Crashkurs
Firewallaufbau unter Linux

Florian Kalhammer

21. März 2001

Inhaltsverzeichnis

Kapitel 1

Vorwort

Die vorliegende Dokumentation beschreibt Funktionsweise, Aufbau und Wartung eines Firewallrechners unter Linux. Sie ist als begleitendes Material für eine Schulung gedacht, nicht als Selbststudium–Hilfe. Das Thema Firewall ist ein weit gefächerter Bereich, der hier nicht abschließend behandelt werden kann, es sollen die in der Schulung und dem Aufbau vorkommenden Punkte besprochen und dargestellt werden.

Eine Firewall ist ein Sicherheitssystem, das – wie alle anderen Sicherheitssysteme auch – im Idealfall nicht genutzt werden muß, weil z.B. kein Angriff auf das Computersystem vorliegt. Wie aber bei Sicherheitssystemen üblich, nützt uns dieses System nur dann, wenn es im Bedarfsfall perfekt funktioniert. Für diese Garantie der perfekten Funktion ist aber eine Sache wesentliche Voraussetzung: Das Verständnis der zugrunde liegenden Technologie. Dieses Verständnis kann bei vielen Menschen, die mit Computern zwar zu tun haben, diese aber nicht restlos verstehen, nicht vorausgesetzt werden. Aus diesem Grund beginnt diese Beschreibung mit einem Kapitel Netzwerkgrundlagen, die für viele vielleicht übertrieben genau erscheint. Trotzdem muß – damit später die Funktionsweise der Firewall hinlänglich sicher aufgebaut werden kann – dieses Wissen vorhanden sein. . .

Die Beschreibung der Techniken bezieht sich hier auf ein Linux–System mit einem Kernel der Versionen 2.2.x, der mit *ipchains* arbeitet. In den früheren Versionen wurde statt dieser Technik ein Programm mit Namen *ipfwadm* eingesetzt, in der allerneuesten Kernelversion (2.4) wird eine neue Technik (*iptables*) verwendet. In Sicherheitstechniken sollten aber immer ausgereifte und erprobte Werkzeuge zum Einsatz kommen. Daher scheint mir *ipchains* die beste Lösung darzustellen. Der Umstieg auf die neue Technik ist – im wesentlichen – auch nur ein syntaktischer, die Regeln bleiben größtenteils die selben.

Diese Dokumentation kann eines nicht sein, eine Einführung in bzw. ein Lehrgang für Linux selbst. Es werden hier also weder die Installation, noch die grundlegenden Techniken des Umgangs mit Linux beschrieben. Eine Stand–alone–Firewall mit Linux bedarf allerdings auch nicht einer großen Verwaltungsarbeit, so daß ein durchschnittliches Wissen um Linux sicher ausreichend ist, um die hier beschriebenen Vorgänge nachzuvollziehen und reproduzieren zu können. Wo immer es notwendig ist, genaue Angaben zu machen, wo bzw. wie etwas in Linux einzustellen ist, wird es ausreichend beschrieben.

Da im vorliegenden Fall die Linux–Distribution der S.u.S.E GmbH eingesetzt wird (S.u.S.E 7.0 Professional), beziehe ich mich im wesentlichen auch auf die in dieser Distribution üblichen Dateien und Verzeichnisse. Im Großen und Ganzen sollten aber alle Techniken – mit kleinen Änderungen – auch in jeder anderen modernen Linux–Distribution implementierbar sein. Die Unterschiede werden sich im Wesentlichen auf verschiedene Startdateien reduzieren, in denen die entsprechenden Firewall–Regeln eingetragen werden. Diese Startdateien sind in fast jeder Distribution unterschiedlich, daher würde eine allgemeingültige Beschreibung den Rahmen dieser Dokumentation sicher sprengen.

Kapitel 2

Grundlagen

Der Begriff *Firewall* hat keine eindeutige Definition und keine standardisierte Bedeutung. Es handelt sich vielmehr um ein Modell zur Absicherung eines Computernetzes gegenüber anderen – meist öffentlichen – Netzen. Im Rahmen dieser Anforderung kommen – je nach Größe und Konzept des zu schützenden Netzes – verschiedene Lösungsansätze in Frage. In jedem Fall muß die gewünschte Sicherheit in verschiedenen Schichten angegangen werden, ein einziger Mechanismus, wie etwa nur eine Paket–Firewall alleine ist nie genug.

Im Allgemeinen wird beim Thema Firewall zwischen dem zu schützenden Netz (internes LAN oder auch *sicheres Netz*) und dem öffentlichen Netz (meist das Internet oder auch *unsicheres Netz*) unterschieden. Im einfachsten Fall ist die Firewall hier ein physikalischer Rechner, der mit zwei Netzwerkverbindungen (Netzwerkkarten, ISDN-Karten, Modems,...) ausgestattet ist. Eine davon ist mit dem sicheren, die andere mit dem unsicheren Netz verbunden. Der einzige Weg, der beide Netze verbindet, ist eben die Firewall. Auf diesem Verbindungsrechner werden nun Strategien implementiert, die die Sicherheit des zu schützenden Netzes – aber auch die Regeln, die beschreiben, wer im sicheren Netz was im unsicheren Netz machen darf – festlegen. Dazu zählen insbesondere:

- Welche Internet–Dienste dürfen vom lokalen Netz aus genützt werden?
- Welche Dienste darf die ganze Welt in Anspruch nehmen?
- Welche Dienste dürfen ausgewählte Benutzer oder Computer in Anspruch nehmen?
- Welche Dienste dürfen nur lokal genutzt werden?

Der grundlegende Ansatz zur Implementierung dieser Strategien ist die sogenannte Paketfilter–Firewall. Im weiteren Verlauf dieser Darstellung wird es also hauptsächlich um die Konzeption und Wartung einer solchen Paketfilter–Firewall gehen, implementiert mit einem Linux–System. Linux bietet für diese Aufgabe alle notwendigen Werkzeuge bereits im Betriebssystem–Kern, es ist also nicht notwendig, komplizierte Zusatzinstallationen vorzunehmen, die ein System nur instabiler werden lassen.

Auf der anderen Seite wurde ja schon erwähnt, daß eine hinreichende Sicherheit nur über verschiedene, aufeinander aufbauende Mechanismen herzustellen ist. Auch für diese höhergelegenen Sicherheitsstufen sind unter Linux alle Werkzeuge bereits in der Standard–Installation vorhanden. Zunächst gilt unser Augenmerk jedoch den Mechanismen des Paketfilterns. Um grundsätzlich diesen Vorgang verstehen zu können, ist es notwendig ein paar grundlegende Begriffe und Techniken der Computervernetzung zu beschreiben. Was ist überhaupt Kommunikation in Netzwerken, wie läuft sie ab und welche Mittel des Eingreifens stehen uns also zur Verfügung?

2.1 Das OSI Referenzmodell

Ende der siebziger Jahre wurde ein theoretisches Referenzmodell für die Vernetzung von Computern entworfen, das auch heute noch vielen Darstellungen zugrunde liegt. Es han-

delt sich dabei um das sogenannte OSI–Referenzmodell. OSI steht für *Open System Interconnection*, also etwa Verbindung offener Systeme untereinander. Dieses Modell ist ein Schichtenmodell mit sieben Schichten, die die Kommunikation der Computer untereinander beschreiben. Es ist sehr akademisch angelegt und wurde von keinem real existierenden Netzwerksystem vollständig implementiert. Trotzdem wird es zur Darstellung der verschiedenen Mechanismen noch heute benutzt. Daher folgt hier eine kurze Beschreibung dieses Modells:

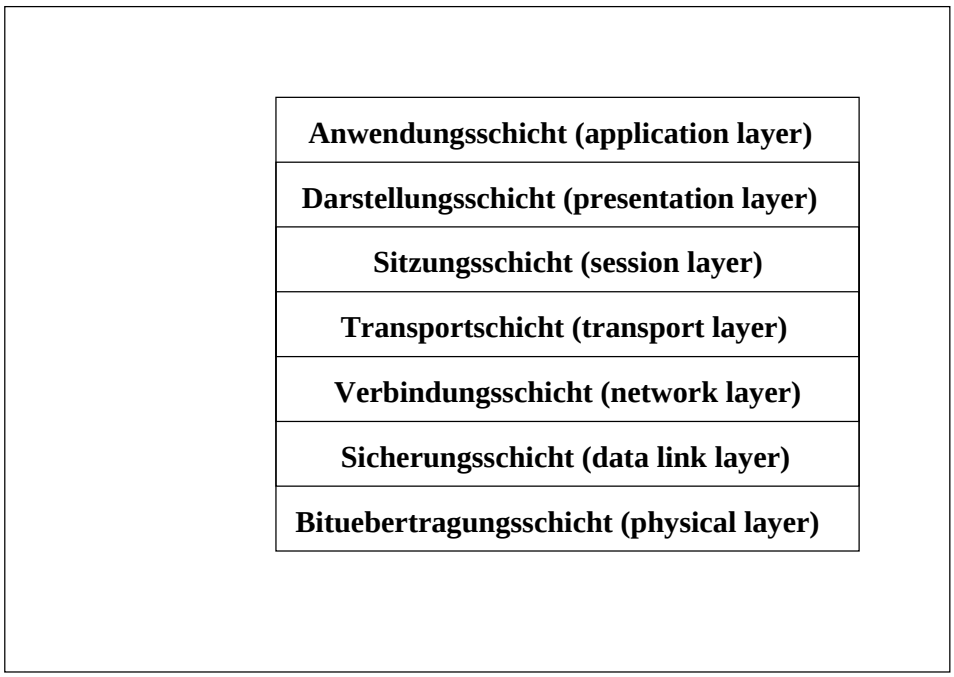

Bild 2.1: Das OSI-Referenzmodell

Die beiden untersten Schichten sind – im Falle einer herkömmlichen Netzverbindung etwa über Ethernet–Karten – auf der Netzwerkkarte selbst implementiert. Sie sind also nicht Bestandteil der Software oder des Betriebssystems.

Von unten nach oben betrachtet kommen den verschiedenen Schichten die folgenden Aufgaben zu:

2.1.1 Die Bitübertragungsschicht

Die Bitübertragungsschicht (physical Layer) ist verantwortlich für die Übertragung der einzelnen Bits, sie hat also einfach die Aufgabe dafür zu sorgen, daß wo eine 1 gesendet wurde auch eine 1 empfangen wird und wo eine 0 gesendet wurde eben eine 0 empfangen wird. Typische Fragen dieser Schicht sind die Spannungen, die jeweils einer 1 oder 0 entsprechen und wielange diese Spannungen aufrecht erhalten werden müssen, damit ein Bit gesendet wird. Diese Schicht ist immer direkt mit der Hardware verbunden, ihre Konstruktion ist also Aufgabe des Hardwareherstellers.

2.1.2 Die Sicherungsschicht

Die Sicherungsschicht (data link layer) verarbeitet die übertragenen Rohdaten indem sie eine Datenreihe daraus erstellt und diese Datenreihe gewöhnlichermaßen in einen sogenannten Datenübertragungsrahmen (data frame) packt. Die Sicherungsschicht kommuniziert mit ihrer gegenüberliegenden Sicherungsschicht durch sogenannte Quittungsrahmen, die korrekt verarbeitet werden müssen.

Um das Ganze einmal im Detail zu sehen folgt hier die Darstellung der Framestruktur eines Datenrahmens (engl. frame) der IEEE 802.3 Norm, also der Norm, die unser Ethernet benutzt:

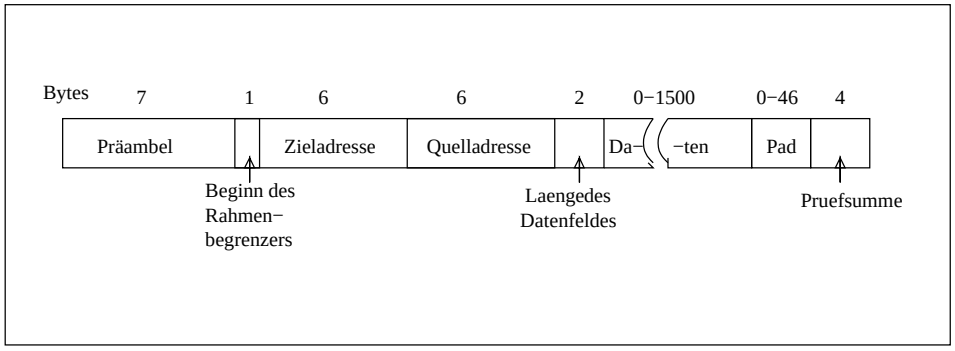

Bild 2.2: Das 802.3 Frameformat

Die beiden Adressfelder enthalten dabei die Hardwareadresse (MAC–Adresse) der jeweiligen Netzwerkkarten. Das Datenfeld enthält die eigentlichen Daten, falls es weniger als 46 Byte sind, so wird das Pad–Feld benutzt um die (physikalisch notwendigen) 46 Byte aufzufüllen. Die Prüfsumme ist schließlich eine Möglichkeit, Fehler bei der Übermittlung direkt zu erkennen und ein fehlerhaftes Paket gegebenenfalls nochmal anzufordern.

2.1.3 Die Vermittlungsschicht

Diese Schicht (Network Layer) steuert die verschiedenen Transportwege innerhalb des Netzes. In kleinen LAN ist diese Schicht also verhältnismäßig dünn, in großen Netzen (z.B. dem Internet) hat sie sehr umfangreiche Aufgaben. Die Wahl der verschiedenen möglichen Routen vom Sender zum Empfänger, das optimale Ausnutzen der bestehenden Verbindungen und ähnliches sind ihre Aufgaben.

2.1.4 Die Transportschicht

Die Aufgabe der Transportschicht (Transport Layer) ist es, den Datenstrom von der Sitzungsschicht zur Vermittlungsschicht weiterzugeben, eventuell in kleinere Pakete zu packen und dafür zu sorgen, daß alle Teile in der korrekten Reihenfolge ankommen. Die Transportschicht baut eine direkte Kommunikation zwischen Sender und Empfänger auf, und arbeitet mit einem entsprechenden direkten Protokoll. Man spricht hier von einer Ende-zu-Ende Schicht, im Gegensatz dazu waren die bisher besprochenen Schichten sogenannte gekettete Schichten. Das heißt, daß sie nicht unbedingt eine direkte Kommunikation mit der Zielmaschine betrieben, sondern nur mit den zwischen Sender und Empfänger liegenden Zwischenstationen (Gateways, Bridges,...)

Viele Computer arbeiten heute im Multitasking-Betrieb, es kann also sein, daß zwischen zwei Rechnern mehrere unabhängige Netzwerkverbindungen bestehen. Auch die Koordination dieser Tatsache ist Aufgabe der Transportschicht.

2.1.5 Die Sitzungsschicht

Die Sitzungsschicht (Session Layer) hat ähnliche Aufgaben wie die Transportschicht, jedoch in einer etwas gehobeneren Form. Sie muß zum Beispiel abgebrochene Transferversuche an der Stelle wiederaufnehmen, an der sie abgebrochen wurden. Insgesamt könnte man es damit zusammenfassen, daß die Sitzungsschicht zuständig für die Dialogsteuerung ist.

2.1.6 Die Darstellungsschicht

Die Darstellungsschicht (Presentation Layer) ist zuständig für die korrekte Darstellung der Daten. Diese Schicht wird z.B. dadurch notwendig, daß verschiedene Computer verschiedene Formen der Darstellung für Ganz- und Kommazahlen haben. Es gibt etwa die sogenannten Big-Endian Systeme, die in einer 16-Bit Ganzzahl das höherwertige Byte zuerst anführen, während die Little-Endian Systeme es genau anders herum machen. Würden Computer miteinander kommunizieren, die diese Darstellung unterschiedlich handhaben, so kämen zwar Ganzzahlen am Zielrechner an, er würde sie aber gänzlich falsch interpretieren.

2.1.7 Die Anwendungsschicht

Die letzte Schicht des OSI-Modells (Application Layer) enthält nun die verschiedenen Anwendungen, die sich gegenseitig über das Netz verständigen. Diese Schicht ist in der Regel direkt in das Anwenderprogramm programmiert und enthält die jeweiligen Protokolle, mit denen die Programme kommunizieren.

2.2 Das TCP/IP Referenzmodell

Nachdem das OSI-Modell ein eher akademisches Modell ist und von keiner Netzwerksoftware wirklich hundertprozentig implementiert wird, ist es jetzt an der Zeit, das Modell zu betrachten, das tatsächlich heute im Internet (und allen lokalen Netzsystemen) verwendet wird, TCP/IP. Im Gegensatz zum OSI-Modell kommt TCP/IP mit nur vier Schichten aus, um den Datentransfer zu erklären:

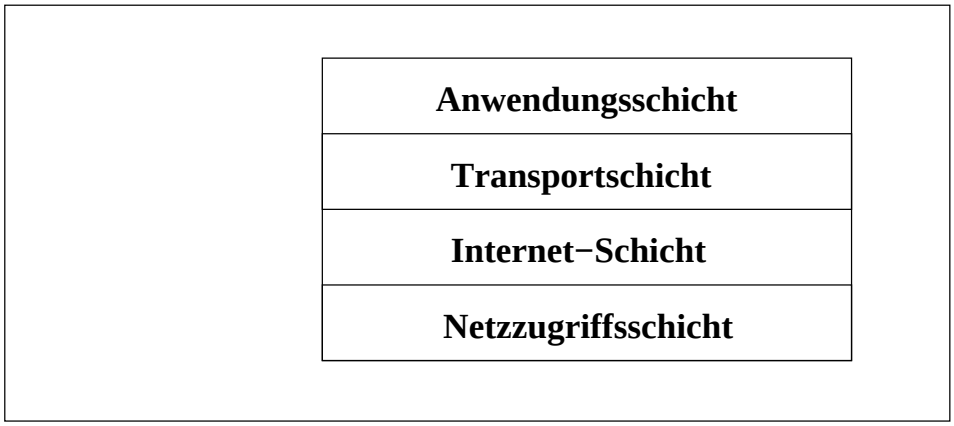

Bild 2.3: Das TCP/IP-Schichtenmodell

Diese vier Schichten kommunizieren nicht genau mit denen des OSI–Modells, vereinfacht kann man sagen, daß die Netzzugriffsschicht die beiden untersten Schichten des OSI–Modells enthalten, die Internetschicht entspricht genau der Vermittlungsschicht, die Transportschicht von TCP/IP erfüllt im Wesentlichen die Aufgaben der Transportschicht des OSI-Modells und die restlichen drei Schichten von OSI sind in der Anwendungsschicht von TCP/IP integriert.

Auf all diesen vier Schichten arbeiten unterschiedliche Protokolle zur Datenübertragung. Vereinfacht könnte das etwa wie folgt dargestellt werden:

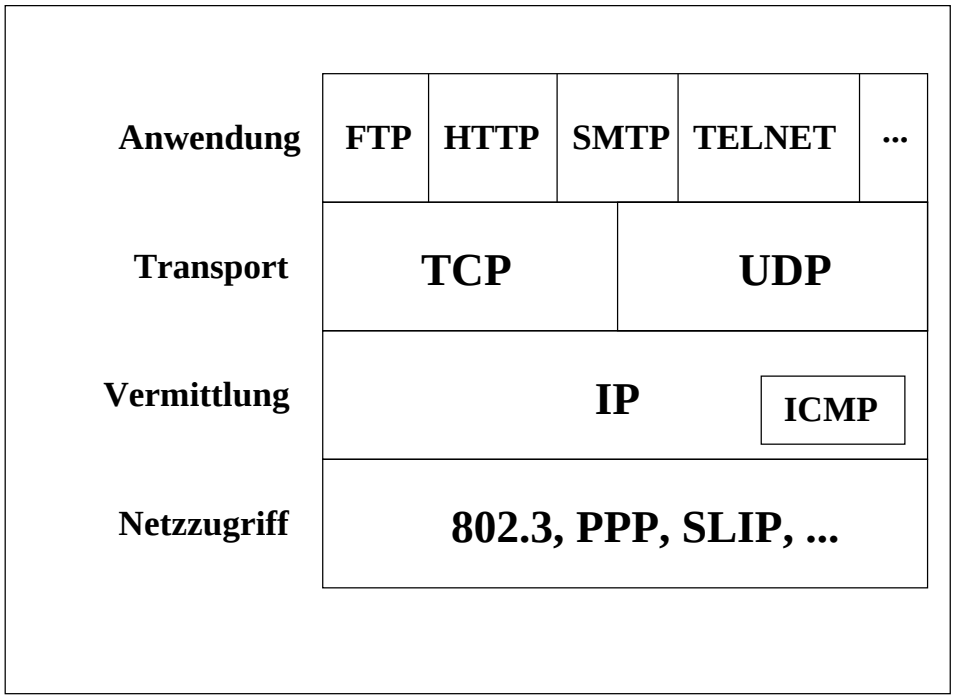

Bild 2.4: Protokolle des TCP/IP-Schichtenmodells

In der Anwendungsschicht arbeiten also sehr viele verschiedene Protokolle, in der Transportschicht gerade noch zwei, in der Vermittlungsschicht nur eines und auch auf der Netzzugriffsschicht eines (allerdings eines aus vielen möglichen).

Jede Schicht dieses Modells muß Informationen besitzen, an welche Abteilung der über ihr liegenden Schicht sie den Datenstrom beim Empfang weitergeben soll. Von unten nach oben betrachtet ergibt sich damit folgender Vorgang:

- Die Netzzugriffsschicht bekommt ein Paket aus dem Netz. Sie hat kein Problem mit der Weitergabe, denn die über ihr liegende Schicht enthält nur ein Protokoll, IP.
- Das Internet–Protokoll (IP) der Vermittlungsschicht muß aufgrund der ihr übermittelten Information schon entscheiden, wem sie das Paket weitergibt. Es stehen hier drei Möglichkeiten zur Auswahl:
  - Das Paket bleibt in der Vermittlingsschicht und wird vom Internet Control Message Protocol (ICMP) weiterverarbeitet (z.B. ping)
  - Das Paket Gehört zu einem TCP–Datenstrom und muß an das Transmission Control Protocol (TCP) der Transportschicht weitergeleitet werden.
  - Das Paket enthält ein UDP-Datagram und muß daher an das User Datagram Protocol (UDP) der Transportschicht weitergegeben werden.

  Diese Information muß IP aus dem IP–Datagram–Header gewinnen. Das Format dieses Headers ist weiter unten erklärt.
- Die Protokolle der Transportschicht (TCP und UDP) haben die größte Aufgabe, denn sie müssen aus einer Vielzahl von möglichen Anwendungen auswählen, für welche dieser Anwendungen das empfangene Paket ist, also an welche Anwendung sie die Daten weiterleiten. Zu diesem Zweck besitzt jede Anwendung der Anwendungsschicht eine Kennummer, die sogenannte Portnummer. Diese Nummer legt genau fest, welche Anwendung das Paket erhält.

Die beiden Schichten, die uns jetzt genauer interessieren, sind die mittleren, also vermittlungsschicht und Transportschicht. Hier finden die Mechanismen statt, die für das Verständnis einer Firewall unerlässlich sind.

2.2.1 Die Vermittlungsschicht

Auf der Vermittlungsschicht (Internet–Schicht) von TCP/IP arbeitet im Wesentlichen nur ein Protokoll, das IP (Internet Protocol). Dieses Protokoll arbeitet nicht mit den physikalischen Adressen von Netzwerkkarten, sondern mit logischen Adressen, eben den allseits bekannten IP–Adressen. Diese Schicht fügt den eigentlich zu übermittelnden Daten noch Verwaltungsinformationen hinzu. Dieses Format wird als Datagram–Format bezeichnet und hat die folgende Struktur:

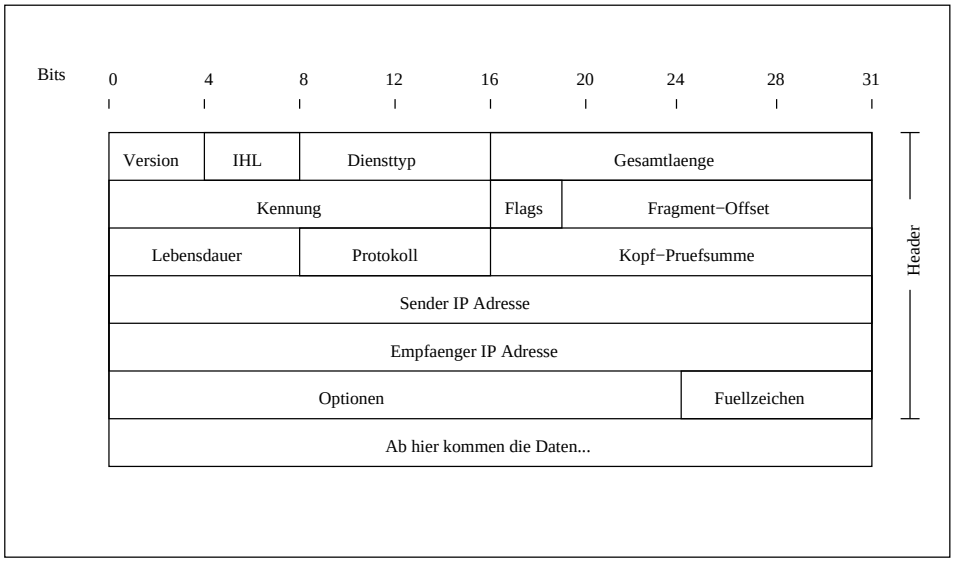

Bild 2.5: Das IP–Datagram Format

Die Bedeutung der einzelnen Felder ist

Version Ein vier Bit breites Feld, daß die Versionsnummer von IP enthält. Im Augenblick wird hier meist eine 4 stehen, in der neuen Version IPV6 folgerichtig dann eine 6.

IHL Internet Header Length – 4 Bit. Gibt die Länge des Headers in 32 Bit Worten an. Damit kann ermittelt werden, ob Optionen gesetzt sind. Normalerweise (ohne Optionen) hat dieses Feld den Wert 5.

Diensttyp TOS (Type of Service) - 8 Bit. Enthält Flags, die zur Steuerung der Zuverlässigkeit des Netzes dienen. Sie werden automatisch erstellt.

Gesamtlänge Enthält die Gesamtlänge des Datagramms (incl. Kopf) in Byte. Aus der Breite dieses Feldes (16 Bit) ergibt sich so eine maximale Größe von Datagrammen von 65535 Byte. Dieser Wert liegt weit über dem maximalen Wert der einzelnen Netzsysteme, daher reichen 16 Bit aus.

Kennung Jedes IP-Datagramm muß eine eigene Kennung haben, um beim Wiederzusammenbau richtig eingeordnet zu werden.

Flags Ein Drei–Bit–Feld das Flags zur Steuerung der Fragmentierung enthält.

Lebensdauer In Netzen wie dem Internet, in denen viele Datagramme auf unterschiedlchsten Wegen versuchen zum Ziel zu kommen, muß eine maximale Lebensdauer haben, sonst werden sie unendlich oft geroutet. Dieses Feld enthält eine Zahl, die vom Sender eingetragen wurde. Jeder Gateway muß diese Zahl um eins herunterzählen, wenn er ein Datagramm routet. Ist der Wert Null, so darf kein Router das Datagramm mehr weiterleiten. Übliche Startwerte sind 32 oder 4 (nur lokale Netze setzen so niedrige Lebensdauer an)

Protokoll Dieses Feld gibt an, von welchem Protokoll der Transportschicht dieses Datagramm kommt. Das ist wichtig für den Empfänger, weil auf der Transportschicht mehrere Protokolle arbeiten bzw. es auch Pakete gibt, die gar nicht bis zur Transportschicht weitergeleitet werden sollen. Nur durch diese Information kann das Empfänger IP das Datagramm an das richtige Transportschichtprotokoll weiterleiten. Übliche Werte sind:

- 17 – UDP
- 6 – TCP
- 1 – ICMP

Kopf–Prüfsumme Prüfsumme über den Inhalt des Headers, nicht der Daten. Die Überprüfung der Daten wird von der Transportschicht erledigt.

Sender IP–Adresse 32 Bit IP-Adresse des Senders

Empfänger IP–Adresse 32 Bit IP-Adresse des Empfängers

Optionen Hier können verschiedene Optionen gesetzt werden, die mit Debugging (Fehlersuche) oder Routenprüfung zu tun haben.

