

Inhaltsverzeichnis

1	ipchains (Paketfilter mit Kernel 2.2)	3
1.1	Firewalls mit Kernel 2.2	3
	ipchains-Grundlagen	3
	ipchains-Beispiel (Mini-Firewall)	8

1 ipchains (Paketfilter mit Kernel 2.2)

HINWEIS: Dieser Abschnitt wurde beinahe unverändert aus der 6. Auflage des Buchs 'Linux: Installation, Konfiguration, Anwendung', Michael Kofler, Addison-Wesley-Verlag 2001 übernommen.

Der Abschnitt gibt einige Informationen zum Aufbau eines Paketfilter-Firewalls mit ipchains (Kernel 2.2). Ab der 7. Auflage werden Firewalls nur noch auf der Basis von iptables beschreiben (verfügbar ab Kernel 2.4).

Dieser Abschnitt wird allerdings *nicht* mehr weiter gewartet, und ich kann auch keine Fragen dazu beantworten.

Mehr Informationen zum Linux-Buch von Michael Kofler finden Sie hier:

<http://www.kofler.cc/linux.html>

1.1 Firewalls mit Kernel 2.2

ipchains-Grundlagen

Kernel 2.2 verwendet so genannte ipchains zur Paketfilterung. Unter Kernel 2.4 können Sie ipchains normalerweise ebenfalls verwenden, weil dieses Filtersystem aus Kompatibilitätsgründen weiterhin unterstützt wird.

Abbildung 1.1 veranschaulicht, welche Wege die Pakete innerhalb des Filtersystems gehen können. Programme, die IP-Pakete auf dem lokalen Rechner verarbeiten bzw. selbst neue IP-Pakete erzeugen (also z. B. alle Netzwerkdienste), werden dabei durch die Box *Local Process* symbolisiert.

Vorsicht

Bei vielen Firewall-Abbildungen sehen Sie links das (gefährliche) Internet, dann die Firewall und rechts das (sichere) lokale Netz. Abbildung 1.1 entspricht nicht diesem Schema! Die Pakete, die links in den Rechner kommen, stammen sowohl aus dem lokalen Netz als auch aus dem Internet. *Input* bezieht sich auf die Netzwerk-Interfaces des Firewall-Rechners, nicht auf die Logik des gesamten Netzes! Ebenso meint *Output* alle Pakete, die den Firewall-Rechner verlassen – egal, ob diese Pakete jetzt in das lokale Netz oder in das Internet fließen.

Für die Weiterleitung von Paketen – egal, ob diese von einer Netzwerkschnittstelle kommen oder von einem lokalen Programm erzeugt wurden – ist der Kernel zuständig. Dieser hat dabei in den unterschiedlichen Stufen des Filtersystems jeweils drei Alternativen:

- *Deny*: Die Weiterleitung des Pakets wird ohne Rückmeldung abgelehnt. (Das Paket wird damit gewissermaßen gelöscht. Es existiert nicht mehr weiter.) Der Sender erfährt nie, was mit seinem Paket passiert ist.
- *Reject*: Die Weiterleitung wird mit einer Rückmeldung abgelehnt. Die Folgen für das Paket sind dieselben, allerdings bekommt der Sender durch ein (anderes) ICMP-Paket die Nachricht, dass sein Paket abgelehnt wurde.
- *Accept*: Das Paket wird weitergeleitet.

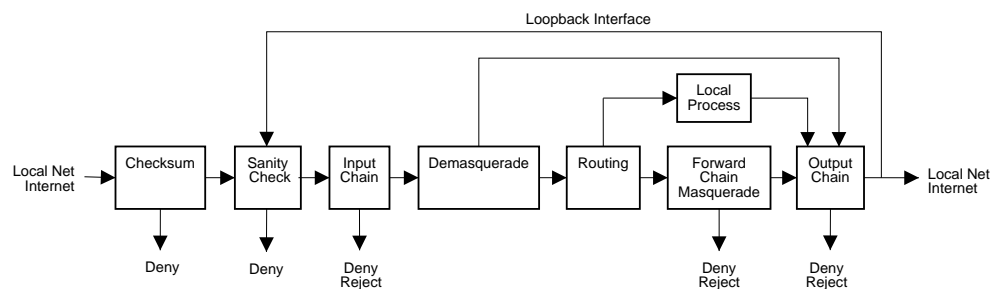


Abbildung 1.1: IP-Filterung im Kernel 2.2

Die folgende Liste beschreibt ganz kurz die Stationen eines IP-Pakets im Kernel:

- *Checksum*: Bei von außen kommenden Paketen wird die Prüfsumme des IP-Pakets kontrolliert. Ist diese falsch, wird das Paket kommentarlos gelöscht.
- *Sanity Check*: Pakete, die nicht den Regeln des Internet Protocols entsprechen, werden ebenfalls gelöscht. (Gewissermaßen ist das ein Syntax-Test.)
- *Input Chain*: Anhand einer Reihe von Regeln wird getestet, ob das Paket zur weiteren Verarbeitung durch den Kernel akzeptiert wird oder nicht.
- *Demasquerade*: Falls das Paket maskiert ist, wird es demaskiert. Maskierte Pakete gelangen nur dann in den Rechner, wenn der Rechner als Masquerading-Internet-Gateway verwendet wird (siehe Seite ??). Maskierte Pakete stellen die Antwort auf Internet-Anfragen eines Rechners Ihres lokalen Netzes dar. Der Kernel muss nun entscheiden, welcher Rechner die Anfrage initiiert hat. Die IP-Adresse des Pakets wird entsprechend geändert (das Paket wird also demaskiert). Dann wird es zur Weiterleitung an das lokale Netz an die Output Chain übergeben.
- *Routing*: Anhand der IP- und Port-Adresse entscheidet der Kernel, ob das Paket lokal bearbeitet werden soll oder ob es an einen anderen Rechner (sei es im lokalen Netz oder auch im Internet) weitergeleitet werden soll.
- *Local Process*: Diese Box symbolisiert gewöhnliche Netzwerkprogramme auf dem lokalen Rechner (z. B. Sendmail, Apache, der Telnet-Dämon etc.) sowie vom Kernel selbst realisierte Netzwerkfunktionen (etwa NFS).

- *Forward Chain*: Anhand einer Reihe von Regeln wird getestet, ob das Paket zur Weiterleitung akzeptiert wird oder nicht. An dieser Stelle wird auch das Masquerading durchgeführt. (Die *Forward Chain* gilt ausschließlich für Pakete, die durch den Rechner nur durchgeschleust werden. Sie gilt nicht für Pakete, die lokal erzeugt oder lokal verarbeitet werden.)
- *Output Chain*: Eine dritte Liste von Regeln entscheidet, ob das Paket den Kernel verlassen darf oder ob es gelöscht wird.

Hinweis

Abbildung 1.1 veranschaulicht sehr gut die Rolle des Loopback-Interface, das in Kapitel ?? erstmals erwähnt wurde. Selbst wenn der Rechner nicht an ein externes Netz angeschlossen ist, kann ein lokaler Prozess ein IP-Paket erzeugen. Dieses durchläuft den Kernel, das Loopback-Interface und wird schließlich einem anderen Prozess übergeben. Wenn Sie etwa `telnet localhost` ausführen, kommunizieren die Prozesse `telnet` (Client) und `telnetd` (Server) auf diesem Weg miteinander.

ipchains-Filter

Es gibt drei Orte, an denen Sie in die Paketweiterleitung eingreifen können: an der *Input*-, der *Forward*- und der *Output-Chain*. Das ganze Geheimnis einer Paketfilter-Firewall besteht also darin, für diese drei Kernel-Komponenten die richtigen Regeln zu definieren. Die Summe dieser Regeln wird als Filter bezeichnet. (Bei einem Rechner ohne Firewall-Konfiguration werden an diesen drei Komponenten einfach alle Pakete durchgelassen. Es gibt also keine Default-Regeln.)

Mit dem Kommando `ipchains` können Sie derartige Regeln definieren. Das Prinzip lässt sich am einfachsten anhand eines Beispiels demonstrieren. Das Beispiel geht davon aus, dass die Beispielregel auf dem Rechner `jupiter` (192.168.0.1) definiert wird. Rechner `uranus` wird verwendet, um die Regel zu testen. Beginnen Sie damit, dass Sie prüfen, ob das lokale Netz zwischen `uranus` und `jupiter` funktioniert:

```
root@uranus# ping -c 1 jupiter
PING jupiter.sol (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=1.428 ms
...
1 packets transmitted, 1 packets received, 0% packet loss
```

