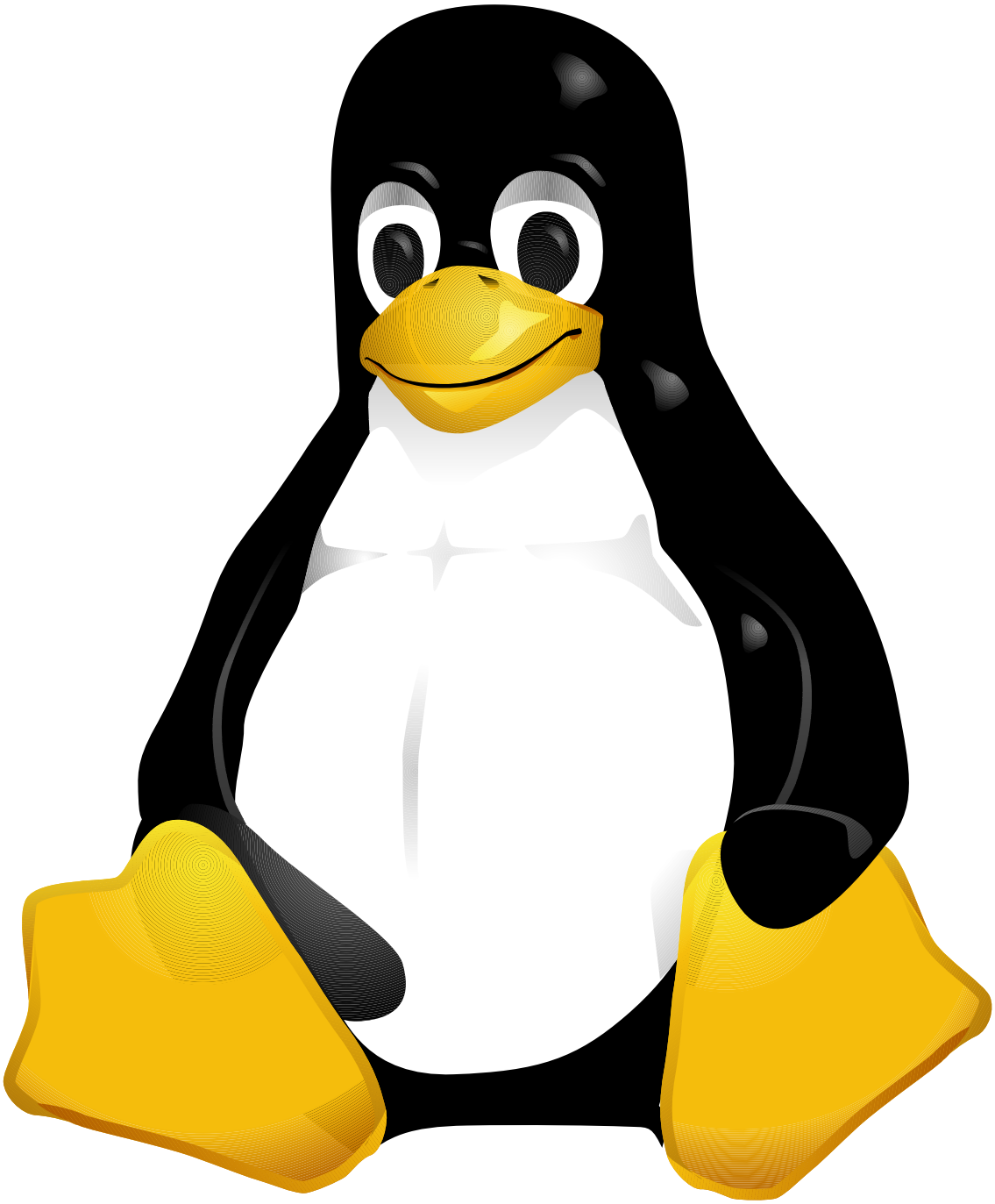
Technische Praxis der Computersysteme

Teil 2-2

Netzwerkadministration
unter Linux/Unix

Roland Steinbauer, Andreas Nemeth, Martin Piskernig,
Gerald Teschl, Florian Wisser

Version 1.01, Februar 2003



Inhaltsverzeichnis

Literatur

[9] Charles Aulds. *Linux Apache Web Server Administration.* Sybex Inc. 2000.

[10] Bryan Costales, Eric Allman. *Sendmail.* 2nd Edition. O'Reilly & Associates Inc. Sebastopol, CA. 1997.

[11] Robert Eckstein, David Collier-Brown. *Using Samba.* O'Reilly & Associates Inc. Sebastopol, CA. 1999.

[12] Craig Hunt. *TCP/IP Netzwerk Administration.* 2nd Edition. O'Reilly & Associates Inc. Sebastopol, CA. 1997.

[13] Craig Hunt. *Linux Sendmail Administration.* Sybex Inc. 2001.

[14] Mohamet J. Kabir. *Apache Server Administrator's Handbook.* Hungry Minds. 1999.

[15] W. R. Stevens. *TCP/IP Illustrated, Vol. 1: The Protocols.* Addison Wesley Longman Publishing. 1994.

13 TCP/IP-Grundlagen

Dieses Kapitel ist eine komprimierte Einführung in die Terminologie und die grundlegende Funktionsweise der TCP/IP-Protokollfamilie. Aus diesem großen Gebiet erklären wir gerade die Begriffe und Konzepte, die in den weiteren Kapiteln verwendet werden und deren Verständnis unbedingte Voraussetzung ist. Darüberhinaus verweisen wir auf die vielen guten Bücher zum Thema und insbesondere auf [12] und [15].

Eines der Zeichen, das den großen Erfolg von Computernetzwerken (diese sind mit großer Geschwindigkeit weit über ihre ursprünglich geplanten Dimensionen hinausgewachsen) im letzten Jahrzehnt begleitet, ist die Verwirrung, die sich um den Begriff „*Internet*“ rankt. Ursprünglich war damit ein Netzwerk gemeint, das auf dem sog. Internet Protokoll aufbaut. Mittlerweile bezeichnet dieser Ausdruck entweder eine Kollektion von verschiedenen physikalischen Netzwerken, die durch ein gemeinsames Protokoll verbunden werden, oder gleich das (ganze) weltweite Computernetzwerk.

Wir besprechen hier die Grundlagen des Internet Protokolls, eigentlich *Transmission Control Protocol/Internet Protocol (TCP/IP)*. In der Netzwerkterminologie bezeichnet „Protokoll“ einen Satz von Regeln, Standards und Vorschriften für die Kommunikation im Netzwerk. (Für eine kleine Begriffserklärung siehe Folie 149.)
Genaugenommen ist TCP/IP nicht nur ein einziges Protokoll, sondern eine ganze Familie von aufeinander abgestimmten und miteinander verbundenen Protokollen. TCP/IP zeichnet sich vor allem durch folgenden Merkmale aus, die auch ein Grund für seine weite Verbreitung sind: *Offene Protokoll Standards*, die unabhängig von Rechner- und Netzwerk-Hardware und Betriebssystem festgelegt und frei zugänglich sind, ein *globales Adressierungssystem*, das eine eindeutige Adressierung aller Hosts (selbst im weltweiten Internet) ermöglicht und die *Standardisierten High-Level-Protokolle* für bequeme, konsistente und verlässliche Anwendungen (siehe auch Folie 150).

Netzwerkterminologie

- Protokoll: Satz von Regeln und Kommunikationsstandards
- TCP/IP: Transport Control Protocol/Internet Protocol Protokollfamilie
- Host: Rechner in einem TCP/IP-basierten Netzwerk
- Internet: Netzwerk von TCP/IP basierten Netzwerken
- Gateway/Router: physikalisch an mehrere Netzwerke angebundener Host

Folie 149

TCP/IP-Features

- offene, frei zugängliche Standards
- hardwareunabhängig (Netzwerk, Host)
- softwareunabhängig
- globales Adressierungssystem
- High-Level-Protokolle

Folie 150

Die TCP/IP-Standards werden von verschiedenen Organisationen (siehe Folie 151) weiterentwickelt und in frei zugänglichen Dokumenten, den sogenannten *RFC's (Requests for Comment)* veröffentlicht. Diese Organisationen übernehmen auch wesentliche Verwaltungsaufgaben für das weltweite Netz, vor allem im Bereich der Adressierung und Namensvergabe.
Jeder Softwarehersteller oder Programmierer kann nach eigenem Belieben TCP/IP-Protokolle in sein Produkt/Betriebssystem implementieren; die erste Implementation von TCP/IP in ein Betriebssystem erfolgte 1983 in BSD-Unix.

13.1 Die TCP/IP-Protokollfamilie

Wir geben nun einen schematischen, kurzen Überblick über die interne Strukturierung der TCP/IP- Protokollfamilie. Unser Ziel ist es, ein grundlegendes Verständnis für die Kommunikationsprozesse zu vermitteln, ohne zu sehr auf Details einzugehen.

Um die Kommunikation nicht nur von Host zu Host, sondern von einer spezifischen Anwendung, die ein bestimmter Benutzer auf einen Host ausführt mit einer anderen Anwendung, die ein anderer Benutzer auf einem anderen Host ausführt (inklusive verlässlicher Datenübertragung) zu ermöglichen, ist ein komplexes Bündel von Kommunikationsstrukturen nötig. Üblicherweise wird ein Schichtenmodell zu Grunde gelegt, wobei die verschiedene Aufgaben jeweils verschiedenen Schichten (Layer) zugewiesen sind. Das sogenannte *OSI-Referenzmodell*, das grundsätzlich den Standard für eine derartige Kommunikationsarchitektur festschreibt, ist in 7 Schichten organisiert.

Die TCP/IP-Protokollfamilie ist nicht in allen Punkten OSI-konform und wird meist vereinfacht in einem 4-Schichten-Modell beschrieben. Die Daten einer Anwendung, die über das Netzwerk transportiert werden sollen, werden beginnend von der höchsten Schicht (*Application Layer*) mit Kontrollinformation (in einem *Header*) versehen an die darunterliegende Schicht weitergegeben. In dieser werden den Datenpaketen gemäß den in der Schicht beheimateten Protokollen weitere Kontrolldaten (in einem weiteren Header) vorangestellt und wiederum an die nächst tiefere Schicht weitergegeben; man spricht von *Encapsulation*: Am unteren Ende des *Stacks* (Stapels) befindet sich der

Network Access Layer, von dem aus die Daten auf das physikalische Netz weitergeleitet werden.
Die Datenpakete werden dann über das Netzwerk bis zum Empfängerhost geleitet und dort zunächst vom Network Access Layer empfangen. Dann werden sie „entpackt“ und an den darüberliegenden Layer weitergegeben, bis sie schließlich im Application Layer angekommen sind und an die entsprechende Anwendung weitergeleitet werden. Die vier Schichten des TCP/IP-Stacks sind auf Folie 152 dargestellt.