Füllzeichen Füllt den Bereich zwischen den jeweils gesetzten Optionen und dem Ende des 32 Bit Wortes auf.

Das Internet Controll Message Protocol (ICMP)

Das *Internet Control Message Protocol* (ICMP) ist ein integraler Bestandteil von IP und dient zur Übermittlung technischer Meldungen, die übers Netz gehen, aber nicht über die IP-Schicht hinauskommen müssen. D.h., daß dieses Protokoll in der Lage ist, Pakete zu schicken, die nicht an ein Transportprotokoll der Transportschicht gebunden sind, sondern eben nur von Vermittlungsschicht zu Vermittlungsschicht gehen.

Die Aufgaben, die dieses Protokoll zu erfüllen hat sind hier kurz aufgelistet:

Flußkontrolle Wenn ein Rechner Datagramme so schnell schickt, daß der Empfänger sie nicht rechtzeitig verarbeiten kann, so schickt der Empfänger über ICMP eine Meldung an den Sender, daß die Sendungen vorübergehend stoppen sollen. Nachdem der Empfänger alle anstehenden Datagramme verarbeitet hat, schickt er erneut eine Meldung, daß er wieder empfangsbereit ist.

Erkennen von unerreichbaren Zielrechnern Wenn ein Gateway erkennt, daß ein bestimmter Rechner nicht erreichbar ist, so schickt er an den Absender des Paketes eine *Destination unreachable* Meldung über ICMP

Routenoptimierung Wenn ein Gateway erkennt, daß er einen Umweg darstellt, so schickt er an den Absender eine Meldung, in der die schnellere Route steht. Der Absender (bzw. seine IP-Schicht) kann dann das nächste Paket schon über den besseren Weg übermitteln.

Überprüfen von erreichbaren Hosts Mit Hilfe des *ICMP Echo Message* kann ein Rechner überprüfen ob ein Empfänger ansprechbar ist. Das *ping*–Kommando nutzt diese Echo-Meldung.

2.2.2 Die Transportschicht

Auf der Transportschicht von TCP/IP arbeiten zwei verschiedene Protokolle, TCP und UDP. Der wesentliche Unterschied zwischen den beiden Protokollen ist die Frage der Kommunikationssteuerung. TCP (Transmission Control Protocol) arbeitet *verbindungsorientiert*, es baut also einen Datenstrom zwischen Empfänger und Sender auf, der über mehrere Pakete verteilt ist und dessen korrekter Empfang permanent durch Quittungspakete vom

Empfänger bestätigt werden muß. Im Gegensatz dazu arbeitet UDP (User Datagram Protocol) *verbindungslos*, das heißt, es werden einfach Pakete (Datagramme) vom Sender zum Empfänger geschickt, ohne daß dieser den Empfang quittieren muß.

Beide Protokolle arbeiten wieder mit einem sogenannten Header, der dem eigentlichen Datenpaket vorangestellt wird. Beiden Headerformen ist eins gemeinsam, sie enthalten einen Eintrag für die Portnummer des Senderprozesses und die des Empfängerprozesses. Diese Portnummern werden uns im nächsten Abschnitt noch genauer beschäftigen. Im folgenden werden beide Headerformate kurz dargestellt:

Das UDP–Message Format

Die interne Struktur des UDP–Headers ist sehr einfach. Als datagramorientiertes (verbindungsloses) Protokoll benötigt UDP keinerlei Informationen über Sequenznummern oder ähnliches, der benötigte Header ist also sehr klein:

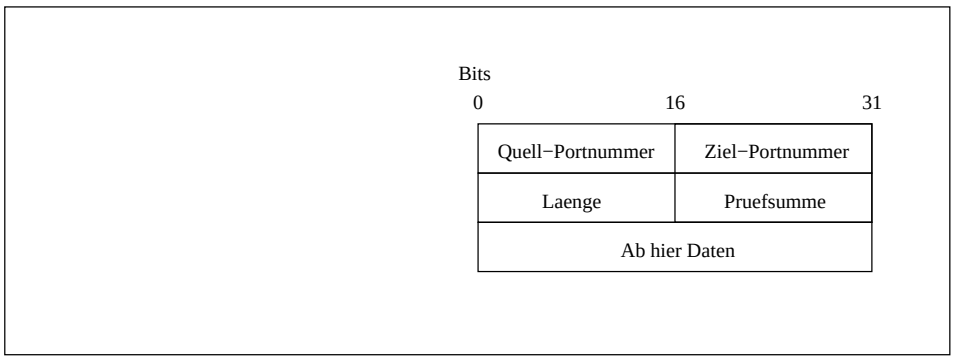

Bild 2.6: Das UDP–Message Format

Die Bedeutung der einzelnen Felder ist

Quellportnummer bezeichnet die Portnummer des Anwenderschichtprogrammes, von dem die UDP-Message abgeschickt wurde.

Zielportnummer bezeichnet die Portnummer des Empfängerprogramms auf der Anwendungsschicht.

Länge Hier steht die Länge der gesamten UDP-Message (incl. Header)

Prüfsumme Eine Prüfsumme über das Datenfeld

UDP wird überall dort verwendet, wo entweder die Datenmenge so klein ist, daß es sich nicht lohnen würde, einen großen Header zu benutzen, weil der größer als die eigentlichen Daten wäre oder wo die Anwendungen selbst noch Überprüfungen des Paketinhaltes vornehmen.

Lohnenswert ist der Einsatz auch dort, wo reine Frage-Antwort Mechanismen auftreten. Dort ist kein verbindungsorientiertes Protokoll nötig, weil ja nach dem Senden einer Frage klar ist, wenn nach einer bestimmten Zeit keine Antwort eingeht, so wird das Paket verlorengegangen sein und die Frage muß nochmal gestellt werden.

Das TCP–Segment Format

TCP ist im Gegensatz zu UDP ein verbindungsorientiertes Protokoll, das Datenströme verarbeitet, die von der Anwendungsschicht kommen. TCP garantiert die Versendung von Daten durch eingebaute Handshake-Mechanismen.

TCP bietet einen verlässlichen Datentransfer durch die Verwendung eines Mechanismus, der Datenpakete (sogenannte Segmente) an einen Empfänger schickt und auf eine Bestätigung des Empfangs seitens des Empfängers wartet. Dabei überprüft der Empfänger auch wieder mittels einer Prüfsumme, ob das Paket fehlerfrei angekommen ist.

Das Format des Headers ist zwangsläufig also wesentlich komplexer als das von UDP:

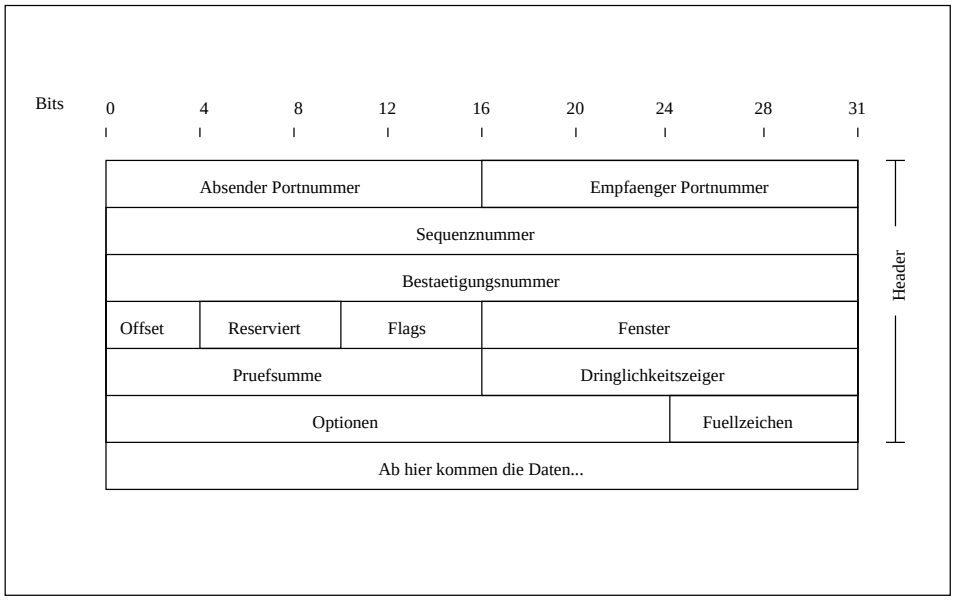

Bild 2.7: Das TCP–Segment Format

Die Bedeutung der einzelnen Felder ist

Absender Portnummer bezeichnet die Portnummer des Anwenderschichtprogramms, von dem der Datenstrom abgeschickt wurde.

Empfänger Portnummer bezeichnet die Portnummer des Empfängerprogramms auf der Anwendungsschicht.

Sequenznummer Gibt die Position im Datenstrom an, an der dieses Segment eingefügt werden soll.

Bestätigungsnummer Dient zur Bestätigung des Empfangs eines Segments. Dieses Feld enthält immer die Nummer, die im nächsten Segment als Sequenznummer stehen soll.

Daten-Offset Gibt die Größe des Headers in 32 Bit Worten an. Ohne Optionen sind es 5. Damit kann genau bestimmt weden, wo der Datenbereich des Segments beginnt.

Flags Verschiedene Flags zur Kommunikationssteuerung (SYN,ACK,...)

Fenster Mit diesem Wert wird die Größe des Buffers angezeigt, der von einem Endknoten für diese Verbindung reserviert wurde. Der sendende Knoten darf keinesfalls mehr Daten als die angegebene Puffergröße senden, ohne auf den Eingang einer Bestätigung zu warten.

Prüfsumme Eine Prüfsumme über das gesamte Segment (Daten und Header)

Dringlichkeitszeiger Wird bei besonders dringend zu verarbeitenden Paketen gesetzt. Wenn gesetzt enthält dieses Feld als Wert die Endadresse des Datenfeldes, das als dringlich gilt

Optionen Verschiedene Optionen zur Kommunikation wie etwa die maximale Segmentgröße

Füllzeichen Zum Auffüllen der Optionszeile auf 32 Bit

2.3 Kommunikationsablauf einer TCP–Verbindung

Die meisten Netzwerkdienste im Internet arbeiten mit TCP als Transportprotokoll. Aus diesem Grund soll hier einmal detailliert der Kommunikationsablauf einer typischen TCP–Verbindung beschrieben werden. Das Verständnis dieses Ablaufs macht uns später das Einrichten einer Firewall erheblich leichter und vermeidet Fehler, die sonst auftreten könnten.

Eine typische Verbindung mit TCP wäre die Kommunikation eines Web–Browsers mit einem Webserver über das *Hypertext Transfer Protocol* (HTTP). Die Portnummer des Webservers ist standardmäßig die 80, der Web-Client (also der Browser) nimmt sich eine beliebige freie Portnummer ab der 1024. In unserem Beispiel wird es die Portnummer 12345 sein.

Der erste Schritt der Kommunikation zwischen Web–Client und Webserver besteht darin, daß der Benutzer des Clients eine Adresse in den Browser – also den Client – eingibt. Diese Adresse wird in eine IP–Adresse umgewandelt und der Client bekommt vom Betriebssystem einen freien Port oberhalb 1024 (also einen sogenannten *unprivilegierten Port*) zugewiesen. In unserem Beispiel ist das der Port 12345. Mun schickt jetzt der Client ein HTTP–Paket an den Server. Die Transportschicht und ihr Protokoll TCP erkennt, daß es sich um eine neue Verbindung handelt und setzt das sogenannte *SYN–Flag* im Flag–Bereich des TCP–Segment–Headers. Webserver benutzen – sofern nicht anders angegeben – standardmäßig den Port 80. Es ergibt sich also folgendes Bild:

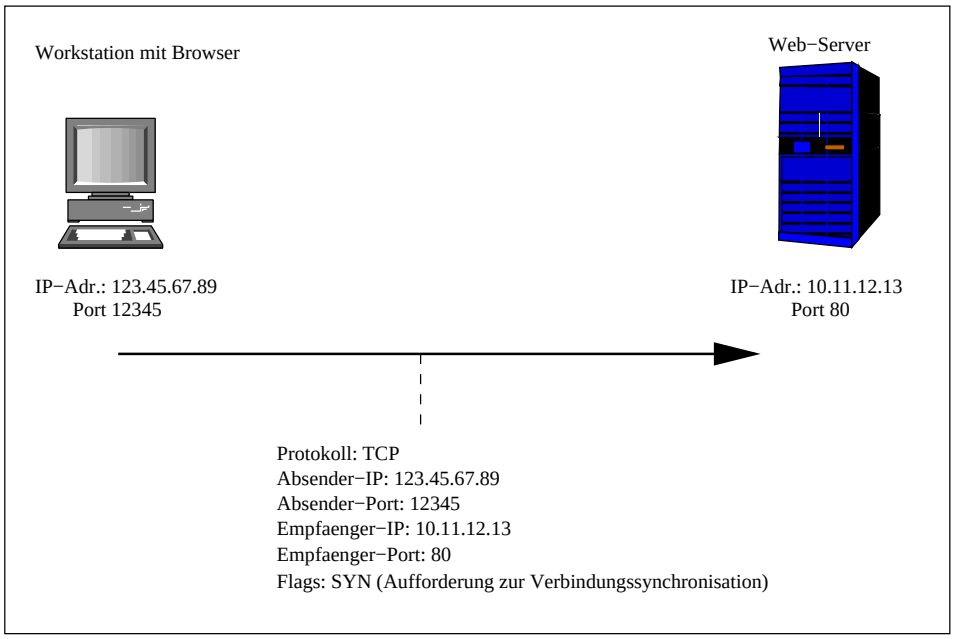

Bild 2.8: Client bittet um Verbindungsaufbau

Der Client schickt in dem Header neben dem SYN–Flag – also der Bitte um Verbindung – auch eine Sequenznummer. Auf der Basis dieser Nummer wird die weitere Verbindung abgewickelt.

Das abgeschickte Paket kommt jetzt auf dem Server an und gelangt bis zur Anwendungsschicht und dort zum Port 80. Dort nimmt ein Serverprogramm (hier der Webserver httpd) den Datenstrom entgegen. Der Server erkennt aufgrund des SYN–Flags, daß es sich um eine neue Verbindung handelt und reserviert einen neuen Socket für die Kommunikation

mit dem Client. Er schickt jetzt ein Paket zurück, das sowohl das ACK–Flag (Acknowledge – Bestätigung), als auch wiederum ein SYN–Flag gesetzt hat. Man spricht jetzt von einer *halb–offenen Verbindung*.

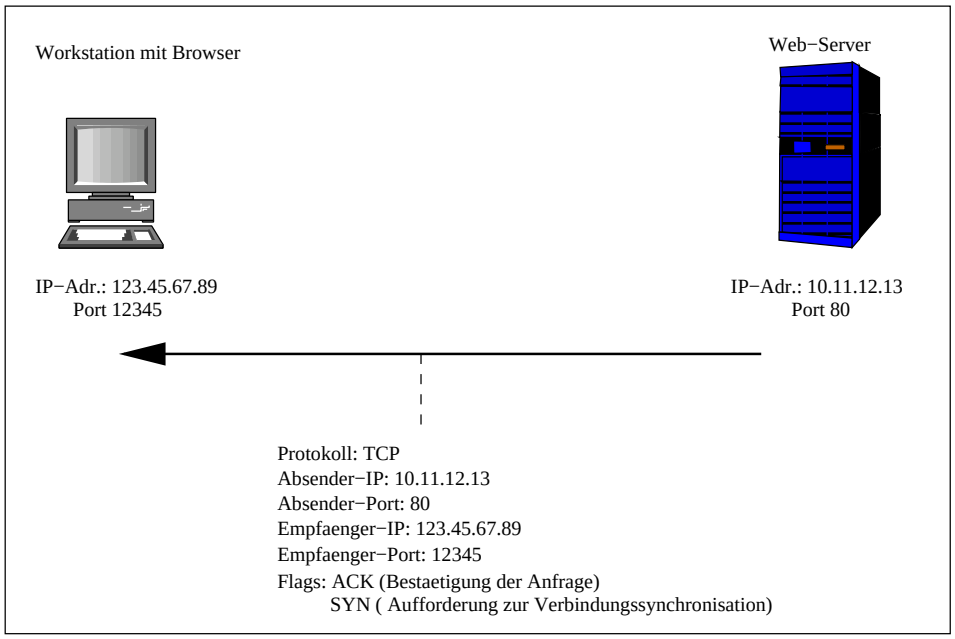

Bild 2.9: Bestätigung der Bitte um Verbindungsaufbau

Die Sequenznummer des Paketes, das der Server in Bild 2.9 dem Client zurückschickt ist die Sequenznummer des ersten Paketes plus eins. Damit bestätigt der Server den Erhalt des letzten Paketes und macht gleichzeitig klar, welche Sequenznummer er im nächsten Paket erwartet. Damit kann der Client das erste Paket jetzt wegwerfen, da er den Erhalt ja jetzt bestätigt bekommen hat.

Diese erste Antwort vom Server hat beide Flags (ACK und SYN) gesetzt. Von nun an wird der gesamte Datenverkehr immer nur noch mit dem ACK–Flag ablaufen. Die Tatsache, daß in allen Paketen des Servers das ACK–Flag gesetzt ist, in der ersten Anfrage des Clients jedoch nicht, wird beim Aufbau einer Firewall ein wichtiges Kriterium sein.

Im nächsten Schritt bestätigt der Client seinerseits den Empfang der Antwort des Servers. Von nun an wird nur noch das ACK–Flag gesetzt sein. Weder Server, noch Client werden im Lauf der Verbindung das SYN–Flag nochmal benutzen.

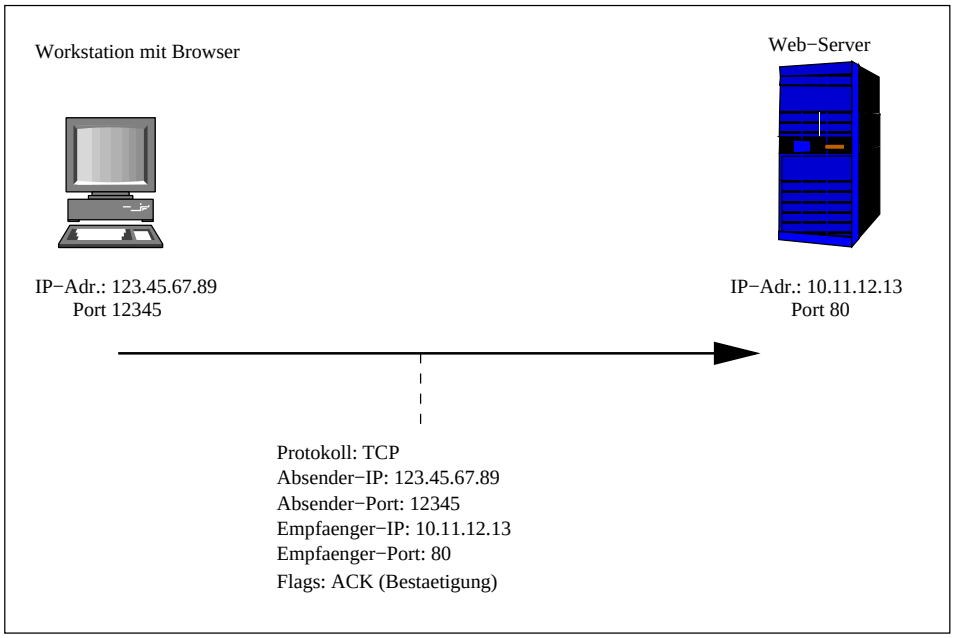

Bild 2.10: Die Verbindung ist hergestellt

Bei jeder weiteren Bestätigung (also jedem weiter erhaltenen ACK–Flag) erhöhen sowohl Client, als auch Server die Sequenznummern des letzten empfangenen Paketes. Damit bestätigen sie also den Empfang des Paketes und geben gleichzeitig bekannt, welches Paket sie als nächstes erwarten. Die jeweilige Gegenseite kann also alle Pakete mit nierigeren Sequenznummern wegwerfen.

Um eine Verbindung abzubauen wird ein weiteres Flag benutzt, das aber für den Aufbau von Firewallsystemen unwichtig ist. Zusammenfassend können wir also festhalten:

Das SYN–Flag ist gesetzt, wenn Client und Server jeweils ihr erstes Paket schicken. Bei allen folgenden Paketen ist das ACK–Flag gesetzt. Ein Paket, in dem nur das SYN–Flag gesetzt ist muß zwangsläufig eine Anfrage eines Clients an einen Server sein.

Nachdem nun die Grundlagen der TCP/IP–Kommunikation geklärt sind, ist es an der Zeit, die Grundlagen der Firewall–Techniken anzusprechen. Die in diesem Kapitel besprochenen Felder der TCP– und IP–Header werden dabei Kriterien für die Auswahl sein.

Kapitel 3

Grundlagen der Firewalltechnik

Die hier vorgestellte Firewall ist eine sogenannte Paketfilter–Firewall. Dabei geht es also darum, die im letzten Kapitel angesprochenen Pakete zu bewerten und anhand verschiedener Kriterien zu entscheiden, ob ein Paket passieren darf, oder nicht.

Eine solche Paketfilter–Firewall arbeitet in der von uns besprochenen Linux–Version mit einer Liste von Annahme– oder Ablehnungskriterien. Daraus entstehen sogenannte Regeln, die genau bestimmen, ob ein Paket passieren darf oder nicht. Diese Kriterien sind

- Netzwerkschnittstelle
- IP–Adressen
- TCP/UDP Portnummern bzw. ICMP Nachrichtentypen
- SYN– und ACK–Flags
- Die Flußrichtung (eingehendes oder abgehendes Paket)

All diese Informationen zeigen, daß die Firewall also auf der Vermittlungs– **und** Transportschicht des TCP/IP–Schichtenmodells arbeitet.

Für beide Flußrichtungen (ankommend – Input und abgehend – Output) stehen jeweils eigene Filter zur Verfügung. Der grundlegende Mechanismus ist dabei der, daß für beide Richtungen eine ganze Anzahl von hintereinanderfolgenden Regeln existieren, die hintereinander abgearbeitet werden. So entsteht eben eine sogenannte Kette (engl. *chain*) von Regeln, daher der Name ip–chains.

Diese Kette von Regeln wird solange abgearbeitet, bis entweder eine Regel zutrifft, oder das Ende der Kette erreicht ist.

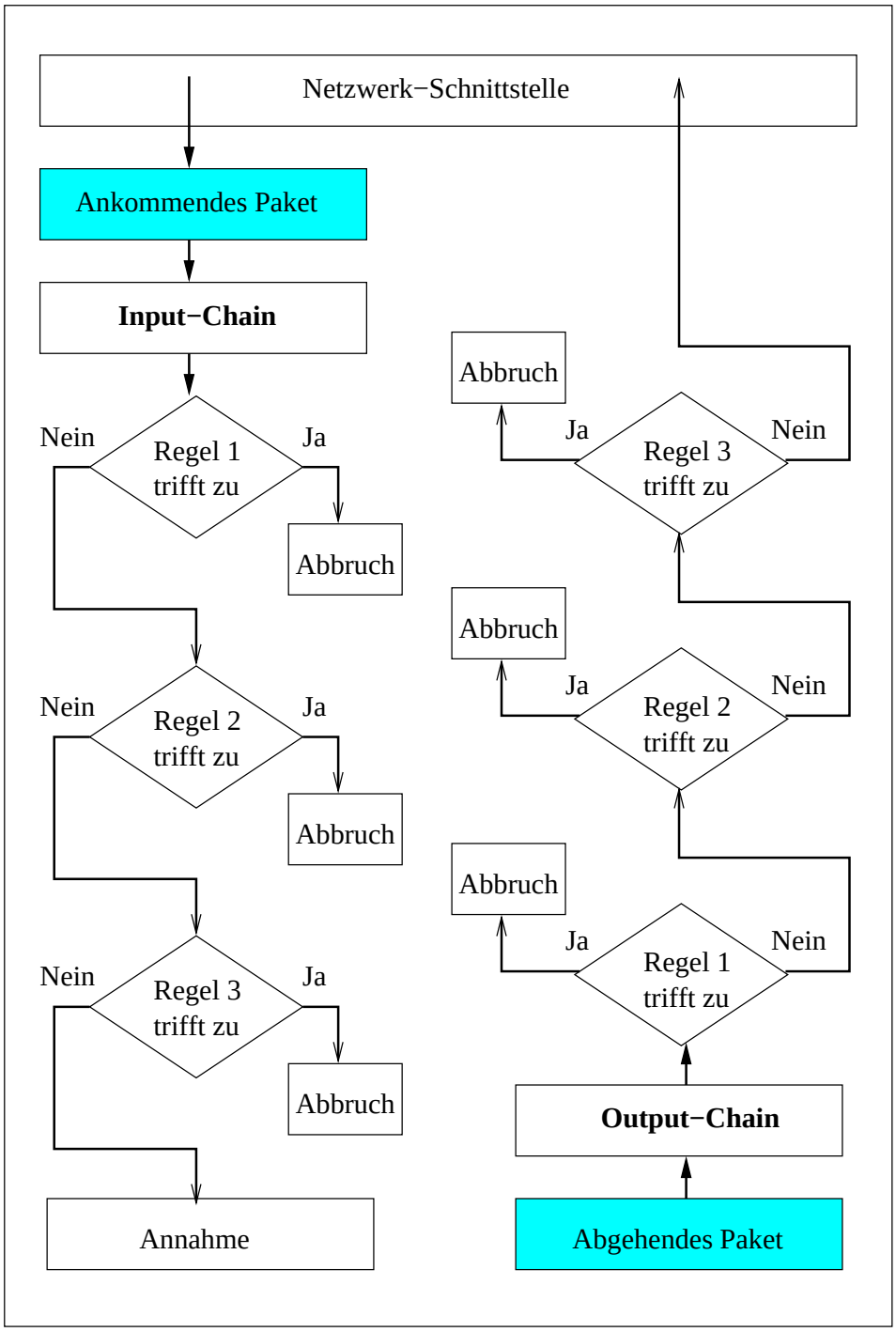

Bild 3.1: Input– und Output–Chains

3.1 Die Voreinstellung für Paketfilter

Aus dem letzten Bild kann der Eindruck entstehen, daß die Regeln grundsätzlich so aufgebaut sind, daß sie einzelne Pakete ablehnen, ansonsten aber alle anderen annehmen. Das ist natürlich möglich, aber weder notwendig, noch empfehlenswert. Linux–Firewalls bieten zwei verschiedene Strategien an, nach denen die Firewall arbeitet – die sogenannten Policies:

- Grundsätzlich wird alles abgewiesen; nur explizit durch Regeln zugelassene Pakete werden akzeptiert. Diese Policy heißt **DENY**.
- Grundsätzlich wird alles akzeptiert; nur explizit durch Regeln verbotene Pakete werden abgewiesen. Der Name dieser Policy ist **ACCEPT**.

Vor– und Nachteile beider Strategien liegen auf der Hand. Die erste Version ist sicherlich die, die mehr Sicherheit und Kontrolle bietet, jedoch auch wesentlich mehr Arbeit bei der Einrichtung erfordert. Wir müssen für alle denkbaren Arten von Netzverkehr Regeln definieren, sonst werden die entsprechenden Pakete ja nicht durchgelassen.

Die zweite Methode (ACCEPT) ist bei der Einrichtung natürlich wesentlich einfacher, sie bietet aber von sich aus wesentlich mehr Angriffsfläche. Um mit dieser Methode eine hinreichend sichere Firewall aufzubauen, müssten wir alle denkbaren Angriffstypen kennen, um sie entsprechend abzuweisen. Und damit wäre es wieder mindestens so viel Arbeit, wie das Einrichten der DENY–Methode.

Zusammengefasst muß also gesagt werden, daß – zumindestens für Pakete, die aus dem unsicheren Netz ins sichere gelangen sollen – die Strategie DENY vorzuziehen ist.

Auf den beiden folgenden Seiten sind die beiden unterschiedlichen Strategien nochmal als Flußdiagramme dargestellt.

