NFTABLES

CARSTEN STROTMANN

2024-09-26 THU 00:00

Created: 2024-09-26 Thu 05:55

RENOVIERUNG IM LINUX-KERNEL - DIE NEUE FIREWALL "NFTABLES"

AGENDA

1. Warum eine neue Firewall
2. Unterschiede zu "iptables"
3. Arbeiten mit dem "nft" Kommando
4. nft Regelwerke
5. "nft" Datenstrukturen
6. Migration von bestehenden "iptables" Regeln

GESCHICHTE DER FIREWALL-SYSTEME IM LINUX-KERNEL

- Kernel 1.1 (1994) - Port der BSD *ipfw* Firewall durch Alan Cox
- Kernel 2.0 (1996) - *ipfwadm* (Jos Vos, Pauline Middelink …)
- Kernel 2.2 (1999) - *ipchains* (Rusty Russel, Netfilter Team)
- Kernel 2.4 (2001) - *iptables* (Rusty Russell, Harald Welte, Netfilter Team)
- Kernel 3.13 (2014) - *nftables* (Patrik McHardy, Netfilter Team)

"NFTABLES" VS. "IPTABLES"

- *iptables* ist fragmentiert, mehrere Kommandos mit sehr ähnlicher Funktion -> Code-Duplizität
  - iptables
  - ip6tables
  - arptables
  - ebtables
- *nftables* vereint diese Funktionen in einem Werkzeug

"NFTABLES" VS. "IPTABLES"

- Bei *iptables* gibt es eine "Race-Condition"-Gefahr durch das Fehlen eines Regelsatz-Kompilers
- *iptables* hat kein eigenes Konfigurationsformat für Firewall-Regeln
    - Das Regelwerk wird über Aufrufe des *iptables* Kommandos aufgebaut.
- Bei *nftables* können die Regeln in einer Regelsatz-Datei definiert und per Transaktion in den Kernel geladen werden

"NFTABLES" VS. "IPTABLES"

- *iptables* Regelwerke werden durch Shell-Scripte aufgebaut
- Erhöhte Komplexität für Firewall-Administratoren mit geringem Shell-Scripting-Wissen
- Die Regel-Beschreibungsprache von *nftables* ist einfach(er) lesbar

IM KERNEL

- *iptables*-Firewall ist durch statische Kernel-Module für die Haupt-Funktionen und Erweiterungen implementiert
- Diese Module benutzen das gleiche Kernel-Framework (*netfilter*), duplizieren aber einen nicht unerheblichen Teil der Firewall-Funktionen in jedem Modul

IM KERNEL

- *nftables* implementiert die Firewall-Funktionalität durch eine generische Virtuelle-Maschine im Kernel
  - Die virtuelle Maschine wird von dem Kommandozeilen-Tool *nft* mit Bytecode geladen
  - Der Bytecode ist ein Repräsentation der Firewall-Regeln als ausführbares Programm für die *nftables*-VM
  - Der Bytecode wird ausserhalb des Kernels erzeugt und in die *nftables*-VM des Kernels geladen
  - Neue Firewall-Funktionen können so unabhähngig von der Kernel-Version realisiert werden

IM KERNEL

Mit dem Befehl

```
# sudo lsmod | grep "nf_t"
```

kann geprüft werden, ob *nftables* im Linux-System schon aktiv ist.

DER NFT BEFEHL

- *nft* verwaltet die *nftables*-Firewall.
- Anders als das *iptables* Kommando werden die Firewall-Regeln nicht durch Kommandozeilen-Parameter angegeben, sondern in der *nftables*-eigenen Beschreibungssprache für Firewall-Regeln.
    - Diese Beschreibungssprache funktioniert ein wenig wie eine kleine Programmiersprache.

DER NFT BEFEHL

- Neben Definitionen von Firewall-Regeln nimmt *nft* auch Befehle entgegen
  - Löschen einer Regel
  - Löschen des gesamten Regelsatzes
  - Export der aktiven Regeln in eine Datei

DER NFT BEFEHL

nft kann Regeln aus einer Datei, von der Kommando-Zeile (Shell) oder in einem interaktiven Modus direkt vom der Tastatur einlesen.

Da die Beschreibungssprache Zeichen beinhalten kann, welche in Unix-Shells besondere Bedeutungen haben, müssen diese Zeichen, oder besser der gesamte *nft* Befehlsteil, per Quoting vor dem Zugriff der Shell geschützt werden.

