SelfLinux-0.13.1

IP-Tables

Autor: Gabriel Welsche (*gabriel.welsche@web.de*)
Formatierung: Matthias Hagedorn (*matthias.hagedorn@selflinux.org*)

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel beschäftigt sich vorrangig mit Paketfiltern. Obwohl ein knapper Einstieg gegeben wird, ist es sinnvoll, wenn detaillierte Kenntnisse zum Thema **Netzwerke** (Topologien, Routing, QoS, Nat, Ports) als auch Grundlagen des Kernels vorhanden sind.

2 Grundlegende Begriffe

Um Doppelungen zu vermeiden, soll an dieser Stelle auf zwei wesentliche Selflinux Kapitel hingewiesen werden, welche die grundlegenden Begriffe erläutern:

OSI-Referenzmodell

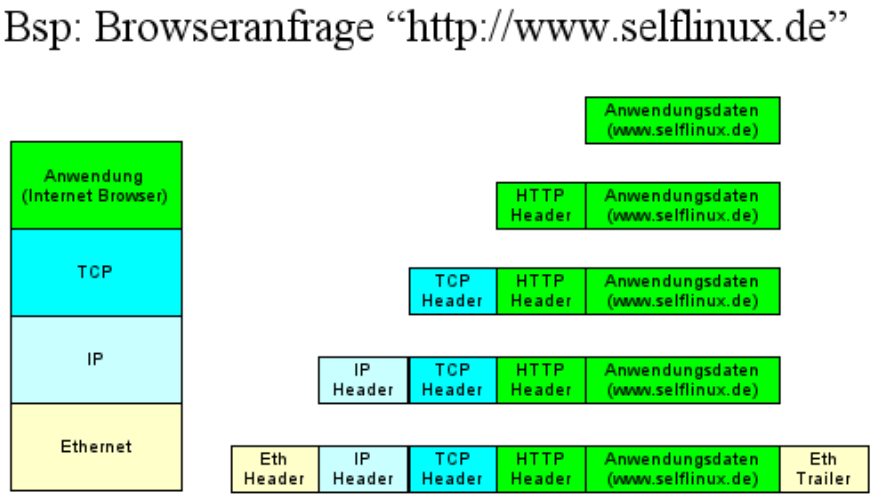
TCP/IP (TCP/IP-Schichtenmodell, Adressierung, Protokolle TCP/UDP/ICMP/IP/ARP, Ports)

3 Grundlegendes zu Paketfiltern

Die Aufgaben und der Zweck eines Paketfilters wurden bereits im Kapitel Überblick zur Sicherheit von Linux-Systemen gründlich vorgestellt. Ebenso wurde eindringlich davor gewarnt, einen Paketfilter als einzige Maßnahme zur Sicherung des heimischen Computers bzw. des kleinen Firmennetzwerks einzusetzen.

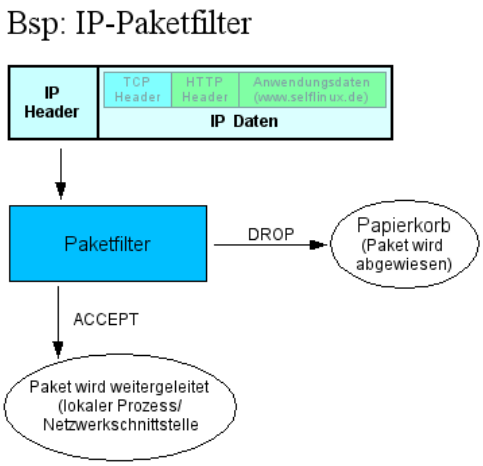
Was ist nun ein Paketfilter und wie arbeitet dieser?

Bei paketorientierten Netzwerkprotokollen (z.B. ip, tcp, udp) werden die Daten in Pakete verpackt und ein Header (Paketkopf) vorangestellt.



Paketfilter werten den Paketkopf (Header) aus und entscheiden anhand dessen über das weitere Schicksal des ganzen Paketes. Entweder wird das Paket verworfen (Drop), das bedeutet, es wird einfach gelöscht, oder es wird akzeptiert (ACCEPT), das heißt es passiert den Filter, oder es wird etwas Komplizierteres gemacht, auf das ich

später eingehen werde.



Bei Linux ist ein Paketfilter im Kernel integriert (einkompiliert oder als eingebundenes Modul). Mit Hilfe eines Programms (z.B. `ipchains` für Kernel 2.2 oder `iptables` für Kernel 2.4/2.6) kann man mit dem Filter kommunizieren. Dies beinhaltet unter anderem das Aufstellen von Regeln, mit denen man definieren kann, welche Pakete den Filter unter welcher Voraussetzung passieren dürfen.

4 Was leistet ein Paketfilter?

4.1 Kontrolle und Sicherheit

Ein Paketfilter kann anhand des Pakettyps, anhand des Herkunftsortes und anhand weiterer Eigenschaften des Paketes bestimmte Kommunikationsarten erlauben oder verbieten. Die Entscheidung basiert nicht auf dem Inhalt des Paketes!

Ein Beispiel:
Die meisten Internetbenutzer wollen keine eigenen Dienste vom heimischen PC aus anbieten. Es sollte also niemandem aus dem Internet heraus erlaubt sein, eine Verbindung zu dem privaten PC aufzubauen.

Zweites Beispiel:
Ein mittelständiges Unternehmen unterhält eine eigene kleine Entwicklungsabteilung, deren Erkenntnisse einer gewissen Geheimhaltungsstufe unterliegen:
Die Dokumente sollen nicht in die Hände der Konkurrenz fallen. Das schwierige an der Sache ist, dass die Mitarbeiter für ihre Arbeit einen WWW-Zugang benötigen. Ein Paketfilter hat die Aufgabe den Datenaustausch von geheimen Dokumenten zu erschweren. Wie - dies wird später erläutert.

Paketfilter können auch sehr differenziert entscheiden, wer etwas darf und wer nicht.

4.2 Beobachtung des Netzverkehres

Paketfilter können auch den Datenverkehr überwachen und bei Unregelmäßigkeiten eine Meldung erzeugen.

Ein Beispiel:
Nachts drei Uhr wird eine Internetverbindung aufgebaut. Da um diese Zeit keiner in der Firma sein dürfte, liegt die Vermutung nahe, es könnte ein Virus oder ein Trojanisches Pferd ungewollte Dinge verrichten.

4.3 Paketumleitung und Paketmanipulation

Paketfilter können den Datenverkehr auf einen Proxy umleiten, der auf Applikationsebene Entscheidungen fällt. Damit lassen sich beispielsweise transparente Virenscanner oder Kindersicherungen einrichten, und zwar ohne dass die Netzwerkteilnehmer (also z.B. Schüler) etwas davon merken.

Ein weiteres Einsatzgebiet stellt NAT bzw. IP-Masquerading dar. Damit lassen sich beispielsweise viele Rechner über eine einzige ISDN- oder DSL-Verbindung mit dem Internet verbinden.

4.4 Attackenbekämpfung

Prinzipiell werden zwei Angriffstypen unterschieden:

Zum einen sind dies Attacken mit dem Ziel, in ein System einzudringen, um dann Daten zu stehlen oder es als Ausgangspunkt für weitere Angriffe zu nutzen. Dazu nutzen die Angreifer Schwachstellen im Betriebssystem oder in Anwendungsprogrammen (Client- und Serversoftware).

Zum anderen besteht das Angriffsziel darin, einen Rechner lahm zu legen oder diesen vom Netzwerk zu trennen. Solche Attacken bezeichnet man auch als DOS- bzw. ihre Weiterentwicklung als DDOS-Attacken (Distributed Denial of Service Attack).

Paketfilter können einige Angriffsvarianten vereiteln, so zum Beispiel TCP-SYN-Flooding-Attacken oder Large Packet Attacks (Ping of Death). Man kann auch das Scannen des Netzwerkes verhindern. Mit dieser Technik versuchen Angreifer, so viel wie möglich Informationen über das System zu erhalten. Ebenfalls erschweren kann man IP-Spoofing Attacken durch das Aufstellen genereller Filterregeln (z.B. Pakete mit interner Absenderadresse dürfen nicht von Außen nach Innen gelangen.)

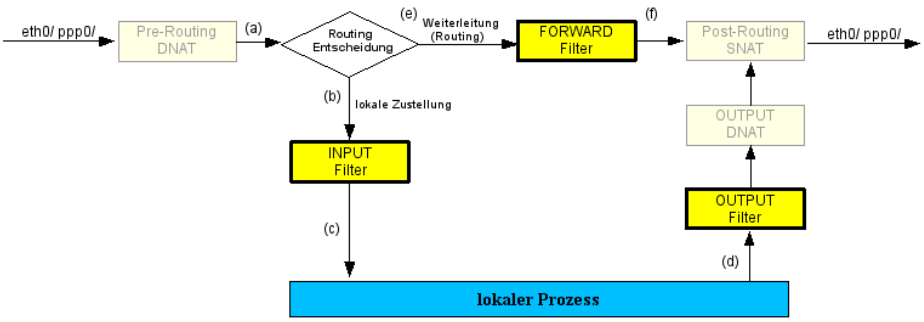
5 Prinzipielle Funktionsweise unter Linux

Folgende Erklärungen beziehen sich auf 2.4er/2.6er Kernel, bei früheren Versionen gab es andere Konzepte (`ipchains`, `ipfw`).