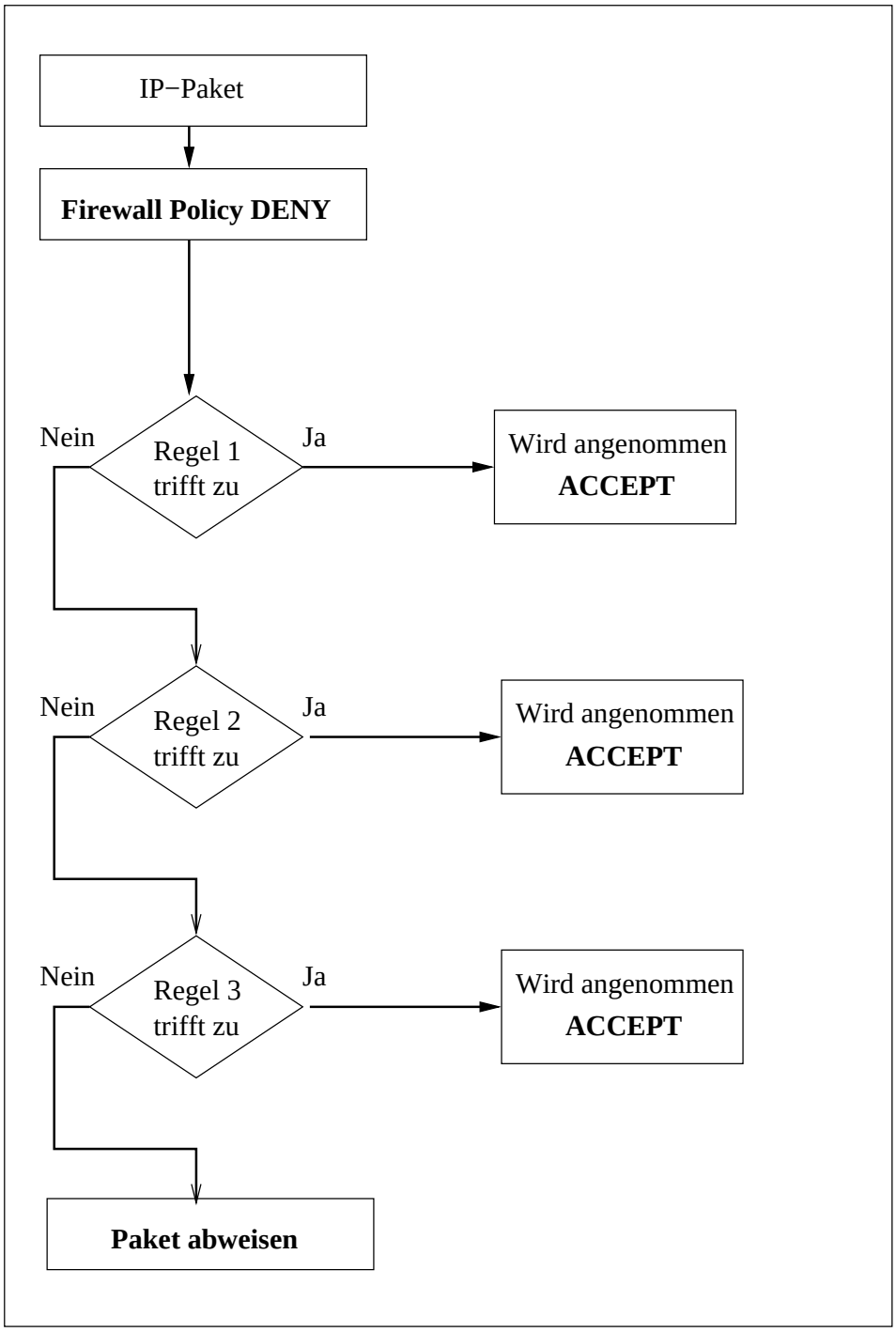

Bild 3.2: Die DENY–Policy

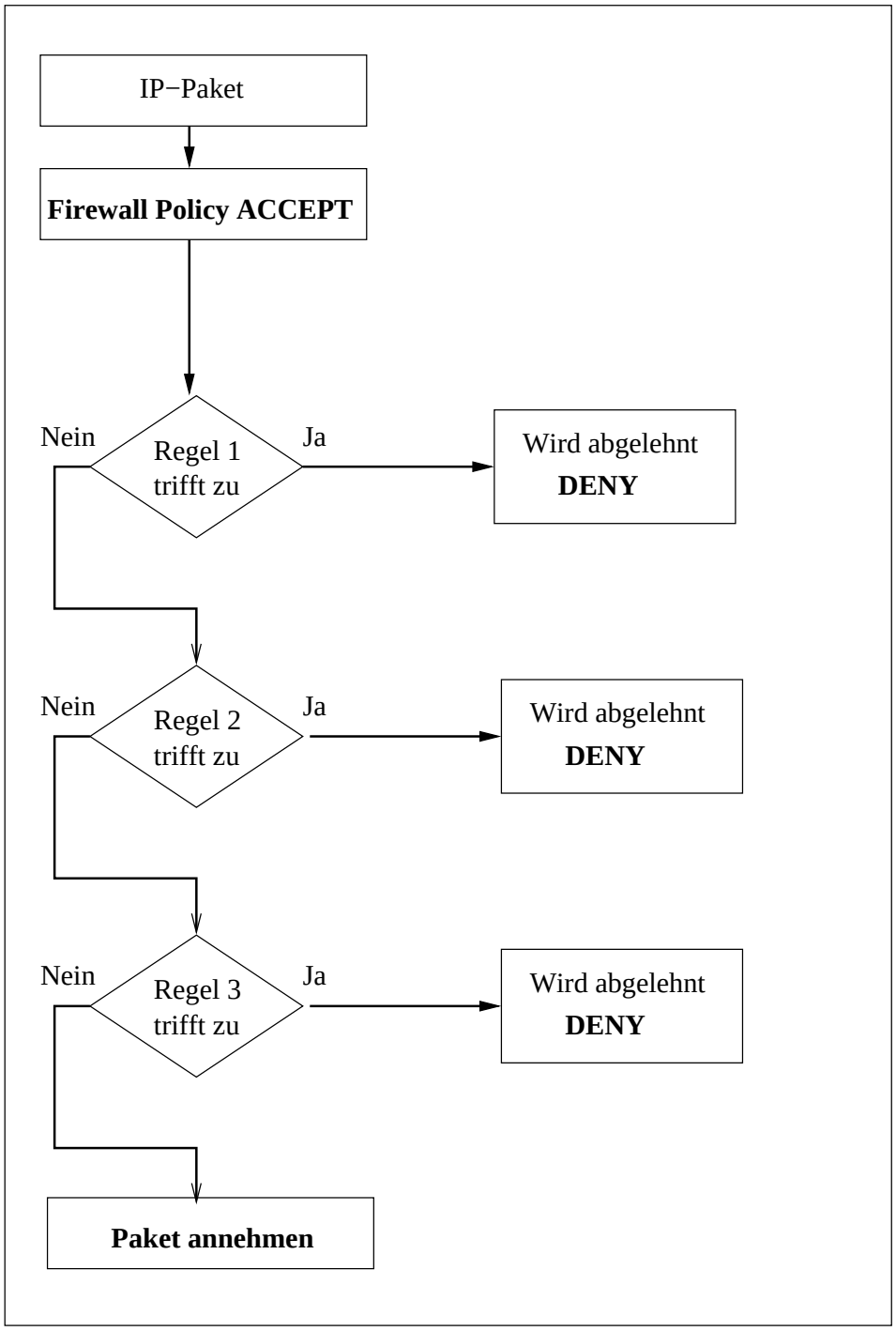

Bild 3.3: Die ACCEPT–Policy

3.2 REJECT oder DENY

Die Linux–Firewall Mechanismen erlauben neben den beiden bereits besprochenen Policies ACCEPT und DENY eine dritte, REJECT. Dabei handelt es sich im Wesentlichen um eine Mutation von DENY, also der Ablehnung eines Paketes. Ein Paket kann entweder einfach verworfen werden, das entspricht hier dem DENY, oder es wird zwar abgelehnt, jedoch erhält der Absender des Paketes eine Fehlermeldung mit Begründung. Diese Methode wird als REJECT (ablehnen) bezeichnet.

Wird ein Paket also durch ein REJECT abgelehnt, so erhält der Absender des Paketes eine ICMP Fehlermeldung, wird es durch DENY abgelehnt, so verwirft die Firewall das Paket einfach und gibt keinerlei Rückmeldung an den Sender des Paketes weiter.

Es ist eigentlich – zumindestens in einem System mit hohen Sicherheitsanforderungen – immer besser, Pakete mit DENY einfach zu verwerfen, als eine Fehlermeldung zurückzugeben. Das hat drei Gründe:

- Durch eine Fehlermeldung wird der Datenverkehr auf dem Netz unnötig vergrößert.
- Fehlermeldungen können als Teil einer Denial–of–Service–Attack Strategie eingesetzt werden. Wenn etwa ein Angreifer, der einen Rechner im Internet blockieren will, seine IP–Adresse auf die des zu blokierenden Rechners fälscht und anschließend tausende von bei Ihnen verbotenen Anfragen macht, dann wird Ihr Rechner tausende von Fehlermeldungen nicht an den Rechner des Angreifers, sondern an den zu blockierenden Rechner schicken.
- Fehlermeldungen – und seien sie an sich noch so wenig aussagekräftig – können einem potenziellen Angreifer doch zumindestens Informationen geben, die er besser gar nicht hat. Etwa alleine die Existenz einer Firewall oder ähnliches.

3.3 Zusammenfassung

Zusammenfassend können wir also feststellen, daß es zwei wesentliche Strategien (Policies) gibt, die eine Paketfilter–Firewall ausmachen. Entweder wir nehmen alle Pakete, außer explizit verbotenen, an – Policy ACCEPT, oder wir verweigern die Annahme aller Pakete außer explizit erlaubten – Policy DENY.

Zum Umgang mit einzelnen Paketen stehen uns drei Methoden zur Verfügung:

ACCEPT Das Paket wird angenommen

DENY Das Paket wird stillschweigend verworfen

REJECT Das Paket wird abgelehnt, aber es wird eine ICMP–Fehlermeldung an den Absender zurückgeschickt.

Grundsätzlich unterscheiden wir zwischen der Input– und Output–Chain, also der Kette von Regeln für eingehende bzw. ausgehende Pakete.

Kapitel 4

Filtern von eingehenden Paketen

Eine Firewall ist grundsätzlich in der Lage, sowohl ankommende, als auch abgehende Pakete zu filtern. In der Regel ist jedoch die Filterung der ankommenden Pakete die wichtigere Technik, wollen wir doch vermeiden, daß Angriffe von außen, also aus dem unsicheren Netz, auf unser zu schützendes Netz stattfinden. In diesem Kapitel werden die Kriterien besprochen, nach denen wir eingehende Pakete filtern können.

4.1 Filtern nach IP–Adressen

Auf der Ebene der Vermittlungs– und Transportschicht ist die IP–Adresse die einzige Möglichkeit, herauszufinden, wer ein Paket an uns geschickt hat. Diese Information steckt im IP–Header, also der Information, die uns die Vermittlungsschicht übermittelt.

Die Authentizität dieser Information ist grundsätzlich fragwürdig. Es ist ein leichtes, die IP–Adresse des Senders eines Paketes zu fälschen, auch wenn dadurch eine funktionierende Kommunikation in der Regel unmöglich gemacht wird.

Grundsätzlich können wir sowohl Absender–, als auch Empfänger–IP–Adressen als Kriterien für eine Firewall benutzen.

4.1.1 Filtern nach Absender–Adressen

Das Kriterium für die Entscheidung, ob wir ein Paket durchlassen oder nicht ist hier also die Adresse des Senders, die uns ja im IP–Header zur Verfügung steht. Hier kommen verschiedene Techniken zur Anwendung.

Ablehnen von gefälschten oder ungültigen Adressen

Es ist natürlich nicht so einfach möglich, gefälschte Adressen als solche zu erkennen, aber es stehen uns jedoch ein paar Mittel zur Verfügung, offensichtlich falsche Adressen abzuweisen. Sechs Hauptgruppen von solchen Adressen sollten wir grundsätzlich ablehnen:

1. Die eigene IP–Adresse: Niemals kann ein regulär eingehendes Paket vom eigenen Rechner stammen. Es muß sich also zwangsläufig um eine Fälschung handeln.

2. Die IP–Adressen, die für lokale Netze ohne Internet–Anschluß reserviert sind: Für jede Klasse (A, B und C) existieren Adressen, die im Internet nicht geroutet werden können. Aus dem Internet können (bzw. dürfen) solche Pakete niemals auf unseren Rechner kommen. Die sogenannten *privaten Adressen* sind:

   Klasse A 10.0.0.0 bis 10.255.255.255

   Klasse B 172.16.0.0 bis 172.31.255.255

   Klasse C 192.168.0.0 bis 192.168.255.255

Pakete, die diese Adressen haben und aus dem Internet kommen, können Sie bedenkenlos ablehnen.

3. Multicast–Adressen der Klasse D: Hierbei handelt es sich um reservierte Adressen für sogenannte Multicast–Broadcasts, wie sie etwa für Audio– oder Video Übertragung an mehrere Empfänger verwendet werden. Diese Adressen dürfen zwar als Empfänger–Adressen, niemals jedoch als Absender vorkommen. Es handelt sich um die Adressen im Bereich von 224.0.0.0 bis 239.255.255.255.

4. Alle Klasse E–Adressen: Adressen dieser Klasse waren für expeimentelle Erweiterungen vorgesehen und werden grundsätzlich nicht öffentlich vergeben. Sie liegen im Bereich von 240.0.0.0 bis 247.255.255.255. In der Regel werden diese Adressen ohnehin nur von US–Militärs und –Geheimdiensten verwendet. Im Internet sollten sie nie auftauchen.

5. Loopback–Adressen: Die reservierte Adresse 127.0.0.1 ist eine Adresse, die immer den lokalen Rechner meint. Ein Paket, das aus dem Internet kommt und diese Adrese als Absenderadresse eingtragen hat kann niemals eine echte Adresse sein.

6. Broadcast Adressen: Broadcast–Adressen sind Adressen, die in Netzen benutzt werden, um mehrere Systeme gleichzeitig anzusprechen. Solche Adressen sind als Empfänger–Adresse legal, als Absender–Adresse jedoch nie.

Bestimmte Adressen herausfiltern

Es existieren bestimmte Adressen, von denen man weiß, daß von ihnen oft Angriffe ausgehen. Es ist möglich, solche Adressen, seien es Host– oder ganze Netz–Adressen einfach zu sperren. Auf der anderen Seite ist es auch möglich, bestimmten Adressen den Zugriff auf bestimmte Dienste eines Netzes zuzulassen. So kann etwa die Filliale unserer Firma, von der wir genau die IP–Adresse wissen, auf unseren Datenbank–Server zugreifen, während sonst nieman das können soll.

4.1.2 Filtern nach Empfänger–Adressen

Das Feld der Empfänger–Adresse ist für die Firewall dann interessant, wenn sie selbst auch Router in ein Netz ist. In diesem Fall kann bestimmt werden, welche Rechner im Netz ein Paket bekommen dürfen und welche nicht. So kann etwa ein bestimmter Rechner (mit bestimmter IP-Adresse) im lokalen Netz ein Webserver sein, der natürlich für andere Netzteilnehmer erreichbar sein soll. Ein Paket, dessen Empfänger–IP–Adressenfeld die Adresse dieses Webservers enthält, muß also durchgelassen werden, alle anderen Pakete könnten wir sperren.

4.2 Filtern nach Portnummern

Portnummern sind Informationen aus der Transportschicht, also entweder aus dem UDP–Header oder dem TCP–Header. Sie geben an, welches Programm auf der Anwendungsschicht ein eingehendes Paket erhält (Empfänger–Portnummer) oder von welcher Anwendung ein Paket geschickt wurde (Absender–Portnummer). Beide Fälle sind wichtige Informationen für eine Firewall.

4.2.1 Filtern nach der Absender–Portnummer

Hier müssen wir unterscheiden, ob es sich um ein Paket handelt, das eine Anfrage nach einem lokalen Service beantragt (also ein Client von außerhalb, der einen Service von innerhalb nutzen will). In diesem Fall wird die Portnummer eine Nummer zwischen 1024 und 65535 sein müssen.

Im umgekehrten Fall, also wenn ein Client von innerhalb einen Service von außerhalb nutzen will, dann wird ein eingehendes Paket die Absender–Portnummer des jeweils genutzten Dienstes haben[1].

4.2.2 Filtern nach der Empfänger–Portnummer

Dieses Feld ist für uns sehr interessant, weil hier – zumindestens bei eingehenden Paketen – die Portnummer des Dienstes steht, den ein User von außerhalb nutzen will. Das heißt, hier ist der Ort, wo wir von vorneherein nur die Pakete durchlassen, die Portnummern von Diensten enthalten, die wir auch anbieten wollen.

Auch hier gibt es jedoch den zweiten Fall, nämlich die Antwort eines fremden Servers auf eine Nachfrage eines lokalen Clients. In dem Fall liegen die Empfänger–Portnummern im unprivilegierten Bereich zwischen 1024 und 65535.

4.3 Filtern nach TCP–Status–Flags

Auch diese Information stammt aus der Transportschicht, steht jedoch – im Gegensatz zu Portnummern – nur bei TCP, nicht bei UDP zur Verfügung. Hier geht es im Speziellen um die Frage des Handshakes beim Verbindungsaufbau. Der zugrundeliegende Mechanismus wurde ab Seite 15 genau dargestellt.

Diese Information ist für eine Firewall von daher sehr wichtig, daß mit ihrer Hilfe unterschieden werden kann, ob ein Paket eine Anfrage von außen oder eine Antwort auf eine Anfrage von innen enthält. Pakete, die Antworten fremder Server enthalten haben grundsätzlich das ACK–Flag gesetzt. Firewall–Regeln können daher z.B. entscheiden, daß in ein internes zu schützendes Netz nur Pakete mit diesem Flag hereingelassen werden.

4.4 Angriffsformen von außerhalb

Um eine Filterung eingehender Pakete überhaupt sinnvoll zu nutzen, ist es notwendig, die gängigen Formen von Angriffen von außerhalb zu kennen. Im Folgenden sollen kurz die häufigsten Angriffsformen beschrieben werden, die keine normalen Zugriffe auf Dienste sind.

4.4.1 Portscans

Ein Portscan ist ein Abtastversuch – ein Versuch, zu einem bestimmten Port eine Verbindung aufzubauen oder zumindestens eine Reaktion zu erhalten, aus der Rückschlüsse möglich sind, ob ein bestimmter Dienst angeboten wird oder nicht. Der Begriff *Scan* bezeichnet dabei eine Serie von Abtastversuchen auf verschiedene Ports. Dazu werden sogenannte Portscanner benutzt, die einfach verschiedene – oder gar alle – Ports eines Rechners ansprechen und auf Antwort warten.

Obwohl der eigentliche Versuch eines Portscans selbst noch kein Einbruchsversuch ist, ist er doch meistens der erste Schritt eines Angriffs. Ein potentieller Angreifer kann mittels Portscan herausfinden, auf welchem Port eines bestimmten Rechners Anwendungen laufen, die Antworten verschicken. Mit dieser Grundinformation kann dann eine Angriffsstrategie festgelegt werden.

Man unterscheidet zwischen *kompletten Portscans*, die alle Ports zwischen 0 und 65534 abfragen und *gezielten Portscans*, die nur bestimmte – als potentiell gefährlich einzuschätzende – Ports überprüfen. Komplette Scans sind heute eher selten, weil sie viel Zeit benötigen und daher auch leicht bemerkt werden. Aktuelle Utilities sind heute in der Lage, bestimmte Ports abzufragen, die sich für Angriffe von außen eignen. Das kostet weniger Zeit und ist daher auch weniger leicht zu erkennen.

[1] Siehe /etc/services

Typische Ports, die gerne gescannt werden sind in der Tabelle im Anhang (Seite 69) aufgelistet.

Mit einer installierten Firewall werden solche Portscanns natürlich bemerkt. Allerdings – und dazu haben wir ja eine Firewall – bedeutet der Scan an sich noch keine reale Gefahr. Jemand versucht etwas über unser System herauszufinden, wird jedoch von der Firewall abgewiesen…

4.4.2 Denial–of–Service–Attacks

Eine wesentlich gefährlichere Art des Angriffs ist die sogenannte *Denial of Service Attack*, oft als DoS abgekürzt. Hierbei wird versucht, einen Computer derart mit Datenpaketen zu überfluten, daß er entweder keine anderen Pakete mehr verarbeiten kann, oder im schlimmsten Fall komplett abstürzt. Eine Abart dieser Angriffsform ist die *Distributed Denial of Service Attack* (DDoS), bei der nicht nur ein, sondern viele Computer gleichzeitig unseren Rechner mit Paketen überschwemmen.

Es gibt keinen vollkommenen Schutz gegen solche Angriffsformen, daher ist es wichtig, zumindestens die wichtigsten Formen dieses Angriffs zu kennen. Im Folgenden sollen ein paar dargestellt werden:

TCP–SYN–Flooding

Diese Angriffsform besteht einfach darin, den dreiteiligen Handshake einer TCP–Verbindung (siehe Seite 15) zu unterbrechen. Wenn ein Client einem Server eine Anfrage stellt, so schickt er dem Server ein Paket mit gesetztem SYN–Flag. Der Server schickt als Antwort ein Paket mit einem gesetzten SYN– und ACK–Flag. Damit ist die Verbindung *halboffen*. Wenn der Client jetzt **keine** Antwort mit gesetztem ACK–Flag schickt, dann hält der Server diese Verbindung in der Regel für etwa 5 Minuten offen.

Wenn der Client jetzt erneut (mit anderer Sequenznummer) eine SYN–Nachfrage schickt, dann muß der Server einen neuen Kanal öffnen. So geht das Spielchen weiter, bis der Server keine Kanäle mehr zur Verfügung hat. Im schlimmsten Fall wird der Server dann gänzlich abstürzen, im mindesten Fall kann er keine weiteren Dienste an andere anbieten.

Die SYN–Flag Anfragepakete werden normalerweise eine gefälschte Sender–IP–Adresse haben, der Sender erwartet ja keine Antwort, er will ja nur stören.

Linux bietet eine speziell gegen diese Angriffsform gerichtete Kerneleigenschaft, die sogenannten SYN–Cookies. Damit kann sich ein Linux–Kernel erfolgreich gegen Angriffe dieser Art wehren.

PING-Flooding

Das Programm `ping` wird benutzt, um festzustellen, ob ein betimmter Rechner über das Netz erreichbar ist. Dieses Programm schickt eine ICMP–Echoanforderung an einen bestimmten Rechner und wartet dann auf eben dieses Echo. Die Tatsache, daß es sich hier um ICMP (siehe Seite 12) handelt, zeigt schon, daß ping hier nicht über die Netzwerkschicht (Vermittlungsschicht) hinausgeht. Es existieren also keinerlei Ports.

Wenn ein Rechner einen anderen Rechner mit ping–Anfragen überhäuft, so kann es dazu kommen, daß dieser andere Rechner keine Bandbreite mehr für andere Netzwerkdienste übrig hat, er ist also auch faktisch lahmgelegt.

Ältere Unix–Versionen (nicht Linux) waren auch anfällig gegen das sogenannte Todesping, eine Echoanforderung, die zu große Datenpakete verschickt. Diese alten Rechner konnten dann tatsächlich zum Absturz gebracht werden.

UDP-Flooding

Das UDP–Protokoll eignet sich hervorragend für Denial–of–Service–Angriffe. Es ist verbindungslos, d.h., es existiert keine Flußkontrolle, kein Handshake, keine Sequenznummern... Es ist also sehr einfach, einen Rechner mit UDP–Anfragen derart zu überhäufen, daß keine Bandbreite mehr für andere Netzwerkdienste übrig bleibt.

UDP ist traditionell aber sowieso für Dienste innerhalb des lokalen Netzes bestimmt, Dienste im Internet benutzen fast immer TCP als Transportprotokoll. Viele Firewalls lassen daher grundsätzlich keine oder nur ein paar ausgewählte UDP–Dienstanfragen aus dem unsicheren Netz zu.

ICMP–Redirect–Angriff

Der Nachrichten–Typ 5 des Internet Control Message Protocol ist dazu gedacht, einen Computer anzuweisen, seine Routing Tables zugunsten anderer Routen umzustellen. Diese Funktionalität ist aber nur dann gewährleistet, wenn auf dem Rechner entweder der `gated` oder der `routed` läuft.

Im Fall eines Angriffs, kann der Angreifer Ihren Rechner miz dieser Methode anweisen, alle ausgehenden Pakete über ihn zu routen. Das käme einer Katastrophe gleich, kann er doch so jedes ein- und ausgehende Paket manipulieren.

Kapitel 5

Filtern von ausgehenden Paketen

Das Filtern ausgehender Pakete ist – sofern den Usern des lokalen Netzes Vertrauen entgegen gebracht wird – deutlich unkritischer, als das Filtern eingehender Pakete. Auf der anderen Seite ist es natürlich in mancherlei Hinsicht trotzdem ratsam, auch diese Pakete einer Prüfung zu unterziehen, bevor man sie in die weite Welt entlässt. Auch hierzu folgt eine kurze Darstellung der jeweiligen Kriterien.

5.1 Filtern nach IP–Adressen

Auch hier unterscheiden wir wieder zwischen Absender– und Empfängeradresse. Beide Fälle bieten ein paar nennenswerte Kriterien für eine Paketfilterfirewall:

5.1.1 Filtern nach Absender–Adresse

Das Kriterium für die Senderadresse für abgehende Pakete ist simpel. Es muß sich um eine im lokalen Netz gültige Adresse handeln. Also entweder um einen bestimmten Adresspool, oder um eine klare Liste von Adressen, die wiederum entweder fest vergeben sind, oder von einem DHCP–Server verwaltet werden.

Bei einer Vergabe von Adressen über DHCP kommt es beim Aufbau der Verbindung zu einer speziellen Situation, die aber nur dann eine Rolle spielt, wenn sich der DHCP–Server außerhalb des zu schützenden Netzes befindet. Und das ist grundsätzlich keine gute Idee.

5.1.2 Filtern nach Empfänger–Adresse

Bei der Frage der Filterung nach der Empfänger–Adresse abgehender Pakete, gibt es zwei wesentliche Bereiche. Wir unterscheiden hier grundsätzlich, ob das Paket an einen Server im Internet gerichtet ist, oder ob es an einen Client im Internet geht, der einen Server im LAN angesprochen hatte.

1. Pakete an einen Server im Internet:
   Hier ist es möglich, bestimmte IP–Adressen oder DNS–Namen komplett zu sperren, weil es unerwünscht ist, daß aus dem LAN heraus diese Dienste angenommen werden. Es kann auch z.B. ein bestimmtes Protokoll (z.B. POP3 oder IMAP) an einen bestimmten Rechner gebunden werden, alle anderen Adressen werden nicht durchgelassen...

2. Pakete an einen Client im Internet:
   Unser Netz hat vielleicht ein paar öffentliche Server, die von jedem Client aus benutzt werden dürfen (Webserver, FTP–Server...), es hat aber andererseits auch Dienste im Angebot, die nur für bestimmte Rechner im Netz zur Verfügung stehen sollen. So etwa die Filiale in einer anderen Stadt, deren IP–Adresse wir kennen. In so einem

Fall ist es durchaus sinnvoll, nicht nur die eingehenden Pakete, sondern eben auch die ausgehenden Pakete von der Firewall unterdrücken zu lassen.

5.2 Filtern nach Portnummern

Auch hier unterscheiden wir wieder zwischen Absender– und Empfängerport. Für ersteren ist zu bemerken, daß ein Absenderport eines Clients aus dm lokalen Netz grundsätzlich eine Portnummer aus dem unprivilegierten Bereich haben sollte, Serverprozesse des lokalen Netzes können wiederum nur die zugelassenen Ports benutzen. Hier haben wir wiederum eine doppelte Möglichkeit, zu verhindern, daß ungewollte Dienste ins globale Netz gelangen.

Die Empfängerports haben eine ähnliche Beschränkung. Wenn ein Serverprozess aus dem lokalen Netz ein Paket an einen Client im Internet schickt, so darf der Empfängerport nur ein unprivilegierter sein. Umgekehrt vermeiden wir das Senden von Paketen, mit privilegierten Portnummern von unseren Clients ins Internet.