DER NFT BEFEHL

Beispiel: Lesen der Firewall-Regeln aus einer Datei

```
# nft -f /etc/nftables/firewall.nft
```

Beispiel: Eine Regel (IPv4, Tabelle "fw", Kette "input") hinzufügen

```
# nft "add rule ip fw input drop"
```

INTERAKTIVE *NFT* SITZUNG

Beispiel: eine interaktive *nft* Sitzung

```
# nft -ia
nft> list ruleset
table inet workstation-fw {
      chain input {
            type filter hook input priority 0;
      }
}
```

INTERAKTIVE *NFT* SITZUNG

Beispiel: eine interaktive *nft* Sitzung

```
nft> add rule inet workstation-fw input tcp dport {ssh, http} accept
nft> list ruleset
table inet workstation-fw {
    chain input {
       type filter hook input priority 0;
       tcp dport { ssh, http} accept # handle 3
    }
}
nft> quit
```

REGELNSATZ LÖSCHEN

- Der Befehl `flush` löscht Objekte aus der *nftables* Kernel-Firewall.
- Ein `flush ruleset` löscht die gesamten Firewall-Regeln.
- Ein `flush ruleset` sollte am Anfang eines *nftables*-Regelsatzes stehen:

```
# nft flush ruleset
```

- Über `flush chain` und `flush table` können Ketten und Tabellen gelöscht werden.

REGELN HINZUFÜGEN

- Der Befehl `add` fügt die Regel immer am Ende der Kette an
- mit `insert` kann eine Regel an einer beliebigen Stelle der Kette eingefügt werden.

REGELN LÖSCHEN

- Der Befehl `delete` löscht eine Regel aus einer Kette. Zum Löschen einer Regel wird das *Handle* der Regel benötigt, das *Handle* wird über den Kommandozeilenparameter –a ausgegeben:

```
# nft -a list ruleset
...
ip6 saddr 2001:db8::/64 # handle 15
...
# nft delete rule inet filter input handle 15
```

MONITOR

Mittels `nft monitor` können Änderungen in der *nftables*-Firewall überwacht werden.

Der Befehl schreibt Modifikationen im Regelsatz der Firewall als `nft`-Befehlszeilen (Alternativ als XML oder JSON) auf die Konsole:

MONITOR

```
# nft monitor
add table inet filter
add chain inet filter input { type filter hook input priority 0; }
add rule inet filter input iif lo accept
add rule inet filter input ct state established,related accept
add rule inet filter input ip6 daddr ff02::1010 udp dport ntp accept
add rule inet filter input ip6 daddr ff02::fb udp dport mdns accept
add rule inet filter input ip daddr 224.0.0.251 udp dport mdns accept
add rule inet filter input ip6 saddr & :: == :: udp dport dhcpv6-client accept
add rule inet filter input udp dport ipp accept
add rule inet filter input ip6 saddr & :: == :: icmpv6
      type { nd-neighbor-solicit, nd-router-advert, nd-neighbor-advert} accept
add rule inet filter input ip6 saddr & :: == :: icmpv6
      type { echo-reply, echo-request, packet-too-big, destination-unreachable,
             time-exceeded, param-problem} accept
add rule inet filter input counter packets 0 bytes 0 log prefix "nftables drop: " drop
```

TABELLEN UND KETTEN

- Die aus *iptables* bekannten Konzepte von Tabellen (Tables) und Ketten (Chains) finden sich auch in *nftables* wieder
- Vordefinierten Tabellen (in *iptables* 'filter', 'nat' und 'mangle') gibt es im *nftables* nicht
- Auch die in *iptables* vorhandenen Ketten ('input', 'output', 'forward') sucht man vergebens

TABELLEN UND KETTEN

- *nftables* kommt ohne vordefinierte Tabellen und Ketten.
- Die vordefinierten Ketten in *iptables* sind ein Performance-Problem bei grossen Datenmengen, da Netzwerkpakete auch durch unbenutze, leere Ketten geschleusst werden.
    - Unter *nftables* sind nur Tabellen und Ketten aktiv, welche der Administrator definiert hat.

TABELLEN UND KETTEN

- In *nftables* sind Tabellen einfache Container für beliebige Ketten.
- Eine Tabelle muss einem Protokoll zugewiesen werden (*ip* = IPv4, *arp*, *ip6* = IPv6, *bridge*).

