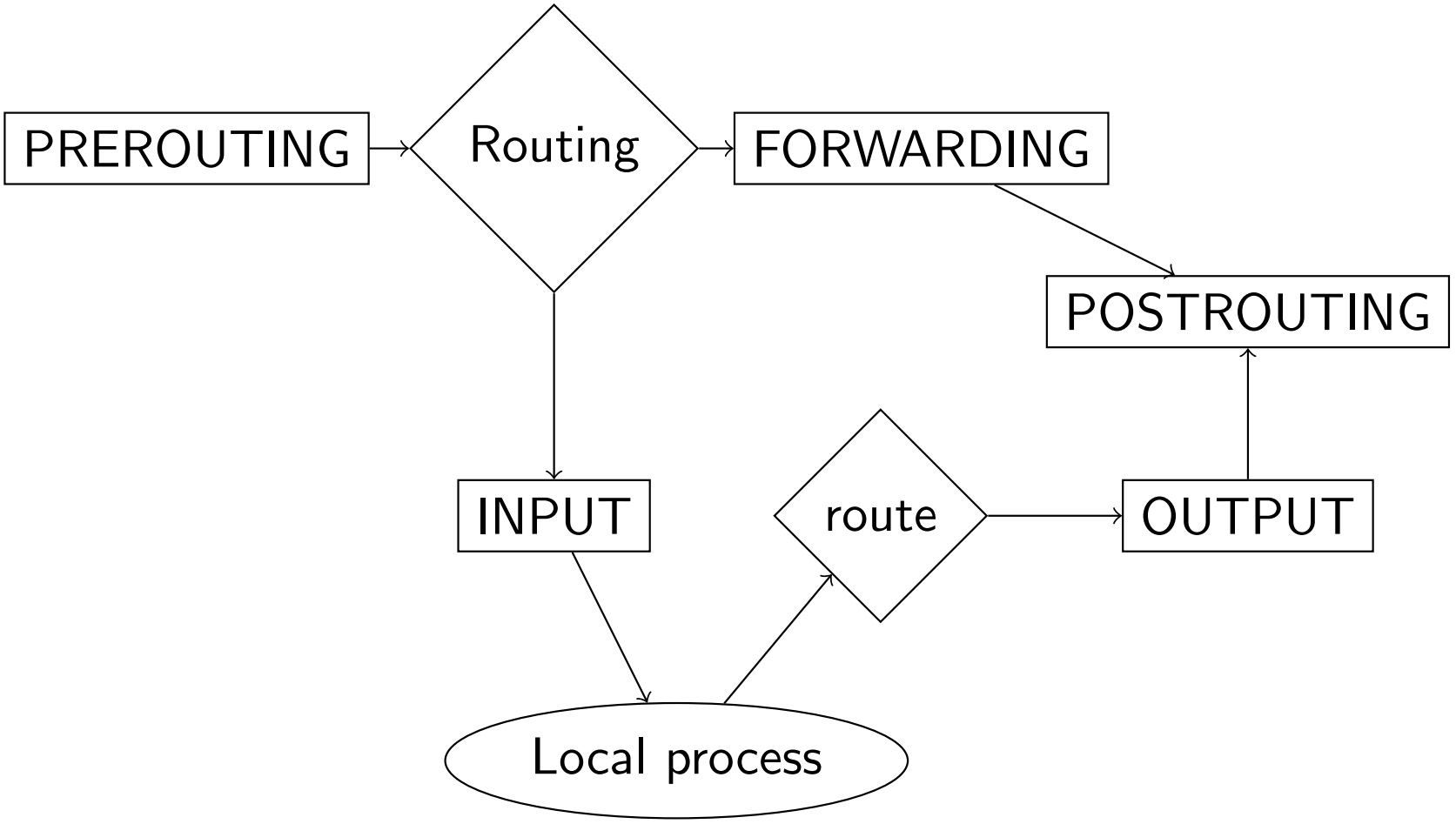
```
return NF_HOOK(NFPROTO_IPV4, NF_INET_LOCAL_IN, skb,
               skb->dev, NULL, ip_local_deliver_finish);
```

Without netfilter support compiled in this expands to:

```
return ip_local_deliver_finish(skb);
```

redhat.