Zuerst benötigt man einen Kernel mit der Netfilter-Infrastruktur, das bedeutet, man braucht das `iptables` Kernelmodul und ein `CONFIG_NETFILTER=Y` in der Kernelkonfiguration.

Das Programm `iptables` kommuniziert mit dem Linux-Kernel und weist diesen an, Pakete nach bestimmten Regeln zu filtern. `iptables` übernimmt also unter anderem das Einfügen, Löschen und Manipulieren von Regeln in die Filtertabellen des Kernels sowie das Setzen der Filterpolitik. Bei einem Neustart des Computers gehen alle angelegten Regeln verloren, deshalb ist ein automatisches Setzen von Regeln sinnvoll. Mit den Tools `iptables-save` und `iptables-restore` können die Regeln gesichert und wiederhergestellt werden. Beim Bootvorgang übernimmt dies ein Init-Script.

Im 2.4er/2.6er Linuxkernel werden Listen von Regeln (Firewall-Ketten) in einer Filtertabelle verwaltet. Standardmäßig sind INPUT, OUTPUT und FORWARD-Ketten vorhanden. Der genaue Weg der Pakete wird später beschrieben, das folgende Diagramm gibt schon mal einen ersten Überblick:

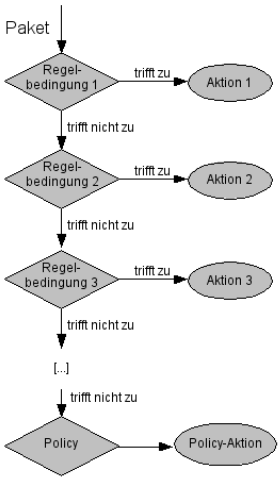


Jede Kette (INPUT, OUTPUT, FORWARD) entscheidet anhand von Regeln, ob ein Paket gelöscht oder akzeptiert wird. Wird es akzeptiert, dann reist es weiter durch das Diagramm bis es letztendlich den Rechner über den Ausgang verlassen kann.

Jede Kette beinhaltet eine Checkliste von Regeln, die ungefähr so aussehen:

WENN (Filteroption) DANN Aktion (Löschen, Akzeptieren, ... des Paketes)

Die Regeln werden nacheinander solange durchlaufen, bis eine Regel auf das eingegangene Paket zutrifft.



Ist das Ende der Regelkette erreicht (keine Regel traf zu) entscheidet die ***Policy*** der Kette (zu deutsch: Politik, Taktik, Vorgehensweise) darüber, was mit dem Paket zu tun ist. In einem sicherheitsbewussten System werden alle Pakete verworfen, für die keine Regel zutrifft. (prohibitive Sicherheitspolitik)

Nun soll noch einmal der Weg der Pakete durch den Paketfilter ganz genau betrachtet werden:

Nach Eingang eines Paketes an einer Netzwerkschnittstelle (a) wird dessen Zieladresse ausgewertet (Routing). Ein Paket mit Zieladresse dieses Rechners (b) wird anschließend durch die INPUT-Kette geprüft. Diese kann das Paket entweder verwerfen oder für die weitere Verarbeitung durchlassen. Letztendlich kann eine Anwendung die empfangenen Daten dem Nutzer zur Verfügung stellen (anzeigen, abspeichern, ...).

Ein lokaler Prozess (z.B. Mozilla) könnte ein neues Paket (Anforderung einer Webseite) erzeugen und dieses ins Netzwerk versenden (d). Die OUTPUT-Kette leitet nach einer erfolgreichen Prüfung das Paket weiter zur gewünschten Netzwerkschnittstelle (z.B. `eth0` / `ippp0`).

Kommunikationsserver oder Router verbinden meist ein inneres (privates) Netzwerk mit einem äußeren (öffentlichen) Netz, zum Beispiel dem Internet. Dafür wird die Paketweiterleitung von einer Netzwerkschnittstelle (Eingang) zu einer anderen (Ausgang) benötigt. Die FORWARD-Kette prüft in diesem Fall nun die Pakete, die vom inneren ins äußere Netz oder umgekehrt (f), (g) übertragen werden sollen.

Es sei an dieser Stelle darauf hingewiesen, dass es neben der Filtertabelle (INPUT-, OUTPUT-, FORWARD-Kette) auch noch eine NAT-Tabelle (PREROUTING-, POSTROUTING-, OUTPUT-Kette) für die sogenannte **Network Address Translation** existiert, welches später beschrieben ist. Ebenso gibt es auch noch eine MANGLE Tabelle mit PREROUTING- und OUTPUT-Kette zur Paketmanipulation

6 Praktischer Einstieg in iptables

6.1 Grundlegendes zu iptables

Wie bereits erwähnt sind nach dem Booten keine Regeln in einer der eingebauten Ketten (INPUT, OUTPUT, FORWARD) vorhanden, und die Policy (Sicherheitspolitik) steht standardmäßig auf **ACCEPT**.

(Achtung: manche Distributionen haben iptables-Befehle in den Init-Skripten)

Die Standard-Policy kann geändert werden, wenn das `iptables_filter` Kernelmodul mit einer entsprechende Option gestartet wurde (z.B. für FORWARD-Kette: `forward=0`).

Das Programm `iptables` bietet folgende Optionen zur Verwaltung von Filterregelketten:

* Eine neue Kette erstellen (`-N`).
* Eine leere Kette löschen (`-X`). (geht nicht mit INPUT, OUTPUT und FORWARD)
* Die Policy für eine eingebaute Kette ändern (`-P`).
* Die Regeln einer Kette auflisten (`-L`).
* Die Regeln aus einer Kette entfernen (flush) (`-F`).
* Paket und Bytezähler aller Regeln einer Kette auf Null stellen (`-Z`).

Verwaltung der Regeln in einer Kette:

* Eine neue Regel an eine Kette anhängen (`-A`).
* Eine neue Regel an eine bestimmte Position in der Kette einfügen (`-I`).
* Eine Regel an bestimmter Position in der Kette ersetzen (`-R`).
* Eine Regel an einer bestimmten Position in der Kette löschen (`-D`).
* Die erste passende Regel in einer Kette löschen (`-D`).

6.2 Operationen auf einer einzelnen Regel (Regeln manipulieren)

Mit den Befehlen zum Anhängen (`-A`) oder Löschen (`-D`) einer Regel stehen die grundlegenden Kommandos zur Verfügung. Weitere Befehle zum Einfügen (`-I`) und zum Ersetzen (`-R`) sind einfache Erweiterungen, die das Leben manchmal erleichtern und die dennoch kaum gebraucht werden.

Jede Regel definiert eine Menge von Bedingungen, die ein eintreffendes Paket erfüllen muss, und ein Ziel (target), welches das weitere Schicksal des Paketes bestimmt.

Zum Beispiel:

Es sollen alle ICMP-Pakete (z.B. von `ping` generiert) verworfen werden, die vom lokalen Rechner 127.0.0.1 stammen:

Bedingungen: Protokoll ICMP und Quelladresse 127.0.0.1
Ziel: ist Verwerfen (DROP).

Ping sendet einen `echo request` (ICMP Typ 8) und alle kooperierenden Hosts antworten darauf verbindlich mit einem `echo reply` (ICMP Typ 0).

ohne Filter

```
root@linux / # ping -c 1 127.0.0.1

PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
```

Jetzt wird die INPUT-Kette um eine Regel erweitert (`-A`), so dass Pakete, die von 127.0.0.1 (`-s 127.0.0.1`) kommen und das Protokoll ICMP (`-p icmp`) verwenden, zu verwerfen sind (`-j DROP`):

```
root@linux / # iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

Test

```
root@linux / # ping -c 1 127.0.0.1

PING 127.0.0.1 (127.0.0.1): 56 data bytes

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Die Regel scheint zu funktionieren. Doch wie löscht man nun diese Regel wieder? Wenn die Position der Regel in der Kette bekannt ist (in diesem Fall gibt es nur die eine Regel, deshalb Position 1), kann man diese Position als Parameter verwenden:

```
root@linux / # iptables -D INPUT 1
```

Sollte die Position jedoch nicht bekannt sein, so muss man die Regel nochmals vollständig angeben, also genau wie beim Anhängen der Regel, nur das anstelle des Befehls `-A` der Befehl `-D` steht:

```
root@linux / # iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
```

Wenn die Regel mehrfach in Regelkette aufgenommen wurde, so wird nur die erste gelöscht. Man sollte also IMMER nach dem Entfernen nochmals alles kontrollieren! (In diesem Falle also mit `ping`)

6.3 Schnelle Lösung ohne viele Worte

Viele Leser nutzen nur Modem/ISDN oder DSL-Verbindung zum Internet und die einzige Sorge ist, dass niemand vom Internet in ihren Computer oder privates Netzwerk kommen kann. Die folgende Anleitung funktioniert mit einem 2.4er/2.6er Kernel.

