
15 Iptables (Netfilter)

In diesem Kapitel lernen Sie

- ▶ die Grundlagen des Paketfilters *iptables* kennen.
- ▶ aus welchen Bausteinen *iptables* besteht.
- ▶ welche Wege TCP/IP-Pakete durch einen als *Firewall* konzipierten Server gehen.
- ▶ Möglichkeiten, diese Wege zu manipulieren.
- ▶ die Begriffe *NAT* und *Masquerading* kennen.
- ▶ wie man ein eigenes kleines Skript programmiert, welches das Erstellen von Filterregeln erleichtert.

15.1 Grundlagen

Es ist beinahe Tradition, dass mit jedem neuen *Minor-Release* von Linux die Paketfilter- und Firewallfunktionalität erneuert wird.

So enthielten die Kernel 1.2.x bis 2.0.x als Paketfilter das aus der BSD-Welt stammende *ipfw* mit dem zugehörigen Tool **ipfwadm**, die Kernel der 2.2.x-Serie wurden mit *ipchains* und den Tools **ipchains** und **ipfwadm** zu Paketfiltern. Als „letzter Stand“ dieser Evolution hat der Kernel 2.4 die Firewall-Architektur *netfilter* mit dem dazugehörigen Userspace-Kommando **iptables** erhalten.

Ebenfalls Tradition ist es, die Funktionalität zu erweitern. Darunter hatte leider in der Vergangenheit auch stellenweise die Übersichtlichkeit gelitten. Mit der *netfilter*-Architektur und dem ihr zugeordneten Kommando **iptables** soll es diesmal anders aussehen.

Eine der wichtigsten Neuerungen von *iptables* ist die Aufteilung in verschiedene *Tabellen* die jeweils vordefinierte Regellisten (*chains*) enthalten:

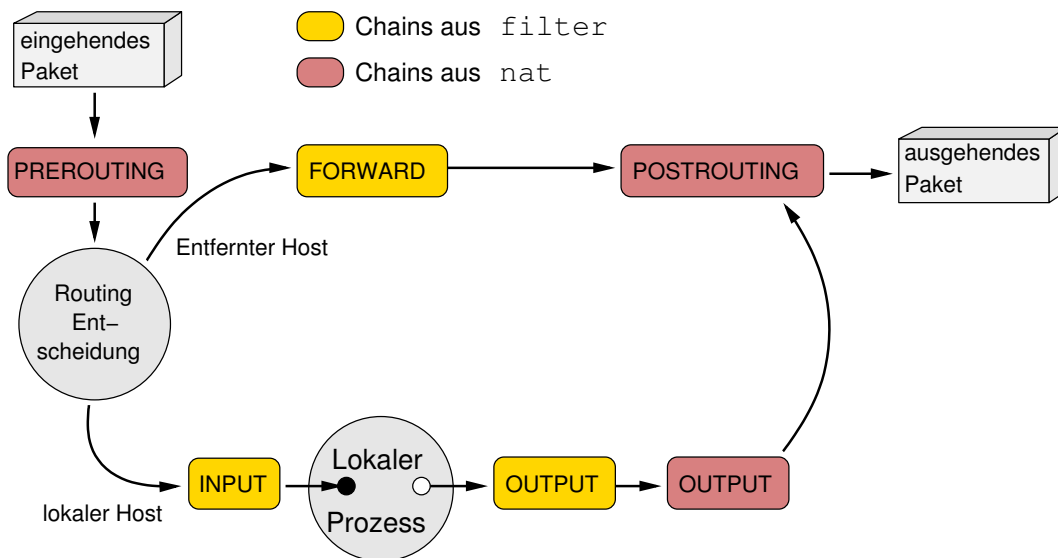
filter enthält die vordefinierten Regellisten INPUT, FORWARD und OUTPUT. *filter* ist die Default-Tabelle und wird für die üblichen Aufgaben eines Paketfilters verwendet.

nat enthält die vordefinierten Regellisten PREROUTING, OUTPUT und POSTROUTING. Diese Tabelle wird vor allem für Network Address Translation verwendet.

Iptables (Netfilter)

mangle enthält die vordefinierten Regellisten PREROUTING, INPUT, FORWARD, OUTPUT und POSTROUTING. Sie wird v.a. zum Markieren von Paketen genutzt, was z.B. paketweises Routing ermöglicht. Für die in dieser Unterlage besprochenen Anwendungen benötigen wir *mangle* nicht.

