

## KAPITEL 11

# Konfiguration der Paketfilter mit ipchains



Dieses Kapitel beschreibt das Filtern von IP-Paketen mittels ipchains. ipchains wurde in den Kernel-Versionen der Serie 2.2 eingesetzt, die mit SuSE 7.0 und Debian 3.0 ausgeliefert wurde. Der Kernel 2.4 wird dagegen standardmäßig mit iptables konfiguriert, kann allerdings auch so kompiliert werden, daß er auch weiterhin ipchains unterstützt.

Wenn Sie also einen 2.2er Kernel benutzen oder einen 2.4er mit Unterstützung von ipchains, erfahren Sie hier, wie Sie die Paketfilterung konfigurieren können. Setzen Sie dagegen iptables ein, so lesen Sie bitte in Kapitel 12 ab Seite 307 weiter.

## Die Idee

Die Grundidee eines Paketfilters besteht darin, alle Paket-Header zu betrachten und anhand definierter Regeln zu entscheiden, was mit diesen geschehen soll. Abbildung 11-1 stellt dar, was unter Linux 2.2.x geschieht, wenn ein Paket eintrifft, das für einen lokalen Prozeß auf der Firewall bestimmt ist oder durch eine spezielle REDIRECT-Regel an einen lokalen Prozeß umgeleitet wird.

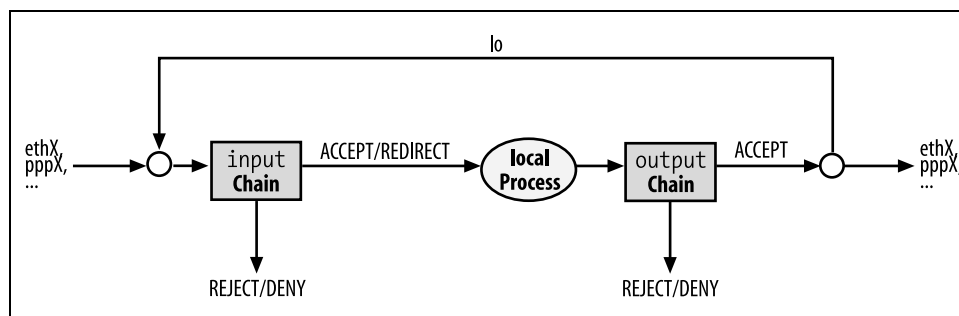


Abbildung 11-1: Das lokale Firewalling im 2.2er Kernel (nach dem IPCHAINS-HOWTO [2])

Trifft ein Paket über ein Netzwerk-Interface ein, so entscheidet das Firewalling anhand eines Satzes von Regeln (Chain) namens `input`, ob das Paket überhaupt angenommen werden soll. Ist dies der Fall, so wird es dem lokalen Prozeß zugestellt.

Pakete, die von Prozessen gesendet werden, die auf der Firewall laufen, landen direkt in der Chain `output`. Wurde schließlich die Chain `output` erfolgreich passiert, so wird das Paket über das jeweils passende Netzwerk-Interface gesendet. Pakete, die an Prozesse auf dem lokalen Rechner gerichtet sind, werden dabei über das Interface `lo` geroutet, so daß sie gleich nach dem Senden wieder über dasselbe Interface zurückkehren und nun wieder wie gehabt die Chain `input` passieren müssen, bevor sie zum Zielprozeß gelangen.

Für Pakete, die nicht für die Firewall selbst bestimmt sind, sondern von ihr nur weitervermittelt werden, werden die in Abbildung 11-2 dargestellten Stationen durchlaufen.

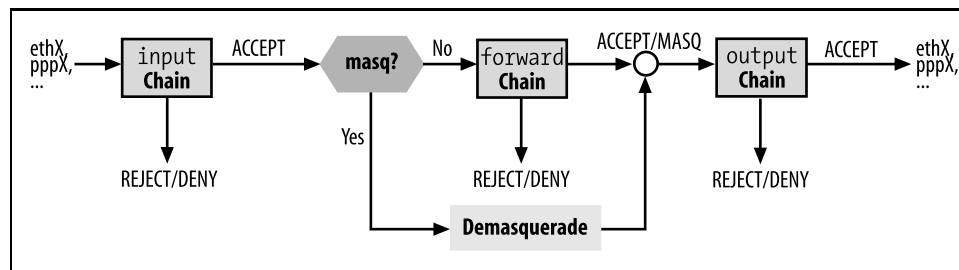


Abbildung 11-2: Das Firewalling für geroutete Pakete im 2.2er Kernel (nach dem IPCHAINS-HOWTO [2])

Auch hier entscheidet das Firewalling anhand eines Satzes von Regeln (Chain) namens `input`, ob das Paket überhaupt angenommen werden soll. Als nächstes wird untersucht, ob das Paket die Antwort auf ein Paket darstellt, dessen Absender maskiert wurde. Ist dies der Fall, wird der wahre Empfänger eingetragen und das Paket an die Chain `output` weitergeleitet.

Liegt kein Masquerading vor, wird es an die Chain `forward` weitergeleitet. Dort kann gegebenenfalls festgelegt werden, daß das Paket maskiert werden soll. Hat das Paket schließlich auch diese Chain passiert, landet es in der Chain `output`.

Auch in Kernen der Serie 2.0 funktionierte die Filterung von Paketen ähnlich. Neu in den 2.2er Kernen ist, daß es jetzt auch möglich ist, eigene Chains zu definieren, in die aus den Standard-Chains verzweigt werden kann. Damit können Regeln gruppiert werden. Hierdurch wird die Funktionalität zwar nicht erweitert, es ist aber deutlich einfacher geworden, die Wechselwirkungen zwischen verschiedenen Regeln zu kontrollieren.

Eine detailliertere Darstellung der Paketfilterung des 2.2er Kernels finden Sie im IPCHAINS-HOWTO [2] beschrieben, auch das Firewall-HOWTO [5] liefert wichtige Zusatzinformationen.

## Policy

Jeder Standard-Chain kann eine Grundregel mitgegeben werden, die Anwendung findet, wenn keine andere Regel greift. Dies nennt man eine *Policy*. Für ipchains existieren vier mögliche Policies:

**ACCEPT** Das Paket darf passieren.

**DENY** Das Paket wird verworfen.

**REJECT** Das Paket wird verworfen, und dem Empfänger wird eine ICMP-Nachricht »Destination unreachable« zugestellt.

**MASQ** Die Absenderadresse des Pakets wird so verändert, daß es vom lokalen Rechner zu kommen scheint. Diese Policy darf nur für die Chain »forward« definiert werden. Wird ein Paket maskiert, braucht keine Forwarding-Regel für den Rückweg angegeben zu werden. Wird ein Masquerading angestoßen, so bedeutet dies, daß eine Art spezieller Proxy für die Verbindung eingerichtet wird, womit die Rückpakete nur die Input- und Output-Chains durchlaufen, nicht aber die Forward-Chain.

Selbstdefinierte Chains besitzen keine Policies. Ist keine der Regeln der Chain anwendbar, so wird die nächste Regel derjenigen Chain untersucht, aus der in die selbstdefinierte Chain verzweigt wurde.

Für das Einrichten einer Firewall erscheint die Policy ACCEPT nur in Sonderfällen sinnvoll. Auch MASQ sollte nicht als generelle Regel eingerichtet werden. Damit bleiben für die drei Haupt-Chains als Policies DENY und REJECT übrig. Welche von beiden gewählt wird, ist eine Glaubensfrage. Während REJECT »sauberer« ist, da es den Sender darüber aufklärt, daß es keinen Sinn hat, weiter auf eine Antwort zu warten oder gar noch ein Paket zu schicken, ermöglicht eben diese Fehlermeldung es einem Angreifer, einen Port Scan in relativ kurzer Zeit durchzuführen. Viele Einrichter von Firewalls wählen aus diesem Grund DENY.

Die Syntax für das Erstellen einer Policy lautet:

```
ipchains -P <Chain> <Policy>
```

## Regeln

Eine Regel besteht aus den drei Teilen Muster, Aktion, Optionen. Die Muster legen fest, auf welche Pakete die Regel anzuwenden ist, die Aktion, was mit dem Paket geschieht, wenn das Muster paßt, und die Optionen regeln zusätzliche Details.

Verwaltet werden die Regeln mit den folgenden Befehlen:

```
ipchains -A chain Muster Aktion Optionen
ipchains -I chain pos Muster Aktion Optionen
ipchains -R chain pos Muster Aktion Optionen
ipchains -D chain Muster Aktion Optionen
ipchains -D chain pos
```

Dabei gilt:

- A** hängt eine Regel hinten an eine Chain an (append).
- I** fügt eine Regel an einer Position *pos* ein (insert).
- R** ersetzt die Regel an Position *pos* (replace).
- D** löscht eine Regel, die entweder über ihre Position *pos* oder ihre genaue Definition spezifiziert wurde (delete).

## Muster

Folgende Muster kennt ipchains:

- p** **[!]** **Protokoll** bezeichnet das verwendete Protokoll (TCP, UDP, ICMP oder eine numerische Angabe).
- s** **[!]** **Adresse[/Maske]** **[!]** **Port[:Port]** bezeichnet die Quelladresse (source).
- sport** **Port[:Port]** bezeichnet den Quellport (source port).
- d** **[!]** **Adresse[/Maske]** **[!]** **Port[:Port]** bezeichnet die Zieladresse (destination).
- dport** **Port[:Port]** bezeichnet den Zielport (destination port).
- icmptype** **[!]** **Typbezeichnung** bezeichnet den Typ eines ICMP-Paketes mit einem logischen Namen. Eine Liste von Typbezeichnungen, die ipchains kennt, kann mit ipchains -h icmp erfragt werden.
- i** **[!]** **interface[+]** bezeichnet das Interface, dabei ist es möglich, mit »+« alle Interfaces zu adressieren, die mit dem richtigen Namen anfangen. So würde -i eth+ sowohl auf eth0 als auch auf eth1 passen. Es ist noch zu beachten, daß für Eingangsregeln<sup>1</sup> das Interface angegeben werden muß, über das das Paket empfangen wurde, während für Weiterleitungsregeln<sup>2</sup> und Ausgangsregeln<sup>3</sup> das Interface angegeben werden muß, über welches das Paket gesendet werden wird.
- [!]** **-y** bezeichnet SYN-Pakete (SYN gesetzt, ACK, FIN gelöscht). Diese Angabe ist nur für TCP-Pakete sinnvoll.

Dabei bedeutet »!«, daß die direkt dahinter folgende Angabe negiert wird, d. h., die Regel trifft für alle Pakete zu, auf die das Teilmuster nicht paßt:

- d 192.168.0.1 !80 Pakete an 192.168.0.1, die nicht für Port 80 bestimmt sind
- d !192.168.0.1 80 Pakete für Port 80 auf irgendeiner Maschine außer 192.168.0.1
- d !192.168.0.1 !80 Pakete für alle Maschinen außer 192.168.0.1, deren Zielport nicht 80 ist
- ! -y Pakete, die keinen Verbindungsaufbau bedeuten

1 Mit Eingangsregeln sind Regeln gemeint, die in der Chain input oder einer Chain, in welche aus input verzweigt wurde, stehen.

2 Gemeint sind Regeln in der Chain forward oder in von forward aus angesprungenen selbstdefinierten Chains.

3 Gemeint sind Regeln in der Chain output oder in von output aus angesprungenen selbstdefinierten Chains.

Für ICMP gilt, daß in diesem Protokoll keine Portnummern existieren. Allerdings werden die Pakete mit Typ und Code gekennzeichnet. Der Typ gibt dabei eine Fehlerkategorie an (z. B. »Empfänger nicht erreichbar«), während der Code eine genauere Unterteilung erlaubt (z. B. »Auf dem Zielport ist kein Dienst aktiv«). Um dies in Regeln zu verwenden, wird der Typ anstelle eines Ports bei der Angabe einer Quelladresse (-s) verwendet, während der Code wie ein Port in der Zieladresse (-d) angegeben wird. Alternativ kann mit --icmp`type` eine symbolische Bezeichnung angegeben werden, die sowohl Typ als auch Code spezifiziert.

## Aktionen

Ist ein Muster spezifiziert, gilt es, eine Aktion festzulegen, die von ihm ausgelöst werden soll. Dies geschieht mit

**-j *target***

*target* kann hierbei u. a. eine der oben benannten Policies (ACCEPT . . .) oder eine benutzerdefinierte Chain sein. Darüber hinaus existieren noch folgende spezielle Targets:

**REDIRECT *Port*** Das Paket wird statt an sein Ziel an einen Port *Port* der Firewall selbst weitergeleitet. Einige Internet-Provider setzen diese Technik ein, um ihre Kunden dazu zu zwingen, für HTTP-Zugriffe einen bestimmten Proxy zu benutzen. Dies ermöglicht es ihnen, den Verkehr zum Internet zu reduzieren, da beliebte Seiten so nicht ständig aufs neue geladen werden müssen.