5.3 Filtern nach TCP–Status–Flags

Wie bei den ankommenden Paketen, können wir auch bei den abgehenden Paketen die Statusflags nutzen, um zu entscheiden, ob ein Paket von einem Client oder von einem Server stammt.

- Wenn ein lokaler Client einen Dienst im Internet anspricht, dann ist im ersten Paket nur sein SYN–Flag gesetzt, nicht jedoch das ACK–Flag. Alle weiteren Pakete haben nur noch das ACK–Flag gesetzt, nicht jedoch das SYN–Flag.
- Wenn ein lokaler Server einen Dienst im Internet anbietet, dann sind im ersten Paket, das er verschickt beide Flags (SYN und ACK) gesetzt, alle weiteren Pakete haben nur das ACK–Flag gesetzt.

Kapitel 6

Gestaltung einer Firewall mit Linux

Nachdem nun alle wichtigen theoretischen Voraussetzungen erfüllt sind, ist es an der Zeit, die konkrete Umsetzung einer Firewall unter Linux anzusprechen. Linux hat, im Gegensatz zu den meisten anderen Betriebssystemen, die Funktionalität der Firewall bereits im Betriebssystemkern (Kernel) fest eingebaut. Das macht die Firewall sowohl stabiler, als auch wesentlich schneller, als wenn sie im sogenannten *Userspace* realisiert werden müsste.

6.1 Die drei Standard–Ketten

Wir haben bereits gesehen, daß eine Firewall unter Linux aus vielen einzelnen Regeln besteht, die zu einer Kette (*chain*) zusammengefasst werden. Standardmäßig existieren drei Ketten, es können aber beliebig viele weitere Ketten erstellt werden – z.B. spezielle Regeln für einen Modemanschluß oder ähnliches. Die drei Standard–Ketten sind *input, output* und *forward*.

Wenn ein Paket von außen an einer beliebigen Netzwerk–Schnittstelle ankommt, dann wird sein Header mit jeder Regel der *input*–Chain verglichen, solange bis entweder eine Regel zutrifft, oder das Ende der Regelkette erreicht ist. Jede Regel, die zutreffen sollte, kann als Folge ein `ACCEPT, REJECT` oder `DENY` definiert haben. Erst, wenn das Ende der Kette erreicht ist, wird die voreingestellte Policy der Chain aktiv, die wiederum einen der drei Werte `ACCEPT, REJECT` oder `DENY` darstellt. Die erste gefundene passende Regel kommt zum Zug.

Genauso verhält es sich mit den Paketen, die zu einer beliebigen Netzwerkschnittstelle geschickt werden, um den Computer zu verlassen. In diesem Fall tritt – statt der *input*–Chain – die *output*–Chain in Kraft.

Wenn ein Paket nicht für den Rechner bestimmt ist, auf dem die Firewall läuft, sondern an andere Rechner weitergeleitet (geroutet) werden soll, dann tritt die dritte Regelkette zu, die *forward*–Chain. Diese Regelkette kann neben den normalen Regeln noch eine weitere Fähigkeit anbieten, das sogenannte *Masquerading*. Hierbei handelt es sich um die Fähigkeit des Kernels, zu routende Pakete zu maskieren, so daß von außen der Eindruck erweckt wird, sie kämen ausschließlich von dem Firewall–Rechner. Das hat zwei Vorteile:

- Ein internes Netzwerk kann mit reservierten IP–Adressen arbeiten, trotzdem kann jeder Rechner des internen Netzes über die Firewall nach außen ins Internet gelangen.
- Von außen ist die Struktur des internen Netzes nicht sichtbar, weil es nach außen hin nur die IP–Adresse der Firewall gibt. Ein Angriff auf einen Rechner innerhalb des Netzes ist von daher schon unmöglich, daß jeder Rechner innen eine Adresse hat, die im Internet gar nicht geroutet werden darf.

Zusammengefasst könnte man eine Firewall mit all ihren Regelketten also folgendermaßen darstellen:

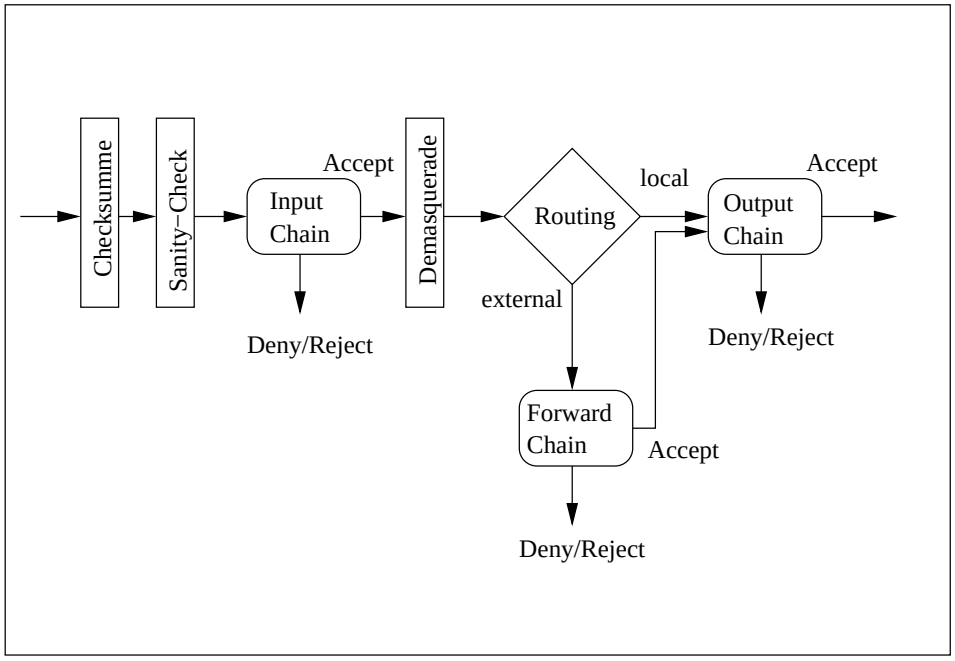

Bild 6.1: Architektur der IP-Chains im Kernel

Die Regeln der einzelnen Ketten werden in Tabellen im Kernel gespeichert, für jede einzelne Kette eine eigene Tabelle. Diese Regeln stehen in der Reihenfolge im Kernel, in der sie eingegeben wurden. In manchen Fällen ist es notwendig, die Reihenfolge zu beachten, weil eine falsche Reihenfolge den Sinn der Regeln umwerfen könnte. Zur Erinnerung, die erste passende Regel zählt.

Es ist auch möglich, Regeln am Anfang einer Kette neu einzufügen, in der Praxis ist jedoch der Aufbau von Firewall–Regeln nicht, was man bei jedem Start von Hand übernimmt. Wir werden im Gegenteil natürlich Scripte erstellen, die die Regeln definieren und die bei jedem Start ausgeführt werden. Jede notwendige Änderung von Regeln sollte auch immer in diesen Scripten stattfinden, denn
Von Hand eingegebene Firewall–Regeln sind nach dem nächsten Neustart verschwunden!

6.2 Das Handwerkszeug: ipchains

Das Programm, mit dem wir Firewall–Regeln erstellen, bearbeiten oder löschen heißt `ipchains` und ist sozusagen ein Werkzeug zur Manipulation der Regeltabellen im Kernel. Dieses Programm sollte im Verzeichnis `/sbin` liegen und nur vom Systemverwalter aufgerufen werden. Als typisches Unix–Programm wird es ausschließlich über die Kommandozeile gesteuert, also über einen Berg von verschiedenen Kommandozeilenparametern. Das ist von daher wichtig, daß wir ja Scripte schreiben wollen, die die einzelnen Regeln erstellen. Nur mit einem Kommandozeilenorientierten Programm kann das gelingen.

`ipchains` ist ein Programm, das für jede Firewall–Regel einmal aufgerufen werden muß. Die wichtigen Parameter werden jetzt kurz dargestellt. Eine vollständige Beschreibung der Parameter finden Sie auf der entsprechenden Man–Page `ipchains(8)` und im `IPCHAINS-HOWTO`.

```
ipchains
  -A | -I [Chain]
  [-i Interface]
  [-p Protokoll]
  [[!] -y]
  [-s Adresse [Port[:Port]]]
  [-d Adresse [Port[:Port]]]
  -j Policy
  [-l]
```

`-A` [*Chain*]	Hängt eine Regel ans Ende der angegebenen Kette (Chain) an. Wenn keine Chain angegeben wird, so gilt die Regel für alle Chains.
`-I` [*Chain*]	Fügt eine Regel vor den Anfang der angegebenen Kette (Chain) ein. Wenn keine Chain angegeben wird, so gilt die Regel für alle Chains.
`-i` *Interface*	Netzwerkinterface, für das die Regel gelten soll. Wird kein Interface angegeben, so gilt die Regel für alle Interfaces.
`-p` *Protokoll*	Protokollnamen oder –nummer, für das die Regel gilt. Gültige Werte sind hier `tcp, udp, icmp` und `all`. Daneben sind alle Protokollnamen und –nummern aus `/etc/protocols` erlaubt.
`-y`	Das SYN–Flag einer TCP–Nachricht muß gesetzt, das ACK–Flag darf nicht gesetzt sein. Das Paket ist also das erste eines Verbindungsaufbaues und kommt vom Client.
`! -y`	Das ACK–Flag einer TCP–Nachricht muß gesetzt sein. Das heißt, das Paket ist entweder der zweite Teil des Verbindungsaufbaues oder, es ist ein Teil einer bestehenden Verbindung. Ist weder `-y` noch `! -y` gesetzt, werden die TCP–Flags nicht überprüft.
`-s` *Adresse* [*Port*]	Absender–Adresse des Paketes. Wenn keine Absenderadresse angegeben wird, so sind alle Adressen angesprochen. Wenn zusätzlich ein Port oder ein Portbereich (Port:Port) angegeben ist, so gilt die Regel nur für diese Ports. Sind keine Ports angegeben, so sind alle Ports gemeint.
`-d` *Adresse* [*Port*]	Empfänger–Adresse des Paketes. Ports wie bei der Absender–Adresse.
`-j` *Policy*	Policy dieser Regel. Gültige Policies sind `ACCEPT, REJECT` und `DENY`. In der `forward`–Chain ist auch die Policy `MASQ` für Masquerading zulässig.
`-l`	Logbucheintrag. Jedesmal, wenn diese Regel zutrifft, wird eine `syslog`–Meldung der Herkunft `kern` und der Priorität `info` erzeugt.

6.3 Format der Angaben

6.3.1 IP–Adressen

Eine Firewall–Regel kann sowohl eine Sender–, als auch eine Empfängeradresse beinhalten. In diesem Fall gilt die Regel dann eben nur für diese Adresse. Statt der Adressen können auch die Rechnernamen angegeben werden, das hat aber ein paar gewichtige Nachteile. Zur Namensauflösung muß ein funktionierendes DNS–System vorhanden sein. Diese

Voraussetzung könnte – zumindestens am Anfang einer Regelkette – noch nicht gegeben sein, wenn DNS selbst einer Beschränkung unterliegt. Es ist daher grundsätzlich ratsam, statt Namen eben IP–Adressen zu verwenden. Zum Anderen können DNS–Namen leichter gefälscht werden, als IP–Adressen.

IP–Adressen können dazu noch mit einer Maske versehen werden, die die signifikannten Bits der Adresse angibt. Diese Maske wird als Bitmaske realisiert und einfach mit einem Slash (`/`) hinten an die Adresse angehängt. Die Bedeutung ist einfach, die Adressangabe

```
192.168.100.123/24
```

bedeutet, daß die ersten 24 Bit der Adressangabe mit der gefundenen Adresse übereinstimmen müssen, damit die Regel greift. In diesem Beispiel sind das also alle Adressen, die vorne `192.168.100` stehen haben. Eine Maske /32 bedeutet also, daß die Adresse exakt übereinstimmen muß, eine Maske /0 bedeutet, daß kein Bit übereinstimmen muß, also alle Adressen gemeint sind. Dafür ist auch die Abkürzung `any/0` zulässig.

6.3.2 Ports

Ports können sowohl als Portnummer, als auch als symbolische Namen angegeben werden, wie sie in `/etc/services` eingetragen sind. Allerdings hat der Eintrag des symbolischen Namens den Nachteil, daß diese Namen nicht immer gleich sind, sie sind nicht fest vorgegeben. Bei einem Distributionswechsel könnte es vorkommen, daß eine Firewallregel nicht mehr funktioniert, wenn eben die Namen der Ports sich verändert haben. Andererseits ist natürlich ein Name aussagekräftiger, als eine Nummer.

6.4 Praktischer Aufbau einer Firewall

Eine Firewall wird grundsätzlich aufgebaut, indem jede einzelne Regel durch einen Aufruf von `ipchains` angegeben wird. Das ist eine Aufgabe, die natürlich nicht jedesmal von Hand ausgeführt wird, sondern immer in Form eines Scripts erledigt werden sollte. Ein solches Script kann nach jedem Neustart neu geladen werden und es wird nichts dabei vergessen. Es ist in fast allen Fällen unsinnig, einzelne Regeln von Hand im Nachhinein einzufügen. Ein solches Script muß selbst erstellt werden, auch wenn Distributoren wie etwa S.u.S.E schon vorgefertigte Scripts enthalten. Wir werden uns hier ein solches exemplarisches Script erstellen.

6.4.1 Grundeinstellungen

Zum besseren Verständnis der einzelnen Regeln ist es üblich, einzelne, immer wiederkehrende Begriffe, Adressen o.ä. als Konstanten vorzudefinieren. Fangen wir mit einem einfachen Beispiel an, in dem die Firewall nur sich selbst schützt (Das vollständige Script liegt im Anhang ab Seite 72 nochmal zur Einsicht):

```
EXTERNAL_INTERFACE=eth0         # Das fremde Netz
LOOPBACK_INTERFACE=lo           # Local Loopback
IPADDR=192.168.100.1            # Eigene Adresse
ANYWHERE=any/0                  # Jede IP-Adresse
MY_ISP=123.45.67.89/16          # Der Bereich meines Providers
LOOPBACK=127.0.0.0/8            # Loopback-Adressbereich
CLASS_A=10.0.0.0/8              # Reservierter Bereich Klasse A
CLASS_B=172.16.0.0/12           # Reservierter Bereich Klasse B
CLASS_C=192.168.0.0/16          # Reservierter Bereich Klasse C
CLASS_D=224.0.0.0/4             # Komplette Klasse D
```

```
CLASS_E=240.0.0.0/5            # Komplette Klasse E
BROADCAST_SRC=0.0.0.0          # Broadcast Absender
BROADCAST_DEST=255.255.255.255 # Broadcast Empfänger
PRIVPORTS=0:1023               # Privilegierte Portnummern
UNPRIVPORTS=1024:65535         # Unprivilegierte Portnummern
```

Der erste Schritt, zum Aufbau einer neuen Regelsammlung ist immer das Löschen eventuell schon bestehender Regeln. Das Programm `ipchains` kann seine Chains mit dem Parameter -F (Flush) löschen. Wenn keine Chain angegeben wird, dann löschen wir alle eingebauten Chains, also `input, output` und `forward`.

```
# Alle bestehenden Regeln löschen
ipchains -F
```

Alle Ketten sind jetzt leer. Allerdings haben wir noch nicht die Grund-Policies gelöscht bzw. verändert. Sie bleiben auch nach dem Löschen vorhanden. Also stellen wir jetzt diese Policies ein, für jede Kette extra. Dazu stellt ipchains den Parameter -P (nicht verwechseln mit -p) zur Verfügung:

```
# Voreingestellte Policies setzen
ipchains -P input    DENY
ipchains -P output   REJECT
ipchains -P forward  REJECT
```

Ab diesem Punkt ist – zumindestens für unseren Rechner – jeder Netzwerkverkehr gesperrt. Die Grundeinstellungen lehnen alles ab, einzelne Regeln haben wir noch nicht. Nun geben wir dem Loopback Interface alle Rechte. Von diesem Interface droht uns keinerlei Gefahr, aber einzelne lokale Dienste wie Drucker oder X11 können nicht ohne Loopback funktionieren:

```
# Loopback ohne Einschränkungen
ipchains -A input  -i $LOOPBACK_INTERFACE -j ACCEPT
ipchains -A output -i $LOOPBACK_INTERFACE -j ACCEPT
```

6.4.2 Ein paar Techniken zur Abwehr von gefälschten Paketen

Nun haben wir eine Grundeinstellung, mit der sich bereits arbeiten lässt. Allerdings ist bisher außer dem lokalen Loopback noch nichts erlaubt. Bevor wir jetzt weitere Regeln erstellen ist es sinnvoll, noch ein paar Sicherheitsmechanismen gegen SYN–Flooding und gefälschte IP–Pakete einzubauen. Der Linux–Kernel enthält bereits einige dieser Mechanismen, die wir nur noch aktivieren müssen. Dazu stehen ein paar sehr Linux–typische Wege zur Verfügung.

Das Verzeichnis `/proc` enthält Dateien, die eigentlich keine Dateien sind, sondern Schnittstellen zum Kernel. Einige dieser Dateien ermöglichen es, bestimmte Fähigkeiten des Kernels ein– oder auszuschalten. Das geschieht, indem in diese Dateien eine 1 oder eine 0 hineingeschrieben wird. Für uns sind dabei insbesondere die folgenden Dateien interessant:

- `/proc/sys/net/ipv4/tcp_syncookies`
  schaltet den Schutzmechanismus gegen SYN-Flooding ein/aus.

- `/proc/sys/net/ipv4/conf/*/rp_filter`
  Das Verzeichnis `/proc/sys/net/ipv4/conf` enthält für jedes IP–Interface ein Unterverzeichnis, in dem jeweils eine Datei `rp_filter` liegt. Diese Dateien ermöglichen es, die sogenannte Source–Address–Verification einzuschalten.

In all diese Dateien schreiben wir jetzt eine 1 hinein, um die jeweiligen Mechanismen zu aktivieren. Auch das geschieht innerhalb unseres Scripts:

```
# SYN_COOKIES aktivieren
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# SOURCE ADDRESS VERIFICATION aktivieren
for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
  echo 1 > $i
done
```

6.4.3 Ausfiltern offensichtlich falscher Adressen

Es gibt ein paar Adressen, die wir grundsätzlich ablehnen können, weil sie nie legal aus dem externen Netz (Internet) kommen können. Dazu zählt zuerst einmal die eigene IP–Adresse, danach alle Adressen aus reservierten Adressbereichen. Also verbieten wir zunächst jedes Paket, das hereinkommt und vorgibt von unserer eigenen Adresse zu stammen:

```
# Pakete ablehnen, die vorgeben von der eigenen Adresse zu stammen
ipchains -A input -i $EXTERNAL_INTERFACE -s $IPADDR -j DENY -l
```

Die beiden verwendeten Begriffe `$EXTERNAL_INTERFACE` und `$IPADDR` hatten wir ja oben bei der Konstantendefinition festgelegt. Das angehängte `-l` bedeutet, daß, falls diese Regel zutrifft eine Meldung an den `syslogd` geht und der Vorfall protokolliert wird.

Der nächste Schritt ist das Verbot aller ein– und ausgehender Pakete mit reservierten Adressen – sowohl Sender– als auch Empfängeradressen. Wir hatten ja bei der Definition der Konstanten extra diese Bereiche definiert, jetzt brauchen wir sie:

```
# Reservierte A-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_A -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_A -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_A -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_A -j DENY

# Reservierte B-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_B -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_B -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_B -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_B -j DENY

# Reservierte C-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_C -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_C -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_C -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_C -j DENY
```

Noch eine Adressform, die denkbar ungültig ist, ist die Verwendung einer Loopback–Adresse als Absender–Adresse vom externen Interface. Weder ein– noch ausgehende Pakete dürfen das vorweisen:

```
# Pakete mit Loopback als Absender verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $LOOPBACK -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $LOOPBACK -j DENY
```

Als nächstes filtern wir illegale Broadcast Adressen aus, das heißt Broadcast Adressen, die die Broadcast–Absender–Adresse als Empfänger–Adresse haben oder umgekehrt:

```
# Pakete mit illegalen Broadcast Adressen verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $BROADCAST_DEST -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $BROADCAST_SRC -j DENY
```

Jetzt folgen die Abweisung der Klasse-D Adressen im Absender–Feld. Klasse D–Adressen sind sogenannte Multicast Adressen, die grundsätzlich nur im Empfänger–Feld auftauchen dürfen. Wir verbieten diese Adressen im Absenderfeld sowohl für ein–, als auch für ausgehende Pakete:

```
# Pakete mit Klasse-D Adresse als Absender verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_D -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_D -j REJECT
```

Als nächstes verbieten wir alle Arten von Klasse–E Adressen, weil diese Adressen reserviert und daher nie legal sein können. Hier genügt es, die eingehenden Pakete zu untersuchen:

```
# Klasse-E Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_E -j DENY
```

Neben den reservierten Adressen der verschiedenen Klassen, hat die zentrale Vergabestelle für Internetadressen (IANA) noch einige weitere A– und B–Klasse Adressen reserviert. Diese Adressen dürfen nicht im öffentlichen Internet auftauchen, also schließen wir sie aus. Es handelt sich um die Netzadressen

- 0–2.*.*.*
- 5.*.*.*
- 7.*.*.*
- 23.*.*.*
- 27.*.*.*
- 31.*.*.*
- 37.*.*.*
- 39.*.*.*
- 41–42.*.*.*
- 58–60.*.*.*

- 65–69.*.*.*
- 80–95.*.*.*
- 96–126.*.*.*
- 217–219.*.*.*
- 220–223.*.*.*

Das ist viel Arbeit, weil leider bei einigen Fällen die Masken andere, legale Adressen mitnehmen würden. Es bleibt uns also nichts anderes übrig, als diese Adressen einzeln anzugeben:

```
# Von der IANA reservierte Adressen verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s 1.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 2.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 5.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 7.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 23.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 27.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 31.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 37.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 39.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 41.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 42.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 58.0.0.0/7 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 60.0.0.0/8 -j DENY
# 65 entspricht 01000001 - also würde die Maske /3 leider die 64
# mit ansprechen. Wir müssen daher 65-79 einzeln angeben
ipchains -A input -i $EXTERNAL_INTERFACE -s 65.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 66.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 67.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 68.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 69.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 70.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 71.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 72.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 73.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 74.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 75.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 76.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 77.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 78.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 79.0.0.0/8 -j DENY
# 80-95 lässt sich mit der Maske /4 ansprechen
ipchains -A input -i $EXTERNAL_INTERFACE -s 80.0.0.0/4 -j DENY
# 96-111 lässt sich mit der Maske /4 ansprechen
ipchains -A input -i $EXTERNAL_INTERFACE -s 96.0.0.0/4 -j DENY
# 126 entspricht 01111110 - die Maske /3 würde 127 beinhalten
# daher müssen wir 112 - 126 einzeln angeben
ipchains -A input -i $EXTERNAL_INTERFACE -s 112.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 113.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 114.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 115.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 116.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 117.0.0.0/8 -j DENY
```

```
ipchains -A input -i $EXTERNAL_INTERFACE -s 118.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 119.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 120.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 121.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 122.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 123.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 124.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 125.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 126.0.0.0/8 -j DENY
# 217-219 einzeln
ipchains -A input -i $EXTERNAL_INTERFACE -s 217.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 218.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 219.0.0.0/8 -j DENY
# 220-223 -> /6
ipchains -A input -i $EXTERNAL_INTERFACE -s 220.0.0.0/6 -j DENY
```

Damit hätten wir die offensichtlich ungültigen Adressen jetzt ausgefiltert. Alleine das Volumen dieser Regeln zeigt nochmal deutlich, daß sie nicht jedesmal beim Start neu eingegeben werden sollten. Natürlich kann so viel Information nur in Form von Scripten verarbeitet werden. Jetzt können wir anfangen, bestimmte Dinge zuzulassen – wir erinnern uns, die voreingestellte Policy ist ja DENY, alles ist verboten, was nicht grundsätzlich erlaubt ist.

6.4.4 ICMP–Nachrichten filtern

ICMP, das Internet Control Message Protocol, wird benutzt, um Fehler– bzw. Kontrollmechanismen zu tansportieren. Es arbeitet auf der IP–Schicht und somit eigentlich nicht mit Portnummern. Trotzdem unterscheidet ICMP verschiedene Nachrichtentypen, die aus der Sicht der Firewall wie Portnummern verwendet werden. Eine Liste der wichtigsten Nachrichtentypen samt ihren Portnummern finden Sie im Anhang auf Seite 71.

Vier ICMP–Typen müssen grundsätzlich die Firewall passieren: `source-quench, parameter-problem`, eingehende `destination-unreachable` sowie ausgehende `destination-unreachable` Nachrichten vom Subtyp `fragmentation-needed`. Vier weitere Typen sind optional, dazu gehören die beiden für ping notwendigen Typen `echo-request` und echo-reply, sowie alle anderen Sorten von abgehender `destination-unreachable` Meldungen und `time-exceeded`. Alle anderen Typen ignorieren wir, sie werden dann von der voreingestellten Policy abgewiesen…

Wir akzeptieren `source-quench` (Typ 4) Pakete. Sie werden zur Synchronisation benötigt, wenn ein Router schneller, als ein anderer ist.

```
# ICMP Typ 4 erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 4 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 4 -d $ANYWHERE -j ACCEPT
```

Die Darstellung ist mittels Backslash in zwei Zeilen gebrochen, das dient nur der besseren Lesbarkeit, nicht der Funktionalität. Wir erlauben also eingehende Pakete vom Typ ICMP–4 von überall her mit unserer Adresse als Empfänger, sowie ausgehende Pakete dieses Typs, von uns überall hin.

Genauso verfahren wir mit der Statusmeldung `parameter-problem` (ICMP–Typ 12). Wir erlauben diese Typen von überall her zu unserer Adresse und von unserer Adresse überall hin:

```
# ICMP Typ 12 erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 12 -d $ANYWHERE -j ACCEPT
```

Bei der Fehlermeldung `destination-unreachable` (ICMP–Typ 3) liegt die Sache etwas anders. Solche Fehlermeldungen entstehen nämlich während eines Portscans, wenn ein Angreifer versucht herauszufinden, welche Ports geöffnet sind. Die einfachste Form wäre jetzt, diesen Typ ganz zu sperren, das geht aber leider nicht, denn ein Untertyp dieser Meldung (`fragmentation-needed`) wird zwingend für das Aushandeln der Paketgröße im Netz benötigt. Aus diesem Grund können wir alle Typ–3 Nachrichten nur an den (oder die) Rechner unseres Providers zulassen, während wir den Subtyp `fragmentation-needed` an alle erlauben:

```
# ICMP-Typ 3 für Providerrechner erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 3 -d $MY_ISP -j ACCEPT

# ICMP-Typ 3 Subtyp fragmentation-needed für alle freigeben
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR fragmentation-needed -d $ANYWHERE -j ACCEPT
```

Die letzte zwingende ICMP–Nachricht ist die Statusmeldung `time-exceeded` (ICMP–Typ 11). Auch hier müssen wir nicht zwingend aller Welt erlauben, diese Nachricht von uns zu empfangen, es reicht unser Provider.

```
# ICMP-Typ 11 erlauben (ausgehend nur an unseren Provider)
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 11 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 11 -d $MY_ISP -j ACCEPT
```

Die wichtigste aller ICMP–Nachrichten wird im Programm ping benutzt. Mit Hilfe dieses Programms kann festgestellt werden, ob ein anderer Computer über das Netz zu erreichen ist, oder nicht. Der suchende Computer schickt an den gesuchten Computer ein `echo-request` (ICMP–Typ 8). Wenn der gesuchte Rechner dieses Paket empfängt, so schickt er ein `echo-reply` (ICMP–Typ 0 oder auch pong) zurück.