Nun definieren Sie auf dem Rechner `jupiter` die folgende Regel: Weise alle ICMP-Pakete zurück, die von `uranus` (also von 192.168.0.2) stammen. (ICMP wird von `ping` verwendet.)

```
root@jupiter# ipchains -A input -s 192.168.0.2 -p icmp -j DENY
```

Wiederholen Sie nun auf `uranus` den ursprünglichen `ping`-Test. Sie werden feststellen, dass `jupiter` nicht mehr antwortet (obwohl beispielsweise eine Telnet-Verbindung zu `jupiter` noch immer funktioniert).

```
root@uranus# ping -c 1 jupiter
PING jupiter.sol (192.168.0.1): 56 data bytes
...
1 packets transmitted, 0 packets received, 100% packet loss
```

Um die Regel auf `jupiter` wieder zu löschen, führen Sie noch ein `ipchains`-Kommando aus, diesmal aber mit der Option `-D` (delete) statt `-A` (add).

```
root@jupiter# ipchains -D input -s 192.168.0.2 -p icmp -j DENY
```

ipchains-Syntax

Die folgende Übersicht zählt nur die allerwichtigsten Syntaxoptionen auf! Eine vollständige Beschreibung von `ipchains` finden Sie in der sehr ausführlichen Manualseite sowie im IPChains-HOWTO.

ipchains -P input/output/forward policy

`ipchains -P` (policy) bestimmt das Defaultverhalten für eine Filterliste. Statt `policy` müssen Sie `ACCEPT`, `REJECT` oder `DENY` angeben. `ipchains -P output DENY` bedeutet also, dass keine IP-Pakete den Rechner verlassen dürfen. (Anschließend müssen Ausnahmeregeln zu diesem Defaultverhalten definiert werden.)

Grundsätzlich gibt es daher zwei Strategien beim Filterentwurf: alles verbieten und dann Ausnahmen erlauben, oder alles erlauben und Ausnahmen verbieten. Naturgemäß ist der erste Ansatz der sicherere Weg. Wenn Sie einen Sonderfall vergessen, funktioniert ein Dienst nicht, aber das ist kein Sicherheitsrisiko. Allerdings passiert es leicht, dass Sie zu viel verbieten, und die Teilnehmer im lokalen Netz werden sich bei Ihnen beschweren, was alles nicht mehr funktioniert (FTP, IRC, MP3-Streaming etc.).

ipchains -A input/output/forward optionen

`ipchains -A` (add) fügt eine neue Regel in eine der Filterlisten `input`, `output` oder `forward` ein. Generell gilt eine Regel für alle möglichen Fälle (z. B. für alle IP-Protokolle). Durch Optionen kann die Gültigkeit eingeschränkt werden. Die meisten Optionen können mit einem Ausrufezeichen auch verneint eingesetzt werden. Mit `-p udp` gilt eine Regel also beispielsweise nur für UDP-Pakete. Mit `-p !udp` gilt sie hingegen für alle Pakete außer für UDP-Pakete.

`-d ipadresse [port:port]`
gibt die Zieladresse und optional einen Port-Bereich an (destination).

`-dport port:port`
gibt den Port-Bereich für das Ziel an (ohne Adresse).

- s *ipadresse [port:port]*
gibt die Absenderadresse und deren Port-Bereich an (source).
 - sport *port:port*
gibt den Port-Bereich für den Absender an (ohne Adresse).
 - p *protocol*
bestimmt das Protokoll (z. B. `tcp`, `udp` oder `icmp`).
 - y gibt an, dass die Regel nur für solche TCP-Pakete gelten soll, bei denen das SYN-Bit gesetzt ist. Derartige Pakete werden dazu verwendet, um eine Verbindung zu initiieren (etwa für alle TCP-Wrapper-Funktionen, für HTTP etc.).
 - i *interface*
gibt an, für welches Interface die Regel gelten soll (z. B. `eth0`, `lo` für das Loopback-Interface). Bei *Input*-Regeln ist das Interface gemeint, von dem die Pakete kommen. Bei *Forward*- und *Output*-Regeln ist das Interface gemeint, zu dem die Pakete fließen. Beim Interface-Namen ist das Sonderzeichen `+` als Platzhalter für alle Interface-Nummern erlaubt, also `ppp+` für `ppp0`, `ppp1` etc.
 - j `ACCEPT/REJECT/DENY/ . .`
gibt an, was mit dem Paket geschehen soll (normalerweise `ACCEPT`, `REJECT` oder `DENY`).
- Daneben gibt es noch zwei seltener eingesetzte Varianten: `MASQ` aktiviert das Masquerading für alle von der Regel betroffenen Pakete. `REDIRECT port` leitet alle Pakete auf einen anderen Port um.
- Eine weitere Möglichkeit besteht darin, dass Sie statt eines der fünf Schlüsselwörter eine selbst definierte Filterkette (siehe unten) angeben. In diesem Fall werden alle Regeln dieser Kette angewandt. (In der prozeduralen Programmierung würde das einem Unterprogrammaufruf entsprechen.)
- l gibt an, dass Informationen über alle Datenpakete, die der Regel entsprechen, protokolliert werden (üblicherweise in `/var/log/messages`). Vorsicht, damit können Sie in kurzer Zeit sehr viele Protokolleinträge erzeugen (und gleichzeitig Ihr Netz massiv ausbremsen)!

