IP Filter Based Firewalls HOWTO

Brendan Conoboy <synk@swcp.com>
Erik Fichtner <emf@obfuscation.org>

Sat Jan 13 11:22:26 EST 2001

Übersetzung von Thorsten Lindloff
E-Mail: tlindloff@t-online.de
URL: www.lindloff.com

Formatierung und Überarbeitung von Peter Bauer
E-Mail: peter.bauer@itserv.de
URL: http://www.itserv.de

Abstraktes: Dieses Dokument soll einen neuen User in den Gebrauch des IPFILTER-Firewalling-Pakets einführen und gleichzeitig dem Benutzer das fundamentale Grundwissen im guten Firewall-Design vermitteln.

Wer Schreibfehler findet darf sie behalten.

1. Einführung

IPFilter ist ein großartiges, kleines Firewallpaket. Es macht alles, was andere freie Firewalls auch tun (ipchains, ipfwadm); aber es ist auch genauso portierbar und leistet ein paar interessante Dinge, die die Anderen nicht tun. Dieses Dokument wurde verfaßt, um die zur Zeit spärliche Dokumentation von IPFilter zu verbessern. Ein paar Vorkenntnisse in der Konfiguration von Paketfiltern sind nützlich; wobei allerdings bei zu genauer Kenntnis dieser Programme das Lesen dieses Dokumentes eine gewisse Zeitverschwendung darstellt. Um einen tieferen Einblick in die Materie zu gewinnen, empfehlen die Autoren das Buch "Building Interent Firewalls" von Chapman & Zwicky aus dem O'Reilly-Verlag und "TCP/IP Illustrated, Volume 1" von Stevens aus dem Verlag Addison-Wesley

1.1 Haftungsausschluss

Die Autoren dieses Dokumentes sind nicht für Probleme verantwortlich, die bei Aktionen, die nach dieser Anleitung durchgeführt werden, entstehen können. Das Dokument ist als eine Einführung in den Bau einer IPFilter-basierten Firewall gedacht. Wenn Du nicht komfortabel damit bist, die Verantwortung für Deine eigenen Aktionen zu übernehmen, dann suche Dir lieber einen qualifizierten Profi für diese Aufgabe, der die Firewall für Dich aufsetzt.

1.2. Copyright

1.3. Wo es die wichtigsten Teile gibt

Die offizielle IPF-Homepage ist unter:
<http://coombs.anu.edu.au/~avalon/ip-filter.html>

Die neueste Version findet sich unter:
<http://www.obfuscation.org/ipf/>

2. Firewalling -Die Basics

Diese Sektion dient dazu, Dich mit der ipfilter-Syntax und der Firewall-Theorie im Allgemeinen vertraut zu machen. Die hier diskutierten Features finden sich in jedem guten Firewallpaket. Dieser Teil des Dokumentes soll ein solides Fundament im einfachen Lesen und Verstehen des "Advanced-Teils" sein. Es muß einem aber bewußt sein, daß dieser Teil für sich genommen nicht ausreichend ist, um eine gute Firewall zu aufzusetzen. Der "Advanced-Teil" ist wirklich Voraussetzung, für jeden, der ein effektives Sicherheitssystem braucht.

2.1. Konfigurationsdatei: Reihenfolge, Dynamik und Vorrang

IPF (IPFilter) besitzt eine Konfigurationsdatei (um nicht zu sagen, daß jedes Kommando noch mal und noch mal für jede Regel ausgeführt wird). Die Datei entspricht den Unix-Standards: Die "#" markiert einen Kommentar; und man kann Regeln und Kommentare gemeinsam in der selben Zeile haben. Zusätzlicher "Whitespace" ist erlaubt und für eine bessere Lesbarkeit der Regeln empfehlenswert.

2.2. Ausführen der Regeln

Die Regeln werden immer von Anfang bis Ende der Datei ausgeführt; immer eine nach der anderen. Das meint schlicht und ergreifend, daß eine Regel wie diese:

```
block in all
pass  in all
```

...vom Computer so gesehen wird:

```
block in all
pass  in all
```

Was bedeutet, daß, falls ein Paket hereinkommt, ist das erste, was IPF tut das hier:

```
block in all
```

Sollte IPF es für nötig halten, zur nächsten Regel zu gehen, dann wendet es die zweite Regel an:

```
pass  in all
```

An diesem Punkt könntest Du Dich selbst fragen "Geht IPF weiter zu zweiten Regel?". Wenn Du Dich mit ipfwadm oder ipfw auskennst, wirst Du das möglicherweise nicht tun. Eine kurze Zeit später wirst Du Dich wundern, daß die Pakete immer dann abgelehnt oder angenommen werden, wenn das nicht geschehen sollte. Viele Paketfilter stoppen mit dem Vergleich der Pakete, wenn eine Regel auf diese Pakete zutrifft. IPF gehört nicht dazu. Anders als andere Paketfilter setzt IPF ein Flag, das beschreibt, ob oder ob das Paket nicht zu einer Regel paßt. Solange du den Fluß nicht unterbrichst, geht IPF durch den gesamten Regelsatz, um eine Entscheidung zu fällen, ob das Paket hereindarf oder in der letzten Regel verworfen werden soll. Die Szenerie: IPFilter ist aktiv. Es nimmt einen Teil der CPU-Zeit für sich in Anspruch. Es hat ein Checkpoint-Clipboard, das dieses hier liest:

```
block in all
pass  in all
```

Ein Paket kommt in das Interface und es ist Zeit, an die Arbeit zu gehen. Es legt einen Blick auf das Paket und auf die erste Regel:

```
block in all
```

"Soweit denke ich, ich sollte das Paket blockieren" sagt IPF. Dann wendet es seinen Blick der zweiten Regel zu:

```
pass  in all
```

"Soweit denke ich, ich sollte das Paket passieren lassen" sagt IPF. Es legt einen Blick auf Regel Nummer drei. Es gibt aber keine dritte Regel. Also tut es das, was seine letzte Entscheidung war: Das Paket darf die Firewall passieren. Jetzt ist die Zeit günstig, herauszufinden, was passiert, wenn der Regelsatz so aussehen würde:

```
block in all
block in all
block in all
block in all
pass  in all
```

Das Paket dürfte immer noch passieren. Es gibt keinen Gesamteffekt. Die letzte Regel hat immer Vorrang.

2.3. Die Ausführung der Regeln kontrollieren:

Wenn Du Erfahrung mit anderen Paketfiltern hast, kann es sein, daß Du dieses Layout als etwas konfus empfindest. Du könntest darüber spekulieren, daß Probleme mit der Austauschbarkeit mit anderen Filtern und der Geschwindigkeit der Überprüfung, ob eine Regel paßt, auftreten könnten. Stelle Dir vor, Du hast 100 Regeln und die wichtigsten sind die ersten zehn. Das gibt einen fürchterlichen Overhead, weil jedes Paket jedes Mal alle 100 Regeln passieren muß. Glücklicherweise gibt es aber ein einfaches Schlüsselwort, das in jede Regel eingesetzt werden kann, um bei einem Paket eine Aktion hervorzurufen. Dieses Schlüsselwort heißt *quick*. Das hier ist eine modifizierte Kopie vom Original-Regelsatz, die das "quick"-Schlüsselwort nutzt:

```
block in quick all
pass  in       all
```

In diesem Fall betrachtet sich IPF die erste Regel:

```
block in quick all
```

Das Paket paßt und die Suche ist vorbei. Das Paket ist geräuschlos verworfen worden. Es gibt keine Notizen, keine Logeinträge, keine feierliche Beerdigung. Kuchen wird übrigens auch nicht serviert.

Und was ist nun mit der nächsten Regel?

```
pass  in       all
```

Diese Regel wird nie abgearbeitet werden. Man kann in der Konfigurationsdatei schlicht und ergreifend auf sie verzichten. Das alles hinwegfegende Match aller Pakete und das terminierende Schlüsselwort "quick" aus der vorhergehenden Regel macht klar, daß nach ihr keine weiteren Regeln folgen werden.

Die Hälfte einer Konfigurationsdatei zu verschwenden, ist nur selten eine gute Idee. Auf der anderen Seite ist da IPF, um Pakete zu blockieren; und es leistet gute Dienste, wenn es richtig konfiguriert ist. Nicht zuletzt ist es auch dazu da, einige Pakete durchzulassen. Mit einer Änderung des Regelsatzes ist das möglich, wenn das gewünscht wird.

2.4. Einfaches Filtern nach IP-Adressen

IPF kann nach vielen Regeln filtern. Eine davon ist es, festzulegen, daß in bestimmten Blöcken des Adressraumes kein Verkehr stattfinden soll. Ein solcher Block ist der aus den nicht routebaren Netzwerken 192.168.0.0/16 (/16 ist die CIDR-Notation für eine Netzmaske). Es kann sein, daß Du mit dem Dezimalformat besser vertraut bist: 255.255.0.0 . IPF akzeptiert beide Formate). Wenn Du diesen Bereich blockieren willst, ist dies hier ein Weg:

```
block in quick from 192.168.0.0/16 to any
pass  in       all
```

Jetzt haben wir einen weniger strengen Regelsatz, der derzeit etwas für uns tut. Die erste Regel wird geprüft:

```
block in quick from 192.168.0.0/16 to any
```

Das Paket kommt von 1.2.3.4, nicht von 192.168.*.*; es gibt hier kein Match Die zweite Regel wird geprüft:

```
pass  in       all
```

Das Paket von 1.2.3.4 ist eindeutig ein Paket von all; das Paket wird gesendet, was auch immer sein Ziel ist und dort mit ihm passiert.
Auf der anderen Seite haben wir vermutlich ein Paket, das von 192.168.1.2 kommt. Die erste Regel wird geprüft:

```
block in quick from 192.168.0.0/16 to any
```

Hier ist ein Match. Das Paket wird verworfen und das ist das bittere Ende. Noch mal: Es geht nicht durch die zweite Regel, weil die erste Regel ein Match war und sie das "quick"-Schlüsselwort enthielt.

An diesem Punkt kannst Du einen ziemlich ausschweifenden Satz von Regeln aufsetzen, der festlegt, was die Firewall passieren darf oder nicht. Seit wir begonnen haben, private Adressen zu blockieren, sollten wir auch den anderen Bereichen die nötige Beachtung schenken:

```
block in quick from 192.168.0.0/16 to any
block in quick from 172.16.0.0/12 to any
block in quick from 10.0.0.0/8 to any
pass  in       all
```

Die ersten drei Regeln blockieren einiges aus den privaten Adressräumen.

2.5. Interfaces kontrollieren

Es kommt öfter vor, daß Firmen schon ein internes Netzwerk besitzen, bevor sie eine Verbindung mit der Welt "draußen" einrichten wollen. Genauer gesagt: Es ist möglicherweise der Hauptgrund für die Einrichtung einer Firewall auf dem "ersten Rechner am Netz". Die Maschine, die die Brücke von der Welt "draußen" zur Welt "drinnen" darstellt, ist der Router. Was den Router von anderen Maschinen unterscheidet, ist einfach: Er hat mehr als ein Netzwerkinterface. Jedes Paket, das Du empfängst und jedes Paket, das Deinen Rechner verläßt, geht durch so ein Interface. Sagen wir mal, Deine Maschine hat drei Interfaces: lo0 (loopback), x10 (3com-ethernet) und tun0 (FreeBSD-spezifischer generischer Tunnel; das Interface, das ppp benutzt); aber Du willst nicht, daß irgendwelche Pakete über das tun0 Interface hereinkommen? Das sieht so aus:

```
block in quick on tun0 all
pass  in               all
```

In diesem Fall bedeutet das "*on*"-Schlüsselwort, daß ein Datenpaket über das genannte Interface hereinkommt. Wenn es über tun0 kommt, blockiert die erste Regel das Paket. Wenn es über lo0 oder x10 kommt, paßt die erste Regel nicht. Die zweite Regel paßt allerdings und das Paket darf die Firewall passieren.

2.6. Die Benutzung von IP-Adressen mit den Interfaces

Es ist schon ein etwas sonderbarer Zustand, wenn sich jemand entscheidet, das tun0-Interface zu aktivieren, ohne das irgendwelche Daten dieses Interface passieren dürfen. Eine weitere Eigenschaft einer Firewall ist die, daß sie mit einer zunehmenden Anzahl an Regeln entweder fester oder lockerer werden kann. Das ist der Startpunkt für eine effektive Firewall.

```
block in quick on tun0 from 192.168.0.0/16 to any
pass  in       all
```

Vergleiche dies mit der vorherigen Regel:

```
block in quick from 192.168.0.0/16 to any
pass  in       all
```

Der alte Weg, allen Verkehr von 192.168.0.0/16 ohne Beachtung eines Interfaces zu blockieren, war der einer vollständigen Blockade. Auf dem neuen Weg, über tun0, wird das Paket nur blockiert, wenn es über tun0 hereinkommt. Wenn ein Paket über das x10-Interface ankommt, darf es herein.

Auch an diesem Punkt kannst Du einen ziemlich ausschweifenden Regelsatz definieren, die entweder durchläßt oder blockiert Schon seit wir damit begonnen haben, private Adressräume über tun0 zu blockieren, sollten wir dem Rest der Regeln etwas Beachtung zukommen lassen.

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
pass  in       all
```

Du hast schon die ersten drei Blöcke gesehen, aber nicht den Rest. Die vierte Regel ist ein ausgedehntes Class-A-Netzwerk, das für den Loopback gebraucht wird. Es gibt viel Software, die mit sich selbst auf 127.0.0.1 kommuniziert. Demnach ist es eine gute Idee, Eingänge von externen Quellen zu blockieren. Die Fünfte, "0.0.0.0 /8" sollte niemals im Internet auftauchen. Viele IP-Stacks sehen "0.0.0.0/32" als den Default-Gateway an. Und der Rest des "0.0.0.0"-Netzwerks wird von verschiedenen Systemen als ein Nebenprodukt aus den Routing-Entscheidungen betrachtet. Du solltest 0.0.0.0/8 genau so betrachten wie 127.0.0.0/8. 169.254.0.0/16 wurde von der IANA für die Autokonfiguration von Rechnern reserviert, die noch nicht in der Lage sind, eine IP-Adresse via DHCP oder Ähnlichem zu beziehen. Microsoft Windows wird Adressen in diesem Bereich nutzen, wenn es auf DHCP eingestellt ist und keinen Server findet. 192.0.2.0/24 ist zum Gebrauch als beispielhafter Netzwerkblock für Autoren von Dokumentationen ebenfalls reserviert. Wir benutzen diese spezifizierten Bereiche nicht, weil das Konfusionen verursachen würde. Darum kommen alle unsere Beispiele von 20.20.20.0/25. 204.152.64.0/23 ist ein merkwürdiger Netzblock, der von Sun Microsystems für private Cluster-Interconnects reserviert wurde. Ob Du diese Adressen blockierst, mußt Du selbst entscheiden. Als Letztes gibt es da noch 224.0.0.0/3, die den Bereich der Klassen D und E abdecken und meistens für Multicast-Verkehr genutzt werden, obwohl ein "Class E"-Adressraum in RFC1166 beschrieben ist.

Es gibt da einen wichtigen Grundsatz im Paketfiltern, der besagt, daß Pakete, die eigentlich nur aus einem privaten Netz kommen können, zu blockieren sind: Wenn Du weißt, daß bestimmte Arten von Daten nur von einer bestimmten Stelle kommen können, dann wird das System so konfiguriert, daß diese Art von Daten nur von diesem Platz kommen kann. Im Fall der unroutebaren Adressen weißt Du, daß von 10.0.0.0/8 nichts auf dem Interface tun0 ankommen darf, weil es keinen Weg gibt, solche Anfragen zu beantworten. Es ist also ein illegitimes Paket. Das Gleiche gilt auch für die anderen unroutebaren Adressen wie 127.0.0.0/8. Viele Teile von Software erledigen ihre gesamte Authentifizierung auf der Basis der Herkunftsadresse des Pakets. Wenn Du ein internes Netzwerk betreibst, sagen wir 20.20.20.0/24, weißt Du, daß der Verkehr nur aus dem lokalen Ethernet kommen kann. Sollte ein Paket über einen ppp-Dialup hereinkommen, hast Du einen wichtigen Grund, das Paket fallen zu lassen oder es in einem dunklen Raum zu verhören. Es sollte aus keinem anderen Grund die Firewall zu seinem Ziel passieren dürfen. Du kannst das sehr einfach erreichen, wenn du etwas über IPF weißt. Der neue Regelsatz sieht jetzt so aus:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
block in quick on tun0 from 20.20.20.0/24 to any
pass  in       all
```

