INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Linux-basierte Personal Firewall für den Einsatz im LRZ

Alexander Müller

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Linux-basierte Personal Firewall für den Einsatz im LRZ

Alexander Müller

Aufgabensteller:	Priv. Doz. Dr. H. Reiser
Betreuer:	Stefan Metzger (LRZ) Claus Wimmer (LRZ)
Abgabetermin:	4. November 2010

Hiermit versichere ich, dass ich die vorliegende Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 4. November 2010

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Am Leibniz Rechenzentrum in Garching wird eine große Anzahl von Desktop-PCs und Server auf Basis des Betriebssystems Linux betrieben. Um einen sicheren Betrieb im Netz sicherzustellen, ist es nötig, geeignete und einfach zu wartende Schutzmaßnahmen zu ergreifen. Als Firewall-Lösung existiert z.B. unter SUSE Linux zwar ein einfach konfigurierbares Programm, welches jedoch nur sehr eingeschränkt geeignet ist, die flexiblen und vielfältigen Anforderungen eines Rechenzentrums zu erfüllen. Aus diesem Grund wurde im Rahmen dieses Fortgeschrittenenpraktikums eine Marktanalyse durchgeführt, bei der aber kein geeignetes Produkt gefunden wurde. Deshalb wurde ein neues Programm entwickelt, das sehr flexibel und zugleich einfach zu konfigurieren ist.

Inhaltsverzeichnis

1 Einleitung

Das Leibniz Rechenzentrum in Garching bei München [lrz] ist das zentrale Rechenzentrum für die Münchner Universitäten und Forschungseinrichtungen. In diesem Zusammenhang betreibt das LRZ das München Wissenschaftsnetz (MWN) - ein großflächiges Netz, welches Institutionen in München und Umgebung an interne Dienste des Rechenzentrums sowie an das Internet anbindet.

Zusätzlich stellt das LRZ eine große Zahl von IT-Dienstleistungen zur Verfügung, wozu ein großer Server-Park betrieben wird. Viele dieser Server sind nur aus dem MWN oder anderen internen Netzen erreichbar, einige jedoch auch direkt aus dem Internet.

Ein extern verfügbarer IT-Dienst stellt grundsätzlich ein mögliches Sicherheitsrisiko dar.

Um dennoch eine angemessene Sicherheit für die einzelnen Systeme zu gewährleisten ist es nötig, den Zugang auf diese so stark wie möglich einzuschränken. So ist es beispielsweise sinnvoll, einen Server oder einzelne Dienste nur von bestimmten Orten aus zugängig zu machen. Dies reduziert den Kreis möglicher Angreifer bereits deutlich.

Als groben Zugangsschutz zu den Servern werden am LRZ dazu virtuelle Firewalls an den Routern eingesetzt, die das gesamte Netz virtuell in einzelne Zonen unterteilen [Haz08]. Dieser Schutz erfolgt üblicherweise auf Basis von IP-Subnetzen und schränkt folglich nur die Anzahl der möglichen Clients ein. Es gibt beispielsweise Netze, die ausschließlich aus dem LRZ-Gebäude erreichbar sind. Befindet sich ein Angreifer nicht im LRZ-Gebäude, so muss er sich erst physisch Zugang verschaffen oder auf anderem Wege in das interne Netz eindringen. Die Hürden für einen erfolgreichen Angriff steigen dadurch beträchtlich.

Für den feingranularen Zugangsschutz auf den einzelnen Servern werden weitere Maßnahmen benötigt. Diese umfassen sowohl TCP-Wrapper [twr] als auch Personal Firewalls. Die aktuell eingesetzten Firewalls auf den Linux-Servern des LRZ sind für diese Aufgabe jedoch sehr unflexibel und es wird aktuell nach einer besseren Software gesucht, welches diese Aufgaben in Zukunft übernehmen kann.

1.1 Aufgabenstellung

Ziel dieses Fortgeschrittenenpraktikums ist die Evaluierung der aktuellen Firewall-Situation bei den Linux-Servern und Linux-Desktop-PCs am Leibniz Rechenzentrum und einer anschließenden Verbesserung eben dieser. Der Fokus liegt dabei speziell auf dem Funktionsumfang, der Flexibilität, dem Konfigurationsaufwand und einer weitestgehenden Automatisierung des Setups.

Dazu kommen sowohl die Anpassung einer bereits am Markt erhältlichen Firewall-Lösung auf Open-Source-Basis, als auch eine Weiterentwicklung der alten Firewalls oder eine vollständige Neuentwicklung in Frage.

1.2 Vorgehensweise

Es wurde zunächst eine genaue Analyse der bereits verwendeten Firewalls angefertigt (Kapitel 3). Zusammen mit den Betreuern und den zuständigen Administratoren am LRZ wurde anschließend die Situation diskutiert und ein Katalog von Anforderungen erarbeitet, der als Grundlage für eine zukünftige Firewall dienen soll (Kapitel 4). Als nächster Schritt erfolgte eine Marktanalyse, speziell nach Open-Source-Firewalls. Die einzelnen Produkte wurden auf Basis des Anforderungskatalog untersucht und bewertet, ob sich ein Einsatz am LRZ lohnt bzw. ob ggf. eine Anpassung der Produkte in Frage kommt (Kapitel 5).

Da dabei keine geeignete Firewall gefunden wurde, wurde beschlossen, eine eigene Firewall zu entwickeln (Kapitel 6).

2 Grundlagen

2.1 Rechnernetz

> Der Begriff Rechnernetz wird für mehrere miteinander mit einer bestimmten Technologie verbundene autonome Computer verwendet. Zwei Computer gelten als miteinander verbunden, wenn sie Informationen austauschen können. [Tan03]

Als Übertragungstechnologien kommen nicht nur Kupferkabel, sondern auch Funk, Lichtleiter und viele andere in Frage. In Rechnernetzen werden heute hauptsächlich zwei verschiedene Vermittlungstechniken eingesetzt: Leitungsvermittlung und Paketvermittlung.

Bei der Leitungsvermittlung wird vor Beginn der Kommunikation ein durchgehender physischer Pfad zwischen den Kommunikationspartnern aufgebaut, über den anschließend die Daten übertragen werden. Diese Technik wird beispielsweise in vielen Telefonnetzen eingesetzt.

Bei der Paketvermittlung werden alle Nachrichten in kleine Einheiten (Pakete) zerlegt, die einzeln in das Netz eingespeist werden. Dort werden sie weiter vermittelt und zum Zielrechner weitergeleitet, der sie wiederum zur vollständigen Nachricht zusammensetzt.

Im Kontext dieser Arbeit werden nur paketvermittelte Netze behandelt, wie sie typischerweise in modernen Kommunikationsnetzen auf TCP/IP-Basis vorzufinden sind.

Vgl. [Tan03].

2.2 OSI-Referenzmodell

Zur Beschreibung von Kommunikationsprotokollen in Netzen existiert das OSI-Referenzmodell (Open Systems Interconnection Reference Model [osi94]). Es definiert sieben aufeinander aufbauende Schichten, die jeweils eigene Aufgaben beschreiben und den Kommunikationsvorgang funktional untergliedern. Alle Schichten bieten für die darüberliegenden Schichten Schnittstellen an um den Zugriff zu abstrahieren. Auf diese Weise sind für eine Schicht alle darunterliegenden Schichten transparent. Um die Funktionen der einzelnen Schichten zu implementieren werden Netzprotokolle definiert, die eine Realisierung des Modells darstellen. Ein Verbund von Netzprotokollen, der das OSI-Referenzmodell implementiert wird auch Protokollstapel genannt.

Eintreffende Nachrichten durchlaufen die Schichten aufsteigend von Schicht 1 beginnend, ausgehende Nachrichten in umgekehrter Reihenfolge. Bei einer Kommunikation zwischen zwei Endgeräten kommunizieren gleiche Schichten virtuell direkt miteinander, d.h. alle darunterliegenden Schichten werden abstrahiert.

Eine grafische Übersicht zum OSI-Referenzmodell findet sich in Abbildung 2.1.

Vgl. [Tan03].

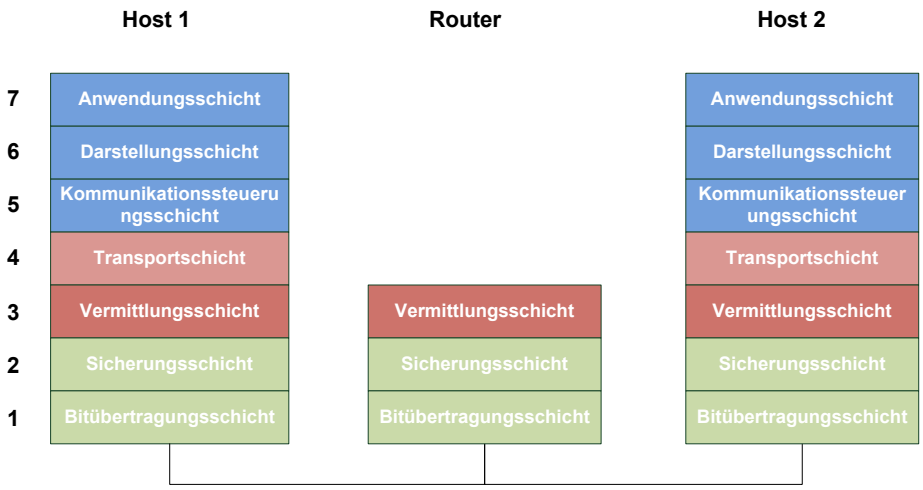

Abbildung 2.1: OSI-Referenzmodell mit zwei Hosts und Transitsystem

2.2.1 Übersicht

2.2.2 Schicht 1 (Bitübertragungsschicht)

Schicht 1 (Bitübertragungsschicht) beschreibt die physikalische Kommunikation, d.h. die Übertragung von einzelnen Bits. Dazu gehören u.A.:

- Das physische Kommunikationsmedium, z.B. Kabel, Licht, Funk, Schall, ..
- Die Modulations- und Kodierungsverfahren, um Daten über das physische Medium zu übertragen
- Falls kabelgebunden:
  - Die Art der Verkabelung, z.B. Kabel- und Steckertypen, Pin-Belegung, ..
  - Die Anordnung der Endgeräte zueinander, z.B. Stern, Ring, Baum, ..

2.2.3 Schicht 2 (Sicherungsschicht)

Schicht 2 (Sicherungsschicht) beschreibt die Verfahren zur zuverlässigen und fehlerfreien Kommunikation innerhalb eines Netzsegments basierend auf Schicht 1. Dazu gehören u.A.:

- Zusammenfassung von Bits zu Netzframes
- Adressierung der physischen Endgeräte

- Erkennung und Korrektur von Übertragungsfehlern
- Zugriffssteuerung für gemeinsam genutzte Übertragungsmedien (Kollisionsvermeidung, ..)

Die Schichten 1 und 2 sind spezifisch für das verwendete Übertragungsmedium. Auf ihrer Basis können nur Geräte miteinander kommunizieren, die sich im gleichen Netzsegment befinden.

Der in vielen Netzen verwendete Ethernet-Standard (IEEE 802.3 [iee]) beschreibt sowohl Schicht 1 als auch das Ethernet-Protokoll auf Schicht 2.

2.2.4 Schicht 3 (Vermittlungsschicht)

Schicht 3 (Vermittlungsschicht) beschreibt die Kommunikation zwischen Endgeräten, die sich nicht im gleichen Netzsegment befinden müssen. Im Falle einer paketorientierten Kommunikation gehört hier insbesondere die Wegsuche (Routing) von Datenpaketen dazu. Dazu werden Transitsysteme (Router) verwendet, die mehrere Netzsegmente miteinander verbinden und somit überbrücken können (2.2.8).

Protokolle auf Basis von Schicht 3 ermöglichen die Kommunikation von Endgeräten im gesamten Netz, unabhängig davon, welche physischen Netze zugrunde liegen.

Die Protokolle IPv4 und IPv6 gehören zu Schicht 3.

2.2.5 Schicht 4 (Transportschicht)

Schicht 4 (Transportschicht) setzt auf die Vermittlungsschicht auf und bietet einen netzunabhängigen Transport von Nachrichten zwischen zwei Endsystemen. Aufgaben von Schicht 4 sind auch die Fragmentierung von größeren Datenmengen auf einzelne Pakete und Mechanismen zur Stauvermeidung.

Es wird für die Schichten 5 bis 7 ein transparenter Zugang zum Netz zur Verfügung gestellt.

Die auf IP aufbauenden Protokolle TCP (Transmission Control Protocol, verbindungsorientiert) und UDP (User Datagram Protokol, verbindungslos) gehören zu Schicht 4.

2.2.6 Schicht 5 (Kommunikationssteuerungsschicht)

Schicht 5 (Kommunikationssteuerungsschicht) setzt auf die Transportschicht auf und dient der Verwaltung von logischen Sitzungen. Dazu gehören auch das Wiederherstellen einer Sitzung nach einem Verbindungsabbruch sowie weitere Features wie z.B. Checkpoints, auf die zu einem späteren Zeitpunkt zurückgegriffen werden kann.

2.2.7 Schicht 6 (Darstellungsschicht)

Schicht 6 (Darstellungsschicht) setzt auf die Kommunikationssteuerungsschicht auf und beschreibt Protokolle zur systemunabhängigen Darstellung von Daten (Syntax). Ebenso fallen Aufgaben wie Kompression und Verschlüsselung in diese Schicht.

2.2.8 Schicht 7 (Anwendungsschicht)

Schicht 7 (Anwendungsschicht) setzt auf die Darstellungsschicht auf und stellt Anwendungsprogrammen und Diensten den Zugang zum Netz bereit. In dieser Schicht finden sich die

meisten dem Benutzer geläufigen Protokolle wie z.B. HTTP, FTP, SMTP, SSH und viele mehr.

2.2.9 Endsystem/Transitsystem

Anhand des OSI-Referenzmodells lässt sich zwischen Endsystemen und Transitsystemen unterscheiden.

Ein Endsystem ist z.B. ein Server oder ein Desktop-PC. Sie zeichnen sich dadurch aus, dass sie den gesamten Protokollstack von Schicht 1 bis Schicht 7 implementieren und somit in der Lage sind als eigenständiges System zu kommunizieren.

Im Gegensatz dazu ist ein Transitsystem (Router) dazu gedacht, unterschiedliche physische Netze miteinander zu verbinden. Dazu sind mindestens die OSI-Schichten 1, 2 und 3 implementiert, was den Systemen einen Zugang zur Vermittlungsschicht ermöglicht. Da diese Schicht die unterste netzunabhängige Schicht ist, wird sie zwingend benötigt um Netzsegmente unterschiedlicher Technologien miteinander zu verbinden. Viele IP-Transitsysteme haben zusätzlich auch höhere Schichten implementiert um Routing-Entscheidungen auch z.B. von Eigenschaften der Transportschicht (speziell Portnummern) abhängig zu machen. Zum Remote-Management der Transitsysteme ist oftmals zusätzlich ein vollständiger Protokollstack wie bei einem Endsystem implementiert, der jedoch nicht für die Kernfunktionalität zuständig ist.

Zwischen zwei Endgeräten können sich prinzipiell eine beliebige Anzahl von Transitgeräten befinden. Ihre Zahl ist im Falle von IP lediglich durch den IP-Header-Wert TTL (Time to Live, IPv4) bzw. HL (Hop limit, IPv6) begrenzt.

Meine Projektarbeit am Leibniz Rechenzentrum beschäftigt sich nur mit Firewalls für Endsysteme.

2.2.10 TCP/IP-Referenzmodell

Im Kontext des Internet-Protokolls wird das OSI-Referenzmodell zum TCP/IP-Referenzmodell vereinfacht. Dazu werden die Schichten 1 und 2 zu einer gemeinsamen Netzzugriffsschicht und die Schichten 5, 6 und 7 zu einer gemeinsamen Anwendungsschicht zusammengefasst. Dies entspricht im Allgemeinen auch eher dem Programmiermodell mit dem das IP-Protokoll unter den meisten Betriebssystemen implementiert ist. [Tan03]

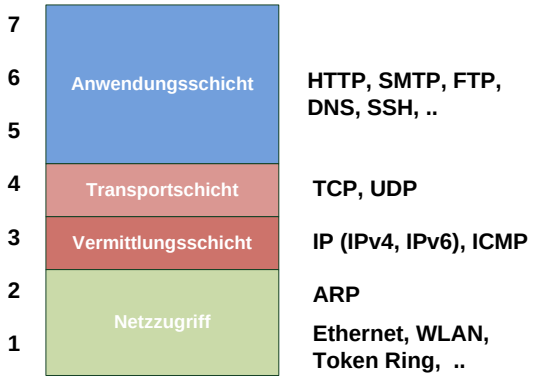

Abbildung 2.2: OSI-Referenzmodell im Kontext von IP

2.3 Firewalls

Eine Firewall (zu Deutsch Brandmauer) stellt einen Mechanismus dar, um den Datenaustausch zwischen zwei oder mehr Netzen oder Netzsegmenten zu kontrollieren und zu reglementieren, z.B. einem LAN (Local Area Network) und dem Internet (WAN - Wide Area Network). Auf diesem Wege lassen sich im Netz mehrere Zonen mit unterschiedlichen Sicherheits-Levels aufbauen. Generell ist es möglich Firewalls sowohl in Form von Hardware, als auch in Form von Software zu realisieren. Sie werden in der Regel eingesetzt, um bestimmte durch einen Anwender oder Administrator vorgegebene Sicherheitsrichtlinien im Netz durchzusetzen.

Dazu fungieren Firewalls als Filtersysteme, die sich physisch oder virtuell zwischen zwei oder mehreren kommunizierenden Endgeräten befinden. Eine Firewall besitzt dazu einen Regelsatz, der detailliert beschreibt, welche Vorgänge im Netz erlaubt sind und welche nicht. Darüberhinaus sind oft erweiterte Funktionen möglich, z.B. das Logging von bestimmten Ereignissen oder das Bearbeiten und Weiterleiten von Netzpaketen.

Im Kontext des Internet-Protokolls arbeiten die meisten Firewalls auf den OSI-Schichten 3 und 4. Einige Firewalls bieten weitergehende Funktionen zur Analyse und Filterung auf der Anwendungsschicht an.

Es existieren auch Firewalls für Schicht 2, die z.B. für Ethernet-Bridges eine wichtige Rolle spielen. [ebt]

2.3.1 Subtypen

Es wird zwischen reinen Netz-Firewalls und sogenannten Personal Firewalls unterschieden.

Netz-Firewalls befinden sich an einem zentralen Ort im Netz (oftmals in Kombination mit einem Router) und überwachen dort die Kommunikation zwischen den von ihnen verwalteten Netzen. Diese Sorte von Firewalls ist meist transparent angelegt und somit für den Endnutzer unsichtbar. Auf den Zustand der Endgeräte kann nur über die mitgeschnittene Kommunikation geschlossen werden, denn es existiert kein direkter Zugang zu den Endgeräten.

Im Gegensatz dazu arbeiten Personal Firewalls direkt auf den einzelnen Endgeräten, z.B. auf einem Server oder einem Desktop-PC. Sie verfügen aus diesem Grund über weitere Zustandsinformationen, die sie direkt aus dem Endgerät entnehmen können. Es ist somit z.B. auch möglich einzelne lokale Prozesse direkt zu überwachen, und ihren Zugang zum Netz entsprechend zu reglementieren.

2.3.2 Zustandsbehaftung

Außerdem unterscheidet man noch zustandslose (stateless) und zustandsbehaftete (stateful) Firewalls.

Zustandslos bedeutet, dass die Firewallregeln statisch auf alle Netzpakete angewendet werden, unabhängig davon, was zuvor kommuniziert wurde. Dies ist der einfachste Firewalltyp.

Bei einer zustandsbehafteten Firewall hängt die Verarbeitung von Datenpaketen hingegen auch von der vorhergegangenen Kommunikation ab. Beispielsweise könnte eine solche Firewall intern eine Liste von aktiven Kommunikationsverbindungen als Zustand halten. Ein solcher Connection-Tracker ermöglicht es, beliebige eingehende Pakete auf ihre Zugehörigkeit zu einer bestimmten Verbindung zu prüfen und so ggf. gesonderte Regelsätze darauf

anzuwenden. Dieser Firewalltyp ist deutlich komplexer und benötigt zum Speichern der Zustandsinformationen außerdem zusätzlich freien Speicherplatz.

Der Regelsatz kann durch eine Zustandsbehaftung deutlich vereinfacht werden. Desweiteren wird es möglich den Regelsatz automatisch an bestimmte Rahmenbedingen anzupassen. So ist es beispielsweise beim File Transfer Protocol (FTP) möglich die eingehende Datenverbindung per zustandsbehaftetem Regelsatz freizugeben. Da diese Verbindungen im Normalfall über zufällige Portnummern abgewickelt werden, ist es nicht möglich, dies nur mit zustandslosen Firewallregeln freizugeben. Es wird auf diesem Wege auch die Sicherheit erhöht, da es nicht mehr nötig ist prophylaktisch große Portranges freizugeben.

2.4 Netfilter/Iptables unter Linux

Unter Linux wird mit der Kernel-Version 2.6 das Firewallframework *Netfilter* ausgeliefert [iptc]. Es handelt sich dabei um ein aufwendiges Hooking-Framework für den Linux-Kernel mit dem bestimmte Netzevents abgefangen werden können, z.B. der Empfang oder die Übertragung von Datenpaketen aus dem Netz.

