OpenBSD Packet-Filter-FAQ (Version 1.0a)

Packet Filter (PF):

OpenBSD besitzt einen eigenen, fest im Kernel integrierten Paket-Filter. Mit diesem ist es möglich in einer speziellen Skriptsprache, den Netzwerk-Traffic zu filtern und darüber hinaus auch *Network Address Translaton* und *Quality of Services* zu betreiben. PF ist eine OpenBSD-eigene Entwicklung und wird mittlerweile auch von den anderen beiden BSD-Varianten unterstützt.

Starten von PF:

Dies wird einfach durch das setzen folgender Zeile in der Datei /etc/rc.conf veranlasst:

pf=YES

Dies veranlasst den Kernel, beim nächsten Bootvorgang den PF zu starten und die Filter-Regeln aus der Datei /etc/pf.conf (sofern nicht eine andere angegeben wurde) zu lesen. In dieser Datei befinden sich alle Angaben über Netzwerkkarten, Protokolle, Zugriffsrechte und Vorgehensweisen für den Paket-Filter. Es ist aber auch möglich, den Paket-Filter nachträglich zu starten / zu beenden:

pfctl -e

pfctl –d

Danach lädt man das Regelwerk mit: **pfctl –R –f /etc/pf.conf**.

PF-Konfiguration:

Die gesamte Konfiguration des PF und seiner Eigenschaften (Filtern, NAT & QoS) wird in einer einzigen Datei erledigt: **/etc/pf.conf**. Natürlich ist es möglich, dem PF mitzuteilen, dass er, anstatt der Standard-Datei, eine andere lädt. Doch in diesem Text werden wir uns mit den Standard-Dateien beschäftigen, da es zu keinen Änderungen am Verhalten von PF oder gar des Systems selbst führt, wenn wir eine andere Datei als Konfigurationsquelle benennen. Eine typische pf.conf besteht aus folgenden Teilen, die *nicht vorhanden sein müssen,* und wenn doch, dann *nur in dieser Reihenfolge*:

Macros & Lists — Dies sind benutzerdefinierte Größen die stellvertretend für z.B. eine bestimmte Menge von IP-Adressen, Netzwerkkarten und Ähnlichem sein können. Ein Beispiel wäre das Setzen eines leichter zu merkenden Namens für die interne Netzwerkkarte → **interne_karte="de0"**

Dabei ist es zu beachten, dass Makros & Listen nur aus Groß- und Kleinbuchstaben, Zahlen und Unterstrichen bestehen können und nicht mit einer Zahl anfangen dürfen. Des Weiteren dürfen sie nicht Schlüsselwörter der Konfigurationssprache des PF selbst sein.

Tables — Eine spezielle Struktur, dafür entworfen um große Mengen an IP-Adressen zu beinhalten. Vor allem dann anwendbar, wenn es darum geht mehr als 50 IP-Adressen zu speichern. Diese Konstruktion wird wesentlich schneller abgearbeitet als die obengenannten Makros. Jedoch kommt die Stärke dieses Konstruktes dann zu Tage, wenn es sich um wirklich große - Mengen an zusammengehörenden IP-Adressen handelt.

Options — Verschiedene eingebaute Paket-Filter Optionen, um das Verhalten von PF zu steuern. Beispielsweise kann man mit den Optionen die Timeouts einer Verbindung setzen, das Log-

Interface bestimmen, die Limits für die maximale Anzahl von Zuständen in der Paket-Filter Tabelle vordefinieren usw.

Scrub Regeln zum Wiederbearbeiten von fragmentierten Paketen und derer Zusammensetzung. Mit dieser Regel ist es möglich festzulegen, was mit den fragmentierten und / oder abnormalen Paketen geschehen soll. Man kann sie automatisch verwerfen oder aber versuchen, sie so zusammen zu setzen, dass sie ein gültiges Paket aufbauen.

Queueing Mit Queuing-Regeln erhalten wir die Möglichkeit, die vorhandenen Ressourcen nach verschiedenen Gesichtspunkten in einem Netzwerk zu verteilen. Dabei stehen und verschiedene Algorithmen und Verfahren zur Verfügung. Das Setzen von Regeln in dieser Sektion wirkt sich unmittelbar auf das aus, was unter dem Namen *Quality of Services* besser bekannt ist.

Translation Hiermit werden verschiedene IP-Adressen übersetzt und Datenpakete umgeleitet. Nach den Filter-Regeln sind es vor allem diese, die in dem Entwurf einer Sicherheitskomponente in Betracht gezogen werden. Durch das Umleiten von Datenpaketen, kann man eine Attacke schon im Voraus vereiteln, da der Angreifer einfach nicht mehr wissen kann, wo das Paket sein Ziel erreicht hat. Durch die Adressübersetzung erhalten wir die Möglichkeit, viele Rechner hinter einer einzigen IP-Adresse zu verstecken, sodass eine Attacke nicht mehr gegen mehrere Rechner durchgeführt werden kann, sondern nur noch gegen den Rechner, der seine IP-Adresse

den anderen zur Verfügung stellt. Und dieser ist meistens die Firewall selbst.

Filter Rules

Und letztendlich findet man in der pf.conf die allgemeinen Filter-Regeln. Mit diesen legt man fest, welche Adressen kontaktiert werden dürfen, über welche Protokolle und Ports. Und umgekehrt gilt es genauso. Hier sind die Filtermöglichkeiten nahezu unendlich kombinierbar, da nicht nur über IP-Adressen gefiltert werden kann, sondern auch über Protokollversion, Protokollart, Portnummmer und die Dienstart. Diese Eigenschaften sind miteinander frei kombinierbar. Beim Entwerfen der Regeln ist folgendes zu beachten: *Die letzte zutreffende Regel, ist diejenige die ausgeführt wird*. D.h., sollte man am Anfang der Filter Regeln deklariert haben, dass eine bestimmte IP-Adresse einen bestimmten Dienst auf einem Server nicht kontaktieren darf und am Ende des Regelwerkes aber eine Regel steht, die besagt, dass jede beliebige IP-Adresse auf diesen Dienst zugreifen darf, dann wird die letzte Regel Vorrang vor der ersteren haben.

Diese Bestandteile des Paket-Filters werden im Verlauf des Dokumentes eingehend besprochen und mit Beispielen versehen. Am Ende soll als Ergebnis eine funktionsfähige und leicht transportierbare Steuerungsdatei entstehen.

Paket-Filter Kontrolle:

Mit folgenden Befehlen wird der Paket-Filter gesteuert:

pfctl –f /etc/pf.conf → lädt die Steuerungsdatei in den Paket-Filter

pfctl –nf /etc/pf.conf → liest und wertet die Datei aus, ohne sie zu laden (gut für die Fehlerkontrolle)

pfctl –Nf /etc/pf.conf → lädt nur die NAT-Regeln von der Datei (Translationsregeln)

pftcl –Rf /etc/pf.conf → lädt nur die Filter-Regeln

pfctl –sn → Zeigt die aktuellen NAT-Regeln an

pfctl –sr → Zeigt die aktuellen Filter-Regeln an

pfctl –ss → Zeigt die aktuelle Zustandstabelle an

pfctl –si → Zeigt die aktuelle Filter-Statistik und die Zähler an

pfctl –sa → Zeigt allen an, was angezeigt werden kann (Kann sehr lang sein!)

