NFTables – Wieso, Weshalb, Warum

Der neue Linux-Paketfilter

Florian Westphal

fw@strlen.de

13. April 2014

Agenda

1. Begriffsdefinitionen/-abgrenzungen
2. Übersicht netfilter-Architektur
   - kernel-hooks
   - iptables Architektur/Funktionsweise
   - Probleme und wünschenswerte Funktionen
3. nftables
   - Aufbau/Funktionsweise
   - Beispiele und Ausblick

Begriffsdefinitionen

- skb: "socketbuffer", Kernel-Datenstruktur, repräsentiert ein Paket
- iptables, ip6tables, arptables, ebtables
  - Benutzerprogramme
  - Kernelseitige Implementierung
  - NICHT: connection tracking, NAT
- netfilter: "Alles"
  - kernel: `net/netfilter/, net/ipv4/netfilter, ...`
  - Benutzerprogramme: iptables, nft, ulogd, conntrack tools
- nftables: Neuer Paketfilter

Netfilter Kernel-Architektur (1)

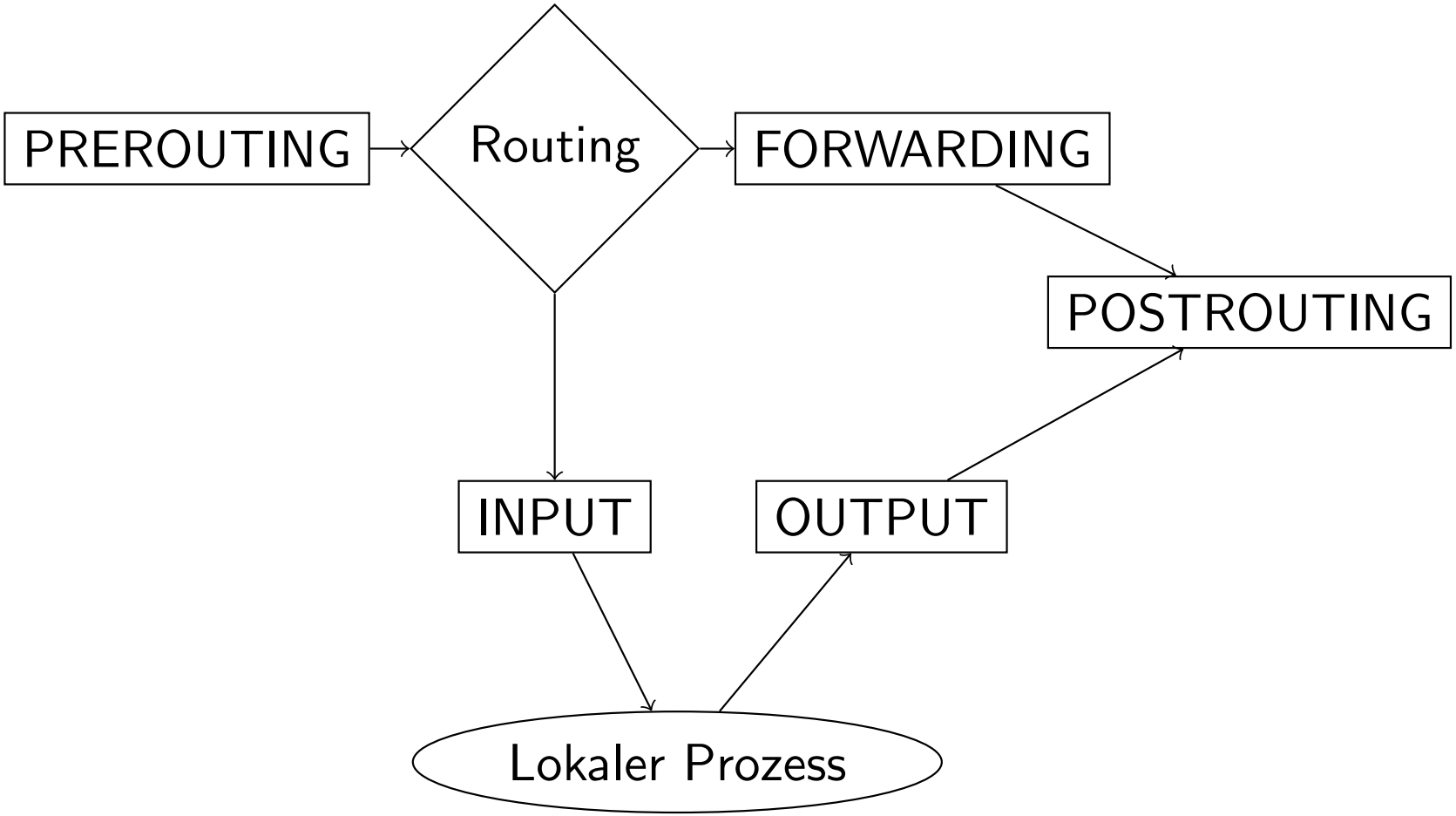
Hooks: Stellen im Kernel, an dem Paketverarbeitung beeinflusst werden kann, z.B. eingehende lokale Pakete:

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

Falls kein iptables support: expandiert zu:

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

Netfilter Kernel-Architektur (2)



Netfilter Hooks

```
struct nf_hook_ops ipv4_conntrack_ops[] = {
        {
                .hook           = ipv4_conntrack_in,
                .pf             = NFPROTO_IPV4,
                .hooknum        = NF_INET_PRE_ROUTING,
                .priority       = NF_IP_PRI_CONNTRACK,
        }, {
 [..]
```

Wenn z.B. ipv4-conntrack-modul geladen wird:

```
 ret = nf_register_hooks(ipv4_conntrack_ops,
                         ARRAY_SIZE(ipv4_conntrack_ops));
```

Hook-Entscheidungen

Rückgabewert entscheidet was passiert

- ACCEPT (okfn wird ausgeführt, $\rightarrow$ "weiter im Stack")
- DROP (skb wird weggerworfen)
- QUEUE (paket an userspace weiterleiten)
- REPEAT (nochmal)
- STOLEN (Schwarze Magie)

Zusammenfassung

- Bewährtes Konzept: besteht seit Linux 2.4
- ändert sich auch mit nftables nicht
- auch Connection-Tracking und NAT bleiben erhalten
- nftables ersetzt:
  - Kernel: Repräsentation und Evaluierung von Paketfilterregeln
  - Userspace: iptables, ip6tables, arptables, ebtables
- Wieso?

Funktionsweise Userspace

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

1. Userspace parst Optionen, konvertiert in binär-Format
2. . . . holt alle Regeln aus dem Kernel (Binärblob, getsockopt)
3. Regel wird in Binärblob eingefügt
4. Userspace schiebt Gesamtblob zurück (setsockopt)
5. Kernel tauscht blobs aus
6. Tables sind 'kleinste Einheit'

Binärblob: "Array aus structs"

iptables: Funktionsweise (2)

```
iptables -A INPUT -m conntrack --state NEW \
          -m limit --limit 1/min -j NFLOG
```

- match: ja/nein Entscheidung, von-links-nach-rechts
- Target "macht etwas mit dem Paket" (terminal oder non-terminal)
- Funktionalität über Module (matches, targets) Erweiterbar
- 1:1 mapping von Targets und Matches zwischen User- und Kernelspace
- `xt_bla.ko` vs. `libxt_bla.so`
- eigentliche Funktionalität nur im Kernel

iptables: Funktionsweise (3)

Lineare Evaluation von Regeln:

```
iptables -A INPUT -p tcp --dport 22 ...
iptables -A INPUT -p tcp --dport 80 ...
iptables -A INPUT -s 10.0.0.0/8 -p tcp --dport 443 ...
iptables -A INPUT -s 10.0.0.0/8 -p udp --dport 53 ...
```

→ Teuer
Verschiedene Workarounds, z.B. `-m multiport`, `ipset` – alle nicht perfekt
Händische Optimierung: 'Common Case' zuerst

Probleme zusammengefasst

- Änderungen sind nicht atomar (dump/modify/restore)
- nur ein Target pro Regel:
  `LOG+DROP`? $\rightarrow$ helper-chain
- Kernel weiss nicht, was sich am Regelset geändert hat
- Userspace weiss nicht, was das Regelset überhaupt tut

Hinter der Kulissen ...

- 4-fache Code-Duplikation für ip,ip6,eb,arptables
- ebtables == '12 Jahre altes iptables'
- `CONFIG_COMPAT`
- über die Zeit viele Erweiterungen:
  `multiport, ipset, HMARK, -m ipcomp, -m bpf...`
- viele Funktionsüberschneidungen/Copy-Paste-Programmierung:
  `ah, esp, ipcomp, dccp, ...`
- 'magic' tables: 'mangle OUTPUT chain rerouting',
  `CT --untracked`', etc.