# Netfilter Kernel Architecture (2)



redhat.

# iptables

```
iptables -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
```

- Only one target per rule:
  `LOG+DROP`? $\rightarrow$ helper-chain
- Ruleset stored as a binary blob generated by iptables
- Adding/removing rules is not atomic (dump/modify/restore cycle)
- Kernel has no idea what changed in the ruleset
- Userspace has no idea what ruleset is doing

redhat.

# Meanwhile, behind the scenes...

- linear evaluation of ruleset
- 4x Code-Duplication for ip,ip6,eb,arptables
- ebtables == 'iptables from 2001' (e.g. still uses rwlock in main traverser)
- Many extensions to work around iptables limitations: `multiport`, `ipset`, `u32`, `bpf`, ...
- ... that duplicate functionality (copy&paste programming): `ah`, `esp`, `ipcomp`, `dccp`, ...
- ... or for special cases: `HMARK`, `physdev`, ...
- 'magic' tables: 'mangle OUTPUT chain rerouting', `-t raw .. -j NOTRACK`', ...

redhat.

# nftables

- All protocol specific knowledge is in userspace
- Kernel: Interpreter/Virtual Machine
- nft commandline tool generates code/instructions
- Focus: efficient data structures/rule representation: Native support for sets
- nftables replaces...
  - Kernel: Representation and evaluation of filter rules
  - Userspace: iptables, ip6tables, arptables, ebtables

```
nft -f /path/to/ruleset
```

redhat.

# nftables - Kernel

rules consist of **Expressions**, no distinction between matches/targets

- Basic expressions are:
  - immediate ("22")
  - payload (load 'X bytes from offset Y into register R')
  - cmp ( >, <, =, != )
  - Bit operations (logical and, or, xor)
  - counter
- More expressions:
  - meta (mark, iif/oif, ..)
  - ct (conntrack)

redhat.

# nft: userspace frontend

```
nft add rule filter input tcp dport 22 accept
```

1. Userspace parses rules/grammar
   no 'options' or 'extensions' like in iptables
2. Userspace translates each rule into expressions
3. Serialized into netlink attributes and sent to kernel
4. Kernel validates netlink attribute, translates them into a private kernel representation

```
[ payload load 1b @ network header + 9 => reg 1 ]
[ cmp eq reg 1 0x00000006 ]
[ payload load 2b @ transport header + 2 => reg 1 ]
[ cmp eq reg 1 0x00001600 ]
[ immediate reg 0 accept ]
```

redhat.

# nft vs iptables

No builtin tables, to create equivalent of iptables:

```
#!/bin/nft -f
table ip filter {
  chain input   { type filter hook input priority 0; }
  chain forward { type filter hook forward priority 0; }
  chain output  { type filter hook output priority 0; }
}

table ip mangle {
  chain output { type route hook output priority -150; }
}

table ip6 name  {
chain name  { type filter hook output priority 0; }
```

redhat.

# Family overview

| | ebt,arp,ip,ip6tables | nft |
|---|---|---|
| arp | arptables | table arp |
| bridge | ebtables | table bridge |
| ipv4 | iptables | table ip |
| ipv6 | ip6tables | table ip6 |
| inet | - | table inet |
| netdev | - | table netdev |

redhat.

# nft vs iptables (2)

- Rule replacement is atomic
- Unified tool for all families, e.g. nft bridge can just use "`ip saddr`"
- New "inet" family for shared ipv4/ipv6 ruleset
- New ingress hook, for early filtering (before l3 demux)

```
table netdev foo {
    chain bar {
      type filter hook ingress device eth0 priority 0;
      ether type arp counter accept
   }
}
```

redhat.

# Sets and concatenations

Fast lookups via sets:

```
ip saddr { 192.168.7.1, 192.168.10.2, .. }
[ payload load 4b @ network header + 12 => reg 1 ]
[ lookup reg 1 set set0 ]
```

Optional: keys can be concatenated via '.':

```
add rule ip filter output ip daddr . tcp dport {
        192.168.0.1 . 22,
        192.168.0.1 . 80,
} accept
```

redhat.