1) Kernelmodule einfügen, wenn diese noch nicht einkompiliert wurden (notwendig wegen Verbindungsverfolgung)

```
root@linux / # modprobe ip_conntrack
root@linux / # modprobe ip_conntrack_ftp
```

2) Befehle ausführen (am besten ein SysV-Initskript erstellen, welches beim Hochfahren des Rechners automatisch gestartet wird)

SysV-Initskript

```
# alle Verbindung blocken, es sei denn, sie kommen von innen

iptables -N block
iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
iptables -A block -j DROP

# Von INPUT und FORWARD Ketten zu dieser Kette springen

iptables -A INPUT -j block
iptables -A FORWARD -j block
```

7 Filterbestimmungen (Filteroptionen)

7.1 Grundlegende Filteroptionen

Wie bereits erwähnt, beinhaltet jede Kette (INPUT, OUTPUT, FORWARD) eine Checkliste von Regeln, die folgendermaßen aussehen:

WENN (Filteroption) DANN Aktion (Löschen, Akzeptieren, ... des Paketes)

Im letzten Abschnitt wurden bereits zwei Filteroptionen (`-s` und `-p`) für die INPUT-Kette angewendet:

Verwerfe alle ankommenden ICMP Pakete von localhost

```
root@linux / # iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

Mit `-p` wurde das Protokoll ICMP bestimmt, die `-s` Option legte die Quelladresse der Pakete fest.

Die folgenden Filteroptionen können angegeben werden:

* Quell- und Zieladresse festlegen:
  `-s`, `--source` oder `-src` - Quelladresse
  `-d`, `--destination` oder `--dst` - Zieladresse
  Quell- und Zieladressen können folgendermaßen angegeben werden:
  1. Namen (z.B. "localhost" oder "www.mydomain.com")
  2. IP-Adresse (Bsp. "127.0.0.1")
  3. Subnetz ("192.168.1.0/24" oder "192.168.1.0/255.255.255.0" sind alle IP-Adressen von 192.168.1.0 bis 192.168.1.255)
     Bsp:
     `iptables -A INPUT -s 192.168.1.0/24 -j DROP`
* Protokoll
  `-p` oder `--protocol`. Die Angabe des Protokolls erfolgt durch
  1. Protokollnamen z.B. "TCP", "UDP" oder "ICMP" (Gross- und Kleinschreibung ist egal, es funktioniert beides) oder
  2. die IP-Protokollnummer
     Bsp:
     `iptables -A INPUT -s 192.168.1.0/24 -p ! icmp -j DROP`
* Netzwerkschnittstelle
  `-i` oder `--in-interface` Eingangsschnittstelle
  `-o` oder `--out-interface` Ausgangsschnittstelle
  Mit dem `ifconfig` Kommando können alle aktiven Schnittstellen angezeigt werden.
  Die INPUT-Kette stellt keine Output-Schnittstelle zur Verfügung (siehe Bild oben), in dieser Kette ist die `-o` Option also zwecklos. Analog dazu hat die OUTPUT-Kette keine INPUT-Schnittstelle, also werden Regeln mit `-i` Option niemals zutreffen und sind deshalb ebenso obsolet. Nur Pakete, welche die FORWARD-Kette durchwandern, können über INPUT- und OUTPUT-Schnittstelle gefiltert werden.
  Es können auch Schnittstellen zuordnet werden, die beim Anlegen der Regel noch gar nicht existieren (Dialup-PPP Schnittstellen) Kein Paket kann bis zur Aktivierung der Schnittstelle die Regel erfüllen, sie stört somit also keinesfalls.
  Schnittstellennamen, die mit einem `+` enden, bezeichnen alle Schnittstellen, die mit dieser Zeichenkette

beginnen, egal ob diese in dem Moment existieren oder nicht.
Beispiel: Eine Regel mit der folgenden Option trifft auf alle PPP-Schnittstellen zu: `-i ppp+`

* Inversion (Verneinung) Jede Option ist invertierbar durch Angabe eines `!` hinter der jeweiligen Option:
Bsp:
```
iptables -A INPUT -s ! 127.0.0.1 -p icmp -j DROP
```
Die Regel gilt für alle ICMP-Pakete, die NICHT vom lokalen Rechner kommen.

7.2 Fragmente filtern

Zu große Pakete werden in Fragmente aufgeteilt und in mehreren Paketen weiterverschickt. Die Gegenstelle setzt diese Fragmente wieder zusammen, um das gesamte Paket zu rekonstruieren.

Bei **connection tracking** oder NAT werden alle Fragmente wieder miteinander verschmolzen, bevor sie den Paketfilter erreichen, alles ist also wie gehabt.

Ansonsten hat unseren Paketfilter nun folgendes Problem: das erste Fragment enthält die kompletten Header-Felder (IP + TCP, UDP und ICMP), die nachfolgenden Fragmente besitzen nur Teilstücke der Header (IP ohne die zusätzlichen Protokoll-Felder) Daher ist es nicht möglich, in nachfolgenden Fragmenten nach Protokoll-Headern zu suchen (wie es zum Beispiel bei TCP, UDP und ICMP Erweiterungen getan wird).

Anders formuliert: Das erste Fragment kann wie jedes andere Paket gefiltert werden, bei dem zweiten und alle weiteren Fragmenten greifen die bisher bekannten Regeln nicht. So wird beispielsweise `-p TCP --sport www` (die einen Quellport von www filtert) ausschließlich auf das erste Fragment zutreffen.

Die Lösung des Problems besteht im Hinzufügen einer Regel für Fragmente mittels `-f` (`--fragment`) Option.

Normalerweise gilt es als sicher, wenn man zweite und weitere Fragmente akzeptiert und nur das erste Fragment herausgefiltert wird. Trotzdem: Werden Fragmente akzeptiert, dann können durch das Ausnutzen von Software-Bugs Rechner zum Absturz gebracht werden. (DOS Attacke)

Anmerkung:
Missgeformte Pakete (die z.B. zu klein sind, um Ports oder ICMP Code oder Type zu lesen) werden verworfen. TCP-Fragmente starten also bei Position 8.

Die folgende Regel verwirft jegliche Fragmente, die an 192.168.1.1 gehen:

```
root@linux / # iptables -A OUTPUT -f -d 192.168.1.1 -j DROP
```

7.3 iptables MATCH-Erweiterungen

7.3.1 Grundlegendes zu Erweiterungen

Sowohl der `iptables` Kernel-Code als auch das iptables Tool können erweitert werden. Viele dieser Erweiterungen sind für eine sichere Firewall unumgänglich. Jeder kann Erweiterungen entwickeln und diese anbieten.

Kernelerweiterungen residieren normalerweise im Unterverzeichnis für Kernelmodule, z.B. /lib/modules/2.4.24/net und werden bei Bedarf geladen. Wenn im Kernel die Option `CONFIG_KMOD` gesetzt ist, dann erfolgt das Laden vollautomatisch.

Erweiterungen für das Tool `iptables` sind Shared Libraries, die gewöhnlich unter /usr/local/lib/iptables , unter

/lib/iptables oder bei manchen Distributionen auch unter /usr/lib/iptables zu finden sind.

Wir betrachten hier erst einmal MATCH-Erweiterungen, es geht also um eine Erweiterung der Filterbedingungen. Implizite Erweiterungen (tcp, udp, icmp) werden mit der Option `-p` geladen, explizite Erweiterungen mit der Option `-m`.

Die Hilfe für eine Erweiterung erhält man, wenn nach der Option zum Laden der Erweiterung (`-p` oder `-m`) ein `-h` oder `--help` angegeben wird.

Zum Beispiel:

```
root@linux / # iptables -p udp --help
```

7.3.2 TCP Erweiterungen

Manchmal sollen bestimmte Dienste innerhalb des Netzes von außen (bzw. anderen Netzbereichen) nicht erreichbar sein. Viele Dienste benutzen das verbindungsorientierte TCP-Protokoll. Der Aufbau dieser TCP-Verbindungen erfolgt nach dem Three-Way-Handshake Protokoll, hier ist nochmals eine Kurzzusammenfassung:

Der Server wurde mit **S** bezeichnet, der Client mit **C**.

1. (C) --> [SYN] --> (S)
   Der Client sendet ein Synchronisationspaket (Synchronize) und erklärt damit, dass er eine Verbindung aufbauen möchte.
2. (C) <-- [SYN/ACK] <--(S)
   Der Server akzeptiert den Verbindungswunsch und sendet ein SYN/ACK Paket (Synchronize/Acknowledgement) zurück.
3. (C) --> [ACK] --> (S)
   Der Client wiederum bestätigt, dass die Verbindung nun aufgebaut ist/wird mit einem ACK-Paket (Empfangsbestätigung)

[Ein SYN-Paket ist ein TCP-Paket, bei dem ausschließlich das SYN-Flag gesetzt ist.]

Alle weiteren TCP-Pakete dieser Verbindung haben übrigens auch ein gesetztes ACK-Flag.

Wenn man nur die [SYN] Pakete eines Rechners blockt, können Verbindungsversuche von diesem unterbunden werden. Die Filterung dieser Pakete erfolgt durch die TCP Erweiterungen, die mit `-p tcp` geladen werden. Inversionen (Verneinungen) sind durch ein `!` hinter der Option möglich.

Es gibt folgende Optionen:

`--tcp-flags`

gefolgt von zwei Zeichenketten. Die erste Zeichenkette enthält eine Liste von Flags, die untersucht werden. Die zweite Zeichenkette sind die Flags, die gesetzt sein sollen. Mögliche Flags sind:
SYN, ACK, FIN, RST, URG, PSH bzw. **ALL** für alle und **NONE** für keine Flags.

`--syn`

entspricht `--tcp-flags SYN,RST,ACK SYN`.

`--source-port` oder `--sport`

Gefolgt von einem bzw. mehreren TCP-Ports. Ports können durch Namen (/etc/services) oder Portnummer angegeben werden. Auch ganze Bereiche sind möglich. (z.B. 10000-11000)

`--destination-port` oder `--dport`

bedeutet Zielport

`--tcp-option`

Gefolgt von einer Nummer, welche die TCP-Option angibt. TCP-Pakete ohne kompletten Header werden bei dem Versuch, die TCP-Option zu bestimmen, automatisch verworfen.

Beispiel:

```
root@linux / # iptables -A INPUT -p TCP -s 192.168.1.0/24 --tcp-flags
SYN,RST,ACK SYN -j DROP
```

7.3.3 UDP Erweiterungen

Diese Erweiterungen werden mit `-p udp` automatisch geladen. Sie bieten die Optionen `--source-port` (`--sport`) und `--destination-port` (`--dport`), wie sie bereits oben für TCP detailliert beschrieben wurden.

7.3.4 ICMP Erweiterungen

Diese Erweiterungen werden mit `-p icmp` geladen und bieten nur eine neue Option:

`--icmp-type`

Gefolgt vom ICMP-Typnamen (zum Beispiel 'host-unreachable'), oder einer entsprechenden Nummer. Eine Liste verfügbarer ICMP Typnamen erhält man mit der Option "-p icmp --help".

7.3.5 Explizite Erweiterungen

Die expliziten Erweiterungen im netfilter-Paket sind keine Standarderweiterungen. Man muss dazu spezielle Kernelmodule (z.B. `ipt_mac`, `ipt_limit`, `ipt_owner`) laden, eine Auflistung der Module befindet sich im Anhang.

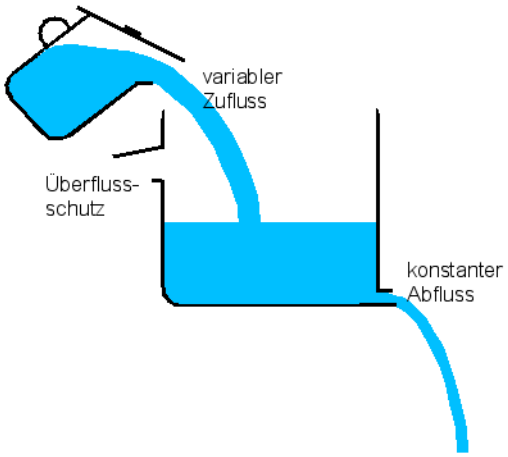
Die Erweiterung ruft man mit der Option `-m` auf.

`mac` (`-m mac` oder "`--match-mac`")

Dieses Modul wird verwendet, um die MAC-Adressen einkommender Pakete zu filtern. Hinter der Option `--mac-source` wird eine Netzwerkadresse in durch Doppelpunkte getrennter Hex-Notation angegeben. (z.B. `-m mac --mac-source 00:60:08:91:CC:B7`).

`limit` (`-m limit` oder `--match-limit`)

Limit schränkt die Paketanzahl auf eine vorgegebene Rate ein. Vorstellen kann man sich das System wie ein Rückhaltebecken mit konstantem Abfluss. Damit das Speicherbecken nicht überläuft, wird im Falle eines zu großem Stromes ein anderer Weg eingeschlagen.



Unser Strom besteht nicht aus Wasser sondern aus Datenpaketen, der **normale** Weg ins Rückhaltebecken bedeutet das Erfüllen der Regelbedingung und mit dem alternativen Weg (Überlaufschutz) ist das Nichterfüllen der Regel gemeint.

Das System kann mit zwei Parametern beschrieben werden:

* Das Speichervolumen (Parameter "`--limit-burst`") beschreibt die Anzahl der Maximaltreffer.
* Die Abflussgeschwindigkeit ("`--limit`") legt die Anzahl der maximalen Treffer pro Zeiteinheit fest. Werden diese beiden optionalen Parameter nicht angegeben, dann gelten die die Standardwerte (3 Treffer/Stunde und Maximalgrenze von 5 Treffern).

Limit kann verwendet werden, um Pakete zu loggen:

```
root@linux / # iptables -A FORWARD -m limit -j LOG
```

Limit bietet Schutz vor "Syn-Flood-Attacken":

```
root@linux / # iptables -A FORWARD -p tcp --syn -m limit 1/s -j ACCEPT
```

Schutz vor "Ping of Death":

```
root@linux / # iptables -A FORWARD -p icmp --icmp-type echo-request -m
limit --limit 1/s -j ACCEPT
```

Portscanner ausschalten:

```
root@linux / # iptables -A FORWARD -p tcp --tcp-flags ALL NONE -m limit
--limit 1/h -j ACCEPT
root@linux / # iptables -A FORWARD -p tcp --tcp-flags ALL ALL -m limit
--limit 1/h -j ACCEPT
```

`owner` (`-m owner`)

Diese Erweiterung filtert Charakteristika von Paketen und bestimmt damit deren Herkunft (lokaler Prozess). Man darf es nur in der Output-Kette benutzen:

`--uid-owner userid`

Das Paket wurde von einem Prozess mit der angegebenen effektiven User ID generiert.

`--uid-owner groupid`

Das Paket wurde von einem Prozess mit der angegebenen effektiven Gruppen ID generiert.

`--pid-owner processid`

Das Paket wurde von einem Prozess mit der angegebenen effektiven Prozess ID generiert.

`--sid-owner processid`

Das Paket wurde von einem Prozess in der angegebenen session group generiert.

`state` (`-m state` oder `--state`)

Mit dieser Erweiterung können durch die Option `--state` Verbindungszustände von Paketen gefiltert werden. Zustände sind:

NEW

Ein Paket, das eine neue Verbindung aufbaut, erfüllt diese Regelbedingung.

ESTABLISHED

Trifft für Pakete zu, die zu einer bereits aufgebauten Verbindung gehören

RELATED

Trifft für verwandte Pakete zu. Verwandt sind z.B. ICMP Fehler einer Verbindung, oder (mit dem eingefügten FTP-Modul) ein Paket, das eine FTP Datenverbindung aufbaut.

INVALID

Ein Paket, das nicht identifiziert werden konnte. (Pakete sollten verworfen werden!)

Die Möglichkeiten, die diese Option bietet, werden noch sehr detailliert im Kapitel Verbindungsverfolgung (Connection Tracking) bespochen werden.

Beispiel:

```
root@linux / # iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT
```

`tos` (`-m`)

Die TOS-Bits (Table-Of-Service) sind Flags im IP-Header, die eine Dienstgüte beschreiben. Im Einzelnen sind

dies:

Dienstgüte	Parameterwert	numerischer Wert
Normal	Normal-Service	0x00
Minimale Kosten	Minimize-Cost	0x02
Maximale Zuverlässigkeit	Maximize-Reliability	0x04
Maximaler Durchsatz	Maximize-Throughput	0x08
Minimale Verzögerung	Minimize-Delay	0x10

* Minimale Verzögerung (minimum delay)
  Die Übertragungsdauer ist das wichtigste Gütekriterium. Verbindungen über Glasfaser sind in diesem Falle einer Übertragung per Satellit vorzuziehen.
  Beispiel: ftp, telnet, ssh
* Maximaler Durchsatz (maximum throughput)
  Der Umfang der zu übertragenden Daten ist wichtig,Verzögerungen werden in Kauf genommen.
  Beispiel: ftp-data, www
* Maximale Zuverlässigkeit (maximum reliability)
  Das Paket sollte möglichst zuverlässig übertragen werden, so dass keine Wiederholungen der Übertagung erforderlich sind.
  Beispiel: snmp, dns
* Minimale Kosten (minimum cost)
  Das Paket ist äußerst unwichtig und sollte demnach den billigsten Weg einschlagen.
  Beispiel: nntp, smtp

Diese Bits können gesetzt und ausgelesen werden, indem der numerische Wert oder der Name angegeben wird. Kombinationen sind nicht sinnvoll.

Bsp:

```
root@linux / # iptables -A FORWARD -m tos --tos 0x10 -j ACCEPT
root@linux / # iptables -A FORWARD -m tos --tos Minimize-Delay -j ACCEPT
```

`mark` (`-m mark`)

Im IP-Header ist etwas Platz zum Markieren von Paketen. Diese Markierungen können mit der Option `--mark` ausgelesen werden. Zum Setzen der Markierung kann die PREROUTING Kette in der mangle-Tabelle dienen

Bsp:

```
root@linux / # iptables -t filter -A FORWARD -m mark --mark 5 -j Drop
```

`unclean`

Dieses Modul ist bislang zu wenig getestet und sollte nicht verwendet werden.

multiport, nat, ...

werden an dieser Stelle nicht betrachtet.

8 Ziele der Regeln bestimmen

8.1 Grundlegendes zu Zielen (Targets)

Wie bereits erwähnt, beinhaltet jede Kette (INPUT, OUTPUT, FORWARD) eine Checkliste von Regeln, die folgendermaßen aussehen:

WENN (Filteroption) DANN Aktion (Löschen, Akzeptieren, ... des Paketes)

```
root@linux / # iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

Neben den einfachen, eingebauten Zielen DROP (Paket Verwerfen) und ACCEPT (Paket annehmen) gibt es spezielle Ziele (RETURN, QUEUE) und Erweiterungen, die entsprechende Kernelmodule erfordern. Weiterhin hat man die Möglichkeit, zu benutzerdefinierten Ketten zu verzweigen.