ipchains -N userchain

`ipchains -N` (new) definiert eine neue, eigene Filterkette. Die Filterkette kann wie eine der drei vordefinierten Filterketten aufgebaut werden, d. h. Sie stellen mit `-P` das Defaultverhalten ein und definieren dann mit `-A` Regeln. Sie können Ihre eigene Filterkette dann in Regeln für die drei Standardfilterketten *Input*, *Output* oder *Forward* einsetzen (Option `-j`, siehe oben).

ipchains -L input/output/forward/userchain

`ipchains -L` (list) zeigt die Liste aller definierten Regeln an.

```
ipchains -D input/output/forward/userchain opts
```

`ipchains -D` (delete) löscht die angegebene Regel. Es müssen exakt dieselben Optionen wie bei der Definition durch `-A` angegeben werden.

```
ipchains -F input/output/forward/userchain
```

`ipchains -F` (flush) löscht alle Regeln aus der angegebenen Filterkette. Wird keine Filterkette angegeben, werden alle Regeln aller Filterketten gelöscht.

```
ipchains -X userchain
```

`ipchains -X` löscht die gesamte selbst definierte Filterkette. Wird keine Filterkette angegeben, werden alle selbst definierten Filter gelöscht.

Vorsicht

Die Reihenfolge der Regeln ist relevant! Die Regeln werden in der Reihenfolge abgearbeitet, in der sie definiert wurden. Sobald eine Regel gefunden wird, die zutrifft (ganz egal ob die Konsequenz nun *Accept*, *Reject* oder *Deny* heißt), wird diese Regel ausgeführt. Alle weiteren Regeln kommen nicht mehr zum Zuge. Das bedeutet, dass Spezialregeln (Ausnahmeregeln) *vor* allgemeineren Regeln angegeben werden müssen.

Das widerspricht der menschlichen Logik, die wie folgt formuliert: Akzeptiere Pakete für die Ports 3000-4000 mit der Ausnahme von Port 3462. Wenn hier die erste Regel zuerst definiert wird, kommt die zweite nie zum Einsatz!

Die einzige Ausnahme vom Grundsatz *zuerst speziell, dann allgemein* ist das Defaultverhalten des Filters. Trifft keine einzige Filterregel zu, dann wird das Paket gemäß dem mit `-P` eingestellten Defaultverhalten behandelt.

Verweis

Ein konkretes Beispiel für einen einfachen Paketfilter folgt im nächsten Abschnitt. Weiterführende Literatur zu `ipchains` finden Sie in den HOWTOs zu den Themen Firewall und IPChains. Daneben gibt es natürlich eine Reihe guter Bücher zu diesem Thema, von denen einige im Literaturverzeichnis am Ende des Buchs genannt sind.

ipchains-Beispiel (Mini-Firewall)

Der Ausgangspunkt für dieses Beispiel ist die Netzwerktopologie aus Abbildung ?? (siehe Seite ??). Die Firewall läuft auf dem Rechner Jupiter mit der IP-Adresse 192.168.0.1. Das lokale Netzwerk im Adressbereich 192.168.0.* ist über das Interface

eth1 angeschlossen und soll geschützt werden. Der Internetzugang erfolgt über das Interface ppp0. Via Masquerading wird allen Rechnern im lokalen Netz der Internet-Zugang gewährt. Alle Regeln werden in der Shell-Script-Datei sbin/myiprules definiert.