Aufbau der /etc/pf.conf

1.) Listen und Makros

Unter einer Liste versteht man eine Menge von Objekten die mit einem gemeinsamen Kriterium verbunden sind. Dies kann Portnummer, Dienst, IP-Adressblock oder die Zugehörigkeit zu einem bestimmten Gerätetyp sein. Das Beispiel für eine Liste kann die Zusammenfassung von Nicht-Routerbaren Dateien in einer Block-Regel sein:

block in on dc0 from { 10.0.0.0/8, 192.168.0.0/16 } to any

Die Liste selbst wird durch eine geschweifte Klammer eingeleitet und endet mit auch mit einer solchen. Die Menge der Mitglieder einer Liste ist beliebig. Sie müssen nur vom gleichen Typ sein. In diesem Beispiel handelt es sich um IP-Adressblöcke die nur mit anderen Blöcken oder aber auch mit IP-Nummern kombinierbar sind. Eine Kombination mit z.B. Diensten wie **ssh** oder **http** ist nicht möglich. Bei der

Auswertung einer solchen Liste wird die obige Zeile in folgende zwei Zeilen expandiert:

block in on dc0 from 10.0.0.0/8 to any

block in on dc0 from 192.168.0.0/16 to any

Makros

Makros sind, wie oben angedeutet, benutzerdefinierte Variabeln die folgendes beinhalten können: IP-Adressen, Portnummern, Dienstnamen, Netzwerkgeräte-Namen und die Paket-Filter-Schlüsselwörter. Man benutzt Makros vor allem dann, wenn es darum geht, komplexe Regeln für ein System zu schaffen, dass in ständigem Wandel begriffen ist und somit man eine kleine Änderung im System nicht mit einer stupiden Schritt-für-Schritt Umbenennung von Regelwerk-Bestandteilen erledigen muss. Das beste Beispiel ist der Einbau einer neuen Netzwerkkarte anstelle der alten. Solle man die alte Karte nicht in ein Makro gepackt haben (z.B. externe_karte=“xl0“), ist man nach dem Einbau der neuen gezwungen, überall im Paket-Filter-Code nach den Einträgen der alten Karte zu suchen und diese mit dem Namen der neuen zu ersetzen. Durch das setzen von Makros erspart man sich diese Arbeit. Daher sollte man Makros so oft wie möglich benutzten. Ein Beispiel für ein Makro wäre die Zusammenfassung aller vorhandenen Netzwerkkarten in unserer imaginären Firewall:

Karten = "{ xl0, xl1, xl2 }"

Dabei ist zu beachten, dass man die geschweifte Klammer erst dann benutzt, wenn mehr als ein Element in der Menge vorhanden ist. Sonst werden die Gänsefüßchen angewandt. Es ist auch möglich Makros innerhalb von Makros zu setzen. Dabei ist zu beachten, dass die Makros im Inneren eines anderen Makros erst dann expandiert werden können, wenn sie sich innerhalb von Gänsefüßchen befinden. Hier ein Beispiel:

interne_karte = "xl0“

externe_karte = "xl1"

dmz_karte = "xl2"

karten = "{ " $interne_karte $externe_karte $dmz_karte " }"

Dabei ist auf jeden Fall zu beachten, dass in solchen Fällen keine Kommas zwischen den Makros gesetzt werden dürfen. In diesem Beispiel fällt sofort auf, dass zuvor definierte Makros mit einem $-Zeichen eingeleitet werden. Das $-Zeichen wird dann benutzt, wenn Makros ausgewertet werden sollen (d.h. wenn diese expandieren). Bei der Definition eines Makro nennt man seinen Namen und teilt ihm einen Wert zu und bei seiner Expansion wird das Makro mit einem $-Zeichen eingeleitet. Daher habein wir im letzten Beispiel die Makros mit einem $-Zeichen versehen, da wir in dem sie umschließenden Makro ihren Wert haben wollten und nicht die Namen der wertetragenden Makros.

Tabellen

Tabellen sind eigentlich nichts anderes als Listen die speziell dafür entworfen sind, um große Mengen an IPv4- und IPv6-Adressen zu speichern. Im Gegensatz zu Listen verbrauchen sie weniger Hauptspeicher und sind daher vor allem dann interessant, wenn es sich um wirklich umfangreiche Adress-Mengen, wie z.B. Mail-Spammer-IPs handelt. Tabellen können, nachdem sie deklariert worden sind, in Filter-, Scrub-, Redirect- und NAT-Regeln benutzt werden. Sie können jedoch nicht das Ziel bei einer NAT-Regel sein oder in Routing-Optionen angewandt werden. Eine Tabelle wird in /etc/pf.conf mit der **table**-Direktive erzeugt:

table <allowed_hosts> { 1.2.3.4, 1.2.3.5, 1.2.3.6, 1.2.3.7 }

Eine Tabelle besteht immer aus einem Schlüsselwort **table**, gefolgt von einem Tabellennamen, der in zwischen spitzen Klammern angegeben wird, sowie einem Tabelleninhalt der sich zwischen den geschweiften Klammern befindet. Es ist durchaus möglich, eine leere Tabelle zu deklarieren, die später mit neuen Werten aufgefüllt werden kann. Beispielsweise bei der Filterung von Spam-Mails oder Nmap-Attacken. Eine solche Tabelle bleibt fest im Speicher verankert und wird nicht, wie es

standardmäßig vom Paket-Filter veranlasst wird, gelöscht. Mit diesem Aufruf erzeugt man eine zunächst leere Tabelle:

table <spam_hosts> persist

Durch die Anwendung des Schlüsselwortes **persist** veranlassen wir den Paket-Filter, für diese Tabelle einen festen Speicher freizulegen und sie selbst dann, wenn sie leer ist, nicht zu löschen. Danach können wir mit folgenden PF-Kontrollkommandos diese Tabelle mit Informationen versorgen:

pfctl –t spam_hosts –Tadd 1.2.3.8

Hiermit teilen wir dem Paket-Filter mit, dass er in die Tabelle <spam_hosts> eine neue Adresse aufnehmen soll. Wollen wir uns jetzt von Paket-Filter anzeigen lassen, welche IP-Adressen bereits in dieser Tabelle vorhanden sind, wenden wir folgenden Befehl an:

pfctl –t spam_hosts –Tshow

Als Ergebnis würden wir in unserem Fall die vorhin eingegebene Adresse rausbekommen. Natürlich kann man auch eine Liste von bestimmten (vielleicht fälschlicherweise aufgenommenen) Adressen reinigen. Dazu bedient man sich der folgenden Befehlskombination:

pfctl –t spam_hosts –Tdel 1.2.3.8

Damit ist unsere spam_hosts-Tabelle um eine Adresse ärmer (und in diesem Falle auch leer).

Es ist ebenfalls möglich logische Operatoren anzuwenden, wenn es darum geht, bestimmte Adressen oder Adressbereiche zu filtern bzw. diese hinzuzufügen. Wenn wir z.B. in unsere Tabelle <spam_hosts> auf keinen Fall einen bestimmten Adressbereich hinzugefügt haben wollen, weil wir uns z.B. ganz sicher sind, dass von dort aus keine Spam-Mails ankommen, dann müssen wir so vorgehen, um sicher zu stellen, dass diese IP-Nummer nicht durch den Reißwolf geschickt werden:

table <spam_hosts> { 1.2.4.9, !1.2.3.0/24 } persist

Damit haben wir sichergestellt, dass erstens die Adresse 1.2.4.9 auf alle Fälle in die Tabelle reinkommt und zweitens, dass alle Adressen aus dem C-Klasse Netzwerk 1.2.3.0 nicht hierher übertragen werden dürfen. Die Notation für die früher als C-Klasse-Netzwerk bekannte Adressaufteilung wird im Paket-Filter mit der neueren CIDR-Notation durchgeführt. Man schreibt hinter der Nummer des Netzwerks nicht mehr die ältere Netzmaske (wie z.B. 255.255.255.0), sondern gibt die Anzahl der besetzten Bits an. In diesem Fall sind es 24. Diesen Wert trennt man von der Netzwerknummer mit einem /-Zeichen. Wer Probleme mit der CIDR-Notation hat, wird sicherlich in dem Tool **cidr** eine gute Hilfe finden. Man kann das Paket direkt von ftp.openbsd.org herunterladen oder aber unter /usr/ports/net/cidr selber kompilieren und installieren. Mit diesem Tool kann man ganze Netzwerkadressbereiche zwischen der alten Schreibweise und der neuen (und wesentlich flexibleren) übersetzen.