2.7. Bidirektionales Filtern; das "out" Schlüsselwort

Bis hierher haben wir nur den eingehenden oder ausgehenden Verkehr durchgelassen oder blockiert. Um Klarheit zu schaffen: Eingehender Verkehr ist alles, was über irgendein Interface in die Firewall hereinkommt. Ausgehender Verkehr ist genau das Gegenteil davon: Er verläßt die Firewall über irgendein Interface. Das heißt auch, daß nicht nur der Verkehr gefiltert wird, der über die Firewall hereinkommt, sondern auch der Verkehr, der

hinausgeht. Bis hierher haben wir eingehenden Verkehr entweder durchgelassen oder blockiert. Um zu klären, was damit gemeint ist, wiederhole ich das hier noch mal: **Eingehender Verkehr sind alle Daten, die die Firewall über irgendein Interface in den Rechner hinein passieren**; egal, welcher Art diese Pakete sind. Im Gegensatz dazu steht der ausgehende Verkehr: Dieser verläßt der Rechner; egal über welches Interface (Hierbei ist es egal, ob dieser Verkehr nur weitergeleitet wird oder vom Host selbst ausgeht.). Das bedeutet, daß alle Pakete nicht nur gefiltert werden, wenn sie hereinkommen, sondern auch, wenn sie den Rechner verlassen. Bisher war das nur ein implementierter Ausgang von allen Paketen, wobei es nicht entscheidend war, ob diese Pakete die Firewall verlassen durften oder auch nicht. Genauso wie eingehender Verkehr blockiert werden kann, kann man das auch mit dem ausgehenden Verkehr machen. Jetzt wissen wir, daß es einen Weg gibt, den ausgehenden Verkehr genauso zu filtern wie den eingehenden. Wir sind jetzt so weit, daß wir uns an die Arbeit machen können, einen Weg zu finden, um genau dieses zu tun. Eine Möglichkeit, diese Funktion zu nutzen ist, zu verhindern, daß gespoofte (und für Angreifer interessante) Pakete das eigene Netzwerk verlassen können. Anstatt den gesamten Verkehr nach außen durch den Router zu lassen, kann der ausgehende Verkehr auf Pakete von 20.20.20.0/24 limitiert werden.

Das kannst Du so machen:

```
pass  out quick on tun0 from 20.20.20.0/24 to any
block out quick on tun0 from any to any
```

Wenn ein Paket von 20.20.20.1/32 kommt, wird es nach der ersten Regel gesendet. Wenn das Paket von 1.2.3.4/32 kommt, wird es von der zweiten Regel blockiert. Du kannst ähnliche Regeln für die unroutebaren Adressen festlegen. Wenn einige Maschinen versuchen sollten, ein Paket durch IPF mit dem Ziel 192.168.0.0/16 zu routen, warum blockierst Du das nicht? Das Schlimmste, das passieren kann, ist, daß Du etwas Bandbreite einbüßt:

```
block out quick on tun0 from any to 192.168.0.0/16
block out quick on tun0 from any to 172.16.0.0/12
block out quick on tun0 from any to 10.0.0.0/8
block out quick on tun0 from any to 0.0.0.0/8
block out quick on tun0 from any to 127.0.0.0/8
block out quick on tun0 from any to 169.254.0.0/16
block out quick on tun0 from any to 192.0.2.0/24
block out quick on tun0 from any to 204.152.64.0/23
block out quick on tun0 from any to 224.0.0.0/3
block out quick on tun0 from !20.20.20.0/24 to any
```

Auf den ersten Blick verbessert das Deine Sicherheit nicht. Aber es verbessert die Sicherheit von vielen anderen. Es ist daher eine gute Idee, das zu machen. Als anderen Gesichtspunkt kann man das unterstützen, um so zu verhindern, daß niemand gespoofte Pakete aus Deinem Netzwerk versenden kann, daß Deine Site nicht als Relay für solche Angriffe verwendet werden kann und als solcher auch kein Ziel darstellen kann. Du wirst noch eine gewisse Anzahl von Anwendungen für das Blockieren von ausgehenden Paketen finden. Eines sollte man dabei noch im Gedächtnis behalten: Daß alle ein- und ausgehenden Richtungen immer in Relation zur Firewall stehen, niemals aber zu anderen Maschinen.

2.8. Das Geschehen protokollieren; das "log"-Schlüsselwort

Bis zu diesem Punkt wurden alle blockierten oder freigegebenen Pakete entweder stillschweigend blockiert oder ebenso stillschweigend durchgelassen. Im Normalfall willst Du aber wissen, ob Du angegriffen wurdest, anstatt ziemlich verwundert darüber zu sein, wenn die Firewall auf einmal irgendwelche Sachen einkauft. Während ich nicht alle Pakete aufzeichnen will, die die Firewall passieren, würde ich gerne mehr über die blockierten Pakete von 20.20.20.0/24 wissen. Um das zu tun, ergänzen wir die Regel um das "log"-Schlüsselwort.

```
block in     quick on tun0 from 192.168.0.0/16 to any
block in     quick on tun0 from 172.16.0.0/12 to any
block in     quick on tun0 from 10.0.0.0/8 to any
block in     quick on tun0 from 127.0.0.0/8 to any
block in     quick on tun0 from 0.0.0.0/8 to any
block in     quick on tun0 from 169.254.0.0/16 to any
block in     quick on tun0 from 192.0.2.0/24 to any
block in     quick on tun0 from 204.152.64.0/23 to any
block in     quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
pass  in       all
```

So weit ist unsere Firewall schon mal ganz gut gerüstet, um Pakete von suspekten Orten zu blockieren. Auf der einen Seite akzeptieren wir ausgehende Pakete, die nach überall hingehen sollen. Eine Arbeit, die wir uns noch machen sollten, ist, sicher zu gehen, daß Pakete nach 20.20.20.0/32 und 20.20.20.255/32 verworfen werden. Das anders zu machen, öffnet das interne Netz für eine Smurf-Attacke. Diese beiden Zeilen schützen das hypothetische Netzwerk vor dem Mißbrauch als Smurf-Relay.

```
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
```

Das läßt unseren Regelsatz in etwa so aussehen:

```
block in     quick on tun0 from 192.168.0.0/16 to any
block in     quick on tun0 from 172.16.0.0/12 to any
block in     quick on tun0 from 10.0.0.0/8 to any
block in     quick on tun0 from 127.0.0.0/8 to any
block in     quick on tun0 from 0.0.0.0/8 to any
block in     quick on tun0 from 169.254.0.0/16 to any
block in     quick on tun0 from 192.0.2.0/24 to any
block in     quick on tun0 from 204.152.64.0/23 to any
block in     quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass  in       all
```

2.9. Komplettes bidirektionales Filtern an den Interfaces

Bisher haben wir nur Fragmente eines kompletten Regelsatzes gesehen. Wenn Du Deinen eigenen Regelsatz kreierst, mußt Du Regeln für jede Richtung und jedes Interface festlegen. Der Defaultstatus von IPFilter ist es, alles durchzulassen. Es ist, als ob es eine unsichtbare Regel gibt, die alle Pakete, also alle ein- und ausgehenden durchläßt. Anstatt sich auf irgendein Standardverhalten zu verlassen, sollte alles so spezifisch wie möglich gemacht werden; Interface für Interface, bis alles verpackt ist. Wir starten zuerst mit dem lo0-Interface, das wild und frei laufen will. Seit es Programme gibt, die mit anderen auf dem System kommunizieren, gehe einfach weiter und lasse es offen:

```
pass out quick on lo0
pass in  quick on lo0
```

Das Nächste ist das x10-Interface. Wir beginnen später damit, Regeln für dieses Interface festzulegen. Für den Moment gehen wir davon aus, daß alles aus Deinem lokalen Netz vertrauenswürdig ist und lassen es genau so laufen wie lo0.

```
pass out quick on xl0
pass in  quick on xl0
```

Zum Schluß ist da noch das tun0-Interface, das wir bis jetzt nur zur Hälfte filtern:

```
block out quick on tun0 from any to 192.168.0.0/16
block out quick on tun0 from any to 172.16.0.0/12
block out quick on tun0 from any to 127.0.0.0/8
block out quick on tun0 from any to 10.0.0.0/8
block out quick on tun0 from any to 0.0.0.0/8
block out quick on tun0 from any to 169.254.0.0/16
block out quick on tun0 from any to 192.0.2.0/24
block out quick on tun0 from any to 204.152.64.0/23
block out quick on tun0 from any to 224.0.0.0/3
pass  out quick on tun0 from 20.20.20.0/24 to any
block out quick on tun0 from any to any

block in     quick on tun0 from 192.168.0.0/16 to any
block in     quick on tun0 from 172.16.0.0/12 to any
block in     quick on tun0 from 10.0.0.0/8 to any
block in     quick on tun0 from 127.0.0.0/8 to any
block in     quick on tun0 from 0.0.0.0/8 to any
block in     quick on tun0 from 169.254.0.0/16 to any
block in     quick on tun0 from 192.0.2.0/24 to any
block in     quick on tun0 from 204.152.64.0/23 to any
block in     quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
```

```
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass  in     all
```

Das ist schon eine signifikante Menge an Regeln, um 20.20.20.0/24 von Spoofing-Angriffen oder der Nutzung als Relay für solche Angriffe zu schützen. Künftige Beispiele werden weiterhin diese Einseitigkeit zeigen, aber behalte im Gedächtnis, daß wir das um der Kürze willen so machen. Wenn Du Deinen eigenen Regelsatz aufsetzen willst, ergänze das bitte um Regeln für jedes Interface und jede Richtung, sofern das nötig ist.

2.10. Die Kontrolle von bestimmten Protokollen; Das "proto"-Schlüsselwort

Denial-of-Service-Angriffe grassieren genauso im Internet wie Buffer-Overflow-Angriffe. Viele Denial-of-Service-Angriffe verlassen sich auf Fehler im TCP/IP-Stack verschiedener Betriebssysteme. Manchmal kommen sie auch als ICMP-Pakete. Warum sollte man sowas nicht gleich ganz blockieren?

```
block in log quick on tun0 proto icmp from any to any
```

Jetzt wird jeder eingehende ICMP-Verkehr aufgezeichnet und anschließend verworfen.

2.11. ICMP Filtern mit dem "icmp-type"-Schlüsselwort; Regelsätze

Natürlich ist das Fallenlassen von allen ICMP-Paketen nicht die Ideallösung. Warum eigentlich nicht? Nun, es ist sinnvoll, das partiell zu erlauben. Es könnte sein, daß Du manchen ICMP-Verkehr behalten willst und andere Arten verworfen werden sollen. Wenn Du pingen und tracerouten willst, brauchst Du die ICMP-Typen 10 und 11. Im Klartext: Das MUß keine gute Idee sein, aber Du mußt zwischen Sicherheit und Komfort abwägen. IPF hilft Dir dabei:

```
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
```

Denke dran, daß die Reihenfolge des Regelsatzes wichtig ist. Seit wir alles "quick" machen, müssen wir unsere "passes" vor die "blocks" setzen. Ergo setzen wir die letzten drei Regeln in dieser Reihenfolge:

```
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
```

Das Anhängen dieser drei Regeln an die Anti-Spoofing-Regeln ist etwas haarig. Ein Fehler könnte es sein, die neuen ICMP-Regeln an den Anfang zu setzen.

```
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
block in     quick on tun0 from 192.168.0.0/16 to any
block in     quick on tun0 from 172.16.0.0/12 to any
block in     quick on tun0 from 10.0.0.0/8 to any
block in     quick on tun0 from 127.0.0.0/8 to any
block in     quick on tun0 from 0.0.0.0/8 to any
block in     quick on tun0 from 169.254.0.0/16 to any
block in     quick on tun0 from 192.0.2.0/24 to any
block in     quick on tun0 from 204.152.64.0/23 to any
block in     quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass  in       all
```

Das Problem dabei ist, daß ein ICMP-Paket des Typs 0 von 192.168.0.0/16 die erste Regel passieren wird und von der vierten Regel niemals blockiert wird.

Also: Seit wir ICMP ECHO_REPLYS passieren lassen öffnen wir uns eine Hintertür für diese ekligen Smurf-Angriffe.

Also erklären wir die letzten zwei Blockaderegeln für nichtig. Um dem aus dem Wege zu gehen, setzen wir die ICMP-Regeln hinter die Anti-Spoofing-Regeln.

```
block in     quick on tun0 from 192.168.0.0/16 to any
block in     quick on tun0 from 172.16.0.0/12 to any
block in     quick on tun0 from 10.0.0.0/8 to any
block in     quick on tun0 from 127.0.0.0/8 to any
block in     quick on tun0 from 0.0.0.0/8 to any
block in     quick on tun0 from 169.254.0.0/16 to any
block in     quick on tun0 from 192.0.2.0/24 to any
block in     quick on tun0 from 204.152.64.0/23 to any
block in     quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
pass  in       all
```

Weil wir gespooften Verkehr blockieren, bevor die ICMP-Regeln ausgeführt werden, schafft es ein gespooftes Paket niemals bis zum ICMP-Regelsatz. Es ist sehr wichtig, solche Situationen im Hinterkopf zu behalten, wenn man Regeln miteinander verschmilzt.

2.12. TCP und UDP Ports; Das "port"-Schlüsselwort

Jetzt haben wir damit begonnen, die Pakete auf ihren Protokollen basierend abzublocken. Wir können die Pakete nach spezifischen Aspekten für jedes Protokoll abblocken. Der am meisten genutzte Aspekt ist die Portnummer. Services wie rsh, rlogin und telnet sind sehr bequem, wenn man sie installiert hat, aber sie verbergen auch gewisse Unsicherheiten gegen Network- Sniffing- und Spoofing-Angriffe. Ein möglicher Kompromiß ist es, diese Dienste nur intern laufen zu lassen und sie nach außen hin zu blockieren. Das ist einfach zu machen, weil diese Dienste immer auf definierten TCP-Ports laufen. Als solches ist das Schaffen von Regeln, mit denen sie blockiert werden können, einfach.

```
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 513
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 514
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 23
```

Stelle sicher, daß alle drei vor dem Pass in All stehen. Dann sind sie für die Außenseite dicht.

```
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass  in     quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 513
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 514
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 23
pass  in     all
```

Du kannst außerdem 514/udp (syslog), 111/tcp und 111/udp (portmap), 515/tcp (lpd), 2049/tcp und 2049/udp (NFS), 6000/tcp (X11) und so weiter und so fort blockieren. Eine komplette Liste mit den Ports bekommst Du mit netstat -a (oder lsof -i, wenn das installiert ist). Wenn anstelle des TCP-Protokolls UDP-Pakete blockiert werden sollen, braucht man das "proto tcp"-Kommando nur durch "proto udp" zu ersetzen. Die Regel für Syslog sieht dann so aus:

```
block in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 514
```

IPF bietet außerdem einen schnellen Weg, um Regeln zu schreiben, die TCP und UDP gleichzeitig blockieren. Portmap und NFS benutzen beide Protokolle. Die Regel für Portmap sieht so aus:

```
block in log quick on tun0 proto tcp/udp from any to 20.20.20.0/24 port = 111
```

3. Advanced Firewalling Introduction

(Anmerkung des Übersetzers: Das sollte besser in Englisch bleiben)

Dieses Kapitel wurde als eine Art "Follow-up" zur Basissektion geschrieben. Der Text unten beinhaltet sowohl Konzepte für ein erweitertes Firewall-Design als auch erweiterte Funktionen, die nur IPFilter in dieser Form bietet. Wenn Du einmal mit dieser Sektion vertraut bist, solltest Du in der Lage sein, eine wirklich gute Firewall zu bauen.