TABELLEN UND KETTEN

- Die Tabellen können beliebige Namen haben, auch wenn in vielen Beispielen im Internet die aus *iptables* bekannten Namen verwendet werden.
  - Bei den Ketten (Chains) unterscheidet man "base-chains" und "auxiliary chains".
  - Bei den "base-chains" handelt es sich um Ketten, welche in den Paketfluss des Netfilter-Framework eingeklinkt werden.
  - Diese Ketten erhalten Netzwerkpakete aus den Zugriffspunkten (Hooks) des Linux-Kernels.

TABELLEN UND KETTEN

- Diese Zugriffspunkte für eine Kette vom Typ "filter" heissen "input", "output" und "forward".
- Die Konzepte sind von *iptables* übernommen worden, der Firewall-Administrator hat nun aber mehr Freiraum, die Ketten zu organisieren.

BEISPIEL-TABELLE

- Beispiel einer Tabelle mit zwei (leeren) "filter" Ketten, für "input" und "output":

```
table inet filter01 {
        chain input01 {
                type filter hook input priority 0;
        }
        chain output01 {
                type filter hook output priority 0;
        }
}
```

BASE- UND AUXILIARY-KETTEN

- Mit dem Schlüsselwort *hook* wird eine Kette mit den Linux-Kernel-Internen Schnittstellen des Netfilter-Frameworks verbunden.
- Eine Kette mit Verbindung zum den Kernel-Schnittstellen ist eine "base-chain".
- Zusätzlich zu den "base-chains" kann der Administrator noch beliebig weitere "auxiliary chains" erzeugen.
- Diese Chains erhalten nur Netzwerkpakete, wenn diese aus einer "base-chain" per `goto` oder `jump` Direktive an die "auxiliary chain" verteilt werden:

BEISPIEL BASE- UND AUXILIARY-KETTEN

```
table inet filter01 {
        # auxiliary chain
        chain link-local-aux01 {
                 counter packets 0 bytes 0 log prefix "link-local: " accept
                 # weitere Regeln für link-local IPv6
                 [...]
        }
        # base chain
        chain input01 {
                type filter hook input priority 0;
                ip6 daddr fe80::/64 jump link-local-aux01
        }
        # base chain
        chain output01 {
                type filter hook output priority 0;
        }
}
```

FILTER-REGELN

- Eine Hauptaufgabe einer Firewall ist das Filtern von Netzwerkverkehr.
- Die *nftables*-Firewall bietet umfangreiche Möglichkeiten, Netzwerkpakete in Filterregeln auszuwählen:
  - nach Quell- und Ziel-Adressen
  - UDP- und TCP-Ports
  - Meta-Informationen wie
    - Netzwerk-Interfaces
    - Protokolle
    - Hardware-MAC-Adressen
    - VLAN-Informationen
    - Status einer IP-Verbindung (Connection-Tracking)

BEISPIEL EINER FILTER-REGEL

- In diesem Beispiel wird eine Regel der Firewall hinzugefügt. Gefiltert wird nach der IPv6-Zieladdresse `daddr ff02::fb` und dem UDP-Port `dport mdns` (Multicast-DNS):

```
# nft "add rule inet filter input ip6 daddr" \
            "ff02::fb udp dport mdns accept"
```

- Eine Liste der Filter-Optionen findet sich in der *man*-Page zu `nft`.

VERDICT STATEMENTS

- Am Ende einer Firewall-Regel wird das Urteil (Verdict) über eine Netzwerk-Verbindung getroffen, das sogennante Verdict-Statement.
- Es gibt entgültige Urteile, welche die Bearbeitung eines Netzwerkpaketes sofort beenden.
- Dazu zählen * `accept` (Paket passieren lassen) * `drop` (Paket ohne Rückmeldung an den Sender verwerfen) und * `reject` (mit Rückmeldung verwerfen).

VERDICT STATEMENTS

- Bei `reject` kann optional die ICMP-Meldung spezifiziert werden, welche an den Sender des Paketes zurückgeliefert wird:

```
# nft "add rule inet filter input ip6 saddr 2001:db8::/64" \
      "reject with icmpv6 type admin-prohibited"
```

VERDICT STATEMENTS

Neben den entgültigen Urteilen kann eine Regel die Bearbeitung eines Paketes auch an anderen Stelle fortsetzen:

- `continue` Bearbeitung des Paketes mit die nächsten Regel in der Kette
- `jump <chain>` verzweigt die Bearbeitung wie ein Unterprogrammaufruf in eine andere Kette. Am Ende der neuen Kette wird die Bearbeitung in der ursprünglichen Kette weitergeführt.
- `goto <chain>` wird direkt in eine andere Kette gesprungen, ohne das die Bearbeitung jemals wieder zurückkehrt.