Damit diese Möglichkeit auch genutzt werden kann, ist es allerdings nötig, daß bei der Kernelkompilation die Einstellung »IP: transparente Proxies« getroffen wurde.

**RETURN** Das Paket »fällt aus der Chain«. D. h., es wird an die vorhergehende Chain zurückgereicht. Ist dies nicht möglich, weil es sich um die Regel einer Standard-Chain handelt, so tritt die Policy der Chain in Kraft.

## Optionen

Zusätzlich können noch die folgenden Optionen angegeben werden:

- b** Es werden zwei Regeln angelegt, zum einen die spezifizierte und zum zweiten dieselbe Regel für die umgekehrte Richtung (Quelle und Ziel vertauscht).
- f** Die Regel soll für Folgefragmente eines fragmentierten Pakets gelten. Normalerweise gelten Firewallregeln nur für das erste Fragment. Die korrekte Filterung fragmentierter Pakete ist allerdings heikel und führt leicht zu Fehlern. Aus diesem Grund sollte bei der Kompilation des Kernels »IP: IP-Pakete immer defragmentieren« angegeben werden. Dadurch werden die Pakete defragmentiert, bevor sie von den Paketfiltern bearbeitet werden.
- l** Die Regel bewirkt einen Eintrag in das Systemlog.

**-t <Andmask> <Xormask>** Diese Option erlaubt es, das Type-of-Service-Feld eines Pakets zu manipulieren. Dieses soll es Routern ermöglichen, Pakete gezielt danach zu behandeln, welche Anforderungen an den mit ihnen verbundenen Dienst gestellt werden.

Gültige Werte sind:

| Bedeutung                | Andmask | Xormask | typische Dienste |
|--------------------------|---------|---------|------------------|
| minimale Verzögerung     | 0x01    | 0x10    | ftp, telnet      |
| maximaler Durchsatz      | 0x01    | 0x08    | ftpdata          |
| maximale Verlässlichkeit | 0x01    | 0x04    | snmp             |
| minimale Kosten          | 0x01    | 0x02    | nntp             |

## Verwaltung von Chains

Eine Reihe von Befehlen dient dazu, Chains zu administrieren.

**ipchain -N Chain** Eine neue Chain anlegen (new). Der Bezeichner *Chain* darf dabei allerdings nicht länger als acht Zeichen sein.

**ipchains -F [Chain]** Alle Regeln (der Chain) löschen (flush).

**ipchains -X Chain** Eine Chain löschen. Dies gelingt nur, wenn sie keine Regeln enthält. (Der Buchstabe wurde laut [2] gewählt, weil alle sprechenden schon vergeben waren.)

**ipchains -L [Chain] [-v] [-n]** Die Regeln (der Chain) anzeigen (-n: numerisch, -v: ausführlich) (list).

**ipchains -Z** Die Paketähler aller Chains zurücksetzen (zero).

## Besondere Masquerading-Befehle

**ipchains -M -L [-n -v]** Anzeige der momentan maskierten Verbindungen (-n: numerisch, -v: ausführlich).

**ipchains -M -S  $t_{tcp}$   $t_{tcpfin}$   $t_{udp}$**  Setzen von Timeouts in Sekunden für maskierte Verbindungen. 0 bedeutet, der betreffende Wert wird nicht geändert. Dabei bezeichnen

$t_{tcp}$  TCP-Verbindungen (Vorgabe: 15 min),

$t_{tcpfin}$  TCP-Verbindungen, nachdem ein FIN-Paket empfangen wurde (Vorgabe: 2 min), und

$t_{udp}$  UDP-Verbindungen (Vorgabe: 5 min).

## Testen von Chains

Ein spezieller ipchains-Befehl dient dazu zu testen, ob eine Chain ein spezielles Paket akzeptiert:

`ipchains -C Chain Paketspezifikation`

Dabei wird das Paket prinzipiell genauso spezifiziert wie in einer Regel, allerdings müssen die Angaben `-s` (source), `-d` (destination), `-p` (protocol) und `-i` (interface) vorhanden und eindeutig sein.

## Sichern und Wiederherstellen der Firewall-Konfiguration

Im Paket `ipchains` sind neben dem Befehl `ipchains` auch die Skripte `ipchains-save` und `ipchains-restore` enthalten. Das erste dient dazu, die gegenwärtig geltenden Firewallregeln in Form von Parametern für `ipchains` auszugeben. Ein Aufruf der Form

```
# ipchains-save >config.ipf
```

sichert die gegenwärtig geltenden Regeln in einer Datei namens `config.ipf`. Man kann dem Skript auch den Namen einer Chain als Parameter mitgeben. In diesem Fall werden nur die Regeln dieser Chain gesichert.

Will man den gesicherten Zustand wiederherstellen, so geschieht dies mit dem folgenden Aufruf:

```
# ipchains-restore <config.ipf
```

Der optionale Parameter `»-p«` sorgt dafür, daß Chains, die momentan nicht existieren, von den wiederherzustellenden Regeln aber angesprungen werden, automatisch angelegt werden. Dies ist vor allem dann sinnvoll, wenn nur eine einzelne Chain gesichert wurde. Ein weiterer Parameter `»-f«` erlaubt es, festzulegen, daß wiederherzustellende Chains, die schon vorhanden sind, automatisch geleert werden. Ohne ihn wird vor dem Löschen von Regeln gefragt, ob dies gewünscht wird.

Beide Skripte kennen zusätzlich einen Parameter `»-v«`. Ist dieser angegeben, so werden die einzelnen Firewallregeln auch angezeigt, wenn wie in den oben wiedergegebenen Beispielen die Standardausgabe in eine Datei umgelenkt wird.

## Einige Beispiele

Im folgenden wollen wir einige konkrete Beispiele für Paketfilterregeln betrachten. Zusammen bilden sie die Basis für einen maskierenden Paketfilter, der leicht an die eigenen Bedürfnisse angepaßt werden kann.

Dabei ist allerdings zu beachten, daß die Reihenfolge der Filterregeln nicht willkürlich gewählt wurde. So sollten zuerst die bisher geltenden Regeln gelöscht, neue Policies definiert und Anti-Spoofing-Regeln eingerichtet werden, bevor man die eigentlichen Regeln für die zu filternden Protokolle aufstellt. Protokollregeln, die Einfluß auf andere Regeln für andere Protokolle haben (insbesondere FTP), sollten dabei als letztes definiert werden. Den Abschluß bilden schließlich Regeln für all jene Pakete, die von den bisherigen Regeln nicht erfaßt wurden.

Wir gehen davon aus, daß wir uns in einem lokalen Netz mit zwei Rechnern befinden, welche die Adressen 192.168.20.100 (Klient) und 192.168.20.15 (Firewall) haben. Unser DNS-Server ist 10.0.0.77. Die Adresse, unter der die Firewall im Internet bekannt ist, sei in der Variablen EXTIP gespeichert. Die Netzwerk-Interfaces seien eth0 für das innere Netz, während der Name des externen Interfaces in EXTIF gespeichert sei. Verwenden wir z. B. ein Modem und haben den pppd wie in Kapitel 10, Abschnitt *Einrichten von Modem oder DSL*, ab Seite 227 beschrieben konfiguriert, müssen wir die folgenden Zeilen in unser Skript eintragen:

```
EXTIP=10.1.0.1
EXTIF=ppp0
```



Vergibt unser Provider IP-Adressen dynamisch, so kann es zu Problemen kommen, wenn wir in unseren Regeln die Adresse des externen Interfaces explizit benutzen<sup>4</sup>. Regeln, die davon betroffen sind, sind hier deswegen mit einem Achtungszeichen markiert. Wir werden später sehen, wie man die daraus resultierenden Probleme umgehen kann.

Unsere Firewall werden wir so aufbauen, daß wir erst einmal alles verbieten, um dann sukzessive die Protokolle freizuschalten, die wir tatsächlich benötigen. Dabei haben wir zwei Möglichkeiten, Pakete zu verwerfen. DENY verwirft Pakete, ohne dem Absender dies mitzuteilen. REJECT schickt eine Fehlermeldung zurück, daß ein Paket nicht zugestellt werden kann.

Welche Variante man wählt, ist Geschmackssache. Beide sind prinzipiell gleich sicher. Wählt man DENY, so erreicht man, daß Port Scans deutlich länger dauern. Statt sofort eine Meldung zu bekommen, ob ein Port offen ist, muß der Scanner bis zum voreingestellten Timeout warten, bevor er entscheiden kann, daß dieser Port nicht zugreifbar ist. Solange der Angreifer nur einen einzelnen Server scannen will, kann er dieses Problem umgehen, indem er mehrere Ports parallel scannt. Will er aber statt dessen ganze Adreßbereiche untersuchen, so hilft ihm dies nicht weiter, da seine Ressourcen begrenzt sind und er nur eine gewisse Anzahl von Ports gleichzeitig untersuchen kann. Durch die Verzögerung muß er mehr Ressourcen pro Rechner aufwenden, wodurch die Anzahl der Rechner sinkt, die er in einem bestimmten Zeitraum untersuchen kann.

DENY hat aber auch Schattenseiten. Ein Rechner, der eine normale Verbindung zu einem so geschützten Port öffnen will, wird den Verbindungsaufbau in der Regel mehrfach versuchen, da er annimmt, daß das erste Paket im Netz einfach verlorengegangen ist. Erst wenn die Verbindung nach mehreren Versuchen nicht zustande kommt, wird er den Verbindungsaufbau abbrechen. Dies bedeutet im Einzelfall, daß man statt eines Pakets mehrere bekommt, die von der Firewall ausgefiltert und protokolliert werden. Grassiert z. B. gerade ein aggressiver Wurm, der sich über bestimmte Windows-Ports verbreitet, so kann sich dadurch das Systemprotokoll deutlich aufblähen.

Ein anderes Problem ist das Ident-Protokoll, das ich später noch beschreibe. Verschiedene Server machen erst eine Ident-Abfrage an den Klienten, bevor sie ihm den Zugriff

<sup>4</sup> Prinzipiell ist es sinnvoll, das Firewalling einzurichten, bevor die Netzwerk-Interfaces aktiviert werden. Allerdings kennen wir bei dynamischer Adreßvergabe die IP-Adresse des externen Interfaces erst, wenn die Verbindung zum Internet schon aufgebaut wurde.

erlauben. Benutzen wir für den dabei verwendeten Port DENY, so wird unsere Anfrage an den Server deutlich länger dauern.

Im folgenden werde ich DENY verwenden. Für Ident werden Pakete mit REJECT verworfen.

## Die absolut sichere Firewall

Bevor wir Firewallregeln aufstellen, müssen wir erst einmal einen bekannten Ausgangszustand herstellen. Hierzu löschen wir alle eventuell noch definierten Regeln und Chains. Außerdem legen wir die Policies für die Standard-Chains fest. Es empfiehlt sich, erst einmal alles zu verbieten. So erhalten wir eine »absolut sichere Firewall«, die keinerlei Pakete passieren läßt:

```
# Alle Regeln und selbstdefinierten Chains löschen
ipchains -F
ipchains -X

# Policies, die alle Pakete abweisen
ipchains -P input DENY
ipchains -P forward DENY
ipchains -P output DENY
```

Nun können wir im folgenden gezielt festlegen, für welche Protokolle wir Zugriffe durch die Firewall erlauben und für welche nicht.

## Schutz vor Spoofing

Bevor wir allerdings beginnen, protokollspezifische Regeln aufzustellen, sollten wir erst einmal sicherstellen, daß es nicht möglich ist, Pakete zu senden, deren Absenderadresse offenkundig gefälscht ist.

Die folgenden Regeln verwerfen eingehende Pakete, die über ein Netzwerk-Interface empfangen werden, das nicht an das Netz angeschlossen ist, aus dem sie laut ihrer Absenderadresse stammen:

```
# gespoofte Pakete des lokalen Netzes
ipchains -A input -i !eth0 -s 192.168.20.0/24 -j DENY -l

# gespoofte Pakete des lokalen Interfaces
ipchains -A input -i !lo -s 127.0.0.1 -j DENY -l

# gespoofte Pakete der Firewall
ipchains -A input -i !lo -s $EXTIP -j DENY -l
```



Dabei haben wir allerdings ein Problem. Wird uns unsere externe IP-Adresse bei der Einwahl ins Internet dynamisch zugewiesen, so kennen wir sie erst, wenn unsere Firewall das erste Paket in das Internet weiterleiten soll.

Tragen wir die Firewallregeln mit einem Runlevel-Skript während des Bootvorganges ein, so kennen wir unsere externe Adresse noch nicht, und wir können die dritte aufgeführte Regel nicht sinnvoll formulieren.