In der folgenden Abbildung beobachten wir den Weg von einem Paket durch den Linux-Kernel eines *iptables*-Firewall-Servers:



Jedes über ein Netzwerkinterface hereinkommende TCP/IP-Paket wird zunächst an die *Regelliste PREROUTING* geleitet. In dieser *Regelliste* lassen sich Zieladresse und -port des Pakets ändern.

Ist das Paket nicht für den Server selbst bestimmt, sondern soll an einen anderen Zielhost weitergeleitet werden, wird unser Beispielpaket an die *Regelliste FORWARD* gereicht. Nachdem mögliche Regeln in der *Regelliste FORWARD* abgearbeitet wurden, wandert unser Paket in die *Regelliste POSTROUTING*. Hier lassen sich Quelladresse bzw. -port manipulieren, bevor das Paket den Server verlässt und seine Reise zum Zielhost antritt.

Pakete, die nicht direkt weitergeleitet, sondern von unserem Server selbst verarbeitet werden sollen, wandern von der *Regelliste PREROUTING* in die *Regelliste INPUT* und werden nach Abarbeiten der dortigen Regeln an einen lokalen Prozess weitergeleitet.

Werden Pakete auf unserem Firewall-Server selbst generiert, so schickt der Kernel diese zunächst in die Regelliste *OUTPUT* der Tabelle *filter*. Von dort gelangt unser Paket in die Regelliste *OUTPUT* der Tabelle *nat* und wird zuletzt ebenfalls an die Regelliste *POSTROUTING* der Tabelle *nat* weitergereicht.

Voraussetzung für einen funktionierenden Paketfilter ist natürlich das Installieren des Pakets *iptables*. Des Weiteren sollten Sie überprüfen, ob alle notwendigen Kernelmodule geladen sind. Dies geschieht mit dem Befehl

lsmod | grep ip

Für den Anfang reicht es, wenn Sie die Module *ip_tables*, *iptables_filter* und *iptables_nat* in der Liste der Kernelmodule sehen.

```
# lsmod
Module                Size  Used by    Tainted: PF
iptables_nat          15470  0  (autoclean) (unused)
iptables_filter        1644  0  (autoclean) (unused)
ip_tables              11040  4  [iptables_nat iptables_filter]
```

☞ **Hinweis:** Mit Hilfe der Befehle **modprobe** bzw. **insmod** können Sie eventuell fehlende Module nachladen.

Der Befehl

```
# iptables -L -t filter
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

bzw.

```
# iptables -L -t nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source                destination
```

Iptables (Netfilter)

```
Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

verschafft Ihnen einen ersten Überblick und listet die Inhalte der verschiedenen *Tabellen* auf. Beachten Sie bitte, dass die Regellisten *OUTPUT* der beiden *Tabellen filter* und *nat* zwar gleich heißen, jedoch unterschiedliche Regellisten darstellen.

Folgende Auflistung soll noch einmal die unterschiedlichen Wege darstellen, die ein TCP/IP-Paket in der *netfilter*-Architektur gehen kann:

von einem Netzwerkinterface kommend, an einen anderen Rechner weitergeleitet:

```
PREROUTING (nat) → FORWARD (filter) → POSTROUTING (nat)
```

von einem Netzwerkinterface kommend, für den eigenen Host:

```
PREROUTING (nat) → INPUT (filter)
```

vom eigenen Host generiert, an ein Netzwerkinterface:

```
OUTPUT (filter) → OUTPUT (nat) → POSTROUTING (nat)
```

Sobald Sie eigene Regeln für *Regellisten* in den verschiedenen *tables* definieren, sollten Sie sich diese Wege immer vor Augen halten, damit nicht irrtümlich Pakete geblockt werden, die akzeptiert werden sollten, oder – schlimmer – Pakete durchschlüpfen können, die Sie eigentlich sperren wollten.