Die *iptables*-Kernelmodule arbeiten als Client am *Netfilter* und bieten vordefinierte Tabellen an, in die der Administrator Firewall-Regeln eintragen kann. *iptables* stellt Module für die Netzprotokolle IPv4, IPv6 und ARP zur Verfügung. Darüberhinaus gibt es auch ein Modul zum Filtern von Ethernet-Bridges [ebt].

Zu beachten ist, dass für jedes dieser Protokolle eigene, voneinander unabhängige Tabellen (Tables) existieren!

Im Rahmen dieser Projektarbeit wird nur auf IPv4 und IPv6 näher eingegangen ([iptb], [ipta]).

Eine Übersicht über die *iptables* für IPv4 und IPv6 findet sich in Abbildung 2.3.

2.4.1 Übersicht

2.4.2 Tables

Die *iptables* für IPv4 und IPv6 unterteilen sich jeweils in mehrere voneinander unabhängige Tabellen:

- filter: Filterung von ein- oder ausgehenden sowie weitergeleiteten Paketen.
- nat (aktuell nur IPv4): Ermöglicht Network Address Translation (NAT).
- mangle: Manipulation von Netzpaketen.

Jede dieser Tabellen besitzt eine Menge von sog. Standard-Chains, die in bestimmter Korrelation zueinander von einzelnen Netzpaketen durchlaufen werden.

Für die Zugriffskontrolle am lokalen Host ist insbesondere die filter-Tabelle relevant.

2.4.3 Chains

Ein leerer *iptables*-Regelsatz zum Filtern von Netzdaten besteht zunächst aus drei leeren Chains. Diese Chains sind geordnete Listen von einzelnen Filter-Regeln, die von Netzpaketen sequentiell durchlaufen werden. Eine Regel enthält üblicherweise eine bestimmte Kombination von Matches und ein Target.

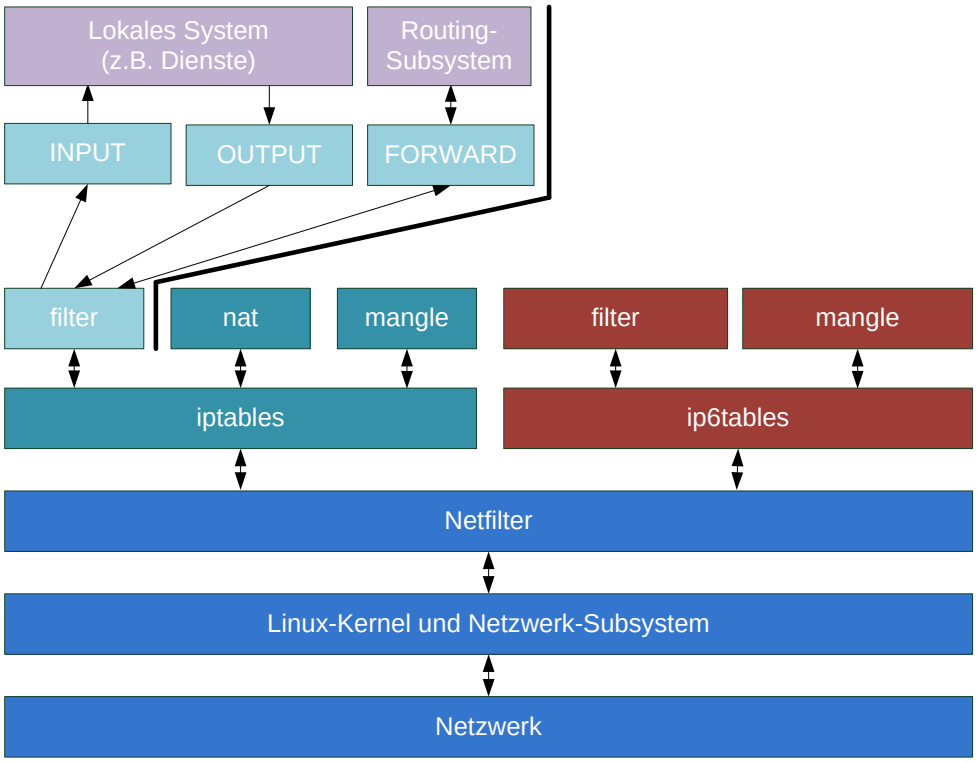

Abbildung 2.3: iptables - Aufbau und Datenfluss

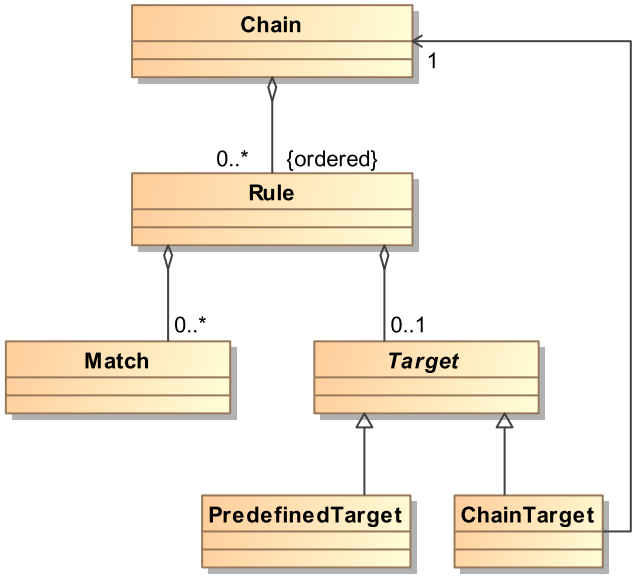

Abbildung 2.4: iptables - Übersicht Chains, Rules, Matches, Targets

In den Matches wird eine bestimmte Menge von Kriterien festgelegt, anhand derer entschieden wird, ob ein Netzpaket auf die Regel matched oder nicht. Im Falle von *iptables* werden alle Matches einer einzelnen Regel über ein logisches AND verknüpft, d.h. ein Paket kann nur dann auf eine Regel matchen wenn alle ihre Matches erfüllt sind. Fehlt bei einer Regel die Angabe von Matches, so matchen implizit alle Netzpakete.

Das Target legt fest was passiert, wenn ein Paket auf die Regel matched. *iptables* bietet eine Reihe von Standard-Targets. Es ist jedoch auch möglich benutzerdefinierte Chains anzulegen und diese als Target anzugeben. Dies wird ähnlich eines Prozeduraufrufs in der Programmierung behandelt. Das Standard-Target RETURN dient in diesem Fall für den Rücksprung zum Caller. Targets können terminierend oder nicht-terminierend sein. Ein terminierendes Target beendet die Abarbeitung für das aktuelle Paket, ein nicht-terminierendes Target nicht.

Abbildung 2.4 zeigt ein UML-Klassendiagramm, das die Zusammenhänge zwischen Chain, Rule, Match und Target genauer illustriert.

Die filter-Tabellen besitzen folgende Standard-Chains:

- INPUT: Alle eingehenden Netzpakete, die an das lokale System adressiert sind.
- OUTPUT: Alle ausgehenden Netzpakete, die vom lokalen System gesendet werden.
- FORWARD: Alle weitergeleiteten Netzpakete - also eingehende Pakete, die nicht an das lokale System adressiert sind ($\rightarrow$ Routing).

Als Target stehen standardmäßig zur Verfügung:

- ACCEPT: Paket akzeptieren. (beendet Abarbeitung)
- DROP: Paket verwerfen. (beendet Abarbeitung)
- QUEUE: Paket an ein Usermode-Programm übergeben. (nicht terminierend)
- RETURN: Verarbeitung bei der nächsten Regel in der vorherigen Chain fortsetzen. (nicht terminierend)

Durch diverse Erweiterungen werden noch zusätzliche Targets bereitgestellt, z.B. zum Logging von Paketen.

Als Match stehen standardmäßig Filter wie Ursprungs- und Zieladresse, ein- oder ausgehende Netz-Schnittstelle und das verwendete Netzprotokoll zur Verfügung. Auch diese werden über weitere Module zahlenmäßig stark erweitert.

Alle vordefinierten Chains besitzen eine Default-Policy, die vom Benutzer separat mit einem Target belegt werden kann. Verwendet wird meist entweder ACCEPT oder DROP. Die Policy kommt nur dann zur Anwendung, wenn ein Paket die gesamte Chain durchlaufen hat ohne dass eine Regel gematched hat, oder die Targets aller gematchten Regeln entweder leer waren oder nicht-terminierend waren. Eine Default-Policy DROP ist also gleichbedeutend mit einer Regel am Ende der Chain, die auf alle Pakete matched und als Target DROP hat ("Alles was nicht explizit erlaubt wird, ist verboten" $\rightarrow$ Whitelist).

Abbildung 2.5 zeigt die Reihenfolge in der ein- und ausgehende Pakete die Standard-Chains durchlaufen.

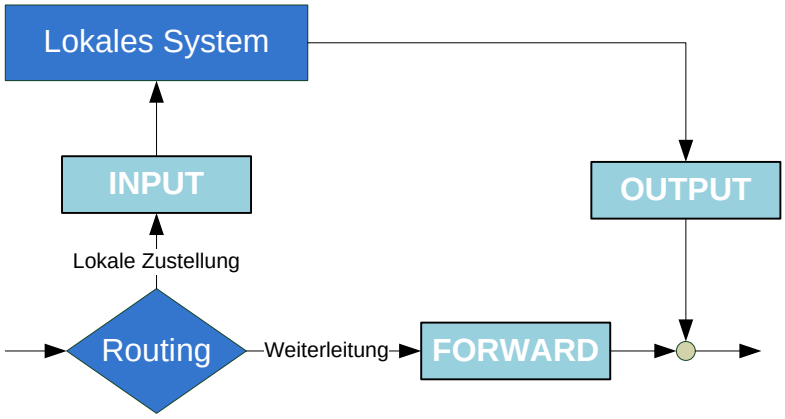

Abbildung 2.5: iptables - Paketfluss Standard-Chains der Tabelle *filter*

2.4.4 Matches

Durch zusätzliche Erweiterungen werden weitere Matches und Targets bereitgestellt.

Im Rahmen der Projektarbeit werden die Matches *state*, *recent*, *hashlimit* sowie weitere protokollspezifische Matches verwendet, die hier kurz vorgestellt werden.

state

Der *state*-Match ist Teil des Connection-Trackers, der in Kapitel 2.4.6 näher beschrieben wird.

recent

Der *recent*-Match ist ein einfacher Mechanismus um im Regelsatz der Firewall eine dynamische Liste von IP-Adressen zu erstellen und anschließend mit verschiedenen Mitteln Matches darauf zu definieren. Jede Liste trägt einen eigenen Namen und kann von mehreren Regeln gelesen und bearbeitet werden. Es kann entweder die Absender- oder die Zieladresse von Paketen für die Liste herangezogen werden.

Einzelne Filter-Regeln können IP-Adressen zur Liste lesen, hinzufügen, updaten oder entfernen. Dies wird meist in Kombination mit anderen Matches durchgeführt, z.B. TCP Port 22 um eine Liste von Verbindungsanfragen an einen SSH-Server zu führen. Als zusätzliche Informationen werden in der Liste u.A. gespeichert wann eine IP-Adresse zuletzt “gesehen” wurde sowie die Zeitpunkte an denen vorhergehende Pakete eingetroffen sind. Deren maximale Anzahl kann über den Parameter *ip_pkt_list_tot* an das Kernel-Modul übergeben werden und hat auf meinem Testsystem einen Default-Wert von 20.

Aus diesen Informationen ist es möglich einen ratenlimitierenden Match zu konstruieren, so dass Pakete die z.B. häufiger als 10x pro Minute auftreten speziell behandelt werden können. Dies ist besonders hilfreich um Fehlkonfigurationen oder Missbrauch zu erkennen. So lassen

sich z.B. exzessive Verbindungsanfragen loggen oder die Anzahl von Verbindungsanfragen pro Minute an bestimmte Dienste beschränken.

Ein einfacher recent-Regelsatz besteht aus mindestens zwei einzelnen Regeln:

```
iptables -A INPUT [weitere Matches] -m recent --name
    "recent-liste" --set   # fuegt die Absender-IP-Adresse in
    die Liste hinzu falls noch nicht vorhanden.
iptables -A INPUT [weitere Matches] -m recent --name
    "recent-liste" --update --hitcount 10 --seconds 60 -j DROP
    # updated die Liste und prueft auf eine Rate von groesser
    oder gleich 10 Paketen in den letzten 60 Sekunden. Falls
    der Match erfolgreich war wird das Paket verworfen.
```

Zusätzlich kann eine IP-Adress-Liste über das proc-Dateisystem gelesen und bearbeitet werden. Beispiel:

```
root@linux-dev:~# cat
    /proc/net/xt_recent/GLOBAL_RULES_000-SSHD_LOG
src=0000:0000:0000:0000:0000:0000:0000:0001 ttl: 64 last_seen:
    212185038 oldest_pkt: 2 212185035, 212185038
src=192.168.178.22 ttl: 128 last_seen: 212194106 oldest_pkt: 3
    212176402, 212193689, 212194106
src=127.0.0.1 ttl: 64 last_seen: 212185038 oldest_pkt: 2
    212185035, 212185038
```

In der Liste befinden sich aktuell drei IP-Adressen (src), jeweils mit dem Zeitpunkt des zuletzt übertragenen Paketes (last_seen) und dessen TTL-Wert (ttl, Time to Live), sowie eine Historie der vorhergehenden Pakete (oldest_pkt). Zur IP-Adresse 192.168.178.22 sind seit Start der Firewall drei Pakete erfasst worden. Die Timestamps sind jeweils linux-spezifische Counter die von internen Einstellungen im Kernel abhängig sind. Aus diesem Grund sind nur die Zeitdifferenzen zwischen verschiedenen Timestamps von Interesse und nicht deren absolute Werte.

limit/hashlimit

Der *limit*-Match besitzt einen eigenen Zähler für jede Regel, der die Anzahl an Paketen zählt, die auf die Regel matchen. Überschreitet der Zähler einen bestimmten Wert, so matched die Regel temporär nicht mehr. Durch ein Timeout-Intervall wird der Zähler anschließend wieder dekrementiert und die Regel wieder aktiviert, sobald das Limit unterschritten wird.
Beispiel (Netzpakete maximal 1x pro Minute loggen):

```
iptables -A INPUT -m limit --limit "1/min" --limit-burst 1 -j
    LOG
```

Der *hashlimit*-Match erweitert diese Funktionalität dahingehend, dass es nicht mehr nur einen einzelnen Zähler pro Regel gibt, sondern Gruppierungskriterien definiert werden können, anhand derer eingehende Pakete zu Gruppen zusammengefasst werden für die jeweils ein eigener Zähler existiert. Als sinnvolle Gruppierung kann z.B. die Absender-IP-Adresse verwendet werden, d.h. pro Absender existiert ein eigener Zähler.
Beispiel (Netzpakete maximal 1x pro Minute loggen, jeweils pro Absender-IP-Adresse):

```
iptables -A INPUT -m hashlimit --hashlimit "1/min"
    --hashlimit-burst 1 --hashlimit-mode srcip
```

```
--hashlimit-name "HASHLIMIT_NAME" -j LOG
```

Der *hashlimit*-Match ist somit ein Gegenstück zum *recent*-Match bei dem nicht ein Überschreiten des Ratenlimits gematched wird, sondern ein Unterschreiten.

Für beide Matches kann darüberhinaus ein Burst-Wert eingestellt werden, der eine Art Softlimit darstellt. Damit kann definiert werden, dass eine bestimmte initiale Anzahl von Paketen immer zulässig ist bevor die Ratenlimitierung einsetzt. Dieses Fenster wird bei Unterschreiten des Ratenlimits wieder neu aufgeladen.

Im Firewall-Regelsatz werden *limit*- oder *hashlimit*-Matches häufig eingesetzt, um in Kombination mit Logging die Anzahl der Logeinträge auf ein sinnvolles Niveau zu reduzieren.

Das *hashlimit*-Match bietet außerdem analog zum *recent*-Match die Möglichkeit der Regel einen Namen zu geben und damit auch über das proc-Dateisystem auf die Hashtabellen zuzugreifen.

udp/tcp/icmp/..

Die *iptables* bieten für häufig verwendete Protokolle jeweils eigene Matches an. Diese werden grundsätzlich in Kombination mit einem Protokoll-Filter (`-p` oder `--protocol`) eingesetzt und bieten zusätzliche Funktionen um auf die Eigenheiten der jeweiligen Protokolle zu matchen.

So bieten der *udp*- und der *tcp*-Match die Möglichkeit nach Portnummern zu matchen sowie beim *tcp*-Match auch auf die Flags im TCP-Header (z.B. Verbindungsanfragen oder Abbrüche). Der *icmp*-Match sowie sein IPv6-Pendant *icmpv6* können verwendet werden um auf bestimmte ICMP-Typen und -Codes wie etwa ICMP-Echo (Ping) zu matchen.

2.4.5 Targets

Neben den Standard-Targets von *iptables* (ACCEPT, DROP, QUEUE und RETURN) werden über Erweiterungen zusätzliche Targets angeboten. Im Rahmen dieser Arbeit wird beispielsweise Logging verwendet. Dazu gibt es die Targets LOG, ULOG und NFLOG, die jeweils Log-Einträge an verschiedene Backends senden. Alle Logging-Targets sind außerdem nicht terminierend, d.h. matched ein Paket auf eine Logging-Regel, so wird zwar ein Log-Eintrag erstellt, jedoch wird die Ausführung mit der nächsten Regel in der Chain fortgesetzt.

Es gibt drei relevante Targets:

- LOG: Erstellt Log-Einträg auf Syslog-Backend.
- ULOG: Erstellt Log-Einträg auf ein oder mehrere Usermode-Backends. Veraltet und durch NFLOG ersetzt.
- NFLOG: Erstellt Log-Einträg auf ein oder mehrere Usermode-Backends, z.B. ulogd. Löst ULOG ab.

2.4.6 Conntrack

Eines der wichtigsten Features der *iptables* ist ein mächtiger Connection-Tracker, der in Kombination mit einem geeigneten Regelsatz eine zustandsbehaftete Firewall bildet. Das zuständige Kernelmodul für den Connection-Tracker überwacht transparent den gesamten

Paketfluss und erstellt in Echtzeit eine Tabelle von Kommunikationsbeziehungen zwischen den Endgeräten, deren Kommunikation die Firewall überwacht.

Der Connection-Tracker arbeitet standardmäßig auf den Schichten 3 und 4 des OSI-Referenzmodells. Es werden also nicht nur Kommunikationsbeziehungen im Sinne von TCP-Verbindungen überwacht. Das Tracking funktioniert also auch bei verbindungslosen Protokollen wie etwa UDP. Selbst IPsec und andere Tunnelprotokolle werden getracked.

Es existieren auch diverse zusätzliche Kernel-Module die den Connection-Tracker erweitern, so dass auch Protokolle aus der Anwendungsschicht getracked werden können. So ermöglicht es z.B. ein FTP-Connection-Tracker, dass bei Verwendung des FTP-Passive-Modes der vom FTP-Server gewünschte Port für die Datenverbindung temporär für den betreffenden Client geöffnet wird.

Im Regelsatz kann über mehrere verschiedene Matches auf die Datenbank des Connection-Trackers zugegriffen werden.

Es existiert z.B. der Match *state*, der einem Paket eines von vier Attributen zuordnet, nach denen gefiltert werden kann:

- NEW: Das Paket gehört noch zu keiner offenen Verbindung. Es muss sich bei NEW allerdings nicht um eine Verbindungsanfrage im Sinne von TCP handeln!
- ESTABLISHED: Das Paket gehört zu einer bereits offenen Verbindung.
- RELATED: Das Paket steht in Beziehung zu einer bereits offenen Verbindung. Beispiele: ICMP Echo Reply, Eingehende Datenverbindung bei PASV-FTP, ..
- INVALID: Alle anderen Pakete. Entweder Fehlerzustand oder nicht zuzuordnen.

Der Connection-Tracker wird im Regelsatz meist so verwendet, dass `ESTABLISHED`- und `RELATED`-Pakete gleich am Anfang der `INPUT`-Chain akzeptiert werden. Im Weiteren werden z.B. bei TCP nur explizite Verbindungsanfragen (SYN-Pakete) berücksichtigt.

2.4.7 Administration der iptables

Zum Bearbeiten von Firewall-Regeln stellt *iptables* eine Reihe von Befehlszeilen-Programmen bereit:

- iptables: IPv4
- ip6tables: IPv6
- arptables: ARP
- ebtables: Ethernet-Bridges

Der Syntax dieser Programme ist identisch aufgebaut, hier am Beispiel von *iptables* für IPv4.