Optionen:

Optionen eignen sich dafür, das Verhalten von PF zu regulieren. Sie werden mit dem Schlüsselwort **set** eingeleitet, gefolgt von der Option, die gesetzt werden soll. Folgende Optionen können bei dem aktuellen Paket-Filter gesetzt werden:

set block-policy → hiermit setzt man die Art des Blockens der unerwünschten Pakete fest. Folgendes ist definierbar: **drop** und **return** Man kann ein Paket entweder stillschweigend fallen lassen (drop) oder aber der Gegenseite mitteilen, dass der gewünschte Dienst nicht erreichbar ist (return).

set limit → Hier stehen zwei Optionen zur Auswahl: **frags** und **states**. Mit *frags* setzt man die maximale Anzahl von empfangenen, fragmentierten Paketen fest, die man zusammensetzen soll. Standard bei PF sind 5.000 *frags*. Mit *states* setzt man die Anzahl an maximalen gleichzeitigen Zuständen im Speicher fest. Standard sind 10.000 Zustände(d.h. Verbindungen)

set loginterface → Hiermit setzt man ein Netzwerkgerät zur statistischen Analyse des Datenflusses fest. Dabei ist zu beachten, dass nur ein Netzwerkgerät zur gleichen Zeit eingesetzt werden kann.

set optimization → In diesem Abschnitt stehen uns vier vordefinierte, für besondere Umgebungen optimierte, Werte zur Verfügung. Mit dieser Option setzt man die Optimierung der einzelnen Verbindungen fest.

set optimization normal → Standard und für die meisten Netzwerke ausreichend.

set optimization high-latency → Für Netzwerke mit Satellitenverbindung

set optimization aggressive → Für eine schnellere und weniger kompromissbereite Verwaltung der einzelnen Zustände in der Verbindungstabelle des Paket-Filters. Hiermit kann man sehr viel an Speicher sparen. Hier werden die langsameren Verbindungen schneller vom Paket-Filter abgebrochen.

set optimization conservative → Dies ist das Gegenstück zu der aggressiven Optimierung. Bei dieser wird eben versucht, jede vorhandene Verbindung so lange aufrecht zu erhalten, wie es nur geht. Dabei wird der Speicher stärker in Anspruch genommen und die Prozessorauslastung steigt ebenfalls an.

set timeout → Hiermit setzt man die Anzahl von Sekunden fest, bist man die ausgelaufenen Verbindungen bzw. deren Zustände aus dem Speicher löscht. Zugleich kann diese Option auch zur Behandlung von fragmentierten Paketen angewandt werden, indem man setzt, wie viele Sekunden vergehen sollen, bis ein Fragment nicht mehr angenommen wird.

set timeout interval → Es werden Intervalle (Pausen) zwischen den Löschaktionen gesetzt.

set timeout frags → Es werden Sekunden gesetzt bin ein Fragment als solches nicht mehr angenommen wird.

Scrub-Regeln

Scrub-Regeln werden für die Normalisierung von Paketen angewandt. Vor allem dann, wenn mehrere Datenpakete über ein Interface ankommen und nicht die vordefinierte Größe aufweisen. In einem solchen Falle wird der Paket-Filter, durch die Scrub-Direktive angewiesen, versuchen, solche Datenpakete wieder in ihre Originalstruktur zusammenzusetzen, damit sie als ein Datenpaket verarbeitet werden können. Um ein standardmäßiges Scrubben auf allen Netzwerkgeräten einzuschalten genügt die folgende Direktive:

scrub in all

Hiermit werden alle (**all**) eingehenden (**in-Pakete**) neu zusammengesetzt (**scrub**). In einigen Fällen ist es aber nicht von Vorteil, Datenpakete neu zusammen zu setzen, wie z.B. wenn man per NFS mit anderen nicht-OpenBSD-Maschinen kommuniziert und diese ein sogenanntes do-not-fragment-Bit in ihre Datenpakete setzen. Im Falle einer eingeschalteten scrub-Direktive würde der Paket-Filter die fragmentierten Pakete wegwerfen. Um dies zu verhindern und auch um einige weitere Möglichkeiten haben zu können, verfügt die scrub-Direktive über verschiedene Optionen, die in bestimmten Fällen aktiviert werden können.

no-df → Hiermit wird das *do-not-fragment*-Bit aus dem Paketheader entfernt, damit der Paket-Filter ihn nicht verwirft. Da es aber bei einigen Betriebssystemen vorkommt, dass neben dem *do-not-fragment*-Bit auch ein *zero-IP-Authentifikationsheader* generiert wird, ist die Benutzung der **no-df**-Option zusammen mit der unten angeführten **random-id**-Option empfohlen.

random-id → Damit wird das Identifikationsfeld der ausgehenden Datenpakete mit Zufallswerfen aufgefüllt, um die Tatsache zu kompensieren, dass es einige Betriebssysteme gibt, die vorhersagbare Zahlenwerte

erzeugen. Diese Option wird aber nur bei ausgehenden Datenpaketen, die nach der Neuzusammensetzung nicht fragmentiert sind, durchgeführt.

min-ttl *Wert* → Setzt das Minimum an Time-To-Live (TTL) Zahlenwerten in IP-Paketheadern.

max-mss *Wert* → Setzt das Maximum eines *Maximum Segment Size* (MSS) in einem IP-Paketheader.

fragment reassemble → Diese Option veranlasst, dass die eingehenden Paketfragmente in einem Puffer abgelegt, dann neu zusammengesetzt und erst dann zum Filter-Mechanismus weitergeleitet werden sollen. Der Vorteil dieser Option ist die Tatsache, dass die Filterregeln nur noch mit ganzen Datenpaketen arbeiten müssen und Fragmente unbeachtet lassen können. Der Nachteil liegt in einem höheren Speicherverbrauch, der daraus resultiert, dass man für eingehende Fragmente einen bestimmten Speicherblock bereitstellen muss. Dies ist das Standardverhalten, wenn keine **fragment**-Option gesetzt wird. Übrigens ist es auch die einzige **fragment**-Option, die mit NAT zusammenarbeitet.

fragment crop → Alle doppelt vorkommenden Fragmente und überlange Datenpakete werden verworfen. Im Unterschied zu **fragment reassemble** werden die Fragmente nicht in einen Puffer zwischengespeichert, sondern werden, sobald sie antreffen, weitergeleitet.

fragment drop-ovl → Dem *fragment crop* ähnlich, wird hiermit der Paket-Filter angewiesen, alle doppelt vorkommenden Fragmente und überlange Datenpakete zu verwerfen, sowie diesen Fragmenten entsprechende Datenpakete, die in der Zukunft ankommen würden.

Ein Beispiel für die Anwendung der scrub-Regel ist folgende Zeile:

scrub in on $externe_karte from any to any fragment reassemble min-ttl 10 max-mss 1300

Hiermit weisen wir den Paketfilter an folgendes zu tun:

1.) alle fragmentierten Pakete die über das Interface $externe_karte (d.h. der Wert in der Variable ist das Interface)
2.) hineinkommen bzw. hinausgehen (*from any to any*),
3.) neu zusammen zu setzen (*fragment reassemble*),
4.) wobei der Mindestwert für Time-To-Live 10
5.) und der Maximalwert für MSS 1300 betragen soll.

Queuing

Unter Queuing versteht man ein Verfahren zum Ausgleich von Lastspitzen in Netzwerksegmenten. Queuing wird bei der Umsetzung von Quality-of-Service Konzepten benutzt, um verschiedene Datenpakete in Gruppen zu unterteilen und diesen bestimmte Prioritäten hinsichtlich ihrer Verarbeitung und Beförderung zu gewährleisten. Queuing ist der Versuch, zeitsensitive Daten möglichst schnell an ihren Bestimmungsort zu bringen. Um dies zu erreichen, bedient man sich der Technik und der strategischen Regeln, die dafür sogen sollen, dass möglichst wenig Datenstau produziert wird. Unter OpenBSD werden zwei Arten von Queuing angewandt: ***Class Based Queuing*** und ***Priority Queuing***.

Class Based Queuing

Bei dieser Art von Queuing wird die Netzwerk-Connection in mehrere Warteschlangen (Classes) unterteilt, die nach Quell - / Zielort, Portnummer, Protokoll usw. aufgeteilt sind. Dabei ist es möglich, dass eine Warteschlange von einer anderen übergeordneten Warteschlage des gleichen Typs die Bandbreite übernimmt, falls diese nicht voll ausgelastet ist. Eine Warteschlange kann über einer anderen höhere Priorität aufweisen, wenn sie z.B. eine interaktive Sitzung (ssh z.B.) überträgt. CB-Queues werden in hierarchischer Art verwaltet. Ganz oben befindet sich die Quell-Queue, die die maximale Bandbreite definiert, die sie zugeteilt bekommen hat.