Wir haben drei Möglichkeiten, das Problem zu lösen:

1. Wir können sämtliche Filterregeln in `/etc/ppp/ip-up` eintragen, wo sie erst dann angewendet werden, wenn die Adresse des externen Interfaces schon bekannt ist.
2. Wir können auch nur die betroffene Regel dort eintragen und sie dann in `/etc/ppp/ip-down` selektiv löschen.
3. Wir können uns auf den Schutz durch die in Kapitel 8, Unterabschnitt *Konfiguration des /proc-Dateisystems*, ab Seite 166 besprochenen Kerneinstellungen verlassen.

Das Skript `/etc/ppp/ip-up` wird nach dem Zustandekommen der Verbindung ausgeführt. Der `pppd` wartet dabei nicht auf die Beendigung des Skriptes, bevor Pakete in das oder aus dem Internet weitergeleitet werden. Damit besteht bei der ersten Variante die Gefahr, daß das lokale Netz eine Zeitlang mit dem Internet verbunden ist, ohne daß die Firewall vollständig konfiguriert ist.

Die zweite Lösung ist dagegen durchaus praktikabel. Wir können Regeln gezielt an einer bestimmten Stelle einer Chain einfügen und sie später wieder löschen. Man sollte aber nur in Sonderfällen davon Gebrauch machen. Da wir angeben müssen, an wievielter Stelle wir eine Regel in eine Chain einfügen wollen, besteht die Gefahr, daß wir uns verzählen und so die Regel an die falsche Stelle setzen. Auch müssen wir beim Löschen der Regel sehr sorgfältig vorgehen, um nicht versehentlich die falsche Regel zu erwischen. Hier empfiehlt es sich, die zu löschende Regel mit einem Muster zu beschreiben und nicht anzugeben, an wievielter Stelle sie in der betreffenden Chain steht.

In diesem speziellen Fall spricht allerdings nichts dagegen, die betreffende Regel an erster Stelle einzutragen. Die Gefahr, sich zu verzählen, ist somit nicht gegeben. In Kapitel 11, Abschnitt *Eintragen der Regeln in die Systemdateien*, ab Seite 296 werden wir noch einmal darauf zurückkommen, wie dies konkret aussehen kann.

Die dritte Möglichkeit ist grundsätzlich genauso sicher wie die zweite. Allerdings protokollierten einige Kernelversionen die abgewiesenen Pakete nicht, so daß wir zwar gegen Angriffe geschützt sind, aber auch nicht merken, daß welche stattgefunden haben.

## Das Loopback-Interface

Über das Loopback-Interface werden Pakete geroutet, die zwischen zwei Prozessen auf demselben Rechner ausgetauscht werden. Es wäre übertrieben, hierfür spezielle Firewallregeln festzulegen. Aus diesem Grund erlauben wir diese Pakete generell:

```
# lokale Pakete
ipchains -A input -i lo -j ACCEPT
ipchains -A output -i lo -j ACCEPT
```

## NetBIOS

Wenn an Ihr lokales Netz auch Windows-Rechner angeschlossen sind, so müssen Sie damit rechnen, daß die Firewall regelmäßig mit NetBIOS angesprochen wird. Dies ist

völlig normal und kein Anzeichen eines Angriffs. Solche Pakete werden auch nicht in das Internet weitergeleitet, sondern von den Standardregeln, die wir noch definieren werden, verworfen. Allerdings werden wir die Standardregeln so definieren, daß alle verworfenen Pakete protokolliert werden, um Angriffe erkennen zu können.

Um nun zu vermeiden, das Systemprotokoll unnötig aufzublähen, bietet es sich an, gezielt Regeln zu definieren, die besagte Pakete ohne Protokollierung entsorgen:

```
# NetBIOS ueber TCP/IP
ipchains -A input -p UDP -s 192.168.20.0/24 137:139 -j DENY
ipchains -A input -p UDP -s 192.168.20.0/24 --dport 137:139 -j DENY
ipchains -A input -p TCP -s 192.168.20.0/24 137:139 -j DENY
ipchains -A input -p TCP -s 192.168.20.0/24 --dport 137:139 -j DENY
```

## ICMP

Auch für ICMP bietet sich eine spezielle Behandlung an. Das Protokoll kennt eine Vielzahl von Fehler- und Diagnosenachrichten. Von diesen sind einige sehr nützlich, während andere zu Angriffen mißbraucht werden können. Aus diesem Grund ist es genauso wenig sinnvoll, alle Nachrichten zu verwerfen, wie alle Nachrichten anzunehmen oder gar weiterzuleiten.

Betrachten wir dazu einmal einige gebräuchliche Nachrichten im einzelnen:

**Echo Reply (0)** Wird gesendet, wenn ein Echo Request (s. u.) empfangen wurde. Dies erlaubt es, mit dem Kommando ping festzustellen, ob ein Rechner netzwerktechnisch erreichbar ist. Viele Rechner im Internet filtern Echo Request und -Reply, da sie zu DoS-Angriffen genutzt werden können. Wenn man allerdings einen halbwegs aktuellen Kernel einsetzt<sup>5</sup> und darauf achtet, nicht zum Reflektor zu werden, indem man auf Pakete antwortet, die an Broadcast-Adressen gerichtet sind, dann überwiegen die sinnvollen Einsatzmöglichkeiten dieser Nachricht ihre Gefahrenpotentiale.

*Empfehlung: Nicht filtern*

**Destination Unreachable (3)** Hierbei handelt es sich um eine ganze Klasse von Fehlermeldungen, die anzeigen, daß eine Nachricht nicht zugestellt werden konnte.

Filtern wir diese Nachricht, so wird darunter die Performance der Firewall spürbar leiden. Zum einen erfahren wir nicht mehr, wenn Pakete nicht zugestellt werden können. Dies bedeutet, daß jedesmal, wenn dies geschieht, erst einmal eine ganze Weile auf eine Antwort gewartet wird. Bei manchen Anwendungen führt dies dazu, daß diese »hängen«.

Zweitens dienen diese Nachrichten auch dazu herauszufinden, wie groß Pakete maximal sein dürfen, damit sie den Empfänger auch erreichen. Dabei werden immer größere Pakete gesendet, bis eine Destination Unreachable-Nachricht mit der Information empfangen wird, daß das Paket für eines der auf dem Weg liegen-

<sup>5</sup> Frühe Linux-Kernels konnten mit einem übergroßen Echo Request-Paket zum Absturz gebracht werden. Dies nannte man »Ping of Death«.

den Teilnetze zu groß ist. Diesen Vorgang bezeichnet man als *Path MTU Discovery*.

Werden die Fehlermeldungen nun unterdrückt, so bewirkt dies, daß das Betriebssystem fälschlich einen zu großen Wert für die maximale Paketgröße (MTU<sup>6</sup>) annimmt. Dies ist kein Problem, solange die weitergeleiteten Datenpakete klein genug sind. Wird aber ein Paket weitergeleitet, das größer als die MTU ist, so wird es das Ziel nicht erreichen. Dies kann zu schwer diagnostizierbaren Fehlern führen, bei denen eine Weile keine Probleme auftreten, dann aber z. B. beim Download einer großen Datei die Verbindung abreißt. Aus diesem Grund wird allgemein davon abgeraten, diese Nachricht zu unterdrücken.

*Empfehlung: Nicht filtern*

**Source Quench (4)** Diese Fehlermeldung fordert den Rechner auf, vorübergehend das Senden einzustellen, da der Empfänger die Pakete nicht schnell genug verarbeiten kann. Sendet ein Angreifer nun gefälschte Source Quench-Pakete, so kann er die Firewall am Senden hindern.

*Empfehlung: Filtern*

**Redirect (5)** Hiermit wird der Rechner aufgefordert, einen anderen Rechner als Router zu verwenden. Dies ermöglicht nicht nur DoS-Angriffe, indem ein nicht existenter Router vorgegeben wird, es ist auch möglich, das Opfer dazu zu bringen, alle Pakete über einen Rechner des Angreifers zu routen. Damit werden auch Man-in-the-Middle-Attacks möglich, bei denen der Angreifer sämtliche Pakete, die sein Opfer sendet, nicht nur mitlesen, sondern bei Bedarf auch manipulieren kann.

*Empfehlung: Filtern*

**Echo Request (8)** Dies ist die Aufforderung, mit einem Echo Reply (s. o.) zu antworten. Für diesen Pakettyp gelten grundsätzlich dieselben Überlegungen wie für sein oben aufgeführtes Gegenstück.

*Empfehlung: Nicht filtern*

**Router Advertisement (9) Router Solicitation (10)** Ein Router Advertisement-Paket wird normalerweise als Antwort auf ein Router Solicitation-Paket gesendet und teilt einem Rechner mit, welche IP-Adresse sein zuständiger Router hat. Auf diese Weise kann ein Rechner automatisch herausfinden, wer sein zuständiger Router ist, ohne daß dies manuell konfiguriert werden muß.

Auch einem Angreifer erlaubt dies, einen Rechner dazu zu bringen, einen von ihm kontrollierten Rechner als Router zu verwenden. Dies ermöglicht es ihm, wie schon im Fall von Redirect ausgeführt, Man-in-the-Middle-Attacks durchzuführen.

Wir haben unseren Default-Router bereits manuell eingestellt. Es ist daher für uns nicht nötig, diese Pakete anzunehmen.

*Empfehlung: Filtern*

**Time Exceeded (11)** Jedes IP-Paket enthält ein Feld mit einem Zähler, der von jedem Router, der das Paket weitervermittelt, verringert wird. Erreicht der Wert 0, so wird

<sup>6</sup> Maximum Transfer Unit

das Paket verworfen und eine `Time Exceeded`-Nachricht an den Absender geschickt. Auf diese Weise soll verhindert werden, daß Pakete durch Routing-Probleme endlos im Kreis herumgereicht werden.

Eine nützliche Anwendung dieses Mechanismus ist das Kommando `tracert`<sup>7</sup>. Dieses sendet eine Reihe von Paketen, deren Zähler auf einen niedrigen Wert eingestellt ist. Begonnen wird dabei mit 1, worauf dann der Zähler bei den folgenden Versuchen jeweils um einen erhöht wird. Dies führt dazu, daß das erste Paket am ersten Router verworfen wird, während es die folgenden Pakete jeweils einen Router weiter schaffen. In der Folge erhält der Sender von allen Routern entlang des Weges Fehlermeldungen. Dies erlaubt es, nachzuvollziehen, welchen Weg Pakete zu einem bestimmten Rechner nehmen.

Üblicherweise werden hierzu entweder UDP-Pakete verwendet, die an einen hohen, aller Wahrscheinlichkeit nach nicht genutzten Port gesendet werden, oder man sendet `Echo Request`-Pakete. In ersterem Fall wird der Zielrechner mit einer `Destination Unreachable`-Fehlermeldung antworten, in letzterem mit einem `Echo Reply`-Paket. In beiden Fällen weiß der Sender, daß er das letzte Glied der Kette erreicht hat und das Senden einstellen kann.

Zwar wäre es theoretisch möglich, `Time Exceeded`-Pakete für DoS-Angriffe zu benutzen, im allgemeinen überwiegen die Vorteile aber doch die möglichen Risiken.

*Empfehlung: Nicht filtern*

**Parameter Problem (12)** Diese Fehlermeldung bedeutet, es wurde ein fehlerhaftes Paket empfangen, dessen Header für den Empfänger keinen Sinn ergeben.

*Empfehlung: Nicht filtern*

Damit bleiben `Echo Reply`, `Destination Unreachable`, `Echo Request`, `Time Exceeded` und `Parameter Problem`. Diese Pakete sollten wir entgegennehmen. Alle anderen ICMP-Pakete verwerfen wir dagegen.

ICMP-Pakete von Rechnern im lokalen Netz sollten wir nicht weiterleiten. Manche Trojaner benutzen solche Pakete, um mit dem Rechner ihres Schöpfers Kontakt aufzunehmen. Für normale Benutzer gibt es dagegen kaum einen Grund, ICMP-Pakete zu versenden.

Schließlich bleibt noch das Senden von ICMP-Paketen durch die Firewall selbst. Es spricht nichts dagegen, dies ohne weitere Einschränkung zu erlauben. Damit kommen wir zu den folgenden Regeln:

```
# ICMP
ipchains -A input -p icmp --sport 0 -j ACCEPT
ipchains -A input -p icmp --sport 3 -j ACCEPT
ipchains -A input -p icmp --sport 8 -j ACCEPT
ipchains -A input -p icmp --sport 11 -j ACCEPT
ipchains -A input -p icmp --sport 12 -j ACCEPT
ipchains -A output -p icmp -j ACCEPT
```

<sup>7</sup> Unter Windows heißt es `tracert`.