Hinzufügen oder Entfernen einer Regel:

```
iptables [-t table] {-A|-D} chain rule-specification
```

Setzen der Default-Policy für eine Standard-Chain:

```
iptables [-t table] -P chain target
```

Erstellen oder Löschen einer benutzerdefinierten Chain:

```
iptables [-t table] {-N|-X} chain
```

Leeren einer Chains (also Entfernen aller Regeln):

```
iptables [-t table] -F chain
```

Falls kein Table angegeben wird, so wird stets *filter* verwendet.
Der Syntax der *rule-specification* ist

```
rule-specification = [matches...] [target]
match = -m matchname [per-match-options]
target = -j targetname [per-target-options]
```

2.4.8 Beispiele

Stateful mit SSH-Server

```
# accept traffic that is part established connections
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
# accept incoming tcp connections on port 22
iptables -A INPUT -p tcp -m tcp --dport 22 -m state --state
    NEW --syn -j ACCEPT
```

3 Personal Firewalls am LRZ

3.1 SUSE-Firewall

Die Standard-Firewall bei SUSE-Distributionen ist die *SUSE-Firewall*. Bei dieser handelt es sich um ein Frontend für die *iptables*-Firewall unter Linux, welches bereits bei der Standardinstallation von SLES/openSUSE [slea][ope] enthalten ist.

Die Konfiguration erfolgt über ein interaktives Befehlszeilen-Programm, welches eine einfache grafische Oberfläche in einer Konsolenumgebung simuliert. Die GUI ist komplex aufgebaut und mit der Tastatur über eine Terminal-Verbindung nur sehr umständlich zu bedienen. Ohne Befehlszeilen-Programm wird es außerdem erschwert automatisiert Regelsätze z.B. per Shellscript zu generieren. Linux-Administratoren bevorzugen zudem meistens normale Befehlszeilen-Tools.

Die Einstellmöglichkeiten der *SUSE-Firewall* sind ebenfalls sehr eingeschränkt. Es existiert ein rudimentäres Netzzonen-Modell, aufgeteilt nach interner, externer und demilitarisierter Zone, was jedoch im praktischen Einsatz am Leibniz Rechenzentrum keine Verwendung findet. Als Kriterium zur Konfiguration von Zonen wird lediglich das Netz-Interface angeboten, d.h. das Zonenmodell ist unbrauchbar sobald ein Netz-Interface mehrere unterschiedliche IP-Subnetze bedient.

Es werden von der *SUSE-Firewall* Standarddienste wie z.B. SSH direkt unterstüzt. Alle weiteren Dienste müssen manuell inklusive des Netzprotokolls und der verwendeten Ports eingegeben werden. Die Freigaben beziehen sich jeweils auf eine bestimmte Zone.

Die sich hinter der GUI befindliche Konfigurationsdatei `/etc/sysconfig/SUSEfirewall2` ist streng nach Variablen aufgebaut. Es existieren also Schlüssel-Wert-Paare, über die verschiedene Charakteristika wie z.B. Namen von Netz-Schnittstellen oder auch Listen von freigegebenen Ports eingestellt werden können.

Erweiterte *iptables*-Features werden ebenso wie benutzerdefinierte *iptables*-Regeln nicht unterstützt. Sie können weder in der GUI, noch in der Konfigurationsdatei selbst eingegeben werden. Es wird allerdings eine Funktion angeboten, die SUSE als benutzerdefinierte Regeln bezeichnet. Hier es ist aber nur möglich benutzerdefinierte Portfreigaben einzutragen.

Als weiteres Feature wird das Filtern von Broadcast-Paketen angeboten. Da diese aber häufig bereits von Routern verworfen werden, ist die Funktion in den meisten Fällen überflüssig [rfc99].

Als Logging-Funktion wird ermöglicht sowohl akzeptierte als auch abgelehnte Pakete im Syslog aufzuzeichnen. Als Filter ist dabei nur wählbar, ob alle Pakete, alle kritischen Pakete oder gar keine Pakete aufgezeichnet werden sollen. Was genau als kritisch angesehen wird ist nicht ersichtlich. In der Standardinstallation wird außerdem keine eigene Konfigurationsdatei für den Syslog-Dienst angeboten. SUSE Linux ist allerdings bereits im Auslieferungszustand so konfiguriert, dass alle iptables-Logeinträge in eine Logdatei gespeichert werden.

Der *iptables*-Regelsatz der *SUSE-Firewall* ist unübersichtlich und nur teilweise nachvollziehbar. Für alle verwendeten Zonen werden dort zunächst eigene Chains erstellt.

In Anhang 8.1 ist der *iptables*-Regelsatz der Standardkonfiguration abgedruckt.

3.2 Lrzpf - LRZ Personal Firewall

Aktuell wird am Leibniz Rechenzentrum ein selbst entwickeltes, rudimentäres Shellscript mit dem Namen *lrzpf* eingesetzt, welches ebenfalls als Frontend für die *iptables*-Firewall unter Linux dient. Die Konfiguration dieses Pakets erfolgt zum einen über vordefinierte Variablen, die per Texteditor angepasst werden. Außerdem ist es möglich direkt *iptables*-Befehle händisch in einer weiteren Konfigurationsdatei zu hinterlegen. In der Standardkonfiguration sind SSH, das Backupsystem TSM und der DHCP-Client als Dienste freigeschalten. Die Firewall ist stateful gehalten, jedoch nur für IPv4. Zum Zeitpunkt der Entwicklung war eine stateful IPv6-Firewall auf Basis von *iptables* noch nicht verfügbar, weshalb dies auch von der *lrzpf* nicht unterstützt wird.

Die *lrzpf* ist zwar sehr flexibel, jedoch ist die Anpassung an die Bedürfnisse der Administratoren hier relativ aufwendig. Es gibt keine hinterlegten Regelsätze, so dass es nur möglich ist neue Dienste durch manuelles Einfügen von *iptables*-Befehlen in den Regelsatz zu integrieren. Eine Konfiguration ist ohne grundlegende Kenntnisse über das native *iptables*-Frontend und die verwendeten Netzprotokolle nicht möglich. Eine gleichzeitige Pflege von IPv4 und IPv6 wird zusätzlich erschwert, da separate Regelsätze für beide Netzprotokolle erstellt werden müssen.

Das *lrzpf*-Paket ist außerdem schwer zu aktualisieren, da für die Konfiguration grundsätzlich Dateien, die zum Firewall-Paket selbst gehören, modifiziert werden müssen. Diese werden entweder überschrieben oder können nicht aktualisiert werden, da selektives Patchen bei manuell editierten Dateien automatisiert nur selten möglich ist.

4 Anforderungen an die Firewall

Aus den Unzulänglichkeiten der *lrzpf* und der *SUSE-Firewall* wurde in Zusammenarbeit mit den Betreuern sowie den zuständigen Desktop- und Server-Administratoren am Leibniz Rechenzentrum ein Katalog ausgearbeitet, der in groben Zügen beschreibt, welche Eigenschaften ein neues Firewall-Frontend erfüllen sollte.

Die neue Firewall für das Leibniz Rechenzentrum trägt den Projektnamen *lrzfw*.

4.1 Systemvoraussetzungen

Als Basis soll wiederum auf die Linux *iptables* gesetzt werden, da dies aktuell die einzig sinnvolle Lösung ist, unter Linux eine professionelle Firewall aufzubauen. Desweiteren gehören die *iptables* zur Standardausrüstung und sind in jeder verbreiteten Distribution enthalten.

Am LRZ werden hauptsächlich SUSE Linux Enterprise Server- (SLES) und openSUSE-Distributionen eingesetzt. Darüberhinaus gibt es auch einige Debian-Maschinen [deb]. Bei SLES und openSUSE sind noch immer viele ältere Versionen im Einsatz. Diese werden zwar noch vom Hersteller unterstützt, verwenden jedoch alte Versionen von *iptables*.

Die Minimalanforderung an das Betriebssystem wird auf SLES 10.0 bzw. SUSE/openSUSE 10.0 festgelegt. Es sind zwar noch einige SLES-Server mit Version 9 aktiv, diese werden jedoch demnächst zum größten Teil auf neuere Versionen aktualisiert. Der offizielle Hersteller-Support für Version 9 endet am 31. August 2011. [sleb]

Ab SLES 11 wird *iptables* 1.4.X verwendet. Dort steht bis inklusive Version 1.4.2 der recent-Match für IPv6 nicht zur Verfügung. Darauf aufbauende Features, wie z.B. das Logging von exzessiven Verbindungsanfragen, müssen deshalb deaktiviert werden.

iptables-Versionen älter als 1.3.0 werden von der *lrzfw* nicht unterstützt.

Das Frontend selbst sollte vollständig als Shellscript [bas] konzipiert sein und seinerseits ausschließlich Standardbefehle und -utilities verwenden, welche durch die Linux Standard Base (LSB, [lsb]) gefordert sind, und deshalb auf allen gängigen Distributionen automatisch zur Verfügung stehen.

4.2 Paketverwaltung

Die neue Firewall soll zukünftig direkt als RPM-Paket per zentralem Repository manuell oder automatisch auf die Server und Desktop-PCs verteilt werden. Updates erfolgen auf dem gleichen Weg per Autoupdater.

Als plattformunabhängiges Shellscript wird dazu nur ein einzelnes RPM-Paket (noarch) benötigt.

Für Debian soll darüberhinaus ein DEB-Paket angeboten werden.

In den Quellcode-Paketen sollen jeweils alle nötigen Daten enthalten sein um das Paket neu erstellen zu können, so dass spätere Updates einfach eingeflegt werden können.

Neben der Installation der *lrzfw* soll das Paket sich auch um die Deinstallation anderer bekannter Firewalls kümmern oder den Administrator darüber in Kenntnis setzen, dass noch andere Firewalls installiert sind, die in Konflikt zur *lrzfw* stehen.

Als praktisches Feature sollte es möglich sein per automatischem RPM/DEB-Update auch den Firewall-Regelsatz aktualisieren zu können. Bei der Installation neuer Dienste kann so direkt die Firewall-Konfiguration angepasst werden.

4.3 IPv4 und IPv6

Es müssen IPv4 und IPv6 mit Stateful Packet Inspection unterstützt werden. Beide Protokolle werden gleichwertig behandelt, d.h. alle Dienste, die über IPv4 erreichbar sind, sollen auch unter IPv6 erreichbar sein.

SLES 10 verwendet die *iptables*-Version 1.3.X, die für IPv6 noch kein state-Match anbietet. Aus diesem Grund muss die *lrzfw* IPv6 auf den alten Distributionen speziell ohne Stateful Packet Inspection behandeln. In diesem Fall soll ein Warnhinweis angezeigt werden, der den Administrator auf das entsprechende Problem hinweist.

Da auf einigen Servern und Desktop-Rechnern am LRZ das IPv6-Protokoll deaktiviert ist, muss die *lrzfw* entsprechend darauf reagieren und keinen IPv6-Regelsatz für das betreffende System erstellen, da dies zu Fehlern führt.

4.4 Konfiguration

Die Firewall-Konfiguration soll im Gegensatz zur *lrzpf* und der *SUSE-Firewall* nicht mehr ausschließlich über große Konfigurationsdateien erfolgen. Es sollte eine große Anzahl von Standard-Regelsätzen (Templates) zur Verfügung stehen, die den größten Teil der angebotenen Dienste abdecken und einfach an- bzw. abgeschalten werden können. Dabei ist es notwendig, dass durch einen einfachen Benutzereingriff bestimmte Standardwerte wie z.B. Portnummern modifiziert werden können.

Die Konfiguration soll so gestaltet sein, dass ein Administrator möglichst keine Dateien aus dem Installationspaket bearbeiten muss, um so eine einfache Möglichkeit für spätere automatisierte Updates zu haben. Es wäre bei einer Textdatei zwar möglich mit den Hilfsprogrammen `diff` und `patch` gewisse Updates einzuspielen. Kollidieren diese allerdings mit den manuellen Änderungen des Administrators, so ist ein Benutzereingriff nötig. Dies sollte möglichst vermieden werden.

Als hilfreiches Feature zur Abwehr von Angriffen oder fehlerhafter Bedienung soll es möglich sein auf einfache Weise eine IP-Blacklist zu pflegen, mit der bestimmten Hosts oder Netzen der Zugang anhand ihrer IP-Adressen gesperrt wird.

Das Zonen-Modell der *SUSE-Firewall* soll weiter ausgebaut werden, um so frei definierbare Zonen zu schaffen, für die separate Regelsätze aktiviert werden können. Als Kriterien für eine Zone sollen sowohl die Netz-Schnittstelle, als auch Absender- und Zieladresse des Pakets zur Verfügung stehen. Die Zonen-Konfiguration soll wie auch bei den Regelsätzen ohne Modifikation von mitgelieferten Dateien auskommen, so dass spätere automatisierte Updates noch möglich sind.

Außerdem sollte die Konfiguration konsolenbasiert sein. Eine GUI wird nicht benötigt.

4.5 Logging

Das Firewall-Paket soll das Logging von sicherheitsrelevanten Ereignissen unterstützen. Dabei ist unbedingt zu verhindern, dass exzessiv Logeinträge erstellt werden, wie es beim naiven Loggen von Netzpaketen leicht vorkommen kann.

Als weitere Anforderung sollen die Firewall-Logs automatisch in Log-Dateien umgeleitet werden, die wiederum vom automatischen Logrotate überwacht werden. So wird sowohl die temporäre Archivierung sichergestellt als auch verhindert, dass sich durch viele Logeinträge die Festplatte unnötig füllt und ein manuelles Eingreifen des Administrators nötig wird.

Die Syslog-Umleitung sowie der Logrotate soll direkt bei der Installation durch das *lrzfw*-Paket automatisch konfiguriert werden.

4.6 Regelsatz

Der finale Regelsatz in Form von *iptables*-Regeln soll leicht lesbar und verständlich sein. Das impliziert, dass Regeln in der Konfiguration leicht im fertigen *iptables*-Regelsatz wiederzufinden sind - und auch eine umgekehrte Zuordnung leicht möglich ist. Die Bezeichnungen von automatisch generierten Chains sollen so gewählt werden, dass sie eindeutig zuzuordnen sind.

4.7 Sonstiges

Es wird ein LSB-konformes init-Script benötigt um die Firewall beim Systemstart automatisch zu starten. Das Script soll mit `insserv` harmonieren, so dass die Firewall inklusive ihrer Abhängigkeiten korrekt im Runlevel eingetragen werden kann. Hier ist besonders die Reihenfolge beim Systemstart nötig. Die *lrzfw* muss dabei geladen werden, bevor die Netz-Schnittstellen initialisiert werden. Andernfalls entsteht ein Timeslot, in dem der Host dem Netz ungeschützt ausgesetzt ist.

4.8 Zusammenfassung

Die wichtigsten Anforderungen in der Übersicht:

Binary-Typ	Bash-Shellscript
init.d-Script (LSB)	ja
Regel-Templates	ja
High/Lowlevel	high
IPv4-Support	ja
IPv6-Support	ja
IPv4/v6 identisch	ja
Netzwerkzonen	ja
Eigene Zonen	ja
Blacklist-Funktion	ja
Minimale iptables-Version	-
RPM-Paket vorhanden	ja
DEB-Paket vorhanden	ja
Regelsatz übersichtlich	ja
Logging via syslogd	ja
Logging via ulogd	ja
Logging konfiguriert	ja
Konsole/GUI	konsole
Konfigurationsschema	dezentral
Konfigurationsart	-
Konfiguration maschinell gut zu bearbeiten?	ja
Dokumentation	>= gut

Abbildung 4.1: Übersicht über die Anforderungen an die lrzfw

5 Marktanalyse

Im Rahmen der Arbeit erfolgte zunächst eine Analyse der aktuellen Marktsituation sowie eine Suche nach *iptables*-Frontends, die den gegebenen Anforderungen möglichst nahe kommen. Dabei wurden speziell Tools gesucht, die über die Befehlszeile arbeiten.

Aufgrund der eher unübersichtlichen Natur der Opensource-Szene wurde für die Suche nach möglichen Kandidaten eine Vorauswahl der großen Linux-Distributoren SUSE und Ubuntu verwendet. Erfahrungsgemäß finden sich dort die ausgereiftetsten und auch verbreitetsten Opensource-Produkte. Darüberhinaus bieten die Distributoren den großen Vorteil, dass bereits betriebsfertige Programme als Pakete ausgeliefert werden.

Wegen der großen Zahl an potentiellen Kandidaten wird im Rahmen dieser Arbeit nur auf die vielversprechendsten darunter eingegangen. Alle Daten basieren dabei auf den Angaben des Herstellers sowie eigenen Tests.

5.1 Arno's IPTABLES Firewall Script

Das Arno's IPTABLES Firewall Script (v1.9.2l) [arn] ist eine Sammlung von Shellscripten, die als flexibles Frontend für die Linux *iptables* fungiert. Das Frontend wird von Ubuntu als Paket ausgeliefert. Die Scriptsammlung besteht aus einem Kern, der den eigentlichen Regelsatz aufbaut, einer Bibliotheksdatei mit einer großen Zahl von Hilfsfunktionen sowie einer größeren Zahl von Plugins, die vom Administrator in der Konfiguration gesteuert werden können. Dazu gehören Freigaben für einzelne Dienste ebenso wie generische Plugins, die z.B. Bruteforce-Attacken auf SSH blockieren sollen. Alle Plugins können einzeln per separater Konfigurationsdatei ein- und ausgeschalten werden.

Die Hauptkonfigurationsdatei von Arno's IPTABLES Firewall besteht aus einer großen Anzahl von gut dokumentierten Schlüssel-Wert-Paaren, über die die grundlegenden Funktionen der Firewall gesteuert werden. Dazu gehören insbesondere auch Portfreigaben, die in einem speziellen Syntax formatiert als Wert angegeben werden.

Die Firewall unterstützt IPv4 und experimentell auch IPv6. Es gibt ein einfaches Zonenmodell mit einer Untergliederung in eine externe Zone (INET), eine interne Zone (LAN) und eine demilitarisierte Zone (DMZ), die jeweils anhand der Netz-Schnittstelle(n) und IP-Subnetzen definiert werden können.

Das Logging der Firewall erfolgt immer für das LOG-Target, also über syslog. Es wird vom Paket aber direkt eine Konfigurationsdatei für den syslogd mitgeliefert, so dass standardmäßig auch in eine separate Logdatei aufgezeichnet wird. Der Kern selbst bietet eine größere Menge von Logging-Funktionen, die einzeln über die zentrale Konfigurationsdatei gesteuert werden können. Dazu gehören einfache Regeln wie das Loggen von ungültigen Paketen, aber auch erweiterte Heuristiken um Portscans zu Detektieren. Zusätzlich lässt sich die Firewall so einrichten, dass alle abgewiesenen Pakete separat geloggt werden.

Weitere Features der Firewall sind NAT, Traffic Shaping, Traffic Accounting, Mangling und viele weitere. Zusammen mit der Firewall wird auch ein LSB-kompatibles Init-Script ausgeliefert.

Als zukünftige Firewall für das Leibniz Rechenzentrum scheidet Arno's IPTABLES Firewall Script aus folgenden Gründen aus:

- IPv4 und IPv6 nicht gleichberechtigt.
- Nur statisches Zonen-Modell.
- Alle Regelsätze befinden sich in einer zentralen Konfigurationsdatei.
- Automatisierte Bearbeitung der Regelsätze schwierig.
- Keine Regel-Templates für einzelne Dienste.
- Sehr viele Features, aber Konfiguration aufwendig.

5.2 ferm

Das *iptables*-Frontend *ferm* (v2.0.7) [fer] ist ein in der Programmiersprache Perl entwickeltes Firewall-Script.

ferm unterscheidet sich grundsätzlich von Arno's Firewall. Das Firewall-Script dient vielmehr als alternatives Frontend oder auch Ergänzung zu den normalen *iptables*-Befehlszeilen-Tools. Es wird eine neue, proprietäre Syntax definiert, mit der *iptables*-Regeln eleganter formuliert werden können. Das Perlscript dient als Parser und übersetzt die Konfigurationsdateien in *iptables*-Befehlszeilenaufrufe.

Der Ansatz ist insofern interessant als dass *ferm* die Entwicklung von Regelsätzen wesentlich eleganter und übersichtlicher macht als einfache Shellscripte, die direkt *iptables*-Befehle enthalten. Der Aufbau der Konfiguration in hierarchischen Blöcken ermöglicht es, Regelsätze sehr kurz zu fassen und ist obendrein aufgrund ihres Low-Level-Charakters überaus flexibel. Es ist so möglich alle von *iptables* unterstützten Features effektiv zu nutzen. Die Konfigurationsdateien können darüberhinaus auch (parametrisierte) Makros enthalten. Mit ihnen können sowohl einzelne Konstanten wie z.B. statische IP-Adressen als auch komplette Filtersätze erstellt werden, die anschließend in den Regelsatz eingebunden werden. Desweiteren werden auch Features wie das Einbinden von externen Dateien oder einfache Conditionals unterstützt.

Der prinzipielle Aufbau der Konfiguration erfordert vom Administrator allerdings wie auch bei *iptables*-Befehlen tiefergehendes Wissen über das Netz und die verwendeten Protokolle. Das Frontend hat somit eher den Vorteil einer übersichtlicheren Darstellung und erleichtert somit das Erstellen eines Regelsatzes. Will man den vollen Funktionsumfang der *iptables* ausschöpfen, so ist die Konfiguration aber nicht mehr zwingend übersichtlicher als reine *iptables*-Befehle.

Die gesamte Konfiguration befindet sich nach wie vor hauptsätzlich in einer zentralen Konfigurationsdatei. Dies wird zwar dadurch etwas entschärft, dass externe Dateien dort eingebunden werden können, ändert jedoch am grundsätzlichen Charakter nichts. Das Problem von Updates über den Paketmanager bleibt somit erhalten.