Organisationen und Standards

- IANA (Internet Assigned Number Authority): IP-Adr., Portnr.
- InterNIC: Registrierung von Domainnamen
- IAB (Internet Advisory Board): policy
- IETF (Internet Engineering Task Force)
- IRTF (Internet Research Task Force)
- RFCs (Requests for Comment): Festlegung der Standards, Dokumentation

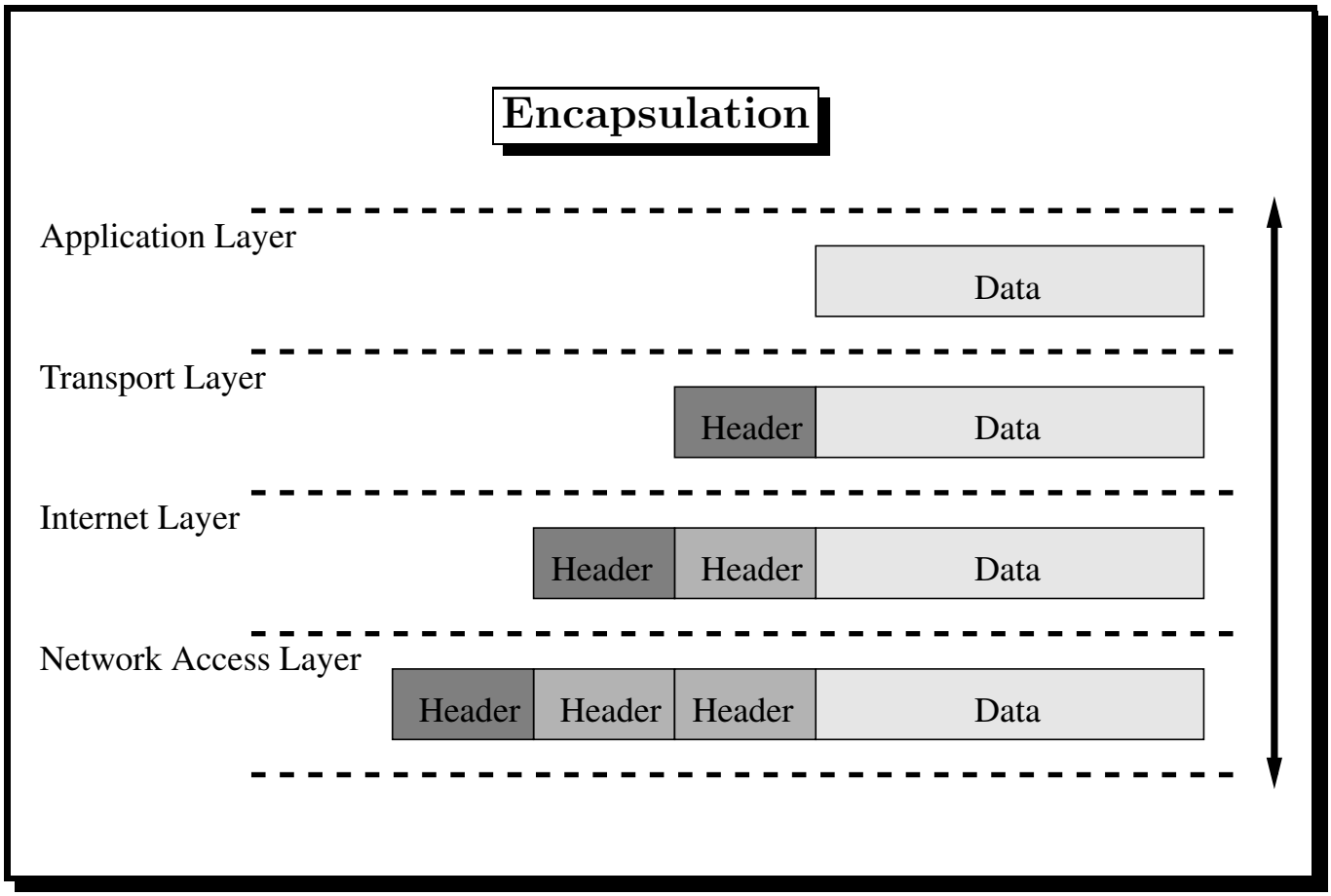
Folie 151

TCP/IP-Layer

Layer	Aufgabe	Protokolle
4. Application Layer	End-User Anw.	FTP, TELNET,...
3. Transport Layer	Host-to-Host Komm.	TCP, UDP,...
2. Internet Layer	Adressierung, Routing	IP, ICMP,...
1. Network Access L.	Zugang zum Netz	ETHERNET, ARP,...

OSI-Referenzmodell: 7 Schichten; TCP/IP: 4 Schichten

Folie 152


Encapsulation
Application Layer
Data
Transport Layer
Header
Data
Internet Layer
Header
Header
Data
Network Access Layer
Header
Header
Header
Data

Folie 153

13.2 TCP/IP-Layer

Wir besprechen nun die einzelnen Layer und die wichtigsten damit assoziierten Protokolle. Der **Network Access Layer** (NAL) ist als unterste Schicht im TCP/IP-Stack für den Zugriff auf das physikalische Netzwerk verantwortlich. Es ist der einzige Layer, der Informationen über die Details der Netzwerkhardware und die Art des Netzwerks (Ethernet, Token Ring, Serielle/Telefon-Leitung, etc.) benötigt. In der Unix-Implementation ist der NAL meist eine Kombination aus (Netzwerkkarten-) Treibersoftware und Software, die zB IP-Adressen auf Ethernet-Adressen umschreibt (ARP Protokoll, siehe Abschnitt 13.4). Die Datenpakete, die vom NAL auf das Netzwerk ausgehen, heißen *Fragments*.

Der **Internet Layer** (IL) bewerkstelligt die grundlegende Datenpaketzustellung. Das wichtigste Protokoll im IL, das Internet Protocol (IP), ist gleichzeitig das Herzstück des TCP/IP-Stacks. Alle Protokolle der anderen Schichten benutzten das IP zum Datentransport; alle ein- und ausgehenden Daten fließen durch das IP, unabhängig von ihrem Quell- bzw. Bestimmungsort.
Einige Aufgaben des IP sind die Definition des *Datagrams*, der Basisdatenübertragungseinheit im Internet sowie des Adressierungsschemas und das *Routen* der Datagrams. Auf den Folien 156 und 157 ist das IP Datagram Format und das Routen von Datagrams dargestellt.
Die ersten 5 (oder 6 je nach Festlegung im Internet Header Field (IHL)) 32-Bit Worte des *Datagrams* bilden den Header, danach folgen die Daten. Der Header enthält alle Informationen, um die Daten richtig transportieren zu können. Die *Destination Address* im 5. Wort ist eine 32-Bit-Nummer, die eindeutig Zielnetzwerk und Zielhost festlegt (zB 131.130.145.110, siehe 13.4). Befindet sich der Zielhost im lokalen Netzwerk (also im selben physikalischen Netzwerk wie der Source Host), so können die Daten direkt übertragen werden (mittels NAL und der Auflösung der IP-Adressen in Hardwareadressen). Befindet sich der Zielhost in einem anderen Netzwerk, so muss das entsprechende Datenpaket über ein oder mehrere Gateways/Router geleitet werden. (Gatway vs Router; die Terminologie ist hier nicht einheitlich. Manchmal wird Gateway als Bezeichnung für einen Host verwendet, der Daten zwischen verschiedenen Protokollen vermit-

telt; dann bezeichnet nur Router das, was wir hier Gateway nennen. Wir verwenden aber beide Begriffe synonym.)

Jeder Host „kennt“ (per Konfiguration) sein Gateway, das die Verbindung vom lokalen Netzwerk (Local Area Network, LAN) zum übergeordneten Netzwerk herstellt. Im Beispiel auf Folie 157 sollen Daten von Host 1 zum Host 2 übertragen werden. Host 1 sendet die Daten als Fragment über das lokale Token-Ring-Netzwerk an Gateway 1. Dessen NAL übernimmt das Fragment und leitet die Daten entpackt zum Datagram an den IL weiter. Dieser entscheidet dann (aufgrund seiner Konfiguration), wie das Paket weiter zu routen ist und sendet es an das entsprechende Gateway. Dieses kann entweder schon das für den Host 2 zuständige Gateway 2 sein (falls im selben physikalischen Netzwerk) oder ein dem Gateway 1 übergeordnetes Gateway. Nach möglicherweise vielen Zwischenschritten gelangt das Fragment schließlich zu Gateway 2. Dessen IL stellt dann (nach Entpacken des Fragments zum Datagram) fest, dass die Daten an Host 2 zu senden sind und schickt das entsprechende Frame über das lokale Ethernet an Host 2. Wichtige Tatsache ist, dass weder Host 1 noch Host 2 irgendeine Information über die Netzwerke jenseits ihrer Gateways benötigen (und auch nicht darüber verfügen).
In der Unix-Implementierung kann man mittels des Kommandos `traceroute host` die Route zum entsprechenden Host abfragen; es wird eine Liste der dabei verwendeten Gateways und der Laufzeit zwischen ihnen ausgegeben.

Wenn der IL ein Datagram erhält, das für den lokalen Host bestimmt ist, so wird dieses an den darüberliegenden Transport Layer weitergereicht, wobei es direkt an das entsprechende Protokoll im TL überstellt wird, das im 3. Wort des IP-Datagrams eingetragen ist.