8.2 Benutzerdefinierte Ketten

Ein relativ sicherer Paketfilter kann nur durch ein sehr komplexes Regelwerk realisiert werden. Um dieses Regelwerk zu strukturieren, gibt es die Möglichkeit, neben den eingebauten Ketten (INPUT, OUTPUT und FORWARD) zusätzliche benutzerdefinierte Ketten zu erstellen.

Diese Ketten sollten mit Kleinbuchstaben benannt werden, um sie besser von den eingebauten unterscheiden zu können.

Wenn ein Paket auf eine Regel (beispielsweise der INPUT Kette) mit dem Ziel einer benutzerdefinierten Kette zutrifft, wird das Filtern des Paketes in dieser Kette fortgeführt. Sollte tatsächlich keine Regel in dieser benutzerdefinierten Kette zutreffen, wird die Suche in der vorherigen Kette fortgesetzt. Benutzerdefinierte Ketten können zu anderen benutzerdefinierten Ketten verzweigen. Wenn die Pakete in einer Schleife gefangen sind, werden sie einfach verworfen (gelöscht).

Ein Beispiel befindet sich im Abschnitt Schnelle Lösung ohne viele Worte

8.3 Erweiterungen der Ziele

Eine Zielerweiterung besteht aus einem Kernelmodul und einer optionalen Erweiterung der Kommandozeilenoptionen von `iptables`.

LOG (`ipt_LOG`)

Dieses Ziel bietet Kernel-Logging. Damit die LOG-Files nicht überflutet werden, sollte es mit einem `limit` Parameter eingesetzt werden. Folgende Optionen sind möglich:

`--log-level`

Gefolgt von einer Level-Nummer (Priorität) oder einem Namen (`debug`, `info`, `notice`, `warning`, `err`, `crit`, `alert` und `emerg`). Im Kapitel über syslog werden diese Level erklärt.

`--log-prefix`

Gefolgt von einer Zeichenkette (max. 30 Zeichen), die zu Beginn der Logmeldung ausgeschrieben wird.

`--log-tcp-sequence`

Zeichnet die TCP-Sequenznummer auf. **Achtung: Mögliches Sicherheitsrisiko, wenn Benutzer Zugriff auf Log-Files hat!**

`--log-tcp-options`

Loggt die Optionen des TCP-Headers

`--log-ip-options`

Loggt die Optionen des IP-Headers

Beispiel:

```
root@linux / # iptables -A INPUT -p icmp -m limit --limit 10/s -j LOG
--log-level debug --log-prefix "icmp-logging started"
```

`REJECT`

Dieses Ziel verwirft das Paket und schickt dem Sender eine ICMP 'port unreachable' Fehlermeldung. Einschränkungen und weitere Informationen siehe RFC 1122.

`MIRROR`

befindet sich noch in einer experimentellen Phase.

Die erweiterten Ziele SNAT, DNAT, MASQUERADE und REDIRECT werden für Network Address Translation verwendet und sind deshalb nur in der NAT-Tabelle gültig:

`SNAT`

Dieses Ziel manipuliert die Quelladresse eines Paketes (siehe Kombinieren von NAT und Paketfilter). Es ist nur in der POSTROUTING-Kette gültig.

`DNAT`

Dieses Ziel manipuliert die Zieladresse eines Paketes. (siehe Kombinieren von NAT und Paketfilter).

`MASQUERADE`

Dieses Ziel wird für die Maskierung von Verbindungen mit dynamisch zugewiesenen IPs benutzt (für Verbindungen mit statischen IPs sollte SNAT verwendet werden). Es ist nur in der POSTROUTING-Kette gültig.

`REDIRECT`

Dieses Ziel ändert die Adresse eines Paketes, so dass es an die lokale Maschine gesendet wird. Es ist nur in den PREROUTING- und OUTPUT-Ketten gültig und wird vorzugsweise für transparente Proxies eingesetzt.

Beispiel: transparenter Proxy (squid)

```
root@linux / # iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j
```

```
REDIRECT --to-port 3128
```

Die folgenden beiden Ziele dienen der Paketmanipulation und sind nur in der MANGLE-Tabelle gültig:

`TOS`

Mit der Option `--set-tos` kann das Type-of-Service-Feld geändert werden. Weitere Informationen zu diesem TOS-Feld gibt es hier.

`MARK`

Mit der Option `--set-mark` kann der Netfilter-Markierungswert gesetzt werden. Weitere Informationen gibt es hier.

Beispiel:

```
root@linux / # iptables -t mangle -A PREROUTING -p tcp -m multiport
--dport 22,25,119,110,143 -j MARK --set-mark 1
```

8.4 Spezielle eingebaute Ziele (RETURN und QUEUE)

`RETURN`

Bei benutzerdefinierten Ketten bedeutet ein RETURN die direkte Rückkehr zur aufrufenden Kette. Die Filterung durch die benutzerdefinerte Kette wird abgebrochen.

Bei eingebauten Ketten (INPUT, FORWARD, OUTPUT) wird die Abarbeitung beendet und es greift die Standard-Policy.

`QUEUE`

Hier können Regelentscheidungen durch Anwendungsprogramme (Benutzerprozesse) getroffen werden. Dazu wird ein **queue handler** benötigt (für IPv4 gibt es das `ip_queue` Modul), der das Paket an ein Anwendungsprogramm überreicht. Wenn Sie tatsächlich ein Programm entwickeln möchten, sollten Sie die `libipq` API verwenden, die mit `iptables` mitgeliefert wird. Beispielcode (z.B. redirect.c) finden Sie in der Testsuite Tools auf dem CVS Server http://cvs.netfilter.org/netfilter/.

Beispiel:

```
root@linux / # iptables -A OUTPUT -p icmp -j QUEUE
root@linux / # ping 141.24.12.2
```

9 Kombinieren von NAT und Paketfilter

Mittels NAT (Network Address Translation) können durch eine Maskierung der internen IP-Adressen private Netze (z.B. 192.168.1.0/24) mit öffentlichen gekoppelt werden.

Es gibt zwei Arten von NAT:

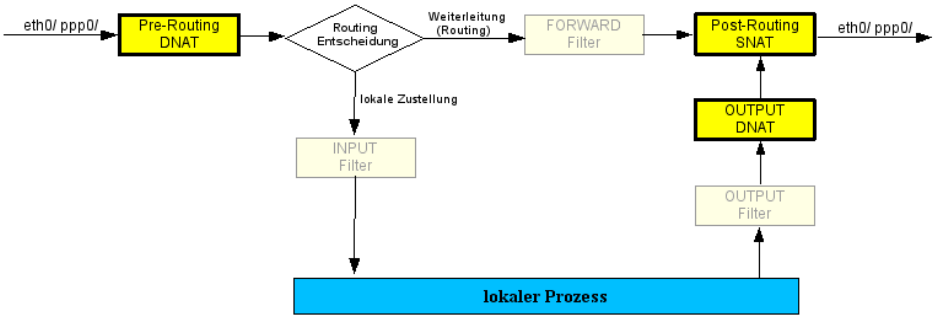
Source-NAT (SNAT):

> Hier wird die Quell-Adresse eines Paketes manipuliert. SNAT findet nach dem Routing statt. Masquerading ist ein spezieller Fall von SNAT.

Destination-NAT (DNAT):

> Hier wird die Zieladresse eines Paketes manipuliert. Es findet vor dem Routing statt. Ein Beispiel fuer DNAT ist Port-Forwarding.

NAT wird nicht in der Filtertabelle sondern in einer eigenen Tabelle verwaltet. In dieser NAT-Tabelle sind standardmäßig die Ketten POSTROUTING, PREROUTING und OUTPUT aufgeführt.



Der Zugriff auf die NAT-Tabelle erfolgt mit der Option `-t nat`, die Auswahl der Filtertabelle wie gewoht mit `-A`. Eine Auflistung der `iptables`-Optionen wurde bereits an dieser Stelle gegeben.