Diese wird dann zwischen den anderen, zu ihr gehörenden Queues aufgeteilt. Als Beispiel definieren wir eine Quell-Queue von maximal 10Mbit Bandbreite, mit drei, ihr zugehörigen, Queues:

Quell-Queue (10Mbit/s)

Queue 1 (2Mbit/s)

Queue 2 (6Mbit/s)

Queue 3 (2 Mbit/s)

Diese Hierarchie kann man weiter unterteilen. Es ist möglich, diese vordefinierten Queues unter den verschiedenen Benutzern so aufzuteilen, dass der Traffic des einen nicht den Traffic des anderen Benutzers belastet. Beispielsweise kann man für alle Benutzer festlegen, wie viel an Bandbreite ihnen zur Verfügung stehen soll, wenn sie per SSH kommunizieren und wie viel, wenn sie FTP-Dienste nutzen:

Quell-Queue (10Mbit/s)

Benutzer A (5 Mbit/s)

ssh (3 Mbit/s)

ftp (2 Mbit/s)

Benutzer B (5Mbit/s)

ssh (3 Mbit/s)

ftp (2 Mbit/s)

Es ist ebenfalls möglich, die Queues so zu definieren, dass sie im Falle einer Belastung die nicht genutzte Bandbreite der anderen Queue übernehmen, falls diese ihr Pensum nicht voll ausgereizt hat. In der obigen Definition wäre es z.B. möglich gewesen, dem Benutzer A in der ftp-Queue durch das Hinzufügen des Schlüsselwortes **borrow**, den Paket-Filter hinzuweisen, im Falle der vollständigen Ausreizung der standardmäßig zugeteilten 2 Mbit/s, von dem Benutzer B die nicht

genutzte Bandbreite an den Benutzer A zu übertragen. Die Deklaration würde in diesem Falle so aussehen:

Benutzer A (5 Mbit/s)

ssh (3 Mbit/s)

ftp (2 Mbit/s, *borrow*)

Sollte jedoch der Benutzer B seinen Traffic-Bedarf erhöhen, findet eine Rückübertragung der Bandbreite statt.

CB-Queuing weist jeder Warteschlange einen Prioritätslevel zu. Warteschlangen mit einer höheren Priorität werden im Falle eine Überlastung des Netzwerkes bevorzugt bearbeitet. Die Prioritäten werden unter den Warteschlangen, die von der gleichen Quell-Queue abstammen, ausgewertet. Das Setzen von Prioritäten findet, ähnlich dem Setzen der borrow-Anweisung, in den Zeilen die für die Unterteilung der Bandbreiten zuständig sind, statt. Dabei wird hinter dem Zahlenwert für die Queue das Schlüsselwort **priority**, gefolgt von einem Dezimalwert, angegeben. Dabei gilt, je höher der Wert, desto höher die Priorität im Falle einer Netzwerkbelastung. Dabei ist zu beachten, dass einzelne Prioritäten zwischen den Queues innerhalb einer Deklaration (z.B. Benutzer A) keine Auswirkung auf andere Prioritäten in den anderen Deklarationen (z.B. Benutzer B) haben. Nur die einer Haupt-Queue zugehörigen Prioritäten werden im Falle einer Netzwerkbelastung untereinander ausgewertet.

Priority Queuing (PRIQ)

Priority Queuing weist mehrere Warteschlangen einem Netzwerkgerät (Interface) zu, wobei eine jede Queue einen einmaligen Prioritätslevel zugewiesen bekommt. Somit wird eine Queue mit einem höheren Prioritätslevel gegenüber einer niedriger eingestuften Queue *immer* bevorzugt bearbeitet. Die Struktur eine PRI-Queue ist immer flach, d.h. man kann keine Queues mit Unter-Queues deklarieren. Es gibt immer eine Haupt-Queue mit mehreren Teil-Queues, die sich voneinander nicht in

der maximalen Bandbreitennutzung unterscheiden, sondern in der Höhe der Priorität. Folgendes Beispiel verdeutlicht die Struktur einer PRI-Queue:

Quell-Queue (10Mbit/s)

Queue A (Priority 1)

Queue B (Priority 2)

Queue C (Priority 3)

Diese Aufteilung hat zum Ziel, dafür zu sorgen, dass die erste Queue erst einmal ihre Datenpakete übertragen kann und der Paket-Filter dann zu der nächstniedrigeren Queue übergeht. Im Falle dass eine höhere Queue ihr Vorrecht nicht nutzt, weil sie leer ist, wird automatisch zur nächsten Queue übergangen. Bei der Paketverarbeitung wird das FIFO-Prinzip (First in – First Out) angewandt. Bei der Anwendung dieser Art von Queues ist es wichtig, genau zu Planen, wie viele Queues wie zueinander stehen sollen, denn die höchste Queue wird in *jedem Falle* vor der niedrigeren bearbeitet. Dies kann bei einer konstant belasteten Queue zu einem Verwerfen der Datenpakete in den niedrigeren Queues führen.

Queue-Algorithmen

Um das Queuing realisieren zu können, werden bei dem OpenBSD-Paket-Filter spezielle Algorithmen angewandt. Deren Aufgabe ist es, dafür zu sorgen, dass Netzwerkbelasungen erkannt und so effektiv wie möglich umgangen werden, damit es nicht zu einem potentiellen Ausfall der benötigten Dienste kommt.

Random Early Detection (RED)

RED ist ein Algorithmus, dessen Aufgabe es ist, Netzwerkbelastungen zu umgehen bzw. gegen Null zu bringen, indem er dafür sorgt, dass die Queue nicht überfüllt wird. Dies wird durch das ständigeVergleichen der mittleren Werte der Queue mit dem Maximal- und Minimalwert der möglichen Aufnahmefähigkeit erreicht. Wenn der mittlere Wert der Queue nicht den Minimalwert der Aufnahmefähigkeit übersteigt, werden keine Pakete weggeworfen. Im Falle eines Mittelwerts, der größer ist als die maximale Aufnahmefähigkeit, wird der Paket-Filter veranlasst, keine weiteren

Datenpakete zu empfangen, bis der Mittelwert nicht die ursprüngliche Größe angenommen hat. Wenn veranlasst wird, Datenpakete zu verwerfen, wird *zufällig* gewählt, welche Verbindung ihre Datenpakete verwerfen soll. Die Verbindungen mit einem höheren Bandbreitenverbrauch haben die größere Chance, dass deren Pakete verworfen werden. RED ist sehr nützlich, das es die Situation einer *globalen Synchronisation* verhindert. Die globale Synchronisation ist ein Phänomen, das auftritt, wenn ein Mechanismus im TCP-Protokoll angewandt wird, das bei Paketverlust die Senderate reduziert und sie wieder erhöht, wenn keine Paketverluste mehr auftreten. Dieser Mechanismus wurde entworfen, um Stausituationen in den Routern zu vermeiden, er führt aber auch leider dazu, dass wenn ein Router in einer Stausituation Pakete von vielen TCP-Verbindungen gleichzeitig verwirft, die betroffenen Sender der Pakete zu gleichen Zeit ihre Senderate reduzieren, diese aber sofort wieder steigen lassen, wenn keine Paketverluste mehr gemeldet werden. Dies führt zur Bildung von sich wellenförmig ausbreitenden Stausituationen mit den Ruhephasen der Unterbelastung. Daher ist die Anwendung von RED zu empfehlen, da er dafür sorgt, dass Pakete vor der eigentlichen Belastung verworfen werden und somit das obengenannte Phänomen erst gar nicht auftritt.

Explicit Congestion Notification (ECN)

ECN wirkt in Verbindung mit RED, um zwei miteinander kommunizierende Hosts über etwaige Belastungen des Kommunikationspfades zu informieren. Dies wird durch das Setzen eines speziellen RED-Flags im Paketheader bewerkstelligt. Ein normalerweise angewandtes Verwerfen des Pakets wird somit nicht benötigt. Vorausgesetzt, dass der empfangene Host ECN versteht, wird er hiermit veranlasst, seinen Netzwerk-Traffic so zu reduzieren, dass es dem Traffic des anderen Hosts entspricht.

Queuing im Paket-Filter konfigurieren

Das in OpenBSD angewandte Queuing-Verfahren heißt **ALTQ** (Alternate Queuing) und ist ein fester Bestandteil des Grundsystems. Es wird direkt in der /etc/pf.conf konfiguriert und besitzt seine eigenen Schlüsselwörter und Verwaltungstools.