Network Access Layer

- unterste Schicht im TCP/IP-Stack
- physikalischer Zugriff auf Netzwerk
- Kombination
  - Netzwerktreiber-Software
  - IP-MAC-Adressenauflösung (ARP-Protokoll)
- Daten aufs Netz als Fragments

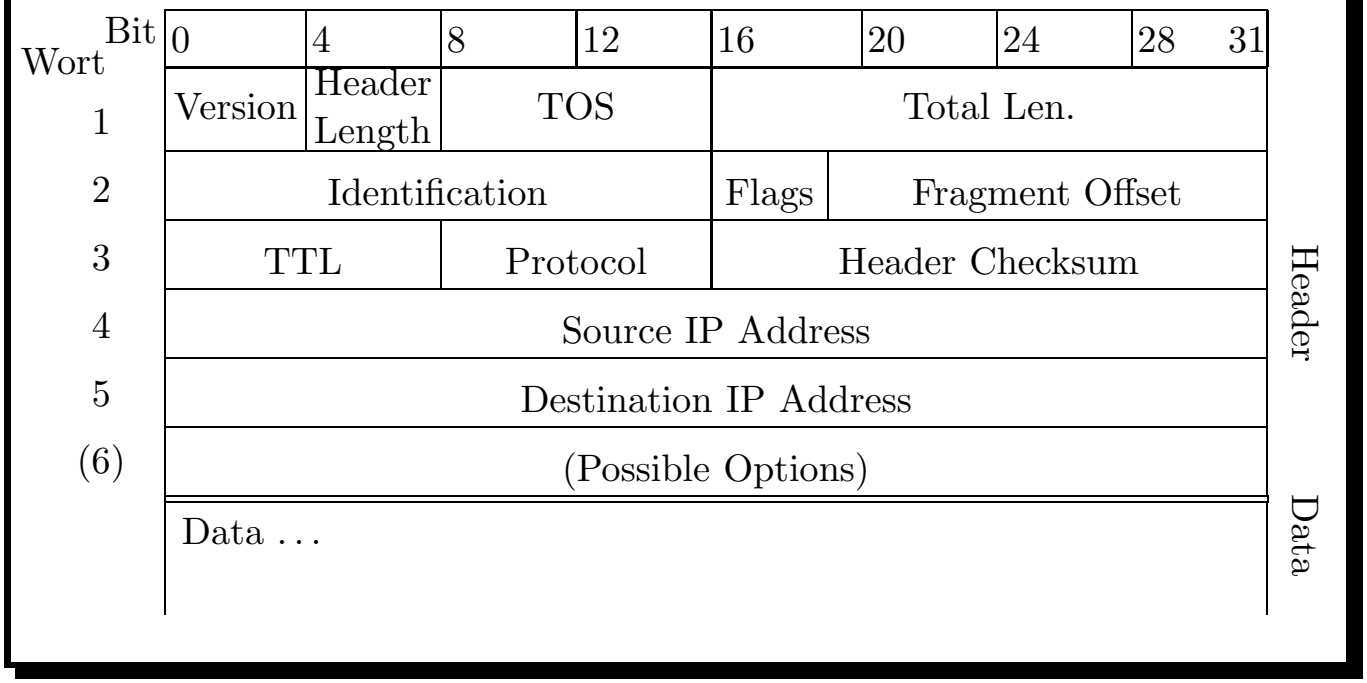
Folie 154

Internet Layer

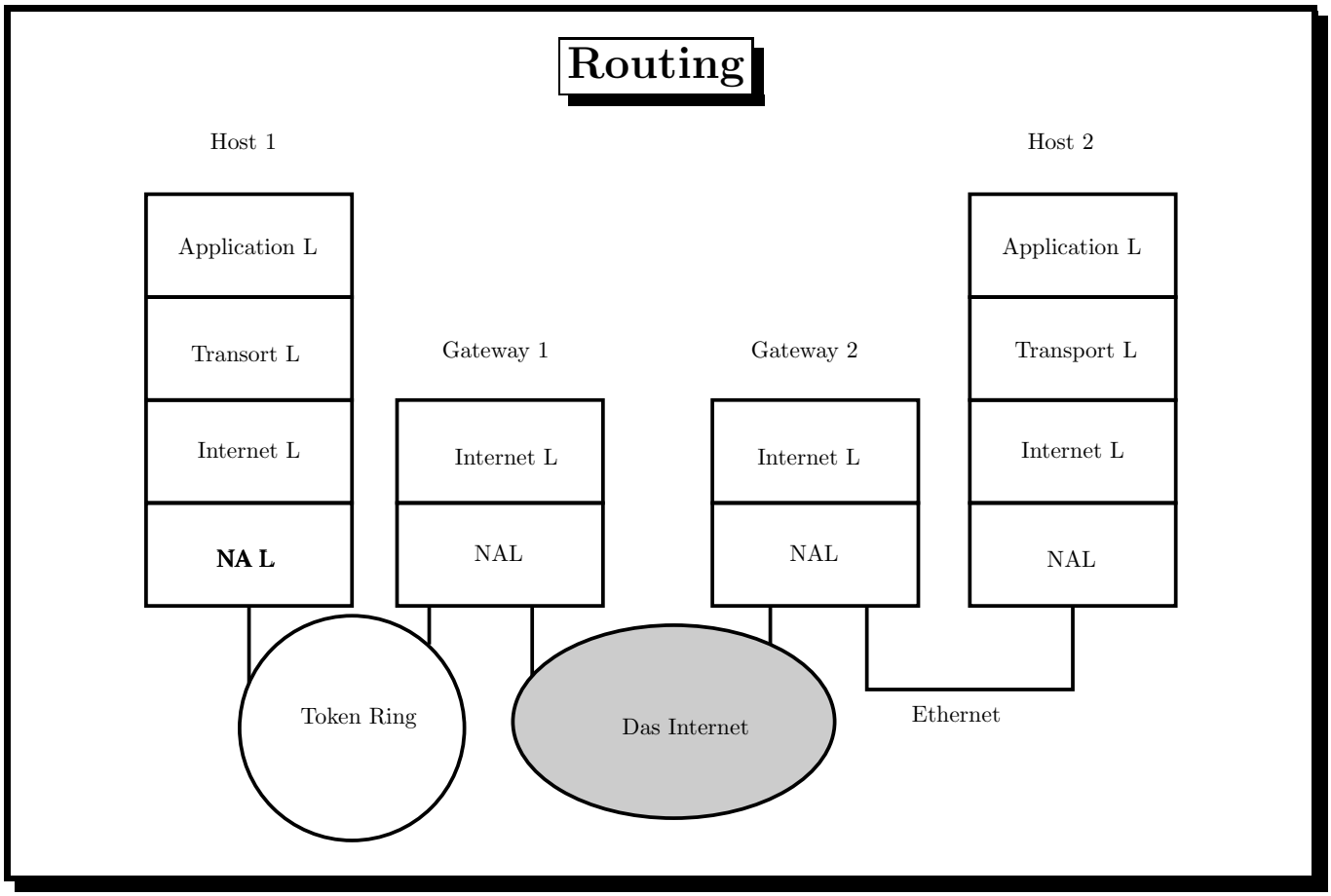
- grundlegende Datenzustellung
- wichtigstes Protokoll: Internet Protocol (IP)
  - definiert Datagram als Standard-Übertragsungseinheit
  - IP-Adressierungsschemas
  - Routen von Datagrams
- Internet Control Message Protocol (ICMP)

Folie 155

IP-Datagram



Folie 156


Routing
Host 1
Host 2
Application L
Transort L
Internet L
NA L
Gateway 1
Internet L
NAL
Gateway 2
Internet L
NAL
Application L
Transport L
Internet L
NAL
Token Ring
Das Internet
Ethernet

Folie 157

Ein weiteres Protokoll innerhalb des IL ist das *Internet Control Message Protocol (ICMP)*. Es definiert die Standards für Flusskontrolle, Fehlermeldungen und wichtige Informationsfunktionen von TCP/IP. Ein wichtiger Teil von ICMP ist das Überprüfen der Erreichbarkeit von Hosts mittels einer *Echo Message*; dies wird zB vom `ping`-Kommando benutzt.

Der **(Host-to-Host) Transport Layer** (TL) liegt im TCP/IP-Stack über dem IL. Seine beiden wichtigsten Protokolle sind das *Transmission Control Protocol (TCP)* und das *User Datagram Protocol (UDP)*. TCP stellt eine verlässliche Datenübertragung mit Fehlererkennung und Korrektur zu Verfügung; UDP bietet eine einfache, nicht verbindungsorientierte, nicht verlässliche Paketübertragung mit wenig Overhead. Der Programmierer einer Anwendung kann – je nach Anforderungen – entscheiden, welches der Protokolle er verwendet. Sollen zB viele kleine Datenpakete übertragen werden, so ist der Aufwand mittels TCP mit seinem Kontroll-Overhead eine verlässliche Verbindung herzustellen größer, als einfach ein verlorenes Paket neu zu übermitteln, wie bei Übertragung mittles UDP-Protokolls vorgesehen.

UDP übernimmt vom Application Layer Daten in Form einer *Message* und verpackt sie zu einem *Packet*. Im ersten Wort des Headers stehen *Source* und *Destination Port*, die jeweils das Protokoll (resp. die Anwendung) im Application Layer definieren, dem die Message übergeben werden soll bzw. von dem sie herrührt. Das UDP Packet Format ist auf Folie 160 dargestellt. Generell können nur solche Anwendungen/Protokolle im AL UDP verwenden, die selbst für einen verlässlichen Datentransfer sorgen und nicht darauf angewiesen sind, die Verlässlichkeit mittels TL-Protokolls zu erzielen. Beispiele dafür ist das sogenannte *Query-Response*-Modell. Das Einlangen einer Antwort (Response) wird als Zeichen dafür gewertet, dass die Anfrage (Query) richtig übertragen wurde. Wird nach einer Timeout-Periode keine Antwort verzeichnet, so wird einfach die Anfrage erneut übertragen.