3.1. Die wuchernde Paranoia; oder die "Default-Deny-Einstellung"

Es kann zu einem große Problem auswachsen, Dienste nach Ports zu blockieren, weil sie sich manchmal verändern. RPC-basierte Programme (lockd, statd, nfs) sind schrecklich gut darin; besonders NFS horcht gerne auch an anderen Ports als 2049. Es ist wirklich schwierig, etwas vorauszusagen und es ist noch viel schwieriger, diese Einstellungen immer automatisch zu korrigieren. Was ist, wenn Du einen Dienst vermißt? Statt mit dieser ganzen Hirselei zu dealen, laß uns noch mal mit einer sauberen Konfiguration neu starten. Der derzeitige Regelsatz sieht so aus:

Jaja, wir starten wirklich noch einmal neu. Die erste Regel, die wir nutzen, ist diese:

```
block in all
```

Kein Netzwerkverkehr geht durch. Keiner. Nicht ein Piep. Du bist wirklich sicher mit dieser Einstellung. Das ist nicht wirklich zu gebrauchen, aber eben auch absolut sicher. Die eigentlich große Sache in dieser Angelegenheit ist, daß es allerdings nicht viel mehr braucht, um Deine Kiste sowohl sicher als auch benutzbar zu machen.

Sagen wir der Maschine, daß auf ihr ein Webserver läuft; nicht mehr, aber auch nicht weniger. Sie macht keine DNS-Lookups. Sie will nur Verbindungen über Port 80/tcp aufnehmen und das war's dann. Wir können das so machen. Und zwar mit einer zweiten Regel; Du weißt schon wie:

```
block in       on tun0 all
pass  in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 80
```

Diese Maschine läßt allen Traffic für 20.20.20.1 auf Port 80 durch. Für eine einfache Firewall reicht das.

3.2. Bedingungsloses "Allow"; die "keep state"(Status halten)-Regel

Der Job Deiner Firewall ist es, unerwünschten Traffic von Punkt A nach Punkt B zu unterbinden. Wir haben allgemeingültige Regeln, die dem Computer sagen "So lange dieses Paket für Port 23 ist, ist das in Ordnung." Wir haben allgemeine Regeln, die sagen "Solange dieses Paket den FIN-Flag gesetzt hat, ist das in Ordnung." Unsere Firewall kennen den Anfang, den Mittelteil oder das Ende einer TCP/UDP/ICMP-Sitzung nicht. Sie besitzt nur vage Regeln, die für alle Pakete angewendet werden. Wir gehen in der Hoffnung, daß das Paket mit einem gesetzten FIN-Flag auch wirklich ein FIN-Scan ist, der zu unseren Diensten paßt. Wir hoffen, daß das Paket an Port 23 kein unerwünschter "Tramper" in unserer Telnet-Sitzung ist. Was ist aber, wenn da ein Weg ist, individuelle TCP/UDP/ICMP-Sitzungen zu identifizieren und zu autorisieren oder sie von Portscannern oder Denial-of-Service-Attacken erkannt werden können? Es gibt da einen Weg. Er wird "keeping state" genannt.

Wir wollen Komfort und Sicherheit auf einmal. Viele Leute wollen das. Das ist auch der Grund, warum Cisco eine "established"-Klausel hat, die "etablierte" oder besser, erwünschte TCP-Sitzungen zuläßt. IPFW hat diese Funktion. IPFWADM hat Setup/established. Sie alle haben dieses Feature, aber der Name ist oft irreführend. Als wir das zu ersten Mal sahen, dachten wir, daß unserem Paketfilter bewußt war, was da passiert, daß er wußte, ob eine Verbindung wirklich "established" war oder eben nicht. Fakt ist, daß sie alle das "Wort" dafür aus einem Teil des Paketes entnehmen, in dem jeder lügen kann. Sie lesen die Flag-Sektion des TCP-Paketes. Das ist der Grund, warum UDP/ICMP damit nicht funktioniert. Diese Pakete haben das schlicht nicht. Jeder der ein Paket mit "Bogus"-Flag erzeugen kann, kann mit diesem Setup von einer Firewall akzeptiert werden. Wo kommt IPF denn nun ins Spiel, fragst Du Dich? Nun, im Gegensatz zu anderen Firewalls kann IPF herausfinden, ob ein Paket "established" ist oder eben nicht. Und es macht das mit TCP, UDP und ICMP, nicht nur mit TCP. Bei IPF heißt das "keeping state", also Status oder Zustand halten. Das Schlüsselwort dafür ist "keep state". Bis jetzt

redeten wir nur über Pakete, die hereinkommen; dann wir der Regelsatz geprüft; Pakete gehen raus; der Regelsatz wird geprüft. Derzeit passiert mit den Paketen dieses hier: Pakete kommen herein und werden vielleicht geprüft. Pakete verlassen den Rechner und werden vielleicht geprüft. Was also derzeit passiert, ist, daß die Pakete entweder nach dem Regelsatz für eingehende oder nach dem Regelsatz für ausgehende Pakete geprüft werden.

Die Statustabelle ist eine Liste von TCP/UDP/ICMP-Sitzungen, die ungefragt die Firewall passieren dürfen, eingeschränkt durch den gesamten Regelsatz. Das klingt für Dich nach einem ernsthaften Sicherheitsloch? Bleib dran; es ist das Beste, was Deiner Firewall passieren kann.

Alle TCP/IP-Sitzungen sehen aus wie ein Schulausatz: Es gibt einen Anfang, einen Hauptteil und ein Ende (besonders, wenn sich alles im selben Paket befindet). Du kannst keinen Hauptteil ohne einen Anfang haben und kein Ende ohne Hauptteil. Das bedeutet, daß alles, was Du wirklich filtern mußt, sich eigentlich auf den Anfang der TCP/UDP/ICMP-Sitzung beschränkt. Wenn der Anfang einer Sitzung von Deiner Firewall zugelassen wird, willst Du wirklich auch den Hauptteil und das Ende davon (Dein IP-Stack sollte aber nicht überflutet werden, damit der Rechner nicht unbrauchbar wird). Den Status zu halten, erlaubt es Dir, die Pakete in der Mitte einer Sitzung und an ihrem Ende zu ignorieren und einfach nur neue Sitzungen zu blockieren oder durchzulassen. Wenn die neue Sitzung die Firewall passieren durfte, werden alle dazugehörigen Pakete ebenfalls durchgelassen. Wenn diese Sitzung blockiert wurde, werden auch alle nachfolgenden Pakete dieser Sitzung nicht durchgelassen. Hier ist ein Beispiel für einen SSL-Server (und nichts anderes als ein SSL-Server):

```
block out quick on tun0 all
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 22 keep state
```

Das Erste, was Du Dir merken solltest, daß es keine "pass-out"-Vorkehrung gibt. Abgesehen davon, ist der Regelsatz komplett. Weil wir den Status halten, wird der gesamte Regelsatz nicht noch einmal abgefragt. Wenn das erste SYN-Paket den SSH-Server erreicht hat, wird der Status erstellt. Die Sitzung behält diesen, ohne daß sich die Firewall einmischt. Hier ist ein anderes Beispiel:

```
block in  quick on tun0 all
pass  out quick on tun0 proto tcp from 20.20.20.1/32 to any keep state
```

In diesem Fall laufen auf dem Server keine Dienste. In Wirklichkeit ist das kein Server, sondern ein Client. Und dieser Client will nicht, daß unauthorisierte Pakete in den IP-Stack der Maschine eindringen. Aber der Client will trotzdem vollen Zugriff auf das Internet und die Antwortpakete, die solche Privilegien enthalten. Dieser einfache Regelsatz erstellt Statuseintragungen für jede neue ausgehende TCP-Sitzung. Zu Wiederholung: Wenn ein Statuseintrag stattgefunden hat, ist es den neuen TCP-Sessions freigestellt, vor- und zurück zu "reden" wie sie es wünschen, und zwar ohne Behinderung oder Kontrolle durch den Regelsatz der Firewall. Wir weisen nochmals darauf hin, daß das gleichermaßen für UDP und ICMP gilt:

```
block in  quick on tun0 all
pass  out quick on tun0 proto tcp  from 20.20.20.1/32 to any keep state
pass  out quick on tun0 proto udp  from 20.20.20.1/32 to any keep state
pass  out quick on tun0 proto icmp from 20.20.20.1/32 to any keep state
```

Jou, Ostfriesland; wir können pingen. Nun können wir "keep state" für TCP, UDP und ICMP durchführen. Jetzt können wir ausgehende Verbindungen genauso herstellen, als ob es keine Firewall gäbe. Und Möchtegern-Hacker können sich nicht in das System einklinken. Das ist sehr praktisch, weil es keinen Bedarf gibt, aufzuzeichnen, auf welchen Ports gehorcht wird. Nur auf die Ports, die wir freigeben, kann von anderen Leuten zugegriffen werden.

State ist schön praktisch, aber auch etwas kitzlig. Du kannst Dir damit selbst auf mysteriösen und befremdlichen Wegen auch selbst in den Fuß schießen. Vergleiche mal den folgenden Regelsatz:

```
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23
pass  out quick on tun0 proto tcp from any to any keep state
block in  quick all
block out quick all
```

Auf der ersten Blick scheint das eine gute Konfiguration zu sein. Wir erlauben eingehende Sitzungen an Port 23 und ausgehende Sitzungen nach überall. Natürlich werden Pakete, die nach Port 23 gehen, ein Antwortpaket bekommen. Aber ein Regelsatz, der so aufgesetzt wird, erzeugt einen Zustandseintrag und alles wird richtig funktionieren. Das ist das Wenigste, denkst Du.

Die unsägliche Wahrheit ist, daß nach 60 Sekunden Idletime der Eintrag geschlossen wird (Im Gegensatz zu den üblichen fünf Tagen). Das liegt daran, daß der "State-Tracker" niemals das Original-SYN-Paket sieht; es sieht nur das SYN-ACK. IPF kann TCP-Sitzungen von Anfang bis Ende sehr gut verfolgen, aber ist nicht so gut darin, in die Mitte der Sitzung zu kommen. Also mußt Du Deine Regel so ändern wie unten beschrieben:

```
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 keep state
pass  out quick on tun0 proto tcp from any to any keep state
block in  quick all
block out quick all
```

Der Zusatz zu dieser Regel trägt das allererste Paket in die Zustandstabelle ein; und alles wird funktionieren wie gedacht. Wenn der 3-Wege-Handshake einmal von der "Zustandsmaschine" (state-engine) bezeugt wurde, ist er im 4/4-Modus markiert; was bedeutet, daß seine Einstellungen für einen langfristigen Datenaustausch gelten, bis die Verbindung beendet wird (Wobei der Modus sich nochmals ändert. Man kann das mit ipfstat -s abfragen, was natürlich für die gesamte Zustandstabelle gilt).

3.3. Zustände mit UDP

Für UDP gibt es solche Zustände nicht; es ist natürlich etwas schwerer, zuverlässig einen Zustand zu definieren und diesen zu halten. Abgesehen davon leistet IPFilter auch hier gute Dienste. Wenn Maschine A mit Quellport X ein UDP-Paket an Maschine B mit Zielport Y schickt, wird ipf eine Antwort von Maschine B, Port Y an Maschine A, zulassen. Das ist ein Kurzzeitzustand, der nur 60 Sekunden besteht.

Hier ist ein Beispiel für das, was passiert, wenn wir einen nslookup ausführen, um herauszufinden, welche IP-Adresse die Firma 3com hat:

```
$ nslookup www.3com.com
```

Ein DNS-Paket wird erzeugt:

```
17:54:25.499852 20.20.20.1.2111 > 198.41.0.5.53: 51979
```

Das Paket kommt von 20.20.20.1, Port 2111; als Ziel ist 198.41.0.5, Port 53 angegeben. Ein einminütiger Statuseintrag wird erzeugt. Wenn das Paket von 198.41.0.5, Port 53, mit dem Ziel 20.20.20.1, Port 2111 zurückkommt, wird das Antwortpaket die Firewall passieren. Wie Du hier sehen kannst, geschieht das bereits Millisekunden später:

```
17:54:25.501209 198.41.0.5.53 > 20.20.20.1.2111: 51979 q: www.3com.com
```

Das Antwortpaket paßt auf die Statuskriterien und wird durchgelassen. Im gleichen Moment, in dem das Paket durchgelassen wurde, wird dieser Weg wieder verschlossen. Neue eingehende Pakete dürfen die Firewall nicht mehr passieren; auch und besonders, wenn sie vorgeben, vom selben Platz zu stammen.

3.4. ICMP mit Status

IPFilter behandelt ICMP-Zustände in der Weise, in der jemand versteht, wie ICMP mit TCP und UDP benutzt wird und wie das "Keep State"-Feature funktioniert. Es gibt zwei grundsätzliche Arten von ICMP-Nachrichten: Anfragen und Antworten. Wenn Du eine Regel schreibst wie diese hier:

```
pass out on tun0 proto icmp from any to any icmp-type 8 keep state
```

um Echo Requests (ein typischer Ping) zu erlauben, wird das resultierende ICMP-Typ 0-Paket zugelassen werden. Dieser Statuseintrag besitzt einen Default-Timeout eines unvollständigen 0/0-Status von 60 Sekunden. Das bedeutet: Wenn Du den Status irgendwelcher icmp-Pakete herausfinden willst, brauchst Du ein ICMP-Protokoll [...] und eine "keep state"-Regel.

Trotzdem, die häufigste Anwendung von ICMP-Nachrichten sind Statusinformationen, die von einigen Fehlern im UDP-Protokoll (Manchmal auch in TCP) und in 3.4x und neueren IPFiltern erzeugt werden (Beispiel: ICMP-Typ 3 Code 3 "port unreachable" oder auch ein "time exceeded"), die für einen aktiven Statuseintrag passen. Es

kann aber auch sein, daß ein aktiver Statuseintrag diese Meldung generiert hat. Beispiel: Wenn Du einen älteren IPFilter benutzt, mußt Du für Traceroute folgendes eintragen:

```
pass out on tun0 proto udp from any to any port 33434><33690 keep state
pass in on tun0 proto icmp from any to any icmp-type timex
```

Du kannst jetzt die richtigen Dinge tun und den Status für UDP so halten:

```
pass out on tun0 proto udp from any to any port 33434><33690 keep state
```

Um einen gewissen Schutz gegen das Einschleichen von fremden ICMP-Paketen in einer aktiven Verbindung zu erreichen, wird das Paket nicht nur auf Herkunft und Ziel geprüft (nebst der Ports, sofern gefragt), wofür ein kleiner Teil des Paketes beansprucht wird.