## Eigene Chains

Um im weiteren Verlauf die Regeln einfacher formulieren zu können, erstellen wir nun eigene Chains, die jeweils Regeln für Pakete festlegen, die die Firewall aus dem Internet erreichen (`ext-in`), die von der Firewall in das Internet gesendet (`ext-out`), die aus dem lokalen Netz empfangen (`int-in`), die für Rechner aus dem lokalen Netz in das Internet weitergeleitet (`int-fw`) bzw. die von der Firewall in das lokale Netz gesendet werden (`int-out`):

```
# Definition der Chains

# - externes Interface (input, output)
ipchains -N ext-in
ipchains -N ext-out

# - internes Netz (input, forward, output)
ipchains -N int-in
ipchains -N int-fw
ipchains -N int-out

# Verteilung der Pakete auf die Chains
ipchains -A input -i eth0 -j int-in
ipchains -A input -i $EXTIF -d $EXTIP -j ext-in
ipchains -A forward -i $EXTIF -s 192.168.20.0/24 -j int-fw
ipchains -A output -i eth0 -j int-out
ipchains -A output -i $EXTIF -s $EXTIP -j ext-out
```



Das Fehlen einer Weiterleitungs-Chain für eingehende Pakete ist übrigens kein Versehen, sondern geschah bewußt. Da wir im folgenden ausgehende Pakete maskieren, sind Antwortpakete aus dem Internet an die externe Adresse der Firewall selbst gerichtet. Sie werden daher in der Firewall von einem speziellen Proxy entgegengenommen und nicht geroutet. Dadurch würde für diese Pakete die `forward`-Chain nicht benutzt.

Die Verteilungsregeln sind etwas komplizierter als eigentlich nötig. Dies hat die Ursache, daß berücksichtigt wurde, daß später ein weiteres Netzwerk-Interface eingerichtet werden könnte, das mit einem Strang von Rechnern verbunden ist, die im Internet gültige Adressen besäßen. Dies wäre z. B. sinnvoll, wenn ein Webserver hinter die Firewall verlegt werden soll (Demilitarized Zone). Pakete aus oder in den neuen Strang können so nicht versehentlich nach den Regeln für maskierte lokale Rechner behandelt werden.

Wird uns vom Provider allerdings nur eine Adresse zugeteilt, welche er auch noch dynamisch vergibt, so ist dieser Aufwand unnötig. Darüber hinaus würden die Regeln für `ext-in` und `ext-out` weitere Überlegungen erforderlich machen, da hier wieder die bekannten Probleme bestünden. In so einem Fall bieten sich die folgenden Regeln an:

```
# Definition der Chains

# - externes Interface (input, output)
ipchains -N ext-in
ipchains -N ext-out
```

```
# - internes Netz (input, forward, output)
ipchains -N int-in
ipchains -N int-fw
ipchains -N int-out

# Verteilung der Pakete auf die Chains
ipchains -A input -i eth0 -j int-in
ipchains -A input -i $EXTIF -j ext-in
ipchains -A forward -i $EXTIF -s 192.168.20.0/24 -j int-fw
ipchains -A output -i eth0 -j int-out
ipchains -A output -i $EXTIF -j ext-out
```

## Blockieren des Zugriffs auf lokale Serverdienste

Betreiben wir auf der Firewall Server, die zwar aus dem lokalen Netz ansprechbar sein sollen, nicht aber aus dem Internet, so müssen wir diese mit entsprechenden Regeln schützen. Um nicht versehentlich einen Dienst zu vergessen, sollten wir zuerst einmal nachsehen, welche Dienste aktuell aktiv sind:

```
netstat -an | grep -v '^unix'
```

Betreiben wir z. B. den Proxy Squid, so bietet sich uns folgendes Bild:

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp      0      0 0.0.0.0:3128 0.0.0.0:* LISTEN
raw      0      0 0.0.0.0:1 0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags Type State I-Node Path
```

Der Eintrag mit dem Protokolltyp `raw` interessiert uns hier nicht. Er ist immer vorhanden. Der TCP-Port 3128 gehört dagegen zum Squid. Dieser sollte besser nicht aus dem Internet zugreifbar sein. Es besteht sonst die Gefahr, daß ein Angreifer ihn dazu benutzt, Anfragen nach Rechnern im lokalen Netz zu stellen. Aus Sicht der dortigen Server kämen diese dann von der Firewall.

Haben wir darüber hinaus die anonymisierenden Funktionen des Proxies eingeschaltet, so können bössartige Menschen unseren Proxy dazu verwenden, Angriffe auf Dritte anonymisiert weiterzuleiten. Diese kämen dann scheinbar von unserem Rechner. Dies hätte den Erfolg, daß die Polizei nach der Tat erst einmal an unsere Türe klopft.

Wenn Sie also einen öffentlichen Proxy betreiben wollen, sollten Sie erst einmal gründlich über die Risiken nachdenken und ein geeignetes Konzept aufstellen, diesen zu begegnen. Gegebenenfalls sollten Sie vorher Ihren Anwalt bzw. Ihre Rechtsabteilung kontaktieren und den Proxy dann auch nicht auf der Firewall selber, sondern davor in einer DMZ betreiben.

Bis dahin ist es sicherer, den Zugriff von außen einfach zu unterbinden:

```
# Zugriffe auf Server der Firewall
# - squid

ipchains -A ext-in -p tcp --dport 3128 -j DENY -1
```

## DNS

Kommen wir nun zum ersten Protokoll, daß wir nicht abblocken, sondern erlauben. Der DNS-Dienst erlaubt es, IP-Adressen sprechende Namen zuzuordnen. So wird aus 10.3.25.6 der logische Name *www.dummy.example*.

Das Protokoll verwendet Anfragen von einem hohen Port ( $\geq 1024$ ) des Klienten an Port 53 des Servers. Dabei wird wahlweise UDP oder TCP verwendet. UDP dient der Übermittlung kurzer Anfragen und Auskünfte, die in ein Paket passen. Für die Übermittlung größerer Datenmengen wird dagegen TCP verwendet.

Beginnen wir damit, daß wir Anfragen der Firewall erlauben und die Antwortpakete annehmen. Dabei erlauben wir nur Anfragen, die an den DNS-Server unseres Providers<sup>8</sup> gerichtet sind bzw. Antworten, die von dort stammen (hier: 10.0.0.77):

```
# DNS
# - Zugriff auf den externen Server

ipchains -A ext-in -p udp -s 10.0.0.77 53 --dport 1024:65535 -j ACCEPT
ipchains -A ext-out -p udp -d 10.0.0.77 53 --sport 1024:65535 -j ACCEPT

ipchains -A ext-in -p tcp -s 10.0.0.77 53 --dport 1024:65535 ! -y -j ACCEPT
ipchains -A ext-out -p tcp -d 10.0.0.77 53 --sport 1024:65535 -j ACCEPT
```

Hierbei ist noch zu beachten, daß wir bei TCP zwischen solchen Paketen unterscheiden können, die eine neue Verbindung einleiten, und solchen, die nur auf einer bestehenden Verbindung übertragen werden. Die obigen Regeln erlauben nur eingehende Pakete einer bestehenden Verbindung (! -y). Der Aufbau von Verbindungen von Port 53 TCP an einen hohen Port ist damit nicht möglich. Da UDP keine Verbindungen kennt, ist hier eine solche Unterscheidung nicht möglich.

Wir haben nun zwei Alternativen. Entweder wir leiten Anfragen der Klienten im lokalen Netz einfach weiter und maskieren nur die Absenderadresse, oder wir betreiben einen DNS-Server auf der Firewall, der Anfragen nach externen IP-Adressen wie ein Proxy an den DNS-Server unseres Providers weitergibt.

Für die erste Variante benötigen wir die folgenden Regeln:

```
# - Maskieren durch die Firewall

ipchains -A int-in -p udp --sport 1024:65535 -d 10.0.0.77 53 -j ACCEPT
ipchains -A int-fw -p udp -d 10.0.0.77 53 -j MASQ
ipchains -A int-out -p udp -s 10.0.0.77 53 -j ACCEPT

ipchains -A int-in -p tcp --sport 1024:65535 -d 10.0.0.77 53 -j ACCEPT
ipchains -A int-fw -p tcp -d 10.0.0.77 53 -j MASQ
ipchains -A int-out -p tcp -s 10.0.0.77 53 ! -y -j ACCEPT
```

Im zweiten Fall ist die Weiterleitungsregel unnötig. Außerdem müssen die Anfragen der Klienten an die Firewall und nicht an den DNS-Server des Providers gerichtet sein:

<sup>8</sup> Betreibt unser Provider mehrere DNS-Server, so bietet es sich an, für diese die gleichen Regeln jeweils mit der entsprechenden IP-Adresse einzurichten.

```
# - DNS-Server auf der Firewall

ipchains -A int-in -p UDP -d 192.168.20.15 53 -j ACCEPT
ipchains -A int-out -p UDP -s 192.168.20.15 53 -j ACCEPT

ipchains -A int-in -p TCP -d 192.168.20.15 53 -j ACCEPT
ipchains -A int-out -p TCP -s 192.168.20.15 53 ! -y -j ACCEPT
```

Wir können auch abfangen, wenn Klienten am lokalen Server vorbei auf einen DNS-Server im Internet zugreifen wollen. Solche Anfragen leiten wir dann direkt an den Server auf der Firewall um. Dabei müssen wir noch beachten, daß die Antwort scheinbar nicht von der Firewall, sondern vom eigentlichen Zielsever kommt. Wir brauchen daher auch eine Regel in int-out, die Antworten von beliebigen DNS-Servern erlaubt:

```
# - Transparente Umleitung

ipchains -A int-in -p UDP --sport 1024:65535 --dport 53 -j REDIRECT 53
ipchains -A int-out -p UDP --sport 53 --dport 1024:65535 -j ACCEPT

ipchains -A int-in -p TCP --sport 1024:65535 --dport 53 -j REDIRECT 53
ipchains -A int-out -p TCP --sport 53 --dport 1024:65535 ! -y -j ACCEPT
```

Praktisch ist so eine Umleitung vor allem dann, wenn die Rechner im lokalen Netz vorher direkt auf DNS-Server im Internet zugegriffen haben und wir uns jetzt nicht die Mühe machen wollen, überall die Adresse der Firewall als DNS-Server einzutragen. Gerade in größeren Netzen kann so eine Umleitung den Umstellungsaufwand bei der Installation der Firewall deutlich verringern.

## Ident (Auth)

Greifen wir auf einen Server im Internet zu, so kommt es vor, daß dieser erst einmal eine Verbindung zu TCP-Port 113 auf unserem Rechner öffnet. Dies dient dazu, mit Hilfe des *Authentication Service* (auch *Ident*) in Erfahrung zu bringen, welcher Benutzer die Anfrage gestellt hat.

Grundsätzlich ist dies aus unserer Sicht wenig wünschenswert. Neben Datenschutzbedenken wäre es auch technisch schwierig, auf der Firewall einen Dienst bereitzustellen, der für jede Verbindung Auskunft erteilen kann, welcher Benutzer diese initiiert hat.

Unsere Standardregeln würden das Paket einfach verwerfen. Dies hätte aber zur Folge, daß der Server keine Rückmeldung erhält und daher erst einmal eine ganze Weile auf eine Antwort wartet, bevor er unsere Anfrage abarbeitet. Um die daraus resultierende Verzögerung zu vermeiden, sollten wir statt DENY lieber REJECT verwenden. Dadurch erhält der Server eine Rückmeldung und kann sofort zum geschäftlichen Teil übergehen:

```
# Ident

ipchains -A ext-in -p TCP --dport 113 -j REJECT -l
```

## Einfache Anwendungsprotokolle über TCP

Viele Anwendungsprotokolle benutzen TCP, um eine einfache Verbindung von einem »hohen« Port (> 1023) zu einem »niedrigen« Port (< 1024) eines fremden Rechners zu öffnen.

Die dafür notwendigen Regeln haben wir schon anhand des DNS-Protokolls kennengelernt:

```
# - Zugriff auf den Server

ipchains -A ext-in -p tcp --dport 1024:65535 --sport <Port> \
! -y -j ACCEPT
ipchains -A ext-out -p tcp --sport 1024:65535 --dport <Port> \
-j ACCEPT