Üblich ist es, daß wir jeden Host im Internet anpingen dürfen, also gehen Typ 8 Pakete raus und Typ 0 Pakete rein:

```
# Ausgehendes Ping erlauben
ipchain -A output -i $EXTERNAL_INTERFACE -p icmp\
        -s $IPADDR 8 -d $ANYWHERE -j ACCEPT

ipchain -A input -i $EXTERNAL_INTERFACE -p icmp\
```

```
        -s $ANYWHERE 0 -d $IPADDR -j ACCEPT
```

Wiederum als Beispiel einer eingeschränkten Zulassung von außen, erlauben wir nur den Rechnern unseres Providers, uns anzupingen:

```
# Ankommendes Ping nur für Provider erlauben
ipchain -A input -i $EXTERNAL_INTERFACE -p icmp\
        -s $MYISP 8 -d $IPADDR -j ACCEPT

ipchain -A output -i $EXTERNAL_INTERFACE -p icmp\
        -s $IPADDR 0 -d $MYISP -j ACCEPT
```

Damit sind die notwendigen ICMP–Nachrichten erlaubt, die unnötigen verhindert. Jetzt können wir uns langsam an die echten Netzwerkdienste heranwagen...

6.4.5 Dienste auf unprivilegierten Ports

Dienste, die auf unprivilegierten Ports laufen, erschweren den Aufbau einer Firewall erheblich. Unprivilegierte Ports haben eine Doppelbedeutung, einerseits können Dienste diese Ports nutzen, andererseits können Clients, die andere Dienste nützen, auf diese Portnummern zurückgreifen (siehe Seite 15).

Mit dem TCP–Transportprotokoll ist diese Doppelbedeutung kein Problem, weil wir ja anhand der Steuerflags (SYN und ACK) unterscheiden können, ob das Paket von einem Client oder von einem Server stammt. Schwieriger wird es mit UDP. Aber eins nach dem anderen, schauen wir zunächst mal ein paar TCP–Dienste an:

Häufige TCP–Dienste auf unprivilegierten Ports

Ein typisches Beispiel ist ein Open Window Display Server (TCP–Port 2000). Ausgehende Verbindungen zu einem solchen Server sollte man sperren. Mit Hilfe der Option `-y` formulieren wir diese Regel so, daß sie nur für den Aufbau einer Verbindung eines lokalen Clients zu einem fremden Server gilt. Fremde Cleients, die zufällig den Port 2000 nutzen, um mit einem unserer Server in Verbindung zu treten, sind von dieser Regel also nicht betroffen. Genausowenig müssen wir eingehende Verbindungswünsche blockieren, denn Linux bietet keinen Open Window Display Server an.

```
# Open Window Verbindungsaufbau sperren
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
         -s $IPADDR -d $ANYWHERE 2000 -j REJECT
```

Bei X11-Display–Servern sieht die Sache ähnlich aus, jedoch müssen wir hier auch eingehende Verbindungen unterbinden. Ausgehende Verbindungen werden aus Sicherheitsgründen verboten, weil X11 grundsätzlich auch einen sicheren ssh–Eingang bietet, der immer vorzuziehen ist. X11–Display–Server benutzen die Ports 6000 bis 6063. Auf einem Rechner können also bis zu 64–Server laufen. Ein typischer Fall eines Portbereichs.

```
# X11 Aufbau zu einem fremden Server verbieten
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
         -s $IPADDR -d $ANYWHERE 6000:6063 -j REJECT

# X11 Aufbau von außen zu einem unserer Server verbieten
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp -y\
         -d $IPADDR 6000:6063 -j DENY -l
```

UDP–Dienste auf unprivilegierten Ports

UDP bietet viel weniger Information für das Betreiben einer Firewall, als TCP. Wir haben keinerlei Anhaltspunkt, ob ein Paket eine Client–Anfrage oder eine Server–Antwort ist. Das ist natürlich gerade bei den unprivilegierten Ports eine große Gefahr. Aus diesem Grund sollten wir UDP gänzlich sperren und nur einzelne Verbindungen zu ganz bestimmten Rechnern zulassen.

Im unprivilegierten Bereich spielt zum Glück nur ein wichtiger UDP–Dienst eine rolle, das Network File System NFS. NFS benutzt den Port 2049. NFS lässt sich zwar auch für TCP konfigurieren, aber das ist wesentlich unüblicher, auch bei TCP benutzt es den Port 2049.

In der Regel wird NFS ausschließlich im LAN benutzt, eine Firewall sollte es eigentlich selbst nicht brauchen. Wir sperren also die NFS–Dienste völlig:

```
# NFS (2049) über UDP sperren
ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -d $IPADDR 2049 -j DENY -l

# NFS (2049) über TCP sperren
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp -y\
         -d $IPADDR 2049 -j DENY -l

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
         -d $ANYWHERE 2049 -j DENY -l
```

6.4.6 Wichtige Dienste erlauben

Nachdem unsere Firewall jetzt soweit konfiguriert ist, können wir uns an die eigentlich wichtige Arbeit machen, die jeweilig benötigten Dienste zuzulassen. Im folgenden sind nur ein paar Beispiele genannt, die wirlich wichtig sind. Im Anhang ab Seite 90 sind die notwendigen Informationen zu den wichtigsten notwendigen Protokollen zusammengestellt.

DNS

Ohne DNS–Serverdienste wäre das Internet – und wohl heute auch die meisten Intranets – nicht zu gebrauchen. DNS ist also ein Dienst, auf den wir keinesfalls verzichten können. Normalerweise laufen alle DNS–Anfragen über UDP, nur wenn eine Antwort zu groß für ein UDP–Paket wäre, schickt der Server ein gekürztes Paket an den Client zurück, worauf der Client die gleiche Anfrage nochmal über TCP versuchen kann. In der Praxis wird das selten benötigt. Für den Abgleich der Daten zwischen primären und sekundären Nameservern einer Zone (dem sogenannten Zone Transfer) wird allerdings immer TCP benutzt. Um also Pakete durch unsere Firewall zu lassen, die eine UDP–Anfrage bei unserem primären Nameserver zu gestatten, beschränken wir uns auf diese eine IP–Adresse des Nameservers:

```
# DNS IP Adresse:
NAMESERVER=xx.xx.xx.xx

# UDP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR $UNPRIVPORTS \
         -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $NAMESERVER 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

Damit das Ganze auch über TCP funktioniert, insbesondere, damit ein eventueller Sekundärserver einen Zone–Transfer durchführen kann, benötigen wir noch zwei weitere Regeln, diesmal für TCP:

```
# TCP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $NAMESERVER 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

Beachten Sie hier die Angabe `! -y`, die besagt, daß das ACK–Flag gesetzt sein muß.

Wenn wir jetzt aber selbst einen Nameserver betreiben, und der in der Lage sein muß, sich mit einem übergeordneten Nameserver zu unterhalten, dann brauchen wir noch einen weiteren Satz Regeln. Diese Kommunikation zwischen zwei Nameservern findet nämlich – standardmäßig – auf dem Port 53 statt – und zwar bei beiden Kommunikationspartnern! Bisher hatten wir ja immer nur Kommunikation zwischen einem unprivilegierten Port mit dem Port 53 des Servers zugelassen. Also los:

```
#DNS Forwarding (Server zu Server)
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53 \
         -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $NAMESERVER 53 \
         -d $IPADDR 53 -j ACCEPT
```

Die letzte Form, die wir für DNS noch besprechen müssen, ist die Frage, ob auch Clients von außerhalb auf unseren Nameserver zugreifen dürfen sollen. Das ist nur dann nötig, wenn wir tatsächlich eine echte Domain mit autoritativen Nameserver betreiben. In der Regel werden wir den Zugriff auf einen bestimmten Kreis von IP–Adressen beschränken. Nennen wir diesen Kreis mal `MYDNSCLIENTS`.

```
# DNS Zugriff fremder Clients
MYDNSCLIENTS=ww.xx.yy.zz/mm

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $MYDNSCLIENTS $UNPRIVPORTS\
         -d $IPADDR 53 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53\
         -d $MYDNSCLIENTS $UNPRIVPORTS -j ACCEPT

# DNS Forwarding für fremde Clients
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53 \
         -d $MYDNSCLIENTS 53 -j ACCEPT
```

```
ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $MYDNSCLIENTS 53 \
         -d $IPADDR 53 -j ACCEPT
```

Für den sehr unwahrscheinlichen Fall, daß Sie einen Zone–Transfer von außen zulassen wollen, gibt es noch die Möglichkeit, auch diesen mit Regeln zuzulassen. Allerdings sollten wir hier die Liste der Clients, die diesen Zone–Transfer ausführen dürfen wirklich nur auf die existierenden Sekundären Server von außen beschränken, die diesen Transfer tatsächlich brauchen. Sonst kann jeder alle DNS Informationen unserer Domäne einfach frei Haus bekommen...

```
# DNS Zone-Transfer fremder Clients
MYDNSZONECLIENTS=ww.xx.yy.zz/mm

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
         -s $MYDNSZONECLIENTS $UNPRIVPORTS\
         -d $IPADDR 53 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $IPADDR 53\
         -d $MYDNSZONECLIENTS $UNPRIVPORTS -j ACCEPT
```

Der identd–Service

Der `identd`–Dienst (manchmal auch `auth` genannt) läuft auf dem TCP–Port 113. Er ist für viele anderen Dienste zwingend notwendig, etwa für das Versenden von E-Mail. Der Server unseres Rechners stellt dabei Informationen über einen bestimmten User zur Verfügung, etwa, ob es diesen User überhaupt gibt. . . .

Wenn unser Rechner einen FTP– oder Mailserver betreibt, dann fungiert er gleichzeitig immer auch als auth–Client. Es gibt keinen sicherheitsrelevanten Grund, warum unser Rechner soche Anfragen nicht stellen dürfte, also lassen wir solche Anfragen zu:

```
# abgehende auth-Anfragen
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE 113 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 113 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

Ob wir diesen Dienst tatsächlich auch anbieten sollen, ist Ansichtssache. Wenn wir uns entscheiden, diesen Dienst zu aktivieren (in /etc/inetd.conf), dann brauchen wir die beiden folgenden Regeln, damit er funktioniert:

```
# ankommende auth-Anfragen
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 113 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
```

```
        -s $IPADDR 113 \
        -d $ANYWHERE $UNPRIVPORTS -j ACCEPT
```

Sollten wir uns gegen diesen Dienst entscheiden, so ist es in diesem einzigen Fall günstiger, das Paket mit REJECT statt mit DENY zu behandeln. Ansonsten entstehen hohe Timeout–Wartezeiten. Also:

```
# ankommende auth-Anfragen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
        -s $ANYWHERE\
        -d $IPADDR 113 -j REJECT
```

Die weiteren Dienste, die wir vielleicht aktivieren wollen, können jetzt anhand der Beschreibungen im Anhang ab Seite 90 aufgebaut werden. Das grundlegende Prinzip einer Firewall sollte jetzt klar geworden sein. Allerdings hat unsere Firewall bisher noch einen Nachteil, der sie eindeutig in die Amateur–Klasse verweist. Sie schützt im Augenblick nämlich nur sich selbst. In den nächsten Kapiteln werden wir das ändern. . .

Um unsere erste Firewall zu aktivieren, muß das Script, das wir gerade erstellt haben nur noch ausführbar gemacht werden und aufgerufen werden. Die Firewall läuft. Natürlich sollte das in einem automatisierten Start–Script – je nach verwendeter Distribution – geschehen.

Kapitel 7

Die Firewall als Router ins Internet

Heute sind lokale Netze etwas selbstverständliches geworden, sowohl die Kosten für solche Netze haben sich dramatisch nach unten bewegt, als auch das Wissen, das nötig ist, um Computer zu vernetzen ist allgemein gestiegen.

Aus dieser Verbreitung von lokalen Netzen folgert sehr schnell der Wunsch, daß ein lokales Netz einen Anschluß ans Internet findet, ohne daß jeder Rechner über ein Modem oder eine ISDN-Verbindung verfügen muß. Es existieren einige Lösungsansätze für dieses Problem, die zunächst einmal kurz dargestellt werden sollen:

7.1 Router oder Proxy

Die Begriffe Router und Proxy werden gerne und oft falsch oder zumindestens unscharf benutzt. Dabei handelt es sich in beiden Fällen um technische Lösungen für das Problem der Anbindung eines lokalen Netzes an das Internet. Der Lösungsansatz findet jedoch an völlig verschiedenen Stellen statt.

7.1.1 Die Proxy–Technik

Proxies wurden schon relativ früh eingesetzt, als Betreiber von einzelnen Netzen oder Internet–Provider gemerkt hatten, daß manche Web–Seiten sehr häufig aufgerufen wurden. Das führte zu der Idee, daß eine lokale Zwischenspeicherung dieser häufig aufgerufenen Webseiten den Datenverkehr im Internet bzw. in der Anbindung zum Internet erheblich reduzieren könnte. Denn anstatt eine Seite pro Tag x mal über diese Internetanbindung zu transportieren, wurde sie nur noch einmal geholt, lokal zwischengespeichert und – falls jemand die Adresse dieser Seite aufrief – von diesem lokalen Zwischenspeicher heraus ausgeliefert.

Das heißt, ein Proxy–Server ist eine Art Webserver, der einen lokalen Zwischenspeicher für Webseiten bereithält und eingehende Anfragen an das Internet entgegennimmt. Sollte die gewünschte Seite schon lokal abgespeichert sein, so liefert er die gespeicherte Seite zurück, ist sie noch nicht gespeichert, so holt er sich die Seite aus dem Internet, speichert sie lokal zwischen und liefert sie dann erst an den entsprechenden Client aus.

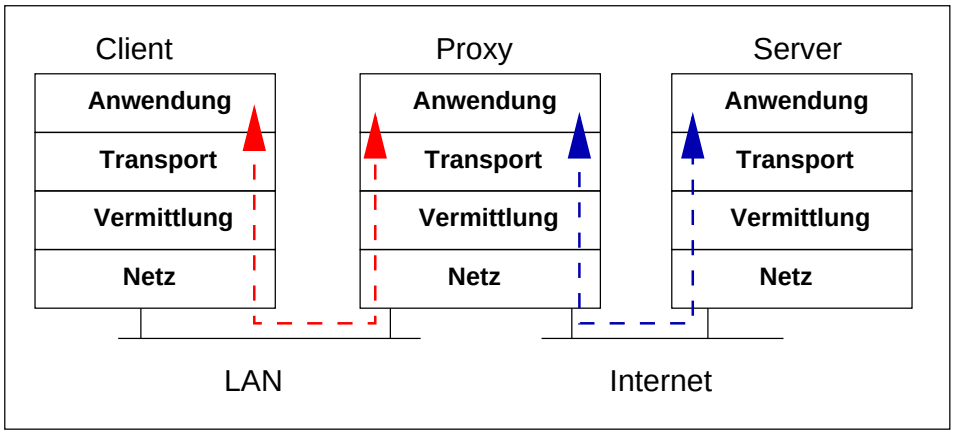

Bild 7.1: Die Proxy–Technik

Das hat nun den Nebeneffekt, daß ein Proxy, der eine Internetanbindung hat, den Rechnern im lokalen Netz auch einen Zugriff, zumindestens für WWW–Seiten erlauben kann, ohne daß die Rechner des lokalen Netzes selbst Zugang zum Internet haben. Sie erhalten alle Webseiten ja eben von diesem Proxy. Der Nachteil dieses Verfahrens liegt auf der Hand, ein Proxy arbeitet auf der Anwendungsschicht des TCP/IP Modells und bietet nur Funktionalität für ein (oder ein paar) Protokoll(e). Moderne Proxy–Systeme gibt es etwa für WWW und FTP. Direkter Internet–Zugang ist damit für die Rechner im Netz nicht möglich.

Was auf den ersten Blick wie ein Nachteil aussieht, kann aber auch genauso gewünscht sein. So ist es vorstellbar, daß ein Netzbetreiber eines lokalen Netzes gar nicht will, daß jeder Rechner im LAN bedingungslosen Zugriff ins Internet bekommt. Die notwendigen Dienste, wie Mail, WWW und FTP können über Proxies zur Verfügung gestellt werden, andere Dienste, wie etwa Netzspiele über das Internet sind so nicht möglich.

7.1.2 Die Router–Technik

Im Gegensatz zum Proxy arbeitet ein Router (oft auch Gateway genannt) auf der Verbindungsschicht des TCP/IP Modells. Das heißt, er gibt Pakete, die er erhält und die nicht für ihn bestimmt sind, von einem Netzwerk–Interface zu einem anderen weiter. Dabei existieren keine Einschränkungen hinsichtlich irgendwelcher Protokolle, denn Protokolle kennt der Router ja gar nicht, für ihn sind alle Pakete nur IP–Pakete, egal was sie weiter enthalten.

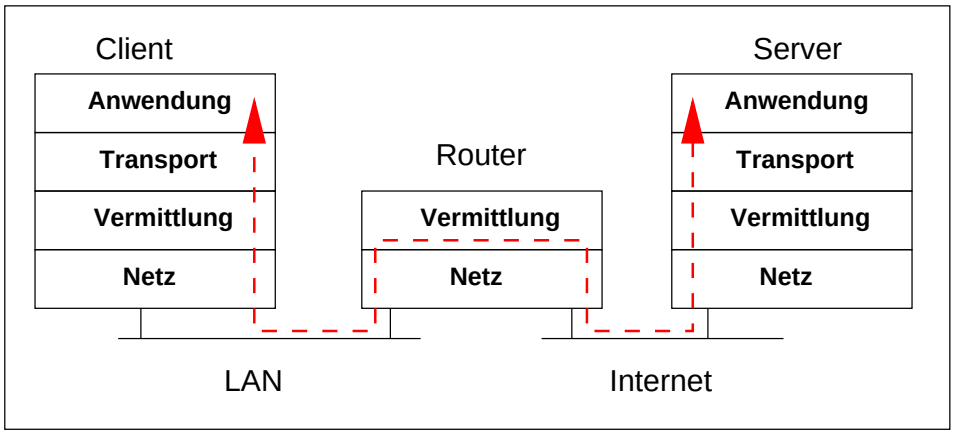

Bild 7.2: Die Router–Technik

Die Arbeit eines Routers im lokalen Netz, der zwei Netze miteinander verbindet, ist einfach zu verstehen. Denn der Router kann ja klar sagen, welche IP–Adresse zu welchem

Netz gehört und welches Netz an welcher Schnittstelle hängt. Was aus diesem Satz schon klar wird – und was ein großer Unterschied zum Proxy ist – ist die Tatsache, daß der Router tatsächlich IP–Pakete weiterleitet. Das ist in einem lokalen Netz weiter kein Problem, wir werden aber gleich sehen, daß es im Zusammenhang mit dem Internet sehr wohl zum Problem werden kann.

Im weiteren Verlauf interessiert uns natürlich die Router–Technik im Zusammenhang mit der Firewall. Dabei ist es uns natürlich unbenommen, auch Proxy–Server einzurichten, die Funktionalität der Firewall als Verbindung vom lokalen Netz ins Internet basiert aber auf der Weitergabe von IP–Paketen, also auf dem Routing.

7.2 Das Problem der reservierten Adressen

In der guten alten Zeit des Internets, als IP–Adressen noch nicht knapp waren und die Rechner, die ans Netz wollten das über eine Standleitung getan haben, war das Routing im Netz ein Kinderspiel. War doch damals die Situation die, daß jeder ans Internet angeschlossene Rechner eine feste, einmalige und von überall erreichbare IP–Adresse hatte. Ein Router hatte also nur noch die Aufgabe, ein Paket an die entsprechend richtige Schnittstelle auszuliefern und schon war alles erledigt.

Die Firewall eines Linux Rechners hat ja neben den beiden Regelketten `input` und `output` noch die chain `forward`. Diese Regelkette ist speziell für das Routing gedacht, das oft auch mit dem Begriff *IP–forwarding* bezeichet wird. Die Zusammenhänge der drei Ketten sind aus dem Bild auf Seite 32 zu entnehmen. Mit Hilfe dieser Regelkette ist es also auch noch möglich, zu entscheiden, welche Pakete geroutet werden sollen.

Heute jedoch haben wir eine Knappheit an IP–Adressen und die lokalen Netze laufen in der Regel mit den sogenannten reservierten Adressen. Diese reservierten Adressen sind im Internet nicht routbar. Das heißt, daß selbst wenn einer unserer Rechner ans Internet angeschlossen ist und dieser eine Rechner auch eine echte IP–Adresse[1] hat, und noch dazu als Router konfiguriert ist, eine Verbindung ans Internet für die anderen Rechner nicht so ohne weiteres möglich wäre. Die Rechner im lokalen Netz haben ja IP–Adressen, die im Internet nicht gültig sind. Selbst wenn ein IP–Paket mit einer solchen ungültigen Adresse jemals einen Internet–Server erreichen würde, so wäre dieser Rechner nicht in der Lage eine Antwort zurückzuschicken, denn es existieren keine Routen zu reservierten Adressen – wohin sollten die auch zeigen? Es gibt wahrscheinlich hunderte von lokalen Netzen auf der Welt, die diese Adressen benutzen.

[1] Besser: eine nicht reservierte, also routefähige Adresse

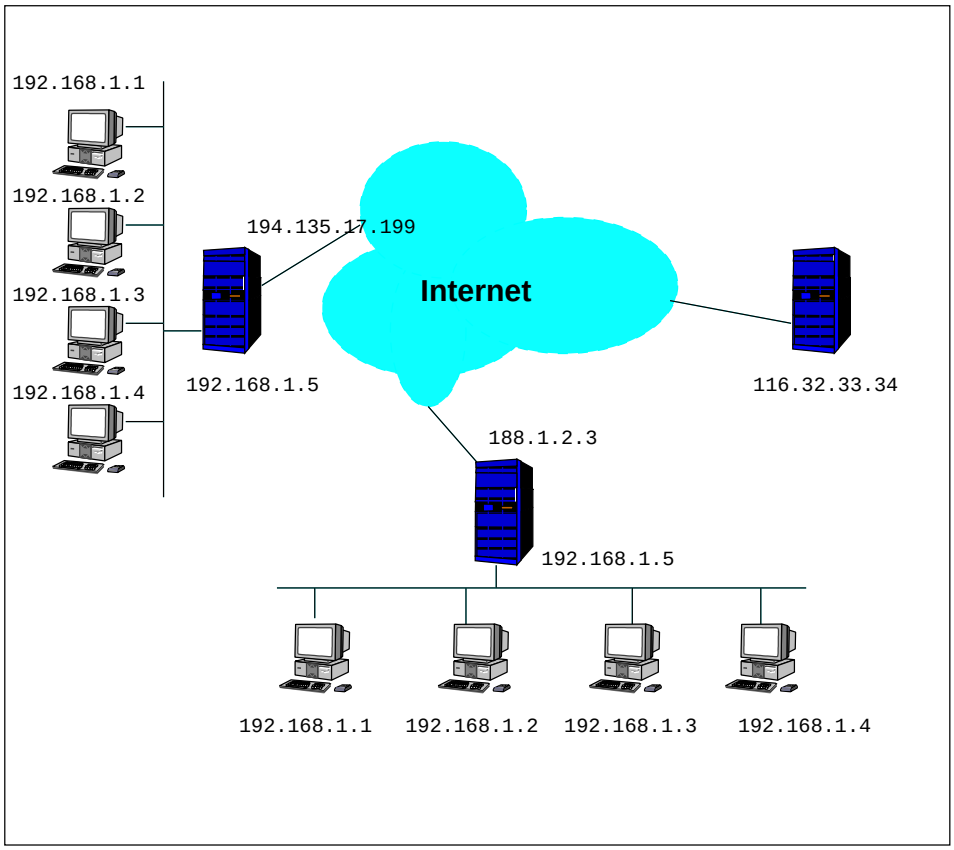

Bild 7.3: Reservierte Adressen im Internet

7.3 Masquerading als Problemlösung

Der Linux–Kernel bietet eine Lösungsmöglichkeit für die Verwendung reservierter Adressen im Internet an, das sogenannte Masquerading. Dabei arbeitet der Router, also der Rechner, der sowohl am lokalen Netz, als auch am Internet hängt, als Masquerading Server. Dieser Rechner hat ja eine echte, im Internet routfähige Adresse. Er nimmt jetzt aus dem lokalen Netz Pakete entgegen und verändert sie dergestalt, daß im Feld der Sender IP–Adresse jetzt nicht mehr die reservierte Adresse des wahren Absenders steht, sondern die echte Adresse des Masquerading–Servers. Gleichzeitig merkt sich der Masquerading Server die Original–Adresse in einer Kernel–Tabelle. Bekommt jetzt der Masquerading Server die Antwort aus dem Internet, so überprüft er anhand dieser Tabellen, für wen diese Antwort war, tauscht die entsprechenden Adresen der Pakete wieder um und gibt das so veränderte Antwortpaket wieder an den Rechner im lokalen Netz, der die Anfrage losgeschickt hatte.

Aus der Sicht des Internet–Servers existiert also gar kein lokales Netz, er kommuniziert ausschließlich mit dem Masquerading Server. Die Adressen des lokalen Netzes sind maskiert. Das hat natürlich den schon erwähnten Vorteil, daß Angriffe aus dem Internet sowieso nur auf den Router stattfinden können, niemals direkt gegen einen Rechner im lokalen Netz. Denn dieses Netz ist nicht adressierbar!

Auf der Ebene von `ipchains` wird das Masquerading genauso behandelt, als wäre es eine Policy. Diese Policy ist aber logischerweise nur für die `forward`–chain gültig. Das angegebene Netzwerkinterface (`-i` bezieht sich immer auf das externe Interface, auf dem die realen Adressen maskiert sein sollen. So wird etwa durch den Befehl