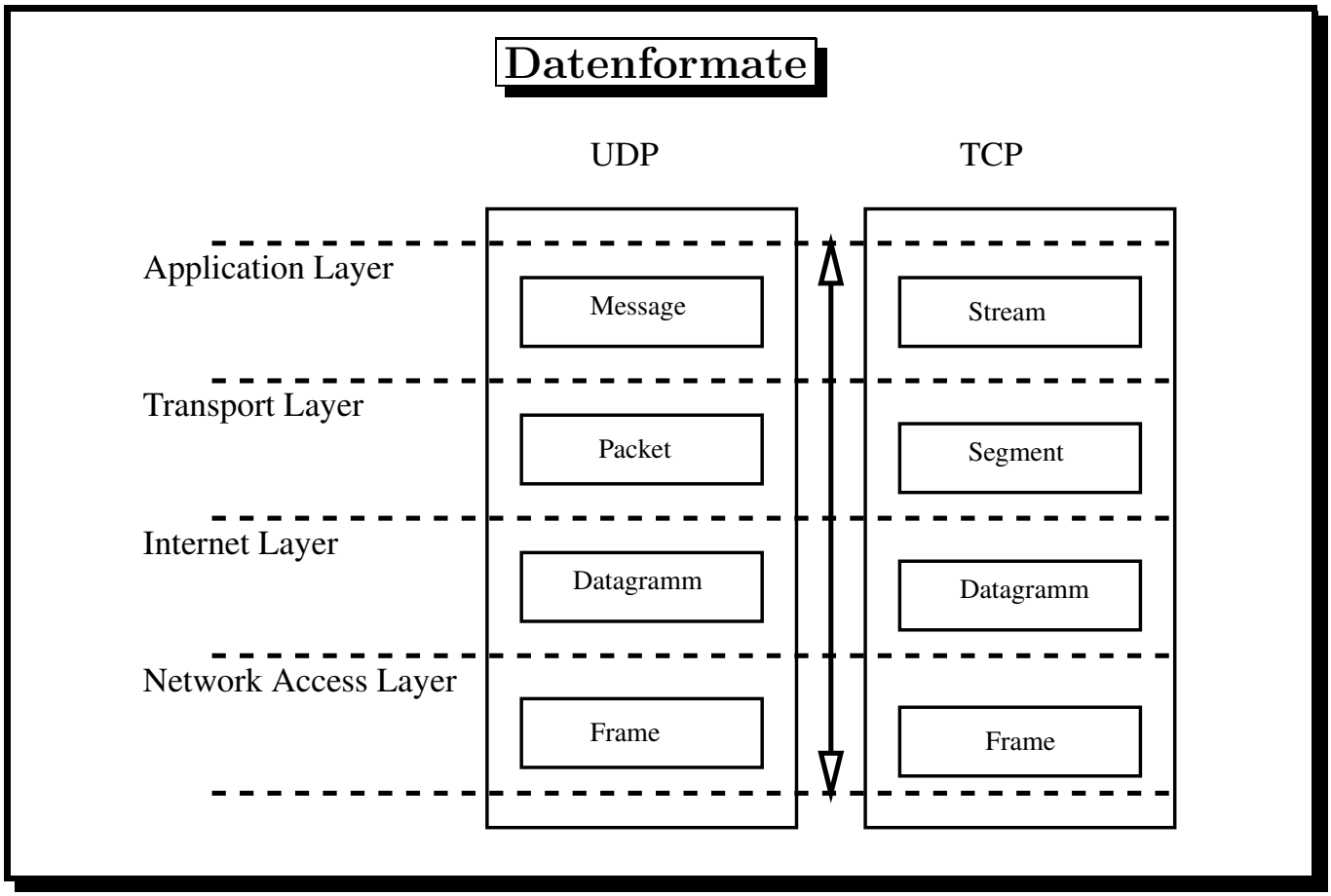
Im Unterschied zu UDP ist TCP ein *verlässliches, verbindungsorientiertes Byte-Stream-Protokoll.* Es übernimmt Daten aus dem AL in Form eines *Streams* und gibt diese als *Segment* an den IL weiter. Das TCP-Segment-Format ist schematisch auf Folie 161 dargestellt. Die Verlässlichkeit der Datenübertragung wird mittels eines *Positive Acknowledgement with Re-Transmission (PAR)* genannten Mechanismus hergestellt. Der Segment Header enthält eine Prüfsumme, die es dem Empfänger ermöglicht festzustellen, ob die Übertragung korrekt verlaufen ist. In diesem Falle schickt er ein *positives Acknowledgement* zurück an den Sender. Bleibt dieses nach einem Timeout aus, so überträgt der Sender das Segment erneut.
Die Verbindungsorientiertheit von TCP besteht darin, dass zu jeder Datenübertragung eine logische *End-to-End*-Verbindung aufgebaut wird. Diese wird durch einen sogenannten *Three-Way-Handshake* initialisiert, bei dem Sender und Empfänger Details des Kommunikationsablaufs festlegen (Synchronize Sequence Numbers (SYN-Flag gesetzt) und Acknowledgement (ACK-Flag gesetzt)). Jede Verbindung wird ebenfalls durch einen Three-Way-Handshake beendet (FIN). Im TCP werden die Daten innerhalb einer Verbindung nicht als individuelle Pakete angesehen, sondern als Datenstrom (Byte-Stream); dieser wird mittels *Sequence* und *Acknowledgement Number* kontrolliert. Mittels des *Acknowledgement Segments (ACK)* wird neben dem positiven Acknowledgement auch die Flusskontrolle durchgeführt. Die Kommunikationspartner teilen sich so mit, wie viel Daten sie im Moment noch aufnehmen können (Window Size). Analog zu UDP definieren Destination- und Source Portnummern im 1. Wort des TCP-Segments die jeweiligen Protokolle im AL für die Daten (Stream) übermittelt werden.

Der TL verfügt über weitere Protokolle, die in der Unix-Implementation von TCP/IP in der Datei `/etc/protocols` definiert sind. Eine Beispieldatei befindet sich auf Folie 162.

(Host-to-Host) Transport Layer

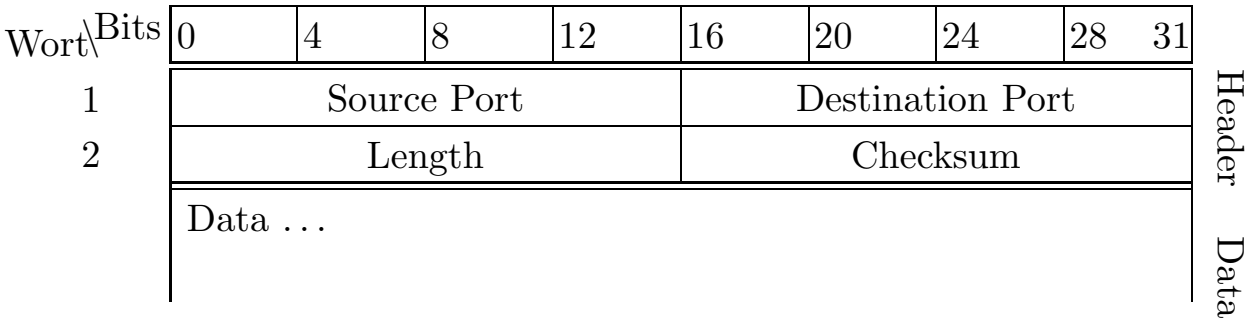
- Host-to-Host Kommunikation
- Transmission Control Protocol (TCP)
  - verbindungsorientiert, verlässlich
  - Fehlerkontrolle und -korrektur
  - Flusskontrolle
  - Format: Segment
- User Datagram Protocol (UDP)
  - nicht verbindungsorientiert
  - keine Fehlerkontrolle
  - wenig Overhead
  - Format: Packet
- Protokollnummern in `/etc/protocols` (TCP=6, UDP=17, ...)

Folie 158


Datenformate
UDP
TCP
Application Layer
Transport Layer
Internet Layer
Network Access Layer
Message
Packet
Datagramm
Frame
Stream
Segment
Datagramm
Frame

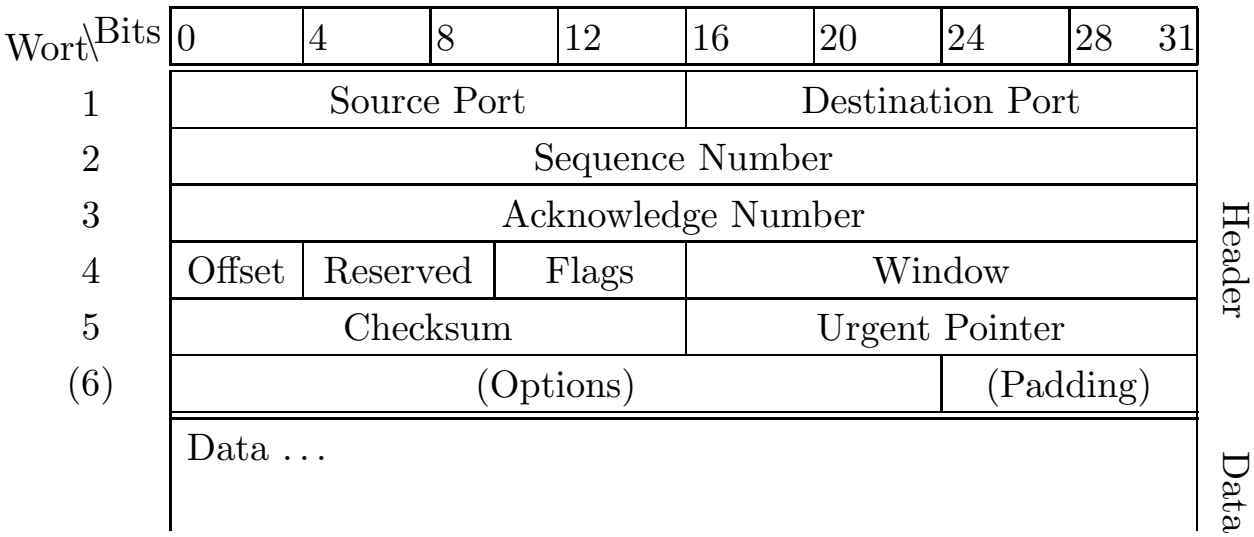
Folie 159

UDP-Packet



Folie 160

TCP-Segment



Folie 161

Protokolle im Transport Layer

```
bash# cat /etc/protocols
# /etc/protocols:
# Internet (IP) protocols
# from: @(#)protocols 5.1 (Berkeley) 4/17/89
ip   0  IP   # internet protocol, pseudo protocol nmb
icmp 1  ICMP # internet control message protocol
igmp 2  IGMP # Internet Group Management
tcp  6  TCP  # transmission control protocol
pup  12 PUP  # PARC universal packet protocol
udp  17 UDP  # user datagram protocol
hmp  20 HMP  # host monitoring protocol
ipip 94 IPIP # Yet Another IP encapsulation
ipip 94 IPIP # Yet Another IP encapsulation
```

Folie 162

Der **Application Layer** (AL) ist die oberste Schicht in unserem Modell und besteht aus einer Vielzahl verschiedener Protokolle, von denen die meisten bereits direkt User-Services, also Anwendungen sind. Neue Services werden laufend zum AL hinzugefügt. Wir bringen eine kurze Aufzählung der wichtigsten AL-Protokolle.