Das ALTQ wird mit dem Schlüsselwort **altq on** eingeschaltet und mit **queue** definiert es die Eigenschaften der einzelnen Teil-Queues.

Die Syntax von **altq** ist die folgende (fett sind die Schlüsselwörter):

altq on *interface scheduler* bandwidth *bw* qlimit *qlim* tbrsize *size* queue { *queue_list* }

interface → Das Netzwerkgerät über das die Warteschlange laufen soll

scheduler → Der Queue-Typ (CBQ oder PRIQ)

bw → Die totale Menge an Bandbreite die dem Algorithmus zugewiesen wird. Sie kann in Form von Suffixen wie: b, Kb, Mb und Gb angegeben werden oder aber auch in Prozent.

qlim → Die maximale Anzahl von Datenpakete die in der Warteschlange gehalten werden können. Dieser Parameter ist optional. Default-Wert ist 50.

size → Die Größe des token-bucket-regulators in Bytes. Wenn nicht angegeben, richtet sich der Wert an der Bandbreite des Netzwerkgerätes.

queue_list → Die Liste der Teil-Queuen die unter der Quell-Queue eingerichtet werden.

Ein Beispiel für die Aktivierung von ALTQ in /etc/pf.conf:

altq on $externe_karte cbq bandwidth 10Mb queue { std, ssh, ftp }

Hiermit haben wir eine *Quell-Queue* auf dem Netzwerkgerät *$externe_karte* mit der maximalen Größe von 10 Mbit/s vom *CBQ-Typ* und den drei Teil-Queues *std*, *ssh* und *ftp* definiert. Um jetzt die einzelnen Teil-Queues (auch Child-Queues genannt) zu beschreiben, bedienen wir uns des **queue**-Schlüsselworts. Seine Syntax ist die folgende:

queue *name* bandwidth *bw* priority *pri* qlimit *qlim scheduler* (*sched_options*) { *queue_list* }

name → Der name der Child-Queue. Dieser muss einem der zuvor angegebenen Namen in der altq-Direktive entsprechen. Dabei darf der Name nicht länger als 15 Zeichen betragen.

bw → Die maximale Menge an Bandbreite die dem Algorithmus zur Verfügung gestellt wird. Sie kann entweder mit den Suffixen b, Kb, Mb, Gb oder aber in Prozenten angegeben werden.

pri → Die Priorität der Queue. Für den CBQ-Algorithmus wird die Priorität mit den Werten zwischen 0 und 7 angegeben. Für den PRIQ-Algorithmus stehen Werte zwischen 0 und 15 zur Verfügung, wobei für beide gilt, dass 0 der niedrigste Prioritätswert ist.

qlim → Die Maximale Anzahl der Pakte die im Queue gehalten werden können. Wenn nicht angegeben, gilt der Wert von 50 als Standard.

scheduler → Der Typ der angewandt wird (CBQ oder PRIQ). Er muss derselbe sein, wie der in der altq-Deklaration vorgegebene.

sched_options → Hier können weitere Optionen angegeben werden, um das Verhalten des Queue-Typs zu manipulieren. Es stehen folgende Möglichkeiten zur Verfügung:

default → Definiert die Queue die alle Pakete, die nicht zu irgendeiner Queue zugehören, übernehmen soll.

red → Schaltet den RED-Algorithmus in dieser Queue ein.

rio → Schaltet den RED mit IN/OUT – Algorithmus in der Queue ein.

ecn → Schaltet das ECN ein. ECN impliziert den RED-Algorithmus.

borrow → Die Queue darf die Bandbreite von ihrem Parent (der übergeordneten Queue) ausleihen. Dies kann nur dann angewandt werden, wenn die Queue von Typ CBQ ist.

queue_list → Die Liste der Child-Queues unter dieser Queue. Diese kann nur dann angelegt werden, wenn die Queue vom CBQ-Typ ist.

Dem altq-Beispiel folgend, definieren wir mehrere Queues:

```
queue std bandwidth 50% cbq(default)
queue ssh { ssh_login, ssh_bulk }
  queue ssh_login priority 4 cbq(ecn)
  queue ssh_bulk cbq(ecn)
queue ftp bandwidth 1 Mb priority 3 cbq(borrow red)
```

Hiermit haben wir die zuvor angegebenen Queues *std*, *ssh* und *ftp* definiert. Die std-Queue bekommt 50% von der Parent-Queue (also 5 Mbit/s) zugewiesen und ist als default-Queue deklariert. Die nächste Queue *ssh* hat zwei Child-Queues *ssh_login* und *ssh_bulk*, wobei der *ssh_login*-Queue eine höhere Priorität zugewiesen bekommt. Die letzte Queue *ftp* hat 1 Mbit/s zur Verfügung bei der Priorität 3 und kann Bandbreite von anderen Queues ausleihen, wenn welche vorhanden ist. Als Algorithmus haben wir RED aktiviert.

Nachdem wir die Queues definiert haben, ist es notwendig, diesen auch einen Traffic zuzuweisen. Der Traffic wird zusammen mit den übrigen pf.conf-Regeln zugewiesen. Nehmen wir an, wir müssten den SSH-Traffic in eine Queue packen. So hätten wir z.B. vorgehen können:

```
pass out on $externe_karte from any to any port 22
```

Alle Pakete die in diese Regel passen, können z.B. so in eine Queue übernommen werden:

```
pass out on $externe_karte from any to any port 22 queue ssh
```

Hiermit werden alle Pakete die über das Interface $externe_karte ausgehen automatisch in die Queue *ssh* übernommen.

Es ist zu beachten, dass im Falle der *block*-Regeln alle TCP-RST und ICMP-Unreachable Pakete ebenfalls in die Queue übernommen werden.

Die obige altq-Regel definiert zwar die Queue mit dem Interface $externe_karte, jedoch ist dies keine unumgängliche Angabe und kann in Falle einer Regel-Entsprechung auf einem anderen Interface dazu führen, dass für dieses ebenfalls die Queue-Regel wirkt, d.h. diese Pakete in die Queue übernommen werden. Ein Beispiel wäre die folgende Regeldefinition:

```
altq on $externe_karte cbq bandwidth 2Mb queue { std, ftp }

queue std cbq(default)

queue ftp bandwidth 1.5Mb


pass in on $interne_karte from any to any port 21 queue ftp
```

Das Queuing findet zwar in der *altq*-Definition auf dem Interface $externe_karte statt, wird aber auch solche Pakete übernehmen, die zwar nicht über das Interface $externe_karte laufen, jedoch zur ftp-Queue gehören. In diesem Falle das Interface $interne_karte.

Unterteilung nach Type of Service

Normalerweise wird mit dem queue-Schlüsselwort eine Queue angegeben. Sollte aber ein weiterer Name angegeben werden, wird dieser für eine ToS-Queue und für TCP-ACK-with-no-payloads-Queues reserviert. Ein gutes Beispiel liefert uns SSH. Während SSH-Logins einen ToS-Paket für low-delay liefern, tun SCP und SFTP dies nicht. Dies kann vom Paket-Filter so genutzt werden, um eine Login-Verbindung von einer Nicht-Login-Verbindungen zu unterscheiden. Damit kann man die Priorität der Logins über die anderen Verbindungen setzen. Ein Beispiel dafür wäre:

pass out on $externe_karte from any to any port 22 queue (ssh_bulk, ssh_login)

Und hier noch einmal die ganze Queue-Deklaration in einem einfachen Regelwerk:

```
# Die Quell-Queue wird definiert

 altq on $externe_karte cbq bandwidth 10Mb queue { std, ssh, ftp }

# Die Child-Queues werden definiert und teilen die Bandbreite untereinander auf.

queue std bandwidth 50% cbq(default)
queue ssh { ssh_login, ssh_bulk }          # Zwei Unter-Queues in der ssh-Queue
queue ssh_login priority 4 cbq(ecn)      # Priorit  ten zwischen den Unter-Queues
queue ssh_bulk cbq(ecn)
queue ftp bandwidth 1 Mb priority 3 cbq(borrow red) # ftp-Queue mit der M  glichkeit
                                                    # Bandbreite auszuleihen.
```

NAT – Network Address Translation

Unter NAT versteht man die Art, hinter einer einzigen IP-Adresse mehrere Rechner zu mappen (zu verbergen). NAT ist vor allem dann notwendig, wenn die IP-Adressen zu knapp sind oder aber der Wunsch besteht, durch das Verbergen der tatsächlichen Anzahl der Rechner, die allgemeine Sicherheit zu erhöhen. NAT kann also zur Verkleinerung der Angriffsfläche eines Netzwerks dienen.