Eine einfache Beispielkonfiguration aus dem Paket sieht wie folgt aus:

```
# -*- shell-script -*-
#
# Ferm example script
#
```

```
# Firewall configuration for a web and SMTP server.
#
# Author: Max Kellermann <max@duempel.org>
#

@def $NET_TRUSTED = 195.135.144.144/28;

table filter {
    chain INPUT {
        policy DROP;

        # connection tracking
        mod state state INVALID DROP;
        mod state state (ESTABLISHED RELATED) ACCEPT;

        # allow local connections
        interface lo ACCEPT;

        # respond to ping
        proto icmp icmp-type echo-request ACCEPT;

        # remote administration from the company network
        saddr $NET_TRUSTED proto tcp dport ssh ACCEPT;

        # our services to the world
        proto tcp dport (http https smtp) ACCEPT;

        # the rest is dropped by the above policy
    }

    # outgoing connections are not limited
    chain OUTPUT policy ACCEPT;

    # this is not a router
    chain FORWARD policy DROP;
}
```

Als zukünftige Firewall für das Leibniz Rechenzentrum scheidet *ferm* aus folgenden Gründen aus:

- Perl wird zwingend benötigt.
- Alle Regelsätze befinden sich in einer zentralen Konfigurationsdatei.
- Konfiguration sehr low-level, wenig Abstraktion, keine fertigen Regel-Templates.
- Detailliertes Wissen über *iptables* zur Konfiguration nötig.
- Automatisierte Bearbeitung der Regelsätze schwierig.

5.3 FireHOL

FireHOL (R5 v1.273) [fir] ist ein Shellscript, das wie *ferm* eine eigene Syntax zur Konfiguration von *iptables* definiert. Darauf aufbauend werden Funktionen angeboten, mit denen die Konfiguration komplexer Regelsätze erleichtert wird. Die Konfigurationsdatei ist selbst ein Shellscript, welches in einer vorgegebenen Anordnung Hilfsfunktionen aufruft. Der Vorteil bei dem Aufbau ist, dass an jeder Stelle in der Konfiguration benutzerdefinierter Shellscript-Programmcode eingesetzt werden kann. So ist es möglich die Filtersätze auch anhand vieler weiterer Kriterien, z.B. über Conditionals oder weitere Hilfsfunktionen, flexibler zu gestalten.

Die Konfiguration selbst ist nach einem Zonen-Prinzip aufgebaut. Es wird anhand bestimmter Kriterien wie Netz-Schnittstellen, IP-Adressen, etc. eine Zone definiert, für die dann einzeln Dienste freigegeben werden.

Die Zugriffssteuerung erfolgt separat für Client und Server. Dazu werden Dienste definiert, die eingehend oder ausgehend freigegeben werden können. Neben den bereits vordefinierten Diensten ist es möglich, zusätzlich durch Erstellen von Dienstdefinitionsdateien weitere Dienste verfügbar zu machen. Dazu muss angegeben werden welche Ports der Dienst ein- und ausgehend nutzt. Es können auch Kernel-Module darüber eingebunden werden.

Neben Zugriffssteuerung wird auch die Reglementierung von IP-Forwarding direkt unterstützt. Sie wird ebenfalls über freigegebene Dienste konfiguriert.

Generell ist die Konfiguration aber viel Schreibarbeit und konzentriert sich auf eine einzige zentrale Konfigurationsdatei, kann aber auch durch Importieren von anderen Dateien aufgespalten werden. Ein weiteres erhebliches Problem ist, dass *FireHOL* kein IPv6 unterstützt.

Eine einfache Beispielkonfiguration aus dem Paket sieht wie folgt aus:

```
version 5

    # The network of our eth0 LAN.
    home_ips="195.97.5.192/28"

    interface eth0 home src "${home_ips}"
        server "dns␣ftp␣samba␣squid␣dhcp␣http␣ssh␣icmp" accept
        client "samba␣icmp"             accept

    interface ppp+ internet src not "${home_ips}␣
    ${UNROUTABLE_IPS}"
        server "smtp␣http␣ftp"  accept
        client all      accept

    router home2internet inface eth0 outface ppp+
        route all accept
```

Als zukünftige Firewall für das Leibniz Rechenzentrum scheidet *FireHOL* aus folgenden Gründen aus:

- Alle Regelsätze befinden sich in einer zentralen Konfigurationsdatei.
- Wenige High-Level-Funktionen verfügbar.
- Automatisierte Bearbeitung der Regelsätze schwierig.
- IPv6 wird nicht unterstützt.

5.4 shorewall

shorewall (v4.4.12.2) [sho] ist ebenfalls ein Shellscript, mit dessen Hilfe eine auf *iptables* basierende Firewall konfiguriert werden kann. Der Regel-Compiler der Firewall ist allerdings ein Perlscript und erfordert eine Perl-Installation auf dem lokalen System.

Es werden verschiedene Konfigurationsdateien angeboten, mit denen sowohl Netzzonen als auch zugehörige Regelsätze definiert werden. Als weitere Features werden auch Network Address Translation, IP-Forwarding und Traffic Shaping angeboten.

shorewall stellt im Auslieferungszustand eine große Anzahl von einfachen Regel-Templates für gängige Netzprotokolle zur Verfügung. Diese enthalten meist nur das verwendete Protokoll (UDP/TCP) und den zugehörigen Port. Es sind jedoch auch weitere Einschränkungen möglich, z.B. erlaubte Absender- und Ziel-Adresse, eine Ratenlimitierung und sogar eine Beschränkung auf lokale Benutzer-IDs und -Gruppen sowie Prozesse des lokalen Systems. Letzteres ist für Server allerdings irrelevant. Regel-Templates können Platzhalter enthalten, die in der Regelkonfiguration überschrieben werden. So sind etwa die Targets der Templates meist Platzhalter und werden in der Konfiguration später z.B. durch ACCEPT oder DROP überschrieben.

Die Konfiguration der Zonen ist sehr komplex. Zunächst werden die Zonen mit einem benutzerdefinierten Namen und einem Typ in der Konfigurationsdatei `zones` aufgelistet und anschließend über die Dateien `interfaces` und `hosts` anhand von Netz-Schnittstellen und/oder IP-Adressen und -Subnetzen beschrieben. Dabei sind auch weitere Kriterien wie IPSec-Verbindungen möglich. Zusätzlich können jeweils weitergehende Eigenschaften von Netz-Schnittstellen und Hosts beschrieben werden. Dazu gehören z.B. DHCP-Unterstützung, MSS, ARP-Proxies, Ethernet-Bridging und viele mehr. Die Anzahl der Zonen ist durch die *shorewall* nicht limitiert. Die Zonen-Funktion ist insgesamt zwar sehr mächtig, aber aufwendig zu konfigurieren.

In einer zentralen Konfigurationsdatei `rules` können anschließend Regelsätze für bestimmte Netzzonen aktiviert werden. Die Konfigurationsdatei ist wie alle anderen Dateien von *shorewall* eine Plaintext-Datei, die durch Whitespace in einer Tabellenform organisiert ist.

Eine separate Blacklist-Datei für IP-Adressen und -Subnetze wird ebenfalls angeboten.

Insgesamt besteht die Konfiguration der *shorewall* aus mehr als drei Dutzend einzelnen Konfigurationsdateien, die jeweils andere Funktionen verwalten, aber teilweise einen gemeinsamen Namensraum (z.B. für Zonen) verwenden. In der Dokumentation der Firewall wird jede einzelne Datei durch eine eigene sehr ausführliche man-Page beschrieben, die überwiegend jeweils mehrere DIN-A4-Seiten lang sind. Die Einarbeitung ist dadurch sehr zeitintensiv. Durch die Vielzahl an Funktionen und deren speziellen Syntax in den Konfigurationsdateien ist ein regelmäßiges Nachschlagen in den man-Pages unausweichlich.

shorewall unterstützt seit Version 4.2.4 das Protokoll IPv6. Als Mindestvoraussetzung wird allerdings ein Linux-Kernel in Version 2.6.24 oder neuer sowie eine *iptables*-Version 1.4.0 oder neuer angegeben. Die IPv6-Version nennt sich *shorewall6* und arbeitet unabhängig von *shorewall*. Es existiert keine gemeinsame Konfiguration.

Als zukünftige Firewall für das Leibniz Rechenzentrum scheidet *shorewall* aus mehreren Gründen aus:

- Perl wird zwingend benötigt.
- Alle Regelsätze befinden sich in einer einzelnen Konfigurationsdatei.

- Die Konfiguration ist unübersichtlich und kompliziert.
- Automatisierte Bearbeitung der Regelsätze schwierig.
- Funktionsumfang übersteigt die Anforderungen des LRZ enorm, dadurch Betrieb unnötig aufwendig.
- IPv6 wird erst ab *iptables* 1.4.0 unterstützt.
- IPv4- und IPv6-Konfiguration komplett unabhängig voneinander.

5.5 Vuurmuur

Vuurmuur (v0.7) [vuu] ist ebenfalls ein Firewall-Frontend für die *iptables* und bietet eine Konfiguration sowohl über eine konsolenbasierte GUI als auch über Befehlszeilen-Tools (CLI). Ich beschränke mich im Nachfolgenden auf die Befehlszeilen-Version.

Alle Komponenten von *Vuurmuur* wurden in der Programmiersprache C entwickelt. Dadurch ist es nötig plattformspezifische Pakete für unterschiedliche Server anzubieten. Ein Programmpaket für Ubuntu liegt aktuell vor, für SUSE leider nicht.

Das Zonen-Modell ist ähnlich zu dem von *shorewall*, jedoch mit weniger Features. Bei *Vuurmuur* werden zunächst sogenannte Interfaces erstellt, denen ein physisches Netz-Interface sowie eine statische IP-Adresse bzw. ein DHCP-enabled-Flag zugeordnet wird. Die Interfaces tragen benutzerdefinierte Namen. Im nächsten Schritt werden Zonen definiert. Diese bestehen aus sogenannten Netzen, die wiederum aus einzelnen Hosts bzw. Gruppen von Hosts zusammengesetzt sind. Einem Netz können ein oder mehrere Interfaces zugeordnet werden.

Die eigentlichen Firewall-Regeln werden schließlich global zugeordnet und beschreiben die Kommunikation zwischen verschiedenen Zonen. Soll beispielsweise ein lokaler Dienst für andere Hosts im Internet angeboten werden, so wird im Allgemeinen eine lokale Zone und eine Internet-Zone benötigt, für die der Dienst freigegeben wird.

Vuurmuur liefert eine große Anzahl von Regel-Templates für verschiedene Standarddienste aus. Die Templates enthalten die Portnummern für TCP und UDP sowie einige zusätzliche Informationen zu ICMP und GRE. Darüberhinaus wird festgelegt, ob Broadcast-Kommunikation für den Dienst erlaubt ist. Leider können die vorgegebenen Standardports im Regelsatz nicht modifiziert werden, was den Einsatz benutzerdefinierter Portnummern erschwert.

IPv6 wird leider nicht unterstützt.

Die Dokumentation von *Vuurmuur* weist schwere Mängel auf. Auf der Webseite des Herstellers wird ein Wiki angeboten, das jedoch sehr kurz gefasst ist und allenfalls einen kurzen, aber unvollständigen Überblick über die Firewall bietet. Die man-Pages sind ebenfalls sehr knapp gefasst und beschreiben lediglich die mitgelieferten Hilfsprogramme und deren Befehlszeilen-Argumente. Eine genaue Beschreibung der Konfigurationsdateien sucht man leider vergeblich.

Eine Konfiguration ist so nur über die grafische Schnittstelle empfehlenswert. Die Bedienbarkeit ist allerdings subjektiv noch bedeutend schlechter als bei der *SUSE-Firewall*.

Als zukünftige Firewall für das Leibniz Rechenzentrum scheidet *Vuurmuur* aus mehreren Gründen aus:

- Plattformspezifische Pakete nötig.
- Alle Regelsätze befinden sich in einer einzelnen Konfigurationsdatei.

- Sehr schlechte Dokumentation.
- Die Konfiguration ist unübersichtlich und kompliziert.
- Automatisierte Bearbeitung der Regelsätze schwierig.
- IPv6 wird nicht unterstützt.

5.6 Übersicht

	ARNO	ferm	firehol	shorewall	Vuurmuur	SuSE-Firewall	lrzpf
Version	1.9.2l	2.0.7	R5 v1.273	4.4.12.2	0.7	-	-
Lizenz	GPLv2	GPLv2	GPL	GPLv2	GPLv2	GPLv2	proprietär
Binary-Typ	Bash	Perl	Bash	Bash+Perl	Native	Bash+Native	Bash
Persistant	nein	nein	nein	nein	ja	nein	nein
init.d-Script (LSB)	ja	nein	nein	ja	ja	ja	ja
Regel-Templates	ja	nein	nein	ja	ja	ja	nein
High/Lowlevel	high	low	low	high	high	high	low
IPv4-Support	ja	ja	ja	ja	ja	ja	ja
IPv6-Support	experimentell	ja	nein	ab ipt 1.4.0	nein	ja	ja (stateless)
IPv4/v6 identisch	nein	nein	nein	nein	nein	ja	nein
Netzwerkzonen	ja	nein	nein	ja	ja	ja	nein
Eigene Zonen	nein	nein	nein	ja	ja	nein	nein
Blacklist-Funktion	ja	nein	ja	ja	ja	nein	nein
Minimale iptables-Version	-	-	1.2.4	1.4.0	-	-	-
Logging via syslogd	ja	ja	ja	ja	ja	ja	ja
Logging via ulogd	nein	ja	ja	ja	nein	nein	ja
Logging konfiguriert	nein	nein	nein	nein	ja	ja	nein
Konsole/GUI	Konsole	Konsole	Konsole	Konsole	GUI	GUI	Konsole
Konfigurationsschema	zentral	zentral	zentral	zentral	zentral	zentral	zentral
Konfigurationsart	key/value	iptables	iptables abstrakt	Text-Tabellen	Pseudo-Syntax	key/value	iptables
Konfiguration maschinell gut zu bearbeiten?	nein	nein	nein	nein	nein	nein	nein
Dokumentation	gut	gut	gut	sehr gut	sehr schlecht	schlecht	nicht vorhanden

Abbildung 5.1: Übersicht Marktanalyse

5.7 Fazit

Das Feld der Kandidaten teilt sich im Groben in zwei Lager: Eines mit Highlevel-Funktionen und ein reines Lowlevel-Lager, welches in erster Linie eine Vereinfachung oder Umgestaltung der *iptables*-Syntax bietet.

Die Highlevel-Firewalls bieten zwar ein großes Maß an Funktionen, haben jedoch alle größere Probleme bei der Konfiguration und sind nicht unmittelbar auf die Anforderungen des LRZ angepasst - sowohl was Versionsabhängigkeiten angeht als auch was die benötigten Features betrifft. Der größte Nachteil der geprüften Firewalls liegt in der Tatsache, dass die Konfiguration der Firewall-Regeln fast ausschließlich über das Bearbeiten von einzelnen Textdateien gelöst ist. Da am Leibniz Rechenzentrum eine große Zahl von Endgeräten mit der Firewall ausgestattet werden soll, bedeutet dies aber einen großen Aufwand: Textdateien lassen sich nicht immer automatisiert bearbeiten. In einem homogenen Umfeld von Servern und Desktop-PCs, auf denen größtenteils die gleichen Dienste laufen, kann dies über das vorherige Erstellen von Konfigurationsdateien und anschließendem Ausrollen auf die Hosts geregelt werden. Ist die Umgebung aber heterogen (also werden viele verschiedene Dienste angeboten), so steigt der Aufwand für die Anpassung der Konfiguration erheblich. Im Idealfall sollte eine Lösung gefunden werden, die beispielsweise mithilfe von Shellscripten automatisch an die Hosts angepasst und auch nachträglich verändert werden kann. Das ist jedoch im Allgemeinen bei zentralen Regeldateien schwierig.

Die Lowlevel-Firewalls sind für das manuelle Erstellen von Regelsätzen durchaus interessant. Ziel der neuen Firewall am LRZ ist es aber nicht primär das Erstellen von Regel-Templates zu vereinfachen, sondern das Konstruieren von Regelsätzen aus einzelnen Templates so einfach wie möglich zu gestalten. Zum Erstellen von Regel-Templates ist nach wie vor Fachwissen über die *iptables* und die verwendeten Dienste nötig. So wäre es zwar denkbar gewesen, die Syntax einer der Lowlevel-Firewalls zu adaptieren und als Basis für die *lrzfw* zu verwenden. Dies hätte allerdings die Komplexität des Programms vergrößert und den Bedienkomfort nur unwesentlich erhöht. Deshalb wurde auf diese Option verzichtet.

Insgesamt ist festzuhalten, dass keine der vorgestellten Firewalls so recht auf die Anforderungen des Leibniz Rechenzentrums passt. Der Portierungsaufwand wäre gerade bei den Highlevel-Firewalls recht groß und brächte zusätzlich das Problem mit sich, dass zukünftige Updates seitens des Herstellers aufwendig in die *lrzfw* eingepflegt werden müssen, was mit einem zusätzlichen Wartungsaufwand verbunden ist.

Aus diesem Grund wurde die Entwicklung einer eigenen Lösung als beste Option befunden. Sie sollte sowohl einfach in der Bedienung und Konfiguration als auch leistungsfähig im Betrieb sein.

6 Lrzfw - LRZ-Firewall

6.1 Architektur

Um die speziellen Anforderungen seitens des Leibniz Rechenzentrums zu erfüllen, ist ein komplett neuer Ansatz nötig, der sich grundlegend von dem der *lrzpf* und der *SUSE-Firewall* unterscheidet.

Die Architektur der *lrzfw* wird in Abbildung 6.1 dargestellt.

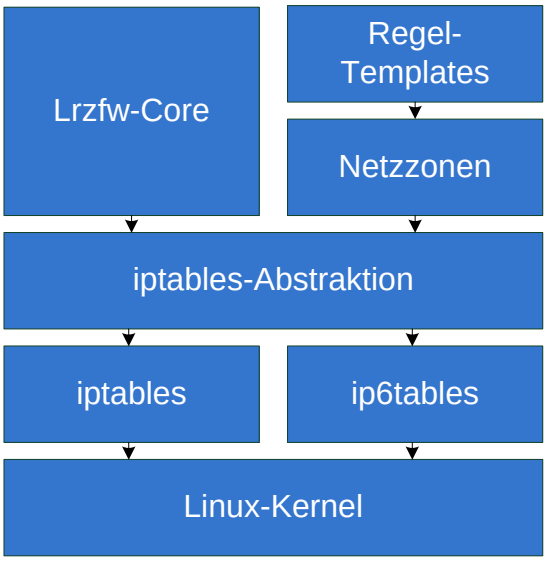

Abbildung 6.1: Architektur der lrzfw-Abstraktionsschicht

6.2 Abstraktionsschicht

6.2.1 Protokoll-Abstraktion

Automatische Protokollerkennung

Die gleichberechtigte Unterstützung von IPv4 und IPv6 wird über eine Abstraktionsschicht realisiert. Dafür wird der *iptables*-Befehl durch eine Shellscript-Subroutine überschrieben. Diese wertet die Argumente aus und entscheidet anhand derer automatisch, ob sich ein Aufruf auf IPv4, IPv6 oder auf beide Protokolle bezieht. Anschließend werden die entsprechenden *iptables*-Befehle ausgeführt. Dieser Mechanismus wird im Folgenden als automatische Protokollerkennung bezeichnet.

Die Liste der für diese Entscheidung relevanten Argumente besteht aus:

- `-s|--source`: Quelladresse
- `-d|--destination`: Zieladresse
- `-m|--match`: Name des Matches. Es gibt Matches die exklusiv IPv4 oder IPv6 sind, z.B. ttl (Time to Live) bzw. hl (Hop limit).
- `-p|--protocol`: Name des Protokolls. Hier sind speziell ICMP (Internet Control Message Protocol) und ICMPv6 von Bedeutung.

Ist ein Aufruf für beide Protokolle zulässig, so werden zwei *iptables*-Aufrufe generiert, ansonsten nur einer. Ist keines von beiden Protokollen zulässig (z.B. weil eine IPv4-Quelladresse und ein IPv6-Match angegeben wurde), so wird ein Fehler angezeigt und der Aufruf abgebrochen.

Wird z.B. der Befehl

```
iptables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

ausgeführt, so mappt die Subroutine dies auf die beiden *iptables*-Befehle

```
/sbin/iptables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
/sbin/ip6tables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

Bei einem Befehl im Stile von

```
iptables -A INPUT -s 192.168.0.0/16 -j ACCEPT
```

wird automatisch erkannt, dass sich das Argument `-s` nur auf IPv4 bezieht, und entsprechend gemappt auf:

```
/sbin/iptables -A INPUT -s 192.168.0.0/16 -j ACCEPT
```

Manuelle Protokollwahl

Es kann außerdem auch manuell mit den Argumenten `-4|--ipv4` und `-6|--ipv6` darauf Einfluss genommen werden, für welches Protokoll der Aufruf ausgewertet wird. Widerspricht das Ergebnis der automatischen Protokollerkennung aus 6.2.1 der manuellen Protokollwahl via `-4|--ipv4` oder `-6|--ipv6`, so wird der Aufruf ignoriert und Erfolg zurückgegeben.