VERDICT STATEMENTS

- `queue` verzweigt die Verarbeitung an ein externes Programm im Userspace.
- `return` beendet die Bearbeitung der aktuellen Kette an dieser Stelle. Wird `return` in einer Kette der obersten Bearbeitungsebene verwendet, so wird das Paket durch die Firewall gelassen (identisch zu einem `accept`).

LOGGING UND ZÄHLER

- Über das Befehlswort *log* werden Informationen zu dem gerade bearbeiteten Netzwerk-Paket in das Kernel-Log (und damit in den meisten Systemen auch in das System-Log) geschrieben.
- Der Befehl *counter* erzeugt einen Zähler für diese Firewall-Regel.

LOGGING UND ZÄHLER

- Anders als bei *iptables* müssen Counter für Regeln explizit erzeugt werden. Dies hat Performance-Gründe, die Zähler werden nur dann mitgeführt, wenn der Administrator diese im Regelsatz angefordert hat.
- In dem Zähler werden die Anzahl der gefilterten Pakete und die Datenmenge in Bytes gezählt.
- Der Zähler wird beim Auflisten des Regelwerkes, z.B. mit `list ruleset`, ausgegeben.

LOGGING UND ZÄHLER

- Die Position des Befehls `log` in der Regel wichtig:
  - Steht der Befehl vor der Filter-Bedingung, so wird ein Log-Eintrag für jedes Paket geschrieben, welches von dieser Regel gesehen wird, unabhängig ob die Regel danach aktiv wird oder nicht.
  - Steht jedoch der Befehl `log` nach der Filterbedingung, so wird der Log-Eintrag nur geschrieben, wenn die Filterbedingung zutrifft.
- Gleiches gilt für die Position des Befehls `counter`.

LOGGING UND ZÄHLER

```
# schreibe einen Log-Eintrag für jedes Paket
# welches von dieser Regel gesehen wird
log tcp dport { 22, 80, 443 } ct state new accept
# schreibe einen Log-Eintrag nur wenn die Regel zutrifft
tcp dport { 22, 80, 443 } ct state new log counter accept
```

- Dem Befehl `log` kann mit dem Parameter `prefix` eine beliebige Zeichenkette mitgegeben werden.
- Diese Zeichenkette erscheint vor jeder Log-Meldung und wird benutzt, um Log-Einträge zu finden und zu filtern.

IPV4 UND IPV6 KOMBINIEREN

- Neben den Protokollen *ip* für IPv4 und *ip6* für IPv6 unterstützt *nftables* auch das Pseudo-Protokoll *inet* für kombinierte IPv4- und IPv6-Filter.
- Anstatt z.B. Port 80 für einen Webserver in zwei getrennten Regeln für je IPv4 und IPv6 freizugeben, kann dies in *nftables* in nur einer Regel geschehen.
- Bei einem Server mit Dual-Stack und vielen Diensten kann dies das Regelwerk vereinfachen.

IPV4 UND IPV6 KOMBINIEREN

- Für das Protokoll *inet* muss eine eigene Tabelle mit Chains erstellt werden.

```
table inet filter {
    chain input {
        ...
    }
}
```

IPV4 UND IPV6 KOMBINIEREN

- Die Tabelle mit dem Pseudo-Protokoll *inet* existiert parallel zu Tabellen mit IPv4- und IPv6-Regeln.
- Werden Tabellen mit dedizierten IPv4 und IPv6 Regeln genutzt, so müssen die Pakete sowohl in der *inet* Tabelle als auch in der jeweiligen Tabelle für das IP-Protokoll erlaubt werden.
- Um Fehler zu vermeiden, sollte entweder kombinierte *inet* Tabellen, oder protokoll-spezifische Tabellen benutzt werden, aber nicht beides.