# - Maskieren durch die Firewall

ipchains -A int-in -p tcp --sport 1024:65535 --dport <Port> \
-j ACCEPT
ipchains -A int-fw -p tcp --sport 1024:65535 --dport <Port> \
-j MASQ
ipchains -A int-out -p tcp --dport 1024:65535 --sport <Port> \
! -y -j ACCEPT
```

Sie brauchen nur für <Port> den vom jeweiligen Protokoll verwendeten Port einzusetzen.

### HTTP

#### **Port:** 80

Das *Hypertext Transfer Protocol* (HTTP) ist die Grundlage des World Wide Web. Wann immer Sie im Browser eine Seite aufrufen, deren Adresse mit *http://* beginnt, benutzen Sie dieses Protokoll. HTTP-Server benutzen standardmäßig den TCP-Port 80.

Beginnt die Adresse allerdings mit *https://*, so benutzen Sie zwar auch HTTP, es wird aber zusätzlich das SSL- bzw. TLS-Protokoll verwendet, um die übertragenen Daten zu verschlüsseln. Hierbei wird ein anderer Serverport verwendet. Mehr dazu in Kapitel 11, Unterabschnitt *SSL und TLS*, ab Seite 289.

Grundsätzlich ist es auch möglich, Webserver auf anderen Ports zu betreiben. Dann wird in solchen Fällen häufig ein hoher Port verwendet. Eine Adresse *http://www.dummy.example:8000* bedeutet zum Beispiel, daß statt Port 80 der Port 8000 verwendet werden soll.

Erlaubt die Firewall FTP, so ist auch dies kein Problem, da dann in der Regel Zugriffe von hohen Ports auf hohe Ports erlaubt sind (siehe Kapitel 11, Unterabschnitt *FTP*, ab Seite 292).

Ist dies nicht der Fall, so müssen Sie den verwendeten Port gezielt freischalten. Da es sich hierbei normalerweise um einen Ausnahmefall handelt, sollten Sie den Zugriff nur für den jeweiligen Rechner erlauben, anstatt den jeweiligen Port generell freizugeben. Dies geschieht, indem Sie in den Regeln

```
--sport <Port> und  
--dport <Port>
```

durch

```
-s <Server> <Port> und  
-d <Server> <Port>
```

ersetzen.

## SMTP

**Port:** 25

Das *Simple Mail Transfer Protocol* (SMTP) dient dem Verschicken von E-Mails. Sie benötigen es immer dann, wenn Sie einen dedizierten E-Mail-Klienten zum Versenden Ihrer E-Mails verwenden. Benutzen Sie dagegen ein Formular auf einer Webseite (z. B. GMX, Hotmail, ...), dann betrifft Sie dieser Abschnitt nicht. In diesem Fall wird Ihre E-Mail wie jedes andere Webformular mittels HTTP zum Webserver übertragen. Erst dieser benutzt dann SMTP, um sie dem Empfänger zuzustellen.

SMTP benutzt Port 25 als Serverport.

E-Mails werden im Internet im Klartext übertragen. Sie müssen daher damit rechnen, daß ihr Inhalt bekannt wird. Grundsätzlich sollten Sie daher nur Dinge schreiben, die Ihnen nicht peinlich wären, wenn sie morgen in einer überregionalen Tageszeitung abgedruckt würden. Vertrauliche Inhalte sollten Sie unbedingt mit einer geeigneten Software verschlüsseln. Allgemein gebräuchlich sind hier PGP<sup>9</sup> oder der GNU Privacy Guard (GPG)<sup>10</sup>.

Beachten Sie dabei aber bitte, daß PGP entgegen der landläufigen Meinung nicht »frei« ist<sup>11</sup>. Die Verwendung der Software ist nur für Privatzwecke kostenlos. Der kommerzielle Einsatz erfordert den Erwerb einer Lizenz. Auch der Quelltext ist zwar verfügbar, darf aber nur dazu verwendet werden, die Software auf Sicherheitslücken und Hintertüren zu überprüfen. Er darf nicht zur Kompilation einer Version für den normalen Einsatz genutzt werden. Auch das Erstellen eigener Varianten auf Basis dieses Quelltextes ist nicht zulässig.

Anders sieht die Situation im Bezug auf den GPG aus. Hierbei handelt es sich um freie Software unter der GPL.

## POP3

**Port:** 110

Das *Post Office Protocol Version 3* (POP3) dient dazu, E-Mails von einem Mailserver abzuholen, um sie dann in Ruhe auf dem eigenen Rechner zu lesen.

---

<sup>9</sup> Erhältlich unter <http://www.pgp.com>.

<sup>10</sup> Erhältlich unter <http://www.gnupg.org>.

<sup>11</sup> Weder wie in »beer« noch wie in »speech«.

Auch dieses Protokoll wird nur von E-Mail-Klienten benutzt. Wenn Sie statt dessen ein Webinterface benutzen und Ihre E-Mails im Browser lesen, ist dieser Abschnitt für Sie nicht relevant.

POP3 benutzt Port 110 als Serverport.

Um Ihre E-Mails vom Server ausgehändigt zu bekommen, müssen Sie sich erst durch Eingabe eines Paßwortes authentisieren. Dieses wird dann im Klartext zum Server übertragen. Dies bedeutet eine Anfälligkeit des Verfahrens gegen Sniffing. Um dieser Gefahr zu begegnen, wurde zum einen mit APOP ein Verfahren entwickelt, bei dem nicht das Paßwort übertragen, sondern nur die Kenntnis des Paßwortes nachgewiesen wird. Als Alternative ist es auch möglich, die Verbindung mittels SSL/TLS zu verschlüsseln (POP3S). In diesem Fall wird allerdings ein anderer Port benutzt (siehe Kapitel 11, Unterabschnitt *SSL und TLS*, ab Seite 289).

Wenn Ihr Provider eine dieser Möglichkeiten anbietet, so sollten Sie diese unbedingt nutzen.

## IMAP

**Port:** 143

Das *Internet Message Access Protocol* (IMAP) dient einem ähnlichen Zweck wie POP3. Auch IMAP erlaubt es einem E-Mail-Klienten, auf die auf einem Mailserver gespeicherten E-Mails zuzugreifen. Während man aber bei POP3 alle E-Mails herunterlädt, um sie dann lokal zu bearbeiten und abzulegen, ist die Idee bei IMAP, die E-Mails permanent auf dem Mailserver zu belassen und sie nur zum Betrachten herunterzuladen. Legt man nun seine E-Mails in verschiedenen Mail-Ordnern ab, so bedeutet dies nicht mehr, daß der Klient verschiedene Dateien oder Verzeichnisse auf der lokalen Festplatte einrichtet, statt dessen geschieht dies auf dem Server. Die Kehrseite der Medaille ist die verlängerte Online-Zeit. Während Sie die E-Mails bearbeiten, sind Sie permanent im Kontakt mit dem Mailserver.

IMAP benutzt Port 143 als Serverport.

Wie schon POP3, so verwendet auch IMAP ein Paßwort, das im Klartext übertragen wird. Wenn Ihr Provider und der von Ihnen verwendete E-Mail-Klient dies unterstützen, dann sollten Sie die Verbindung unbedingt mittels SSL/TLS verschlüsseln (IMAPS). Hierzu wird allerdings ein anderer Port benutzt (siehe Kapitel 11, Unterabschnitt *SSL und TLS*, ab Seite 289).

## NNTP

**Port:** 119

Das *Network News Transfer Protocol* (NNTP) dient zur Kommunikation mit und unter Newsservern. Newsserver bieten eine Reihe von sogenannten Newsgroups an, elektronische Diskussionsforen zu den unterschiedlichsten Themen, in denen jeder mitreden kann. Die Nachrichten selbst, die von den Benutzern in eine Newsgroup gepostet werden, bleiben dabei nicht lokal auf dem verwendeten Newsserver, sondern werden an

andere Server weitergeleitet. So entsteht ein weltweiter Verbund von Rechnern, das sogenannte *Usenet*, in dem Anwender aus aller Herren Länder miteinander diskutieren können.

Neben Diskussionen zu weltanschaulichen Fragen oder den aktuellen Fernsehserien erweist sich das Usenet auch immer wieder als eine Quelle von Lösungen für technische Probleme. Für viele Betriebssysteme, Programmiersprachen und Anwendungen gibt es Newsgroups, in denen man Gleichgesinnte findet, die bereits dieselben Probleme gehabt haben wie man selber und die vielleicht auch eine Lösung kennen. In manchen Newsgroups lesen auch die Entwickler eines Produktes mit und sind gern bereit, bei verwickelteren Problemen zu helfen.

NNTP benutzt Port 143 als Serverport.

## Gopher

**Port:** 70

Gopher ist ein Vorläufer des World Wide Web. Es handelt sich allerdings um ein rein textbasiertes System, bei dem der Benutzer sich durch Menüs hangelt. Die einzelnen Menüpunkte führen dabei entweder zu Dateien, die dann vom Klienten angezeigt werden, oder zu neuen Menüs. Diese Menüs brauchen dabei nicht alle auf ein und demselben Rechner zu liegen. Es kann sich auch um Verweise auf einen anderen Rechner handeln, wie wir dies von Webseiten her kennen. Seit der allgemeinen Verbreitung des Web hat die Bedeutung von Gopher immer mehr abgenommen. Heutzutage wird es immer schwieriger, noch aktive Server zu finden.

Kennen Sie aber noch Gopher-Server und wollen den Zugriff durch die Firewall auf diese erlauben, so müssen Sie Zugriffe auf Port 70 erlauben. Darüber hinaus ist es möglich, Gopher-Server auf anderen Ports zu betreiben. Hier gilt das gleiche wie für HTTP (siehe Kapitel 11, Unterabschnitt *HTTP*, ab Seite 286).

## WAIS

**Port:** 210

Das *Wide Area Information Server*-Protokoll erlaubt es, auf Server zuzugreifen, die große Sammlungen von Textdateien verwalten. Hierdurch ist es möglich, komplexe Suchanfragen zu stellen, um aus dem Berg der Dokumente jene herauszufinden, die einen interessieren. WAIS ist damit ein Vorläufer der modernen Suchmaschinen.

Heutzutage wird WAIS kaum noch eingesetzt. Sie sollten Port 210 TCP daher nur freigeben, wenn ein konkreter Bedarf danach besteht.

## SSL und TLS

**Ports:** z. B. 443 (HTTPS), 995 (POP3S), 993 (IMAPS), 563 (NNTPS), 636 (LDAPS), 992 (Telnets)

Der *Secure Socket Layer* (SSL) wurde von Netscape entwickelt, um Daten transparent zu verschlüsseln, ohne daß das eigentlich verwendete Anwendungsprotokoll geändert werden muß. Dies wird möglich, indem SSL als zusätzliche Schicht zwischen dem Anwendungsprotokoll (z. B. HTTP) und TCP sitzt. Die Daten des Anwendungsprotokolls werden beim Sender verschlüsselt und dann als Nutzdaten an TCP übergeben, das sie zum Zielrechner transportiert. Dort werden sie durch die Gegenstelle entschlüsselt und an die Anwendung übergeben. So wie TCP die Anwendung davon befreit, eigene Maßnahmen zu implementieren, um dafür zu sorgen, daß alle gesendeten Pakete in der richtigen Reihenfolge beim Empfänger ankommen, so sorgt SSL dafür, daß die zu verwendenden Kryptoverfahren ausgehandelt werden, dem Gegenüber der jeweilige Schlüssel zugänglich gemacht wird und gegebenenfalls die Identität des anderen überprüft wird. SSL ist so konzipiert worden, daß grundsätzlich beliebige Kryptoverfahren verwendet werden können, um die Verschlüsselung dem eigenen Sicherheitsbedürfnis und der Geschwindigkeit der verwendeten Hardware anzupassen.

Das Protokoll wurde dann im Rahmen seiner Annahme als Internet-Standard in *Transport Layer Security* (TLS) umgetauft.

Für verschiedene Protokolle wurden eigene Portnummern für die Verwendung mit SSL reserviert, so z. B. 443 für HTTPS, 563 für NNTPS, 636 für LDAPS, 992 für Telnets, 993 für IMAPS, 995 für POP3S.

Am gebräuchlichsten ist hierbei HTTPS, das insbesondere für Bestellungen bei Internet-Händlern und für das Online-Banking benutzt wird. Dieses Protokoll sollten Sie daher auf jeden Fall zulassen. Darüber hinaus bietet es sich an, POP3S bzw. IMAPS zu erlauben, wenn diese Protokolle von dem von Ihnen verwendeten Mailserver unterstützt werden. Andere Protokolle über SSL werden derzeit noch eher selten eingesetzt.

## Proxies auf der Firewall

Wollen wir einen Proxy auf der Firewall betreiben, so sind die nötigen Regeln denen sehr ähnlich, die wir für den Betrieb ohne Proxy verwenden. Es sind allerdings einige kleine Änderungen vorzunehmen:

1. Da die Pakete nun an die Firewall selbst gesendet werden, benötigen wir keine Forwarding-Regel. Alle Regeln für int-fw entfallen somit.
2. Aus demselben Grund sollte in int-in und int-out nicht nur der Zielport, sondern auch die Zieladresse (internes Interface der Firewall) abgeprüft werden.
3. Je nach Proxy wird unter Umständen ein spezieller Port benutzt. Webproxies benutzen beispielsweise gern 3128, 8000, 8080 oder 8118.

Wir benötigen also weiterhin die Regeln, die wir für den Zugriff auf den Server definiert haben. Die Regeln im Abschnitt »Maskieren durch die Firewall« entfallen dagegen und werden durch neue ersetzt, die nur den Zugriff auf den jeweiligen Proxy auf der Firewall erlauben.

Benutzen wir z. B. den squid als HTTP-Proxy, so können wir die folgenden Regeln verwenden:

```
# HTTP
# - Zugriff auf den Server

ipchains -A ext-in -p tcp --dport 1024:65535 --sport 80 ! -y -j ACCEPT
ipchains -A ext-out -p tcp --sport 1024:65535 --dport 80 -j ACCEPT

# - Proxy auf Port 3128

ipchains -A int-in -p TCP --sport 1024:65535 -d 192.168.20.15 3128 -j ACCEPT
ipchains -A int-out -p TCP -s 192.168.20.15 3128 --dport 1024:65535 ! -y -j ACCEPT
```

Für andere Webproxies ist gegebenenfalls der Serverport (hier: 3128) anzupassen.

Benutzen wir für den Zugriff auf das Internet einen Webbrowser, so sind wir schon fast fertig. Der Browser stellt sämtliche Anfragen an die Firewall mittels HTTP. Dies gilt auch, wenn das Zielsystem z. B. ein FTP-Server ist. In diesem Fall ist es die Aufgabe des Proxys, die HTTP-Anfrage z. B. in eine FTP-Anfrage zu »übersetzen«. Alle Anfragen der Klienten im lokalen Netz haben wir daher bereits mit der oben unter »Proxy auf Port 3128« angegebenen Regel abgehandelt.

Was noch fehlt, sind Regeln, die es dem Proxy erlauben, mit dem jeweils gewünschten Protokoll auf den Zielserver zuzugreifen. Diese sind die gleichen, wie sie bereits beim normalen Betrieb ohne Proxy definiert wurden. Sie finden sie beim jeweiligen Protokoll im Abschnitt »Zugriff auf den Server«.

Für Gopher würde eine entsprechende Regel z. B. so aussehen:

```
# Gopher
# - Zugriff auf den Server

ipchains -A ext-in -p TCP --sport 70 --dport 1024:65535 ! -y -j ACCEPT
ipchains -A ext-out -p TCP --dport 70 --sport 1024:65535 -j ACCEPT
```

## Transparente Proxies

Mit `ipchains` ist es möglich, Anfragen, die an einen Server im Internet gerichtet waren, auf einen Dienst umzuleiten, der auf der Firewall selbst läuft. Diese Technik wird gerne von Internet-Providern benutzt, die möchten, daß ihre Kunden einen Proxy verwenden, die ihnen aber nicht zumuten wollen, den Browser entsprechend zu konfigurieren.

Nicht jeder Proxy ist dafür allerdings geeignet. Kein Problem stellen Proxies für Protokolle wie SMTP oder NNTP dar, bei denen im Design vorgesehen ist, daß Nachrichten von einem Server an den nächsten weitergeleitet werden, bis die Nachricht ihr Ziel erreicht (*Store and Forward-Protokolle*). Bei diesen Protokollen ist der Empfängerrechner entweder in den eigentlichen Nutzdaten des Protokolls angegeben (SMTP), oder die Nachricht ist nicht an einen speziellen Rechner gerichtet (NNTP).

Bei Protokollen wie FTP, deren Design vorsieht, daß grundsätzlich eine direkte Verbindung zum Zielrechner besteht, ist die Verwendung eines transparenten Proxys dagegen nur möglich, wenn der Proxy speziell dazu vorgesehen ist, mit dem transparenten Proxying der Firewall zusammenzuarbeiten. Ohne die Information, für welchen Zielrechner

die Verbindung eigentlich bestimmt war, bevor sie umgeleitet wurde, kann er seine Aufgabe nicht sinnvoll erfüllen.

HTTP gehörte ursprünglich zur zweiten Kategorie. Mittlerweile wurde aber ein zusätzlicher Header definiert, in dem der Zielrechner angegeben werden kann. Dieser Header wird von den gängigen Browsern auch genutzt, so daß einer Einrichtung eines transparenten Proxys für HTTP heutzutage nichts mehr im Weg steht.

Wollen wir also z. B. alle HTTP-Anfragen auf einen Squid umleiten, so benötigen wir die folgenden Regeln:

```
# Transparente Umleitung von HTTP-Zugriffen auf den Proxy auf Port 3128
ipchains -A int-in -p TCP --sport 1024:65535 --dport 80 -j REDIRECT 3128
ipchains -A int-out -p TCP --sport 80 --dport 1024:65535 ! -y -j ACCEPT
```

## FTP

Das *File Transfer Protocol* (FTP) dient der Übertragung von Dateien. Das wichtigste Einsatzgebiet ist der Download von Software.

Das Protokoll ist deutlich komplizierter als die bereits betrachteten. Es beginnt damit, daß eine Verbindung von einem hohen TCP-Port des Klienten zu Port 21 des Servers geöffnet wird. Über diese Verbindung werden dann Kommandos an den Server gesendet. Sobald nun eine Datei heruntergeladen oder der Inhalt eines Verzeichnisses angezeigt werden soll, wird eine weitere Verbindung geöffnet, über die dann die angeforderten Daten übertragen werden.

Hierbei unterscheidet man zwei Varianten:

**aktives FTP** Beim aktiven FTP öffnet der Server eine Verbindung von Port 20 auf seiner Seite zu einem hohen Port des Klienten, den dieser vorher über die Kontrollverbindung festgelegt hat.

**passives FTP** Beim passiven FTP wird die Datenverbindung vom Klienten initiiert. Beide Seiten verwenden hohe Ports, der zu benutzende Port des Servers wurde dem Klienten zuvor über die Kontrollverbindung mitgeteilt.

Die Verwendung einer zweiten Verbindung macht das Erstellen von Firewallregeln deutlich komplizierter. Beim aktiven FTP müssen wir eingehende Verbindungen von Port 20 TCP an einen beliebigen hohen Port erlauben. Dies bedeutet, daß wir eingehende Verbindungen auf hohe TCP-Ports nicht generell verbieten können. Betreiben wir auf hohen Ports Server (z. B. Webproxies), so müssen wir für jeden verwendeten Port eine eigene Regel aufstellen. Haben wir einen Port übersehen, so bleiben wir angreifbar. Dies bedeutet, wir müssen hier von unserem Grundprinzip abweichen, erst einmal alles zu verbieten, um dann gezielt die Zugriffe zu erlauben, die notwendig sind.

Ein weiteres Problem besteht darin, daß wir Verbindungen maskieren. Die Datenverbindung wird daher zur Firewall geöffnet, nicht zum eigentlichen Zielrechner. Die Firewall muß dann dafür sorgen, daß die Daten den richtigen Zielrechner erreichen. Um das

bewerbstelligen zu können, muß sie wissen, daß diese Verbindung zu einer bereits bestehenden Verbindung gehört. Dazu ist es notwendig, daß die Firewall das FTP-Protokoll kennt und die übertragenen Kommandos auswertet.

Dies unterscheidet FTP von den bereits betrachteten Protokollen. Es macht für die Firewall keinen Unterschied, ob sie ein HTTP- oder ein NNTP-Paket weiterleitet. Um es zu maskieren, ändert sie die Absenderangabe auf die externe Adresse der Firewall und den Quellport auf einen noch nicht benutzten der Firewall. Dann notiert sie in einer Tabelle die ursprüngliche Quelladresse, den ursprünglichen Quellport und den neu eingetragenen Quellport. Empfängt sie nun ein Paket aus dem Internet, so kann sie anhand des Zielports den passenden Eintrag in der Tabelle herausuchen und das Paket dem eigentlichen Empfänger zustellen. Kenntnisse des verwendeten Anwendungsprotokolls benötigt sie dafür nicht.

Diese Nachteile hat passives FTP nicht. Hier werden die Verbindungen grundsätzlich vom Klienten geöffnet. Man könnte sie daher als zwei voneinander unabhängige Vorkommnisse betrachten, so als würde man gleichzeitig mit HTTP eine Webseite herunterladen, während man mit SMTP eine E-Mail verschickt. Dies war auch die Begründung, warum man seinerzeit zusätzlich zum bis dato genutzten aktiven FTP einen neuen Übertragungsmodus einführte<sup>12</sup>.

Passives FTP ist allerdings auch nicht ganz ohne Nachteile. Bei der Datenverbindung werden zwei beliebige hohe Ports verwendet. Damit müssen wir alle Verbindungen aus dem lokalen Netz zulassen, die an beliebige hohe Ports von Servern im Internet gerichtet sind. Wir können dabei nicht kontrollieren, ob es sich bei einer Verbindung tatsächlich um eine FTP-Datenverbindung handelt. Sie könnte auch zu einem beliebigen anderen Dienst gehören, der auf einem hohen Port auf Anfragen wartet.

Davon abgesehen ist passives FTP aus unserer Sicht aktivem FTP vorzuziehen. Allerdings existieren noch immer Server, die passives FTP nicht unterstützen. Es hilft auch nicht, unseren Anwendern zu erzählen, daß solche Server nicht dem Stand der Technik entsprechen. Wir müssen uns damit abfinden und ihre Existenz akzeptieren.

Lassen Sie mich dies an einem Beispiel belegen. Letztens hatte ich es mit einem namhaften Hersteller von Antivirenprodukten zu tun, dessen FTP-Server sich hinter einer Firewall befindet. Die Firewall war dabei so konfiguriert, daß ein Zugriff mittels passivem FTP nicht möglich war. Wieso diese Einstellung getroffen wurde, ist mir nicht bekannt. In der Folge waren Downloads von Pattern-Updates jedenfalls nur möglich, wenn sie mit aktivem FTP durchgeführt wurden. Da der eigentliche FTP-Server darüber hinaus so konfiguriert war, daß er passives FTP unterstützte, war es nicht möglich, die Downloads mit einem Browser durchzuführen. Dieser versuchte immer, den Download mit passivem FTP durchzuführen, und brach dann ab, wenn die Datenverbindung nicht zustande kam. Glücklicherweise sind moderne FTP-Klienten intelligenter programmiert. Wenn passives FTP nicht funktioniert, schalten sie um auf aktives FTP. Der Versuch, den Betreiber davon zu überzeugen, daß seine Konfiguration nicht sinnvoll ist, dauert mittlerweile schon über zwei Monate. Würden wir in dieser Zeit die Pattern unserer Virens-

<sup>12</sup> Genaugenommen ging es nicht um die Probleme beim Maskieren von Verbindungen, sondern um die Tatsache, daß damals viele Firewalls eingehende Verbindungen generell unterbanden.

canner nicht aktualisieren, wären wir ein lohnendes Ziel für jeden neuen E-Mail-Wurm, der in dieser Zeit den Weg in unser Netz findet.<sup>13</sup>

Kommen wir nun zur technischen Umsetzung. Wie bereits erwähnt, ist es zur Maskierung von FTP-Verbindungen notwendig, daß die Firewall das FTP-Protokoll versteht. Dazu ist es nötig, ein Modul nachzuladen:

```
modprobe ip_masq_ftp
```

Nachdem dies geschehen ist, können wir die eigentlichen Regeln definieren:

```
# FTP

# - Vorbereitung
modprobe ip_masq_ftp

# - Kontrollverbindung
# - - Zugriff auf den Server

ipchains -A ext-in -p TCP --sport 21 --dport 1024:65535 ! -y -j ACCEPT
ipchains -A ext-out -p TCP --dport 21 --sport 1024:65535 -j ACCEPT

# - - Maskieren durch die Firewall

ipchains -A int-in -p TCP --sport 1024:65535 --dport 21 -j ACCEPT
ipchains -A int-fw -p TCP --dport 21 --sport 1024:65535 -j MASQ
ipchains -A int-out -p TCP --sport 21 --dport 1024:65535 ! -y -j ACCEPT

# - aktives FTP
# - - Zugriff auf den Server

ipchains -A ext-in -p TCP --sport 20 --dport 1024:65535 -j ACCEPT
ipchains -A ext-out -p TCP --dport 20 --sport 1024:65535 ! -y -j ACCEPT

# - - Maskieren durch die Firewall

ipchains -A int-in -p TCP --sport 1024:65535 --dport 20 ! -y -j ACCEPT
ipchains -A int-fw -p TCP --dport 20 ! -y -j MASQ
ipchains -A int-out -p TCP --sport 20 --dport 1024:65535 -j ACCEPT

# - passives FTP
# - - Zugriff auf den Server

ipchains -A ext-in -p TCP --sport 1024:65535 --dport 1024:65535 ! -y -j ACCEPT
ipchains -A ext-out -p TCP --dport 1024:65535 --sport 1024:65535 -j ACCEPT