3.5. FIN Scan Detection; "flags" Schlüsselwort, "keep frags" Schlüsselwort

Laßt uns mal zurückgehen zum Vierer-Regelsatz aus dem vorherigen Kapitel:

```
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 keep state
pass  out quick on tun0 proto tcp from any to any keep state
block in  quick all
block out quick all
```

Das ist meistens, aber nicht insgesamt, zufriedenstellend. Das Problem dabei ist, daß nicht nur SYN-Pakete, die das dürfen, an Port 23 gehen, sondern daß jedes ältere Paket durchkommen kann. Wir können das mit der "flags"-Option abstellen:

```
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 flags S keep state
pass  out quick on tun0 proto tcp from any to any flags S keep state
block in  quick all
block out quick all
```

Jetzt werden nur TCP-Pakete, die an 20.20.20.1, Port 23, mit einem einzelnen SYN-Flag hereingelassen und in die Statustabelle eingetragen. Ein einzelner SYN-Flag ist nur im allerersten Paket vorhanden (der sog. TCP-Handshake); und das ist es, was wir eigentlich wollen. Diese Sache hat zwei Vorteile: Kein unerwünschtes Paket kann hereinkommen und die Statustabelle durcheinanderbringen. Gleichzeitig werden FIN- und XMAS-Scans scheitern, wenn sie andere Flags setzen als das SYN-Flag. Nun müssen alle Pakete einen Handshake durchführen oder bereits einen Status in der Tabelle besitzen. Wenn möglicherweise etwas anderes hereinkommt, ist es vielleicht ein Portscan oder ein "geschmiedetes" (gefährliches) Paket. Eine Ausnahme gibt es allerdings: Ein eingehendes Paket, das auf seiner Reise fragmentiert wurde.

IPF bietet auch hier etwas an, das "keep frags"-Schlüsselwort. Damit wird IPF fragmentierte Pakete bemerken und die Spur dieser Pakete halten, indem es die erwarteten Fragmente durchläßt. Wir schreiben die drei Regeln, mit denen wir die "geschmiedeten" Pakete aufzeichnen und fragmentierte Pakete zulassen:

```
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 \
   flags S keep state keep frags
pass  out quick on tun0 proto tcp from any to any keep state flags S keep frags
block in  log quick all
block out log quick all
```

Das funktioniert, weil jedes Paket, das durchgelassen werden soll, seine Weg durch die Statustabelle macht, bevor die Blockierregeln erreicht werden. Wenn Du wirklich darüber besorgt bist, kannst Du besonders die initialen SYN-Pakete aufzeichnen.

3.6. Ein blockiertes Paket beantworten

Bis jetzt haben wir alle abgelehnten Pakete auf den Boden fallen lassen; Protokolliert oder auch nicht. Wir haben nie etwas an den Absender zurückgeschickt. Manchmal ist das aber nicht das beste Verfahren. Wir erzählen einem Angreifer nämlich, daß da ein Paketfilter seinen Dienst versieht. Es scheint weitaus besser zu sein, daß man den Angreifer so irreführt, daß dieser glaubt, daß ein Paketfilter seinen Dienst tut, obwohl gar keiner da ist.

Das ist genauso, als wenn keine Dienste vorhanden sind, durch die man einbrechen kann. Das ist der Bereich, in dem das Vorspiegeln einer Firewall ins Spiel kommt.

Wenn ein Service auf einem Unixsystem nicht läuft, läßt das System den anderen Host das normalerweise mit einigen Arten von Return-Paketen wissen. In TCP wird das mit einem RST (Reset)-Paket gemacht. Wenn ein TCP-Paket blockiert wird, kann IPF zur Zeit ein RST an den Absender schicken, indem es das "return-rst"-Schlüsselwort verwendet. Wo wir zuerst das getan haben:

```
block in log on tun0 proto tcp from any to 20.20.20.0/24 port = 23
pass  in     all
```

...sollten wir jetzt dies hier machen:

```
block return-rst in log from any to 20.20.20.0/24 proto tcp port = 23
block in log quick on tun0
pass  in     all
```

Wir brauchen zwei "Block"-Statements, seit Return-RST nur mit TCP funktioniert und wir wollen Protokolle wie TCP, ICMP und andere abblocken. Das ist jetzt getan. Die andere Seite bekommt jetzt ein "Connection refused" anstelle eines "connection timed out"

Es ist genauso möglich, eine Fehlermeldung zu senden, wenn jemand ein UDP-Paket an Dein System schickt. Wobei Du dieses hier benutzten solltest:

```
block in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

Du kannst statt dessen das "return-icmp"-Schlüsselwort verwenden, um eine Antwort zu schicken:

```
block return-icmp(port-unr) in log quick on tun0 proto udp \
   from any to 20.20.20.0/24 port = 111
```

Dem illustren TCP/IP entsprechend ist "port-unreachable" die korrekte ICMP-Antwort, wenn kein Dienst auf dem abgefragten Port auf Anfragen wartet. Du kannst jeden ICMP-Typen benutzen, den Du magst, aber "port-unreachable" ist vermutlich die beste Wahl. Es ist auch der Standard-ICMP-Typ für "Return-icmp".

Aber: Wenn Du "return-icmp" verwendest, sollte Dir bewußt sein, daß das nicht gerade unsichtbar ist. Es schickt das ICMP-Paket mit der Adresse der Firewall zurück, nicht etwa mit der IP-Adresse des eigentlichen Ziels. Das wurde in IPFilter 3.3 abgestellt. Außerdem wurde ein neues Schlüsselwort, "return-icmp-as-dest", hinzugefügt. Das neue Format ist das hier:

```
block return-icmp-as-dest(port-unr) in log on tun0 proto udp \
   from any to 20.20.20.0/24 port = 111
```

3.7. Beliebte Logging-Techniken

Es ist wichtig zu wissen, daß das Vorhandensein des "log"-Schlüsselwortes nur sicherstellt, daß das Paket für das Logging-device (/dev/ipl) verfügbar ist. Um diese Log-Information sehen zu können, muß das Ipmon-Utility laufen (oder ein anderes, daß von /dev/ipl lesen kann). Typischerweise wird Log in Verbindung mit "ipmon -s" verwendet, um die Informationen an Syslog weiterzureichen. Seit Ipfilter-3.3 kann eine Regel das Aufzeichnungsverhalten von Syslog durch die Nutzung des "log level"-Schlüsselwortes geregelt werden. Beispielsweise in Regeln wie dieser:

```
block in log level auth.info quick on tun0 from 20.20.20.0/24 to any
block in log level auth.alert quick on tun0 proto tcp from any to 20.20.20.0/24 port = 21
```

Zusätzlich kannst Du Dir zurechtstricken, welche Informationen aufgezeichnet werden sollen. Beispiel: Es könnte sein, daß Du nicht daran interessiert bist, daß jemand Deinen Telnet-Port 500mal testet; aber Du wirst durchaus daran interessiert sein, wenn es jemand geschafft hat, Dich oder Dein System zu untersuchen. Du kannst das "log first"-Schlüsselwort benutzen, um nur das erste Exemplar eines Paketsatzes aufzuzeichnen. Natürlich trifft diese Idee nur auf Pakete einer bestimmten Sitzung zu. Und für das typische abgelehnte Paket wirst Du unter den Druck geraten, eine Situation nachzuvollziehen, wo dieses Paket tut, was Du erwartest. Dennoch, wenn das in Zusammenarbeit mit "pass" und "keep state" benutzt wird, kann das ein wertvolles Schlüsselwort sein, um bestimmte Tabellen im Verkehr zu behalten. Eine andere nützliche Sache, die Du damit

machen kannst, ist es, interessante Teile eines Paketes zusätzlich zu den Header-Informationen aufzuzeichnen, die normalerweise aufgezeichnet werden. IPFilter gibt Dir die ersten 128 Bytes des Paketes, wenn Du das "body"-Schlüsselwort benutzt.

Du solltest den Gebrauch des "body-Logging" aber beschränken, weil es die Logdaten sehr ausführlich und umfangreich macht. Aber für bestimmte Applikationen ist es oft von Nutzen, wenn man in der Lage ist, zurückzugehen und einen Blick auf das Paket zu werfen oder diese Daten an eine andere Anwendung zu schicken, die die Sachen weiterprüft.

3.8. Der Zusammenbau

So, jetzt haben wir eine schöne feste Firewall. Aber sie kann noch fester werden. Einiges aus dem originalen Regelsatz das wir weggewischt haben, ist zu Zeit immer noch sehr brauchbar. Ich denke, man sollte das Anti-Spoofing-Zeugs wieder einbauen. Das sieht dann so aus:

```
block in           on tun0
block in     quick on tun0 from 192.168.0.0/16 to any
block in     quick on tun0 from 172.16.0.0/12 to any
block in     quick on tun0 from 10.0.0.0/8 to any
block in     quick on tun0 from 127.0.0.0/8 to any
block in     quick on tun0 from 0.0.0.0/8 to any
block in     quick on tun0 from 169.254.0.0/16 to any
block in     quick on tun0 from 192.0.2.0/24 to any
block in     quick on tun0 from 204.152.64.0/23 to any
block in     quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass  out quick on tun0 proto tcp/udp from 20.20.20.1/32 to any keep state
pass  out quick on tun0 proto icmp    from 20.20.20.1/32 to any keep state
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 80 flags S keep state
```

3.9. Leistungssteigerung durch Regelgruppen

Laß uns die Möglichkeiten unserer Firewall erweitern, indem wir sie viel komplizierter machen; und hoffen, daß sie eher dem wirklichen Leben entgegenkommt. In diesem Beispiel ändern wir die Interfacenamen und die Netzwerknummern. Laß uns annehmen, wir haben drei Interfaces in unserer Firewall, x10, x11 und x12.

xl0 ist mit unserem externen Netzwerk verbunden: 20.20.20.0/26
x11 ist mit unserer DMZ verbunden, Netzwerk 20.20.20.64/26
x12 ist mit unserem geschützten Netz verbunden: 20.20.20.128/25

Wir werden den gesamten Regelsatz in einem Rutsch definieren, seit wir wissen, daß Du diese Regeln jetzt lesen kannst:

```
block in     quick on xl0 from 192.168.0.0/16 to any
block in     quick on xl0 from 172.16.0.0/12 to any
block in     quick on xl0 from 10.0.0.0/8 to any
block in     quick on xl0 from 127.0.0.0/8 to any
block in     quick on xl0 from 0.0.0.0/8 to any
block in     quick on xl0 from 169.254.0.0/16 to any
block in     quick on xl0 from 192.0.2.0/24 to any
block in     quick on xl0 from 204.152.64.0/23 to any
block in     quick on xl0 from 224.0.0.0/3 to any
block in log quick on xl0 from 20.20.20.0/24 to any
block in log quick on xl0 from any to 20.20.20.0/32
block in log quick on xl0 from any to 20.20.20.63/32
block in log quick on xl0 from any to 20.20.20.64/32
block in log quick on xl0 from any to 20.20.20.127/32
block in log quick on xl0 from any to 20.20.20.128/32
block in log quick on xl0 from any to 20.20.20.255/32
pass out on xl0 all

pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state
pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 21 flags S keep state
pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 20 flags S keep state
pass out quick on xl1 proto tcp from any to 20.20.20.65/32 port = 53 flags S keep state
```

```
pass out quick on xl1 proto udp from any to 20.20.20.65/32 port = 53 keep state
pass out quick on xl1 proto tcp from any to 20.20.20.66/32 port = 53 flags S keep state
pass out quick on xl1 proto udp from any to 20.20.20.66/32 port = 53 keep state
block out on xl1 all
pass in quick on xl1 proto tcp/udp from 20.20.20.64/26 to any keep state

block out on xl2 all
pass in quick on xl2 proto tcp/udp from 20.20.20.128/25 to any keep state
```

Von diesem eigensinnigen Beispiel ausgehend, sehen wir, daß unser Regelsatz weniger einflußreich geworden ist. Um die Situation nicht noch weiter zu verschlechtern, indem wir besondere Regeln für unsere DMZ hinzufügen, ergänzen wir unsere Firewall durch weiter Tests für jedes Paket, das die Performance der x10 <--> x12-Verbindungen beeinflußt. Wenn du eine Firewall mit einem Regelsatz wie diesem aufsetzt und Du eine Menge Bandbreite mit hinreichender CPU-Leistung hast, wird jeder, der eine Workstation im x12-Netzwerk hat, nach Deinem Kopf suchen, um ihn auf einem Tablett zu präsentieren. Also: Halte Dein Torso <--> Gehirnnetz in Ordnung. Du kannst Die Sachen beschleunigen, wenn Du Regelgruppen einrichtest.

Regelgruppen erlauben es, statt einer linearen Liste, den Regelsatz in einer Baumstruktur zu schreiben. Das bedeutet, daß Dein Paket, wenn es mit den Tests nichts zu tun hat (alle x11-Regeln beispielsweise), diese Regeln nicht konsultieren wird. Es ist sowas wie mehrere Firewalls, die alle auf derselben Maschine laufen. Hier ist ein einfaches Beispiel:

```
block out quick on xl1 all head 10
pass out quick proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state group 10
block out on xl2 all
```

In diesem simplifizierten Beispiel können wir einen kleinen Hinweis auf die Leistung der Regelgruppe erkennen. Wenn das Paket nicht an x11 gerichtet ist, wird der Kopf der Regelgruppe 10 nicht zutreffen und das System wird mit den Tests fortfahren. Wenn das Paket nicht für x11 paßt, wird das "quick"-Schlüsselwort alles weitere im Root-Level (Regelgruppe 0) ausführen und seine Tests auf die Regeln, die der Gruppe 10 angehören; genaugenommen den SYS-Check für Port 80/tcp. Auf diesem Weg können wir die Regeln oben neu schreiben, damit wir die Performance unserer Firewall maximieren können.