DATENSTRUKTUREN

- *nftables* bietet verschiedene Datenstrukturen, welche die Erstellung eines Regelwerkes vereinfachen:
  - Variablen
  - Intervalle
  - Sets
  - benannte und anonyme Maps
  - benannte und anonyme Dictionaries/Verdict-Maps

VARIABLEN

- Variablen erlauben symbolische Namen für Werte und IP-Adressen in einem *nftables*-Regelwerk.
- Variablen sind jeweils in der Umgebung sichtbar, in der sie definiert wurden (Table, Chain).
- Variablen werden über das Schlüsselwort `define` deklariert und mit einem Wert versehen. Im Regelwerk wird dann der Variablenname mit einem vorangestellem Dollar-Zeichen "$" verwendet, vergleichbar mit Variablen der Unix-Shell.

```
define lan_if = eth0
define DMZ-Dienste = { ssh, smtp, dns, http, https }
add rule inet filter input iif $lan_if tcp dport $DMZ-Dienste accept
```

INTERVALLE

- Intervalle in *nftables* sind Wertefolgen mit einem Start- und End-Wert.
- Intervalle werden z.B. für IP-Adresse-Bereiche und TCP/UDP-Port-Bereiche verwendet.

```
nft "add rule inet filter input ip6 daddr 2001:db8::0-2001:db8::1000 drop"
nft "add rule inet filter input tcp dport 0-1023 drop"
```

SETS

- Sets definieren eine Sammlung von Werten eines gleichen Typs. Ein Set wird durch geschweifte Klammern eingeleitet und die Elemente des Sets werden mit Komma getrennt.
- Ein anonymes Set mit Port-Nummern (Microsoft SMB/CIFS-Protokoll):

```
nft "add rule inet filter input udp dport { 137, 138, 139, 445 } reject"
```

SETS

- Bei der Anlage eines benannten Sets muss *nftables* der Werte-Typ mitgeteilt werden.
- Eine (leider noch unvollständige) Liste der verfügbaren Werte-Typen ist in der *nft* man-Page aufgelistet:

```
nft "add set inet filter bogus { type ipv4_addr; }"
nft "add element inet filter bogus { 203.0.113.0; }"
nft "add element inet filter bogus { 203.0.113.1; }"
nft "add rule inet filter input ip saddr @bogus drop"
nft "list set inet filter bogus"
```

- Der letzte Befehl des Beispiel gibt den aktuellen Inhalt des benannten Sets *bogus* aus.

SETS

- Ein benanntes Set kann in einer Regel mit vorangestelltem "@" eingefügt werden.
- Eine Filterregel mit benanntem Set wird vom der *nftables* Virtuellen-Maschine schneller bearbeitet als eine Reihe von Einzel-Regeln.

ANONYME UND BENANNTE MAPS UND DICTIONARIES

- Maps und Dictionaries existieren in anonymen (statischen) und benannten Versionen.
- Die anonymen Versionen werden direkt in die Regel geschrieben und sind damit fester Bestandteil der Regel.
- Die Inhalte von anonymen Datenstrukturen können nicht zu einem späeteren Zeipunk geändert werden.

ANONYME UND BENANNTE MAPS UND DICTIONARIES

- Bei den benannten Maps und Dictionaries können die Inhalte nach dem Laden des Firewall-Regelwerkes noch angepasst werden.
- So können z.B. Programme wie *fail2ban*, *crowdsec* oder *denyhost* direkt mit der Firewall interagieren und IP-Adressen von Angreifern sperren.
- Oder eine Anti-Spam-Dienst kann das Einliefern von Spam-E-Mail über die IP-Adressen von identifizierten Spammern mittels einer Rate-Limit-Regel drosseln.

MAPS

- Maps oder Data-Maps verbinden zwei Werte miteinander.
- Der Eingangswert wird in der Liste der Index-Werte der Map gesucht und der dort gespeicherte Wert wird an die Regel zurückgegeben.
- Eine Beispiel-Anwendung ist das Verzweigen auf verschiedene interne Server mit privaten Adressen per NAT, basierend auf der Port-Nummer des Dienstes:

```
# nft "add rule ip nat prerouting dnat" \
  "tcp dport map { 80 : 192.168.1.100, 8888 : 192.168.1.101 }"
```

VERDICT-MAPS/DICTIONARIES

- Verdict-Maps, auch Dictionaries genannt, verbinden einen Eingangswert des zu betrachtenen Pakets mit der Filter-Entscheidung einer Regel.
- Somit lassen sich für die verschiedenen Werte einers IP-Paketes automatisch unterschiedliche Aktionen auslösen.