Dies ist ein gewünschter Effekt, da die manuelle Protokollerkennung im technischen Teil verwendet wird, um optionale Funktionen, z.B. aufgrund fehlender Matches bei alten *iptables*-Versionen, stillschweigend durch automatisiertes Anfügen der jeweiligen Argumente zu deaktivieren (siehe 6.5.1). Dies sorgt für einen kürzeren Quellcode und erleichtert die Entwicklungsarbeit.

Widerspricht sich die manuelle Protokollwahl allerdings selbst (z.B. indem `--ipv4` und `--ipv6` gleichzeitig verwendet werden), so wird ein Fehler angezeigt und der Aufruf abgebrochen.

6.2.2 Logging-Abstraktion

Die Abstraktions-Subroutine überschreibt automatisch alle Logging-Prefixes, um für die gesamte Firewall eine einheitliche Konfiguration für rsyslogd bzw. ulogd verwenden zu können. Dazu wird jedem Logging-Präfix die Zeichenkette `"lrzfw:␣"` vorangestellt.

Alle weiteren Logging-Argumente werden jeweils soweit möglich auf das gewählte Backend gemappt. Dabei sind `--log-level`, `--log-tcp-sequence`, `--log-tcp-options`, `--log-ip-options` und `--log-uid` nur bei `LOG` möglich, sowie `--ulog-nlgroup`, `--nflog-group`, `--ulog-cprange`, `--nflog-range`, `--ulog-qthreshold` und `--nflog-threshold` nur bei `ULOG` oder `NFLOG`.

Wird ein durch das Logging-Backend nicht unterstütztes Argument dennoch verwendet, so wird eine Warnung ausgegeben und das Argument automatisch entfernt. Der Vorgang wird anschließend normal fortgesetzt.

6.3 LRZ-Core, Netzzonen und Regel-Templates

6.3.1 Konfigurations-Modell

Das Debian-Modell

Um große textbasierte Konfigurationsdateien zu vermeiden, wurde für die *lrzfw* ein Konfigurations-Modell aus Debian adaptiert. Es kommt sowohl beim Zonen-Modell in Kapitel 6.3.2, als auch beim Regelsatz in Kapitel 6.3.3 zur Anwendung.

Das Grundkonzept ist, dass der Administrator Systemdienste konfigurieren kann, indem er neue Konfigurationsdateien hinzufügt oder entfernt, anstatt zentrale Konfigurationsdateien zu editieren. Das Dateisystem dient dabei als eine Art Indexdatei. Beim Laden des Systemdienstes werden entsprechende Konfigurationsverzeichnisse durchsucht und alle vorhandenen Dateien in alphabetischer Reihenfolge ausgewertet.

Als weitere Abstraktion werden symbolische Links verwendet. Anstatt die einzelnen Konfigurationsdateien direkt in das entsprechende Konfigurationsverzeichnis zu kopieren, ist es oftmals sinnvoll, sie an einem anderen Ort im Dateisystem zu pflegen und anschließend durch einen symbolischen Link dem Systemdienst zur Verfügung zu stellen. Durch diese Vorgehensweise entsteht ein N-zu-1-Mapping auf die eigentlichen Konfigurationsdateien. Es ist also auch möglich eine einzelne Konfigurationsdatei mehrfach einzubinden, indem mehrere symbolische Links mit unterschiedlichen Dateinamen auf die Datei verweisen. Da zum Festlegen der Auswertungsreihenfolge der Dateien in diesem Falle die Dateinamen der symbolischen Links verwendet werden, kann auch darauf Einfluss genommen werden, indem lediglich der Link umbenannt wird. Dazu wird die lexikalische Ordnung verwendet.

Als Konvention wird in der Praxis oft ein Zahlencode vor den eigentlichen Dateinamen gestellt, z.B. `000-datei.conf`.

Ein weiterer Vorteil ergibt sich beim Löschen einer Konfigurationsdatei. Es reicht dazu aus, den symbolischen Link zu entfernen. Die eigentliche Datei muss weder gelöscht, noch umbenannt oder verschoben werden, und kann zu einem späteren Zeitpunkt wieder neu eingebunden werden.

In Kombination mit Debian-Paketen, die zur Installation von zusätzlicher Software genutzt werden, ergibt sich zusätzlich der Vorteil, dass Konfigurationsdateien bei einer Installation durch den Paketmanager erstellt werden können. Viele Pakete erstellen automatisch einen symbolischen Link und starten den betreffenden Systemdienst neu, so dass die neue Konfiguration sofort aktiv ist. Beim Update auf eine neuere Paketversion kann die Konfigurationsdatei einfach ausgetauscht werden, da meist keine manuellen Modifikationen daran vorgenommen wurden. Gleiches gilt bei der Deinstallation eines Pakets, bei der nur die Konfigurationsdatei und der symbolische Link entfernt werden müssen.

Das Modell ist somit sehr gut für die automatisierte Bearbeitung geeignet.

Beispiel: Apache Webserver

Das Debian-Modell aus Kapitel 6.3.1 wird im Folgenden am Beispiel des Apache Webservers [apaa] in Version 2.2 illustriert.

Das zentrale Konfigurationsverzeichnis des Servers unter Debian ist `/etc/apache2`. Dort liegt die Konfigurationsdatei *apache2.conf*, welche von Apache beim Startvorgang ausgewertet wird. In dieser Datei sind im Auslieferungszustand mehrere `Include`-Direktiven enthalten:

```
[..]
# Include module configuration:
Include /etc/apache2/mods-enabled/*.load
Include /etc/apache2/mods-enabled/*.conf

[..]
# Include generic snippets of statements
Include /etc/apache2/conf.d/

# Include the virtual host configurations:
Include /etc/apache2/sites-enabled/
```

Die `Include`-Direktive bindet mehrere Konfigurationsdateien in alphabetischer Reihenfolge in die Apache-Konfiguration ein. Sie erlaubt zudem die Verwendung von Platzhaltern bzw. Wildcards. [apab]

```
Include /etc/apache2/sites-enabled/
```

bindet beispielsweise alle Dateien im Verzeichnis `/etc/apache2/sites-enabled/` ein.

Zusätzlich zu den Verzeichnissen `mods-enabled` und `sites-enabled` werden die Verzeichnisse `mods-available` und `sites-available` bei der Installation automatisch erstellt. Dort befinden sich die eigentlichen Konfigurationsdateien zum Laden von Apache-Modulen und zum Konfigurieren von einzelnen durch den Webserver angebotenen Webseiten.

In `mods-enabled` und `sites-enabled` befinden sich nur symbolische Links, die auf Dateien in `mods-available` und `sites-available` verweisen.

Soll nun eine neue Webseite vom Server angeboten werden, so wird die entsprechende Konfiguration in `sites-available` angelegt und durch einen symbolischen Link in `sites-enabled` eingebunden.

6.3.2 Zonen-Modell

Die *lrzfw* implementiert ein generisches Zonen-Modell. Es ist möglich eine beliebige Anzahl von Netzzonen zu definieren und anhand vorgegebener Filterattribute zu beschreiben. Jede Zone besitzt einen eigenen Regelsatz.

Zusätzlich bietet die *lrzfw* eine globale Zone, die für den gesamten Host gilt, und auf die automatisch alle empfangenen Datenpakete matchen.

Das Zonen-Modell ist abstrakt gestaltet und kann sowohl genutzt werden um ganze Netze zu beschreiben, als auch einzelne Hosts (Host-Zonen).

Sollten sich Zonen gegenseitig überlappen, so sind besondere Maßnahmen zu treffen. Ein Datenpaket kann, abgesehen von der globalen Zone, immer nur auf eine einzelne Zone matchen! Gibt es beispielsweise eine Zone, die anhand des IP-Subnetzes 192.168.0.0/16 beschrieben wird, und eine Zone, die nur den Host mit der IP-Adresse 192.168.0.1 innerhalb dieses Subnetzes beschreibt, so ist auf die Reihenfolge der Zonendefinition zu achten. Wird die Subnetz-Zone zuerst ausgewertet, so wird dadurch die Host-Zone implizit deaktiviert. Andererseits gelten alle Regeln der Subnetz-Zone dann nicht für den Host 192.168.0.1 und müssen ggf. auch manuell in der Host-Zone eingefügt bzw. dupliziert werden!

Zonendefinition

Das erweiterte Zonen-Modell der *lrzfw* baut auf dem Debian-Modell aus Kapitel 6.3.1 auf. Es existiert ein Verzeichnis `/etc/lrzfw/zones`, in dem der Administrator eine beliebige Anzahl von `*.zone`-Dateien hinterlegen kann, die anhand bestimmter Kriterien eingehende Netzpakete entsprechend den dieser Zone zugewiesenen Regelsätze behandelt. Die Zonen-Dateien sind nach einem einfachen Schema aufgebaut. Pro Zeile werden mehrere per logischem AND verknüpfte Filterbedingungen, durch Whitespace (Leerzeichen, Tabulator) getrennt, hintereinander aufgeführt. Jede weitere Zeile kann weitere Filterausdrücke enthalten, die dann jeweils per logischem OR angehängt werden. Ein Paket kann jeweils nur auf eine einzige Zone matchen. Zonen dürfen sich deshalb nicht überlappen bzw. müssen in geeigneter Reihenfolge ausgewertet werden. Desweiteren darf pro Zeile nur ein Filterausdruck gleichen Typs verwendet werden. Die *lrzfw* nimmt einfache Plausibilitätsprüfungen auf den einzelnen Zeilen vor. Dabei wird geprüft, ob Filterausdrücke mehrfach vorkommen oder ungültig sind. Beim Erstellen der *iptables*-Regel wird außerdem implizit durch die Abstraktionsschicht geprüft, ob sich Filterausdrücke in ihrer IP-Version widersprechen. Leere Zeilen werden ignoriert, einzeilige Kommentare beginnen mit `#`. Bei fehlerhafter Syntax oder ungültigen Filterausdrücken wird der Ladevorgang mit Fehler abgebrochen.

Der Administrator kann auf die Auswertungsreihenfolge der Zonen Einfluss nehmen, indem er dem Dateinamen numerische Präfixe beifügt, z.B. `100-mwn.zone`. Die *lrzfw* selbst macht keine eigenen Annahmen über das Nummerierungsschema. Es wird jedoch empfohlen, dass Subnetze mit einer großen Zahl und einzelne Hosts mit einer kleinen Zahl versehen werden um ein mögliches unerwünschtes Überschatten zu verhindern.
Beispiel:

```
000-einzelner_host.zone
100-subnetz.zone
200-grosses_subnetz.zone
```

Als Filterausdrücke stehen aktuell zur Verfügung:

- **src**:addr - Quelladresse/-subnetz des Pakets
- **dst**:addr - Zieladresse/-subnetz des Pakets
- **net**:addr - Quell- und Zieladresse/-subnetz des Pakets
  (entspricht src:addr AND dst:addr)
- **dev**:iface - Netz-Schnittstelle, über die das Paket empfangen wurde

Als Syntax für Adressen und Subnetze wird alles akzeptiert, was *iptables* für die Argumente `--source` und `--destination` vorsieht. Das sind entweder einzelne IP-Adressen, Subnetze

in Form von IP-Adresse und Netzmaske (z.B. 192.168.0.0/255.255.0.0) oder IP-Subnetze in CIDR-Schreibweise (z.B. 192.168.0.0/16). Als Netz-Interface wird nur der Basisname der Netz-Schnittstelle erlaubt, d.h. Aliases wie eth0:1 sind unzulässig und müssen z.B. über dev:eth0 dst:XXX realisiert werden.

Es ist auf diesem Wege möglich mit einfachen Mitteln selbst komplexere Netzstrukturen abzubilden. Möchte der Administrator einzelnen Hosts den Zugriff auf bestimmte Dienste erlauben, so ist auch hier das Zonen-Modell das geeignete Mittel. Es können sowohl einzelne Hosts, als auch Gruppen von Hosts zu einer Zone zusammengefasst werden und mit separaten Regelsätzen versehen werden.

Für jede Zone kann separat ein Verzeichnis von Regelsätzen definiert werden, was in Kapitel 6.3.3 beschrieben wird.

Die Umsetzung des zonenspezifischen Regelsatzes in *iptables*-Regeln erfolgt über Chains mit dem Namen der Zone. Heißt die Zonendatei z.B. `mwn.zone`, so wird eine neue Chain `ZONE_MWN` erstellt, in die die aktiven Regelsätze für diese Zone eingefügt werden. Jede Zonen-Chain enthält als letzte Regel `DROP` für alle Pakete. Die Zonen schließen sich so gegenseitig aus, d.h. gehört ein Paket zu einer Zone, so kann es nicht auch zu einer anderen Zone gehören. In der `INPUT`-Chain wird für jede Filterzeile in der Zonendatei `mwn.zone` eine Sprungregel auf `ZONE_MWN` erstellt.

Beispiele

Beispiel 1 Alle eingehenden Pakete über die Netz-Schnittstelle eth0 und einer IPv4-Absenderadresse innerhalb von 192.168.0.0/16 oder 10.0.0.0/8 werden in dieser Zone behandelt.

```
dev:eth0 src:192.168.0.0/16
dev:eth0 src:10.0.0.0/8
```

Der Ausdruck wird gemappt auf:

```
iptables -A INPUT -i eth0 -s 192.168.0.0/16 -j ZONE_XXX
iptables -A INPUT -i eth0 -s 10.0.0.0/8 -j ZONE_XXX
```

Beispiel 2 Alle eingehenden Pakete mit einer Absender- und Zieladresse innerhalb von 192.168.0.0/16 werden in dieser Zone behandelt.

```
net:192.168.0.0/16
```

Der Ausdruck wird gemappt auf:

```
iptables -A INPUT -s 192.168.0.0/16 -d 192.168.0.0/16 -j
    ZONE_XXX
```

Beispiel 3 In einer Zonen-Datei können IPv4- und IPv6-Zonen auch gemischt vorkommen, jedoch nicht innerhalb einer einzelnen Zeile.

```
src:129.187.0.0/16
src:2001:4CA0::/32
```

Der Ausdruck wird gemappt auf:

```
iptables -A INPUT -s 129.187.0.0/16 -j ZONE_XXX
ip6tables -A INPUT -s 2001:4CA0::/32 -j ZONE_XXX
```

6.3.3 Regelsatz und Regel-Templates

Regelsatz

Der Regelsatz der *lrzfw* ist zustandsbehaftet und arbeitet nach einem Whitelist-Konzept, d.h. alles was nicht explizit erlaubt ist wird blockiert.

Die einzige Ausnahme wird bei IPv6 gemacht, wenn die verfügbare *iptables*-Version älter als 1.4.0 ist, da bei diesen ein entsprechender *state*-Match fehlt. In diesem Fall generiert die *lrzfw* lediglich zustandslose Freigaben für UDP- und TCP-Ports sowie für einige ICMPv6-Typen, die ansonsten nur durch die Zustandsbehaftung zugelassen würden. Dazu gehören insbesondere ICMPv6-Fehler, wie z.B. *destination unreachable*, *time exceeded*, aber auch *echo reply*.

Außerdem wird im Falle von IPv6 die Neighbor Discovery statisch zugelassen, da diese für einen reibungslosen Betrieb nötig sind.

Die Auswertungsreihenfolge des Regelsatzes ist:

- Blacklist
- Statische Regeln (Zustandsbehaftung, ICMPv6 Neighbor Discovery)
- Globale Zone
- Eigene Zonen

Netzzonen Die *lrzfw* unterstützt sowohl einen globalen (globale Zone) als auch einen zonenbezogenen (eigene Zonen) Regelsatz. Der globale Regelsatz gilt für den gesamten eingehenden Netz-Traffic, der zonen-bezogene Regelsatz bezieht sich spezifisch auf den Traffic, der einer bestimmten Netzzone zugeordnet werden kann. In der Auswertungsreihenfolge befindet sich der globale Regelsatz vor dem zonen-bezogenen.

Regel-Templates

Zur Vereinfachung der Firewall-Konfiguration werden durch die *lrzfw* sogenannte Regel-Templates verwendet. Diese enthalten einen spezifischen Regelsatz für bestimmte Dienste. Bei der Konfiguration werden eine Reihe von Templates eingebunden, die zusammen den fertigen Regelsatz generieren. Durch diese Vorgehensweise ist beim Konfigurieren der Firewall in vielen Fällen kein detailliertes Wissen über die angebotenen Dienste des Hosts nötig, da die Komplexität durch die Regel-Templates verschattet wird.

Die Templates sind Shellscript-Dateien und können verwendet werden, um komplexe Aktionen auszuführen. Der *lrzfw*-Core ermöglicht allen Template-Dateien den Export bestimmter Konfigurations-Variablen, die jeweils mit einem Standardwert belegt sind.

Wird ein Regel-Template in den Regelsatz eingebunden, so besteht grundsätzlich die Möglichkeit, diese Variablen zu überschreiben und so beispielsweise Portnummern oder Logging-Optionen an die Anforderungen des Hosts anzupassen.

Die Regel-Templates werden im Verzeichnis `/etc/lrzfw/rules-available/` in Form von `*.rule`-Dateien vorinstalliert.

Konfiguration des Regelsatzes

Die *lrzfw* ermöglicht einen globalen Regelsatz für den gesamten Host, sowie separate Regelsätze für jede eigene Zone .

Beim Bearbeiten der Regelsätze ist folgendes zu beachten:
Wird im laufenden Betrieb ein Port geschlossen (z.B. indem eine Regel deaktiviert wird), so kann es vorkommen, dass bereits aktive Verbindungen weiterhin aktiv bleiben. Dies ist eine direkte Folge aus der Zustandsbehaftung. Die *lrzfw* lässt alle Pakete mit Zustand ESTABLISHED oder RELATED zu. Es ist nicht vorgesehen bei einem Neustart der Firewall die Liste der aktuellen Verbindungen zu löschen!

Globale Zone Wie beim Zonen-Modell aus Kapitel 6.3.2 wird auch für die Konfiguration des Regelsatzes das Debian-Modell aus Kapitel 6.3.1 verwendet. Dazu existieren die Verzeichnisse `/etc/lrzfw/rules-available/` (siehe 6.3.3) und `/etc/lrzfw/rules-enabled/`.

In `rules-enabled` befinden sich symbolische Links, die auf Regeldateien im Verzeichnis `rules-available` verweisen. Die *lrzfw* wertet beim Startvorgang die `*.rule`-Dateien in `rules-enabled` in alphabetischer Reihenfolge aus. Die Regel-Templates werden initialisiert und erstellen anschließend ihre spezifischen *iptables*-Firewallregeln.

Um auf die Reihenfolge direkt Einfluss zu nehmen kann der Administrator den symbolischen Links einen numerischen Präfix im Dateinamen beifügen, z.B. `000-sshd.rule`.

Eigene Zonen Der Regelsatz für eigene Zonen befindet sich im Verzeichnis `/etc/lrzfw/zones/<zone>-rules/*.rule`, wobei `<zone>` der Basisname der `*.zone`-Datei ist, durch die die Zone beschrieben wird. Der Aufbau des Verzeichnisses ist identisch mit der Konfiguration der globalen Zone.

Beispiel:
Es existiert eine LAN-Zone, die durch die Datei `/etc/lrzfw/zones/lan.zone` beschrieben wird. Der Regelsatz für diese Zone befindet sich im Verzeichnis `/etc/lrzfw/zones/lan-rules/*.rule`.

Anpassung von Regel-Templates Falls der Administrator z.B. andere Portnummern als die Standardports für bestimmte Dienste verwendet, so muss er dies dem betreffenden Regel-Template mitteilen. Dazu existiert die Möglichkeit, dass im Verzeichnis `rules-enabled` bzw. `zones/*-rules/` zusätzlich zu den symbolischen Links `*.conf`-Dateien mit gleichem Basisnamen erstellt werden. Diese sind grundsätzlich optional. Die Namen der Konfigurationsvariablen können direkt im Regel-Template nachgelesen werden. Die `*.conf`-Dateien sind ebenfalls Shellscripte, welche den Variablen der Regeldatei eigene, statische Werte zuweisen (Schlüssel-Wert-Paare). Desweiteren ist es auch möglich eine `*.conf`-Datei mit eigenem Shellscript-Programmcode darin zu erstellen, um z.B. Portnummern aus den Konfigurationsdateien anderer Dienste direkt auszulesen oder Regel-Templates anhand bestimmter Eigenschaften des Hostsystems zu konfigurieren. Den Möglichkeiten sind dabei kaum Grenzen gesetzt.

Beispiel:
Es existiert eine aktive Regel `/etc/lrzfw/rules-enabled/000-sshd.rule`. Der SSH-Port auf dem aktuellen Server wird jedoch aus Sicherheitsgründen nicht auf den Standardport 22

gelegt, sondern z.B. auf 323. Dies wird in der Praxis häufig gemacht, um den SSH-Server vor oberflächlichen Portscans zu verbergen.

Dazu wird eine neue Datei `/etc/lrzfw/rules-enabled/000-sshd.conf` mit dem Inhalt
`LRZFW_SSHD_PORT=323`

angelegt.