# Named sets

nft also supports **named** sets:

```
nft add set filter foo { type ipv4_addr . inet_service; }
nft add rule filter input ip daddr . tcp dport @foo accept
```

. . . and then add/remove entries later:

```
nft add element filter foo { 192.168.0.1 . 22 }
```

Can't recall datatype to use?
`nft describe ip saddr` (or `tcp dport`, etc).

redhat.

# Maps

```
nft add rule nat postrouting snat \
  ip saddr map { 192.168.1.1 : 1.1.1.1,
                 192.168.2.2 : 2.2.2.2,
                 192.168.3.3 : 3.3.3.3 }
```

```
[ payload load 4b @ network header + 12 => reg 1 ]
[ lookup reg 1 set map%d dreg 1 ]
[ nat snat ip addr_min reg 1 addr_max reg 0 ]
```

redhat.

# Verdict maps

- Jump tables, i.e. lookup yields next chain to process

```
add filter input ip saddr vmap {
  8.8.8.8 : accept,
  192.168.0.0/16 : drop,
  10.0.0.0/8 : jump chain1,
  172.16.0.0/16 : jump chain2,
}
```

redhat.

# monitoring and debugging

```
nft monitor
```

... works just like 'ip monitor': you get text output, e.g.:

```
add rule ip filter output ip saddr 1.2.3.4 accept
delete rule ip filter output handle 24
delete rule ip filter output handle 25
...
```

redhat.

# iptables TRACE

iptables supports `TRACE` target to enable per-rule tracing, e.g.

```
iptables -t raw -A PREROUTING -p tcp \
                  --syn --dport 22 -j TRACE
```

And then you need to look at `dmesg` to figure out whats going on.

redhat.

# iptables TRACE (2)

```
TRACE: raw:PREROUTING:policy:2 IN=eth0 SRC=192.168.0.8 \
       DST=192.168.0.10 SPT=7627 DPT=22 SYN
TRACE: mangle:PREROUTING:policy:1 IN=eth0 SRC=192.168.0.8 \
       DST=192.168.0.10 SPT=7627 DPT=22 SYN
TRACE: nat:PREROUTING:rule:1 IN=eth0 SRC=192.168.0.8 \
       DST=192.168.0.10 SPT=7627 DPT=22 SYN \
TRACE: mangle:INPUT:policy:1 IN=eth0 SRC=192.168.0.8 \
       DST=192.168.0.10 SPT=7627 DPT=2222 SYN
TRACE: filter:INPUT:rule:11 IN=eth0 SRC=192.168.0.8 \
       DST=192.168.0.10 SPT=7627 DPT=2222 SYN
```

You manually have to match up the rule numbers with whatever rule set you're using. With nft, you'd instead ...

redhat.

# nft trace

```
nft add rule raw prerouting tcp dport 22 \
  tcp flags & (syn|ack) == syn meta nftrace set 1
```

and then use monitor trace mode:

```
# nft monitor trace
trace id 834d rule tcp dport 22 tcp flags \
  & (syn|ack) == syn meta nftrace set 1
trace id 834d raw pre policy verdict accept iif eth0
trace id 834d filter input rule ip saddr \
 . tcp dport vmap { } verdict jump iif eth0
trace id 834d filter test rule accept iif eth0
```

redhat.

# Future Work

- High-level library for 3rd party applications
- (Not) yet feature-complete
- Some of the missing iptables extensions:
  - matches:
    - policy (ipsec)
    - rateest
    - rpfilter
    - hashlimit (work in progress)
  - targets:
    - CT
    - TCPMSS
    - RATEEST
- No nfqueue or conntrack for nft bridge family so far
- performance tests and optimization work

`http://www.devconf.cz/feedback/325`

`http://wiki.nftables.org`

redhat.

# simple ipv4/v6 filter set

```
table inet filter {
        chain input {
                type filter hook input priority 0;
                ct state established,related accept

                iif lo accept
                ip6 nexthdr icmpv6 icmpv6 type {
                        nd-neighbor-solicit,
                        echo-request, nd-router-advert,
                        nd-neighbor-advert } accept
                counter drop
        }
}
```