Tip

Es ist keine gute Idee, Firewall-Regeln via Telnet auszuprobieren! Es wird Ihnen unweigerlich passieren, dass Sie während der Definition der Regeln die Netzwerkverbindung trennen – und dann können Sie natürlich nur noch lokal am Rechner arbeiten.

Variablen: Öfter vorkommende IP-Adressen, Interfaces etc. sollten in Variablen gespeichert werden. Das erleichtert spätere Änderungen und reduziert die Gefahr von Tippfehlern:

```
#!/bin/sh
# /sbin/myiprules (einfaches Firewall-Beispiel)
# Variablen
IPC=/sbin/ipchains
LAN=192.168.0.0/24 # lokales Netz
ETHLAN=eth1      # Ethernet-Device für das lokale Netz
PPPIN=ppp0       # Netzwerk-Device für den Internet-Zugang
```

Ausgangszustand herstellen: Zu Beginn muss ein klar definierter Grundzustand hergestellt werden. Dazu werden alle vorhandenen Regeln gelöscht und das Default-Verhalten für die drei Filterlisten definiert.

Mit `input DENY` ist die Firewall vorerst dicht, d. h. von außen können keine IP-Pakete in den Rechner. ('Außen' bedeutet sowohl vom lokalen Netz als auch vom Internet aus!)

Mit `forward DENY` wird zudem jeder direkte Verkehr zwischen dem lokalen Netz und dem Internet unterbunden (also IP-Pakete, die die Firewall unverändert passieren möchten). Vorerst ist diese Regel noch übervorsichtig, weil der *Input*-Filter ja ohnedies nichts durchlässt. Der *Input*-Filter muss aber natürlich im Folgenden durch eine Menge Ausnahmeregeln geöffnet werden, damit das Netzwerk überhaupt funktioniert.

`output ACCEPT` bedeutet, dass alle Pakete, die die *Input*- und *Forward*-Filter passieren durften, das Netzwerk auch wieder verlassen dürfen. Der *Output*-Filter wird bei diesem Beispiel nicht mehr verändert. Die Firewall verlässt sich also auf die Wirksamkeit der beiden anderen Regeln – und darauf, dass auf dem Firewall-Rechner selbst keine (vom Angreifer hineingeschmuggelten) böartigen Programme ausgeführt werden, die die Firewall unterlaufen. (Wenn es einmal so weit ist, hat der Angreifer ohnedies schon mehr oder weniger die Kontrolle über den Rechner.)

```
# Defaulteinstellungen
$IPC -F          # vorhandene Standardfilter löschen
$IPC -X          # selbst definierte Filter löschen
```

```
$IPC -P input    DENY    # Defaultverhalten für die Standardfilter
$IPC -P forward DENY
$IPC -P output   ACCEPT
```

Masquerading: Die Masquerading-Regel besagt, dass alle Datenpakete aus dem lokalen Netz maskiert und an das Interface PPPIN weitergeleitet werden sollen. Die Regel aktiviert automatisch auch das Demasquerading (damit auch die Antworten empfangen werden können). Das `echo`-Kommando aktiviert eine Kernel-Option, die für das Masquerading erforderlich ist. Unter Umständen müssen Sie noch zusätzliche Kernel-Module (z. B. `ip_masq_ftp`) laden, um so Masquerading-Probleme mit bestimmten Protokollen zu vermeiden. Weitere Informationen zum Masquerading finden Sie auf Seite ??.

```
# Masquerading
echo 1 > /proc/sys/net/ipv4/ip_forward
$IPC -A forward -s $LAN -i PPPIN -j MASQ
```

Input-Regeln: Im Folgenden wird der *Input*-Filter für die verschiedenen Interfaces eingerichtet. Durch `input DENY` ist ja momentan alles blockiert.

Loopback-Interface: Das Kommando aktiviert das Loopback-Interface (ohne Einschränkungen).

```
# Input-Regeln für das Loopback-Device
$IPC -A input -i lo -j ACCEPT
```

LAN-Interface: Das erste Kommando akzeptiert alle Pakete, die vom Ethernet-Device `$ETHLAN` kommen und aus dem lokalen Netz (Maske `$LAN`) stammen.