- TELNET (Network Terminal Protocol) Remote Login über ein Netzwerk, siehe Abschnitt 15.3
- FTP (File Transfer Protocol) Interaktiver Filetransfer, siehe Abschnitte 15.3 und 17.1
- SMTP (Simple Mail Transfer Protocol) E-Mail senden und empfangen, siehe Abschnitt 17.3
- SSH (Secure Shell) Sicherer Remote Login und Filetransfer, siehe Abschnitt 15.3
- POP3 (Post Office Protocol) E-Mail abholen
- IMAP2 (Interim Mail Access Protocol) E-Mail abholen
- PRINTER Netzwerkdrucker (lpd), vgl. Kapitel 7
- HTTP (Hyper Text Transfer Protocol), www, siehe Abschnitt 17.2
- NFS (Network File System), siehe Abschnitt 16.2
- NIS (Network Information System), siehe Abschnitt 16.3
- SUNRPC (Portmapper, Remote Procedure Call), siehe Abschnitt 16.1
- NETBIOS Windows Netzwerk Protokoll, siehe Kapitel 19
- DNS (Domain Name Service) Auflösen von IP-Adressen bzw. Hostnamen, siehe Abschnitt 13.4
- ...

Die Protokolle in AL werden vom TCP/IP-Stack aus über die *Portnummern* angesprochen. In der Unix-Implementation werden diese in der Datei `/etc/services` definiert. Diese Datei enthält neben den Portnummern auch Information darüber, welche TL-Protokolle von den entsprechenden Services verwendet werden. Eine Beispieldatei befindet sich auf Folie 164.

Application Layer

- oberste Schicht im TCP/IP-Stack
- End-User Applikationen
- Portnummern in `/etc/services`
  - 1-255: „well-known services“
    - TELNET
    - FTP
    - SMTP
    - POP
    - HTTP,...
  - 256-1024: „Unix specific services“
    - NFS
    - PRINTER
    - TALK, ...
  - 1024–: unprivilegierte Ports (dynamically allocated)

Folie 163

Ports und Services

```
bash# cat /etc/services:
tcpmux        1/tcp       # TCP port service multiplexer
echo          7/tcp
echo          7/udp
ftp-data     20/tcp
ftp          21/tcp
ssh          22/tcp       # SSH Remote Login Protocol
ssh          22/udp       # SSH Remote Login Protocol
telnet       23/tcp
# 24 - private
smtp         25/tcp  mail
# 26 - unassigned
time         37/tcp  timserver
```

Folie 164

```
www          80/tcp  http # WorldWideWeb HTTP
www          80/udp       # HyperText Transfer Protocol
# 100 - reserved
pop-3       110/tcp       # POP version 3
pop-3       110/udp
sunrpc      111/tcp portmapper # RPC 4.0 portmapper TCP
sunrpc      111/udp portmapper # RPC 4.0 portmapper UDP
netbios-ns  137/tcp            # NETBIOS Name Service
netbios-ns  137/udp
imap2       143/tcp imap  # Interim Mail Access Proto v2
imap2       143/udp imap
# UNIX specific services
syslog      514/udp
printer     515/tcp spooler# lpr spooler
talk        517/udp
```

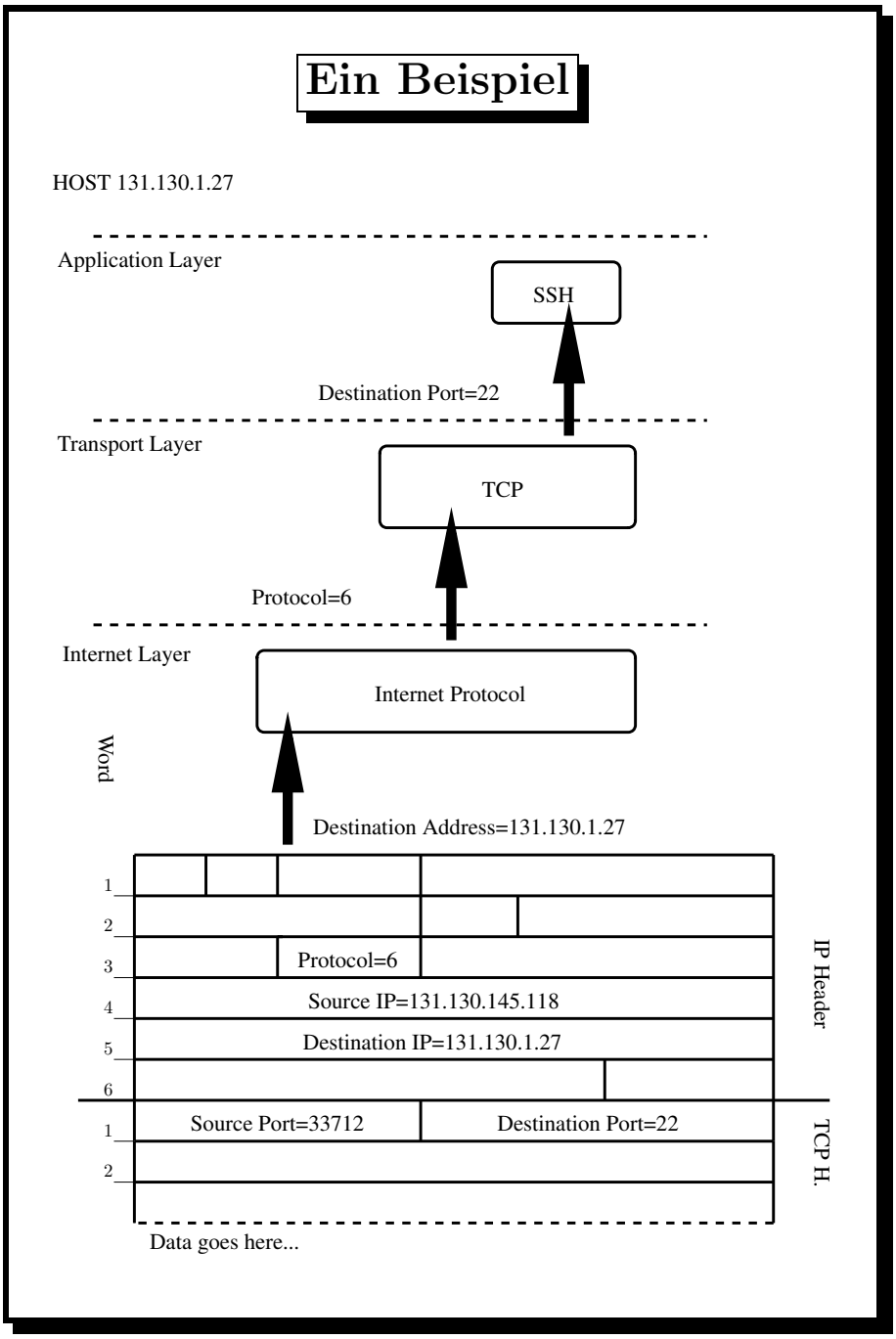
Folie 165

Die Ports 1–255 sind für „well-known services“ reserviert, die Ports von 256–1023 für „Unix specific services“. Der Name ist historisch; diese Services sind längst nicht mehr alle Unix-eigen und auch auf anderen Systemen implementiert. Die sogenannten unprivilegierten Ports ab 1024 werden als *„dynamically assigned ports“* (siehe Abschnitt 13.3) verwendet.