```
block in quick on xl0 all head 1
  block in     quick on xl0 from 192.168.0.0/16 to any  group 1
  block in     quick on xl0 from 172.16.0.0/12 to any   group 1
  block in     quick on xl0 from 10.0.0.0/8 to any      group 1
  block in     quick on xl0 from 127.0.0.0/8 to any     group 1
  block in     quick on xl0 from 0.0.0.0/8 to any       group 1
  block in     quick on xl0 from 169.254.0.0/16 to any  group 1
  block in     quick on xl0 from 192.0.2.0/24 to any    group 1
  block in     quick on xl0 from 204.152.64.0/23 to any group 1
  block in     quick on xl0 from 224.0.0.0/3 to any     group 1
  block in log quick on xl0 from 20.20.20.0/24 to any   group 1
  block in log quick on xl0 from any to 20.20.20.0/32   group 1
  block in log quick on xl0 from any to 20.20.20.63/32  group 1
  block in log quick on xl0 from any to 20.20.20.64/32  group 1
  block in log quick on xl0 from any to 20.20.20.127/32 group 1
  block in log quick on xl0 from any to 20.20.20.128/32 group 1
  block in log quick on xl0 from any to 20.20.20.255/32 group 1
  pass  in           on xl0 all group 1

pass out on xl0 all

block out quick on xl1 all head 10
  pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 80 \
           flags S keep state group 10
  pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 21 \
           flags S keep state group 10
  pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 20 \
           flags S keep state group 10
  pass out quick on xl1 proto tcp from any to 20.20.20.65/32 port = 53 \
           flags S keep state group 10
  pass out quick on xl1 proto udp from any to 20.20.20.65/32 port = 53 \
           keep state group 10
  pass out quick on xl1 proto tcp from any to 20.20.20.66/32 port = 53 \
           flags S keep state
  pass out quick on xl1 proto udp from any to 20.20.20.66/32 port = 53 \
           keep state group 10
```

```
pass in quick on xl1 proto tcp/udp from 20.20.20.64/26 to any keep state

block out on xl2 all

pass in quick on xl2 proto tcp/udp from 20.20.20.128/25 to any keep state
```

Jetzt können wir die Regelgruppen in Aktion bewundern. Für einen Host auf dem x12-Netzwerk können wir auf alle Checks für Gruppe 10 verzichten, wenn wir nicht mit den Hosts aus diesem Netzwerk kommunizieren.

Situationsabhängig kann es auch sinnvoll sein, die Regel nach Protokollen, verschiedenen Maschinen, Netzwerkblöcken oder was auch immer den Datenfluß verbessert, zu ordnen.

3.10. "Fastroute"; Das Stichwort für Heimlichkeit

Gerade wenn wir einige Pakete weiterleiten und andere blockieren, benehmen wir uns, wie es ein wohlerzogener Router tut, der die TTL eines Paketes reduziert und der ganzen Welt mitteilt, daß hier ein Hop ist. Aber wir können unsere Präsenz vor inquisitorischen Programmen wie Traceroute, das UDP-Pakete mit verschiedenen TTL-Werten nutzt, um die Hops zwischen zwei Sites zu protokollieren, schützen.

Wenn wir wollen, daß eingehende traceroutes funktionieren, aber wir die Gegenwart unserer Firewall nicht als Hop bekanntmachen wollen, können wir das mit einer Regel wie dieser hier tun:

```
block in quick on xl0 fastroute proto udp from any to any port 33434 >< 33465
```

Das Vorhandensein des Fastroute-Schlüsselworts wird das Paket nicht für ein Routing durch den IP-Stack lassen, wenn daraus eine Reduzierung der TTL resultiert. Das Paket wird von IPFilter auf das Output-Interface geschickt, so daß eine solche Reduzierung nicht stattfinden wird. IPFilter wird natürlich die Routing-Tabelle des Systems verwenden, um herauszufinden, welches das richtige Output-Interface ist; aber es wird vorsichtig damit sein, selbst zu routen.

Das ist auch ein Grund, aus dem wir "block quick" in unserem Beispiel benutzen. Hätten wir "pass" benutzt und IP-Forwarding in unserem Kernel zugelassen, würden wir damit realisieren, daß wir zwei Pfade für ein Paket hätten, über die dieses Paket kommen kann; und würden unseren Kernel möglicherweise in Panik versetzen. Es sollte im Gedächtnis bleiben, daß die meisten Unix-Kernels über einen weitaus effizienteren Routing-Code verfügen als IPFilter. Dieses Schlüsselwort sollte nicht als ein Weg zur Steigerung der Firewallperformance verwendet werden. Und es sollte nur an Stellen verwendet werden, wo diese Heimlichkeit eine Bedingung ist.

4. NAT und Proxies

Außerhalb der Firmenumgebung ist eine der größten Verlockungen der Firewalltechnologie die Möglichkeit für den Enduser, mehrere Rechner durch ein gemeinsames externes Interface an das Internet anzustöpseln. Oftmals passiert das ohne Erlaubnis, Wissen oder Zustimmung des Service-Providers. Für alle, die Linux kennen: Diese Konzept wird IP-Masquerading genannt, aber der Rest der Welt kennt das als Network Adress Translation, oder kurz NAT.

4.1. Viele Adressen in eine umsetzen

Anmerkung für pedantische Leute:
Was IPFilter anbietet, wird wirklich NPAT genannt, kurz für "Network and Port Translation"; was heißt, daß wir die Quell- und Zieladressen nebst deren Quell- und Zielports ändern können. Wirkliches NAT erlaubt nur das Ändern der Adressen.

Die Standardanwendung von NAT ist ziemlich das Gleiche wie Linuxens IP-Masquerading. Und es wird nur eine Regel dafür gebraucht:

```
map tun0 192.168.1.0/24 -> 20.20.20.1/32
```

Sehr einfach: Wann immer ein Paket mit der CIDR-Netzmaske 192.168.1.0/24 durch das tun0-Interface geht, wird das Paket in den IP-Stack zurückgeschrieben, so das seine Quelladresse 20.20.20.1 ist und es wird an sein Originalziel geschickt. Das System pflegt außerdem eine Liste, in der steht, welche übersetzten Adressen in Arbeit sind, damit wir die Antwort zurück an den internen Host leiten können, der das Paket generiert hat (und an 20.20.20.1 gerichtet ist).

Da ist aber ein Nachteil in der gerade geschriebenen Regel. In vielen Fällen wissen wir die IP-Adresse unserer Verbindung nach draußen nicht (Wenn wir tun0 und einen typischen ISP benutzen). Das macht das Aufsetzen unserer NAT-Tabellen zu einer schwierigen Aufgabe. Glücklicherweise ist NAT so klug, daß es eine Adresse Namens 0/32 als ein Signal akzeptiert, nachzuschauen, welche Adresse das Interface wirklich hat. Wir müssen unsere Regel nur so umschreiben:

```
map tun0 192.168.1.0/24 -> 0/32
```

Jetzt können wir unsere IPNAT-Regeln ungestraft laden und uns mit der Außenwelt verbinden, ohne irgend etwas editieren zu müssen. Du mußt "ipf -y" aufrufen, um die Adresse neu zu laden, wenn Deine Verbindung abgebrochen wurde und Dich neu einwählen, wenn Deine DHCP-Adresse sich geändert hat.

Einige Leute werden sich wundern, was mit dem Quellport passiert, wenn das Paket gemappt wird. Mit unserer gegenwärtigen Regel ist der Quellport des Pakets der Gleiche wie der Original-Quellport. Es gibt aber besondere Fälle, in denen wir dieses Verhalten nicht wollen. Das kann sich um eine andere Firewall handeln, durch die unser Upstream gehen muß oder vielleicht um viele Hosts, die den selben Port benutzen, was eine Kollision verursachen kann, wenn das Paket nicht paßt und das Paket unübersetzt das System verläßt. Ipnat unterstützt uns hier mit dem "portmap"-Schlüsselwort:

```
map tun0 192.168.1.0/24 -> 0/32 portmap tcp/udp 20000:30000
```

Unsere Regel schuhlöffelt alle übersetzten Verbindungen (das kann tcp, udp oder tcp/udp sein) in die Ports von 20,000 bis 30,000.

4.2. Viele Adressen in einen Adressenpool mappen

Eine andere allgemeine Nutzung von NAT ist es, einen kleinen statischen Adressblock zu nehmen und viele Computer in diesen kleinen Adressraum zu mappen. Das ist einfach zu realisieren, wenn Du das nutzt, was Du über die "map"- und "portmap"-Schlüsselworte weißt und eine Regel schreibst, die aussieht wie diese hier:

```
map tun0 192.168.0.0/16 -> 20.20.20.0/24 portmap tcp/udp 20000:60000
```

Es gibt auch Fälle, in denen eine Remote-Applikation voraussetzt, daß multiple Verbindungen immer von derselben IP-Adresse kommen. Wir können das unterstützen, indem wir in solchen Fällen NAT erzählen, daß es bestimmte Sitzungen von einem Host im Adressenpool statisch mappen und bei der Portauswahl zaubern soll. Das geht mit dem "map-block"-Schlüsselwort wie folgt:

```
map-block tun0 192.168.1.0/24 -> 20.20.20.0/24
```

4.3. Eins-zu-eins Mappings

Manchmal ist es entscheidend, ein System mit einer IP-Adresse hinter der Firewall zu betreiben, das der Öffentlichkeit eine völlig andere Adresse zeigt. Ein Beispiel ist ein Labor aus Computern, die zu verschiedenen Netzwerken gehören und alle mit einem bestimmten Versuch laufen sollen. In diesem Beispiel mußt Du das gesamte Labornetz nicht umkonfigurieren, wenn Du ein NAT-System an der "Front" aussetzen kannst und die Adressen einfach nur an einer Stelle wechselst. Wir können das bidirektionale Mapping mit dem "bimap"-Schlüsselwort machen. Bimap bietet ein paar zusätzliche Schutzfunktionen, um einen bekannten Zustand für eine Verbindung abzusichern, wenn das "map"-Schlüsselwort benutzt wird, um eine Quelladresse nebst Port zu lokalisieren, um das Paket umzuschreiben und die Verbindung am Leben zu halten.

```
bimap tun0 192.168.1.1/32 -> 20.20.20.1/32
```

... erledigt das Mapping für einen Host.

4.4. Spoofing-Dienste

Spoofing-Dienste? Was hat das mit alledem zu tun? Viel. Stellen wir uns vor, wir haben auf Maschine 20.20.20.5 einen Webserver laufen und unsere Zweifel hinsichtlich der Sicherheit unseres Netzwerkes wachsen. Wir wollen den Server nicht mehr auf Port 80 laufen lassen, weil dieser Prozess auf Port 80 eine gewisse Lebensspanne als Root voraussetzt. Aber wie sollen wir das auf einem weniger privilegierten Port wie 8000 in der Welt der Dot-Coms laufen lassen? Wir wird man unseren Server finden? Wir können wir die Umleitungsfunktion von NAT nutzen, um dieses Problem zu lösen. Indem wir die Verbindungen, die für 20.20.20.5:80 bestimmt sind, auf 20.20.20.5:8000 umleiten. Für diese Funktion benutzen wir das "rdr"-Schlüsselwort:

```
rdr tun0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000
```

Wir können hier auch das Protokoll spezifizieren, wenn wir einen anstelle eines TCP- einen UDP-Dienst umleiten wollen (Default ist TCP). Beispiel: Wir haben einen Honigtopf auf unserer Firewall, um das populäre Back Orifice für Windows nachzuahmen. Wir können den gesamten Netzwerkverkehr auf diesen einen Platz mit einer einzigen einfachen Regel umschaufeln:

```
  rdr tun0 20.20.20.0/24 port 31337 -> 127.0.0.1 port 31337 udp
```

Ein extrem wichtiger Punkt muß bei "rdr" beachtet werden: Man kann das nicht einfach als Reflektor benutzen. Beispiel:

```
rdr tun0 20.20.20.5/32 port 80 -> 20.20.20.6 port 80 tcp
```

...wird in einer Situation, in der sich .5 und .6 im selben LAN-Segment befinden, nicht funktionieren. Dir "rdr"-Funktion wird nur an Pakete angehängt, die die Firewall über ein bestimmtes Interface betreten. Wenn ein Paket kommt, für das eine "rdr"-Regel paßt, wird es zum Filtern nach IPF umgeleitet; sollte es den Satz an Filterregeln erfolgreich durchlaufen, wird es an den Unix-Routingcode geschickt. So lange dieses Paket immer noch an dasselbe Interface gebunden ist und das System nicht verlassen kann, um einen Host zu erreichen, wird das System durcheinandergeraten. Reflektoren funktionieren einfach nicht. Nichts beschreibt die Adresse des Interfaces, auf dem das Paket eben hereingekommen ist. Behalte immer im Gedächtnis, daß "rdr"-Ziele das System auf einem anderen Interface verlassen müssen.

4.5. Transparente Proxies; Umleitungen benutzen

In der Zeitspanne, in der Du eine Firewall installiert hast, könntest Du entschieden haben, daß es sinnvoll ist, einen Proxy-Server für viele Deiner ausgehenden Verbindungen zu benutzen, um die Regeln für das interne Netzwerk für einen besseren Schutz noch enger zu setzen. Es könnte aber auch sein, daß Du in eine Situation kommst, in der der NAT-Mappingprozess nicht vernünftig gehandhabt werden kann. Das kann ebenfalls mir einem Redirection-Statement gemacht werden:

```
rdr xl0 0.0.0.0/0 port 21 -> 127.0.0.1 port 21
```

Dieses Statement besagt, daß alle Pakete, die über das x10-Interface hereinkommen und für jede Adresse (0.0.0.0/0) auf dem FTP-Port auf einen Proxy umgeleitet werden sollen, der auf dem NAT-System läuft.

Dieses spezielle Beispiel von FTP-Proxying bringt ein paar Komplikationen mit sich, wenn man das mit Web-Browsern anwendet, die nicht in der Lage sind, mit einem Proxy-Server zu kommunizieren. Es gibt da allerdings einige Patches um das TIS-FWTK für den NAT-Prozess einsatzfähig zu machen, damit es für die automatische Weiterleitung einer Verbindung herausfinden kann, wo Du hinwillst. Es gibt mittlerweile viele Proxy-Pakete, die in einer transparenten Umgebung funktionieren. Squid, zu finden unter http://squid.nlanr.net, ist sehr gut in dieser Disziplin.

Diese Anwendung der "rdr"-Schlüsselwortes ist oft besser, wenn Du das automatische Authentifizieren der User mit dem Proxy vorantreiben willst (Beispiel: Deine Ingenieure sollten im Web surfen dürfen, aber Du willst nicht, daß Deine Call-Center-Agents das auch tun).

4.6. Magisch versteckt hinter NAT; Application Proxies

Weil Ipnat eine Methode anbietet, nach der Pakete überschrieben werden, wenn sie die Firewall passieren, ist das ein bequemer Platz, um ein paar Anwendungsschichts-Proxies zu bauen, um bekannte Unterschiede zwischen typischen Firewalls und Application-Firewalls zu überschminken. Beispiel FTP: Wir können unsere Firewall so einrichten, daß sie den Paketen Beachtung schenkt, wenn sie diese passieren; und wenn sie merkt, daß sie mit einer aktiven FTP-Sitzung verhandelt, kann sie temporäre Regeln selbst schreiben, sehr ähnlich den "keep state"-Regeln. Um das zu benutzen nehmen wir eine Regel wie diese hier:

```
map tun0 192.168.1.0/24 -> 20.20.20.1/32 proxy port ftp ftp/tcp
```

Du mußt immer beachten, daß Du diese Proxy-Regel vor die Portmap-Regeln setzt; sonst wird Portmap, wenn es an der Reihe ist und sich für das Paket zuständig fühlt, das Paket überschreiben. Ein Proxy wird dann keine Chance haben, mit diesem Paket zu arbeiten. Denke daran, daß "ipnat"-Regeln immer die ersten sind, die Gültigkeit besitzen. Es gibt auch Proxies für "rcmd" (was wir zweifelhaft finden; Berkeley-r*-Kommandos sollten immer verboten werden, vor allem solche, von denen wir nicht wissen, was der Proxy tut, wenn sie aufgerufen werden) und "raudio" für die Ral-Audio-PMN-Streams. Beide Regeln dieser Art sollten vor die Portmap-Regeln gesetzt werden, wenn Du NAT verwendest.

5. Laden und Manipulieren von Filterregeln; das IPF-Utility

IP-Filter-Regeln werden bei Nutzung des Programms geladen. Die Filterregeln können in jeder Datei im System abgelegt werden. Typischerweise werden diese Regeln aber in /usr/local/etc/ipf.rules oder /etc/opt/ipf/ipf.rules gespeichert.

IP-Filter hat zwei Arten von Regeln: einen aktiven und einen inaktiven Satz. Im Normalfall werden alle Operationen nach dem aktiven Regelsatz durchgeführt.

Du kannst den inaktiven Satz mit "-I" hinter der IPF-Kommandozeile manipulieren. Zwischen beiden Sätzen kann mit der "-s" Kommandozeilenoption hin- und her geschaltet werden. Das ist sehr sinnvoll, wenn man die neuen Regeln testen will, ohne den alten Regelsatz gleich zewatechnisch zu beseitigen. Die Regeln können auch mit der Option "-r" aus der Liste entfernt werden. Aber es ist generell besser, den benutzten Regelsatz mit der Option "-F" zu flushen und nach Änderungen komplett neu zu laden, wenn Du was verändert hast.

Summa summarum: Der einfachste Weg, einen Regelsatz zu laden ist mit "ipf -FA -f /etc/ipf.rules. Für kompliziertere Änderungen sei hier auf die ipf(1) Manpage hingewiesen.

6. Laden und Manipulieren von NAT-Regeln; Das ipnat-Utility

NAT-Regeln werden bei Benutzung des ipnat-Werkzeuges geladen. Die NAT-Regeln können ebenfalls in jeder Datei im System untergebracht werden, aber es ist üblich, diese in den Dateien /etc/ipnat.rules oder in /etc/opt/ipf/ipnat.rules unterzubringen.

Auch diese Regeln können aus der Liste mit der "-r"-option in der Kommandozeile gelöscht werden; aber es ist auch hier besser, den gesamten Regelsatz zu löschen und ihn dann mit der Option "-C" komplett neu zu laden, wenn etwas geändert wurde. Irgendwelche aktiven Mappings werden von "-C" nicht angerührt und können mit "-F" entfernt werden.

NAT-Regeln und aktive Mappings können mit der "-I" Option geprüft werden.

Der einfachste Weg ist es, die NAT-Regeln mit "ipnat -CF -f /etc/ipnat.rules" neu zu laden.

7. Monitoring und Debugging

Es wird die Zeit kommen, zu der Du wissen willst, was Deine Firewall gerade tut. IPFilter wäre nicht komplett, wenn es keinen vollen Satz an Status-Monitoring-Werkzeugen hätte.

7.1. Das ipfstat-Utility

In seiner einfachsten Form zeigt Ipfstat eine Tabelle mit interessanten Daten darüber, wie Deine Firewall arbeitet: eine Tabelle darüber, wie viele Pakete die Firewall passieren durften oder blockiert wurden, ob sie aufgezeichnet wurden oder nicht, wieviele Zustandseinträge gemacht wurden und so weiter. Hier ist ein Beispiel für etwas von dem, was du sehen kannst, wenn Du das Tool laufen läßt:

```
# ipfstat
 input packets:         blocked 99286 passed 1255609 nomatch 14686 counted 0
 output packets:         blocked 4200 passed 1284345 nomatch 14687 counted 0
 input packets logged:  blocked 99286 passed 0
 output packets logged:  blocked 0 passed 0
 packets logged:        input 0 output 0
 log failures:          input 3898 output 0
 fragment state(in):     kept 0  lost 0
```

```
fragment state(out):    kept 0  lost 0
packet state(in):       kept 169364     lost 0
packet state(out):      kept 431395     lost 0
ICMP replies:   0       TCP RSTs sent:  0
Result cache hits(in):  1215208 (out):  1098963
IN Pullups succeeded:   2       failed: 0
OUT Pullups succeeded:  0       failed: 0
Fastroute successes:    0       failures:       0
TCP cksum fails(in):    0       (out):  0
Packet log flags set: (0)
       none
```

ipfstat kann Dir auch Deine aktuelle Regelliste anzeigen. Mit dem "i-" oder dem "-o"-flag werden die aktuell geladenen Regeln für der Ein- oder Ausgang angezeigt. Ein angehängtes "-h" bietet gleichzeitig brauchbarere Informationen, indem es einen "Hitcount" für jede Regel anzeigt. Beispiel:

```
    # ipfstat -ho
    2451423 pass out on xl0 from any to any
    354727 block out on ppp0 from any to any
    430918 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep state keep
frags
```

Daraus können wir ersehen, ob vielleicht etwas anormal gelaufen ist, seit wir eine Menge blockierter Pakete ausgeschlossen haben. Besonders bei einer sehr liberalen Regelung für den Ausgang. Einiges davon rechtfertigt ein vorangehendes Verhör; oder es müßte ein perfektes Design besitzen. ipfstat kann Dir nicht sagen, ob Deine Regeln richtig oder falsch sind. Es kann Dir nur sagen, was aufgrund Deiner Regeln passiert ist oder passieren wird. Um Deine Regeln vorher zu debuggen, kannst Du das "-n"-Flag benutzen, das die Nummer jeder einzelnen Regel anzeigt:

```
    # ipfstat -on
    @1 pass out on xl0 from any to any
    @2 block out on ppp0 from any to any
    @3 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep state keep frags
```

Das finale Stück einer wirklich interessanten Information, das ipfstat uns bieten kann, ist ein Dump der Zustandstabelle. Das wird mit dem "-s"-Flag gemacht:

```
    # ipfstat -s
            281458 TCP
            319349 UDP
            0 ICMP
            19780145 hits
            5723648 misses
            0 maximum
            0 no memory
            1 active
            319349 expired
            281419 closed
    100.100.100.1 -> 20.20.20.1 ttl 864000 pass 20490 pr 6 state 4/4
            pkts 196 bytes 17394    987 -> 22 585538471:2213225493 16592:16500
            pass in log quick keep state
            pkt_flags & b = 2,              pkt_options & ffffffff = 0
            pkt_security & ffff = 0, pkt_auth & ffff = 0
```

Hier sehen wir, daß wir einen Statuseintrag für eine TCP-Verbindung haben. Der Output variiert etwas von Version zu Version, aber die eigentliche Information ist immer dieselbe. Wir können in dieser Verbindung sehen, daß sie voll etabliert ist (wird durch den 4/4-Zustand angezeigt. Andere Zustände sind unvollständig und werden später dokumentiert.). Wir können sehen, daß der Statuseintrag eine Lifetime von 240 Stunden hat, was zwar eine reichlich absurde Zeitspanne ist, aber dem Standardwert einer etablierten TCP-Verbindung entspricht.

Dieser TTL-Counter wird jede Sekunde heruntergezählt, in der dieser Statuseintrag nicht benutzt wird, was am Ende bedeutet, daß die Verbindung gelöscht wird, wenn sie einige Zeit unbenutzt bleibt. Die TTL wird auch nach 864000 zurückgesetzt, wann immer dieser Status benutzt wird; Bedingung dabei ist, daß der Eintrag während seiner aktiven Benutzung nicht ungültig wird. Wir können auch sehen, daß 196 Pakete mit einer Gesamtgröße von 17KB die Firewall passieren durften. Wir können die Ports beider Endpunkte der Verbindung sehen (diesmal 987 und 22); was bedeutet, daß dieser Statuseintrag eine Verbindung von 100.100.100.1, Port 987 nach 20.20.20.1, Port 22 repräsentiert. Die wirklich großen Nummern in der zweiten Zeile sind die TCP-Sequenznummern dieser Verbindung, die uns dabei helfen, daß es nicht so einfach ist, böse Pakete in diese

Verbindung einzuschleusen. Das TCP-Fenster wird ebenfalls gezeigt. Die dritte Zeile ist eine genaue Darstellung der Regel, die vom "keep state"-Code generiert wurde und anzeigt, das diese Verbindung eine Eingehende ist.

7.2. Das ipmon-Utility

ipfstat ist auch gut darin, Schnappschüsse von dem einzusammeln, was gerade auf dem System passiert. Aber es ist oft praktisch, eine Art von Protokollierung zu haben, mit der man die Events beobachten kann, wenn sie gerade stattfinden. ipmon heißt das Werkzeug dafür. ipmon kann packet log (wenn das mit dem "log"-Schlüsselwort in den Regeln gefordert ist), state log oder NET-Log beobachten oder eben auch alle zusammen. Dieses Tool kann sowohl im Vordergrund laufen als auch im Hintergrund als Dämon; um die Daten an Syslog zu übergeben oder in eine Datei zu schreiben. Wenn wir die Statustabelle in Aktion beobachten wollen, würde "ipmon -O -s" dieses hier ausgeben:

```
    # ipmon -o S
    01/08/1999 15:58:57.836053 STATE:NEW 100.100.100.1,53 -> 20.20.20.15,53 PR udp
    01/08/1999 15:58:58.030815 STATE:NEW 20.20.20.15,123 -> 128.167.1.69,123 PR udp
    01/08/1999 15:59:18.032174 STATE:NEW 20.20.20.15,123 -> 128.173.14.71,123 PR udp
    01/08/1999 15:59:24.570107 STATE:EXPIRE 100.100.100.1,53 -> 20.20.20.15,53 PR udp Pkts 4
Bytes 356
    01/08/1999 16:03:51.754867 STATE:NEW 20.20.20.13,1019 -> 100.100.100.10,22 PR tcp
    01/08/1999 16:04:03.070127 STATE:EXPIRE 20.20.20.13,1019 -> 100.100.100.10,22 PR tcp Pkts
63 Bytes 4604
```

Hier sehen wir einen Statuseintrag für eine externe DNS-Abfrage aus unserem Nameserver, zwei xntp-Pings an bekannte Timeserver und eine sehr kurze ausgehende SSH-Verbindung. Ipmon kann uns außerdem anzeigen, welche Pakete aufgezeichnet wurden. Wenn beispielsweise "state" benutzt wird, wirst Du oft auf Pakete stoßen wie diese hier:

```
    # ipmon -o I
    15:57:33.803147 ppp0 @0:2 b 100.100.100.103,443 -> 20.20.20.10,4923 PR tcp len 20 1488 -A
```

Was bedeutet das? Das erste Feld ist einfach zu erkennen: Es zeigt den Zeitstempel. Für das zweite Feld gilt das Gleiche. Es zeigt uns das Interface, auf dem das Event stattgefunden hat. Das dritte Feld (@0:2) ist etwas, das viele Leute vermissen: Es ist die Regel, die einem Event sagt, daß er stattzufinden hat. Erinnerst Du Dich an "ipfstat -in"? Wenn Du wissen willst, wo das herkam, kannst Du Dir hier Regel 2 in Gruppe 0 heraussuchen. Das vierte Feld, das kleine "b" besagt, daß dieses Paket blockiert wurde. Generell wirst Du das ignorieren, bis Du Pakete, die passieren durften, ebenfalls protokollierst. Dann erscheint an dieser Stelle ein kleines "p".

Die Felder fünf und sechs erklären sich eigentlich selbst. Sie sagen aus, wo das Paket herkam und wohin es gegangen ist. Das siebte ("PR") und das achte Feld erzählen Dir, über welches Protokoll das Paket gegangen ist und Feld neun stellt die Größe des Pakets dar. Der letzte Teil, das "-A" zeigt uns die die Flags des Pakets. Dieses was ein ACK-Paket. Warum erwähnte ich die Zustände nicht eher? Durch die oftmals etwas krude Natur des Internets werden die Pakete manchmal regeneriert. Manchmal bekommst zwei Kopien eines Pakets und Deine Zustandsregel, die die Sequenznummern in der Spur hält, wird das Paket auch zweimal gesehen haben und davon ausgehen, daß das Paket ein Bestandteil einer anderen Verbindung ist Eventuell läuft das Paket durch eine reale Route, die damit gedealt hat. Das letzte Paket einer geschlossenen Sitzung wird oft protokolliert weil der "keep state"-Code die Verbindung schon beendet hat, bevor das letzte Paket die Chance hatte, Deine Firewall zu passieren. Das ist normal. Du brauchst Die also keine Sorgen zu machen. Ein anderes Beispiel für ein Paket, das aufgezeichnet sein könnte:

```
    12:46:12.470951 xl0 @0:1 S 20.20.20.254 -> 255.255.255.255 PR icmp len 20 9216 icmp 9/0
```

Das hier ist ein ICMP-Router-Discovery-Broadcast. Wir können das mittels des ICMP-Typs 9/0 bekanntmachen.

Zu guter Letzt läßt uns ipmon auch die NAT-Tabelle in Aktion begutachten:

```
    # ipmon -o N
    01/08/1999 05:30:02.466114 @2 NAT:RDR 20.20.20.253,113 <--> 20.20.20.253,113
[100.100.100.13,45816]
    01/08/1999 05:30:31.990037 @2 NAT:EXPIRE 20.20.20.253,113 <--> 20.20.20.253,113
[100.100.100.13,45816] Pkts 10 Bytes 455
```

Das wäre eine Umleitung zu einem identd, der einem Außenstehenden vorflunkert, einen Ident-Dienst für die Hosts hinter unserer NAT anzubieten, seit sie typischerweise nicht mehr in der Lage sind, sich selbst diesen Service mit der normalen NAT anzubieten.

8. Bestimmte Anwendungen

der IPFilter-Sachen sollten aber auch erwähnt werden.

8.1 Keep State mit Servern und Flags.

Einen Zustand zu halten ist eine gute Sache, aber es ist ziemlich leicht, einen Fehler in der Richtung zu machen, in der Du "keep state" ausführen willst. Normalerweise wird ein "keep state"-Schlüsselwort in der ersten Regel eines Paketes gesetzt, das mit einer Regel für eine Verbindung interagiert. Ein verbreiteter Fehler wird gemacht, wenn man das Halten eines Zustandes mit Filter-Flags wie diesen mischt:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 flags S
pass out all keep state
```

Das erscheint so, als ob Du eine Verbindung zum Telnet-Server auf 20.20.20.20 nebst der ausgehenden Antworten erlaubst. Wenn Du versuchst, diese Regel zu benutzen, wirst Du sehen, daß sie nur momentan funktioniert. Seit wir nach den SYN-Flags filtern, wird der Zustandseintrag niemals vollständig erstellt; und die Standard-TTL eines unvollständigen Zustandes ist immer 60 Sekunden. Wir können das Problem lösen, indem wir diese Regel auf einem dieser beiden Wege neu schreiben:

1)

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 keep state
block out all
```

oder:

2)

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 flags S keep state
pass out all keep state
```

Jeder dieser Regelsätze wird zu einem voll etabliertem Zustandseintrag für eine Verbindung zu Deinem Server führen.

8.2. Kopieren mit FTP

FTP ist eines dieser Protokolle, bei denen Du Dich zurücklehnen und die Frage stellen mußt "Was zur Hölle haben die sich dabei gedacht?". FTP hat viele Probleme, mit denen ein Systemadministrator umgehen muß. Was schlimmer ist: Die Probleme, die der Administrator finden muß unterscheiden sich zwischen dem Aufsetzen von FTP-Servern und dem von -Clients. Im FTP-Protokoll gibt es zwei Formen von Datentransfer: aktiv und passiv genannt. Aktive Transfers sind die, in denen der Server sich mit einem offenen Port am Client verbindet, um Daten zu schicken. Bei passiven Verbindungen verbindet sich der Client mit dem Server, um Daten zu empfangen.

8.2.1. Betrieb eines FTP Server

Bei einem laufenden FTP-Server ist es einfach, aktive FTP-Sitzungen aufzusetzen und sie zu händeln. Gleichzeitig ist das Handling von passivem FTP ein großes Problem. Als Erstes regeln wir, wie aktives FTP zu handhaben ist. Dann machen wir das Gleiche mit der passiven Variante. Grundsätzlich können wir aktive FTP-Sitzungen als eingehende HTTP-oder SMTP-Sitzung handhaben, ganz wie es gefällt. Du mußt nur den FTP-Port öffnen und "keep state" den Rest machen lassen:

```
pass in quick proto tcp from any to 20.20.20.20/32 port = 21 flags S keep state
pass out proto tcp all keep state
```

Diese Regeln erlauben alle aktiven FTP-Sitzungen (den gängigsten Typ) mit dem FTP-Server auf 20.20.20.20.

Die nächste Herausforderung ist das Handling von passiven FTP-Verbindungen. Webbrowser benutzen normalerweise diesen Modus. Er ist deshalb ziemlich populär und deshalb sollte das unterstützt werden. Das Problem dabei ist, daß bei jeder passiven Sitzung der Server beginnt, an einem neuen Port auf Verbindungen zu warten (normalerweise oberhalb 1023). Das ist fast wie das Erschaffen eines neuen unbekannten Dienstes auf dem Server. Ausgehen davon, daß wir eine gute Firewall mit einer Default-Deny-Regel haben, wird der neue Dienst blockiert und diese aktiven FTP-Sitzungen werden abgebrochen. Aber verzweifle nicht! Es gibt noch Hoffnung. Die erste Idee eines Menschen wird es sein, einfach alle Ports oberhalb 1023 zu öffnen. Die Wahrheit ist, daß das funktioniert:

```
pass in quick proto tcp from any to 20.20.20.20/32 port > 1023 flags S keep state
pass out proto tcp all keep state
```

Das ist natürlich schon etwas unbefriedigend. Wenn man alles oberhalb Port 1023 hereinläßt, öffnen wir uns selbst für eine Anzahl potentieller Probleme. Während der Bereich 1-1023 eigentlich für die Serverdienste gedacht ist, laufen bestimmte Programme (z.B. NFS oder X) auf Nummern >1023.