VERDICT-MAPS/DICTIONARIES

- In diesem Beispiel werden die Ergebnisse einer Regel an die Quell-IPv4-Adresse des Netzwerk gebunden:

```
nft "add map inet filter aktion { type ipv4_addr : verdict; }"

nft "add element inet filter aktion" \
    "{ 192.0.2.80 : accept, 192.0.2.88 : drop, 192.0.2.99 : drop }"

nft "add rule inet filter input ip saddr vmap @aktion"
```

BEISPIEL: EIN REGELWERK FÜR EINE HOST-FIREWALL

- Das nachfolgende *nftable*-Regelwerk kann als Grundlage für eine Host-Firewall für eine Linux-Workstation benutzt werden.
- Gefiltert werden die eingehenden IPv4- und IPv6-Verbindungen.
- Gängige Dienste auf einer Linux-Workstation wie Internet Printing Protocol (IPP/Cups) und Multicast-DNS werden erlaubt:

BEISPIEL: EIN REGELWERK FÜR EINE HOST-FIREWALL

```
flush ruleset

define any = 0::0/0

table inet filter {
        chain input {
                type filter hook input priority 0;

                # accept any localhost traffic
                iif lo accept

                # accept traffic originated from us
                ct state established,related accept

                # activate the following line to accept common local services
                #tcp dport { 22, 80, 443 } ct state new accept

                # NTP multicast
                ip6 daddr ff02::1010 udp dport 123 accept

                # mDNS (avahi)
                ip6 daddr ff02::fb udp dport 5353 accept
                ip  daddr 224.0.0.251 udp dport 5353 accept

                # DHCPv6
                ip6 saddr $any udp dport 546 accept

                # IPP (CUPS)
                udp dport 631 accept

                # Accept neighbour discovery otherwise IPv6 connectivity breaks.
                ip6 saddr $any icmpv6 type { nd-neighbor-solicit,  nd-router-advert, nd-neighbor-advert }  accept

                # Accept essential icmpv6
                # nft cannot parse "param-problem" (icmpv6 type 4)
                ip6 saddr $any icmpv6 type { echo-reply, echo-request, packet-too-big, destination-unreachable, time-exceeded, 4 } accept

                # count and drop any other traffic
                counter log prefix "nftables drop: " drop
```

```
    }
}
```

BEISPIEL HOST-FIREWALL IM DETAIL

```
flush ruleset

define any = 0::0/0

table inet filter {
        chain input {
                type filter hook input priority 0;

                # accept any localhost traffic
                iif lo accept

                # accept traffic originated from us
                ct state established,related accept

...
```

BEISPIEL HOST-FIREWALL IM DETAIL

```
...
            # activate the following line to accept common local services
            #tcp dport { 22, 80, 443 } ct state new accept

            # NTP multicast
            ip6 daddr ff02::1010 udp dport 123 accept

            # mDNS (avahi)
            ip6 daddr ff02::fb udp dport 5353 accept
            ip  daddr 224.0.0.251 udp dport 5353 accept

            # DHCPv6
            ip6 saddr $any udp dport 546 accept

            # IPP (CUPS)
            udp dport 631 accept
...
```

BEISPIEL HOST-FIREWALL IM DETAIL

```
    ...
                # Accept neighbour discovery otherwise IPv6 connectivity breaks.
                ip6 saddr $any icmpv6 type {
                   nd-neighbor-solicit,
                   nd-router-advert,
                   nd-neighbor-advert
                }  accept

                # Accept essential icmpv6
                # nft cannot parse "param-problem" (icmpv6 type 4)
                ip6 saddr $any icmpv6 type {
                  echo-reply,
                  echo-request,
                  packet-too-big,
                  destination-unreachable,
                  time-exceeded, 4
                } accept

                # count and drop any other traffic
                counter log prefix "nftables drop: " drop
        }
}
```

BEISPIEL: EIN REGELWERK FÜR EINEN GATEWAY-ROUTER MIT IPV4-NAT

- Nachfolgend ein *nftables*-Regelwerk für eine einfache Firewall mit DMZ-, WAN- und LAN-Schittstelle. In der DMZ befinden sich die Server mit Web und E-Mail.
- In der Postrouting-Kette werden alle Pakete aus dem LAN-Segment, welche die Firewall in Richtung Internet (WAN) verlassen, per Masquerading-NAT auf die IPv4-Adresse der WAN-Netzwerkschnittelle umgeschrieben.