```
ipchains -A forward -i $EXTERNAL_INTERFACE -s 192.168.100.0/24 -j MASQ
```

das Masquerading für das Netz 192.168.100.0 aktiviert. Das obige Beispiel erlaubt grundsätzlich alle Arten von Datenverkehr, natürlich könnten wir auch alle Regeltechniken, wie etwa Ports oder Protokolle hier benutzen, es handelt sich ja tatsächlich um eine Policy...

Die drei Regelketten `input, forward` und `output` werden in dieser Reihenfolge nacheinander abgearbeitet. Das bedeutet, daß nur diejenigen Pakete, die durch die `input`–chain durchkamen, and die `forward`–chain weitergegeben werden. Und nur diejenigen Pakete, die durch diese Kette kommen werden an die `output`–chain weitergereicht. Erst, wenn das Paket auch durch diese Kette durch ist, ohne ausgesondert worden zu sein, gelangt es tatsächlich ins Internet. Für aus dem Internet ankommende Pakete gilt dann wieder das entsprechende.

7.4 Verschiedene Modelle von routenden Firewalls

Eine Firewall, die nicht nur sich selbst schützt, wie die aus dem letzten Kapitel, sondern auch gleichzeitig als Router fungiert, wird als *dual homed host* bezeichnet, also ein Host, der in zwei Netzen zuhause ist. Mit dieser Technik ist es möglich, verschiedene Sicherheitsstufen zu realisieren, je nach Anforderung des zu schützenden Netzes. Im Folgenden werden die beiden wichtigsten Modelle dargestellt, die Bastion–Host–Firewall und die Demilitarisierte Zone (DMZ).

7.4.1 Die Bastion–Host–Firewall

Die einfachste – aber auch unsicherste – Form einer routenden Firewall ist die Bastion–Host–Firewall. Sie besteht grundsätzlich aus einem Rechner, der als *dual homed host* zwischen dem zu schützenden Netz auf der einen Seite und dem unsicheren Netz (Internet) auf der anderen Seite plaziert ist. Dieser Rechner routet Pakete von einem Netz ins andere – ob mit Masquerading oder ohne[2]. Jedes Paket, das durch diesen Rechner geroutet werden soll, muß die drei Regelketten der Firewall durchlaufen.

Der Begriff Bastion stammt aus dem Militärischen, er kann aber auch wirklich so verstanden werden. Eine Bastion ist eine Verteidigungseinrichtung, wenn sie fällt, dann gibt es keinen Schutz mehr. Diese Analogie kann ungebremst ins Computernet übernommen werden. Wenn die Bastion–Host–Firewall geknackt werden würde, dann läge das gesamte Netz schutzlos da, der Angreifer, der die Firewall überwunden hätte, kan im LAN tun und lassen, was er will.

Trotzdem ist dieses Modell für die meisten kleinen Netze völlig ausreichend, wenn das zu schützende Netz nicht in großem Maß selbst Dienste im Internet anbieten soll. In der Regel ist das aber ja auch gar nicht möglich, denn normalerweise würden wir in diesem Modell sowieso mit reservierten Adressen im lokalen Netz arbeiten – damit haben wir uns der Möglichkeit beraubt, selbst Dienste anzubieten.

[2] Ohne natürlich nur dann, wenn das zu schützende Netz echte, routfähige IP–Adressen benutzt

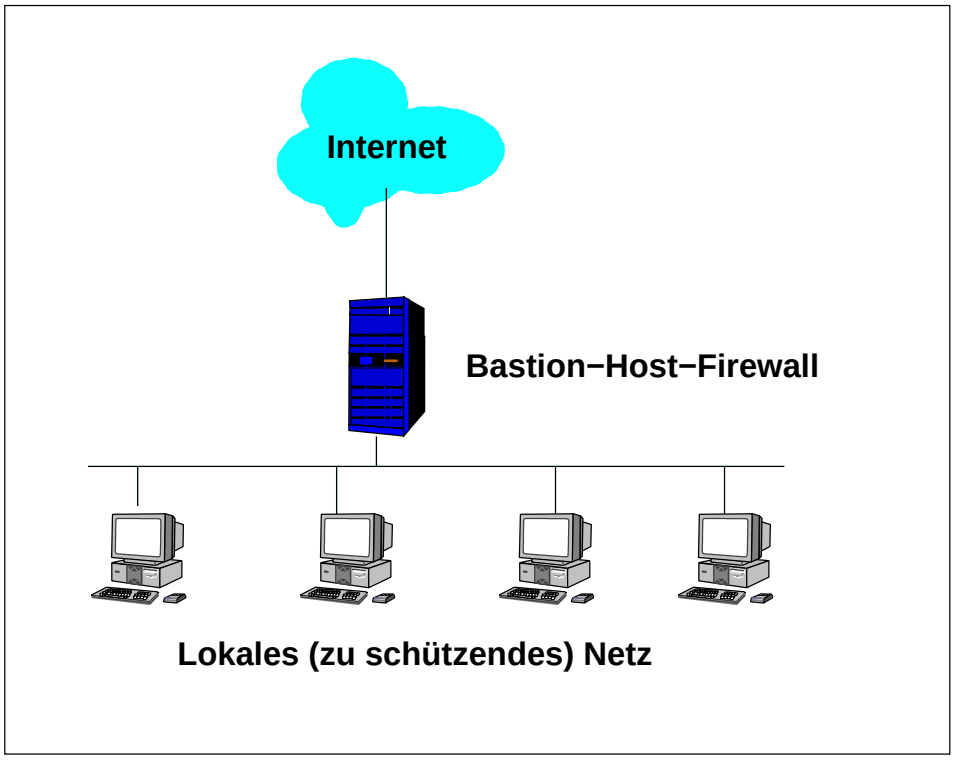

Bild 7.4: Bastion Host Firewall

7.4.2 Die Demilitarisierte Zone (DMZ)

Erheblich sicherer für große Netze ist das Modell der demilitarisierten Zone, manchmal auch Perimeter Netzwerk genannt. Dieses Modell trennt das zu schützende Netz und das öffentlich zugängige Netz nochmal auf und arbeitet folgerichtig mit zwei Firewall–Rechnern.

Das lokale Netz, das die eigentlichen Arbeitsplätze verbindet, ist weiterhin das zu schützende Netz. Zwischen diesem lokalen Netz und der bösen großen weiten Welt des Internet liegt jetzt aber noch ein Netz, die demilitarisierte Zone. Wir haben zwei Firewall–Rechner, einmal der Bastion–Host, der das Internet mit der DMZ verbindet, und zum zweiten die sogenannte Choke–Firewall. Sie verbindet das zu schützende Netz mit der DMZ. Innerhalb der DMZ stehen jetzt die Rechner, die von außerhalb zugänglich sein sollen. Also etwa der Webserver des Unternehmens, der FTP–Server oder ähnliches. Die Rechner der DMZ müssen dabei aber tatsächlich echte, routfähige IP–Adressen besitzen, damit dieses Modell funktioniert.

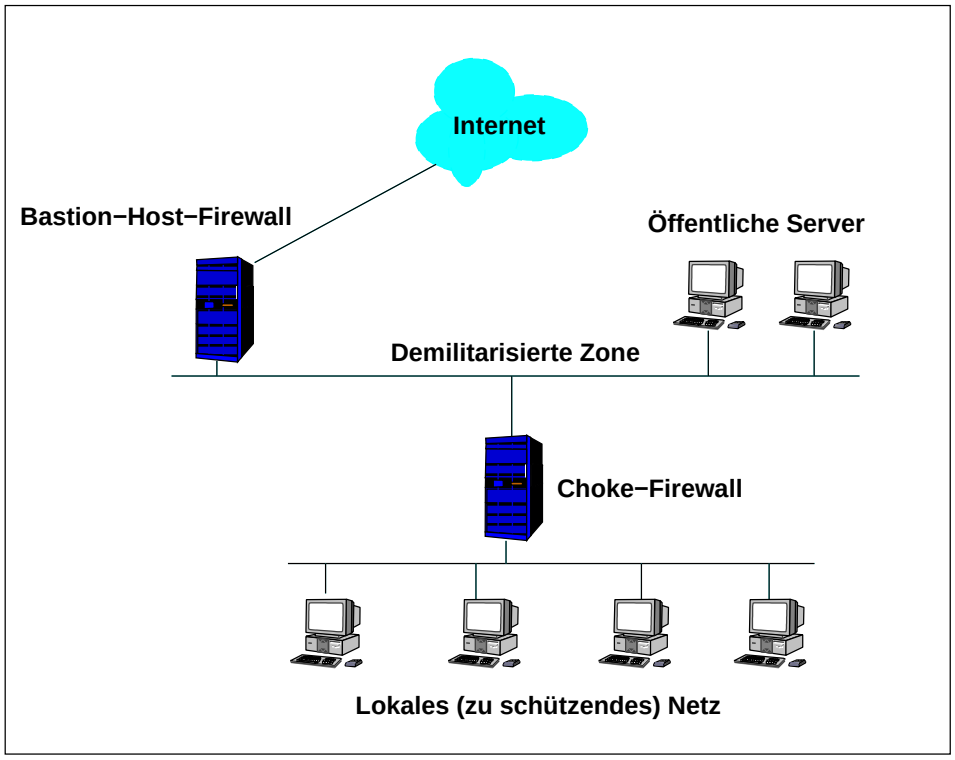

Bild 7.5: DMZ Firewall

Diese Form der Firewall ist natürlich deutlich schwieriger zu verwirklichen, dafür aber auch wesentlich sicherer. Es existiert kein einziger Knackpunkt mehr, an dem alleine alle Sicherheit hängt. Der Ausfall bzw. die Eroberung eines Elements alleine führt noch nicht zur Unsicherheit des zu schützenden Netzes. Im nächsten Kapitel werden wir detailiert eine solche Firewall[3] aufbauen.

[3] Wenn hier von einer Firewall die Rede ist, dann meint dieser Begriff nicht die einzelnen Rechner, denn Firewallrechner haben wir ja zwei in diesem Fall. Hier meint der Singular also die Gesamtheit des Sicherheitssystems, bestehend aus Bastion–Firewall, DMZ und Choke–Firewall.

Kapitel 8

Exemplarischer Aufbau einer DMZ–Firewall

Um die Komplexität des Aufbaus einer Firewall mit DMZ am Stück darzustellen, werden wir in diesem Kapitel eine komplette solche Firewall aufbauen. Zunächst einmal aber noch etwas Begriffsklärung für das Projekt:

Bastion Der Rechner, der das Internet mit der DMZ verbindet.

Choke Der Rechner, der das zu schützende Netz mit der DMZ verbindet.

DMZ Die demilitarisierte Zone, das Netz zwischen dem Internet und dem zu schützenden Netz.

LAN Das zu schützende Netz.

Wir gehen also davon aus, daß Bastion und Choke beide als Gateway (dual homed host) zur DMZ dienen. Die DMZ enthält öffentliche und halböffentliche Server. Jedes Netzwerkinterface der beiden Firewall–Rechner (Bastion und Choke) besitzt eigene, individuelle Regeln. Wir benötigen also mindestens vier Regelsätze, je einen für das externe und interne Interface beider Maschinen. Die Regeln für das externe Interface der Bastion sind fast identisch mit denen aus dem Beispiel aus dem Abschnitt 6.4 (Seite 34).

Die wirklichen Neuigkeiten dieses Kapitels beziehen sich hauptsächlich auf die Schnittstellen zur DMZ, also die interne Schnittstelle der Bastion und die externe Schnittstelle der Choke. Diese Regeln verhalten sich spiegelbildlich zueinander. Für die öffentlichen Server innerhalb der DMZ muß es noch ein paar separate Regeln geben, wir gehen aber in diesem Beispiel davon aus, daß es sich hierbei um spezialisierte Rechner handelt, die jeweils nur einen Dienst anbieten und daher sehr einfache Regeln benötigen.

Um den Rahmen und die Übersicht über dieses Kapitel nicht zu gefährden, werden innerhalb des Kapitels Vereinfachungen zugelassen. Weil die externen Regeln der Bastion praktisch identisch sind mit denen aus Abschnitt 6.4, werden hier nur die Unterschiede dargestellt. Im Anhang ab Seite 78 sind die beiden Scripts für Bastion und Choke jedoch beide noch einmal vollständig abgedruckt.

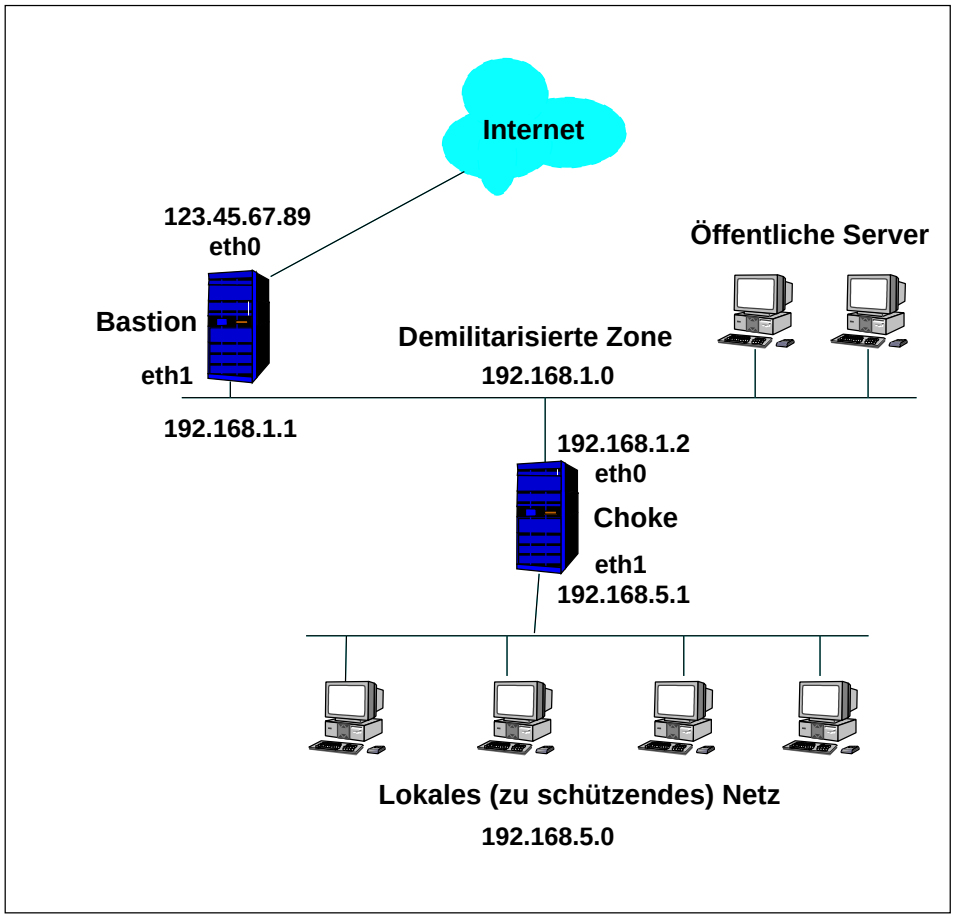

Bild 8.1: Der Aufbau des Beispiels

Das externe Interface der Bastion liegt auf der Schnittstelle `eth0` und hat die echte IP–Adresse 123.45.67.89 zugewiesen. Das interne Interface liegt auf `eth1` und hat die reservierte Adresse 192.168.1.1

Die DMZ hat die 192.168.1.0 als Netzwerkadresse.

Die Choke hat ihr externes Netz am Interface `eth0` und dort die Adresse 192.168.1.2, ihr internes Netz liegt auf `eth1` und bekommt die Adresse 192.168.5.1

Das LAN hat die Netzadresse 192.168.5.0 als Netzwerkadresse.

8.1 Die symbolischen Konstanten

Ein Firewallscript lässt sich grundsätzlich leichter lesen und hauptsächlich flexibler an Veränderungen anpassen, wenn wir nicht bei jeder Regel erneut die IP–Adressen angeben, sondern am Anfang symbolische Konstanten definieren, die wir im weiteren Verlauf dann benutzen. Das ist immer das erste, was beim Aufbau neuer Scripts zu beachten ist und daher halten wir uns an diese Vorgaben:

8.1.1 Die Konstanten für die Bastion

```
# Konstantendefinition
EXTERNAL_INTERFACE=eth0         # Das Interface ins Internet
IPADDR=123.45.67.89             # Adresse des Internetzugangs
MY_ISP=123.45.67.90/16          # Der Bereich meines Providers
```

```
BASTION_DMZ_INTERFACE=eth1     # internes Interface
BASTION_DMZ_IPADDR=192.168.1.1 # Adresse dazu

LOOPBACK_INTERFACE=lo          # Local Loopback
LOOPBACK=127.0.0.0/8           # Loopback-Adressbereich

CHOKE_DMZ_IPADDR=192.168.1.2   # ext. Interface der Choke
DMZ_ADDRESSES=192.168.1.0/24   # IP-Bereich der DMZ
DMZ_BROADCAST=192.168.1.255    # Broadcastadresse DMZ
ANYWHERE=any/0                 # Jede IP-Adresse

CLASS_A=10.0.0.0/8             # Reservierter Bereich Klasse A
CLASS_B=172.16.0.0/12          # Reservierter Bereich Klasse B
CLASS_C=192.168.0.0/16         # Reservierter Bereich Klasse C
CLASS_D=224.0.0.0/4            # Komplette Klasse D
CLASS_E=240.0.0.0/5            # Komplette Klasse E
BROADCAST_SRC=0.0.0.0          # Broadcast Absender
BROADCAST_DEST=255.255.255.255 # Broadcast Empfänger
PRIVPORTS=0:1023               # Privilegierte Portnummern
UNPRIVPORTS=1024:65535         # Unprivilegierte Portnummern
```

8.1.2 Die Konstanten für die Choke

```
#Konstantendefinition
CHOKE_DMZ_INTERFACE=eth0       # externes Interface der Choke
CHOKE_DMZ_IPADDR=192.168.1.2   # externe Adresse der Choke
CHOKE_LAN_INTERFACE=eth1       # internes Interface der Choke
CHOKE_LAN_IPADDR=192.168.5.1   # interne Adresse der Choke
LOOPBACK_INTERFACE=lo          # Local Loopback
LOOPBACK=127.0.0.0/8           # Loopback-Adressbereich

BASTION_DMZ_IPADDR=192.168.1.1 # interne Adresse der Bastion
DMZ_ADDRESSES=192.168.1.0/24   # IP-Bereich der DMZ
LAN_ADDRESSES=192.168.5.0/24   # IP-Bereich des LAN
DMZ_BROADCAST=192.168.1.255    # Broadcastadresse DMZ
ANYWHERE=any/0                 # Jede IP-Adresse

CLASS_A=10.0.0.0/8             # Reservierter Bereich Klasse A
CLASS_B=172.16.0.0/12          # Reservierter Bereich Klasse B
CLASS_C=192.168.0.0/16         # Reservierter Bereich Klasse C
CLASS_D=224.0.0.0/4            # Komplette Klasse D
CLASS_E=240.0.0.0/5            # Komplette Klasse E
BROADCAST_SRC=0.0.0.0          # Broadcast Absender
BROADCAST_DEST=255.255.255.255 # Broadcast Empfänger
PRIVPORTS=0:1023               # Privilegierte Portnummern
UNPRIVPORTS=1024:65535         # Unprivilegierte Portnummern
```

8.2 Weitere Grundeinstellungen

Wie üblich löschen wir zunächst einmal in beiden Scripts (für Choke und Bastion) die bisher bestehenden Regeln. Das ist nur eine Sicherheitsmaßnahme, um zu vermeiden, daß

eventuell existierende Regeln, die vor dem Aufruf des Scripts erstellt wurden, nicht weiter gelten. In beiden Scripts steht also jetzt

```
# Alle bestehenden Regeln löschen
ipchains -F
```

Die Chains sind jetzt gelöscht, nicht jedoch eine eventuell bestehende Policy. Also definieren wir sofort eine neue, die alles grundsätzlich verbietet, was nicht explizit erlaubt ist. Hier machen wir aber einen Unterschied zwischen Bastion und Choke. Die Bastion arbeitet grundsätzlich mit DENY, gibt also keinerlei Rückmeldungen zurück, außer ins interne Netz, also die DMZ. Das ist wichtig, weil solche Rückmeldungen einem potenziellen Angreifer schon bestimmte Dinge mitteilen kann. Die Choke jedoch arbeitet in einem Umfeld, das wesentlich mehr Vertrauen von uns verdient. Hier arbeiten wir mit REJECT, um Fehlermeldungen zu erzeugen und Timeout–Wartezeiten dadurch zu vermeiden. Das Script der Bastion erhält also folgenden Eintrag:

```
# Voreingestellte Policies setzen
ipchains -P input    DENY
ipchains -P output   REJECT
ipchains -P forward  REJECT
```

während die Choke überall mit REJECT arbeitet:

```
# Voreingestellte Policies setzen
ipchains -P input    REJECT
ipchains -P output   REJECT
ipchains -P forward  REJECT
```

An diesem Punkt sind alle Regeln gelöscht und alle Policies auf DENY oder REJECT gesetzt. Es ist keinerlei Netzverkehr mehr möglich. Wie im letzten Beispiel sollten wir jetzt zunächst wieder das Loopback–Device ermöglichen, dort sind Angriffe nicht möglich, es droht keinerlei Gefahr, wir lassen einfach alles auf diesem Device zu. Das gilt wieder für beide Maschinen, Bastion und Choke:

```
# Loopback ohne Einschränkungen
ipchains -A input  -i $LOOPBACK_INTERFACE -j ACCEPT
ipchains -A output -i $LOOPBACK_INTERFACE -j ACCEPT
```

Es kann auch nicht schaden, wenn wir an dieser Stelle wieder bei beiden Maschinen die Kernel–eigenen Schutzroutinen aktivieren, wie wir das auch schon im Abschnitt 6.4.2 gemacht hatten. Dort ist es schon erklärt, daher hier nur noch die entsprechenden Anweisungen:

```
# SYN_COOKIES aktivieren
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# SOURCE ADDRESS VERIFICATION aktivieren
for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
  echo 1 > $i
```

```
done
```

Jetzt filtern wir wieder die Adressen aus, die offensichtlich ungültig sind. Zunächst mal wieder jeweils die eigenen Adressen. Die Bastion Firewall bekommt also die Anweisungen

```
# Pakete ablehnen, die vorgeben von der eigenen Adresse zu stammen
ipchains -A input -i $EXTERNAL_INTERFACE \
         -s $IPADDR -j DENY -l
ipchains -A input -i $BASTION_DMZ_INTERFACE \
         -s $BASTION_DMZ_IPADDR -j DENY -l
```

während die Choke folgende Zeilen braucht:

```
# Pakete ablehnen, die vorgeben von der eigenen Adresse zu stammen
ipchains -A input -i $CHOKE_LAN_INTERFACE\
         -s $CHOKE_LAN_IPADDR -j REJECT -l
ipchains -A input -i CHOKE_DMZ_INTERFACE \
         -s $CHOKE_DMZ_IPADDR -j REJECT -l
```

Bei der Bastion belassen wir die Filterung der restlichen Adressen wie in unserem Beispiel aus Abschnitt 6.4. Die Choke hingegen muß nur Adressen aus dem Pool der reservierten A– und B–Klassen aussondern, weil sie ja selbst auf beiden Netzwerkkarten eine C–Klasse reservierte Adresse benutzt. Nebenbei filtern wir wie gehabt auch Loopback– und fehlerhafte Broadcasts aus. Die Angabe der von der IANA reservierten Adressen sparen wir uns für die Choke, die Bation hat sie ja schon.

```
# Pakete mit privaten A-Klasse Adressen als Sender oder
# Empfänger verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_A -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_A -j REJECT -l
ipchains -A output -i $CHOKE_DMZ_INTERFACE -d $CLASS_A -j REJECT -l
ipchains -A output -i $CHOKE_LAN_INTERFACE -d $CLASS_A -j REJECT -l

# Pakete mit privaten B-Klasse Adressen als Sender oder
# Empfänger verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_B -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_B -j REJECT -l
ipchains -A output -i $CHOKE_DMZ_INTERFACE -d $CLASS_B -j REJECT -l
ipchains -A output -i $CHOKE_LAN_INTERFACE -d $CLASS_B -j REJECT -l

# Pakete mit D-Klasse Adressen als Sender verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_D -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_D -j REJECT -l
ipchains -A output -i $CHOKE_DMZ_INTERFACE -s $CLASS_D -j REJECT -l
ipchains -A output -i $CHOKE_LAN_INTERFACE -s $CLASS_D -j REJECT -l

# Pakete mit E-Klasse Adressen verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_E -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_E -j REJECT -l

# Pakete mit Loopback Adressen als Sender verwerfen
```

```
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $LOOPBACK -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $LOOPBACK -j REJECT -l

# Pakete mit fehlerhaften Broadcast Adressen verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE \
         -s $BROADCAST_DEST -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE \
         -s $BROADCAST_DEST -j REJECT -l
ipchains -A input -i $CHOKE_DMZ_INTERFACE \
         -d $BROADCAST_SRC -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE \
         -d $BROADCAST_SRC -j REJECT -l
```

Damit hätten wir die Grundeinstellung bei beiden Scripts erledigt. Jetzt geht es an die eigentliche Arbeit, verspricht aber auch interessanter zu werden.

8.3 ICMP–Nachrichten filtern

Die Bastion hat schon Regeln für ICMP–Nachrichten, die sich bisher aber nur auf die externe Schnittstelle, also auf die Schnittstelle ins Internet beziehen. Diese Regeln lassen wir wie gehabt, fügen aber noch ein paar Regeln für die Kommunukation mit der DMZ hinzu. Hier wird das erste Mal die Symetrie zwischen dem internen Interface der Bastion und dem externen der Choke sichtbar.

8.3.1 Source–Quench Nachrichten

Die ICMP–Nachrichten vom Typ 4 (`source-quench`) werden erstellt, wenn ein Kommunikationspartner mit der Geschwindigkeit eines anderen nicht mehr mithalten kann, weil z.B. seine internen Buffer voll sind. Bastion und Choke akzeptieren alle `source-quench` Nachrichten, die bei der Kommunikation mit der DMZ entstehen.

Die Befehle für die Bastion

```
# source-quench zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $BASTION_DMZ_IPADDR 4 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 4 -d $BASTION_DMZ_IPADDR -j ACCEPT
```

Die Befehle für die Choke

```
# source-quench zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 4 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 4 -d $DMZ_ADDRESSES -j ACCEPT
```

Sichtbar wird hier, daß die Regeln für Bastion und Choke sehr ähnlich sind, aber eben nur die für die interne Bastion Schnittstelle und die der externen Choke Schnittstelle. Das ist logisch, denn diese beiden Schnittstellen transportieren ja die Informationspakete vom LAN ins Internet.

8.3.2 Parameter–Problem Nachrichten

Diese Nachrichtenpakete (Typ 12) werden gesendet, wenn ein Paket empfangen wurde, das ungültige Daten im Header aufweist. Auch diese Nachrichten lassen wir für die Kommunikation mit der DMZ grundsätzlich durch.

Die Befehle für die Bastion

```
# parameter-problem zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 12 -d $ANYWHERE -j ACCEPT
```

Die Befehle für die Choke

```
# parameter-problem zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 12 -d $ANYWHERE -j ACCEPT
```

8.3.3 Destination–Unreachable Nachrichten

Dieser Nachrichtenpakettyp (Typ 3) ist eine allgemeine Fehlermeldung. Auch diese Nachrichten lassen wir für die Kommunikation mit der DMZ grundsätzlich durch.

Die Befehle für die Bastion

```
# destination-unreachable zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 3 -d $ANYWHERE -j ACCEPT
```

Die Befehle für die Choke

```
# destination-unreachable zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 3 -d $ANYWHERE -j ACCEPT
```

8.3.4 Time–Exceeded Nachrichten

Diese Nachrichtenpakete (Typ 11) zeigen an, daß ein Paket zu oft geroutet wurde, daß seine Lebensdauer auf 0 gesunken ist. Es wird aber auch als Antwort für Traceroute benutzt. Auch diese Nachrichten lassen wir für die Kommunikation mit der DMZ grundsätzlich durch.

Die Befehle für die Bastion

```
# time-exceeded zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $BASTION_DMZ_IPADDR 11 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 11 -d $BASTION_DMZ_IPADDR -j ACCEPT
```

Die Befehle für die Choke

Die Choke darf nur mit der Bastion Nachrichten vom Typ `time-exceed` austauschen.

```
# time-exceeded zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $BASTION_DMZ_IPADDR 11 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 11 -d $BASTION_DMZ_IPADDR -j ACCEPT
```

8.3.5 Ping Konfiguration

Der Befehl `ping` verwendet zwei unterschiedliche Nachrichtentypen, jeweils für Anfragen (`echo-request` – Typ 8) und Antworten (`echo-reply` – Typ 0).

ping Konfiguration der Bastion

Alle Rechner der DMZ dürfen ins Internet pingen:

```
# Ausgehendes Ping aus der DMZ
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 8 -d $ANYWHERE -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 0 -d $DMZ_ADDRESSES -j ACCEPT
```

Andersherum darf aber nur die Bastion selbst Rechner in der DMZ anpingen:

```
# Ankommendes Ping in die DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $BASTION_DMZ_IPADDR 8 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 0 -d $BASTION_DMZ_IPADDR -j ACCEPT
```

ping Konfiguration der Choke

Die Choke darf jeden Rechner im Internet anpingen:

```
# Ausgehendes Ping ins Internet
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 8 -d $ANYWHERE -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 0 -d $CHOKE_DMZ_IPADDR -j ACCEPT
```

Ankommende Pings sind jedoch nur aus der DMZ gestattet.

```
# Ankommende Pings aus der DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 8 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 0 -d $DMZ_ADDRESSES -j ACCEPT
```

8.4 Dienste auf unprivilegierten Ports

Die Dienste auf den unprivilegierten Ports lassen wir auf der Bastion so, wie wir sie im Abschnitt 6.4 formuliert hatte. Die Choke muß diese Dienste überhaupt nicht beachten, denn es waren ja nur Verbote aus dem globalen Netz. Von außen droht uns also keine Gefahr mehr.

Wir brauchen diese Dienste nicht vom LAN in die DMZ weitergeben und da grundsätzlich alles verboten ist, was nicht explizit erlaubt, benötigen wir keine Regeln mehr. Innerhalb des LAN stehen sie uns natürlich weiter zur Verfügung. Dort werden Pakete ja nicht gefiltert, sondern nur weitergegeben.

8.5 Domain Name Service

Das DNS–System (TCP– und UDP–Port 53) ist heute für eine Zugriff auf das Internet von zwingender Bedeutung. Ohne Nameserver müssten die IP–Addressen der einzelnen Server im Internet bekannt sein, das ist mehr oder weniger unmöglich. Alle Rechner des LAN müssen also Zugriff auf einen Nameserver haben. Dabei gibt es aber verschiedene Möglichkeiten, die hier kurz beschrieben werden sollen.

8.5.1 Nutzung eines öffentlichen Nameservers

Die einfachste Form ist die, daß ein öffentlicher Nameserver im Internet, etwa der Ihres Providers benutzt wird. In diesem Fall muß der Choke nur mitgeteilt werden, daß DNS–Anfragen durchgelassen werden, so wie wir das auch schon im Beispiel in Abschnitt 6.4 getan haben. Die Bastion behält die Zeilen aus dem genannten Beispiel ebenfalls. Damit ist ein Durchgang geschaffen, der alle notwendigen DNS–Dienste ermöglicht. Die Choke erhält also die Zeilen:

```
# DNS IP Adresse:
NAMESERVER=xx.xx.xx.xx

# UDP-Nameserverzugriff
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p udp\
         -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp\
         -s $NAMESERVER 53 \
         -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT


# TCP-Nameserverzugriff
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp\
         -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
```

```
          -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y\
          -s $NAMESERVER 53 \
          -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

Wir werden ja später das Masquerading aktivieren, daher reicht es, wenn die Adresse immer nur die der Choke ist. Alle Adressen des LAN werden ja in diese Adresse maskiert.