# - - Maskieren durch die Firewall

ipchains -A int-in -p TCP --sport 1024:65535 --dport 1024:65535 -j ACCEPT
ipchains -A int-fw -p TCP --dport 1024:65535 --sport 1024:65535 -j MASQ
ipchains -A int-out -p TCP --sport 1024:65535 --dport 1024:65535 ! -y -j ACCEPT
```

<sup>13</sup> Wenn man in einer großen Firma arbeitet, dann gibt es immer irgendwo einen Anwender, der auch dann noch auf den Anhang einer englischsprachigen E-Mail klickt, wenn sie heiße Liebesgrüße von einem deutschsprachigen Geschäftspartner enthält, der dem gleichen Geschlecht angehört und glücklich verheiratet ist.

Wollen wir einen FTP-Proxy auf der Firewall betreiben, statt die Verbindungen zu maskieren, so sehen unsere Regeln etwas anders aus. Die oben aufgeführten Befehle unter einer Überschrift »- - Maskieren durch die Firewall« entfallen und müssen durch die im folgenden aufgeführten Regeln ersetzt werden.

Wie schon bei den einfachen Anwendungsprotokollen über TCP entfallen auch hier alle Regeln für die Chain `int-fw`, und der Zielrechner wird explizit angegeben:

```
# - Proxy auf der Firewall
# - - Kontrollverbindung
ipchains -A int-in -p tcp --sport 1024:65535 -d 192.168.20.15 21 -j ACCEPT
ipchains -A int-out -p tcp --dport 1024:65535 -s 192.168.20.15 21 ! -y -j ACCEPT

# - - aktives FTP
ipchains -A int-in -p tcp --sport 1024:65535 -d 192.168.20.15 20 ! -y -j ACCEPT
ipchains -A int-out -p tcp --dport 1024:65535 -s 192.168.20.15 20 -j ACCEPT

# - - passives FTP
ipchains -A int-in -p tcp --sport 1024:65535 -d 192.168.20.15 1024:65535 -j ACCEPT
ipchains -A int-out -p tcp --dport 1024:65535 -s 192.168.20.15 1024:65535 ! -y -j ACCEPT
```

Wenn Sie den `ftp-proxy` im `chroot-Käfig` benutzen, dann kann es sein, daß Sie als Quellport für Datenverbindungen bei aktivem FTP nicht Port 20 verwenden können, sondern z. B. auf Port 2020 ausweichen müssen. In diesem Fall müssen Sie die Regeln entsprechend anpassen:

```
# - - aktives FTP
ipchains -A int-in -p tcp --sport 1024:65535 -d 192.168.20.15 2020 ! -y -j ACCEPT
ipchains -A int-out -p tcp --dport 1024:65535 -s 192.168.20.15 2020 -j ACCEPT
```

Der `ftp-proxy` von SuSE kann darüber hinaus noch mit dem transparenten Firewalling des Kernels zusammenarbeiten. Damit ist es möglich, jede FTP-Verbindung auf den Proxy umzuleiten, ohne daß der Benutzer etwas davon merkt.

Hierzu ist es nötig, zusätzlich zu den allgemeinen Proxy-Regeln zwei weitere Regeln für die Kontrollverbindung zu definieren, die greifen, wenn eine direkte Verbindung zu einem FTP-Server im Internet versucht wird:

```
# - - transparente Umleitung
ipchains -A int-in -p TCP --sport 1024:65535 --dport 21 -j REDIRECT 21
ipchains -A int-out -p TCP --sport 21 --dport 1024:65535 ! -y -j ACCEPT
```

Bitte tragen Sie diese beiden Regeln unbedingt nach den anderen Proxy-Regeln ein. Zuerst sollte die Regel greifen, die Zugriffe aus dem lokalen Netz auf den Proxy erlaubt. Erst wenn diese nicht passt, sollte die Regel angesprochen werden, die alle Verbindungen auf den Proxy auf der Firewall umleitet.

## Logging ungewöhnlicher Pakete

Nun haben wir für alle Protokolle, deren Nutzung wir zulassen wollen, Regeln definiert. Pakete, für die immer noch keine Regel gegriffen hat, sind entweder Fehler oder stellen einen Angriff dar. Ohne weitere Regeln würde die Policy der jeweiligen Chain greifen und das Paket stillschweigend verworfen werden.

Dies hätte aber den Nachteil, daß wir von einem möglichen Angriff nicht erfahren. Darum sollten wir nun noch eine letzte Regel für jede Chain definieren, die das Paket nicht nur verwirft, sondern den Vorgang auch im Systemprotokoll vermerkt:

```
# Pakete, die nicht von den normalen Regeln abgedeckt werden
ipchains -A input -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A forward -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A output -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A int-in -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A int-out -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A int-fw -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A ext-in -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
ipchains -A ext-out -s 0.0.0.0/0 -d 0.0.0.0/0 -j DENY -l
```

## Eintragen der Regeln in die Systemdateien

Nun haben wir alle Regeln beisammen und müssen sie nur noch in Skripte eintragen, die vor dem Aufbau einer Verbindung mit dem Internet ausgeführt werden.

Grundsätzlich kommen dafür ein selbsterstelltes Runlevel-Skript und `/etc/ppp/ip-up` in Frage. `/etc/ppp/ip-up` hat aber den Nachteil, daß die Verbindung während seiner Ausführung schon besteht. Den Großteil der Regeln sollten wir daher in ein Runlevel-Skript eintragen. Dieses verlinken wir dann so, daß es vor dem Skript gestartet wird, das die Netzwerk-Interfaces einrichtet und nach diesem gestoppt<sup>14</sup> wird.

Unter SuSE-Linux legen wir keine Links an, sondern bestimmen mit entsprechenden Kommentaren in `proconff` die Startreihenfolge. Wir können das Skript dann ohne weiteres nach `/etc/init.d/` kopieren und mit `insserv` installieren.

Hier ein Beispielskript, das eine einfache Firewall realisiert. Es enthält Regeln für Webproxies auf Port 3128, 8000, 8080 und 8118, einen FTP-Proxy und einen DNS-Server auf der Firewall, diese sind aber durch Voranstellen eines »#« auskommentiert. Wollen Sie für ein Protokoll einen dieser Proxies vorschreiben, so müssen Sie das Zeichen entfernen und bei dem betreffenden Protokoll den Abschnitt »Maskieren durch die Firewall« auskommentieren.

Im Fall von FTP existieren zwei Abschnitte für die Datenverbindung bei aktivem FTP. Die erste Variante benutzt Port 20 und ist damit für konventionelle Proxies wie den `ftp-gw` geeignet.

<sup>14</sup> D. h., mit dem Parameter `stop` aufgerufen wird.

Der zweite Abschnitt ist für Sie dagegen relevant, wenn Sie den ftp-proxy aus der SuSE Proxy-Suite verwenden und von der Möglichkeit Gebrauch machen wollen, diesen in einem chroot-Käfig zu betreiben. In diesem Fall müssen Sie einen Port über 1023 vorgeben. Bei der Beschreibung des Proxys habe ich Port 2020 gewählt.

Am Anfang des Skripts stehen mehrere Variablen, die noch an die konkrete Anwendungssituation angepaßt werden müssen:

**EXTIF** Der Name des externen Interfaces. Übliche Namen sind ppp0 (Modem, DSL) und ipp0 (ISDN).

**INTIP** Die Adresse der Firewall im lokalen Netz.

**INTIF** Der Name des internen Interfaces. Üblicherweise handelt es sich um ein Ethernet-Interface, z. B. eth0.

**INTBITS** Eine andere Art, eine Netzwerkmaske anzugeben. Hat Ihr internes Interface z. B. die Adresse 192.168.20.15 und es sollen im LAN alle Adressen der Art 192.x.x.x möglich sein, so interessieren uns nur die ersten 8 Bit einer Adresse, wenn wir entscheiden wollen, ob ein Rechner sich im selben Subnetz wie die Firewall befindet. Wir geben dann also eine 8 an. Für einen Adreßbereich 192.168.x.x nehmen wir dementsprechend 16, und für 192.168.20.x ist der zugehörige Wert 24.

**TCPPROTECTED** Die Nummern der TCP-Ports über 1023, auf denen Server aktiv sind, die nicht aus dem Internet zugreifbar sein sollen (z. B. Proxies).

**UDPPROTECTED** Die Nummern der UDP-Ports über 1023, auf denen Server aktiv sind, die nicht aus dem Internet zugreifbar sein sollen (z. B. Proxies).

Zwei weitere Variablen werden automatisch gefüllt. Sie brauchen sie daher nicht zu ändern:

**INTNET** Der Adreßbereich, der im lokalen Netz verwendet wird. Diesen Wert brauchen Sie nicht zu ändern, er wird aus *INTIP* und *INTBITS* berechnet.

**DNSSERVER** Die IP-Adressen der DNS-Server. Die Adressen werden aus der Datei */etc/resolv.conf* ausgelesen. Sie brauchen sie daher hier nicht noch ein zweites Mal anzugeben.

Hier nun das Skript:

```
#!/bin/sh
#####
#
# pfilter.ipchains
#
#   Filterregeln für IP-Pakete mit ipchains
#
# Usage: pfilter {start|stop}
#
# Copyright (C) 2003 Andreas Lessig
#
```

```
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
#
#####

### BEGIN INIT INFO
# Provides:          pf-ipc
# Required-Start:    $local_fs
# Should-Start:
# Required-Stop:     proccnf
# Should-Stop:
# Default-Start:     2 3 5
# Default-Stop:      0 1 6
# Short-Description: Firewalling
# Description:       Firewalling
### END INIT INFO

# -----
# Grundeinstellungen
# -----

# Eine Kurzbezeichnung für ipchains

R=/sbin/ipchains

# Ein paar grundsätzliche Daten
# Diese müssen an die eigenen Bedürfnisse angepaßt werden

# - Das externe Interface

EXTIF="ppp0"

# - Das interne Interface

INTIP="192.168.20.15"
INTIF="eth0"
INTBITS=24
INTNET="$INTIP"/"$INTBITS"

# - Der DNS-Server des Providers

DNSSERVER='cat /etc/resolv.conf | grep '^nameserver' | sed 's/nameserver//''

# - Hohe Ports, die aus dem Internet nicht zugreifbar sein sollen

# - - TCP

TCPPROTECTED='3128 8000 8080 8118'
```

```
# - - UDP
UDPPROTECTED='515'

# -----
# Falls das Skript mit falschen Parametern aufgerufen wurde
# -----

case "$1" in
start)
    echo "Starte die Firewall ..."
    ;;
stop)
    echo "Beende die Vermittlung von Paketen ..."
    ;;
*)
    echo Usage: $0 {start|stop}"
    exit 1
    ;;
esac

# -----
# Regeln, die immer gelten
# -----

# Alle Regeln und selbstdefinierten Chains löschen

$R -F
$R -X

# Alle Pakete, die nicht explizit erlaubt sind, sind verboten

$R -P input DENY
$R -P forward DENY
$R -P output DENY

# Protokollierung gespoofter Pakete

$R -A input -i !"$INTIF" -s "$INTNET" -j DENY -l
$R -A input -i !lo -s 127.0.0.1 -j DENY -l

# Lokale Pakete sind erlaubt

$R -A input -i lo -j ACCEPT
$R -A output -i lo -j ACCEPT

# NetBIOS über TCP/IP

$R -A input -p UDP -s "$INTNET" 137:139 -j DENY
$R -A input -p UDP -s "$INTNET" --dport 137:139 -j DENY
$R -A input -p TCP -s "$INTNET" 137:139 -j DENY
$R -A input -p TCP -s "$INTNET" --dport 137:139 -j DENY

case $1 in
# -----
# Die Firewall soll heruntergefahren werden
# -----
stop)
```

```
# Protokollierung ungewöhnlicher Pakete

$R -A input -s 0.0.0.0/0 -j DENY -l
$R -A output -s 0.0.0.0/0 -j DENY -l
$R -A forward -s 0.0.0.0/0 -j DENY -l

;;

# -----
# Die Firewall soll ihre Arbeit aufnehmen
# -----
start)

# ICMP

$R -A input -p icmp --sport 0 -j ACCEPT
$R -A input -p icmp --sport 3 -j ACCEPT
$R -A input -p icmp --sport 8 -j ACCEPT
$R -A input -p icmp --sport 11 -j ACCEPT
$R -A input -p icmp --sport 12 -j ACCEPT
$R -A output -p icmp -j ACCEPT

# Eigene Chains

# - Externes Interface

$R -N ext-in
$R -N ext-out

# - Internes Interface

$R -N int-in
$R -N int-fw
$R -N int-out

# - Verteilung der Pakete auf die Chains

$R -A input -i "$INTIF" -j int-in
$R -A input -i "$EXTIF" -j ext-in
$R -A forward -i "$EXTIF" -s "$INTNET" -j int-fw
$R -A output -i "$INTIF" -j int-out
$R -A output -i "$EXTIF" -j ext-out