Durch die benutzerdefinierte Konfiguration ist es auch möglich ein einzelnes Regel-Template mehrfach in der gleichen Zone einzubinden. Dazu wird ein weiterer symbolischer Link mit einem anderen Dateinamen erstellt, welcher auf die gleiche Regeldatei im Verzeichnis `rules-available` verweist und ggf. eine entsprechende Konfigurationsdatei hinzugefügt. Dies ist besonders dann sinnvoll, wenn mehrere Instanzen gleicher Dienste auf einem einzelnen System aktiv sind.

6.3.4 Blacklist

Die Blacklist wird über eine zentrale Konfigurationsdatei `/etc/lrzfw/blacklist` realisiert, in der pro Zeile eine einzelne IP-Adresse oder ein IP-Subnetz eingetragen wird. Dies hat den Vorteil, dass automatisiert per Script neue IP-Adressen hinzugefügt oder alte gelöscht werden können. Als Syntax für Adressen und Subnetze wird alles akzeptiert was *iptables* für die Argumente `--source` und `--destination` vorsieht. Das sind entweder einzelne IP-Adressen, Subnetze in Form von IP-Adresse und Netzmaske (z.B. 192.168.0.0/255.255.0.0) oder IP-Subnetze in CIDR-Schreibweise (z.B. 172.16.0.0/12). Durch die Abstraktionsschicht ist es hier möglich IPv4- und IPv6-Adressen beliebig zu mischen. Leere Zeilen werden ignoriert, einzeilige Kommentare beginnen mit `#`. Bei fehlerhafter Syntax wird der Ladevorgang mit Fehler abgebrochen.

Beispiele:

```
# Einzeiliger Kommentar
1.2.3.4
172.16.0.0/12 # Kommentar..
192.168.0.0/255.255.255.0
fe80::1
fec0::/10
```

In den *iptables*-Regeln wird die Blacklist als separate Chain mit Namen `BLACKLIST` umgesetzt. Dort ist es ebenso möglich im laufenden Betrieb temporär neue Adressen einzufügen oder alte zu entfernen.

Im fertigen Regelsatz wird die Blacklist als erste Regel in der `INPUT`-Chain ausgewertet - noch vor den zustandsbehafteten Regeln.

ACHTUNG: Blockt sich der Administator versehentlich selbst mithilfe der Blacklist, so werden auch alle seine aktiven Verbindungen blockiert. Dies hat zur Folge, dass möglicherweise physischer Zugriff zum System erforderlich ist, um den Zugang wieder freizugeben!

Die fertigen *iptables*-Chains für das Beispiel sehen wie folgt aus (IPv4):

```
Chain BLACKLIST (1 references)
target     prot opt source               destination
DROP       all  --  1.2.3.4              0.0.0.0/0
DROP       all  --  172.16.0.0/12        0.0.0.0/0
DROP       all  --  192.168.0.0/24       0.0.0.0/0
```

```
RETURN     all  --  0.0.0.0/0            0.0.0.0/0

Chain INPUT (policy DROP)
target     prot opt source               destination
BLACKLIST  all  --  0.0.0.0/0            0.0.0.0/0
# ...
```

Analog dazu IPv6:

```
Chain BLACKLIST (1 references)
target     prot opt source               destination
DROP       all      fe80::1/128          ::/0
DROP       all      fec0::/10            ::/0
RETURN     all      ::/0                 ::/0

Chain INPUT (policy DROP)
target     prot opt source               destination
BLACKLIST  all      ::/0                 ::/0
# ...
```

6.3.5 Umsetzung in iptables

Der fertige *iptables*-Regelsatz der *lrzfw* gliedert sich in mehrere, voneinander getrennte Teile. Alle Teilkomponenten der Firewall sind in eigene *iptables*-Chains ausgelagert. Die Blacklist wird über die `BLACKLIST`-Chain modelliert, die globale Zone über die `GLOBAL_ZONE`-Chain und die eigenen Zonen über eigene Zonen-Chains.

In der `INPUT`-Chain erfolgt zunächst ein Aufruf der `BLACKLIST`-Chain, anschließend werden die zustandsbehafteten Regeln eingefügt, welche u.A. dafür zuständig sind, dass bereits offene Verbindungen direkt fortgeführt werden. Bei IPv6 sind zusätzlich noch Regeln für die Neighbor Discovery über ICMPv6 vorhanden. Schließlich folgen Aufrufe der `GLOBAL_ZONE`-Chain sowie der Chains der eigenen Zonen. Für jede Zeile in der Zonen-Definitionsdatei wird dabei eine Sprungregel auf die jeweilige Zonen-Chain erstellt.

In den Zonen-Chains sind die *iptables*-Regeln enthalten, die durch die in der Konfiguration gewählten Regel-Templates generiert wurden. Sind bei diesem Vorgang weitere Unter-Chains nötig geworden, so tragen diese als Namenspräfix den Namen der Zonen-Chain. Bei Unter-Chains auf tieferer Ebene ist der Präfix der Name der jeweils übergeordneten Chain. Durch diese Baumstruktur lassen sich alle derartigen Chains eindeutig einer bestimmten Zone zuordnen.

Beispiel: Heißt die Zonendatei z.B. `mwn.zone`, so wird eine neue Chain `ZONE_MWN` erstellt. Ist eine Unter-Chain mit Namen `LOGGING` nötig, so heißt diese `ZONE_MWN_LOGGING`. Bei einer weiteren Unter-Chain zweiter Ebene mit Namen `LOCK`, heißt die Chain entsprechend `ZONE_MWN_LOGGING_LOCK`.

Durch die genannten Maßnahmen ist der Regelsatz sehr übersichtlich. Betrachtet der Administrator die fertigen *iptables*-Regeln, so kann er einzelne Teile direkt bestimmten Konfigurationsdateien der *lrzfw* zuordnen.

Der Standard-Regelsatz nach der Installation der *lrzfw* befindet sich im Anhang 8.2.

6.4 Betriebsthemen

6.4.1 RPM-Paket

Das *lrzfw*-Paket kann am Leibniz Rechenzentrum als RPM-Paket automatisch auf alle gewünschten Zielrechner installiert werden. Dazu existiert ein zentraler RPM-Server, der von den Desktop-PCs und Servern automatisch abgefragt wird. Dieses System wird ebenfalls dazu genutzt, um Updates für veraltete RPM-Pakete einzuspielen.

Das Paket *lrzfw* ist als "conflicting" und "obsoleting" *SUSE-Firewall* markiert. Es kann deshalb nur installiert werden, nachdem die *SUSE-Firewall* deinstalliert wurde. Dies geschieht automatisch bei der Installation von *lrzfw* durch den RPM-Paketmanager. Für ein anschließendes Downgrade zurück auf *SUSE-Firewall* muss *lrzfw* manuell deinstalliert werden! Bei der Installation des Pakets wird das System außerdem auf eine Installation von *lrzpf* geprüft und eine Warnung angezeigt, falls sie noch installiert ist.

Im Auslieferungszustand wird eine lokale Zone auf der Netz-Schnittstelle `lo` installiert, in der alle eingehenden Pakete akzeptiert werden. Außerdem ist für den gesamten Host eine Regel aktiv, die den SSH-Zugriff auf dem Standardport tcp/22 erlaubt. So wird verhindert, dass sich der Administrator bei der Paketinstallation selbst aus dem System aussperrt. Direkt nach der Installation ist die Firewall inaktiv. Ebenso muss der Autostart über das init.d-Script manuell durch den Administrator aktiviert werden.

Zusätzlich zur *lrzfw* werden Konfigurationsdateien für rsyslogd und logrotate installiert sowie die betreffenden Dienste anschließend neu geladen (Kapitel 6.4.2).

Das RPM-Paket ist so konfiguriert, dass alle Dateien in `/etc/lrzfw/` als Konfigurationsdateien markiert sind. Alle benutzerdefinierten Dateien in diesem Verzeichnis sowie den Unterverzeichnissen werden bei einem Update der *lrzfw* nicht überschrieben. Bei den anderen Konfigurationsdateien wird, sofern sie manuell bearbeitet wurden, zunächst eine Sicherheitskopie angelegt bevor sie überschrieben werden.

In Kombination mit der Regel- und Zonenkonfiguration kann ein zusätzliches RPM-Paket verwendet werden, um neue Zonen und entsprechende Regelsätze bei der Installation zu erstellen. Wird auf dem Host beispielsweise ein Tivoli Storage Manager (TSM) Backup-Manager [tsm] installiert, so kann automatisch eine entsprechende Zone mit den IP-Adressen der Backup-Server sowie eine entsprechende Portfreigabe durch Einfügen von neuen Dateien aus dem RPM-Paket erstellt werden. So vereinfacht sich die Wartung einer größeren Anzahl von Endgeräten deutlich.

Die Liste aller Dateien im SUSE-RPM-Paket ist in Kapitel 8.3 abgedruckt.

6.4.2 Logging

Der Bereich Logging wird durch die *lrzfw* nur als Feature verschiedener Regel-Templates abgebildet. Ein Logging ist nur für sicherheitsrelevante Aktivitäten vorgesehen. Diese lassen sich häufig auf bestimmte Dienste zurückführen, die bevorzugte Angriffsziele sind.

Beispiel SSH-Server: Ein besonders häufiges Angriffsziel sind SSH-Server, die direkt mit dem Internet verbunden sind. Diese sind regelmäßigen automatisierten Angriffen durch Bots und ähnlichen Schadprogrammen ausgesetzt. Das gängigste Angriffsmuster ist hierbei eine Wörterbuch-Attacke, bei der häufig verwendete Kombinationen von Benutzername und Passwort systematisch durchgetestet werden. Solche Scans sind meist nur dann erfolgsversprechend, wenn eine ausreichende Zahl an Scanversuchen in vertretbarer Zeit möglich ist. *iptables* bietet zum Überwachen solcher Aktionen das Modul *recent* (Kapitel 2.4.4) an, mit

dessen Hilfe z.B. beim Überschreiten einer vorgegebenen Zahl von Verbindungsanfragen pro Zeiteinheit ein Firewall-Logging ausgelöst werden kann.

Um ein exzessives Logging zu vermeiden, nutzen alle ausgelieferten Regel-Templates den *hashlimit*-Match, der in Kapitel 2.4.4 beschrieben wird.

Backends

Als Logging-Backends stehen bei der *lrzfw* wahlweise syslogd (LOG) oder ulogd (ULOG, NFLOG) zur Verfügung. Diese können in der Konfigurationsdatei `/etc/default/lrzfw` ausgewählt werden. Der Standardwert ist LOG.

Das LOG-Target sendet Logeinträge an den Kernel-Ringbuffer, wo sie z.B. durch die Usermode-Daemons rsyslogd oder syslog-ng ausgelesen und weiterverarbeitet werden können. Die Hauptaufgabe dieser Programme liegt im Speichern der Nachrichten in Logdateien. Der Nachteil dieses Targets ist es, dass der Kernel-Ringbuffer durch seine feste Größe bei starker Logging-Aktivität schnell überläuft und so alte Logeinträge überschrieben werden. Dadurch wird der Ringbuffer für normales Kernel-Logging oft unbrauchbar.

Die Targets ULOG und NFLOG senden ihre Logging-Einträge direkt über einen Netlink-Socket an einen Usermode-Daemon, z.B. ulogd. Dort werden die Logeinträge entsprechend weiterverarbeitet.

Systemkonfiguration

rsyslog Für ein effektives Logging wird von *lrzfw* im RPM-Paket eine Konfigurationsdatei für den Logging-Dienst rsyslogd mitgeliefert, so dass alle auftretenden syslog-Logging-Events automatisch in die Logdatei `/var/log/lrzfw.log` umgeleitet werden. Dazu wird der `"lrzfw:␣"`-Präfix genutzt, der durch die Abstraktionsschicht aus Kapitel 6.2 automatisch eingefügt wird.

Mit diesem Dienst ist es darüberhinaus möglich die Logdateien in Echtzeit auf einen zentralen Syslog-Server zu übertragen. Dies ist jedoch nicht Teil des RPM-Pakets und muss ggf. manuell vom Administrator konfiguriert werden. Auf diese Weise wird der Administrator bereits während eines laufenden Angriffs auf das Problem hingewiesen und kann entsprechend reagieren.

Der Inhalt der *rsyslog*-Konfigurationsdatei ist in Kapitel 8.4 abgedruckt.

syslog-ng Das bei älteren SUSE-Distributionen eingesetzte *syslog-ng* wird durch die *lrzfw* nicht direkt unterstützt. Es bietet keine Möglichkeit neue Logging-Filter durch Hinzufügen von Konfigurationsdateien zu erstellen. Die derzeit einzige Möglichkeit das *lrzfw*-Logging in eine eigene Logdatei umzuleiten ist die Modifikation der Konfigurationsdatei `/etc/syslog-ng/syslog-ng.conf`, was bei automatisierter Editierung die Gefahr birgt, dass andere Einstellungen überschrieben werden. *syslog-ng* leitet sämtliches *iptables*-Logging allerdings bereits automatisch nach `/var/log/firewall` um.

logrotate Um ein Überlaufen der *lrzfw*-Logdatei zu verhindern, liefert das RPM-Paket eine Konfigurationsdatei für das *logrotate*-Dienstprogramm [log] aus. Aufgabe von *logrotate* ist es in vorgegebenen Zyklen Logdateien zu sichern und zu komprimieren ("Rotieren"). Nach einem festgelegten Zeitraum werden alte Logdateien automatisch gelöscht.

Für die *lrzfw*-Logdatei wird ein tägliches Rotieren mit einer Aufbewahrungszeit von sieben Tagen eingerichtet. Alle Logdateien werden ab dem zweiten Tag komprimiert.

Der Inhalt der *logrotate*-Konfigurationsdatei für SUSE Linux ist in Kapitel 8.5 abgedruckt.

6.4.3 Backup

Die *lrzfw* verfügt über eine rudimentäre Backup-Funktion. Beim Starten der Firewall wird die aktuelle Konfiguration in `/etc/lrzfw/` in eine bzip2-komprimierte tar-Datei gesichert und in einem distributionsspezifischen Backup-Verzeichnis abgelegt. Diese Datei wird bei jedem Start der Firewall automatisch überschrieben. Backups können auch manuell ausgelöst werden. In diesem Fall enthält der Dateiname der Backupdatei das aktuelle Datum und wird in Folge nicht mehr automatisch überschrieben. Manuelle Backups müssen auch manuell gelöscht werden.

Als Sicherheitsfunktion legt die *lrzfw* beim Starten automatisch ein temporäres Backup der aktuell aktiven *iptables*-Regeln an. Falls der Ladevorgang durch einen Fehler unterbrochen wird, so wird die ursprüngliche Firewallkonfiguration durch das temporäre Backup wiederhergestellt. So kann verhindert werden, dass halbfertige Regelsätze entstehen und sich der Administrator ggf. selbst aus dem System aussperrt. Die *iptables*-Regeln werden vollständig oder gar nicht initialisiert.

Wird die Ausführung von *lrzfw* durch `SIGINT`- oder `SIGTERM`-Signale unterbrochen, so erfolgt ebenfalls ein Rollback durch das temporäre Backup des *iptables*-Regelsatzes.

6.5 Entwicklung

6.5.1 Variablen und (Hilfs-)Funktionen

Um die Entwicklung von Regel-Templates zu vereinfachen und den fertigen *iptables*-Regelsatz zu standardisieren stellt die *lrzfw* einige Hilfsfunktionen und globale Variablen im Shellscript zur Verfügung, von denen einige kurz beschrieben werden.

`LRZFW_CURRENT_CHAIN`

`LRZFW_CURRENT_CHAIN`: Globale Variable, die den Namen der *iptables*-Chain enthält, die aktuell von *lrzfw* bearbeitet wird. Regel-Templates sollten diesen Wert verwenden, um ihren Regelsatz in die korrekte Chain einzufügen. Die Variable wird zudem von den Funktionen `iptables_port_tcp` und `iptables_port_udp` bei der Regel-Erstellung verwendet, um dem Entwickler Schreibarbeit zu ersparen und die Regel-Templates so zu vereinfachen. Erstellt ein Regel-Template Unter-Chains, so sollte `LRZFW_CURRENT_CHAIN` als Präfix dienen und durch den Basisnamen der Template-Datei ergänzt werden. Der alte Wert von `LRZFW_CURRENT_CHAIN` ist vor einer Modifikation zu sichern und anschließend wiederherzustellen.

Beispiel:

```
add_logging() {
    # backup chain name
    local old_chain="$LRZFW_CURRENT_CHAIN"

    # new chain name
```

```
    LRZFW_CURRENT_CHAIN="${LRZFW_CURRENT_CHAIN}_LOGGING"

    # add new chain
    iptables -N "$LRZFW_CURRENT_CHAIN" || return $?

    # add logging rules
    iptables_port_tcp 80 -j LOG --log-prefix "http␣connect:␣"
    || return $?

    # restore old chain
    LRZFW_CURRENT_CHAIN="$old_chain"

    return 0
}
```

`iptables`

`iptables <args>`: Diese Funktion kann genutzt werden wie ein normaler *iptables*-Aufruf per Konsole. Wie bereits im Kapitel 6.2 beschrieben, entscheidet die `iptables`-Funktion anhand der Aufrufargumente für welche IP-Version der Aufruf geeignet ist. Diese Prüfung beschränkt sich der Einfachkeit halber auf
`-s|--source` (Absenderadresse)
`-d|--destination` (Zieladresse)
`-p|--protocol` (Protokoll, z.B. TCP/UDP/ICMP)
`-m|--match` (Match-Erweiterung)

Zusätzlich gibt es die Möglichkeit über die Argumente `-4|--ipv4` bzw. `-6|--ipv6` eine IP-Version zu forcieren. Stellt die Hilfsfunktion einen Widerspruch bei der automatischen Protokollerkennung fest (d.h. weder IPv4 noch IPv6 ist möglich), so wird mit der Aufruf mit einer Fehlermeldung abgebrochen und der Exitcode 1 zurückgegeben.

Eine Ausnahme stellt hier ein Widerspruch zwischen dem automatisch erkannten Protokoll und den Argumenten `-4|--ipv4` bzw. `-6|--ipv6` dar. Ein Konflikt wird hier ohne Fehler mit Exitcode 0 ignoriert.

Um ein einheitliches Logging zu ermöglichen werden durch *iptables* außerdem die Log-Präfixe `--log-prefix|--ulog-prefix|--nflog-prefix` mit dem gemeinsamen *lrzfw*-Präfix `"lrzfw:␣"` erweitert. Dabei ist die maximale Länge des Log-Präfix zu beachten, die sich je nach verwendetem Logging-Backend unterscheidet:

- `--log-prefix`: 22 Zeichen
- `--ulog-prefix`: 25 Zeichen
- `--nflog-prefix`: 57 Zeichen

Die Logging-Argumente `--log-level`, `--log-tcp-sequence`, `--log-tcp-options`, `--log-ip-options` und `--log-uid` sind nur beim Syslog-Backend (`LOG`) möglich und `--ulog-nlgroup`, `--nflog-group`, `--ulog-cprange`, `--nflog-range`,

`--ulog-qthreshold` und `--nflog-threshold` nur bei `ULOG` oder `NFLOG`. Diese Argumente sollten durch Regel-Templates deshalb nur selektiv nach Prüfung des Logging-Backends verwendet werden!

Die Prefixes und die Logging-Targets werden jeweils auf das in der Konfiguration gewählte Logging-Backend gemappt.

Beim Zusammenfügen der finalen Aufrufparameter an `ip4tables` und `ip6tables` wird die Reihenfolge der Argumente verändert. `-m|--match` wird dabei an den Anfang der Befehlszeile gelegt und Duplikate eliminiert. Es spielt also keine Rolle in welcher Reihenfolge die zu einem Match gehörenden Argumente an die *iptables*-Funktion übergeben werden. Dies gilt **nicht** für Argumente, die sich auf ein bestimmtes Target beziehen! In diesem Fall muss das Target `-j|--jump` **vor** den weiteren Argumenten stehen, die sich auf dieses Target beziehen.

`ip4tables`

`ip4tables <args>`: Diese Funktion ruft direkt den IPv4-*iptables*-Befehl des Hosts auf. Siehe `iptables (8)`.

`ip6tables`

`ip6tables <args>`: Diese Funktion ruft direkt den IPv6-*iptables*-Befehl des Hosts auf. Falls IPv6 auf dem Host nicht verfügbar ist wird lediglich der Exitcode 0 zurückgegeben. Siehe `ip6tables (8)`.

`iptables_port_tcp`

`iptables_port_tcp <ports> [..]`: Erstellt eine zustandsbehaftete Regel für das TCP-Protokoll in die aktuelle Chain (`LRZFW_CURRENT_CHAIN`). Die Regel matcht nur auf TCP-Verbindungsanfragen (SYN). Es ist zulässig mehrere Portnummern gleichzeitig zu übergeben. Diese sind per Whitespace getrennt als erstes Argument anzugeben. Alle weiteren Argumente sind *iptables*-spezifische Argumente und werden unverändert an die *iptables*-Funktion durchgereicht.

Die `iptables_port_tcp`-Funktion fügt kein Target zur eigentlichen Regel hinzu. Dies muss der Benutzer selbst tun. Dabei ist die gleiche Reihenfolge der Argumente zu beachten wie bei normalen *iptables*-Aufrufen.

ACHTUNG: Falls für IPv6 keine Zustandsbehaftung verfügbar ist, generiert `iptables_port_tcp` für IPv6 eine nicht zustandsbehaftete Regel, die **alle** TCP-Pakete der Verbindung matched!