Beispiel:
ankommende Verbindungen an der ppp0-Schnittstelle verwerfen:

Maskiere ppp0

```
root@linux / # iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Verbiete Verbindungen von außen

```
root@linux / # iptables -A INPUT -i ppp0 -m state --state NEW,INVALID -j
DROP
root@linux / # iptables -A FORWARD -i ppp0 0 -m state --state NEW,INVALID
-j DROP
```

IP-Forwarding aktivieren

```
root@linux / # echo 1 > /proc/sys/net/ipv4/ip_forward
```

10 Verbindungsverfolgung (Connection Tracking)

10.1 Grundlegendes zur Verbindungsverfolgung

Unter Verbindungsverfolgung oder auch connection tracking versteht man das Speichern von Statusinformationen einer Verbindung (z.B. Quell- und Zieladresse, Portnummern, Protokolltyp, Timeouts,...). Bei `iptables` ist dafür die `state`-Option zuständig (siehe ▶ Explizite Erweiterungen).

Die Verbindungsverfolgung erfolgt entweder in der PREROUTING- oder in der OUTPUT- Kette. Die Pakete werden automatisch defragmentiert, Fragmente (siehe ▶ Fragmente filtern) brauchen also nicht behandelt werden.

Die Statustabellen für UDP- und TCP Verbindungen werden in /proc/net/ip_conntrack gehalten. Wie die Einträge in dieser Tabelle aussehen, wird später gezeigt. Die maximale Anzahl der Verbindungen, die diese Tabelle aufnehmen kann, wird in /proc/sys/net/ipv4/ip_conntrack_max gespeichert und je nach verfügbaren Hauptspeicher mit einem Standardwert belegt (z.B. 512 MB, ip_conntrack_max = 32760).

Die nächsten drei Unterkapiteln erläutern die Verbindungsverfolgung für die Protokolle udp, tcp und icmp, im letzten Kapitel wird das FTP-Protokoll behandelt.

10.2 UDP

UDP wird auch als verbindungsloses (zustandsloses) Protokoll bezeichnet, da im Header keine Sequenznummern oder Ähnliches zu finden sind. Das bedeutet aber nicht, dass man UDP-Verbindungen nicht aufspüren und verfolgen kann. Es gibt immer noch andere nützliche Informationen, welche das genau sind, zeigt der folgende Eintrag in der Statustabelle:

```
udp  17 21 src=192.168.1.3 dst=192.168.1.88 sport=1032 dport=53
[UNREPLIED] src=192.168.1.88 dst=192.168.1.3 sport=53 dport=1032 use=1
```

Aus diesem Eintrag kann Folgendes herausgefunden werden:

* Protocol = udp (IP Protokollnummer ist 17)
* Der Eintrag läuft nach 21 Sekunden ab, und ist danach nicht mehr gültig.
* Quell und Zieladressen mit entsprechenden Ports der Anfrage
* Quell und Zieladressen mit entsprechenden Ports der erwarteten Antwort
* Die Verbindung ist als UNREPLIED markiert, es wurde also noch nichts empfangen

Die folgenden Regel erlaubt UDP-Verbindungen vom eigenen Rechner (bei einer restriktiven Politik)

```
root@linux / # iptables -A INPUT -p udp -m state --state ESTABLISHED -j
ACCEPT
root@linux / # iptables -A OUTPUT -p udp -m state --state NEW,ESTABLISHED
-j ACCEPT
```

Udp Timeouts (Zeitlimits) werden vor dem Kompilieren in /usr/src/linux/net/ipv4/netfilter/ip_conntrack_proto_udp.c gesetzt, und zwar genau an dieser Stelle:

ip_conntrack_proto_udp.c

```
*** Datei /usr/src/linux/net/ipv4/netfilter/ip_conntrack_proto_udp.c
#define UDP_TIMEOUT (30*HZ)
#define UDP_STREAM_TIMEOUT (180*HZ)
****** Ende Datei
```

Eine einzelne TCP-Anforderung wird normalerweise 30 Sekunden in der Tabelle vorgehalten, im Falle unseres Eintrages (Beispiel oben) sind es nur noch 21 Sekunden. Das bedeutet, dass die Verbindungsanfrage bereits seit 9 Sekunden bestand ohne dass ein Antwortpaket empfangen wurde. Wenn nun ein Antwortpaket eintrifft, wird das Zeitlimit auf 30 Sekunden zurückgesetzt und die UNREPLIED Markierung gelöscht. Der Tabelleneintrag hätte nun folgendes Aussehen:

```
udp 17 28 src=192.168.1.3 dst=192.168.1.88 sport=1032 dport=53
src=192.168.1.88 dst=192.168.1.3 sport=53 dport=1032 use=1
```

Sollten mehrere Verbindungsanfragen (multiple requests) und Verbindungsantworten zwischen gleichen Socket-Paaren auftreten, so wird ein Stream festgestellt und das Zeitlimit auf 180 Sekunden erhöht. Der Tabelleneintrag hätte nun folgendes Aussehen:

```
udp  17 177 src=192.168.1.2 dst=192.168.1.50 sport=1032 dport=53
src=192.168.1.50 dst=192.168.1.2 sport=53 dport=1032 [ASSURED] use=1
```

Man erkennt die Markierung ASSURED, das bedeutet, dass die Verbindung am längsten aufrechterhalten wird. Wir erinnern uns an den Parameter `ip_conntrack_max`, der die maximale Verbindungsanzahl festlegte. Bei Erreichen dieses Limits werden die mit UNREPLIED markierten Verbindungen zuerst gelöscht und die mit ASSURED Markierten zuletzt.

Anmerkung: Es gibt weder für UDP noch für TCP Verbindungen ein absolutes Zeitlimit.

10.3 TCP

Jede TCP-Verbindung wird mittels eines Drei-Wege-Händeschüttelns (three-way handshake) aufgebaut. Kurz zur Erinnerung:
Es beginnt mit einer Synchronisationsanforderung (SYN) des Client, der Server bestätigt diese Synchronisationsanforderung (SYN+ACK) und letztendlich bestätigt der Client die aufgebaute Verbindung (ACK). Alle weiteren TCP-Pakete dieser Verbindung werden ebenfalls mit einem ACK-Flag gekennzeichnet.

Neben SYN und ACK-Flags beinhaltet der TCP-Header eine 32 bit Sequenz (Sequence Number) eine ACK-Nummer (Acknowledgment Number), sodass ein TCP-Paket eindeutig einer Verbindung zugewiesen werden kann.

Zur Verfolgung von TCP-Verbindungen dienen beispielsweise folgende Regeln:

```
root@linux / # iptables -A INPUT -p tcp -m state --state ESTABLISHED -j
ACCEPT
```

```
root@linux / # iptables -A OUTPUT -p tcp -m state --state NEW,ESTABLISHED
-j ACCEPT
```

10.3.1 Tabelleneinträge während des Verbindungsaufbaus

An dieser Stelle sollen die Tabelleneinträge während des Verbindungsaufbau analysiert werden:

1) Eine Verbindungsanforderung (SYN-Paket) wandert in die OUTPUT-Kette und wird dort akzeptiert.

```
tcp  6 119 SYN_SENT src=140.208.5.62 dst=207.46.230.218 sport=1 dport=80
[UNREPLIED] src=207.46.230.218 dst=140.208.5.62 sport=80 dport=1 use=1
```

Der Verbindungsstatus ist SYN_SENT und die Verbindung wurde als UNREPLIED markiert.

2) Nach Empfang eines SYN+ACK Paketes wird der Verbindungsstatus auf SYN_RECV gesetzt und die UNREPLIED Markierung verschwindet.

```
tcp  6 57 SYN_RECV src=140.208.5.62 dst=207.46.230.218 sport=1 dport=80
src=207.46.230.218 dst=140.208.5.62 sport=80 dport=1 use=1
```

3) Nun sollte eine Bestätigung, also ein ACK-Paket, folgen. Dieses müsste dieselbe Sequenznummer wie das ACK Paket des Servers aufweisen. Bei Übereinstimmung erhält die Verbindung eine ESTABLISHED Markierung und der Tabelleneintrag wird als ASSURED markiert. (Hinweis: Bei Erreichen des ip_conntrack_max Limits werden die mit UNREPLIED markierten Verbindungen zuerst gelöscht und die mit ASSURED Markierten zuletzt.)

```
tcp  6 431985 ESTABLISHED src=140.208.5.62 dst=207.46.230.218 sport=1
dport=80 src=207.46.230.218 dst=140.208.5.62 sport=80 dport=1 [ASSURED]
use=1
```

10.3.2 Die Statustabelle aus Sicht der Verbindungsverfolgung

Bei der Verbindungsverfolgung gibt es lediglich den Status NEW, ESTABLISHED, RELATED und INVALID (Explizite Erweiterungen).

Achtung! Dieser Status ist **nicht äquivalent** zum TCP Status. Wenn beispielsweise ein ACK-Paket zu einem nicht-existierenden Rechner hinter der Firewall versendet wird, wird ein Verbindungseintrag in der Statustabelle erzeugt, weil es als erstes Paket einer noch nicht bestehenden Verbindung (und damit als NEW) gilt. Unter dem Gesichtspunkt, dass solche ACK Pakete die Statustabelle überfluten können, ist die folgende Regel sehr sinnvoll:

```
root@linux / # iptables -A INPUT -p tcp ! --syn -m state --state NEW -j
DROP
```

Diese Regel stellt sicher, dass der Aufbau von TCP-Verbindungen ausschließlich durch SYN-Pakete initiiert werden kann. Sie hat jedoch einen kleinen Nachteil:
Manchmal treten bei TCP-Verbindungen äußerst lange Wartezeiten auf, und die korrespondierenden Einträge verschwinden dann aus der Statustabelle. Wenn aber nun die Verbindung mit ACK-Paketen (Daten) von außen fortgesetzt wird, dann verhindert die Regel den Transport nach innen. Anders ausgedrückt: Die Firewall dachte, die Verbindung wurde abgebrochen und entfernte deshalb den Eintrag in der Statustabelle. Das ankommende Datenpaket (ACK) kann keiner Verbindung zugeordnet werden und gilt deshalb als "NEW". Die obenstehende Regel blockiert aber TCP-Pakete ohne SYN-Flag, die zu keiner Verbindung zugeordnet werden können. [besser Workaround tcp-window-tracking]

10.3.3 Zeitbeschränkungen (Timeouts)

Die Zeitbeschränkungen ähneln sehr denen von UDP. Wenn eine Verbindung ein Paket erhält, wird der Timeout zurückgesetzt. Die Zeitbeschränkungen sind fest eincompiliert und in der Datei ersichtlich:
/usr/src/linux/net/ipv4/netfilter/ip_conntrack_proto_tcp.c
Wichtig sind die folgenden Zeilen:

ip_conntrack_proto_tcp.c