Die gute Nachricht ist, daß Dein FTP-Server entscheiden wird, welche Ports für passive Sitzungen benutzt werden. Das heißt: Statt alle Ports oberhalb von 1023 zu öffnen, kannst Du festlegen, daß nur die Ports 15001 bis 19999 für diesen Zweck verwendet werden sollen und nur diesen Bereich durch Deine Firewall öffnest. In wu-ftp wird das mit der "passive ports"-Option in ftpaccess gemacht. Auf der IPFilter-Seite müssen wir nur die Regeln für die Korrespondenz setzen:

```
pass in quick proto tcp from any to 20.20.20.20/32 port 15000 >< 20000 flags S keep state
pass out proto tcp all keep state
```

Wenn gerade diese Lösung Dich nicht zufriedenstellt, kannst Du den IPF-Support auch in Deinen FTP-Server hacken oder eben FTP-Server-Support in IPF mit einbauen.

8.2.2. Betrieb eines FTP Clients

Während der FTP-Server-Support in IPF immer noch nicht perfekt ist, funktioniert der FTP-Client-Support ab Version 3.3.3 schon ganz gut. Genau wie bei den Servern gibt es zwei Arten von FTP-Client-Transfers: Aktiv und passiv. Die einfachste Art eines Client-Transfers ist aus dem Standpunkt der Firewall der passive Transfer. Ausgehen davon, daß Deine "keep state"-Regeln alle ausgehenden TCP-Verbindungen enthalten, werden passive Transfers schon funktionieren. Wenn Du das noch nicht tust, denke bitte mal über folgendes nach:

```
pass out proto tcp all keep state
```

Der zweite Typ des Client-Transfers (aktiv) ist etwas schwieriger, aber nichts desto weniger ein gelöstes Problem. Aktive Transfers fordern den Server auf, eine zweite Verbindung für den Datendurchfluß zurück zum Client aufzubauen. Das ist normalerweise ein Problem, wenn sich in der Mitte eine Firewall befindet, die eingehende Verbindungen stoppt. Um das zu lösen, enthält IPFilter einen IPNAT-Proxy, der temporär einen Weg nur für den FTP-Server die Firewall zurück zum Client öffnet. Sogar, wenn Du IPNAT nicht für NAT benutzt, ist dieser Proxy aktiv. Die folgenden Regeln sind das nackte Minimum, das in die IPNAT-Konfiguration eingebaut werden sollte (ep0 soll das Interface für die ausgehende Verbindung sein):

```
map ep0 0/0 -> 0/32 proxy port 21 ftp/tcp
```

Weitere Details für interne IPFilter-Proxies stehen in Kapitel 3.6.

8.3. Kernel-Variablen

Es gibt da einige nützlich Kernel-Tuneups, die entweder gesetzt werden müssen, damit IPF funktioniert oder die man einfach kennen sollte, wenn man eine Firewall bauen will. Das Wichtigste, was Du aktivieren mußt, ist das IP-Forwarding. Sonst wird IPF nur sehr wenig tun. Beispielsweise werden im darunterliegenden IP-Stack keine Pakete geroutet.

IP Forwarding:

openbsd:
```
net.inet.ip.forwarding=1
```

freebsd:
```
net.inet.ip.forwarding=1
```

netbsd:
```
net.inet.ip.forwarding=1
```

solaris:
```
ndd -set /dev/ip ip_forwarding 1
```

Ephemeral Port Adjustment:

openbsd:
```
net.inet.ip.portfirst = 25000
```

freebsd:
```
net.inet.ip.portrange.first  =  25000  net.inet.ip.por-
trange.last = 49151
```

netbsd:
```
net.inet.ip.anonportmin = 25000 net.inet.ip.anonportmax
= 49151
```

solaris:
```
ndd -set /dev/tcp tcp_smallest_anon_port 25000
ndd -set /dev/tcp tcp_largest_anon_port 65535
```

Andere nützliche Werte:

openbsd:
```
net.inet.ip.sourceroute = 0
net.inet.ip.directed-broadcast = 0
```

freebsd:
```
net.inet.ip.sourceroute=0
net.ip.accept_sourceroute=0
```

netbsd:
```
net.inet.ip.allowsrcrt=0
net.inet.ip.forwsrcrt=0
net.inet.ip.directed-broadcast=0
net.inet.ip.redirect=0
```

solaris:
```
ndd -set /dev/ip ip_forward_directed_broadcasts 0
ndd -set /dev/ip ip_forward_src_routed 0
ndd -set /dev/ip ip_respond_to_echo_broadcast 0
```

FreeBSD hat einige zuätzliche IPF-spezifische sysctl-Variablen:

```
net.inet.ipf.fr_flags: 0
net.inet.ipf.fr_pass: 514
net.inet.ipf.fr_active: 0
net.inet.ipf.fr_tcpidletimeout: 864000
```

```
net.inet.ipf.fr_tcpclosewait: 60
net.inet.ipf.fr_tcplastack: 20
net.inet.ipf.fr_tcptimeout: 120
net.inet.ipf.fr_tcpclosed: 1
net.inet.ipf.fr_udptimeout: 120
net.inet.ipf.fr_icmptimeout: 120
net.inet.ipf.fr_defnatage: 1200
net.inet.ipf.fr_ipfrttl: 120
net.inet.ipf.ipl_unreach: 13
net.inet.ipf.ipl_inited: 1
net.inet.ipf.fr_authsize: 32
net.inet.ipf.fr_authused: 0
net.inet.ipf.fr_defaultauthage: 600
```

9. Spaß mit ipf!

Dieses Kapitel zeigt Dir nichts notwendigerweise irgendwas Neues über IPF; aber es bringt unter Umständen ein paar Dinge zur Sprache, über die Du vielleicht bisher noch nicht nachgedacht hast. Es könnte auch passieren, daß es Deine Hirnwindungen in eine Dankrichtung führt, aus der heraus Du etwas neues entdeckst, über das wir bisher noch nicht nachgedacht haben.

9.1. Localhost filtern

Vor langer Zeit in einer weit entfernten Universität, schuf Weitse Venema das TCP-Wrapper-Paket. Und immer wurde es benutzt, um überall in der Welt für die Netzwerkdienste einen gewissen Schutz zu bieten. Das ist gut. Aber die TCP-Wrapper haben ein paar nicht zu kleine Fehler.

Für Anfänger: Die TCP-Wrappers schützen nur TCP-Dienste, wie der Name schon sagt. Das bedeutet: Wenn Du die Dienste vom inetd startest, oder Du die Sachen speziell mit libwrap oder kompatiblen Varianten compiliert hast, sind Deine Dienste nicht geschützt. Das hinterläßt als Folge gigantische Sicherheitslöcher in Deinen Hosts. Wir können diese mittels IPF schließen, indem wir das auch auf dem Localhost anwenden. Beispiel: Mein Laptop wird oftmals in Netzwerke gestöpselt, denen ich nicht unbedingt traue. Und weil das eben so ist, benutze ich den folgenden Regelsatz:

```
pass in quick on lo0 all
pass out quick on lo0 all

block in log all
block out all

pass in quick proto tcp from any to any port = 113 flags S keep state
pass in quick proto tcp from any to any port = 22 flags S keep state
pass in quick proto tcp from any port = 20 to any port 39999 >< 45000 flags S keep state

pass out quick proto icmp from any to any keep state
pass out quick proto tcp/udp from any to any keep state keep frags
```

Er ist eine ganze Weile so geblieben und hat mir in dieser Zeit keine Schmerzen oder Ängste bereitet, weil IPF die ganze Zeit geladen war. Als ich diesen Regelsatz enger setzen wollte, konnte ich auf den NAT-FTP-Proxy umschalten und ein paar weitere Regeln einbauen, um Spoofing-Angriffe zu verhindern. Aber gerade wie sie jetzt dasteht, ist diese Kiste weitaus restriktiver als es für das lokale Netzwerk aussieht und sie tut weit mehr, als es ein typischer Host tut. Das ist eine gute Idee, wenn Du eine Maschine laufen lassen mußt, die viele User benutzen dürfen. Und Du kannst sicher gehen, daß keiner von ihnen einen Dienst starten kann, wenn das nicht erlaubt ist. Es wird einen richtigen Hacker nicht davon abhalten, mittels eines Root-Accesses die IPF-Regeln zu ändern oder auf irgendeinem Weg einen Dienst zu starten. Aber es wird das "ehrliche Volk" ehrlich bleiben lassen und deine Dienste sicher, gemütlich und warm halten; besonders, wenn das LAN unsicher ist. Aus meiner Sicht ist das ein großer Gewinn. Das Localhost-Filtering in Verbindung mit etwas wie einer "wenig restriktiven" Firewall-Maschine kann genauso viele Performance-Probleme lösen wie auch solche politischen Alpträume wir "Warum geht IRC nicht?" und "warum kann ich keinen Webserver auf meiner Workstation installieren?! Das ist doch MEINE WORKSTATION!!". Ein anderer sehr großer Gewinn. Wer sagt denn, daß man Sicherheit und Bequemlichkeit nicht gleichzeitig haben kann?

9.2. Welche Firewall? Transparentes Filtern.

Ein der Hauptsorgen beim Aufsetzen einer Firewall ist die Integrität der Firewall selbst. Kann jemand in Deine Firewall eindringen und dabei den Regelsatz umdrehen? Das ist ein allgegenwärtiges Problem, das Administratoren nicht aus den Augen verlieren sollten. Besonders, wenn sie Firewall-Lösungen auf ihren Unix- oder NT-Maschinen benutzen. Einige Leute mißbrauchen diesen Aspekt als ein Argument für eine Blackbox-Hardwarelösung. Und das mit dem falschen Argument, daß die innere Obskurität von deren geschlossenen Systemen ihre Sicherheit verbessere. Wir haben da einen besseren Weg.

Viele Network-Admins wissen, wie die gängige Ethernet-Bridge funktioniert. Das ist ein Gerät, das zwei Ethernet-Segmente zu einem verbindet. Eine Ethernet-Bridge wird meistens dazu verwendet, separate Gebäude miteinander zu verbinden, Geschwindigkeiten umzuschalten oder die maximalen Kabellängen zu vergrößern. Hubs und Switches sind auch Bridges; manchmal sind sie nur Geräte mit zwei Anschlüssen, die man Repeater nennt. Verschiedene Versionen von NetBSD, OpenBSD, FreeBSD und Linux beinhalten auch Software, mit der man einen 1000$-PC in eine 10$-Bridge verwandeln kann. Was alle Bridges gemeinsam haben ist die Eigenschaft, daß sie, wenn sie in der Mitte einer Verbindung sitzen, von beiden Maschinen nicht bemerkt werden. Man nehme IPFilter und OpenBSD, sagt da der Chefkoch.

Das Ethernet-Bridging hat seinen Platz in Layer 2 des ISO-Stacks. IP hat seinen Platz in Layer 3. IPFilter befaßt sich hauptsächlich Layer 3, aber es beschäftigt sich auch mit Layer 2, wenn es mit den Interfaces arbeitet. Wenn wir IPFilter mit OpenBSD's Bridge-Device mischen, können wir eine Firewall bauen, die sowohl unerreichbar als auch unsichtbar ist. Das System braucht keine IP-Adresse; und es muß auch seine Ethernet-Adresse nicht preisgeben. Das einzig sichtbare Zeichen für das Vorhandensein eines Filters ist, daß die Latenzzeiten von cat5 etwas höher sind als bei einem normalen cat5 und daß es so aussieht, daß die Pakete es nicht bis zu ihrem endgültigen Ziel schaffen. Das Aufsetzen eines solchen Regelsatzes ist überraschenderweise auch ziemlich einfach. In OpenBSD wird das erste Bridging-Interface "bridge0" genannt. Nehmen wir an, wir haben zwei Ethernet-Karten in unserer Maschine, genannt x10 und x11. Um diese Maschine in eine Bridge umzuwandeln ist alles, was getan werden muß, die folgenden Kommandos einzugeben:

```
brconfig bridge0 add xl0 add xl1 up
ifconfig xl0 up
ifconfig xl1 up
```

An diesem Punkt wird der gesamte ankommende Verkehr auf x10 an x11 geschickt und der gesamte Verkehr von x11 geht zur Weiterleitung an x10. Du wirst sicher gemerkt haben, daß weder x10 noch x11 eine IP-Adresse besitzen. Wir müssen diese Interfaces auch nicht mit einer Adresse versehen. Wenn man über alle diese Dinge nachdenkt, ist es das Beste, wenn wir auch keine hinzufügen.

Regelsätze verhalten sich grundsätzlich so, wie sie es immer tun. Breit und bräsig ist da ein Bridge0-Interface; wir filtern nicht auf seiner Basis. Dir Regeln folgen basierend auf das spezielle Interface, das wir benutzen; was es wichtig macht, welches Kabel denn nun in welchem Interface auf der Rückseite der Maschine eingestöpselt ist. Laß uns mal mit ein paar einfachen Regeln starten, um zu illustrieren, was den nun genau passiert ist. Gehe mal davon aus, daß ein Netzwerk benutzt wird wie dieses hier:

```
20.20.20.1 <--------------------------------> 20.20.20.0/24 network hub
```

Das sieht so aus: Wir haben einen Router auf 20.20.20.1, der an das 20.20.20.0/24-Netzwerk angeschlossen ist. Alle Pakete aus dem 20.20.20.0/24-Netz gehen durch diesen Router, um in die Außenwelt zu gelangen und zurück. Jetzt fügen wir die IPF-Bridge hinzu:

```
20.20.20.1 <-------/xl0 IpfBridge xl1/-------> 20.20.20.0/24 network hub
```

Wir haben zusätzlich den folgenden Regelsatz auf dem IPFBridge-Host geladen:

```
pass in  quick  all
pass out quick  all
```

Mit diesem geladenen Regelsatz ist die Netzwerkfunktionalität identisch. Soweit das den 20.20.20.1-Router und die 20.20.20.0/24-Hosts betrifft, sind die beiden Netzwerkdiagramme identisch. Laß uns den Regelsatz mal ein wenig ändern:

```
block in quick on xl0 proto icmp
pass  in  quick all
pass  out quick all
```

20.20.20.1 und 20.20.20.0/24 denken immer noch, daß das Netzwerk identisch ist. Aber wenn 20.20.20.1 20.20.20.2 anpingen soll, wird der Host niemals eine Antwort erhalten. Was auch sonst. 20.20.20.2 wir gerade das erste Paket nicht bekommen. IPFilter wird das Paket abfangen, bevor es an der anderen Seite des virtuellen Kabels ankommt. Wir können einen Bridged-Filter überall einbauen. Mit dieser Methode können wir den Vertrauenskreis im Netzwerk auf einen individuellen hostbasierten Level einschränken (sofern genug Ethernet-Karten vorhanden sind...). Das Blockieren des ICMP-Verkehrs scheint etwas unglücklich zu sein, besonders, wenn Du Systemadministrator bist und Du zuweilen pingen, tracerouten oder Deine MTU zurücksetzen mußt. Laß uns mal einen besseren Regelsatz bauen und mit Hilfe des Originalen IPF-Schlüsselfeatures einen Vorteil verschaffen: Die stateful inspection (keep state).

```
pass  in quick on xl1 proto tcp  keep state
pass  in quick on xl1 proto udp  keep state
pass  in quick on xl1 proto icmp keep state
block in quick on xl0
```

In dieser Situation kann das 20.20.20.0/24-Netzwerk (vielleicht etwas einfacher als x11-Netz bezeichnet) die Welt draußen erreichen; aber von draußen kann weder jemand in das Netz hinein; noch kann er herausfinden, warum das so ist. Der Router ist erreichbar; die Hosts sind aktiv, aber von außen kann niemand hinein. Besonders, wenn der Router bloßgestellt wurde, bleibt die Firewall aktiv und erfolgreich.