15.2 Regeln erstellen und bearbeiten

Die Basis ist geschaffen, wie aber erstelle ich nun eigene Regeln für einen funktionierenden Paketfilter? Einen Befehl haben Sie bereits im vorherigen Abschnitt kennen gelernt:

```
iptables -L -t filter bzw. iptables -L -t nat
```

Schauen wir uns den Output dieses Befehls etwas genauer an:

```
# iptables -L -t filter
Chain INPUT (policy ACCEPT)
target      prot opt source                destination

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

Wir sehen hier die drei *Regellisten* INPUT, FORWARD und OUTPUT der Tabelle *filter*. Alle drei Regellisten haben die Standardpolicy *ACCEPT*, d.h. alle Pakete, welche diese Regellisten durchlaufen, werden grundsätzlich akzeptiert. Aufgelistet werden das Ziel (*target*), der Protokolltyp des Pakets (*prot*), mögliche Optionen (*opt*) sowie Quelle (*source*) und Ziel (*destination*) unserer Pakete. Offensichtlich enthält jedoch keine der Regellisten eine Regel.

Der Befehl **iptables** wird generell benutzt, um Regeln zu erstellen, aufzulisten, sie zu manipulieren oder zu löschen. Die allgemeine Syntax lautet:

```
# iptables [-t Tabelle] -[Aktion] [Regelliste] \  
> [Optionen] -j [Ziel]
```

Ein einfaches Beispiel soll uns weiterhelfen: Wir möchten verhindern, dass hereinkommende Pakete mit der Quelle 192.168.1.0/24 unseren Zielhost erreichen. Die Befehl hierzu lautet:

```
# iptables -t filter -A INPUT -s 192.168.1.0/24 -j DROP
```

Was genau ist hier passiert? Wir haben mit dem Befehl **iptables** der Tabelle *filter* in der Regelliste *INPUT* eine Regel hinzugefügt (**-t filter -A INPUT**). Die Regel besagt, dass Pakete mit der Quelle (*source*) 192.168.1.0/24 (**-s 192.168.1.0/24**) verworfen werden sollen (**-j DROP**).²⁸

Was zeigt uns nun der Output von **iptables -L -t filter** an?

```
Chain INPUT (policy ACCEPT)  
target      prot opt source                destination  
DROP        all  --  192.168.1.0/24        anywhere  
  
Chain FORWARD (policy ACCEPT)  
target      prot opt source                destination  
  
Chain OUTPUT (policy ACCEPT)  
target      prot opt source                destination
```

Aha! Es hat sich etwas getan. In die Regelliste *INPUT* ist eine Regel hinzugefügt worden, die alle Pakete mit der Quelladresse 192.168.1.0/24 unabhängig von Ziel und Protokolltyp verwirft.

Die wichtigsten Parameter sind in den folgenden Tabellen aufgelistet.


²⁸Die Netzmaske kann sowohl in Bit-Schreibweise, wie hier geschehen, als auch „herkömmlich“ mit 255.255.255.0 angegeben werden.

Iptables (Netfilter)

- A** Eine Regel hinzufügen (add)
 - D** Eine Regel löschen (delete)
 - F** Alle Regeln löschen (flush)
 - R** Eine Regel ersetzen (replace)
 - I** Eine Regel einfügen (insert)
 - L** Alle Regeln auflisten (list)
 - P** Die Standardpolicy ändern (policy) [ACCEPT, DROP]
-
- s** von der Quelle (source)
 - d** mit dem Ziel (destination)
 - p** Protokolltyp (protocol) [tcp, udp, icmp oder numerische Angabe]
 - sport** mit dem Quellport
 - dport** zum Zielport
 - i** vom hereinkommenden Interface (in)
 - o** mit dem herausgehenden Interface (out)
 - j** mit dem Ziel (jump to)

- DROP** ein Paket wird verworfen
- REJECT** wie DROP, jedoch wird der Absender benachrichtigt
- ACCEPT** ein Paket wird akzeptiert
- LOG** ein Paket wird geloggt

Grundsätzlich gilt: Wird eine *Tabelle* nicht explizit angegeben, so ist immer die Tabelle *filter* gemeint (Defaulteinstellung). Wird bei den Parametern **-F** und **-L** keine Regelliste angegeben, so sind alle Regellisten der Tabelle gemeint. Besonders aufpassen sollten Sie deshalb beim „flushen“ einer Regelliste.

Hinweis: Wenn Sie in einer Regel Quell- oder Zielport angeben möchten, so müssen Sie zwingend ebenfalls den Protokolltyp in der Regel definieren. 

Wenn in der Konfiguration des *Syslog-Daemon* nichts geändert wurde, werden die Logs in die Datei `/var/log/messages` geschrieben.

Einige Beispiele sollen das eben Gelernte vertiefen. Gehen Sie in allen Beispielen davon aus, dass `eth0` das nach innen zeigende Interface (192.168.0.1) und `eth1` das nach außen zeigende Interface ist. Versuchen Sie zunächst, die Regel mit eigenen Worten wiederzugeben, bevor Sie den erläuternden Text lesen.