```
[static unsigned long tcp_timeouts]
= { 30 MINS,    /*      TCP_CONNTRACK_NONE,           */
    5 DAYS,     /*      TCP_CONNTRACK_ESTABLISHED,    */
    2 MINS,     /*      TCP_CONNTRACK_SYN_SENT,       */
    60 SECS,    /*      TCP_CONNTRACK_SYN_RECV,       */
    2 MINS,     /*      TCP_CONNTRACK_FIN_WAIT,       */
    2 MINS,     /*      TCP_CONNTRACK_TIME_WAIT,      */
    10 SECS,    /*      TCP_CONNTRACK_CLOSE,          */
    60 SECS,    /*      TCP_CONNTRACK_CLOSE_WAIT,     */
    30 SECS,    /*      TCP_CONNTRACK_LAST_ACK,       */
    2 MINS,     /*      TCP_CONNTRACK_LISTEN,         */
};
```

10.3.4 Terminierung der Verbindung

Der Verbindungsabbau kann auf zwei unterschiedliche Arten erfolgen. Der natürliche Weg (FIN+ACK) wurde kurz skizziert:

```
      Verbindungspartner 1     Verbindungspartner 2
                         .........
                         .........
         FIN+ACK    --->
                         <---     ACK
```

Sollte die Statustabelle den Verbindungswert auf TIME_WAIT setzen, dann wird der Eintrag nach zwei Minuten automatisch gelöscht.

Der andere Weg des Verbindungsabbaus ist, wenn ein Verbindungspartner ein Reset-Paket sendet (RST-Flag gesetzt). Reset Pakete werden nicht bestätigt (kein resultierendes ACK-Paket). Der Verbindungsstatus in der Tabelle wird auf CLOSE geändert und läuft nach 10 Sekunden ab. Das passiert häufig bei (ausgelasteten) HTTP-Servern.

10.4 ICMP

Es gibt nur vier ICMP-Typen, deren Pakete als NEW oder ESTABLISHED markiert werden können.

1. Echo request (ping, 8) und echo reply (pong, 0).
2. Timestamp request (13) und reply (14).
3. Information request (15) und reply (16).
4. Address mask request (17) und reply (18).

Die Anforderung (request) wird in jedem fall mit NEW markiert und die Antwort (reply) mit ESTABLISHED.

Pakete anderer ICMP-Typen können nur eine **verwandtschaftliche** Beziehung zu einer Verbindung haben und werden deshalb mit RELATED markiert.

Hier ein paar Beispiele:

```
root@linux / # iptables -A OUTPUT -p icmp -m state --state
NEW,ESTABLISHED, RELATED -j ACCEPT
root@linux / # iptables -A INPUT -p icmp -m state --state ESTABLISHED,
RELATED -j ACCEPT
```

1. Ein `icmp echo request` ist NEW somit in der OUTPUT Kette erlaubt.
2. Ein `icmp echo reply` als Antwort auf einen `echo request` ist ESTABLISHED und somit in der INPUT Kette erlaubt. Ein `echo reply` würde immer in der OUTPUT Kette herausgefiltert werden, da keine korrespondierende Echo-Anforderung (`echo request`) die INPUT-Kette passieren kann.
3. Ein `icmp redirect` gilt als RELATED und passiert somit sowohl INPUT- als auch OUTPUT-Kette, vorausgesetzt es existiert eine korrespondierende TCP- oder UDP-Verbindung in der Statustabelle.

10.5 Verbindungsverfolgung und ftp

10.5.1 FTP-Freigabe

Man benötigt das `ip_conntrack_ftp` Kernelmodul.

Für den Hausgebrauch (einzelner PC am Internet) eignen sich die folgenden zwei Regeln, um FTP-Verbindungen zu akzeptieren:

```
root@linux / # iptables -A INPUT -p tcp --sport 21 -m state --state
ESTABLISHED -j ACCEPT
root@linux / # iptables -A OUTPUT -p tcp --dport 21 -m state --state
NEW,ESTABLISHED -j ACCEPT
```

(Voraussetzung: icmp RELATED wird erlaubt, siehe vorangehendes Unterkapitel)

Leider ist es damit noch nicht getan, eine FTP-Verbindung benötigt einen eigenständigen Datenkanal, welcher durch zwei unterschiedliche Arten aufgebaut werden kann. Die folgenden zwei Unterkapitel beschreiben die Vorgehensweise.

10.5.2 Aktives ftp

Der FTP-Client sendet eine Port-Nummer über den FTP-Kanal zum FTP-Server. Dieser verbindet sich von Port 20 zu diesem Port und sendet über diese neue Verbindung (ftp-data) die Daten, z.B. das Resultat eines `ls` oder eines `get`-Kommandos. Die FTP-Datenverbindung (ftp-data) wird also vom Server aufgebaut, und nicht wie die FTP-Verbindung vom Client.

Um aktives FTP zu erlauben (Der Zielport ist nicht bekannt), müsste eine generelle Regel für alle einkommenden Verbindungen von Port 20 (FTP-Server) auf hohe Port-Nummern (>1023) des Clients anlegt werden. Dies stellt jedoch eine sehr unsichere Lösung dar.

Eine besseren Ansatz besteht in der Verbindungsverfolgung. Dazu muß das Modul `ip_conntrack_ftp` eingebunden werden. Es durchsucht das PORT Kommando nach der Port-Nummer, mit der sich der Server verbinden wird. Damit kann eine Beziehung zwischen ftp und ftp-data hergestellt werden (RELATED). Die folgenden Regeln sind also vollkommen ausreichend:

```
root@linux / # iptables -A INPUT -p tcp --sport 20 -m state --state
ESTABLISHED,RELATED -j ACCEPT
root@linux / # iptables -A OUTPUT -p tcp --dport 20 -m state --state
ESTABLISHED -j ACCEPT
```

10.5.3 Passives ftp

Im Gegensatz zu aktivem ftp wird die gewünschte PORT-Nummer der Datenverbindung nicht vom Client sondern vom Server festgelegt (via PORT-Kommando). Der Client verbindet sich dann zu diesem Port auf dem Server und der Datenverkehr kann beginnen. Obwohl diese Vorgehensweise als sicherer gilt, sollte man bedenken, dass fast gar nichts mehr über die Portnummern der Verbindung bekannt ist.

Analog zu aktiven FTP kann man die Filterregeln aufstellen, allerdings wird anstelle von NEW für die OUTPUT Kette der Parameter RELATED angewendet:

```
root@linux / # iptables -A INPUT -p tcp --sport 1024: --dport 1024: -m
state --state ESTABLISHED -j ACCEPT
root@linux / # iptables -A OUTPUT -p tcp --sport 1024: --dport 1024: -m
state --state ESTABLISHED,RELATED -j ACCEPT
```

10.6 Patches

Zur erweiterten Verbindungsverfolgung (andere Protokolle) eignen sich auch folgende Patches, die unter http://www.netfilter.org/documentation/HOWTO/de/netfilter-extensions-HOWTO.html dokumentiert sind und von dort auch heruntergeladen werden können.

* amanda-conntrack-nat
* eggdrop-conntrack
* h323-conntrack-nat
* ip_conntrack-timeouts
* mms-conntrack-nat
* pptp-conntrack-nat
* quake3-conntrack
* rpc
* rsh
* talk-conntrack-nat
* tcp-window-tracking

* tftp-conntrack-nat

11 Testen der Firewall

Ein erster Test sollte mit dem Netzwerkscanner nmap erfolgen. Mit ihm kann man sowohl einzelne Rechner als auch ganze Netzwerke nach Schwachstellen untersuchen. Ein weiterer Netzwerkscanner ist Nessus, der unter der GPL steht und sehr umfangreiche Tests ermöglicht.

Man sollte auch unbedingt stets die Log-Files auswerten, Hilfe dabei bietet das Tool logwatch.

Je nach Sicherheitsbedarf ist die Firewall regelmäßig mit den jeweils aktuellen Versionen der Netzwerkscanner zu testen, denn Sicherheit ist kein Produkt, sondern ein Prozess, der niemals als abgeschlossen betrachtet werden soll!

12 Tipps

Deaktivieren Sie alle Dienste, die Sie nicht unbedingt benötigen!

Deinstallieren Sie alles, was zum Arbeiten auf der Maschine nicht ständig gebraucht wird!

Deinstallieren Sie alle Programme mit bekannten Sicherheitslücken, selbst wenn diese gebraucht werden!

Seien Sie stets über Sicherheitslücken informiert!

Spielen Sie Updates sofort nach Erscheinen ein!

Aktualisieren Sie ständig Ihr System, auch wenn keine Lücke bekannt gegeben wurde. Hersteller gehen oftmals von aktuellen Systemen aus.

Lesen Sie Artikel zu folgenden Themen:

* syslog
* tcp-wrapper
* Proxies (z.B. Squid, SOCKS)
* Einbruchserkennungssystemen (Intrusion Detection Systems) beispielsweise Tripwire und Snort
* SSL, IPsec und IPv6
* (un)sichere Passwörter
* Verschlüsselung

und kombinieren Sie diese Maßnahmen.

Ein Firewall-Rechner ist ein Firewall-Rechner und nichts als ein Firewall-Rechner!

Eine Firewall kann nur so sicher sein, wie die Sicherheitspolitik, die sie realisiert. Investieren Sie also sehr viel Zeit in die Aufstellung einer für Sie sicheren Politik!

Halten Sie sich an das Konzept:

Alles, was nicht ausdrücklich erlaubt wird, ist verboten.

und setzen Sie eine prohibitive Sicherheitspolitik ein. (Policy = DROP für alle Ketten)!

Sicherheit bedeutet Minimalismus, denn

* jeder Schnipsel Code kann einen Fehler enthalten,
* jedes Programm und jede Funktion kann auch für etwas verwendet werden, für das sie eigentlich nicht gedacht war,
* bei zuviel Komplexität schleichen sich Fehler ein, die man nur sehr schwer findet,
* die umfangreiche Funktionalität muss ständig aktualisiert werden und
* Probleme potenzieren sich anstatt einander aufzuheben. (nach Murphy's Gesetz).

Aktivieren Sie **Route Verification**. Das bedeutet, dass Pakete, die von einer unerwarteten Schnittstelle kommen, verworfen werden oder anders formuliert: Wenn ein Paket mit einer Quelladresse des internen Netzwerkes an einer externen Schnittstelle ankommt, wird es verworfen.

Für die Schnittstelle `ppp0` können Sie es folgendermaßen aktivieren:

```
root@linux / # echo 1 > /proc/sys/net/ipv4/conf/ppp0/rp_filter
```

Oder für alle Netzwerkschnittstellen:

```
# for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
#     echo 1 > $f
# done
```

Verwenden Sie Logging nur mit Limit-Option, ansonsten besteht die Gefahr, dass die Log-Files schnell sehr groß werden und **überlaufen**.

Wenn Sie Sicherheit der Performance vorziehen, sollten Sie Regeln zur Verbindungsverfolgung (**connection tracking**) aufstellen. Es ist eingewisser Overhead nicht vermeidbar, da meist alle Verbindungen verfolgt werden, aber ein besser kontrollierter Zugang lässt Sie vielleicht etwas besser schlafen.

Beispiel:

```
root@linux / # iptables -N no-conns-from-ppp0
root@linux / # iptables -A no-conns-from-ppp0 -m state --state
ESTABLISHED,RELATED -j ACCEPT
root@linux / # iptables -A no-conns-from-ppp0 -m state --state NEW -i !
ppp0 -j ACCEPT
root@linux / # iptables -A no-conns-from-ppp0 -i ppp0 -m limit -j LOG
--log-prefix "Bad packet from ppp0:"
root@linux / # iptables -A no-conns-from-ppp0 -i ! ppp0 -m limit -j LOG
--log-prefix "Bad packet not from ppp0:"
root@linux / # iptables -A no-conns-from-ppp0 -j DROP
```

```
root@linux / # iptables -A INPUT -j no-conns-from-ppp0
root@linux / # iptables -A FORWARD -j no-conns-from-ppp0
```

Nochmals: Agieren Sie sehr minimalistisch, umso weniger Sie erlauben, desto weniger Fehler können sich einschleichen und desto weniger müssen Sie auch aktualisieren.

13 Ausblick

So wie man Sicherheit ständig aktualisieren und pflegen muss, so hat auch dieses Kapitel regelmäßige Updates und Erweiterungen nötig. Geplant sind unter anderem Fallbeispiele mit Skripten zu

* Internetanbindung einer einzelne Workstation
* Internetanbindung eines LAN mit Masquerading
* Filterregeln für `ssh` und `apache`
* Logging.

Weiterhin wollen wir einen Ausblick auf IPv6 geben, und das Thema mit Kapiteln zur

* Einbruchserkennung
* Spoofing
* Netzwerkscanner,
* Verschlüsselung
* Proxies (Squid und SOCKS) und
* Zusammenspiel des TCP_WRAPPER mit dem Superserver `xinetd` erweitern. Und seien sie immer auf der Hut, denn

Es gibt keine absolute Sicherheit!

14 Anhang

14.1 Links/ Verweise

14.1.1 Allgemein:

BSI Grundschutzhandbuch

http://www.bsi.bund.de/gshb/deutsch/menue.htm

DFN-Cert: Firewalls - Klassifikation und Bewertung

http://www.cert.dfn.de/team/ue/fw/workshop/workshop.html

(Computer-)Sicherheit allgemein

http://www.ping.at/guides/sicher/

Linksammlung Hardening HowTos,

http://www.linux-sec.net/Harden/howto.gwif.html

DFN- Vortragsfolien

http://www.cert.dfn.de/events/ws/
(Bericht 2002)
http://www.cert.dfn.de/dfn/berichte/db093/

Linux-Security Quick-Start

http://www.linuxsecurity.com/docs/LDP/Security-Quickstart-HOWTO/

Linux Security HOWTO

http://www.linuxsecurity.com/docs/LDP/Security-HOWTO/

14.1.2 Sicherheitspflege

CERT /CC Vulnerability Notes Database

http://www.kb.cert.org/vuls

DFN-CERT

http://www.cert.dfn.de/infoserv/

Bugtraq-Mailingliste

http://online.securityfocus.com/archive

14.1.3 Sicherheit v. Distribution

SuSE-Linux Security-Updates

http://www.suse.com/de/security/index.html

RedHat-Linux Security-Updates

http://www.redhat.com/apps/support/errata/index.html
Securing and Optimizing Red Hat Linux
http://www.linuxsecurity.com/docs/Securing-Optimizing-v1.3/

Debian-Linux Security-Updates

http://www.debian.org/security/
Securing Debian HOWTO
http://www.linuxsecurity.com/docs/harden-doc/html/securing-debian-howto

14.1.4 weiterführende Artikel

kleine Sammlung von Online-Artikeln

http://www.linuxsecurity.com/docs/

Snort und Nmap, Linux-Magazin

http://www.linux-magazin.de/ausgabe/2000/12/SnortNmap/SnortNmap.html

Tripwire-Artikel, Linux-Magazin

http://www.linux-magazin.de/ausgabe/2001/01/tripwire/tripwire.html
http://www.linux-magazin.de/ausgabe/2001/02/tripwire/tripwire.html
http://www.linux-magazin.de/ausgabe/2001/03/Tripwire/tripwire.html

Firewall Handbuch für LINUX 2.0 und 2.2

http://www.little-idiot.de/firewall/zusammen.html

14.1.5 Dokumentationen außerhalb Selflinux

TCP-Wrapper

http://www.cert.dfn.de/infoserv/dib/dib-2002-03-tcpwrapper/

SSH

http://www.cert.dfn.de/infoserv/dib/dib-2002-01-ssh/

Distributed Denial of Service Angriffe

http://www.cert.dfn.de/infoserv/dib/dib-2000-01.html

14.2 Übersicht über die Standard-Module von IP-Tables

Modul	Beschreibung
ip_conntrack	Verbindungsverfolgung (connection tracking)
ip_conntrack_ftp	Verbindungsverfolgung für FTP
ip_conntrack_irc	Verbindungsverfolgung für IRC
ip_nat_ftp	NAT-Support für FTP
ip_queue	packet queueing (Weiterreichen an Userspace)
ipchains	ipchains Kompatibilität
ipfwadm	ipfwadm Kompatibilität
ipt_limit	Begrenzungsfilter
ipt_mac	Filter für MAC-Adressen
ipt_multiport	Filter für mehrere Ports
ipt_owner	Filter auf Paketherkunft (UID,GID,PID des lokalen Prozesses)
ipt_state	Filter für Verbindungsstatus
ipt_unclean	Filter für "komische" Pakete
ipt_tos	Filter für Type Of Service Flags
ipt_mark	Filter für MARK-Symbole (markierte Pakete)
ipt_LOG	Ziel zum Logging
ipt_MARK	Ziel für Markierung von Paketen
ipt_MASQUERADE	Ziel für Masquerading
ipt_MIRROR	Ziel zur Spiegelung, noch experimentell
ipt_REDIRECT	Ziel zum Umleiten von Paketen
ipt_REJECT	Ziel zum Zurückweisen von Paketen
ipt_TOS	Ziel für Type Of Service Flags
iptable_filter	Tabelle filter
iptable_mangle	Tabelle mangle
iptable_nat	Tabelle nat

14.3 Regeln zum Verhindern eines NMAP-Scan

```
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags ALL
FIN,URG,PSH -j LOG --log-prefix "NMAP-XMAS SCAN:" --log-tcp-options
--log-ip-options
```

```
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags ALL
NONE -j LOG --log-prefix "NMAP-NULL SCAN:" --log-tcp-options
--log-ip-options
```

```
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags
SYN,RST SYN,RST -j LOG --log-prefix "SYN/RST SCAN:" --log-tcp-options
--log-ip-options
```

```
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags
SYN,FIN SYN,FIN -j LOG --log-prefix "SYN/FIN SCAN:" --log-tcp-options
--log-ip-options
```

```
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags ALL
```

```
FIN,URG,PSH -j DROP
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags ALL
NONE -j DROP
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags
SYN,RST SYN,RST -j DROP
root@linux / # IPTABLES -t mangle -A PREROUTING -p tcp --tcp-flags
SYN,FIN SYN,FIN -j DROP
```