Bis hierher haben wir nur nach Interfaces und Protokollen gefiltert. Besonders das Bridging berührt auch Layer 2; so können wir auch bestimmte IP-Adressen diskriminieren. Normalerweise laufen auch ein paar Dienste. Unser Regelsatz könnte also so aussehen:

```
pass  in quick on xl1 proto tcp  keep state
pass  in quick on xl1 proto udp  keep state
pass  in quick on xl1 proto icmp keep state
block in quick on xl1 # nuh-uh, we're only passing tcp/udp/icmp sir.
pass  in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
pass  in quick on xl0 proto tcp from any to 20.20.20.2/32 port=53 flags S keep state
pass  in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25 flags S keep state
pass  in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80 flags S keep state
block in quick on xl0
```

Jetzt haben wir ein Netzwerk, in dem 20.20.20.2 als Nameserver dient; 20.20.20.3 ist der Server für eingehende Mail und 20.20.20.7 ist ein Webserver. Wir müssen allerdings einräumen, daß das Bridging mit IPFilter noch nicht perfekt ist.

Zuerst solltest Du wissen, daß alle aufgesetzten Regeln anstelle einer bidirektionalen Ausrichtung nur die "in"-Richtung benutzen. Das liegt daran, daß die "out"-Richtung unter OpenBSD in der Zusammenarbeit mit dem Bridgeing noch nicht implementiert ist. Das wurde ursprünglich so gemacht, um große Performanceverluste mit vielen Netzwerkkarten zu verhindern. Es wurde bereits daran gearbeitet, die Geschwindigkeit zu steigern, aber es gilt noch als unimplementiert. Wenn Du dieses Feature wirklich willst, kannst Du selbst Hand an den Code legen oder die OpenBSD-Leute fragen, wie Du ihnen helfen kannst.

Zum Zweiten macht der Gebrauch von IPFilter mit den Bridges IPF's NAT-Features unbenutzbar, wenn das nicht gefährlich werden soll. Das erste Problem ist, daß es eben eine Filtering-Bridge ist. Das zweite Problem ist, daß die Bridge keine eigene IP-Adresse zum Maskieren besitzt, was mit Sicherheit zu Konfusion und vielleicht auch einer Kernel-Panic beim Booten führen wird. Du kannst natürlich das ausgehende Interface mit einer IP-Adresse versehen, damit NAT funktioniert, aber dann wird ein Teil der Schadenfreude beim Bridging vermindert.

9.2.1. Transparent Filtering zum Korrigieren von Designfehlern im Netz

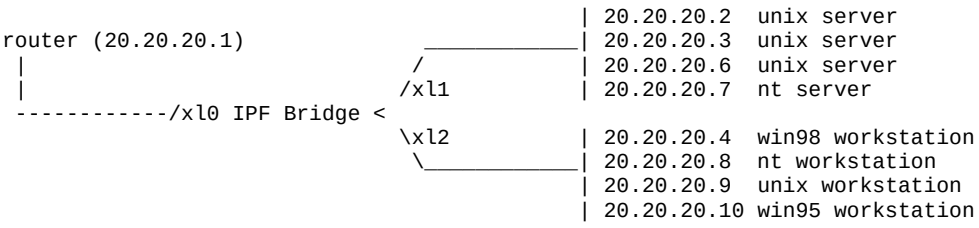
Viele Organisationen begannen, IP zu benutzen, bevor sie auf die Idee kamen, daß es gut sein könnte, eine Firewall oder ein Subnetz einzurichten. Jetzt haben sie ein Class-C-Netz oder größer, das alle ihre Server, Workstations, Router, Kaffeemaschinen, Staubsauger, eben einfach alles, beinhaltet. Das ist der Grusel schlechthin. Das Umadressieren in brauchbare Subnetze, Trustlevels, Filter etc. ist zeitaufwendig und teuer. Die Ausgaben für Hardware und Mannstunden alleine können so teuer werden, daß die meisten Organisationen nicht

daran interessiert sind, das Problem zu lösen. Dabei wird allerdings nicht die daraus resultierende Downtime erwähnt. Ein typisches Problem-Netzwerk sieht so aus:

```
20.20.20.1   router               20.20.20.6   unix server
20.20.20.2   unix server          20.20.20.7   nt workstation
20.20.20.3   unix server          20.20.20.8   nt server
20.20.20.4   win98 workstation    20.20.20.9   unix workstation
20.20.20.5   intelligent switch   20.20.20.10  win95 workstation
```

Nur ist es 20mal größer und entsprechend chaotischer und außerdem zu einem guten Teil undokumentiert. Im Idealfall hast Du alle vertraulichen Server in einem Subnetz, die Workstations in einem anderen und die Netzwerk-Switches in einem dritten. Dann würde der Router die Pakete zwischen den Subnetzen filtern und den Workstation limitierten Zugriff auf die Server erlauben; kein Zugriff auf die Switches erlauben und nur der Workstation des Systemadministrators den Zugriff auf den Kaffeepott erlauben. Ich habe noch nie ein Class-C-Netzwerk in so einem Zusammenhang gesehen. IPFilter kann da helfen.

Um damit zu starten, trennen wir zuerst den Router, die Workstations und die Server voneinander. Um das zu können, brauchen wir zwei Hubs, die wir vielleicht sogar schon haben und eine IPF-Maschine mit drei Ethernet-Karten. Wir beginnen damit, daß wir die Server auf einen Hub legen und die Workstations mit dem anderen verbinden. Normalerweise verbinden wir dann die Hubs untereinander und dann mit dem Router. Statt dessen werden wir den Router mit IPFs x10-Interface verbinden; die Server kommen an x11 und die Workstations werden mit x12 verbunden. Unser Netzwerkplan sieht dann etwa so aus wie dieser hier:



Wo vorher nichts außer Kabeln war, ist jetzt eine Filtering-Bridge, für die kein einziger Host modifiziert werden muß, um davon zu profitieren. Wir haben das Bridging aktiviert, damit sich das Netzwerk im Normalfall perfekt verhält. Vorher starten wir mit einem Regelsatz, der weitgehend unserem letzten Satz entspricht:

```
pass  in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
pass  in quick on xl0 proto tcp from any to 20.20.20.2/32 port=53 flags S keep state
pass  in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25 flags S keep state
pass  in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80 flags Skeep state
block in quick on xl0
pass  in quick on xl1 proto tcp keep state
pass  in quick on xl1 proto udp  keep state
pass  in quick on xl1 proto icmp keep state
block in quick on xl1 # juh-uh, wir lassen nur tcp/udp/icmp durch, Sir
pass  in quick on xl2 proto tcp  keep state
pass  in quick on xl2 proto udp  keep state
pass  in quick on xl2 proto icmp keep state
block in quick on xl2 # nuh-uh, we're only passing tcp/udp/icmp sir.
```

Zu Wiederholung: Verkehr, der vom Router kommt, ist auf DNS, SMTP und HTTP beschränkt. In diesem Moment können die Server frei miteinander kommunizieren. Abhängig davon, welcher Organisation Du angehörst, könnte es sein, daß Du einige dieser Funktionen nicht magst. Vielleicht willst Du nicht, daß Deine Workstations immer Zugriff auf Deine Server haben? Nimm diesen x12-Regelsatz:

```
pass  in quick on xl2 proto tcp  keep state
pass  in quick on xl2 proto udp  keep state
pass  in quick on xl2 proto icmp keep state
block in quick on xl2 # nuh-uh, we're only passing tcp/udp/icmp sir.
```

... und ändere ihn nach diesem Muster:

```
block in quick on xl2 from any to 20.20.20.0/24
pass  in quick on xl2 proto tcp  keep state
pass  in quick on xl2 proto udp  keep state
pass  in quick on xl2 proto icmp keep state
block in quick on xl2 # nuh-uh, we're only passing tcp/udp/icmp sir.
```

Vielleicht willst Du ja nur, das sie die Server für das Versenden von Mail mittels IMAP benutzen? Leicht gemacht:

```
    pass  in quick on xl2 proto tcp from any to 20.20.20.3/32 port=25
    pass  in quick on xl2 proto tcp from any to 20.20.20.3/32 port=143
    block in quick on xl2 from any to 20.20.20.0/24
    pass  in quick on xl2 proto tcp  keep state
    pass  in quick on xl2 proto udp  keep state
    pass  in quick on xl2 proto icmp keep state
    block in quick on xl2 # nuh-uh, we're only passing tcp/udp/icmp Sir.
```

Jetzt sind Deine Workstations und Server vor Zugriffen von außen geschützt; außerdem sind Deine Server vor Deinen Workstations geschützt. Vielleicht ist das Gegenteil richtig; es könnte sein, daß Deine Workstations Zugriff auf die Server haben sollen, aber nicht auf die Welt draußen. Vielleicht betrifft die nächste Exploit-Generation die Workstations und nicht die Server. In diesem Fall kannst Du die x12-Regeln so ändern, daß sie wie diese hier aussehen:

```
    pass  in quick on xl2 from any to 20.20.20.0/24
    block in quick on xl2
```

Jetzt sind die Server die Herrscher, aber die Clients können sich nur mit den Servern verbinden. Wir können die Schotten auf den Servern auch dichtmachen:

```
    pass  in quick on xl1 from any to 20.20.20.0/24
    block in quick on xl1
```

Mit der Kombination dieser beiden können die Clients und Server miteinander reden, aber sie bekommen keinen Zugang zur Außenwelt (Allerdings kann die Außenwelt auf die paar Dienste von früher zugreifen). Der gesamte Regelsatz sieht dann etwa so aus:

```
    pass  in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
    pass  in quick on xl0 proto tcp from any to 20.20.20.2/32 port=5 flags S keep
state
    pass  in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25 flags S keep
state
    pass  in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80 flags S keep
state
    block in quick on xl0
    pass  in quick on xl1 from any to 20.20.20.0/24
    block in quick on xl1
    pass  in quick on xl2 from any to 20.20.20.0/24
    block in quick on xl2
```

Man merke sich: Wenn Dein Netzwerk ein Chaos aus verschiedenen IP-Adressen und Rechnerklassen ist, können Filter-Bridges ein Problem lösen, mit dem man sonst leben müßte und vielleicht auch eines Tages angegriffen wird.

9.3. Drop-Safe-Logging mit dup-to und to.

Bis jetzt haben wir den Filter benutzt, um Pakete zu verwerfen. Anstatt sie einfach fallenzulassen, können wir diese Pakete auch auf ein anderes System umleiten und etwas nützliches mit ihren Informationen anstellen, indem wir sie mit IPMon aufzeichnen. Unser Firewall-System, sei es eine Bridge oder ein Router, kann so viele Interfaces haben, wie wir einbauen können. Wir können diese Information nutzen, um ein "drop-safe" für unsere Pakete zu schaffen. Ein gutes Beispiel wäre es, ein Intrusion-Detection-Netzwerk zu implementieren. Für Anfänger kann es entscheidend sein, unser Detection-Netzwerkes vor unserem realen Netzwerk zu verstecken, so daß wir verhindern können, entdeckt zu werden. Bevor wir anfangen: Es gibt ein paar Charakteristiken, die wir uns merken sollten. Wenn wir nur mit blockierten Paketen dealen wollen, können wir jeweils das "to"- oder das "fastroute"-Schlüsselwort verwenden (Wir erklären den Unterschied zwischen diesen beiden später). Wenn wir die Pakete passieren lassen, was wir im Normalfall wollen, brauchen wir eine Kopie des Paketes für unsere "drop-safe"-Aufzeichnung mittels des "dup-to"-Schlüsselwortes.

9.3.1. Die dup-to Methode

Wenn wir beispielsweise eine Kopie von allem, was über das x13-Interface auf ed0 in Richtung unseres "drop-safe"-Netzes verläßt, benutzen wir diese Regel in unserer Filterliste:

```
pass out on xl3 dup-to ed0 from any to any
```

Es kann auch sein, daß Du das Paket direkt zu einer bestimmten IP-Adresse in Deinem "drop-safe"-Netz schicken willst, anstatt nur eine Kopie davon zu machen und aus das Beste zu hoffen. Dafür modifizieren wir unsere Regel ein wenig:

```
pass out on xl3 dup-to ed0:192.168.254.2 from any to any
```

Wir warnen aber davor, daß diese Methode die Zieladresse des kopierten Pakets ändern wird, was die Aufzeichnung unbrauchbar machen kann. Aus diesem Grund empfehlen wir die Methode mit den bekannten Adressen nur, wenn Du sicher bist, daß die Adresse, die Du aufzeichnest in irgendeiner Form mitteilt, für wen Du aufzeichnest (Beispiel: Benutze nicht dieselbe Adresse, um sowohl den Verkehr für den Webserver als auch den Mailserver aufzuzeichnen. Du wirst schwere Zeiten erleben, wenn Du später herausfinden mußt, welches System das Ziel für einen bestimmten Paketsatz gewesen ist.). Diese Technik kann auch effektiv genutzt werden, wenn Du eine IP-Adresse auf Deinem "drop-safe"-Netz in ziemlich demselben Weg verwendest, wie eine Multicast-Gruppe im realen Internet (Beispiel: 192.168.254.2 könnte der Kanal für Deinen HTTP-Traffic-Analysesystem sein; 23.23.23.23 wäre vielleicht der Kanal für Telnet-Sitzungen; etc.). Du mußt diesen Adressensatz zu Zeit nicht als eine Adresse oder ein Alias auf irgendeinem Deiner Analysen-Systeme setzen. Normalerweise wird die IPFilter-Maschine gebraucht, um für die neue Zieladresse zu ARPen (dup-to ed0:192.168.254.2 natürlich). Aber wir können diese Entscheidung umgehen, indem wir einen statischen ARP-Eintrag für diesen Kanal In unserem IPFilter-System vornehmen. Grundsätzlich ist "dup-to ed0" alles, was benötigt wird, um eine neue Kopie des Paketes über unser "drop-safe"-Netz für Aufzeichnung und Überprüfung zu bekommen.

9.3.2. Die "to" Methode

Die "dup-to"-Methode hat einen unmittelbaren Nachteil. Wenn es eine Kopie eines Paketes machen und dieses für das neue Ziel modifizieren soll, wird es eine Weile brauchen, um die ganze Arbeit zu tun und mit dem nächsten Paket zu dealen, das über das IPFilter-System hereinkommt.

Wenn wir nicht darauf achten, daß das Paket die Firewall zu seinem normalen Ziel passiert und wir das irgendwie blockieren, können wir das "to"-Schlüsselwort verwenden, um das Paket hinterher in die normale Routingtabelle zu drücken und es dahin treiben, daß es über ein anderes Interface als im Normalfall den Rechner verläßt.

```
block in quick on xl0 to ed0 proto tcp from any to any port < 1024
```

Wir benutzen "block quick" für das Interface-Routing; weil, wie fastroute, der "to-interface"-Code zwei Paketpfade durch IPFilter generieren wird, die mit "pass" benutzt werden und diese Dein System in Panik versetzen werden.

Viel Spaß damit!