Beispiel:

```
# open tcp ports 80 (http) and 443 (https)
iptables_port_tcp "80␣443" -j ACCEPT
# log tcp port 22 (ssh)
iptables_port_tcp 22 -j LOG --log-prefix "SSH␣connect:␣"
```

Der Rückgabewert von `iptables_port_tcp` ist ein normaler Exitcode.

`iptables_port_udp`

`iptables_port_udp <ports> [..]`: Erstellt eine Regel für das UDP-Protokoll in die aktuelle Chain (`LRZFW_CURRENT_CHAIN`). Analog zu `iptables_port_tcp`, jedoch nicht zustandsbehaftet.

`iptables_port_ex`

`iptables_port_ex <ports> [..]`: Dient als Backend von `iptables_port_tcp` und `iptables_port_udp`. Erzeugt aus der gegebenen Portliste eine Folge von iptables-Aufrufen mit `--dport`-Argument. Es sind verpflichtend als weitere Argumente das Protokoll (`-p|--protocol`) sowie der zugehörige Match (`-m|--match`) anzugeben.

Beispiel:

```
# open tcp ports 80 (http) and 443 (https)
iptables_port_ex "80␣443" -p tcp -m tcp -j ACCEPT
```

`environ_ipt_has_match`

`environ_ipt_has_match <name> <protocol>`: Prüft ob ein bestimmter Match bei den installierten *iptables* der gewünschten IP-Version (ipv4 oder ipv6) verfügbar ist. Falls er verfügbar ist gibt die Funktion den Exitcode 0 zurück, andernfalls einen Wert größer als 0.

Beispiel:

```
if ! environ_ipt_has_match "recent" "ipv6"; then
    echo "Match␣recent␣is␣not␣available␣for␣IPv6.␣Disabling␣
    features.."
fi
```

`environ_ipt_match_filter`

`environ_ipt_match_filter <name> <protocol>`: Gibt analog zu `environ_ipt_has_match` über stdout ein Filter-Argument (`--ipv4` oder `--ipv6`) aus sofern der Match nicht verfügbar ist. Dient somit als Vereinfachung beim Erstellen von *iptables*-Aufrufen mit Matches, die nicht bei allen *iptables*-Versionen verfügbar sind.

Beispiel:

```
local filter=$(environ_ipt_match_filter "state" "ipv6")
# Falls der state-Match unter IPv6 nicht verfuegbar ist,
# so traegt die Variable filter den Wert "--ipv4", andernfalls
# ist der Wert leer.
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
    ACCEPT $filter
```

`counter_autoid`

`counter_autoid <prefix>`: Erstellt eine globale Zählervariable mit dem Namen `prefix` und initialisiert sie mit dem Wert 0. Falls die Variable bereits vorhanden ist, so wird sie

inkrementiert. Der aktuelle Wert der Variablen ist anschließend in der globalen Variable `LRZFW_COUNTER` hinterlegt.

Diese Funktion dient zur Generierung von eindeutigen Bezeichnern, z.B. als Suffix für den Name bei einem *iptables*-Match.

Beispiel:

```
counter_autoid "HASHLIMIT_SSHD"   # first call with prefix
    "HASHLIMIT_SSHD"
echo "Counter␣is␣$LRZFW_COUNTER"  # "Counter is 0"
counter_autoid "HASHLIMIT_SSHD"
echo "Counter␣is␣$LRZFW_COUNTER"  # "Counter is 1"
```

`modprobe_multi`

`modprobe_multi <mod1> [mod2] [mod3] [..]`: Dient zum Laden von einem oder mehreren Kernel-Modulen in einem einzelnen Aufruf.

Beispiel:

```
modprobe_multi ip_conntrack_ftp ip_nat_ftp
```

`warning`

`warning <msg>`: Gibt eine Warnung über die Konsole aus.

`error`

`error <msg>`: Gibt einen Fehler über die Konsole aus. Die Funktion beendet **nicht** die Ausführung!

`debug`

`debug <msg>`: Gibt eine Debug-Meldung über die Konsole aus, sofern VERBOSE aktiv ist.

`is_int`

`is_int <arg>`: Gibt Exitcode 0 zurück, falls `arg` eine dezimale Zahl ist, andernfalls 1.

`is_ipv4`

`is_ipv4 <arg>`: Gibt Exitcode 0 zurück, falls `arg` eine IPv4-Adresse oder ein IPv4-Subnetz ist. Es erfolgt eine reine Syntaxprüfung. Als Schreibweisen sind sowohl IP-Adresse mit Netzmaske als auch IP-Adresse mit Präfixlänge (CIDR) zugelassen.

Beispiel:

```
test() {
    if is_ipv6 "$1"; then
        echo "$1␣is␣an␣ipv4␣address␣or␣subnet"
    else
```

```
        echo "$1 is no ipv4 address or subnet"
    fi
}

# 127.0.0.1 is an ipv4 address or subnet
test "127.0.0.1"

# 192.168.178.1/255.0.0.0 is an ipv4 address or subnet
test "192.168.178.1/255.0.0.0"

# 127.0.0.1/15 is an ipv4 address or subnet
test "127.0.0.1/15"

# 127.0.0.1/xyz is no ipv4 address or subnet
test "127.0.0.1/xyz"

# ::1 is no ipv4 address or subnet
test "::1"

# 2001:4ca0::1/32 is no ipv4 address or subnet
test "2001:4ca0::1/32"

# xyz is no ipv4 address or subnet
test "xyz"
```

`is_ipv4_addr`

`is_ipv4_addr <arg>`: Gibt Exitcode 0 zurück, falls `arg` eine IPv4-Adresse. Es erfolgt eine reine Syntaxprüfung. Die Verwendung ist analog zu `is_ipv4`.

`is_ipv6`

`is_ipv6 <arg>`: Gibt Exitcode 0 zurück, falls `arg` eine IPv6-Adresse oder ein IPv6-Subnetz ist. Es erfolgt eine reine Syntaxprüfung. Als Subnetz-Schreibweise ist nur IP-Adresse und Präfixlänge (CIDR) zugelassen.

Beispiel:

```
test() {
    if is_ipv6 "$1"; then
        echo "$1 is an ipv6 address or subnet"
    else
        echo "$1 is no ipv6 address or subnet"
    fi
}

# ::1 is an ipv6 address or subnet
test "::1"
```

```
# 2001:4ca0:0:103:0:80:2:2 is an ipv6 address or subnet
test "2001:4ca0:0:103:0:80:2:2"

# 2001:4ca0::1/32 is an ipv6 address or subnet
test "2001:4ca0::1/32"

# 127.0.0.1 is no ipv6 address or subnet
test "127.0.0.1"

# xyz is no ipv6 address or subnet
test "xyz"
```

`is_ipv6_addr`

`is_ipv6_addr <arg>`: Gibt Exitcode 0 zurück, falls `arg` eine IPv6-Adresse. Es erfolgt eine reine Syntaxprüfung. Die Verwendung ist analog zu `is_ipv6`.

`check_protocol`

`check_protocol <arg>`: Gibt Exitcode 0 zurück, falls `arg` ein in `/etc/protocols` registriertes Protokoll ist.

Beispiel:

```
if check_protocol "http"; then
    echo "http␣protocol␣registered␣in␣/etc/protocols"
fi
```

`str_toupper`

`str_toupper <arg>`: Gibt eine in Großbuchstaben konvertierte Kopie von `arg` per stdout aus.

Beispiel:

```
STRING=$(str_toupper "test_string") # ergibt "TEST_STRING"
```

`str_append_nodup`

`str_append_nodup <list> <entry>`: Gibt per stdout eine durch Leerzeichen getrennte Liste von Einträgen aus. Ist `entry` nicht in der Liste `list` enthalten, so wird es hinzugefügt. Andernfalls wird nur `list` ausgegeben. Die Funktion kann nur mit Einträgen arbeiten, die zusammenhängende Worte sind und weder Leerzeichen noch Tabulatoren oder Newlines enthalten.

Beispiel:

```
LIST="recent␣state"
```

```
# "recent state hashlimit"
echo $(str_append_nodup "$LIST" "hashlimit")

# "recent state"
echo $(str_append_nodup "$LIST" "state")
```

`str_make_variable_name`

`str_make_variable_name <string>`: Generiert aus einem String einen gültigen Shellscript-Variablennamen. Alle ungültigen Zeichen werden dabei durch `_` ersetzt. Die Ausgabe erfolgt über stdout.

`str_word_in_string`

`str_word_in_string <string> <word>`: Prüft ob das einzelne Wort `word` im String `string` vorkommt. Gibt Exitcode 0 zurück, falls das Wort enthalten ist.

Beispiel:

```
STRING="test this out"

# "this" ist enthalten -> exitcode 0
str_word_in_string "$STRING" "this"

# "is" nicht als einzelnes Wort enthalten
str_word_in_string "$STRING" "is"
```

`cmd_quiet`

`cmd_quiet [args]`: Führt einen Befehl mit den gegebenen Argumenten aus. Alle Ausgaben des Programms (stdout und stderr) werden nach `/dev/null` umgeleitet. `cmd_quiet` speichert die Befehlszeile in der globalen Variable `LAST_CMDLINE` für spätere Fehlermeldungen.

Beispiel:

```
cmd_quiet /sbin/iptables -A INPUT -j ACCEPT
cmd_quiet /etc/init.d/lrzfw restart
```

`get_date`

`get_date`: Gibt das aktuelle Datum formatiert als `YYMMDD_HHMMSS` nach stdout aus.

Beispiel:

```
FILENAME="/var/backup/backup_$(get_date).bak"
echo "Backup this data" > "$FILENAME"
```

`get_logging_backend`

`get_logging_backend`: Gibt den Namen des per Konfiguration gewählten *iptables*-Logging-Backends nach stdout aus (log, ulog, nflog). Der Wert ist immer gültig. Bei einem Konfigurationsfehler wird log ausgegeben.

6.5.2 Regel-Templates

Regel-Templates sind Shellscripte, die im Kontext der *lrzfw* ausgeführt werden. Als Template zum Erstellen eigener Regel-Templates existiert die Datei `/etc/lrzfw/rules-available/skel`, die ein einfaches Grundgerüst bietet.

Die Templates werden vom *lrzfw*-Core aufgerufen. Der Datenaustausch läuft dabei über einige globale Variablen:

- `RULE_CMD`: Wird von der *lrzfw* gesetzt und enthält die Art des Aufrufs. Die zulässigen Werte sind `start` und `info`.
- `RULE_NAME`: Wird vom Regel-Template beim `info`-Aufruf gesetzt und enthält einen kurzen Namen für das Regel-Template.

Als Rückgabewert soll das Template bei Erfolg den Wert 0 verwenden, andernfalls einen Wert größer als 0.

Beim Laden eines Regel-Templates erfolgt zunächst ein `info`-Aufruf, bei dem das Template die globale Variable `RULE_NAME` setzt und eigene Konfigurationsparameter in Form von globalen Variablen definiert und mit Standardwerten initialisiert. Es handelt sich dabei um genau die Werte, die anschließend durch `*.conf`-Dateien überschrieben werden können.

War der `info`-Aufruf erfolgreich, so sucht der Core nach einer zugehörigen `*.conf`-Datei und führt diese aus (sofern vorhanden).

Dann erfolgt ein `start`-Aufruf, den das Template nutzt um den *iptables*-Regelsatz zu erstellen. Dabei sind die Konfigurationsparameter des Regel-Templates mit den Werten der `*.conf`-Datei überschrieben.

Die `skel`-Datei hat folgenden Inhalt:

```
# this is an example rule template that should be used to
    create additional rule templates

# global variables are
# [in] RULE_CMD={start,info}        # command to the rule file
# [out] RULE_NAME=[string]          # name string of the rule
    file

# init rule module
rule_init() {
    # return the name of the rule
    RULE_NAME="skeleton␣rule"

    # initialize all global variables that can be modified by
    adding a .conf file to the rule
    LRZFW_SKEL_PORT=123
    LRZFW_SKEL_EXAMPLE_LOG=1
```

```
    # ..

    return 0
}

# start rule module
rule_start() {
    # call iptables_xxx subroutines to add the ruleset to the
    iptables chains.
    # you can skip chain names. they will be added
    automatically depending on the
    # current chain we are adding this rule to. that chain
    name is kept in global
    # variable LRZFW_CURRENT_CHAIN. you can also load
    additional kernel modules here.


    return 0
}

# module entry point
case "$RULE_CMD" in
    start)
        rule_start
        ;;
    init)
        rule_init
        ;;
    *)
        return 1
        ;;
esac
```

6.6 Beispiel-Konfiguration

Im folgenden Kapitel werden einige einfache Einsatz-Szenarien exemplarisch vorgestellt und erklärt, wie die *lrzfw* passend dazu konfiguiert wird.

6.6.1 Installation

Im einfachsten Fall befindet sich das zugehörige RPM-Paket auf der lokalen Festplatte des Servers, z.B. im Verzeichnis `/root`. Zum Installieren ist dann folgendes zu tun:

```
rpm -U /root/lrzfw-1.0a-0.1.noarch.rpm
```

Die Firewall ist anschließend installiert, jedoch nicht im Autostart per Runlevel eingetragen. Das wird erledigt mit:

```
insserv lrzfw
```

Nun ist die Firewall betriebsbereit. Es ist in der Standard-Konfiguration der SSH-Port 22 für alle Hosts offen.

6.6.2 TSM Backup Service

Am Leibniz Rechenzentrum wird ein TSM Backup System von IBM eingesetzt um Backups von Hosts zu erledigen. Dazu verbindet sich der Backup-Server automatisch mit den Endgeräten auf Port `1501/tcp` und lädt die betreffenden Daten über das Netz. Der TSM-Server hat eine statische IP-Adresse und greift aktuell über IPv4 auf die Clients zu.

Als erster Schritt wird eine neue Zone eingerichtet, die ausschließlich die IP-Adresse des TSM-Servers abbildet. Dazu wird die Datei `/etc/lrzfw/zones/000-tsm.zone` mit dem Inhalt

```
src:10.156.8.46
```

angelegt und das Regel-Verzeichnis `/etc/lrzfw/zones/000-tsm-rules` erstellt.

HINWEIS: Es ist ratsam Zonen, die nur einzelne IP-Adressen abbilden, mit einem numerischen Präfix zu versehen, so dass sie in der Sortierreihenfolge weit oben stehen. Andernfalls könnten größere Zonen, die das ganze Subnetz, in dem sich die einzelnen Adressen befinden, abbilden, die kleinere Zone überschatten und somit implizit überschreiben!

Anschließend wird ein Regel-Template erstellt, indem die Datei
`/etc/lrzfw/rules-available/skel` nach
`/etc/lrzfw/rules-available/tsm.rule` kopiert wird. Diese Datei wird folgendermaßen bearbeitet:

```
[..]

# init rule module
rule_init() {
    # return the name of the rule
    RULE_NAME="tsm␣backup␣rule"

    # initialize all global variables that can be modified by
     adding a .conf file to the rule
    LRZFW_TSM_PORT=1501

    return 0
}

# start rule module
rule_start() {
    # call iptables_xxx subroutines to add the ruleset to the
     iptables chains.
    # you can skip chain names. they will be added
     automatically depending on the
    # current chain we are adding this rule to. that chain
     name is kept in global
    # variable LRZFW_CURRENT_CHAIN. you can also load
     additional kernel modules here.
    iptables_port_tcp "$LRZFW_TSM_PORT" -j ACCEPT
```

```
    if [ $? -ne 0 ]; then
        return $?
    fi

    return 0
}

[..]
```

Nun wird die Regel für die Zone `000-tsm` aktiviert mit:
`lrzfw -z tsm tsm`

Als letzten Schritt muss die Firewall neu geladen werden:
`lrzfw restart`

Der TSM-Server kann nun auf den Host zugreifen.

7 Zusammenfassung

7.1 Beurteilung

Zusammenfassend kann das *lrzfw*-Projekt als Erfolg angesehen werden. Es wurden mit wenigen Einschränkungen alle gesetzten Anforderungen erfüllt.

Der Grund dafür liegt in den veralteten Paketversionen der SLES- und openSUSE-Versionen, bei deren *iptables*-Installationen wichtige Matches fehlen.

Deshalb erstellt die *lrzfw* für die betreffenden Versionen automatisch nur einen zustandslosen Regelsatz für IPv6, da der *state*-Match dort teilweise nur für IPv4 angeboten wird (siehe Kapitel 6.3.3).

Außerdem fehlen auch andere Matches, so dass etwa ein SSH-Flooding aufgrund eines fehlenden *recent*-Matches nicht sinnvoll geloggt werden kann.

Ein weiteres Problem stellt die Verwendung von syslog-ng bei den alten SUSE-Distributionen dar. Es ist zwar eine Umleitung des Loggings in entsprechende Logdateien bereits im Auslieferungszustand eingerichtet, jedoch nicht spezifisch für die *lrzfw* (siehe Kapitel 6.4.2).

Eine Übersicht über die Kandidaten der Marktanalyse, der aktuell verwendeten Personal Firewalls am LRZ und der *lrzfw* findet sich in Abbildung 7.1.

	ARNO	ferm	firehol	shorewall	Vuurmuur	SuSE-Firewall	lrzpf	lrzfw
Version	1.9.2l	2.0.7	R5 v1.273	4.4.12.2	0.7	-	-	0.1a
Lizenz	GPLv2	GPLv2	GPL	GPLv2	GPLv2	GPLv2	proprietär	proprietär
Binary-Typ	Bash	Perl	Bash	Bash+Perl	Native	Bash+Native	Bash	Bash
Persistant	nein	nein	nein	nein	ja	nein	nein	nein
init.d-Script (LSB)	ja	nein	nein	ja	ja	ja	ja	ja
Regel-Templates	ja	nein	nein	ja	ja	ja	nein	ja
High/Lowlevel	high	low	low	high	high	high	low	high
IPv4-Support	ja	ja	ja	ja	ja	ja	ja	ja
IPv6-Support	experimentell	ja	nein	ab ipt 1.4.0	nein	ja	ja (stateless)	ja
IPv4/v6 identisch	nein	nein	nein	nein	nein	ja	nein	ja
Netzwerkzonen	ja	nein	nein	ja	ja	ja	nein	ja
Eigene Zonen	nein	nein	nein	ja	ja	nein	nein	ja
Blacklist-Funktion	ja	nein	ja	ja	ja	nein	nein	ja
Minimale iptables-Version	-	-	1.2.4	1.4.0	-	-	-	-
Logging via syslogd	ja	ja	ja	ja	ja	ja	ja	ja
Logging via ulogd	nein	ja	ja	ja	nein	nein	ja	ja
Logging konfiguriert	nein	nein	nein	nein	ja	ja	nein	ja
Konsole/GUI	Konsole	Konsole	Konsole	Konsole	GUI	GUI	Konsole	Konsole
Konfigurationsschema	zentral	zentral	zentral	zentral	zentral	zentral	zentral	dezentral
Konfigurationsart	key/value	iptables	iptables abstrakt	Text-Tabellen	Pseudo-Syntax	key/value	iptables	Symlink, Textfile
Konfiguration maschinell gut zu bearbeiten?	nein	nein	nein	nein	nein	nein	nein	ja
Dokumentation	gut	gut	gut	sehr gut	sehr schlecht	schlecht	nicht vorhanden	gut

Abbildung 7.1: Gesamtübersicht über alle Firewalls

7.2 Ausblick

7.2.1 Mögliche Weiterentwicklungen

Das *lrzfw*-Projekt ist als Grundlage für eine solide und anpassbare Firewall gedacht. Im Rahmen dieser Projektarbeit wurden deshalb nur die grundlegenden Funktionen entwickelt. Das Design ist leicht erweiterbar und wird zukünftig direkt vom Leibniz Rechenzentrum gepflegt und ggf. auch weiterentwickelt.

Zonen-Modell

Eine mögliche Weiterentwicklung wäre die Erweiterung des Zonen-Modells. Wie bereits anhand der *shorewall*-Firewall zu sehen war gibt es weitere, sehr vielfältige Möglichkeiten eine Netzzone zu definieren. Das Grundkonzept der *lrzfw* unterstützt als Kriterien nur die Netz-Schnittstelle sowie Quell- und Zieladresse von Paketen. Als weitere Möglichkeiten kommen z.B. auch policy-basierte Filter für den Einsatz von IPsec-Tunneln in Frage. Eine andere Möglichkeit wäre die Definition einer Zone anhand eines lokalen Serverdienstes oder auch einer lokalen Benutzergruppe. Ebenso wäre es denkbar, ineinander verschachtelte Netzzonen zu erlauben. Dadurch ist es möglich den Konfigurationsaufwand zu reduzieren und innerhalb einer Zone z.B. selektiv für einzelne Hosts oder Hostgruppen weitere Regeln zu aktivieren. In diesem Fall wäre es auch möglich den Syntax der Zonen-Dateien hierarchisch zu gestalten.

IP-Forwarding

Ein Thema, welches die *lrzfw* im Urzustand nicht behandelt, ist IP-Forwarding bzw. Routing. Das Zonen-Modell der Firewall bietet allerdings eine sehr günstige Möglichkeit um Routing-Policies zu konfigurieren. Dazu könnte man die Firewall dahingehend erweitern, dass die Kommunikation zwischen zwei gewählten Netzzonen per eigenem Regelverzeichnis im Sinne von `rules-enabled` gezielt reglementiert werden kann.