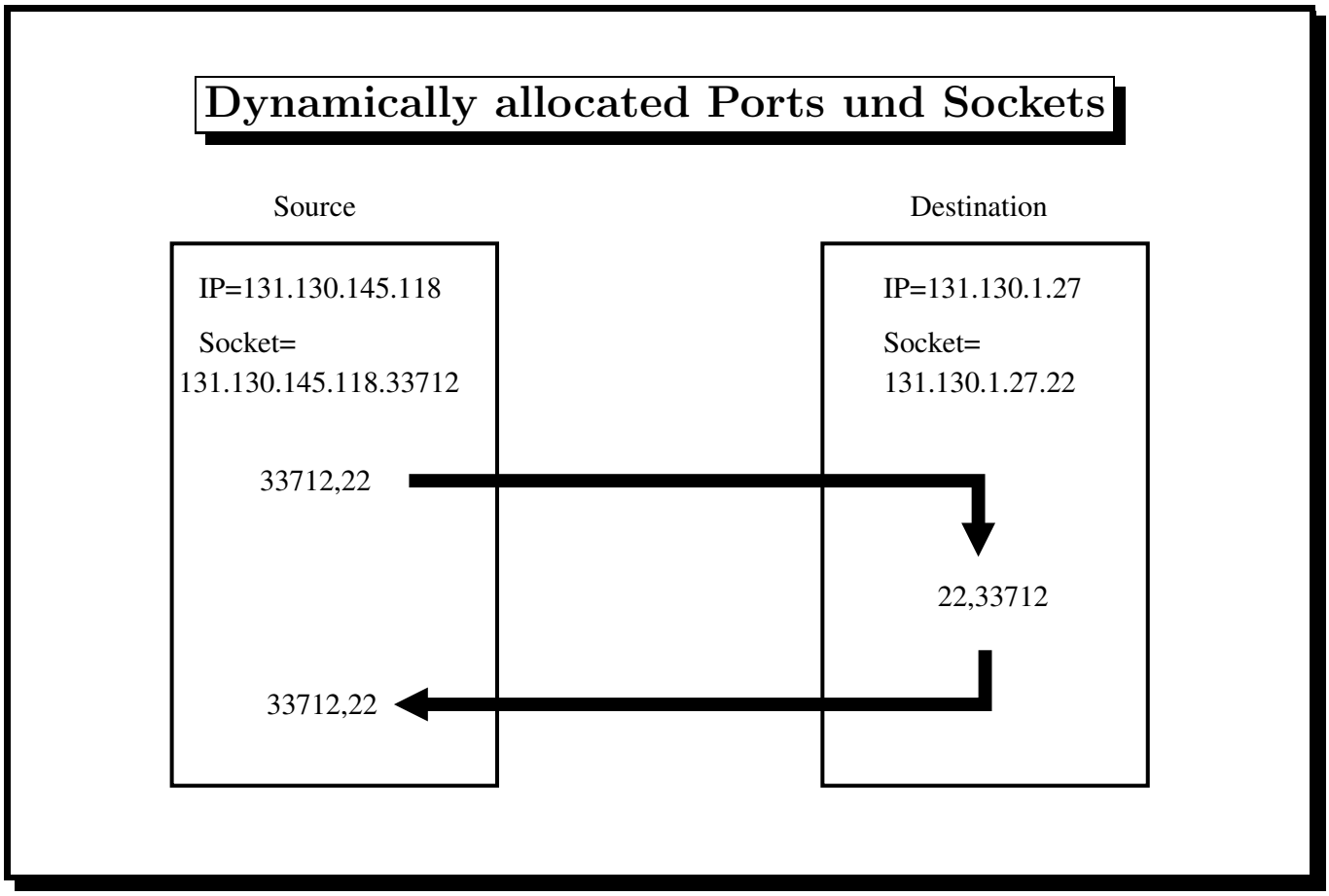
13.3 Protokolle, Ports und Sockets

Nachdem wir nun im Überblick gesehen haben, wie der Transport von Daten im TCP/IP-Stack nicht nur zwischen Hosts, sondern auch zwischen den spezifischen Anwendungen auf den jeweiligen Hosts erfolgt, diskutieren wir als einfaches Beispiel eine Secure-Shell-Verbindung.
Angenommen ein Ethernet-Frame erreicht einen Host mit IP-Adresse 131.130.1.27, so wird es in dessen NAL in ein IP-Datagram verwandelt. Ist die Zieladresse (im 5. Wort des IP Headers, siehe Folie 156) tatsächlich 131.130.1.27, so wird das Datagram im IL weiter entpackt (sonst wird es geroutet, siehe oben). Anhand des Protokolleintrags (im 3. Wort des IP Headers) entscheidet der IL an welches Protokoll im TL die Daten zu überstellen sind. In unserem Beispiel (SSH, siehe Folie 166) wird das TCP-Protokoll (Protokollnummer 6, siehe `/etc/protocols`) verwendet (Es gibt zwar SSH-Varianten, die über UDP laufen; diese werden aber selten verwendet.) Daher leitet der IL die Daten als TCP-Segment entpackt an den TL weiter. Das TCP-Protokoll im TL liest die Destination Port Number (im 1. Wort des TCP Headers, siehe Folie) und stellt die Daten an das SSH-Service (Portnummer 22, siehe `/etc/services`) im AL zu.
Jetzt müssen wir allerdings noch den Mechanismus erklären, wie der Transport zwischen spezifischen Prozessen erfolgt. Das ist notwendig, um mehreren Benutzern die gleichzeitige Verwendung einer Anwendung (hier `ssh`) zu ermöglichen. Dazu konkretisieren wir unser Beispiel: Angenommen ein Benutzer 1 am Host 1 mit der IP-Adresse 131.130.145.118 möchte eine Secure Shell Session zum Benutzeraccount 2 am Host 2 mit der IP-Adresse 131.130.1.27 aufbauen. Nun kann es klarerweise nicht sein, dass die Datenübertragung zwischen den Hosts mit Source-Portnummer = Destination-Portnummer = 22 funktioniert, da sonst nicht zwischen einzelnen Prozessen resp. Benutzern un-

terschieden werden könnte. Daher wird dem `ssh`-Client des Benutzers 1 am Source Host (131.130.145.118) eine *„dynamically allocated Portnumber“* im nichtprivilegierten Bereich (größer 1024) zugewiesen, die für die gesamte folgende Kommunikation als Source Port Number verwendet wird; in unserem Beispiel 33712 (siehe Folie 166). Für die Kommunikation wird am Source Host also der Source Port 33712 und der Destination Port 22 verwendet. Umgekehrt verwendet der Destination Host 131.130.1.27 als Source Port 22 und als Destination Port 33712. Es ist dieser aus IP-Adresse und Portnummer gebildete *Socket*, der die Verbindung auf den beiden Seiten eindeutig definiert; auf Seite des Source Hosts wird der Socket 131.130.145.118.33712 verwendet, Destination-seitig 131.130.1.27.22. Beide Kommunikationspartner kennen die Portnummer des Destination Sockets (22), weil er für das „well-known Service“ SSH festgelegt ist (unter Unix in `/etc/services`). Ebenso kennen beide Hosts die Portnummer des Source Sockets (33712), da sie am Source Host dynamisch zugewiesen und im TCP-Segment dem Destination Host mitgeteilt wurde.


Ein Beispiel
HOST 131.130.1.27
Application Layer
SSH
Destination Port=22
Transport Layer
TCP
Protocol=6
Internet Layer
Internet Protocol
Word
Destination Address=131.130.1.27
1
2
3
4
5
6
Protocol=6
Source IP=131.130.145.118
Destination IP=131.130.1.27
IP Header
1
2
Source Port=33712
Destination Port=22
TCP H.
Data goes here...

Folie 166


Dynamically allocated Ports und Sockets
Source
Destination
IP=131.130.145.118
Socket=
131.130.145.118.33712
IP=131.130.1.27
Socket=
131.130.1.27.22
33712,22
22,33712
33712,22

Folie 167

13.4 Adressierung

Zum Abschluss dieses Kapitels besprechen wir noch *IP-Adressen* und *Adressauflösung* in der TCP/IP-Protokollfamilie.
Im Internet Protokoll (IP) werden die Daten in Form von Datagrams bearbeitet. Jedes Datagram trägt im Header eine Source- (Wort 4) und eine Destination-IP-Adresse (Wort 5). Beide sind 32-Bit Wörter und identifizieren sowohl Netzwerk als auch Host von Source bzw. Destination.
Welcher Teil des 32-Bit-Worts das Netzwerk resp. den Host spezifiziert, ist von der *(Sub)Netzmaske* abhängig (vgl. Folien 169 und 170). In sogenannten *Klasse-A*-Netzwerken ist das erste Bit auf 0 gesetzt; die nächsten 7 Bits bilden den Netzwerkteil, die restlichen 24 den Hostteil. In *Klasse-B*-Netzwerken sind die ersten beiden Bits 1 0, die nächsten 14 Bits für Netzwerk- und die letzten 16 für den Hostteil reserviert. *Klasse-C*-Netzwerke werden durch die ersten Bits 1 1 0 bezeichnet; dann folgen 21 plus 4 Bits für Netzwerk- und Hostteil.
Schreibt man IP-Adressen in Dezimalzahlen an, so ergeben sich 4 Blöcke (jeweils 8 Byte) mit Einträgen zwischen 0 und 255. In dieser Schreibweise haben Klasse-A-Adressen den ersten Eintrag unter 128; es gibt also weniger als 128 Klasse-A-Netze, dafür können in jedem Millionen Hosts adressiert werden. Liegt der erste Eintrag zwischen 128 und 191, so handelt es sich um ein Klasse-B-Netz. Das erste und zweite Byte adressieren das Netzwerk, die letzten beiden Bytes den Host. Klasse-C-Netz haben den ersten Blockeintrag zwischen 192 und 223 (die Adressen darüber sind reserviert); die ersten drei Blocks bezeichnen das Netzwerk, der letzte den Host. In einem Klasse-C-Netzwerk können also nur 256 Hosts adressiert werden, dafür gibt es Millionen von Klasse-C-Netzwerken. Die Netzmasken für Klasse A, B bzw. C-Netzwerke sind respektive 255.0.0.0, 255.255.0.0, 255.255.255.0 (siehe auch Folie 170).

Da IP-Adressen eindeutig sein müssen, werden sie von der Organisation IANA (Internet Assigned Number Authority) bzw. für Subnetze von den jeweiligen Netzwerkadministratoren vergeben. In allen Netzwerkklassen sind die Hostadressen 0 und 255 reserviert. Eine IP-Adresse mit allen Hostnummern gleich 0 bezeichnet das gesamte Netzwerk (zB 26.0.0.0, das Klasse-A-Netz mit der Netzwerkadresse 26). Eine IP-Adresse mit allen Host-

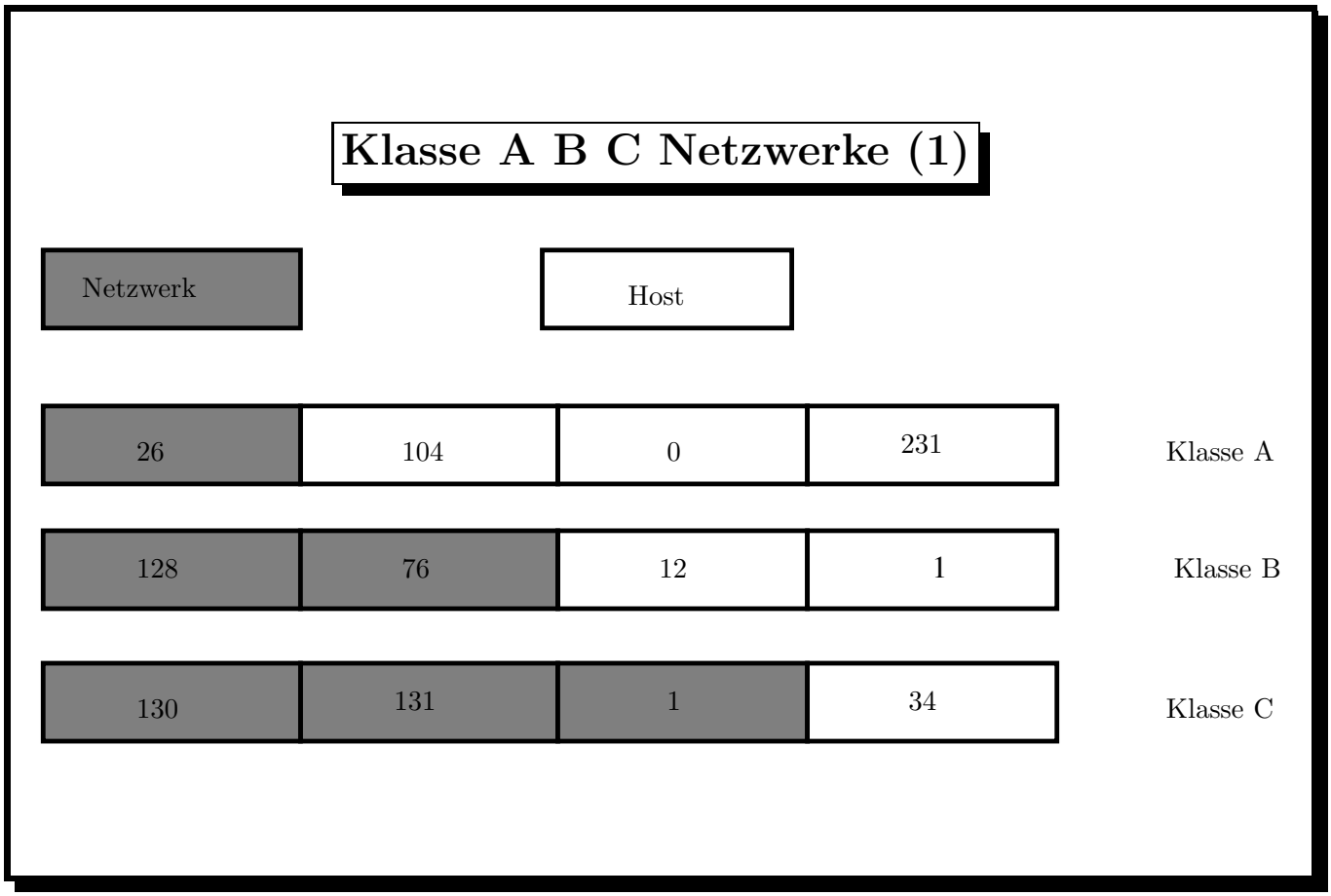
nummern auf 255 bezeichnet alle Hosts im Netzwerk und wird *Broadcastadresse* genannt (zB 131.130.145.255 ist die Broadcast-Adresse im Klasse-C-Netz 131.130.145.0). Gewisse IP-Adressen sind für private Netzwerke reserviert (die dann mittels Gatway an das Internet angeschlossen sind) und im Internet *nicht routbar* (zB 192.168.0.0 für private Klasse-B-Netze und 10.0.0.0 für private Klasse-A-Netzwerke).