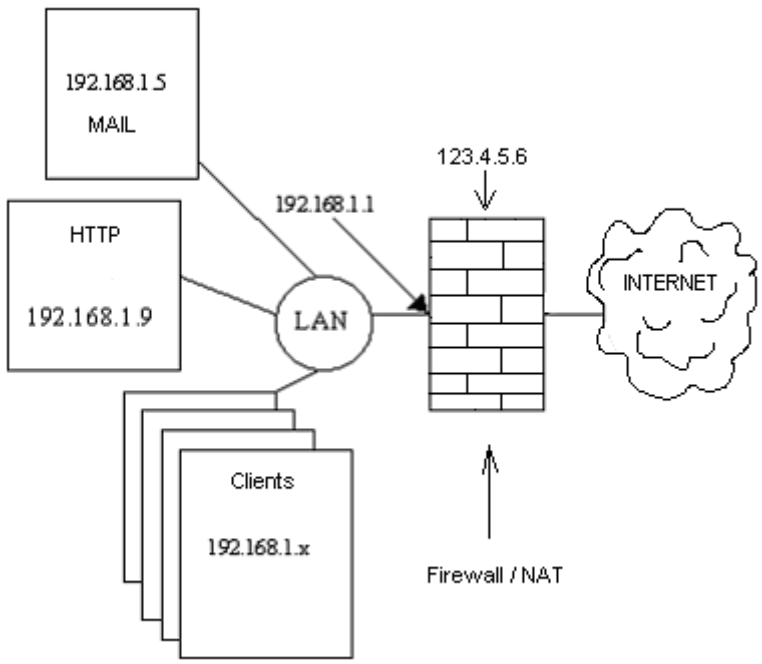
Wie funktioniert NAT?

Nehmen wir folgende Situation an:

Mehrere Rechner befinden sich im internen Netzwerk und besitzen fest zugewiesene, nicht-routerbare RFC 1918-IP-Adressen. Ein weiterer Rechner kommt hinzu, wobei dieser im Gegensatz zu den anderen, zwei Netzwerkkarten besitzt. Eine

Karte hat ebenfalls eine RFC-1918-IP-Adresse, die andere aber eine vom ISP zugewiesene, sich bei jeder neuen Einwahl ins ISP-Netz ändernde IP-Nummer. Und alle Rechner sind an einem Switch angeschlossen. Wie könnten wir vorgehen, um sicher zu stellen, dass einerseits alle Maschinen ins Internet gehen können und andererseits kein Rechner aus dem Internet auf die drei internen Maschinen zugreifen kann?

Mit NAT werden wir den am Internet hängenden Rechner (der OpenBSD als System haben muss) anweisen, die Datenpakete der internen Rechner mit den im Header befindlichen no-route-Quell-Adressen, so zu verändern, dass die Header seine externe IP-Adresse erhalten, damit diese Pakete nach draußen befördert werden können. Und wenn die Außenwelt dann antwortet, wird der externe Rechner den Prozess umkehren und die ankommenden Datenpakete auf ihre Header überprüfen und die Zieladressen so setzen, dass sie zu den eigentlichen Quellen im Innern des Netzwerkes gelangen können. NAT ist sozusagen nichts anderes, als ein Prozess, bei dem ein Rechner die Header der IP-Pakete so umwandelt, dass sie nicht mehr die Original-IPs behalten, sondern seine eigene übernehmen und der Empfänger im Glauben gelassen wird, er kommuniziere mit ein und derselben Maschine. Um es zu verdeutlichen, schauen wir uns die folgende Grafik an:



Hier sehen wir die internen Maschinen mit ihren 192.168.1.0-Adressen. Nur der Rechner vorne (der zugleich auch die Firewall selbst ist) hat eine externe IP-Adresse. Die Rechner im Innern gehören zu einem Netzwerk das sich der nicht-routerbaren IP-Nummern bedient, um vor dem Internet abgeschottet zu werden. Einige der Rechner bieten als Server mehrere Dienste wie z.B. HTTP oder MAIL an. Des Weiteren verbergen sich in diesem imaginären Netzwerk auch mehrere Workstations mit ebenfalls nicht-routerbaren IP-Nummern. Und falls ein Rechner aus dem Innern mit der Außenwelt kommunizieren möchte, dann sendet er seine Datenpakete über den Firewall/NAT-Router. Dort werden diese übersetzt (Translation), d.h. mit der Firewall-IP besetzt. Und wenn die Antworten antreffen, werden sie in umgekehrter Reihenfolge wieder übersetzt und in das Innere des Netzwerkes weiter geleitet. Um NAT unter OpenBSD zum Laufen bringen zu können, genügt es nicht nur der NAT-Regeln in der pf.conf. Man muss auch das Weiterleiten der Pakete im Kernel aktivieren, denn wir haben es hier mit unterschiedlichen Netzwerken und Netzwerkkarten zu tun und normalerweise ist ein Paket-Weiterleiten von einem in das andere Netzwerk nicht eingeschaltet. Jedoch ist das Aktivieren einer solchen Funktion keine schwierige Sache. Sie kann entweder direkt über die Kommandozeile eingegeben werden (als root):

sysctl -w net.inet.ip.forwarding=1

Oder aber permanent in der /etc/sysctl.conf eingetragen werden. Dazu muss man nur das #-Zeichen vor der Zeile: net.inet.ip.forwarding=1 entfernen. Damit wird bei jedem Neustart des Rechners ein Weiterleiten der Pakete sichergestellt. Nachdem dies erledigt worden ist, kann in der /ect/pf.conf NAT konfiguriert werden. Die Syntax der NAT-Regel ist wie folgt:

nat on *extif* [*af*] from *src_addr* [port *src_port*] to *dst_addr* [port *dst_port*] -> *ext_addr*

nat on → Schlüsselwort zum Einschalten von Adressübersetzung

extif bzw. af → Die externe Netzwerkkarte, die als einzige eine routerbare IP besitzt oder aber die Angabe eine ganzen Adressfamilie wie z.B: 192.168.0.0/16.

src_addr → Von wo aus man die zu übersetzenden Pakete erwartet.

src_port → Quellport von dem aus die Pakete in Richtung Außenwelt ankommen werden.

dst_addr → Dies ist die gewünschte Zieladresse, die über NAT kontaktiert werden soll.

dst_port → Der gewünschte Zielport.

ext_addr → Die externe Adresse die in den Headern der zu übersetzenden Pakete eingesetzt wird.

Dabei ist es wichtig den symbolischen Pfeil -> in der NAT-Deklaration zu setzen. Und im Falle, dass z.B. die externe Netzwerkkarte nicht über eine feste IP-Adresse verfügt, sondern diese über DHCP bekommt, kann man das externe Interface durch das setzen von Klammern so markieren, dass der Paket-Filter sich das Interface merkt und bei jeder Änderung der IP-Nummer die Regeln an die neue Adresse anpasst. In dem folgenden Beispiel werden wir NAT zusammen mit einem solchen Interface, das immer eine neue IP-Nummer zugeweisen bekommt, deklarieren:

```
nat on $externe_karte from $internes_netz to any -> ($externe_karte)
```

Hiermit weisen wir den Paket-Filter an, alle Datenpakete mit den im Header befindlichen internen Adressen, so umzuwandeln, das sie die IP-Adresse des Interfaces $externe_karte übernehmen, und dass dabei überprüft wird, ob das externe Interface nicht eine neue IP-Adresse bekommen hat.