8.5.2 Betreiben eigener Nameserver

Ein wesentlich komplexeres Verfahren besteht darin, eigene Nameserver zu betreiben. In unserem Beispiel würde sich dabei – wenn hohe Sicherheit erwünscht ist – folgendes Szenario anbieten:

Auf der Bastion läuft ein öffentlicher Nameserver. Er ist konfiguriert als autoritativer Server für die Site, verfügt aber nur über unvollständige Daten. Er hat keine Informationen über die Rechner des LAN, nur über die der DMZ. Die Programme der Bastion selbst greifen **nicht** auf diesen Server zu.

Auf der Choke läuft auch ein Nameserver. Dieser Nameserver enthält tatsächlich die realen Daten des LAN und der DMZ. Alle Clients aus dem LAN greifen ausschließlich auf diesen Nameserver zu. Auch die Clients der Bastion (die Programme der Bastion, die einen DNS–Dienst benötigen) greifen auf den Server der Choke zu. Wenn der Choke–Server eine Anfrage nicht beantworten kann, leitet er sie weiter an den Server der Bastion, der sie wiederum an einen externen Server im Internet weitergibt.

Für diese Konfiguration sind natürlich auch erheblich komplexere Regeln notwendig.

Konfiguration der Bastion als öffentlicher Nameserver

```
# Der DNS-Server der Bastion akzeptiert Anfragen der Choke (UDP 53)
ipchains -A input -i $BASTION_DMZ_INTERFACE -p udp\
          -s $CHOKE_DMZ_IPADDR 53\
          -d $BASTION_DMZ_IPADDR 53 -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p udp\
          -s $BASTION_DMZ_IPADDR 53\
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT

# DNS Anfragen der Bastion an den Server der Choke (UDP/TCP 53)
ipchains -A output -i $BASTION_DMZ_INTERFACE -p udp\
          -s $BASTION_DMZ_IPADDR $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p udp\
          -s $CHOKE_DMZ_IPADDR 53 \
          -d $BASTION_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp\
          -s $BASTION_DMZ_IPADDR $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp ! -y\
          -s $CHOKE_DMZ_IPADDR 53 \
          -d $BASTION_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT

# DNS IP Adresse (Nameserver im Internet):
NAMESERVER=xx.xx.xx.xx
```

```
#DNS Forwarding (Server zu Server)
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53 \
         -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $NAMESERVER 53 \
         -d $IPADDR 53 -j ACCEPT

# UDP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 53 -j ACCEPT
ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $ANYWHERE 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# TCP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 53 -j ACCEPT
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# DNS Zugriff fremder Clients
ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 53 -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53\
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 53 -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $IPADDR 53\
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT
```

Konfiguration der Choke als privater Nameserver

Der private Nameserver liegt auf der Choke. Clients aus dem LAN und der Bastion greifen auf diesen Server zu. Wenn er eine Anfrage nicht beantworten kann, dann leitet er sie an den Server der Bastion weiter.

Die Regeln für die Choke verhalten sich gewissermaßen spiegelbildlich zu denen der Bastion.

```
# Der DNS-Server der Choke gibt Anfragen an die Bastion (UDP 53)
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p udp\
         -s $CHOKE_DMZ_IPADDR 53\
         -d $BASTION_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp\
         -s $BASTION_DMZ_IPADDR 53\
         -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
```

```
# Der Nameserver der Choke lässt Anfragen aller Clients aus der
# DMZ zu (TCP/UDP 53)
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp \
         -s $DMZ_ADDRESSES $UNPRIVPORTS \
         -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp \
         -s CHOKE_DMZ_IPADDR 53 \
         -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp \
         -s $DMZ_ADDRESSES $UNPRIVPORTS \
         -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y \
         -s CHOKE_DMZ_IPADDR 53 \
         -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT
```

8.6 Der ident–Service (auth)

Dieser Service (TCP 113) wird von manchen Internet–Diensten benutzt, um Namen oder ID eines bestimmten Users zu erfragen. Das ist etwa bei fremden Mailservern der Fall, die Mails weitergeben. Es ist nicht zwingend nötig, selbst einen solchen Server laufen zu haben, obwohl nichts dagegen spricht. Auf jeden Fall sollten wir eine Firewall–Regel erstellen, um unnötige Timeouts zu vermeiden.

Für unser Beispiel werden wir davon ausgehen, daß sowohl auf der Bastion, als auch auf der Choke sowohl Clients, als auch Server mit dem `auth`–Protokoll arbeiten.

8.6.1 Konfiguration der Bastion

```
  # abgehende auth-Anfragen ins Internet
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE 113 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 113 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# ankommende auth-Anfragen aus dem Internet
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 113 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $IPADDR 113 \
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT

# Bastion als auth-Client nach innen
ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp \
         -s $BASTION_DMZ_IPADDR $UNPRIVPORTS \
         -d $DMZ_ADDRESSES 113 -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp ! -y\
         -s $DMZ_ADDRESSES 113 \
         -d $BASTION_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

```
# Bastion als Server nach innen
ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp \
         -s $DMZ_ADDRESSES $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 113 -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp ! -y\
         -s $BASTION_DMZ_IPADDR 113 \
         -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT
```

8.6.2 Konfiguration der Choke

```
# Choke als auth Server
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp \
         -s $DMZ_ADDRESSES $UNPRIVPORTS \
         -d $CHOKE_DMZ_IPADDR 113 -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp ! -y\
         -s $CHOKE_DMZ_IPADDR 113 \
         -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT

# Choke als auth-Client
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp\
         -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 113 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y \
         -s $ANYWHERE 113 \
         -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

8.7 E–Mail

Es gibt natürlich – auch nur für unser Modell – eine Vielzahl von denkbaren Kombinationen für E–Mail Empfang und Versand. Wir werden hier exemplarisch eine Kombination durchspielen, die wahrscheinlich in der Praxis die häufigste ist. Die Bastion, die ja eine echte IP–Adresse hat, sendet und empfängt Mail mit SMTP (TCP 25). Die Rechner des LAN holen sich die Mails bei der Bastion mittels POP3 (TCP 110) ab.

Wir müssen also der Bastion erlauben SMTP–Pakete zu empfangen und zu senden, und weiterhin POP-Anfragen zu empfangen und zu beantworten. Die Choke muß auch SMTP durchlassen, schließlich senden die Rechner des LAN ihre Mails ja mit SMTP. Sie muß ebenfalls POP–Pakete durchlassen, in beide Richtungen.

8.7.1 Abgehende Mails über die Bastion verschicken

Hier haben wir zunächst einmal zwei Möglichkeiten. Entweder wir schicken alle ausgehende Mail an den Mailserver unseres Providers oder wir schicken sie gleich an die entsprechenden Empfängerrechner. Die einfachere Möglichkeit ist die des Providerservers, weil der sich dann um die korrekte Weiterleitung kümmern muß. Die dazu nötigen Regeln der Bastion sind:

```
# Mail über SMTP an Provider Gateway
# Zuerst der Name des Gateways
SMTP_GATEWAY=smtp.server.provider.beispiel
```

```
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $SMTP_GATEWAY 25 -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $SMTP_GATEWAY 25 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

Wenn aber stattdessen erwünscht ist, daß die Bastion selbst Mailserver ist, also die Mail direkt zu verschicken, dann muß einfach statt der Angabe des SMTP–Gateways ein `$ANYWHERE` verwendet werden:

```
# Mail über SMTP an Provider Gateway
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 25 -j ACCEPT
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 25 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

Damit die Bastion aber auch ankommende Aufträge aus der DMZ akzeptieren kann, bedarf es noch eines weiteren Regelsatzes für das Bastion–Script:

```
# Bastion nimmt SMTP Anfragen aus der DMZ entgegen
ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp \
         -s $DMZ_ADDRESSES $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 25 -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp ! -y \
         -s $BASTION_DMZ_IPADDR 25 \
         -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT
```

Jetzt müssen wir aber noch der Choke erlauben, SMTP-Pakete an die Bastion durchzulassen. In das Choke Script muß folglich eingetragen werden:

```
# Choke erlaubt SMTP-Pakete an die Bastion
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp\
         -s CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 25 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y\
         -s $BASTION_DMZ_IPADDR 25 \
         -d CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

Damit sollte der Versand von E–Mail auch für die Rechner im LAN möglich sein.

8.7.2 Mailempfang über POP–Server auf der Bastion

Die Rechner des LAN holen ihre Mails direkt bei der Bastion ab. Das geschieht mit dem POP3 Protokoll (TCP 110). Die Bastion muß nur Pakete aus der DMZ durchlassen, aus dem Internet ist dieser Service nicht verfügbar. Das Bastion–Script bekommt also noch die Regeln:

```
# POP-Server der Bastion nimmt Pakete aus der DMZ entgegen
```

```
ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp \
         -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 110 -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp ! -y \
         -s $BASTION_DMZ_IPADDR 110 \
         -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

Natürlich muß jetzt auch wieder die Choke noch soweit konfiguriert werden, daß sie Pakete durchlässt, die an die Bastion gerichtet sind. Das Choke–Script wird um die folgenden Regeln erweitert:

```
# POP3 Pakete an die Bastion werden weitergegeben:
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp \
         -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 110 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y \
         -s $BASTION_DMZ_IPADDR 110 \
         -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

8.8 Alle weiteren Dienste

Damit wir hier jetzt nicht jeden denkbaren Internet–Dienst in epischer Breite darstellen müssen, hier noch ein paar allgemeingültige Tips zum Aufbau der einzelnen Regeln.

Der grundsätzliche Aufbau der Regeln ist immer gleich, die Bastion erlaubt bestimmte Pakete, die Choke erlaubt die selben Pakete. Aber während die Bastion die Pakete auf ihre interne Schnittstelle erlaubt, tut die Choke es auf die externe. Man könnte fast von einer Art Achsensymetrie sprechen, die Achse wäre in dem Fall die DMZ.

Um die richtigen Regeln für einen bestimmten Dienst zu erstellen, ist es notwendig, ein Protokoll hinlänglich zu verstehen. Dazu können die im Anhang aufgeführten Protokollbeschreibungen dienen. Diese Beschreibungen sind selbstverständlich nicht vollständig, das würde den Rahmen der Darstellung sprengen. Genauere Details können in den jeweiligen HOWTOS, den RFCs und anderen Informationsquellen entnommen werden. Trotzdem sollte nach der Durcharbeitung der letzten Abschnitte genügend Verständnis für eigene Weiterentwicklungen vorhanden sein.

8.9 LAN Zugriff auf die Choke

Bei allen Regeln, die wir in den letzten Abschnitten aufgestellt haben, haben wir eine Kleinigkeit einfach weggelassen. Wir haben zwar der Choke jeweils Regeln gegeben, die bestimmten Paketen erlauben, an die Bastion weitergeleitet zu werden, wir haben aber der Choke niemals eine Regel gegeben, die überhaupt Pakete aus dem LAN entgegennehmen würde.

In der Regel stellt es kein Sicherheitsproblem dar, wenn die Choke alle Pakete aus dem zu schützenden Netz (LAN) akzeptiert, damit sparen wir uns eine Menge zusätzlicher Regeln, wenn wir das pauschal für die Choke formulieren:

```
# Vollständige Öffnung der Kommunikation zwischen LAN und Choke
ipchains -A input -i $CHOKE_LAN_INTERFACE \
         -s $LAN_ADDRESSES -j ACCEPT
ipchains -A output  -i $CHOKE_LAN_INTERFACE \
         -d $LAN_ADDRESSES -j ACCEPT
```

8.10 Und schließlich das Masquerading

Bis zu diesem Punkt der Darstellung haben weder die Bastion, noch die Choke tatsächlich Pakete maskiert, aus diesem Grund hätte auch ein Datenverkehr mit dem Internet bis jetzt noch nicht funktionieren können. In unserer DMZ und im LAN arbeiten wir ja mit reservierten Adressen. (Auch wenn es in der Praxus wenig Sinn macht, die DMZ wäre ja so von außen nicht ansprechbar)

Der erste Schritt ist jetzt also der, daß wir die Bastion zum Masquerading–Server erklären, der alle Pakete, die von der Choke kommen, auf seine echte Internet–Adresse maskiert. Das gesammte Netz wird so zum Internet komplett versteckt.

```
# Bastion maskiert alle Pakete der Choke
ipchains -A forward -i $EXTERNAL_INTERFACE \
         -s $CHOKE_DMZ_IPADDRESS -j MASQ
```

Damit jetzt aber auch alle Pakete aus dem LAN unter der Adresse der Choke an die Bastion gehen, müssen wir auch die Choke als Masquerading Server konfigurieren, die jetzt alle Pakete aus dem LAN mit ihrer DMZ–Adresse maskiert. Das bietet auch eine doppelte Sicherheit, denn innerhalb der DMZ ist damit das LAN unsichtbar, es existiert nur noch die Choke:

```
# Choke maskiert alle Pakete des LAN
ipchains -A forward -i $CHOKE_DMZ_INTERFACE \
         -s $LAN_ADDRESSES -j MASQ
```

Anhang A

Wichtige Portnummern

A.1 Häufig gescannte Ports

Port	Dienst	Protokoll	Gefahr	Bemerkung
0	reserviert	TCP/UDP	hoch	Sowohl als Absender–, als auch als Empfängerport ungültig
0-5		TCP	hoch	Signatur von `sscan`
7	`echo`	TCP/UDP	hoch	UDP–Angriffsmöglichkeit
11	`systat`	TCP	hoch	Informationen über laufende Prozesse
15	`netstat`	TCP	hoch	Abfrage des Netzwerkstatus, offene Verbindungen, Routingtabellen,...
19	`chargen`	TCP/UDP	hoch	UDP–Angriffsmöglichkeit
20,21	`ftp`	TCP	mittel	FTP-Zugriff, Dateitransfer
22	`ssh`	TCP	mittel	Secure Shell – Eine sichere (verschlüsselte) Möglichkeit des remote login
22		UDP	niedrig	Eine alte Version von PC–Anywhere
23	`telnet`	TCP	hoch	TELNET – Unverschlüsselte Möglichkeit des remote login
25	`smtp`	TCP	hoch	Simple Mail Transfer Protocol – jemand sucht nach Spam–Relay oder einer Sicherheitslücke älterer `sendmail` Versionen
53	`domain`	TCP	hoch	TCP–Zone–Transfer oder Fälschung von DNS–Informationen
67	`bootps`	UDP	niedrig	
69	`tftpd`	UDP	mittel	Unsichere (UDP) Alternative zu FTP
79	`finger`	TCP	niedrig	Informationen über die User eines Systems
87	`link`	TCP	hoch	`tty-link`, gerne von Angreifern benutzt
109,110	`pop2/3`	TCP	hoch	Post Office Protocol - Einer der drei häufigst angegriffenen Ports
111	`sunrpc`	TCP/UDP	hoch	Remote Procedure Call - Der häufigste Angriffsport überhaupt
119	`nntp`	TCP	mittel	Öffentlicher Newsfeed (wird zum Versenden von Spam benutzt)

Tabelle A.1: Häufig gescannte Portnummern und ihre Bedeutung

Port	Dienst	Protokoll	Gefahr	Bemerkung
123	ntp	UDP	niedrig	Zeitsynchronisation übers Netz - ungefährlich aber vielleicht lästig
137	netbios-ns	TCP/UDP	mittel	Nameservice für Windows Netze – für Unix harmlos
138	netbios-dgm	TCP/UDP	mittel	Für Windows Netze – für Unix harmlos
139	netbios-ssm	TCP/UDP	mittel	Für Windows Netze – für Unix harmlos
143	imap	TCP	hoch	Internet Mail Protocol – Der Nachfolger von POP3. Einer der häufigsten Angriffsziele
161,162	snmp	UDP	mittel	Simple Network Management Protocol – Netzwerkadministration übers Netz
177	xdmcp	UDP	hoch	Login Manager des X-Window-Systems
512	exec	TCP	hoch	Remote process execution
512	biff	UDP	hoch	Mail–Benachrichtigung
513	login	TCP	hoch	Remote Login
513	who	UDP	hoch	Wer ist eingeloggt
514	shell	TCP	hoch	Remote Shell
514	syslog	UDP	hoch	Syslog Eingang
515	printer	TCP	hoch	Netzwerk–Drucker
517	talk	UDP	mittel	Netzwerk–Telephon
518	ntalk	UDP	mittel	Netzwerk–Telephon
520	route	UDP	hoch	Routing Tabellen
540	uucp	TCP	mittel	UUCP Dienste
635	mount	UDP	hoch	Sicherheitslücke im Mountdaemon
1114	tt SQL	TCP	hoch	Signatur von sscan
2000	openwin	TCP	hoch	Open Windows
2049	nfs	TCP/UDP	hoch	Network File System – Fernzugriff aufs Dateisystem
5632	pcanywhere	UDP	niedrig	PC Anywhere
6000+*n*	X11	TCP	hoch	X–Window–System
12345	netbus	TCP	hoch	NetBus, ein Trojaner, der auch auf anderen Ports erscheinen kann (häufig sind etwa 12346 oder 20034) – für Unix harmlos
31337	backorifice	UDP	hoch	Noch ein Trojaner – für Unix harmlos

Tabelle A.2: Häufig gescannte Portnummern und ihre Bedeutung (Fortsetzung)

A.2 ICMP Nachrichten Typen

Name:	Nummer:	Bemerkung:
`pong`	0	Antwort auf Ping - auch `echo-reply` genannt
`destination-unreachable`	3	Fehlermeldung: Ein Router konnte das Paket nicht weitergeben und sendet diese Meldung zurück.
`source-quench`	4	Flusskontrolle zwischen zwei Rechnern
`redirect`	5	Routeninformation. Ein Rechner kann mit dieser Nachricht einem anderen Rechner (auf dem entweder der `routed` oder der `gated` laufen muß) mitteilen, daß es eine bessere Route gibt.
`ping`	8	Echoanforderung. Ein Rechner, der dieses Paket erhält, schickt eine Kopie des Paketes zurück. Auch `echo-request` genannt
`time-exceeded`	11	Fehlermeldung: Paket wurde zu oft geroutet. Das Lebensdauerfeld ist auf 0
`parameter-problem`	12	Fehlermeldung: Der Header des IP–Paketes enthält ungültige Werte

Tabelle A.3: ICMP–Nachrichtentypen

Anhang B

Die Beispiel–Scripts

B.1 Das erste Firewallscript

```
# Konstantendefinition
EXTERNAL_INTERFACE=eth0        # Das fremde Netz
LOOPBACK_INTERFACE=lo          # Local Loopback
IPADDR=192.168.100.1           # Eigene Adresse
ANYWHERE=any/0                 # Jede IP-Adresse
MY_ISP=123.45.67.89/16         # Der Bereich meines Providers
LOOPBACK=127.0.0.0/8           # Loopback-Adressbereich
CLASS_A=10.0.0.0/8             # Reservierter Bereich Klasse A
CLASS_B=172.16.0.0/12          # Reservierter Bereich Klasse B
CLASS_C=192.168.0.0/16         # Reservierter Bereich Klasse C
CLASS_D=224.0.0.0/4            # Komplette Klasse D
CLASS_E=240.0.0.0/5            # Komplette Klasse E
BROADCAST_SRC=0.0.0.0          # Broadcast Absender
BROADCAST_DEST=255.255.255.255 # Broadcast Empfänger
PRIVPORTS=0:1023               # Privilegierte Portnummern
UNPRIVPORTS=1024:65535         # Unprivilegierte Portnummern

# Alle bestehenden Regeln löschen
ipchains -F

# Voreingestellte Policies setzen
ipchains -P input    DENY
ipchains -P output   REJECT
ipchains -P forward  REJECT

# Loopback ohne Einschränkungen
ipchains -A input  -i $LOOPBACK_INTERFACE -j ACCEPT
ipchains -A output -i $LOOPBACK_INTERFACE -j ACCEPT

# SYN_COOKIES aktivieren
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# SOURCE ADDRESS VERIFICATION aktivieren
for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
  echo 1 > $i
done

# Pakete ablehnen, die vorgeben von der eigenen Adresse zu stammen
```

```
ipchains -A input -i $EXTERNAL_INTERFACE -s $IPADDR -j DENY -l

# Reservierte A-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_A -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_A -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_A -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_A -j DENY

# Reservierte B-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_B -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_B -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_B -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_B -j DENY

# Reservierte C-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_C -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_C -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_C -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_C -j DENY

# Pakete mit Loopback als Absender verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $LOOPBACK -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $LOOPBACK -j DENY

# Pakete mit illegalen Broadcast Adressen verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $BROADCAST_DEST -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $BROADCAST_SRC -j DENY

# Pakete mit Klasse-D Adresse als Absender verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_D -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_D -j

# Klasse-E Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_E -j DENY

# Von der IANA reservierte Adressen verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s 1.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 2.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 5.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 7.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 23.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 27.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 31.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 37.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 39.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 41.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 42.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 58.0.0.0/7 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 60.0.0.0/8 -j DENY
# 65 entspricht 01000001 - also würde die Maske /3 leider die 64
# mit ansprechen. Wir müssen daher 65-79 einzeln angeben
ipchains -A input -i $EXTERNAL_INTERFACE -s 65.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 66.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 67.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 68.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 69.0.0.0/8 -j DENY
```

```
ipchains -A input -i $EXTERNAL_INTERFACE -s 70.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 71.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 72.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 73.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 74.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 75.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 76.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 77.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 78.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 79.0.0.0/8 -j DENY
# 80-95 lässt sich mit der Maske /4 ansprechen
ipchains -A input -i $EXTERNAL_INTERFACE -s 80.0.0.0/4 -j DENY
# 96-111 lässt sich mit der Maske /4 ansprechen
ipchains -A input -i $EXTERNAL_INTERFACE -s 96.0.0.0/4 -j DENY
# 126 entspricht 01111110 - die Maske /3 würde 127 beinhalten
# daher müssen wir 112 - 126 einzeln angeben
ipchains -A input -i $EXTERNAL_INTERFACE -s 112.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 113.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 114.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 115.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 116.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 117.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 118.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 119.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 120.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 121.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 122.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 123.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 124.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 125.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 126.0.0.0/8 -j DENY
# 217-219 einzeln
ipchains -A input -i $EXTERNAL_INTERFACE -s 217.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 218.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 219.0.0.0/8 -j DENY
# 220-223 -> /6
ipchains -A input -i $EXTERNAL_INTERFACE -s 220.0.0.0/6 -j DENY

# ICMP Typ 4 erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 4 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 4 -d $ANYWHERE -j ACCEPT

# ICMP Typ 12 erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 12 -d $ANYWHERE -j ACCEPT

# ICMP-Typ 3 für Providerrechner erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $IPADDR -j ACCEPT
```

```
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 3 -d $MY_ISP -j ACCEPT

# ICMP-Typ 3 Subtyp fragmentation-needed für alle freigeben
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR fragmentation-needed -d $ANYWHERE -j ACCEPT

# ICMP-Typ 11 erlauben (ausgehend nur an unseren Provider)
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 11 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 11 -d $MY_ISP -j ACCEPT

# Ausgehendes Ping erlauben
ipchain -A output -i $EXTERNAL_INTERFACE -p icmp\
        -s $IPADDR 8 -d $ANYWHERE -j ACCEPT

ipchain -A input -i $EXTERNAL_INTERFACE -p icmp\
        -s $ANYWHERE 0 -d $IPADDR -j ACCEPT

# Ankommendes Ping nur für Provider erlauben
ipchain -A input -i $EXTERNAL_INTERFACE -p icmp\
        -s $MYISP 8 -d $IPADDR -j ACCEPT

ipchain -A output -i $EXTERNAL_INTERFACE -p icmp\
        -s $IPADDR 0 -d $MYISP -j ACCEPT

# Open Window Verbindungsaufbau sperren
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
         -s $IPADDR -d $ANYWHERE 2000 -j REJECT

# X11 Aufbau zu einem fremden Server verbieten
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
         -s $IPADDR -d $ANYWHERE 6000:6063 -j REJECT

# X11 Aufbau von außen zu einem unserer Server verbieten
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp -y\
         -d $IPADDR 6000:6063 -j DENY -l

# NFS (2049) über UDP sperren
ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -d $IPADDR 2049 -j DENY -l

# NFS (2049) über TCP sperren
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp -y\
         -d $IPADDR 2049 -j DENY -l

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
         -d $ANYWHERE 2049 -j DENY -l

# DNS IP Adresse:
NAMESERVER=xx.xx.xx.xx

# UDP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
```

```
          -s $IPADDR $UNPRIVPORTS \
          -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
          -s $NAMESERVER 53 \
          -d $IPADDR $UNPRIVPORTS -j ACCEPT


# TCP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
          -s $IPADDR $UNPRIVPORTS \
          -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
          -s $NAMESERVER 53 \
          -d $IPADDR $UNPRIVPORTS -j ACCEPT

#DNS Forwarding (Server zu Server)
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
          -s $IPADDR 53 \
          -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
          -s $NAMESERVER 53 \
          -d $IPADDR 53 -j ACCEPT

# DNS Zugriff fremder Clients
MYDNSCLIENTS=ww.xx.yy.zz/mm

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
          -s $MYDNSCLIENTS $UNPRIVPORTS\
          -d $IPADDR 53 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
          -s $IPADDR 53\
          -d $MYDNSCLIENTS $UNPRIVPORTS -j ACCEPT

# DNS Forwarding für fremde Clients
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
          -s $IPADDR 53 \
          -d $MYDNSCLIENTS 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
          -s $MYDNSCLIENTS 53 \
          -d $IPADDR 53 -j ACCEPT

# DNS Zone-Transfer fremder Clients
MYDNSZONECLIENTS=ww.xx.yy.zz/mm

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
          -s $MYDNSZONECLIENTS $UNPRIVPORTS\
          -d $IPADDR 53 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
          -s $IPADDR 53\
          -d $MYDNSZONECLIENTS $UNPRIVPORTS -j ACCEPT
```

```
# abgehende auth-Anfragen
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE 113 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 113 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

# ankommende auth-Anfragen
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 113 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $IPADDR 113 \
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT

# oder statt der letzten zwei Befehle
# ankommende auth-Anfragen ablehnen
# ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
#          -s $ANYWHERE\
#          -d $IPADDR 113 -j REJECT
```

B.2 Das Bastion Firewallscript

```
# Konstantendefinition
EXTERNAL_INTERFACE=eth0        # Das Interface ins Internet
IPADDR=123.45.67.89            # Adresse des Internetzugangs
MY_ISP=123.45.67.90/16         # Der Bereich meines Providers

BASTION_DMZ_INTERFACE=eth1     # internes Interface
BASTION_DMZ_IPADDR=192.168.1.1 # Adresse dazu

LOOPBACK_INTERFACE=lo          # Local Loopback
LOOPBACK=127.0.0.0/8           # Loopback-Adressbereich

CHOKE_DMZ_IPADDR=192.168.1.2   # ext. Interface der Choke
DMZ_ADDRESSES=192.168.1.0/24   # IP-Bereich der DMZ
DMZ_BROADCAST=192.168.1.255    # Broadcastadresse DMZ
ANYWHERE=any/0                 # Jede IP-Adresse