```
# iptables -A FORWARD -p tcp -s 192.168.1.0/24 -j REJECT
```

In dieser Regel sollen alle TCP-Pakete, deren Quelle das Netzwerk 192.168.1.0/24 ist, abgelehnt werden. Der Sender wird jedoch informiert.

```
# iptables -R INPUT 3 -p icmp -i eth0 -j DROP
```

Hier wird die 3. Regel der Regelliste *INPUT* ersetzt. Nun sollen alle ICMP-Pakete, die über das Interface `ppp0` hereinkommen, verworfen werden. Der Sender erhält keine Nachricht über den Verbleib der Pakete.

```
# iptables -I INPUT 2 -p tcp --dport 22 -i eth+ \  
> -j ACCEPT
```

Diese Regel soll an der 2. Position der Regelliste *INPUT* eingefügt werden. TCP-Pakete, die über alle Ethernet-Netzwerkkarten²⁹ hereinkommen und als Ziel den Port 22 (secure shell) haben, werden akzeptiert.

```
# iptables -D INPUT -s 192.168.1.0/24 -j DROP
```

Die Regel, die besagt, dass alle Pakete aus dem Netzwerk `192.168.1.0/24` verworfen werden sollen, wird gelöscht.

```
# iptables -A INPUT -s 192.168.0.0/24 -i eth1 -j LOG
```

Pakete, die als Quelle das Netzwerk `192.168.0.0/24` angeben und über das Netzwerkkinterface `eth1` hereinkommen, werden geloggt.

```
# iptables -A INPUT -s 127.0.0.0/24 -i eth1 -j DROP
```

Pakete, die als Quelle das *Loopback*-Interface angeben, jedoch über die Netzwerkschnittstelle `eth1` hereinkommen, werden verworfen.

```
# iptables -A INPUT -s 192.168.0.0/24 -i eth1 -j DROP
```

Alle Pakete, die als Quelle das Netzwerk `192.168.0.0/24` angeben, jedoch über die Schnittstelle `eth1` kommen, werden ebenfalls verworfen.



Tip: Die drei letzten Regeln sind ein typisches und praktisches Beispiel für die Funktionsweise von `iptables`. Was soll hier erreicht werden?

Grundsätzlich stellt das *Spoofen* von IP-Adressen ein Problem für Paketfilter dar. Spoofen bedeutet hier, dass der Header des TCP/IP-Pakets eine gefälschte IP-Adresse

²⁹+ wird hier als Wildcard interpretiert

enthält. Solche gefälschten Pakete sind relativ einfach herzustellen und dafür umso gefährlicher.

Es nützt wenig, eine Regel basierend auf IP-Adresse aufzustellen, wenn ein TCP/IP-Paket mit einem gefälschten *Header* und einer „erlaubten“ IP-Adresse über ein anderes Netzwerkinterface hereinkommt.

TCP/IP-Pakete mit der Netzwerkadresse 192.168.0.0 dürfen eigentlich nur im internen Netzwerk entstehen, folglich auch nur an der Netzwerkschnittstelle `eth0` erscheinen. Ebenso darf ein Paket mit der Adresse des Loopbackdevices nicht an der externen Schnittstelle auftauchen. Die drei letzten Beispielregeln verhindern genau dies. Hier wird nicht nur überprüft, welche IP-Adresse im Header eines TCP/IP-Pakets enthalten ist, sondern auch, über welche Netzwerkschnittstelle dieses Paket auf unseren Server gelangt.

15.3 Erweiterungen

Die oben vorgestellten Parameter zeigen aber noch längst nicht alles, was mit Hilfe des Befehls `iptables` gefiltert werden kann. Im diesem Abschnitt zeigen wir Ihnen weitere Möglichkeiten.

15.3.1 Negieren von Merkmalen

Oft ist es notwendig, ein zu filterndes Merkmal zu negieren. Dies geschieht mit Hilfe eines vorangestellten „!“: Ein Beispiel:

```
# iptables -A INPUT -s 192.168.0.0/24 -i ! eth0 -j LOG
```

Diese Regel protokolliert alle Pakete, deren Ursprung das Netzwerk `192.168.0.0/24` ist und die **nicht** über das Interface `eth0` hereinkommen.

15.3.2 Flags im TCP-Header

Manche Pakete möchte man anhand ihres gesetzten Flags filtern. So kann es z.B. wünschenswert sein, Verbindungsaufbauten zu Diensten zu verhindern, bestehende Verbindungen jedoch durchzulassen. Das gesetzte *Syn-Bit* in einem TCP-Header bedeutet, dass ein Verbindungsaufbau gewünscht wird. Hier hilft uns der Parameter `-syn` weiter.