Bidirektionales Mappen

Neben dem recht einfachen NAT unterstützt der Paket-Filter auch das bidirektionale Mappen, das durch die *binat*-Regel eingeschaltet wird. Diese Form der Adressübersetzung ist vor allem dann sinnvoll, wenn man für einen Rechner verschiedene Connection-Typen zu verschiedenen IP-Adressen zuweisen möchte. Denkbar wäre z.B. ein Server, der Anfragen vom Internet aus auf seine interne Adresse zugeschickt haben möchte und alle Anfragen

von ihm aus ins Internet über die externe Adresse laufen sollen. Hier ein Beispiel dazu:

server_intern = "192.168.45.67"

server_extern = "134.234.45.7"

Binat on $externe_karte from any to $server_extern -> $server_intern

Man kann diese Form des NAT dazu nutzen, den ankommenden Traffic auf verschiedene Netzwerkkarten zu verteilen. In Kombination mit Queue-Regeln ergeben sich hier vielfältige Möglichkeiten.

NAT-Ausnahmeregeln

Mit NAT kann man auch festlegen, welche internen IP-Adressen *nicht* übersetzt werden sollen. Dies erledigt man durch das hinzufügen des Schlüsselwortes **no**.

no nat on $externe_karte from 192.168.54.10 to any

nat on $externe_karte from 192.168.54.0/24 to any -> ($externe_karte)

Somit haben wir klargestellt, dass über das Interface $externe_karte eine Adressübeesetzung der Datenpakete vom internen Netzwerk 192.168.54.0/24 stattfinden soll, bis auf die Adresse 192.168.54.10.

Um den Status der Adressübersetzung in Echtzeit überprüfen zu können, benutzen wir das Mangement-Tool **pfctl**, das zur Grundausstattung der Paket-Filters gehört.

pfctl –s state

Dieses Kommando veranlasst den Paket-Filter, alle zur Zeit vorhandenen NAT-Verbindungen aufzulisten.

NAT ist in jedem Fall eine sehr nützliche Möglichkeit, einerseits IP-Adressen zu sparen, denn es gibt nicht mehr so viele IPv4-Adressen und das neue IPv6-Protokoll hat sich noch nicht so richtig durchgesetzt und andererseits ermöglicht es, ein gewisses Maß an Sicherheit durch das bloße Verbergen der Rechner zu gewährleisten. Denn je weniger Angriffsfläche (in diesem Falle IP-Adressen) man einem potentiellen Angreifer anbietet, desto weniger Einbruchsmöglichkeiten hat er letztendlich auch zum ausprobieren vorhanden. Und in Kombination mit den auf der nächsten Seite folgenden Redirection-Regeln, erweist sich NAT als eine umfangreiche Möglichkeit, Rechner und auch ihre Dienste so transparent wie möglich zu verteilen, dass eine Attacke zwar nicht unmöglich, doch auf alle Fälle sehr zeit- und ressourcenverbrauchend erscheint.

Redirection

Wenn man NAT laufen hat, hat man auch die Möglichkeit, mit internen Maschinen das Internet zu erreichen. Doch was, wenn wir den Wunsch haben, einen kleinen Webserver im internen Netzwerk aufzubauen und diesen auch für den Zugriff von Außen zugänglich zu machen. Da unser Netzwerk mit no-route-IP-Adressen ausgestattet ist, wird keine Maschine aus dem Internet den Server erreichen können. Und bevor wir uns Gedanken darüber machen, ein Adress-Pool zu kaufen, sollten wir uns dem *Redirection*-Regelwerk zuwenden. Mit Redirection können wir nämlich einen internen Rechner hinter einem NAT-Gateway erreichen bzw. die Dienste die dieser Rechner anbietet. Und Redirection wird mit folgender Befehlszeile eingeschaltet:

```
rdr on $externe_karte proto tcp from any to any port 80 -> 192.168.1.9
```

Hiermit haben wir mit dem Schlüsselwort **rdr** dem Paket-Filter befohlen, alle Datenpakete die auf den NAT-Gateway (also die einzige Maschine in unserem Netzwerk mit einer routerbaren IP) zukommen und Webdienste über Port 80 in Anspruch nehmen wollen, auf die interne Adresse 192.168.1.9 umzuleiten. Somit erreichen die äußeren Hosts unseren Webserver ohne aber direkt diesen anzusprechen. Sie schicken brav ihre Pakete an unser NAT-Gateway und dieser erkennt anhand der Angabe über den Zielport (80), von wem diese

Pakete bearbeitet werden sollen. Und wenn unser Webserver diese bearbeitet hat und eine Antwort darauf zurückschickt, antwortet der NAT-Gateway als ob er selber der Webserver wäre.

Beim Entwerfen von *rdr*-Regeln ist es aber wichtig zu wissen, dass diese ebenfalls vom Paket-Filter analysiert werden und im Falle des Fehlens einer eindeutigen *pass*-Regel geblockt werden.

Packet Filtering

Packet Filtering ist das Selektieren von Datenpaketen anhand verschiedener Kriterien, wie z.B. Adresse, Protokoll, Port und Interface. Mit den Filter-Regeln können wir letztendlich entscheiden, ob ein Paket ein bestimmtes Interface passieren, einen Dienst kontaktieren oder gar eine Antwort auf eine Anfrage liefern darf. Paket-Filter erreicht seine endgültige Stärke aber auch seine größte Schwäche im Setzen von Filter-Regeln. Alles hängt von der für die Situation und an die Umstände angepassten Konfiguration der Filter-Regeln ab. Sind die Regeln klar und scharf genug, kann der Paket-Filter ein wirkungsvolles Gerät zum Selektieren des Netzverkehrs sein. Sind aber die Regeln zu nachlässig oder gar falsch und der Situation gar nicht entsprechend formuliert worden, ist der Paket-Filter nichts mehr als trügerische Sicherheit. Daher widmen wir uns erstens der genauen Analyse der Syntax der beiden Selektions-Regeln: der ***pass***- und der ***block***-Regel:

action direction [log] [quick] on *int* [*af*] [proto *protocol*] from *src_addr* [port *src_port*] to *dst_addr* [port *dst_port*] [*tcp_flags*] [*state*]

action

Die Aktion die gestartet wird, wenn ein auf die Regel zutreffendes Datenpaket ankommt. Es gibt zwei Arten von Aktionen: *pass* und *block*. Bei einem *pass* wird das Paket zur weiteren Verarbeitung weitergeleitet, während bei einem

block das Paket gemäß der gültigen **block-policy** angehalten wird. Die standardmäßige **block-policy** kann durch das setzen von *block-drop* oder *block return* Anweisung geändert werden.

direction

Die Richtung des Pakets über das zutreffende Interface. Kann in oder out sein.

log

Deklariert, ob ein zutreffendes Paket via pflogd geloggt wird oder nicht. Wenn die gesamte Regel mit der Option keep state bzw. modulate state ausgestattet ist, wird nur das Initialpaket, das die Verbindung aufbaut, geloggt. Um dennoch alle Pakete zu loggen, kann man die Option log-all anwenden.

quick

Wenn ein Paket auf die Regel zutrifft, wird die jeweilige Regel als die letzte Regel für dieses Paket angesehen und wird nicht, wie standardmäßig vorgegeben, auf andere, möglicherweise ebenfalls zutreffenden Regeln getestet und die in der aktuellen Regel vorgegebene Aktion (*pass* oder *block*) wird durchgeführt.

int

Der Name der Netzwerkkarte durch die sich das Datenpaket fortbewegt.

af

Die Adressfamilie des Datenpakets. Kann IPv4 oder IPv6 sein und wird in der CIDR-Notation geschrieben..

protocol

Das Schicht-4- Protokoll des Datenpakets:

- tcp
- udp
- icmp
- icmp6
- Ein gültiger Protokollname aus /etc/protocols
- Eine gültige Protokollnummer zwischen 0 und 255
- Ein Set von Protokollen unter Benutzung von Listen.

src_addr, *dst_addr*

Die Quell-/Zieladresse des zutreffenden Datenpakets. Kann folgendermaßen spezifiziert werden:

- Als einzelne IPv4 oder IPv6 Adresse.
- Als CIDR-Netzwerkblock.
- Als Fully-Qualified-Domain-Name, der über DNS erfragt werden kann, nachdem das Regelwerk geladen worden ist. Alle daraus resultierenden IP-Adressen, werden in die Regel übernommen (Substituiert).
- Der Name der Netzwerkkarte. Alle IP-Adressen der jeweiligen Karte werden in die Regel übernommen.
- Der Name der Netzwerkadressen gefolgt von der Netzmaske. Jede IP-Adresse auf der Netzwerkkarte wird mit der Netz-Maske verglichen und wenn zutreffend in die Regel übernommen.
- Der Name der Netzwerkkarte zwischen den runden Klammern. Hiermit bewirkt man, dass der Paket-Filter ein Update der Netzwerkkarten-IP startet und falls eine neue IP erkannt wird, dass diese in die Regel übernommen wird.
- Der Name des Netzwerks gefolgt vom Schlüsselwort :network oder :broadcast. Das daraus resultierende CIDR-Netzwerk (z.B. 192.168.0.0/24) oder Broadcast-Adresse (z.B. 192.168.0.255) wird in die Regel übernommen, wenn das Regelwerk geladen wird.
- Als ein table, d.h. Zusammenfassung von IP-Adressen.
- Als alles Obengenannte negiert durch den Operator ! (für NOT)
- Als eine Menge von Adressen unter Benutzung von Listen.
- Als das Schlüsselwort **any**, dass alles zusammen meint.
- Als das Schlüsselwort **all**, das nichts anderes als die Kurzform für **any to any** ist..

src_port, *dst_port*

Der Quell- / Zielport in der Netzwerkschicht 4.