NAT

Im gleichen Schritt mit dem IP-Forwarding könnte auch eine Konfigurationsmöglichkeit geschaffen werden um zwischen zwei Zonen eine Network Address Translation zu schalten. Dies ist allerdings nur für IPv4 relevant und wird von den *iptables* auch nur dafür unterstützt.

Traffic Shaping

Auf Grundlage des Zonen-Modells ist es auch möglich eine Funktion zum Traffic Shaping, also einer Bandbreitenreservierung um eine gewisse Dienstgüte auch bei hoher Last zu erhalten, hinzuzufügen. Dies geht allerdings über den reinen Einsatz der *iptables* hinaus und erfordert weitere Einstellungen am System.

Traffic Accounting

Als zusätzlichen Dienst könnte auch ein Traffic Accounting angeboten werden. Die *lrzfw* erstellt aktuell für jede Netzzone eine eigene *iptables*-Chain. Als erste Regel könnte dort eine leere Dummy-Regel eingebaut werden, die nur dazu dient, die Anzahl der eingehenden Pakete sowie die übertragene Datenmenge aufzuzeichnen. Ein gleiches Vorgehen könnte auch bei der

`OUTPUT`- und der `FORWARD`-Chain genutzt werden. Dabei würde ein Cronjob in regelmäßigen Abständen die Werte auslesen und archivieren.

8 Anhang

8.1 iptables-Regelsatz SUSE-Firewall

8.1.1 IPv4

```
# Generated by iptables-save v1.4.4 on Wed Oct 13 23:14:25 2010
*raw
:PREROUTING ACCEPT [5:1138]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -i lo -j NOTRACK
-A OUTPUT -o lo -j NOTRACK
COMMIT
# Completed on Wed Oct 13 23:14:25 2010
# Generated by iptables-save v1.4.4 on Wed Oct 13 23:14:25 2010
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:forward_ext - [0:0]
:forward_int - [0:0]
:input_ext - [0:0]
:input_int - [0:0]
:reject_func - [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state ESTABLISHED -j LOG --log-prefix
    "SFW2-IN-ACC-EST␣" --log-tcp-options --log-ip-options
-A INPUT -m state --state ESTABLISHED -j ACCEPT
-A INPUT -p icmp -m state --state RELATED -j LOG --log-prefix
    "SFW2-IN-ACC-REL␣" --log-tcp-options --log-ip-options
-A INPUT -p icmp -m state --state RELATED -j ACCEPT
-A INPUT -i eth0 -j input_int
-A INPUT -j input_ext
-A INPUT -j LOG --log-prefix "SFW2-IN-ILL-TARGET␣"
    --log-tcp-options --log-ip-options
-A INPUT -j DROP
-A FORWARD -j LOG --log-prefix "SFW2-FWD-ILL-ROUTING␣"
    --log-tcp-options --log-ip-options
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -j LOG --log-prefix "SFW2-OUT-ERROR␣"
    --log-tcp-options --log-ip-options
```

```
-A input_ext -m pkttype --pkt-type broadcast -j DROP
-A input_ext -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix
    "SFW2-INext-ACC-SOURCEQUENCH␣" --log-tcp-options
    --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 4 -j ACCEPT
-A input_ext -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix
    "SFW2-INext-ACC-PING␣" --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A input_ext -p tcp -m tcp --dport 22 -j LOG --log-prefix
    "SFW2-INext-ACC-TCP␣" --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 22 -j ACCEPT
-A input_ext -m pkttype --pkt-type multicast -j DROP
-A input_ext -m pkttype --pkt-type broadcast -j DROP
-A input_ext -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -j
    LOG --log-prefix "SFW2-INext-DROP-DEFLT␣"
    --log-tcp-options --log-ip-options
-A input_ext -p icmp -j LOG --log-prefix
    "SFW2-INext-DROP-DEFLT␣" --log-tcp-options --log-ip-options
-A input_ext -p udp -m state --state NEW -j LOG --log-prefix
    "SFW2-INext-DROP-DEFLT␣" --log-tcp-options --log-ip-options
-A input_ext -j DROP
-A input_int -j LOG --log-prefix "SFW2-INint-ACC-ALL␣"
    --log-tcp-options --log-ip-options
-A input_int -j ACCEPT
-A reject_func -p tcp -j REJECT --reject-with tcp-reset
-A reject_func -p udp -j REJECT --reject-with
    icmp-port-unreachable
-A reject_func -j REJECT --reject-with icmp-proto-unreachable
COMMIT
# Completed on Wed Oct 13 23:14:25 2010
```

8.1.2 IPv6

```
# Generated by ip6tables-save v1.4.4 on Wed Oct 13 23:14:32
    2010
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -i lo -j NOTRACK
-A OUTPUT -o lo -j NOTRACK
COMMIT
# Completed on Wed Oct 13 23:14:32 2010
# Generated by ip6tables-save v1.4.4 on Wed Oct 13 23:14:32
    2010
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
```

```
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
COMMIT
# Completed on Wed Oct 13 23:14:32 2010
# Generated by ip6tables-save v1.4.4 on Wed Oct 13 23:14:32
    2010
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:forward_ext - [0:0]
:forward_int - [0:0]
:input_ext - [0:0]
:input_int - [0:0]
:reject_func - [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state ESTABLISHED -j LOG --log-prefix
    "SFW2-IN-ACC-EST␣" --log-tcp-options --log-ip-options
-A INPUT -m state --state ESTABLISHED -j ACCEPT
-A INPUT -p ipv6-icmp -m state --state RELATED -j LOG
    --log-prefix "SFW2-IN-ACC-REL␣" --log-tcp-options
    --log-ip-options
-A INPUT -p ipv6-icmp -m state --state RELATED -j ACCEPT
-A INPUT -i eth0 -j input_int
-A INPUT -j input_ext
-A INPUT -j LOG --log-prefix "SFW2-IN-ILL-TARGET␣"
    --log-tcp-options --log-ip-options
-A INPUT -j DROP
-A FORWARD -j LOG --log-prefix "SFW2-FWD-ILL-ROUTING␣"
    --log-tcp-options --log-ip-options
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -p ipv6-icmp -j ACCEPT
-A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -j LOG --log-prefix "SFW2-OUT-ERROR␣"
    --log-tcp-options --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 128 -j LOG
    --log-prefix "SFW2-INext-ACC-PING␣" --log-tcp-options
    --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 128 -j ACCEPT
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 133 -j LOG
    --log-prefix "SFW2-INext-ACC-ICMP␣" --log-tcp-options
    --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 133 -j ACCEPT
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 134 -j LOG
    --log-prefix "SFW2-INext-ACC-ICMP␣" --log-tcp-options
    --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 134 -j ACCEPT
```

```
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 135 -j LOG
    --log-prefix "SFW2-INext-ACC-ICMP␣" --log-tcp-options
    --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 135 -j ACCEPT
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 136 -j LOG
    --log-prefix "SFW2-INext-ACC-ICMP␣" --log-tcp-options
    --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 136 -j ACCEPT
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 137 -j LOG
    --log-prefix "SFW2-INext-ACC-ICMP␣" --log-tcp-options
    --log-ip-options
-A input_ext -p ipv6-icmp -m icmp6 --icmpv6-type 137 -j ACCEPT
-A input_ext -p tcp -m tcp --dport 22 -j LOG --log-prefix
    "SFW2-INext-ACC-TCP␣" --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 22 -j ACCEPT
-A input_ext -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -j
    LOG --log-prefix "SFW2-INext-DROP-DEFLT␣"
    --log-tcp-options --log-ip-options
-A input_ext -p ipv6-icmp -j LOG --log-prefix
    "SFW2-INext-DROP-DEFLT␣" --log-tcp-options --log-ip-options
-A input_ext -p udp -m state --state NEW -j LOG --log-prefix
    "SFW2-INext-DROP-DEFLT␣" --log-tcp-options --log-ip-options
-A input_ext -j DROP
-A input_int -j LOG --log-prefix "SFW2-INint-ACC-ALL␣"
    --log-tcp-options --log-ip-options
-A input_int -j ACCEPT
-A reject_func -p tcp -j REJECT --reject-with tcp-reset
-A reject_func -p udp -j REJECT --reject-with
    icmp6-port-unreachable
-A reject_func -j REJECT --reject-with icmp6-addr-unreachable
-A reject_func -j DROP
COMMIT
# Completed on Wed Oct 13 23:14:32 2010
```

8.2 iptables-Regelsatz lrzfw

8.2.1 IPv4

```
# Generated by iptables-save v1.4.4 on Mon Oct 11 13:04:44 2010
*filter
:INPUT DROP [2:156]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:BLACKLIST - [0:0]
:GLOBAL_RULES - [0:0]
:GLOBAL_RULES_000-SSHD_LOG - [0:0]
:LRZFW_ONLINE - [0:0]
:ZONE_000-LO - [0:0]
```

```
-A INPUT -j BLACKLIST
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m state --state INVALID -j DROP
-A INPUT -j GLOBAL_RULES
-A INPUT -i lo -j ZONE_000-LO
-A BLACKLIST -j RETURN
-A GLOBAL_RULES -p tcp -m recent --set --name
    GLOBAL_RULES_000-SSHD_LOG --rsource -m tcp --dport 22
    --tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW
-A GLOBAL_RULES -p tcp -m recent --update --seconds 60
    --hitcount 10 --name GLOBAL_RULES_000-SSHD_LOG --rsource
    -m tcp --dport 22 --tcp-flags FIN,SYN,RST,ACK SYN -m state
    --state NEW -j GLOBAL_RULES_000-SSHD_LOG
-A GLOBAL_RULES -p tcp -m tcp --dport 22 --tcp-flags
    FIN,SYN,RST,ACK SYN -m state --state NEW -j ACCEPT
-A GLOBAL_RULES -j RETURN
-A GLOBAL_RULES_000-SSHD_LOG -m hashlimit --hashlimit-upto
    1/min --hashlimit-burst 1 --hashlimit-mode srcip
    --hashlimit-name SSHD_HL_0 -j LOG --log-prefix "lrzfw:␣
    poss.␣ssh␣brutef.:␣"
-A GLOBAL_RULES_000-SSHD_LOG -j RETURN
-A ZONE_000-LO -j ACCEPT
-A ZONE_000-LO -j DROP
COMMIT
# Completed on Mon Oct 11 13:04:44 2010
```

8.2.2 IPv6

```
# Generated by ip6tables-save v1.4.4 on Mon Oct 11 13:04:48
    2010
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:BLACKLIST - [0:0]
:GLOBAL_RULES - [0:0]
:GLOBAL_RULES_000-SSHD_LOG - [0:0]
:LRZFW_ONLINE - [0:0]
:ZONE_000-LO - [0:0]
-A INPUT -j BLACKLIST
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m state --state INVALID -j DROP
-A INPUT -p ipv6-icmp -m icmp6 --icmpv6-type 133 -j ACCEPT
-A INPUT -p ipv6-icmp -m icmp6 --icmpv6-type 134 -j ACCEPT
-A INPUT -p ipv6-icmp -m icmp6 --icmpv6-type 135 -j ACCEPT
-A INPUT -p ipv6-icmp -m icmp6 --icmpv6-type 136 -j ACCEPT
-A INPUT -p ipv6-icmp -m icmp6 --icmpv6-type 137 -j ACCEPT
-A INPUT -j GLOBAL_RULES
```

```
-A INPUT -i lo -j ZONE_000-LO
-A BLACKLIST -j RETURN
-A GLOBAL_RULES -p tcp -m recent --set --name
    GLOBAL_RULES_000-SSHD_LOG --rsource -m tcp --dport 22
    --tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW
-A GLOBAL_RULES -p tcp -m recent --update --seconds 60
    --hitcount 10 --name GLOBAL_RULES_000-SSHD_LOG --rsource
    -m tcp --dport 22 --tcp-flags FIN,SYN,RST,ACK SYN -m state
    --state NEW -j GLOBAL_RULES_000-SSHD_LOG
-A GLOBAL_RULES -p tcp -m tcp --dport 22 --tcp-flags
    FIN,SYN,RST,ACK SYN -m state --state NEW -j ACCEPT
-A GLOBAL_RULES -j RETURN
-A GLOBAL_RULES_000-SSHD_LOG -m hashlimit --hashlimit-upto
    1/min --hashlimit-burst 1 --hashlimit-mode srcip
    --hashlimit-name SSHD_HL_0 -j LOG --log-prefix "lrzfw:␣
    poss.␣ssh␣brutef.:␣"
-A GLOBAL_RULES_000-SSHD_LOG -j RETURN
-A ZONE_000-LO -j ACCEPT
-A ZONE_000-LO -j DROP
COMMIT
# Completed on Mon Oct 11 13:04:48 2010
```

8.3 Dateiliste RPM-Paket

```
/etc/default/lrzfw
/etc/init.d/lrzfw
/etc/logrotate.d/99-lrzfw.conf
/etc/lrzfw
/etc/lrzfw/blacklist
/etc/lrzfw/rules-available
/etc/lrzfw/rules-available/allow-all.rule
/etc/lrzfw/rules-available/cifsd.rule
/etc/lrzfw/rules-available/ftpd.rule
/etc/lrzfw/rules-available/httpd.rule
/etc/lrzfw/rules-available/icmp-echo.rule
/etc/lrzfw/rules-available/iked.rule
/etc/lrzfw/rules-available/imapd.rule
/etc/lrzfw/rules-available/ipsec-ah.rule
/etc/lrzfw/rules-available/ipsec-esp.rule
/etc/lrzfw/rules-available/ldapd.rule
/etc/lrzfw/rules-available/nfs4d.rule
/etc/lrzfw/rules-available/pop3d.rule
/etc/lrzfw/rules-available/skel
/etc/lrzfw/rules-available/smtpd.rule
/etc/lrzfw/rules-available/sshd.rule
/etc/lrzfw/rules-enabled
/etc/lrzfw/rules-enabled/000-sshd.rule
```

```
/etc/lrzfw/zones
/etc/lrzfw/zones/000-lo-rules
/etc/lrzfw/zones/000-lo-rules/allow-all.rule
/etc/lrzfw/zones/000-lo.zone
/etc/lrzfw/zones/100-mwn.zone.disabled
/etc/lrzfw/zones/skel
/etc/rsyslog.d/99-lrzfw.conf
/usr/sbin/lrzfw
/usr/share/doc/packages/lrzfw
/usr/share/doc/packages/lrzfw/INSTALL
/usr/share/doc/packages/lrzfw/README
/usr/share/lrzfw/lrzfw_const
/usr/share/lrzfw/lrzfw_counter
/usr/share/lrzfw/lrzfw_environ
/usr/share/lrzfw/lrzfw_helper
/usr/share/lrzfw/lrzfw_ipv4
/usr/share/lrzfw/lrzfw_ipv6
/usr/share/lrzfw/lrzfw_matches
/usr/share/lrzfw/lrzfw_mixed
/usr/share/lrzfw/lrzfw_rules
/usr/share/lrzfw/lrzfw_zones
/usr/share/lrzfw/matches
/usr/share/lrzfw/matches/latest-ipv4.txt
/usr/share/lrzfw/matches/latest-ipv6.txt
/usr/share/lrzfw/matches/v1.3.0-ipv4.txt
/usr/share/lrzfw/matches/v1.3.0-ipv6.txt
/usr/share/lrzfw/matches/v1.3.1-ipv4.txt
/usr/share/lrzfw/matches/v1.3.1-ipv6.txt
/usr/share/lrzfw/matches/v1.3.2-ipv4.txt
/usr/share/lrzfw/matches/v1.3.2-ipv6.txt
/usr/share/lrzfw/matches/v1.3.3-ipv4.txt
/usr/share/lrzfw/matches/v1.3.3-ipv6.txt
/usr/share/lrzfw/matches/v1.3.4-ipv4.txt
/usr/share/lrzfw/matches/v1.3.4-ipv6.txt
/usr/share/lrzfw/matches/v1.3.5-ipv4.txt
/usr/share/lrzfw/matches/v1.3.5-ipv6.txt
/usr/share/lrzfw/matches/v1.3.6-ipv4.txt
/usr/share/lrzfw/matches/v1.3.6-ipv6.txt
/usr/share/lrzfw/matches/v1.3.7-ipv4.txt
/usr/share/lrzfw/matches/v1.3.7-ipv6.txt
/usr/share/lrzfw/matches/v1.3.8-ipv4.txt
/usr/share/lrzfw/matches/v1.3.8-ipv6.txt
/usr/share/lrzfw/matches/v1.4.0-ipv4.txt
/usr/share/lrzfw/matches/v1.4.0-ipv6.txt
/usr/share/lrzfw/matches/v1.4.1-ipv4.txt
/usr/share/lrzfw/matches/v1.4.1-ipv6.txt
/usr/share/lrzfw/matches/v1.4.2-ipv4.txt
```

```
/usr/share/lrzfw/matches/v1.4.2-ipv6.txt
/usr/share/lrzfw/matches/v1.4.3-ipv4.txt
/usr/share/lrzfw/matches/v1.4.3-ipv6.txt
/usr/share/lrzfw/matches/v1.4.4-ipv4.txt
/usr/share/lrzfw/matches/v1.4.4-ipv6.txt
/usr/share/lrzfw/matches/v1.4.5-ipv4.txt
/usr/share/lrzfw/matches/v1.4.5-ipv6.txt
/usr/share/lrzfw/matches/v1.4.6-ipv4.txt
/usr/share/lrzfw/matches/v1.4.6-ipv6.txt
/usr/share/lrzfw/matches/v1.4.7-ipv4.txt
/usr/share/lrzfw/matches/v1.4.7-ipv6.txt
/usr/share/lrzfw/matches/v1.4.8-ipv4.txt
/usr/share/lrzfw/matches/v1.4.8-ipv6.txt
/usr/share/lrzfw/matches/v1.4.9-ipv4.txt
/usr/share/lrzfw/matches/v1.4.9-ipv6.txt
/usr/share/man/man8/lrzfw.8.gz
/var/adm/backup/lrzfw
```

8.4 rsyslogd

```
:msg,contains,"lrzfw:" -/var/log/lrzfw.log
& ~
```

8.5 logrotate

```
/var/log/lrzfw.log
{
        rotate 7
        daily
        missingok
        notifempty
        delaycompress
        compress
        postrotate
                if [ -x /etc/init.d/syslog ]; then
                        /etc/init.d/syslog reload > /dev/null
                fi
                if [ -x /etc/init.d/rsyslog ]; then
                        /etc/init.d/rsyslog reload > /dev/null
                fi
        endscript
}
```

Abbildungsverzeichnis

Literaturverzeichnis

[apaa] *Apache Webserver*. http://httpd.apache.org.

[apab] *Apache Webserver Documentation*. http://httpd.apache.org/docs/current/mod/core.html#include.

[arn] *Arno's IPTABLES Firewall Script*. http://freshmeat.net/projects/iptables-firewall/.

[bas] *bash manual*. http://linux.die.net/man/1/bash.

[deb] *Debian*. http://www.debian.org/.

[ebt] *ebtables - Linux Ethernet Bridge Firewall*. http://ebtables.sourceforge.net/.

[fer] *ferm*. http://ferm.foo-projects.org/.

[fir] *FireHOL*. http://firehol.sourceforge.net/.

[Haz08] Hazrat, L.: *Virtuelle Firewalls im MWN: Dokumentation, Standard–Regelsatz und Aufbereitung von Router–ACLs für den Einsatz in virtuellen Firewalls*. Diplomarbeit, Ludwig–Maximilians–Universität München, 2008.

[iee] *IEEE 802.3 Ethernet Working Group*. http://www.ieee802.org/3/.

[ipta] *ip6tables manual*. http://linux.die.net/man/8/ip6tables.

[iptb] *iptables manual*. http://linux.die.net/man/8/iptables.

[iptc] *Netfilter und iptables*. http://www.netfilter.org/.

[log] *logrotate manual*. http://linux.die.net/man/8/logrotate.

[lrz] *Leibniz Rechenzentrum*. http://www.lrz.de.

[lsb] *Linux Standard Base*. http://www.linuxbase.org/.

[ope] *openSUSE Linux*. http://de.opensuse.org/.

[osi94] *ISO/OSI-Referenzmodell (ISO 7498)*, 1994. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[rfc99] *RFC 2644: Changing the Default for Directed Broadcasts in Routers*, 1999. http://tools.ietf.org/html/rfc2644.

[sho] *Shorewall*. http://www.shorewall.net/.

[slea] *SUSE Linux Enterprise Server (SLES)*. http://www.novell.com/de-de/linux/.

[sleb] *SUSE Linux Enterprise Server (SLES) Lifecycle*. `http://support.novell.com/lifecycle/`.

[Tan03] Tanenbaum, Andrew S.: *Computernetwerke*. Pearson Education Deutschland GmbH, 4. Auflage, 2003.

[tsm] *IBM Tivoli Storage Manager (TSM)*. `http://www.ibm.com/software/tivoli/products/storage-mgr/`.

[twr] *TCP-Wrapper manual*. `http://linux.die.net/man/5/hosts_access`.

[vuu] *Vuurmuur*. `http://www.vuurmuur.org/`.