Das zweite Kommando ist spezifisch für mein System, das außer als Firewall auch als DHCP-Server dient. Die DHCP-Kommunikation erfolgt mit UDP-Paketen über die Ports 67 und 68. (Das erste Kommando blockiert auch die UDP-Pakete, weil diese anfänglich noch keine IP-Adresse aus dem LAN haben, sondern 0.0.0.0.)

```
# Input-Regeln für das LAN-Netzwerk-Device
$IPC -A input -i $ETHLAN -s $LAN -j ACCEPT
$IPC -A input -i $ETHLAN -p udp --dport 67:68 --sport 67:68 -j ACCEPT
```

PPP-Interface: Jetzt folgt eine ganze Reihe von Regeln, die angeben, welche Pakete aus dem Internet weitergeleitet werden dürfen. Die Regeln sind in drei Gruppen für die Protokolle ICMP, UDP und TCP gegliedert. Diese Regeln bestimmen, wie sicher die Firewall ist (es gäbe natürlich noch viele Optimierungsmöglichkeiten).

ICMP-Pakete werden ohne Ausnahme akzeptiert. (Wenn Sie Ihr Netzwerk vor Angriffen schützen möchten, deren einziges Ziel darin besteht, Ihren Rechner lahm zu legen, sind hier viel restriktivere Regeln erforderlich.)

```
# Input-Regeln für das Internet-Device: ICMP-Pakete
$IPC -A input -i $PPPIN -p icmp -j ACCEPT
```

UDP-Pakete für die Ports 0 bis 1023 und 2049 (NFS) werden abgelehnt, alle anderen werden akzeptiert.

```
# Input-Regeln für das Internet-Device: UDP-Pakete
$IIPC -A input -i $PPPIN -p udp --dport 0:1023 -j DENY
$IIPC -A input -i $PPPIN -p udp --dport 2049 -j DENY
$IIPC -A input -i $PPPIN -p udp -j ACCEPT
```

Etwas feinmaschiger ist der Filter für TCP-Pakete. SYN-Pakete (das sind besondere TCP-Pakete mit gesetztem SYN-Bit, die zur Initiierung von Verbindungen dienen) werden für die Ports 0 bis 1023 abgelehnt. Die einzige Ausnahme ist Port 113, auf dem der Service `identd` läuft. Dieser Service wird oft von externen Mail- und News-Servern verwendet, um die Herkunft empfangener Nachrichten zu prüfen. Es gibt unterschiedliche Meinungen darüber, ob es ein Risiko darstellt, Port 113 für diesen Zweck offen zu lassen oder nicht.

Beachten Sie bitte, dass die Ports 0 bis 1023 nicht generell für TCP-Pakete blockiert werden können – dann würde niemand im lokalen Netz ein HTML-Dokument empfangen oder via FTP ein Dokument übertragen. Blockiert werden nur die SYN-Pakete (Option `-y`).

Die weiteren Kommandos blockieren einige Ports vollständig: 139 (NetBIOS/SMB), 1433 (Microsoft SQL Server), 2049 (NFS), 3306 (MySQL), 5999-6003 (X-Displays), 7100 (X-Font-Server) und 31337 (Back Orifice). Was bis jetzt nicht blockiert ist, wird schließlich mit der letzten Regel akzeptiert:

```
# Input-Regeln für das Internet-Device: TCP-Pakete
$IIPC -A input -i $PPPIN -p tcp -y --dport 113 -j ACCEPT
$IIPC -A input -i $PPPIN -p tcp -y --dport 0:1023 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 139 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 1433 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 2049 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 3306 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 5999:6003 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 7100 -j DENY
$IIPC -A input -i $PPPIN -p tcp --dport 31337 -j DENY
$IIPC -A input -i $PPPIN -p tcp -j ACCEPT
```

PPTP: Die letzte Regel ist wieder sehr spezifisch für meine eigenen Netzwerkanforderungen. Meine Internet-Verbindung erfolgt durch ADSL via PPTP. Die Verbindung zwischen dem Firewall-Rechner und dem ANT erfolgt über das Ethernet-Device `eth0`. (Das lokale Netz ist an `eth1` angeschlossen.) Die folgende Regel akzeptiert Pakete von `eth0`, allerdings nur, wenn sie vom ANT kommen (IP-Adresse 10.0.0.138):

```
# Input-Regeln für das ADSL-Netzwerk-Device
$IIPC -A input -i eth0 -s 10.0.0.138 -j ACCEPT
```

Firewall testen

Logging: `ipchains` bietet mit der Logging-Option `-l` die Möglichkeit, die Wirksamkeit einzelner Regeln zu protokollieren. Wenn Sie als letzte Regel für eine Filterliste `ipchains -A input/output/forward -l` angeben, werden alle Pakete protokolliert, die durch das Regelwerk gelangen (und dann, je nach Defaultverhalten, akzeptiert oder abgelehnt werden).