GATEWAY-ROUTER MIT IPV4-NAT

```
#!/usr/bin/nft -f
flush ruleset
define mgt_if = eth0
define wan_if = eth1
define lan_if = eth3
define dmz_if = eth2
define any_v6  = 0::0/0

table inet gateway-fw {
        chain forward {
                type filter hook forward priority 0
                # accept established traffic
                ct state established,related accept
                # allow all outgoing traffic from LAN
                iif $lan_if accept
                # access to services in the DMZ
                oif $dmz_if tcp dport { http, https, smtp, imap, imaps } counter accept
                # Accept essential icmpv6
                # nft cannot parse "param-problem" (icmpv6 type 4)
                ip6 saddr $any_v6 icmpv6 type { echo-reply, echo-request, packet-too-big, destination-unreachable, time-exceeded, 4 } accept
                # reject everything else
                counter log reject
       }
       chain input {
                type filter hook input priority 0
                iif lo accept
                iif $mgt_if tcp dport ssh accept
                ip6 saddr $any_v6 icmpv6 type { nd-neighbor-solicit,  nd-router-advert, nd-neighbor-advert }  accept
                ip6 saddr $any_v6 icmpv6 type { echo-reply, echo-request, packet-too-big, destination-unreachable, time-exceeded, 4 } accept
                counter log reject
       }
}
table ip nat {
        chain prerouting {
                type nat hook prerouting priority 0
        }
        chain postrouting {
                type nat hook postrouting priority 0
                oif $wan_if log masquerade
        }
}
```

DETAIL: GATEWAY-ROUTER MIT IPV4-NAT

```
#!/usr/bin/nft -f
flush ruleset
define mgt_if = eth0
define wan_if = eth1
define lan_if = eth3
define dmz_if = eth2
define any_v6  = 0::0/0
...
```

DETAIL: GATEWAY-ROUTER MIT IPV4-NAT

```
...
table inet gateway-fw {
        chain forward {
             type filter hook forward priority 0

             # accept established traffic
             ct state established,related accept

             # allow all outgoing traffic from LAN
             iif $lan_if accept

             # access to services in the DMZ
             oif $dmz_if tcp dport { http, https, smtp, imap, imaps } counter accept

             # Accept essential icmpv6
             # nft cannot parse "param-problem" (icmpv6 type 4)
             ip6 saddr $any_v6 icmpv6 type {
                  echo-reply, echo-request,
                  packet-too-big, destination-unreachable,
                  time-exceeded, 4 } accept

             # reject everything else
             counter log reject
    }
...
```

DETAIL: GATEWAY-ROUTER MIT IPV4-NAT

```
...
        chain input {
                type filter hook input priority 0

                iif lo accept

                iif $mgt_if tcp dport ssh accept

                ip6 saddr $any_v6 icmpv6 type {
                  nd-neighbor-solicit, nd-router-advert,
                  nd-neighbor-advert
                }  accept

                ip6 saddr $any_v6 icmpv6 type {
                  echo-reply, echo-request,
                  packet-too-big, destination-unreachable,
                  time-exceeded, 4
                } accept

                counter log reject
        }
}  # end table inet gateway-fw

...
```

DETAIL: GATEWAY-ROUTER MIT IPV4-NAT

```
...
table ip nat {

        chain prerouting {
                type nat hook prerouting priority 0
        }

        chain postrouting {
                type nat hook postrouting priority 0
                oif $wan_if log masquerade
        }
}
```

- Die leere Pre-Routing Kette ist notwendig, um die aus dem Internet kommenden Pakete per NAT zurück auf die privaten LAN-IPv4-Adressen umzuschreiben.

MIGRATION VON *IPTABLES* REGELN

- *nftables* bietet eine Kompatibilitätsschicht für bestehende *iptables*-Scripte.
- In aktuellen Linux Distributionen sind die *iptables* Befehle Links auf *xtables-nft-multi*, ein Tool welches *iptables* Regeln automatisch in *nft* Regeln umschreibt
    - In diesen Distributionen wird auf Kernel-Ebene immer *nft* benutzt

MIGRATION VON *IPTABLES* REGELN

- Diese Kommandos sind kompatibel zu den bisherigen *iptables* Befehlen, doch anstatt der *iptables*-Schnittstelle wird *nftables* benutzt.
- Die *iptables* Befehle werden dabei in den Bytecode für die *nftables* VM übersetzt.
    - Bestehende *iptables*-Regelwerke können geladen und mit dem *nft* Befehl als *nft*-Regelwerke ausgegeben werden