Ports können wie folgt spezifiziert werden:

- Als Nummer zwischen 1 und 65535
- Als ein gültiger Dienstname aus /etc/services
- Als eine Menge von Ports unter Benutzung von Listen-Konstrukten.
- Als ein Bereich:
    - != (nicht gleich)
    - < (kleiner als)
    - > (größer als)
    - <= (kleiner oder gleich)
    - >= (größer oder gleich)
    - >< (Bereich zwischen)
    - <> (Bereich nicht zwischen)

Die letzten zwei sind binäre Operatoren und besitzen keine Argumente innerhalb des Bereiches.

tcp_flags

Spezifiziert die Flags die gesetzt sein müssen, wenn das Protokoll TCP benutzt wird. Flags werden als *flags check*/*mask* implementiert. Beispielsweise: flags S/SA – Dies instruiert den Paket-Filter nur auf die S and A (SYN and ACK) Flags zu schauen und als zutreffend zu deuten, wenn der SYN Flag gesetzt ist.

state

Spezifiziert ob der Zustand der Verbindung beibehalten werdens soll.
Es gibt zwei Zustandsarten: keep state und modulate state

keep state – funktioniert mit TCP, UDP, und ICMP.

modulate state – funktioniert nur mit TCP. Paket-Filter wird hiermit komplexe Initial Sequence Numbers (ISNs) für Pakete, die auf diese Regel zutreffen, generieren.

Nachdem wir den ganzen Aufbau der Selektionsregeln verinnerlicht haben, können wir uns wagen, den Datenfluss mit neuen Regeln zu filtern. Nehmen wir z.B. an, wir wollen zulassen, dass von unserem internen Netzwerk aus alles ausgehen darf (d.h. unsere internen Maschinen dürfen überall hin), vom

Internet aus aber wollen wir nur HTTP- und MAIL-Dienste angesprochen sehen. Wie würden wir vorgehen?

Erstens definieren wir eine allgemeine Block-Regel, die als Standard-Aktion auftreten wird, wenn es keine passendere Regel für ein Datenpaket gibt. Wir geben also in die /etc/pf.conf folgendes ein:

block in all

Damit werden alle Datenpakete geblockt, die in unser Netzwerk gelangen möchten. Das Rausgehen wollen wir ja unseren Maschinen nicht verbieten. Daher haben wir **block *in* all** gesetzt. Hiermit haben wir den Paketfilter angewiesen, dass nur eingehende Pakete geblockt werden. Wir wollten aber doch zulassen, dass die Außenwelt unsere Web- und Maildienste nutzen darf? Dies ist kein Problem, denn der Paketfilter liest, wenn nicht anders angeordnet, immer die ganze Konfigurationsdatei aus und handelt nach der letzten zutreffenden Regel. Man sagt auch: **Last matching rule wins**. Daher können wir unsere erste Regel stehen lassen und zwei weitere definieren:

block in all

pass in on de0 inet proto tcp from any to 192.168.1.9 port 80

pass in on de0 inet proto tcp from any to 192.168.1.5 port 25

Somit haben wir klargestellt, dass *normalerweise* alle eingehenden Datenpakete geblockt werden, *es sei denn,* es trifft folgendes auf das Paket zu:

1.) es kommt über die Netzwerkkarte **de0,**
2.) ist dem IPv4-Protokoll zuzuordnen (inet),
3.) läuft über TCP (proto tcp),
4.) seine Quelladresse ist egal (from any),
5.) jedoch ist seine Zieladresse 192.168.1.5
6.) und der Port über den er mit diesem Rechner kommunizieren will ist 80 (also http-Dienst).

Erst dann wird ein Paket angenommen, denn sonst greift die letzte zutreffende Regel ein und das Paket wird geblockt, d.h. verworfen. In der ersten Regel (block in all) muss das Datenpaket nur ein einzige Pflicht erfüllen: es muss nämlich *eingehend* sein. In den beiden anderen, muss es mehrere Pflichten erfüllen, damit es als dieser Regel zugehörig zugerechnet wird und seinen Weg in Richtung Web-Server weiter gehen darf. Daher sollte man bei der Umsetzung von Regeln genau darauf achten, was man eigentlich erlauben will, denn es gibt sehr viele Möglichkeiten, etwas zu erlauben bzw. zu verbieten. Jedoch bekommt man zugleich auch die unerwünschte Möglichkeit, unwissentlich etwas zu erlauben, was auf keinen Fall erlaubt sein sollte. Z.B. definiert man zuerst eine Regel, die alle Zugriffe auf Mail-Server für bestimmte Adressen definiert. Und am Ende des Regelwerks wird Folgendes eingebaut:

pass in on $externe_karte from any to $mail_server port 25

In so einem Falle nützen nicht einmal 1.000 Verbotsregeln vor dieser letzten Regel. Denn der Spruch „last matching rule wins“ *gilt immer* und kann nicht umgangen werden. Es sei denn man bedient sich der einer Ausnahme, die bekanntlich die Regel bestätigt. Die Ausnahme ist das in den Selektionsregeln aufgeführte Schlüsselwort ***quick***. Mit *quick* kann man den Paket-Filter anweisen, die gerade gelesene Regel als die letzte (und somit auch als die einzig zutreffende Regel) anzusehen und sofort die Aktion auszuführen, die vorgegeben ist. Um dies zu verdeutlichen, holen wir uns die obige Konfiguration noch einmal zum Experimentieren zurück:

block in all

pass in on de0 inet proto tcp from any to 192.168.1.9 port 80

pass in on de0 inet proto tcp from any to 192.168.1.5 port 25

Jetzt setzen wir **quick** direkt in der block-Regel hinter **in** ein und lassen den Paket-Filter das Regelwerk neu laden. Und, kann man jetzt die Server kontaktieren? Natürlich nicht, denn der Paket-Filter kommt erst gar nicht dazu, die anderen beiden Regeln auszuwerten, weil jedes ankommende Datenpaket zuerst in die Block-Regel passt und die Block-Regel *zufälligerweise* dazu noch

das Schlüsselwort **quick** beinhaltet, was soviel bedeutet wie „ich treffe als die letzte Regel zu und führe die Block-Aktion durch“. Damit könnten wir jetzt unter ihr ruhig 1.000 Pass-Regeln setzen uns es würde nichts helfen, denn sie werden einfach nicht ausgewertet werden können. Daher ist auch mit diesem Schlüsselwort Vorsicht geboten. Und vor allem dann, wenn es darum geht, etwas zu erlauben. In dem obigen Beispiel haben wir etwas verboten, was wesentlich weniger Schaden bringen kann, als etwas unwissentlich zu erlauben.

Viel Spaß und Erfolg mit OpenBSD und pf!

Und vergesst nicht das OpenBSD-Projekt zu unterstützen:

http://www.openbsd.org/orders.html

Diese FAQ hatte die OpenBSD-pf-FAQ als Grundlage gehabt.
Homepage: http://www.openbsd.org/faq/pf

Kritik, Anregungen, Vorschläge u.ä. bitte an: coldbsd@yahoo.de

Spenden bitte am besten an: http://www.bsdforen.de