Am User-Ende der Kommunikation ist es natürlich unpraktisch, Hosts mit 32-Bit-Wörtern zu bezeichnen. Daher können Hosts auch mit sog. *Fully Qualified Domainnames (FQDN)* referenziert werden. Diese tragen dem Domänenkonzept im Internet Rechnung, das baumartig (ähnlich dem Unix-Verzeichnisbaum) organisiert ist. Die Organisation InterNIC (Network Information Center) vergibt Domainnamen an Organisationen, die dann berechtigt sind, Subdomänen zu vergeben (siehe `http://www.nic.at` oder `http://www.internic.at`).
FQDN wie zB `pablo.mat.univie.ac.at` beginnen mit dem spezifischen Teil, dem Hostnamen, hier `pablo`. Diesem folgen die Subdomänen `mat` (bezeichnet das Institut für Mathematik), `univie` (Universität Wien), `ac` (Academic) und schließlich die Top Level Domäne `at` (Austria). FQDN werden vom Domain Nameservice (DNS) in IP-Adressen aufgelöst bzw. umgekehrt. Jeder Host benötigt zur Auflösung von FQDN in IP-Adressen entweder Zugang zu einem Nameserver (der in der Konfiguration festgelegt ist; unter Unix in `/etc/resolv.conf`) und mittels DNS-Protokolls die Namensauflösung vermittelt oder eine lokale Referenztabelle (unter Unix `/etc/hosts`). Da die Verfügbarkeit des Nameservices ein kritischer Faktor für den gesamten Netzwerkverkehr ist (ohne dieses sind die Hosts am Netzwerk nur unter ihren IP-Adressen erreichbar, die natürlich die wenigsten Benutzer auswendig wissen), sind in `/etc/resolv.conf` mehrere Nameserver angegeben. Außerdem werden in dieser Datei Domänen definiert, in denen Hosts nur mit Hostnamen (ohne Domainnamen) angesprochen werden. Für den Fall, dass zB durch einen Fehler im übergeordneten Netzwerk kein Nameserver erreichbar ist, findet die Datei `/etc/hosts` Verwendung, die Namen und IP-Adressen der wichtigsten Hosts im LAN enthalten sollte (siehe auch Kapitel 14).

Adressen, Name Service

- Host-to-Host Kommunikation
  - User-End: FQDN, zB milkwood.mat.univie.ac.at
  - IL: IP-Adresse, zB 131.130.145.51
  - NAL: MAC-Adresse, zB 00:00:19:CE:C5:FB
- DNS, Nameserver, `/etc/hosts`: FQDN $\leftrightarrow$ IP-Adresse
- ARP, RARP: IP-Adresse $\leftrightarrow$ MAC Adresse

Folie 168


Klasse A B C Netzwerke (1)
Netzwerk
Host
26
104
0
231
Klasse A
128
76
12
1
Klasse B
130
131
1
34
Klasse C

Folie 169

Klasse A B C Netzwerke (2)

Klasse	(Sub)Netzmaske	Adressen
A	255.0.0.0	0.0.0.0–127.255.255.255
B	255.255.0.0	128.0.0.0–191.255.255.255
C	255.255.255.0	192.0.0.0–223.255.255.255
Multicast.	240.0.0.0	224.0.0.0–239.255.255.255

Folie 170

IP-Adressen

- 32-Bit-Wort
- dezimal vier 3er-Blöcke mit Eintragungen von 0 bis 255
- eindeutig vergeben von IANA
- Klasse A, B, C
- reservierte Adressen
  - Netzwerk = alle Hostbits 0: 193.130.145.0, 26.0.0.0
  - Broadcast = alle Hostbits 255: 193.130.145.255, 26.255.255.255
  - Private Netze: 10.0.0.0, 192.168.7.0

Folie 171

FQDN

- eindeutig vergeben von InterNIC
- Baumstruktur
- Subdomänenkonzept
- Format: hostname.subsub· · ·subdomain.· · ·.subdomain.domain
- Beispiel: durruti.mat.univie.ac.at

Folie 172

Am anderen Ende des TCP/IP-Stacks – im NAL – werden die IP-Adressen in Hardwareadressen umgewandelt (ARP, RARP). Jedes Netzwerkdevice (zB Ethernetkarte) verfügt über eine eindeutige, in der Hardware kodierte MAC-Adresse, zB 00:00:19:CE:C5:FB (12-stellige HEX-Nummer). Dem Host sind mittels ARP-Protokoll alle Hardwareadressen im LAN bekannt (`arp`-Kommando) und so können Frames direkt zugestellt werden. Soll ein IP-Datagram an einen Host gesendet werden, der sich nicht im lokalen Netz befindet, so wird es vom NAL in einen Frame an das Gateway verpackt und dort geroutet (siehe Folie 157). Jeder Host muss (mittels Konfiguration) alle *Routen* kennen. Typischerweise ist das die direkte Route zum lokalen Netz sowie die Gateway Route zu allen anderen Netzen. Die Routen können unter Unix mittels `route`-Kommandos angesehen bzw. modifiziert werden.

route, arp

```
bash-2.04$ /sbin/arp
Address                  HWtype HWaddress         Iface
pan.cc.univie.ac.at      ether  00:50:53:87:B8:00 eth0
phobos.mat.univie.ac.at  ether  00:50:BF:10:F7:D8 eth0
golch.mat.univie.ac.at   ether  00:A0:24:59:35:D3 eth0
sirk.mat.univie.ac.at    ether  00:02:44:04:2B:9A eth0

bash-2.04$ /sbin/route
Kernel IP routing table
Destination     Gateway          Genmask         Iface
127.0.0.0       *                255.0.0.0       lo
193.130.145.0   *                255.255.255.0   eth0
default         pan.cc.univie.a  0.0.0.0         eth0
```

Folie 173

14 Netzwerkkonfiguration

In diesem Kapitel erklären wir, wie unter Linux/Unix das TCP/IP-Protokoll, das Nameservice und Netzwerkinterfaces konfiguriert werden.

Bevor man überhaupt Internetanwendungen wie einen Browser oder ein Email-Programm verwenden kann, müssen die darunterliegenden Netzwerkeinstellungen vorgenommen werden. Leider ist viel dieser Konfigurationsarbeit je nach Linux-Distribution und Unix-Variante stark unterschiedlich, weshalb sich der Einsatz von graphischen Tools (zB `YaST` bei SuSE) besonders für Anfänger anbietet. Im Folgenden beschränken wir uns hauptsächlich auf die Netzwerkkonfiguration, wie sie sich unter RedHat-Linux darstellt und besprechen die einzelnen RedHat-spezifischen Konfigurationsdateien.
Grundsätzlich und auf jedem System gliedert sich die Netzwerkkonfiguration in einen Bereich, in dem Parameter für das TCP/IP-Protokoll gesetzt werden und in einen Bereich für jedes Netzwerkdevice, in dem nur die Parameter für das jeweilige Device gesetzt werden. Zum ersten Bereich gehören das Definieren des Hostnamens, des Domainnamens, des Gateways und das Konfigurieren des Nameservices (meist die Angabe eines oder mehrerer Nameserver). Der zweiten Bereich umfasst das Konfigurieren der IP-Adresse des Interfaces, sowie das Festlegen seiner Netzmaske, des Netzwerks, der Broadcastadresse und gewisser Steuerungsparameter (wer das Interface (de)aktivieren darf, wann es aktiviert wird etc.).

Es sei auch darauf hingewiesen, dass die Erläuterungen dieses Kapitel nicht komplett auf Modem-, ISDN- oder ähnliche Verbindungen zutreffen, wobei hier ebenfalls moderne (graphische) Konfigurationstools wie `kppp` einen Großteil der (distributionsspezifischen) Arbeit abnehmen. Für Details sei auf das Modem-HOWTO bzw. das DE-ISDN-HOWTO verwiesen. Auch ADSL- oder Telekabel-Verbindungen sind unter Linux gut unterstützt; es gibt dazu einige informelle HOWTOs im Internet, siehe zB das „Austrian Highspeed Internetconnection & Linux HOWTO“ unter `http://howto.htlw16.ac.at/at-highspeed-howto.html` oder die Webpage der „Telekabel Linux User Group (TKLUG)“

auf der Homepage der „Linux User Group Austria (LUGA)“ `http://www.luga.at`.

Ein besonderer Fall eines Netzwerkinterfaces ist das Loopback (`lo`)-Interface. Es wirkt wie eine virtuelle Netzwerkkarte mit der IP-Adresse `127.0.0.1` und (meistens) dem Hostnamen `localhost`. Alle Pakete, die vom lokalen Rechner an ihn selbst gehen, werden über das Loopback Device geschickt.