# Zugriffe auf Server der Firewall

# - TCP

for port in $TCPPROTECTED
do
    $R -A ext-in -p tcp --dport $port -j DENY -l
done

# - UDP

for port in $UDPPROTECTED
do
    $R -A ext-in -p udp --dport $port -j DENY -l
done
```

```
# DNS
# - Alle eingetragenen Server freischalten
for DNS in $DNSSERVER
do
# - Zugriff auf den externen Server
$R -A ext-in -p udp -s "$DNS" 53 --dport 1024:65535 -j ACCEPT
$R -A ext-out -p udp -d "$DNS" 53 --sport 1024:65535 -j ACCEPT

$R -A ext-in -p tcp -s "$DNS" 53 --dport 1024:65535 ! -y -j ACCEPT
$R -A ext-out -p tcp -d "$DNS" 53 --sport 1024:65535 -j ACCEPT

# - Maskieren durch die Firewall

$R -A int-in -p udp --sport 1024:65535 -d "$DNS" 53 -j ACCEPT
$R -A int-fw -p udp -d "$DNS" 53 -j MASQ
$R -A int-out -p udp -s "$DNS" 53 -j ACCEPT

$R -A int-in -p tcp --sport 1024:65535 -d "$DNS" 53 -j ACCEPT
$R -A int-fw -p tcp -d "$DNS" 53 -j MASQ
$R -A int-out -p tcp -s "$DNS" 53 ! -y -j ACCEPT

done

# - Server auf der Firewall

# $R -A int-in -p udp -d "$INTIP" 53 -j ACCEPT
# $R -A int-out -p udp -s "$INTIP" 53 -j ACCEPT

# $R -A int-in -p tcp -d "$INTIP" 53 -j ACCEPT
# $R -A int-out -p tcp -s "$INTIP" 53 ! -y -j ACCEPT

# - Transparente Umleitung

$R -A int-in -p UDP --sport 1024:65535 --dport 53 -j REDIRECT 53
$R -A int-out -p UDP --sport 53 --dport 1024:65535 -j ACCEPT

$R -A int-in -p TCP --sport 1024:65535 --dport 53 -j REDIRECT 53
$R -A int-out -p TCP --sport 53 --dport 1024:65535 ! -y -j ACCEPT

# Ident

$R -A ext-in -p tcp --dport 113 -j REJECT -l

# HTTP

# - Zugriff auf den Server

$R -A ext-in -p tcp --dport 1024:65535 --sport 80 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 80 -j ACCEPT
```

```
# - Maskieren durch die Firewall

$R -A int-in -p tcp --sport 1024:65535 --dport 80 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 80 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 80 ! -y -j ACCEPT

# - Proxies
# - - Proxy auf Port 3128 (squid)

# $R -A int-in -p TCP --sport 1024:65535 -d "$INTIP" 3128 -j ACCEPT
# $R -A int-out -p TCP --dport 1024:65535 -s "$INTIP" 3128 ! -y -j ACCEPT

# - - Proxy auf Port 8000 (junkbuster)

# $R -A int-in -p TCP --sport 1024:65535 -d "$INTIP" 8000 -j ACCEPT
# $R -A int-out -p TCP --dport 1024:65535 -s "$INTIP" 8000 ! -y -j ACCEPT

# - - Proxy auf Port 8080 (http-gw)

# $R -A int-in -p TCP --sport 1024:65535 -d "$INTIP" 8080 -j ACCEPT
# $R -A int-out -p TCP --dport 1024:65535 -s "$INTIP" 8080 ! -y -j ACCEPT

# - - Proxy auf Port 8118 (privoxy)

# $R -A int-in -p TCP --sport 1024:65535 -d "$INTIP" 8080 -j ACCEPT
# $R -A int-out -p TCP --dport 1024:65535 -s "$INTIP" 8080 ! -y -j ACCEPT

# - Transparente Umleitungen
# - - Transparente Umleitung auf Port 3128 (squid)

# $R -A int-in -p TCP --sport 1024:65535 --dport 80 -j REDIRECT 3128
# $R -A int-out -p TCP --sport 80 --dport 1024:65535 ! -y -j ACCEPT

# - - Transparente Umleitung auf Port 8118 (privoxy)

# $R -A int-in -p TCP --sport 1024:65535 --dport 80 -j REDIRECT 8118
# $R -A int-out -p TCP --sport 80 --dport 1024:65535 ! -y -j ACCEPT

# HTTPS

# - Zugriff auf den Server

$R -A ext-in -p tcp --dport 1024:65535 --sport 443 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 443 -j ACCEPT

# - Maskieren durch die Firewall

$R -A int-in -p tcp --sport 1024:65535 --dport 443 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 443 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 443 ! -y -j ACCEPT

# SMTP

# - Zugriff auf den Server

$R -A ext-in -p tcp --dport 1024:65535 --sport 25 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 25 -j ACCEPT
```

```
# - Maskieren durch die Firewall
$R -A int-in -p tcp --sport 1024:65535 --dport 25 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 25 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 25 ! -y -j ACCEPT

# POP3
# - Zugriff auf den Server
$R -A ext-in -p tcp --dport 1024:65535 --sport 110 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 110 -j ACCEPT

# - Maskieren durch die Firewall
$R -A int-in -p tcp --sport 1024:65535 --dport 110 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 110 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 110 ! -y -j ACCEPT

# POP3S
# - Zugriff auf den Server
$R -A ext-in -p tcp --dport 1024:65535 --sport 995 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 995 -j ACCEPT

# - Maskieren durch die Firewall
$R -A int-in -p tcp --sport 1024:65535 --dport 995 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 995 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 995 ! -y -j ACCEPT

# IMAP
# - Zugriff auf den Server
$R -A ext-in -p tcp --dport 1024:65535 --sport 143 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 143 -j ACCEPT

# - Maskieren durch die Firewall
$R -A int-in -p tcp --sport 1024:65535 --dport 143 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 143 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 143 ! -y -j ACCEPT

# IMAPS
# - Zugriff auf den Server
$R -A ext-in -p tcp --dport 1024:65535 --sport 993 ! -y -j ACCEPT
$R -A ext-out -p tcp --sport 1024:65535 --dport 993 -j ACCEPT

# - Maskieren durch die Firewall
$R -A int-in -p tcp --sport 1024:65535 --dport 993 -j ACCEPT
$R -A int-fw -p tcp --sport 1024:65535 --dport 993 -j MASQ
$R -A int-out -p tcp --dport 1024:65535 --sport 993 ! -y -j ACCEPT
```

```
# NNTP

# - Zugriff auf den Server

$I -A ext-in -p tcp --dport 1024:65535 --sport 119 ! -y -j ACCEPT
$I -A ext-out -p tcp --sport 1024:65535 --dport 119 -j ACCEPT

# - Maskieren durch die Firewall

$I -A int-in -p tcp --sport 1024:65535 --dport 119 -j ACCEPT
$I -A int-fw -p tcp --sport 1024:65535 --dport 119 -j MASQ
$I -A int-out -p tcp --dport 1024:65535 --sport 119 ! -y -j ACCEPT

# FTP

# - Vorbereitung

/sbin/modprobe ip_masq_ftp

# - Zugriff auf den Server

# - - Kontrollverbindung

$I -A ext-in -p tcp --dport 1024:65535 --sport 21 ! -y -j ACCEPT
$I -A ext-out -p tcp --sport 1024:65535 --dport 21 -j ACCEPT

# - - Aktives FTP

$I -A ext-in -p tcp --dport 1024:65535 --sport 20 -j ACCEPT
$I -A ext-out -p tcp --sport 1024:65535 --dport 20 ! -y -j ACCEPT

# - - Passives FTP

$I -A ext-in -p tcp --dport 1024:65535 --sport 1024:65535 ! -y -j ACCEPT
$I -A ext-out -p tcp --sport 1024:65535 --dport 1024:65535 -j ACCEPT

# - Maskieren durch die Firewall

# - - Kontrollverbindung

$I -A int-in -p tcp --sport 1024:65535 --dport 21 -j ACCEPT
$I -A int-fw -p tcp --sport 1024:65535 --dport 21 -j MASQ
$I -A int-out -p tcp --dport 1024:65535 --sport 21 ! -y -j ACCEPT

# - - Aktives FTP

$I -A int-in -p tcp --sport 1024:65535 --dport 20 ! -y -j ACCEPT
$I -A int-fw -p tcp --sport 1024:65535 --dport 20 ! -y -j MASQ
$I -A int-out -p tcp --dport 1024:65535 --sport 20 -j ACCEPT

# - - Passives FTP

$I -A int-in -p tcp --sport 1024:65535 --dport 1024:65535 -j ACCEPT
$I -A int-fw -p tcp --sport 1024:65535 --dport 1024:65535 -j MASQ
$I -A int-out -p tcp --dport 1024:65535 --sport 1024:65535 ! -y -j ACCEPT
```

```
# - Proxy auf der Firewall
# - - Kontrollverbindung
# $R -A int-in -p tcp --sport 1024:65535 -d "$INTIP" 21 -j ACCEPT
# $R -A int-out -p tcp --dport 1024:65535 -s "$INTIP" 21 ! -y -j ACCEPT

# - - Aktives FTP
# - - - "Normaler Proxy"
# $R -A int-in -p tcp --sport 1024:65535 -d "$INTIP" 20 ! -y -j ACCEPT
# $R -A int-out -p tcp --dport 1024:65535 -s "$INTIP" 20 -j ACCEPT

# - - - Proxy im chroot-Käfig mit ungewöhnlichem Datenport
#       (SuSE ftp-proxy)
# $R -A int-in -p tcp --sport 1024:65535 -d "$INTIP" 2020 ! -y -j ACCEPT
# $R -A int-out -p tcp --dport 1024:65535 -s "$INTIP" 2020 -j ACCEPT

# - - Passives FTP
# $R -A int-in -p tcp --sport 1024:65535 -d "$INTIP" 1024:65535 -j ACCEPT
# $R -A int-out -p tcp --dport 1024:65535 -s "$INTIP" 1024:65535 ! -y -j ACCEPT

# - - Transparente Umleitung
# $R -A int-in -p TCP --sport 1024:65535 --dport 21 -j REDIRECT 21
# $R -A int-out -p TCP --sport 21 --dport 1024:65535 ! -y -j ACCEPT

# Protokollierung ungewöhnlicher Pakete
$I -A input -s 0.0.0.0/0 -j DENY -l
$I -A output -s 0.0.0.0/0 -j DENY -l
$I -A forward -s 0.0.0.0/0 -j DENY -l
$I -A int-in -s 0.0.0.0/0 -j DENY -l
$I -A int-out -s 0.0.0.0/0 -j DENY -l
$I -A int-fw -s 0.0.0.0/0 -j DENY -l
$I -A ext-in -s 0.0.0.0/0 -j DENY -l
$I -A ext-out -s 0.0.0.0/0 -j DENY -l

;;
esac
```

Dieses Skript enthält aber noch keine Regel, um Pakete zu verwerfen, die aus dem Internet stammen, als Absenderadresse aber die externe Adresse der Firewall eingetragen haben. Dies liegt daran, daß wir zu dem Zeitpunkt, an dem das Skript ausgeführt wird, diese Adresse noch nicht kennen. Sie wird uns in der Regel erst vom Provider zugewiesen, wenn wir uns ins Internet einwählen.

Hier kommt nun das Skript `/etc/ppp/ip-up` ins Spiel. Dieses wird ausgeführt, wenn die Verbindung zustande gekommen ist und uns eine Adresse zugewiesen wurde.

Ist eine solche Datei bereits vorhanden und haben wir sie nicht selbst erzeugt (z. B. für ISDN), so sollten wir sie umbenennen und unsere eigene erzeugen. Das gleiche gilt für das Skript `/etc/ppp/ip-down`:

```
# cd /etc/ppp
# mv ip-up ip-up.orig
# mv ip-down ip-down.orig
# touch ip-up
# touch ip-down
# chmod 700 ip-up ip-down
```

Beachten Sie bitte den `chmod`-Befehl. Die Skripte müssen die richtigen Dateirechte besitzen, oder sie werden nicht ausgeführt.

In `/etc/ppp/ip-up` können wir nun die entsprechende Regel eintragen. Das Skript bekommt die externe Adresse als viertes Argument übergeben:

```
#!/bin/sh

EXTIP=$4

# gespoofte Pakete der Firewall
ipchains -I input 1 -i !lo -s $EXTIP -j DENY -1
```

Die Adresse ist nur so lange gültig, wie eine Verbindung zum Internet besteht. Ist die Verbindung beendet, so müssen wir die Regel wieder löschen. Hierzu verwenden wir am besten das Skript `/etc/ppp/ip-down`:

```
#!/bin/sh

EXTIP=$4

# gespoofte Pakete der Firewall
ipchains -D input -i !lo -s $EXTIP -j DENY -1
```