Wishlist

1. mehrere Targets in einer Regel
2. besseres Filtering: nicht bloss `-s 10.0.0.0/8`
   - `-s 10.0.0.11-10.0.0.24 -d {blacklist}`
3. Automatische Optimierung von Regeln
4. Dead-Rule Erkennung/Entfernung
5. Sets mit Sprung-Support,
   `switch ($saddr) { case 1.2.3.4: ... }`
6. iptables-Äquivalent zu `ip monitor` & co.
7. Bibliothek/Schnittstelle für Anwendungen

. . . und weniger Code im Kernel

nftables

- alle Protokollspezifischen Unterschiede im Userspace
- nft-Kommandozeilenprogramm generiert (Pseudo-)Code
- Kernel: Interpreter/Virtuelle Maschine
- Tables beinhalten chains, chains beinhalten Regeln
- Regeln bestehen aus **Expressions**
- Schwerpunkt: Effiziente Datenstrukturen

nftables - Kernel-seite (1)

- momentan 4 128bit GP-Register
- 1 Verdict-Register

```
struct nft_data {
    union {
        u32 data[4];
        struct {
            u32 verdict;
            struct nft_chain *chain;
        };
    };
} __attribute__((aligned(__alignof__(u64))));
```

nftables - Kernel-seite (2)

- iptables match/target → nft expression
- Grundlegende Expressions sind:
  - immediate ("42")
  - payload ('X byte ab offset Y nach Register R')
  - cmp (>, <, =, !=)
  - Bit-Operationen
- Weitere Expressions (Auswahl):
  - counter
  - meta (mark, iif/oif, ..)
  - lookup

Funktionsweise Userspace

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

1. Userspace parst Regel(n)/Grammatik
   'Optionen' bzw. 'Extensions' wie in iptables gibt es nicht
2. wird in 'Expressions' übersetzt
3. diese werden in netlink-Attribute serialisiert und an Kernel gesendet
4. Kernel validiert netlink-Attribute, übersetzt in Kernel-Format

```
[ 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 ]
```

nft - Unterschiede

Keine festen Tables mehr:

```
#!/bin/nft -f
table 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 mangle {
  chain output { type route hook output priority -150; }
}

table bridge blablubb { ...
```

nft - Unterschiede (2)

- Regel-Ersetzung ist stets atomar
- sehr viel flexibler als iptables, z.B.

```
add rule ip mangle prerouting ip daddr \
192.168.7.1 meta mark set (ip saddr & 0xff)
[ payload load 4b @ network header + 16 => reg 1 ]
[ cmp eq reg 1 0x0107a8c0 ]
[ payload load 4b @ network header + 12 => reg 1 ]
[ bitwise reg 1 = (reg=1 & 0xff000000 ) ^ 0x00000000 ]
[ meta set mark with reg 1 ]
```

nft - Unterschiede (3)

viele Expression-Kombionationen möglich

```
nft add rule filter forward tcp flags syn,fin \
     counter log prefix synfin drop
[ payload load 1b @ network header + 9 => reg 1 ]
[ cmp eq reg 1 0x00000006 ]
[ payload load 1b @ transport header + 13 => reg 1 ]
[ bitwise reg 1 = (reg=1 & 0x00000003 ) ^ 0x00000000 ]
[ cmp neq reg 1 0x00000000 ]
[ counter pkts 0 bytes 0 ]
[ log prefix 'synfin' group 0 snaplen 0 qthreshold 0 ]
[ immediate reg 0 drop ]
```

Sets und Set-Concatenations

- beliebig viele Elemente in einer Regel
- effizienter Lookup in hash/tree

```
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 können mit '.' verbundenen werden

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

Verdict-Maps

- Sprungtabellen/Verzweigungen

```
add filter input ip saddr vmap {
  8.8.8.8 : accept,
  192.168.0.0/16 : drop,
  10.0.0.0/24 : jump chain1,
  10.0.0.0/8  : jump chain2,
}
```

nftables TODO

- (noch) nicht feature-complete
  - fehlender Set-support für conntrack-attribute (ausser ctmark)
  - fehlende iptables-Erweiterungen, z.B TCPMSS, hashlimit, ipsec policy matching...
- route-lookup Expression
- generischer Paket-Editor
- nft für socket-filter oder für QoS-Entscheidungen (tc Ersatz)
- kein Filtering auf Elemente $<$ 1 byte (z.B. iphdrlen)
- Dokumentation ausbaufähig
- high-level Programmierschnittstelle für Drittanwendungen

Fazit

- Grundfunktionen(filtering, sets) sind nutzbar
- wird noch weiterentwickelt, auch inkompatible Änderungen denkbar
- iptables trotz aller Probleme enormer Erfolg
- ... bleibt noch mehrere Jahre erhalten