```
# iptables -A INPUT -p tcp --dport ! 22 --syn -j DROP
```

In diesem Beispiel werden alle Pakete verworfen, die ein Syn-Bit gesetzt haben, und deren Ziel nicht ein Verbindungsaufbau zum SSH-Server ist. Wichtig ist hier der Zusatz **-p tcp**, ohne den unsere Regel nicht funktionieren würde.

Möchte man nach weiteren FLAGS filtern, benutzt man den Zusatz **-tcp-flags**. *FLAGS* können hier die Werte **SYN, ACK, FIN, RST, PSH, URG, ALL** oder **NONE** sein. *ALL* steht für alle Flags, *NONE* für keines. Nach **-tcp-flags** gibt man zwei komma-separierte Listen mit Flags an. Alle Flags in der ersten Liste werden überprüft. Flags aus der ersten Liste die auch in Liste 2 auftauchen, müssen gesetzt sein. Alle Flags aus Liste 1 die *nicht* in Liste 2 auftauchen, dürfen nicht gesetzt sein. Auch hierzu ein Beispiel:

```
#: iptables -A INPUT -p tcp --dport 22 --tcp-flags \  
> SYN,ACK,FIN SYN -j LOG
```

Diese Regel loggt alle TCP-Pakete mit dem Zielport 22 (*ssh*) bei denen das SYN-Flag gesetzt, die Flags ACK und FIN aber nicht gesetzt sind. Auch in diesem Beispiel muss zwingend der Zusatz **-p tcp** verwendet werden.

15.3.3 ICMP-Pakete

Manchmal sehr notwendig zur Ursachenforschung für Fehler, manchmal sehr lästig sind ICMP-Pakete. Auch diese lassen sich auf bestimmte Merkmale hin filtern. Dies geschieht mit dem Zusatz **-icmp-type** und – analog zu den obigen Beispielen – mit einem vorangestellten **-p icmp**. Die verschiedenen ICMP-Typen erfahren Sie mit Hilfe des Befehls:

```
# iptables -p icmp -h  
Valid ICMP Types:  
echo-reply (pong)  
destination-unreachable  
    network-unreachable  
    host-unreachable  
    protocol-unreachable  
    port-unreachable  
    fragmentation-needed  
    source-route-failed  
    network-unknown  
    host-unknown  
    network-prohibited  
    host-prohibited  
    TOS-network-unreachable
```

Iptables (Netfilter)

```
TOS-host-unreachable
communication-prohibited
host-precedence-violation
precedence-cutoff
source-quench
redirect
network-redirect
host-redirect
TOS-network-redirect
TOS-host-redirect
echo-request (ping)
router-advertisement
router-solicitation
time-exceeded (ttl-exceeded)
ttl-zero-during-transit
ttl-zero-during-reassembly
parameter-problem
ip-header-bad
required-option-missing
timestamp-request
timestamp-reply
address-mask-request
address-mask-reply
```

Die Ausgabe dieses Befehls wurde aus Platzgründen leicht verkürzt.

Wir wollen z.B. das Senden und Empfangen von ICMP-Paketen unterbinden, es dem Rechner aber ermöglichen, den **ping**-Befehle zu verwenden. Zunächst müssen wir dazu `echo-request` Pakete nach außen durchlassen, damit **ping** eine ICMP-Anforderung senden kann:

```
# iptables -A OUTPUT -p icmp --icmp-type echo-request \
> -j ACCEPT
```

Alle anderen ICMP-Pakete lassen wir nicht durch:

```
# iptables -A OUTPUT -p icmp -j DROP
```

Dann müssen wir `echo-reply` Pakete von außen akzeptieren, damit **ping** die Antwort des angepingten Rechners empfangen kann:

```
# iptables -A INPUT -p icmp --icmp-type echo-reply \
> -j ACCEPT
```