CLASS_A=10.0.0.0/8             # Reservierter Bereich Klasse A
CLASS_B=172.16.0.0/12          # Reservierter Bereich Klasse B
CLASS_C=192.168.0.0/16         # Reservierter Bereich Klasse C
CLASS_D=224.0.0.0/4            # Komplette Klasse D
CLASS_E=240.0.0.0/5            # Komplette Klasse E
BROADCAST_SRC=0.0.0.0          # Broadcast Absender
BROADCAST_DEST=255.255.255.255 # Broadcast Empfänger
PRIVPORTS=0:1023               # Privilegierte Portnummern
UNPRIVPORTS=1024:65535         # Unprivilegierte Portnummern

# Alle bestehenden Regeln löschen
ipchains -F

# Voreingestellte Policies setzen
ipchains -P input    DENY
ipchains -P output   REJECT
ipchains -P forward  REJECT

# Loopback ohne Einschränkungen
ipchains -A input  -i $LOOPBACK_INTERFACE -j ACCEPT
ipchains -A output -i $LOOPBACK_INTERFACE -j ACCEPT

# SYN_COOKIES aktivieren
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# SOURCE ADDRESS VERIFICATION aktivieren
for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
  echo 1 > $i
done

# Pakete ablehnen, die vorgeben von der eigenen Adresse zu stammen
ipchains -A input -i $EXTERNAL_INTERFACE \
         -s $IPADDR -j DENY -l
ipchains -A input -i $BASTION_DMZ_INTERFACE \
         -s $BASTION_DMZ_IPADDR -j DENY -l
```

```
# Reservierte A-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_A -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_A -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_A -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_A -j DENY

# Reservierte B-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_B -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_B -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_B -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_B -j DENY

# Reservierte C-Klasse Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_C -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $CLASS_C -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_C -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -d $CLASS_C -j DENY

# Pakete mit Loopback als Absender verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $LOOPBACK -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $LOOPBACK -j DENY

# Pakete mit illegalen Broadcast Adressen verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $BROADCAST_DEST -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -d $BROADCAST_SRC -j DENY

# Pakete mit Klasse-D Adresse als Absender verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_D -j DENY
ipchains -A output -i $EXTERNAL_INTERFACE -s $CLASS_D -j

# Klasse-E Adressen ablehnen
ipchains -A input -i $EXTERNAL_INTERFACE -s $CLASS_E -j DENY

# Von der IANA reservierte Adressen verwerfen
ipchains -A input -i $EXTERNAL_INTERFACE -s 1.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 2.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 5.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 7.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 23.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 27.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 31.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 37.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 39.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 41.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 42.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 58.0.0.0/7 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 60.0.0.0/8 -j DENY
# 65 entspricht 01000001 - also würde die Maske /3 leider die 64
# mit ansprechen. Wir müssen daher 65-79 einzeln angeben
ipchains -A input -i $EXTERNAL_INTERFACE -s 65.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 66.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 67.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 68.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 69.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 70.0.0.0/8 -j DENY
```

```
ipchains -A input -i $EXTERNAL_INTERFACE -s 71.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 72.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 73.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 74.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 75.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 76.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 77.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 78.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 79.0.0.0/8 -j DENY
# 80-95 lässt sich mit der Maske /4 ansprechen
ipchains -A input -i $EXTERNAL_INTERFACE -s 80.0.0.0/4 -j DENY
# 96-111 lässt sich mit der Maske /4 ansprechen
ipchains -A input -i $EXTERNAL_INTERFACE -s 96.0.0.0/4 -j DENY
# 126 entspricht 01111110 - die Maske /3 würde 127 beinhalten
# daher müssen wir 112 - 126 einzeln angeben
ipchains -A input -i $EXTERNAL_INTERFACE -s 112.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 113.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 114.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 115.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 116.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 117.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 118.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 119.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 120.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 121.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 122.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 123.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 124.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 125.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 126.0.0.0/8 -j DENY
# 217-219 einzeln
ipchains -A input -i $EXTERNAL_INTERFACE -s 217.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 218.0.0.0/8 -j DENY
ipchains -A input -i $EXTERNAL_INTERFACE -s 219.0.0.0/8 -j DENY
# 220-223 -> /6
ipchains -A input -i $EXTERNAL_INTERFACE -s 220.0.0.0/6 -j DENY

# ICMP Regeln nach außen
# ICMP Typ 4 erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 4 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 4 -d $ANYWHERE -j ACCEPT

# ICMP Typ 12 erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 12 -d $ANYWHERE -j ACCEPT

# ICMP-Typ 3 für Providerrechner erlauben
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $IPADDR -j ACCEPT
```

```
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 3 -d $MY_ISP -j ACCEPT

# ICMP-Typ 3 Subtyp fragmentation-needed für alle freigeben
ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR fragmentation-needed -d $ANYWHERE -j ACCEPT

# ICMP-Typ 11 erlauben (ausgehend nur an unseren Provider)
ipchains -A input -i $EXTERNAL_INTERFACE -p icmp\
         -s $ANYWHERE 11 -d $IPADDR -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p icmp\
         -s $IPADDR 11 -d $MY_ISP -j ACCEPT

# Ausgehendes Ping erlauben
ipchain -A output -i $EXTERNAL_INTERFACE -p icmp\
        -s $IPADDR 8 -d $ANYWHERE -j ACCEPT

ipchain -A input -i $EXTERNAL_INTERFACE -p icmp\
        -s $ANYWHERE 0 -d $IPADDR -j ACCEPT

# Ankommendes Ping nur für Provider erlauben
ipchain -A input -i $EXTERNAL_INTERFACE -p icmp\
        -s $MYISP 8 -d $IPADDR -j ACCEPT

ipchain -A output -i $EXTERNAL_INTERFACE -p icmp\
        -s $IPADDR 0 -d $MYISP -j ACCEPT

# source-quench zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $BASTION_DMZ_IPADDR 4 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 4 -d $BASTION_DMZ_IPADDR -j ACCEPT

# parameter-problem zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 12 -d $ANYWHERE -j ACCEPT

# destination-unreachable zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 3 -d $ANYWHERE -j ACCEPT

# time-exceeded zur DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $BASTION_DMZ_IPADDR 11 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 11 -d $BASTION_DMZ_IPADDR -j ACCEPT

# Ausgehendes Ping aus der DMZ
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 8 -d $ANYWHERE -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
```

```
          -s $ANYWHERE 0 -d $DMZ_ADDRESSES -j ACCEPT

# Ankommendes Ping in die DMZ
ipchains -A output -i $BASTION_DMZ_INTERFACE -p icmp\
          -s $BASTION_DMZ_IPADDR 8 -d $DMZ_ADDRESSES -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p icmp\
          -s $DMZ_ADDRESSES 0 -d $BASTION_DMZ_IPADDR -j ACCEPT

# Open Window Verbindungsaufbau sperren
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
          -s $IPADDR -d $ANYWHERE 2000 -j REJECT

# X11 Aufbau zu einem fremden Server verbieten
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
          -s $IPADDR -d $ANYWHERE 6000:6063 -j REJECT

# X11 Aufbau von außen zu einem unserer Server verbieten
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp -y\
          -d $IPADDR 6000:6063 -j DENY -l

# NFS (2049) über UDP sperren
ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
          -d $IPADDR 2049 -j DENY -l

# NFS (2049) über TCP sperren
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp -y\
          -d $IPADDR 2049 -j DENY -l

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp -y\
          -d $ANYWHERE 2049 -j DENY -l


# Der DNS-Server der Bastion akzeptiert Anfragen der Choke (UDP 53)
ipchains -A input -i $BASTION_DMZ_INTERFACE -p udp\
          -s $CHOKE_DMZ_IPADDR 53\
          -d $BASTION_DMZ_IPADDR 53 -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p udp\
          -s $BASTION_DMZ_IPADDR 53\
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT

# DNS Anfragen der Bastion an den Server der Choke (UDP/TCP 53)
ipchains -A output -i $BASTION_DMZ_INTERFACE -p udp\
          -s $BASTION_DMZ_IPADDR $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p udp\
          -s $CHOKE_DMZ_IPADDR 53 \
          -d $BASTION_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp\
          -s $BASTION_DMZ_IPADDR $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp ! -y\
          -s $CHOKE_DMZ_IPADDR 53 \
          -d $BASTION_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT


# DNS IP Adresse (Nameserver im Internet):
```

```
NAMESERVER=xx.xx.xx.xx

#DNS Forwarding (Server zu Server)
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53 \
         -d $NAMESERVER 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $NAMESERVER 53 \
         -d $IPADDR 53 -j ACCEPT


# UDP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $ANYWHERE 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT


# TCP-Nameserverzugriff
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 53 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 53 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT


# DNS Zugriff fremder Clients

ipchains -A input -i $EXTERNAL_INTERFACE -p udp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 53 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p udp\
         -s $IPADDR 53\
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT
ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
         -s $ANYWHERE $UNPRIVPORTS\
         -d $IPADDR 53 -j ACCEPT

ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $IPADDR 53\
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT

# abgehende auth-Anfragen ins Internet
ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE 113 -j ACCEPT

ipchains -A input -i $EXTERNAL_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 113 \
```

```
          -d $IPADDR $UNPRIVPORTS -j ACCEPT

  # ankommende auth-Anfragen aus dem Internet
  ipchains -A input -i $EXTERNAL_INTERFACE -p tcp\
          -s $ANYWHERE $UNPRIVPORTS\
          -d $IPADDR 113 -j ACCEPT

  ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
          -s $IPADDR 113 \
          -d $ANYWHERE $UNPRIVPORTS -j ACCEPT

  # Bastion als auth-Client nach innen
  ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp \
          -s $BASTION_DMZ_IPADDR $UNPRIVPORTS \
          -d $DMZ_ADDRESSES 113 -j ACCEPT
  ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp ! -y\
          -s $DMZ_ADDRESSES 113 \
          -d $BASTION_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT

  # Bastion als Server nach innen
  ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp \
          -s $DMZ_ADDRESSES $UNPRIVPORTS \
          -d $BASTION_DMZ_IPADDR 113 -j ACCEPT
  ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp ! -y\
          -s $BASTION_DMZ_IPADDR 113 \
          -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT

  # Mail über SMTP

  # Zuerst der Name des Gateways
  # statt einem Provider kann hier auch "any/0" stehen, um die Bastion
  # selbst zum Mailserver zu ernennen.
  SMTP_GATEWAY=smtp.server.provider.beispiel

  ipchains -A output -i $EXTERNAL_INTERFACE -p tcp\
          -s $IPADDR $UNPRIVPORTS \
          -d $SMTP_GATEWAY 25 -j ACCEPT
  ipchains -A output -i $EXTERNAL_INTERFACE -p tcp ! -y\
          -s $SMTP_GATEWAY 25 \
          -d $IPADDR $UNPRIVPORTS -j ACCEPT

  # Bastion nimmt SMTP Anfragen aus der DMZ entgegen
  ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp \
          -s $DMZ_ADDRESSES $UNPRIVPORTS \
          -d $BASTION_DMZ_IPADDR 25 -j ACCEPT
  ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp ! -y \
          -s $BASTION_DMZ_IPADDR 25 \
          -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT

  # POP-Server der Bastion nimmt Pakete aus der DMZ entgegen
  ipchains -A input -i $BASTION_DMZ_INTERFACE -p tcp \
          -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
          -d $BASTION_DMZ_IPADDR 110 -j ACCEPT
  ipchains -A output -i $BASTION_DMZ_INTERFACE -p tcp ! -y \
          -s $BASTION_DMZ_IPADDR 110 \
          -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT
```

```
# Bastion maskiert alle Pakete der Choke
ipchains -A forward -i $EXTERNAL_INTERFACE \
         -s $CHOKE_DMZ_IPADDRESS -j MASQ
```

B.3 Das Choke Firewallscript

```
#Konstantendefinition
CHOKE_DMZ_INTERFACE=eth0       # externes Interface der Choke
CHOKE_DMZ_IPADDR=192.168.1.2   # externe Adresse der Choke
CHOKE_LAN_INTERFACE=eth1       # internes Interface der Choke
CHOKE_LAN_IPADDR=192.168.5.1   # interne Adresse der Choke
LOOPBACK_INTERFACE=lo          # Local Loopback
LOOPBACK=127.0.0.0/8           # Loopback-Adressbereich

BASTION_DMZ_IPADDR=192.168.1.1 # interne Adresse der Bastion
DMZ_ADDRESSES=192.168.1.0/24   # IP-Bereich der DMZ
LAN_ADDRESSES=192.168.5.0/24   # IP-Bereich des LAN
DMZ_BROADCAST=192.168.1.255    # Broadcastadresse DMZ
ANYWHERE=any/0                 # Jede IP-Adresse

CLASS_A=10.0.0.0/8             # Reservierter Bereich Klasse A
CLASS_B=172.16.0.0/12          # Reservierter Bereich Klasse B
CLASS_C=192.168.0.0/16         # Reservierter Bereich Klasse C
CLASS_D=224.0.0.0/4            # Komplette Klasse D
CLASS_E=240.0.0.0/5            # Komplette Klasse E
BROADCAST_SRC=0.0.0.0          # Broadcast Absender
BROADCAST_DEST=255.255.255.255 # Broadcast Empfänger
PRIVPORTS=0:1023               # Privilegierte Portnummern
UNPRIVPORTS=1024:65535         # Unprivilegierte Portnummern

# Alle bestehenden Regeln löschen
ipchains -F

# Voreingestellte Policies setzen
ipchains -P input    REJECT
ipchains -P output   REJECT
ipchains -P forward  REJECT

# Loopback ohne Einschränkungen
ipchains -A input  -i $LOOPBACK_INTERFACE -j ACCEPT
ipchains -A output -i $LOOPBACK_INTERFACE -j ACCEPT

# SYN_COOKIES aktivieren
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# SOURCE ADDRESS VERIFICATION aktivieren
for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
  echo 1 > $i
done

# Pakete ablehnen, die vorgeben von der eigenen Adresse zu stammen
ipchains -A input -i $CHOKE_LAN_INTERFACE\
         -s $CHOKE_LAN_IPADDR -j REJECT -l
ipchains -A input -i CHOKE_DMZ_INTERFACE \
         -s $CHOKE_DMZ_IPADDR -j REJECT -l

# Pakete mit privaten A-Klasse Adressen als Sender oder
# Empfänger verwerfen
```

```
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_A -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_A -j REJECT -l
ipchains -A output -i $CHOKE_DMZ_INTERFACE -d $CLASS_A -j REJECT -l
ipchains -A output -i $CHOKE_LAN_INTERFACE -d $CLASS_A -j REJECT -l

# Pakete mit privaten B-Klasse Adressen als Sender oder
# Empfänger verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_B -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_B -j REJECT -l
ipchains -A output -i $CHOKE_DMZ_INTERFACE -d $CLASS_B -j REJECT -l
ipchains -A output -i $CHOKE_LAN_INTERFACE -d $CLASS_B -j REJECT -l

# Pakete mit D-Klasse Adressen als Sender verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_D -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_D -j REJECT -l
ipchains -A output -i $CHOKE_DMZ_INTERFACE -s $CLASS_D -j REJECT -l
ipchains -A output -i $CHOKE_LAN_INTERFACE -s $CLASS_D -j REJECT -l

# Pakete mit E-Klasse Adressen verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $CLASS_E -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $CLASS_E -j REJECT -l

# Pakete mit Loopback Adressen als Sender verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE -s $LOOPBACK -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE -s $LOOPBACK -j REJECT -l

# Pakete mit fehlerhaften Broadcast Adressen verwerfen
ipchains -A input -i $CHOKE_DMZ_INTERFACE \
         -s $BROADCAST_DEST -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE \
         -s $BROADCAST_DEST -j REJECT -l
ipchains -A input -i $CHOKE_DMZ_INTERFACE \
         -d $BROADCAST_SRC -j REJECT -l
ipchains -A input -i $CHOKE_LAN_INTERFACE \
         -d $BROADCAST_SRC -j REJECT -l

# source-quench zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $DMZ_ADDRESSES 4 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 4 -d $DMZ_ADDRESSES -j ACCEPT

# parameter-problem zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 12 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 12 -d $ANYWHERE -j ACCEPT

# destination-unreachable zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $ANYWHERE 3 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
         -s $CHOKE_DMZ_IPADDR 3 -d $ANYWHERE -j ACCEPT

# time-exceeded zur DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
```

```
          -s $BASTION_DMZ_IPADDR 11 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
          -s $CHOKE_DMZ_IPADDR 11 -d $BASTION_DMZ_IPADDR -j ACCEPT

# Ausgehendes Ping ins Internet
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
          -s $CHOKE_DMZ_IPADDR 8 -d $ANYWHERE -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
          -s $ANYWHERE 0 -d $CHOKE_DMZ_IPADDR -j ACCEPT

# Ankommende Pings aus der DMZ
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p icmp\
          -s $DMZ_ADDRESSES 8 -d $CHOKE_DMZ_IPADDR -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p icmp\
          -s $CHOKE_DMZ_IPADDR 0 -d $DMZ_ADDRESSES -j ACCEPT

# Der DNS-Server der Choke gibt Anfragen an die Bastion (UDP 53)
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p udp\
          -s $CHOKE_DMZ_IPADDR 53\
          -d $BASTION_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp\
          -s $BASTION_DMZ_IPADDR 53\
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT

# Der Nameserver der Choke lässt Anfragen aller Clients aus der
# DMZ zu (TCP/UDP 53)
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp \
          -s $DMZ_ADDRESSES $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p udp \
          -s CHOKE_DMZ_IPADDR 53 \
          -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp \
          -s $DMZ_ADDRESSES $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 53 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y \
          -s CHOKE_DMZ_IPADDR 53 \
          -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT

# Choke als auth Server
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp \
          -s $DMZ_ADDRESSES $UNPRIVPORTS \
          -d $CHOKE_DMZ_IPADDR 113 -j ACCEPT
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp ! -y\
          -s $CHOKE_DMZ_IPADDR 113 \
          -d $DMZ_ADDRESSES $UNPRIVPORTS -j ACCEPT

# Choke als auth-Client
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp\
          -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
          -d $ANYWHERE 113 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y \
          -s $ANYWHERE 113 \
          -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT

# Choke erlaubt SMTP-Pakete an die Bastion
```

```
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp\
         -s CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 25 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y\
         -s $BASTION_DMZ_IPADDR 25 \
         -d CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT

# POP3 Pakete an die Bastion werden weitergegeben:
ipchains -A output -i $CHOKE_DMZ_INTERFACE -p tcp \
         -s $CHOKE_DMZ_IPADDR $UNPRIVPORTS \
         -d $BASTION_DMZ_IPADDR 110 -j ACCEPT
ipchains -A input -i $CHOKE_DMZ_INTERFACE -p tcp ! -y \
         -s $BASTION_DMZ_IPADDR 110 \
         -d $CHOKE_DMZ_IPADDR $UNPRIVPORTS -j ACCEPT

# Vollständige Öffnung der Kommunikation zwischen LAN und Choke
ipchains -A input -i $CHOKE_LAN_INTERFACE \
         -s $LAN_ADDRESSES -j ACCEPT
ipchains -A output  -i $CHOKE_LAN_INTERFACE \
         -d $LAN_ADDRESSES -j ACCEPT

# Choke maskiert alle Pakete des LAN
ipchains -A forward -i $CHOKE_DMZ_INTERFACE \
         -s $LAN_ADDRESSES -j MASQ
```

Anhang C

Protokollbeschreibungen

Hier finden Sie die wichtigsten Protokollbeschreibungen für Dienste im Internet, mit deren Hilfe eine Firewall beliebig aufgebaut werden kann. Das Format ist immer gleich, für jede Zeile jeder Tabelle ist eine ipchains–Regel erforderlich. Die Beschreibungen sind ausführlich, nicht in jedem Fall müssen alle Zeilen beachtet werden, aber für alle Fälle sind hier jeweils die gesammten Kommunikationswege dargestellt.

Die Angaben der Adressen beziehen sich natürlich zunächst auf eine simple ein–Rechner–Firewall. Sollen sie für komplexere Situationen dienen, so muß anstelle der Angabe ADDR z.B. eine Adressangabe für ein ganzes Netz stehen…

C.1 DNS

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	UDP	Nameserver	53	output	IPADDR	1024:65535	–
Antwort des fremden Servers	UDP	Nameserver	53	input	IPADDR	1024:65535	–
Anfrage eines lokalen Clients	TCP	Nameserver	53	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	Nameserver	53	input	IPADDR	1024:65535	ACK
Anfrage des lokalen Servers	UDP	Nameserver	53	output	IPADDR	53 oder andere	–
Antwort des fremden Servers	UDP	Nameserver	53	input	IPADDR	53 oder andere	–
Bitte um Zone Transfer	TCP	Primärer Nameserver	53	output	IPADDR	1024:65535	Egal
Antwort auf Bitte um Zone Transfer	TCP	Primärer Nameserver	53	input	IPADDR	1024:65535	ACK
Anfrage eines fremden Clients	UDP	DNS Client	1024:65535	input	IPADDR	53	–
Antwort des lokalen Servers	UDP	DNS Client	1024:65535	output	IPADDR	53	–
Anfrage eines fremden Clients	TCP	DNS Client	1024:65535	input	IPADDR	53	Egal
Antwort des lokalen Servers	TCP	DNS Client	1024:65535	output	IPADDR	53	ACK
Anfrage eines fremden Servers	UDP	DNS Server	53 oder andere	input	IPADDR	53	–
Antwort des lokalen Servers	UDP	DNS Server	53 oder andere	output	IPADDR	53	–
Jemand bittet um Zone Transfer	TCP	Sekundärer Nameserver	1024:65535	input	IPADDR	53	Egal
Antwort auf Bitte um Zone Transfer	TCP	Sekundärer Nameserver	1024:65535	output	IPADDR	53	ACK

Tabelle C.1: Das DNS–Protokoll

C.2 identd oder auth

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage von lokalem Client	TCP	ANYWHERE	113	output	IPADDR	1024:65535	Egal
Antwort von fremdem Server	TCP	ANYWHERE	113	input	IPADDR	1024:65535	ACK
Anfrage von fremdem Client	TCP	ANYWHERE	1024:65535	input	IPADDR	113	Egal
Antwort von lokalem Server	TCP	ANYWHERE	1024:65535	output	IPADDR	113	ACK

Tabelle C.2: Das identd–Protokoll

C.3 Usenet News (NNTP)

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	News Server	119	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	News Server	119	input	IPADDR	1024:65535	ACK
Anfrage eines fremden Clients	TCP	NNTP Client	1024:65535	input	IPADDR	119	Egal
Antwort des lokalen Servers	TCP	NNTP Client	1024:65535	output	IPADDR	119	ACK
Anfrage des lokalen Servers	TCP	Newsfeed	119	output	IPADDR	1024:65535:Egal	
Antwort des fremden Servers	TCP	Newsfeed	119	input	IPADDR	1024:65535:ACK	

Tabelle C.3: Das NNTP–Protokoll

C.4 E-Mail (SMTP/POP/IMAP)

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Abgehende Mail senden	TCP	ANYWHERE	25	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	ANYWHERE	25	input	IPADDR	1024:65535	ACK
Ankommende Mail empfangen	TCP	ANYWHERE	1024:65535	input	IPADDR	25	Egal
Antwort des lokalen Servers	TCP	ANYWHERE	1024:65535	output	IPADDR	25	ACK
POP Anfrage eines lokalen Clients	TCP	POP Server	110	output	IPADDR	1024:65535	Egal
Antwort des fremden POP Servers	TCP	POP Server	110	input	IPADDR	1024:65535	ACK
Anfrage eines fremden POP Clients	TCP	POP Client	1024:65535	input	IPADDR	110	Egal
Antwort des lokalen POP Servers	TCP	POP Client	1024:65535	output	IPADDR	110	ACK
IMAP Anfrage eines lokalen Clients	TCP	IMAP Server	143	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	IMAP Server	143	input	IPADDR	1024:65535	ACK
Anfrage eines fremden IMAP Clients	TCP	IMAP Client	1024:65535	input	IPADDR	143	Egal
Antwort des lokalen Servers	TCP	IMAP Client	1024:65535	output	IPADDR	143	ACK

Tabelle C.4: Die E-Mail Protokolle

C.5 Telnet

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	ANYWHERE	23	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	ANYWHERE	23	input	IPADDR	1024:65535	ACK
Anfrage eines fremden Clients	TCP	Telnet Client	1024:65535	input	IPADDR	23	Egal
Antwort des lokalen Servers	TCP	Telnet Client	1024:65535	output	IPADDR	23	ACK

Tabelle C.5: Das Telnet–Protokoll

C.6 Secure Shell (SSH)

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	ANYWHERE	22	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	ANYWHERE	22	input	IPADDR	1024:65535	ACK
Anfrage eines lokalen Clients	TCP	ANYWHERE	22	output	IPADDR	513:1023	Egal
Antwort des fremden Servers	TCP	ANYWHERE	22	input	IPADDR	513:1023	ACK
Anfrage eines fremden Clients	TCP	ssh Clients	1024:65535	input	IPADDR	22	Egal
Antwort des lokalen Servers	TCP	ssh Clients	1024:65535	output	IPADDR	22	ACK
Anfrage eines fremden Clients	TCP	ssh Clients	513:1023	input	IPADDR	22	Egal
Antwort des lokalen Servers	TCP	ssh Clients	513:1023	output	IPADDR	22	ACK

Tabelle C.6: Das SSH–Protokoll

C.7 FTP

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	ANYWHERE	21	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	ANYWHERE	21	input	IPADDR	1024:65535	ACK
Datenkanal Aufbau vom fremden Server, aktiver Modus	TCP	ANYWHERE	20	input	IPADDR	1024:65535	Egal
Antwort auf Kanalaufbau durch lokalen Client, akt. Modus	TCP	ANYWHERE	20	output	IPADDR	1024:65535	ACK
Datenkanal Aufbau zum fremden Server, passiver Modus	TCP	ANYWHERE	1024:65535	output	IPADDR	1024:65535	Egal
Antwort auf Kanalaufbau durch fremden Server, passiver Modus	TCP	ANYWHERE	1024:65535	input	IPADDR	1024:65535	ACK
Anfrage eines fremden Clients	TCP	ANYWHERE	1024:65535	input	IPADDR	21	Egal
Antwort des lokalen Servers	TCP	ANYWHERE	1024:65535	output	IPADDR	21	ACK
Datenkanal Aufbau vom lok. Server, aktiver Modus	TCP	ANYWHERE	1024:65535	output	IPADDR	20	Egal
Antwort auf Kanalaufbau durch fremden Client, akt. Modus	TCP	ANYWHERE	1024:65535	input	IPADDR	20	ACK
Datenkanal Aufbau zum lok. Server, passiver Modus	TCP	ANYWHERE	1024:65535	input	IPADDR	1024:65535	Egal
Antwort auf Kanalaufbau durch lok. Server, passiver Modus	TCP	ANYWHERE	1024:65535	output	IPADDR	1024:65535	ACK

Tabelle C.7: Das FTP–Protokoll

C.8 HTTP - Normal

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	ANYWHERE	80	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	ANYWHERE	80	input	IPADDR	1024:65535	ACK
Anfrage eines fremden Clients	TCP	ANYWHERE	1024:65535	input	IPADDR	80	Egal
Antwort des lokalen Servers	TCP	ANYWHERE	1024:65535	output	IPADDR	80	ACK

Tabelle C.8: Das HTTP–Protokoll

C.9 HTTP - mit Secure Socket Layer (SSL)

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	ANYWHERE	443	output	IPADDR	1024:65535	Egal
Antwort des fremden Servers	TCP	ANYWHERE	443	input	IPADDR	1024:65535	ACK
Anfrage eines fremden Clients	TCP	ANYWHERE	1024:65535	input	IPADDR	443	Egal
Antwort des lokalen Servers	TCP	ANYWHERE	1024:65535	output	IPADDR	443	ACK

Tabelle C.9: Das HTTP/SSL–Protokoll

C.10 HTTP-Proxy Zugriff

Beschreibung	Protokoll	Remote IP	Remote Port	Chain	Local IP	Local Port	TCP Flags
Anfrage eines lokalen Clients	TCP	Proxy Server	Proxy Port	output	IPADDR	1024:65535	Egal
Antwort des Proxy Servers	TCP	Proxy Server	Proxy Port	input	IPADDR	1024:65535	ACK

Tabelle C.10: Das Web–Proxy–Protokoll