Paketsimulation: Um zu testen, ob Ihre Regeln dem entsprechen, was Sie beabsichtigt haben, können Sie mit `ipchains -C` Pakete simulieren. (Sie müssen also nicht warten, bis Ihr Rechner wirklich angegriffen wird.)

Zur Ausführung des Kommandos müssen Sie zumindest das Interface, das Protokoll sowie Sender- und Empfängeradresse angeben. Die zwei folgenden Beispiele zeigen, dass 1.2.3.4 auf dem Rechner 192.168.0.2 keine Telnet-Verbindung (Port 23) mit einem SYN-Paket initiieren kann. TCP-Pakete ohne SYN-Bit kommen aber zum Rechner 192.168.0.2 durch (sodass dieser Rechner Telnet als Client verwenden kann):

```
root# ipchains -C input -i ppp0 -p tcp -y \  
      -s 1.2.3.4 23 -d 192.168.0.2 23  
denied  
root# ipchains -C input -i ppp0 -p tcp \  
      -s 1.2.3.4 23 -d 192.168.0.2 23  
accepted
```

Scanner: Ideal wäre, wenn Sie Ihre Firewall von außen testen könnten, also von einem Rechner aus, der nicht Teil Ihres lokalen Netzwerks ist und der über einen anderen Provider mit dem Internet verbunden ist. Es gibt eine Reihe so genannter Scanner mit klingenden Namen wie SATAN oder SAINT, um derartige Sicherheitstests durchzuführen. (Die Werkzeuge können natürlich umgekehrt auch für einen Angriff eingesetzt werden. Auf keinen Fall sollten Sie diese Werkzeuge ohne genaues Studium der Dokumentation auf Ihre eigene Firewall loslassen – und schon gar nicht auf ein fremdes Netz!)

Wenn Sie keinen Zugang zu einem externen Rechner haben, auf dem Sie gewissenmaßen spielen dürfen, können Sie sich diverser Websites bedienen, die Sicherheitstests nach der Angabe der Netzparameter automatisch durchführen (meist gegen eine kleine Gebühr). Relativ bekannt ist etwa Shields Up (www.grc.com). Dabei stellt sich natürlich immer die Frage, wem Sie so weit vertrauen können, dass Sie ihm quasi die Erlaubnis zu einem Angriff auf Ihr eigenes Netz geben. Kurzum: Auch hier ist sehr viel Vorsicht angebracht.

Filterregeln automatisch ausführen

Damit die Filterregeln bei jedem Rechnerstart im Rahmen des Init-V-Prozesses automatisch ausgeführt werden, müssen Sie das Filter-Script im `init.d`-Verzeichnis speichern und in den `rcn.d`-Verzeichnissen die entsprechenden Links einrichten. Bei

manchen Distributionen ist bereits ein Firewall-Script vorgesehen – dann ersparen Sie sich diese Arbeit.

IP-Spoofing

Beim IP-Spoofing versucht der Angreifer, Ihnen Pakete unterzujubeln, die so aussehen, als kämen sie aus dem lokalen Netz. Dazu wird die Absenderadresse im IP-Paket manipuliert. Sie zeigt nicht auf den tatsächlichen Absender, sondern enthält eine IP-Adresse aus Ihrem Netzwerk (oder auch eine andere IP-Adresse). Das Ziel kann darin bestehen, auf diese Weise Ihre Firewall-Regeln zu unterlaufen oder aber bei bestimmten Attacken die wahre Herkunft des Angreifers zu verschleiern. Naturgemäß soll beides verhindert werden!

Zum Glück ist es sehr einfach, das IP-Spoofing zu verhindern. Seit Kernel 2.2 müssen Sie dazu keine komplizierten Filterregeln, sondern lediglich den Kernel-Parameter `rp_filter` für jedes Interface verändern:

```
# /sbin/myiprules (Fortsetzung)
# IP-Spoofing verhindern
for rpf in /proc/sys/net/ipv4/conf/*/rp_filter; do
    echo 1 > $rpf
done
```