14.1 TCP/IP-Konfiguration

Der Großteil der TCP/IP-Konfiguration unter RedHat ist in der Datei `/etc/sysconfig/network` festgelegt. In diesem File definiert man ob überhaupt die Netzwerkfunktionen aktiviert werden sollen (was natürlich geschehen sollte (X Window System!) und daher defaultmäßig auf `YES` gesetzt ist). Weiters wird hier der Hostname (zB `auto.mat.univie.ac.at`) und das Standardgateway (das meist die IP-Adresse `a.b.c.1` oder `a.b.c.254` hat) für die Routingtabelle konfiguriert. Letztere wird beim Netzwerkstart automatisch aus den Informationen in den Konfigurationsdateien erstellt. Verfügt das System über mehr als ein Netzwerkgerät (das Loopbackinterface nicht mitgezählt), so muss das Gatewaydevice angegeben werden, also jenes Device, über das das Gateway erreichbar ist. Eine Beispieldatei befindet sich auf Folie 175.

Ein weiterer wichtige Punkt ist die Konfiguration des *Domain Name Services (DNS)*. Ist dieses nicht verfügbar, können Hostnamen nicht in IP-Adressen aufgelöst werden, was dem Administrator meist viele Useranfragen beschert, da dann nur mehr derjenige auf Netzwerkdienste bestimmter Hosts zugreifen kann, der ihre IP-Adresse kennt.

Netzwerkkonfiguration

- TCP/IP-Konfiguration: Hostname, Domainname, Gateway, Gatewaydevice, ...
  `/etc/sysconfig/network`
- DNS Konfiguration: Nameserver, Searchdomains, ...
  `/etc/resolv.conf, /etc/nsswitch.conf`
- Interfacekonfiguration: IP-Adresse, Netzwerk, Netzmaske, Broadcast, Funktionssteuerung, ...
  `/etc/sysconfig/network-scripts/ifcfg-device`

Folie 174

Es gibt einige Dateien und Dienste, die in einer bestimmten Reihenfolge durchsucht bzw. befragt werden, mit welcher IP-Adresse ein Hostname assoziiert ist. Je nach Einstellung der `hosts` Direktive in `/etc/nsswitch.conf` (bei früheren Linux-Distributionen und anderen Unix-Dialekten die `order` Direktive in `/etc/host.conf`) wird einerseits ein IP-Host(s)-Paar aus der Datei `/etc/hosts` ausgelesen, andererseits – was bei weitem die häufigere Methode darstellt Hostnamen aufzulösen – ein DNS-Server befragt. Welchen DNS-Server der Rechner kontaktieren soll, kann man in der Datei `/etc/resolv.conf` eintragen. Nach Möglichkeit sollte hier mehr als ein Server zu finden sein, damit bei einem Ausfall des ersten Servers der ungestörte Betrieb des Rechners gewährleistet bleibt. Eine komfortable Möglichkeit, Schreibarbeit zu sparen, stellt die `search` Direktive dar. Steht in `/etc/resolv.conf` etwa `search mat.univie.ac.at`, kann man einen beliebigen Rechner unter diesem Domainnamen erreichen, ohne die gesamte Domain mitanzugeben. Also statt `ssh sirk.mat.univie.ac.at` in diesem Fall nur `ssh sirk`. Auch mehrere `search` Direktiven sind zulässig.
Eine gute `/etc/hosts` sollte die wichtigsten IP-Adressen (zB `127.0. 0.1` für localhost, die eigene IP-Adresse, Adressen häufig benötigter Server, ...) enthalten, hauptsächlich aus Geschwindigkeitsgründen (Anfrage an den Server dauert länger als Auswerten einer Datei), aber auch, um im Falle eines totalen DNS-Ausfalls den Betrieb des Rechners möglichst weitgehend aufrecht zu erhalten.
Schließlich kann das DNS Service auch über das NIS Service bereitgestellt werden (siehe Kapitel 16).

TCP/IP-Konfiguration

```
$ cat /etc/sysconfig/network
 NETWORKING=yes
 FORWARD_IPV4=false
 HOSTNAME=milkwood.mat.univie.ac.at
 DOMAINNAME=mat.univie.ac.at
 GATEWAY=131.130.145.1
 GATEWAYDEV=eth0

$ grep hosts /etc/nsswitch.conf
 hosts:      files dns nis
```

Folie 175

/etc/hosts, /etc/resolv.conf

```
bash-2.04$ cat /etc/hosts
127.0.0.1       localhost.localdomain   localhost
131.130.145.118 soweto.mat.univie.ac.at soweto
131.130.145.101 phobos.mat.univie.ac.at phobos
131.130.14.151  erebus.mat.univie.ac.at erebus
131.130.87.22   banach.mat.univie.ac.at banach
131.130.14.152  radon.mat.univie.ac.at  radon

bash-2.04$ cat /etc/resolv.conf
search mat.univie.ac.at ap.univie.ac.at univie.ac.at
nameserver 131.130.1.11
nameserver 131.130.1.12
nameserver 131.130.11.3
```

Folie 176

14.2 Konfiguration der Netzwerkinterfaces

Gängige Netzwerkinterfaces für den LAN Betrieb sind Ethernetdevices (Bezeichnung `eth0, eth1,...`), oder (seltener) Token Ring Devices (`tr0,...`), für serielle Leitungen über Modems PPP-Interfaces (Point-to-Point Protokoll, `ppp0,...`), für parallele Verbindungen (heute sehr selten verwendet) gibt es PLIP-Interfaces.
Bevor ein Netzwerkinterface konfiguriert werden kann, muss sichergestellt sein, dass die Netzwerkhardware bzw. die Treiber richtig funktionieren. Meist werden modulare Treiber verwendet (siehe Kapitel 8). Wenn die Karte nicht bzw. nur durch einen Patch aktiviert werden kann, bietet es sich an, eventuell selbst einen Kernel zu compilieren.
Selbstverständlich können unter Unix mehrere Netzwerkdevices auf einem System betrieben werden zB ein Gateway (mit Firewallfunktion) mit zwei Ethernetkarten oder ein Gateway mit einer Ethernetkarte in einem privaten Netz und einer Modemverbindung (PPP Interface) zu einem Internetprovider.

Alle Dateien zur Interfacekonfiguration befinden sich unter RedHat in `/etc/sysconfig/network-scripts/`; die Konfiguration für ein Interface des Namens `device` ist in `/etc/sysconfig/network-scripts/ifconfig-device` festgelegt. Hier wird die IP-Adresse, die Netzmaske, das Netzwerk und die Broadcastadresse eingestellt. Weiters lässt sich in dieser Datei festlegen, ob das Interface bereits beim Booten oder später händisch gestartet werden soll. Bei Ethernetkarten wird man in den allermeisten Fällen `ONBOOT` auf `YES` setzen; bei PPP-Devices hingegen wird man eventuell mit `USERCTL=YES` allen Benutzern erlauben, das Interface zu (de)aktivieren.

Interfacekonfiguration

```
# cat ifcfg-eth0
 DEVICE=eth0
 IPADDR=131.130.145.112
 NETMASK=255.255.255.0
 NETWORK=193.130.145.0
 BROADCAST=193.130.145.255
 ONBOOT=yes

#cat ifcfg-ppp0
 DEVICE=ppp0
 USERCTL=yes
 DEFROUTE=yes
 ONBOOT=no
 INITSTRING=ATZ
 MODEMPORT=/dev/modem
 LINESPEED=115200
 ESCAPECHARS=no
 PAPNAME=steinbr5
```

Folie 177

Es gibt auch die Möglichkeit, Netzwerkinterfaces automatisch mittels eines *Bootprotokolls* zu konfigurieren. Ein Beispiel dafür ist DHCP (Dynamic Host Configuration Protocol) ein Client-Server basierter Netzwerkdienst. Der Client sucht (mittels Broadcast) einen DHCP-Server im LAN; in der Antwort vom Server sind Daten wie IP-Adresse, Hostname oder zu verwendendes Gateway und Nameserver enthalten. Diese Form der Netzwerkkonfiguration bietet sich besonders bei internen größeren Netzen mit systemweiter Defaultkonfiguration an (zB PC-Labors). Um ein Interface mittels DHCP zu konfigurieren, muss in `ifcfg-device` der Eintrag `BOOTPROTO=DHCP` vorgenommen werden. Ein anderes Bootprotokoll zur automatischen Konfiguration von Netzwerkinterfaces ist das meist für sog. Diskless Clients verwendete BOOTP.

14.3 Aktivieren und Testen der Netzwerkinterfaces

Hat man die Konfigurationsarbeit vorgenommen, kann man nun darangehen, die Interfaces zu aktivieren und auf ihre Funktionstüchtigkeit zu testen. Der Befehl zur Aktivierung eines Interfaces, der unter beinahe allen Unix/Linux-Varianten verfügbar ist, ist `ifconfig`. Mit `ifconfig eth0 up|down` resp. `ifup eth0` oder `ifdown eth0` lässt sich ein Interface aktivieren bzw. deaktivieren. Gibt man `ifconfig` ohne Parameter ein, werden alle aktiven Interfaces aufgelistet. Hier sollte zumindest ein Eintrag für `lo` (Loopback) und ein weiteres Device aufscheinen, um von einer erfolgreichen Aktivierung des Netzwerkes ausgehen zu können.
Alle (`ONBOOT=YES`)-Interfaces können gleichzeitig mittels des entsprechenden Initscripts `/etc/init.d/network start|stop` (de)aktiviert werden; dabei wird im wesentlichen das `ifconfig`-Kommando mit den passenden Optionen und Argumenten ausgeführt und mittels `route` die Routen zu den verschiedenen Netzwerken gesetzt. Ein praktisch bedeutsames Detail ist, dass das Ändern der Netzwerkkonfiguration im laufenden Betrieb (aber heruntergefahrenem Device) erfolgen kann und kein Booten des Systems nötig ist.

Beim Testen überzeugt man sich zuerst davon, dass das Interface ordnungsgemäß geladen wurde (`ifconfig`) und dass die Routen richtig gesetzt sind (`route`). Danach kann man versuchsweise in folgender Reihenfolge IPs zu `ping`en beginnen (das `ping` Kommando dient – vereinfacht – dazu, die Kommunikation mit einem anderen Rechner durch ein Anfrage-Antwort-Verfahren mittels ICMP Protokoll zu testen, siehe Abschnitt 152); `localhost` (Loopback), eigene IP, Gateway, DNS Server und wenn das alles klappt, einen Rechner außerhalb des eigenen Subnetzes (etwa einen bekannten Webserver, der üblicherweise Pings akzeptiert). Durch diese Systematik kann man meist einfach erkennen, wo genau der Fehler liegt (sofern man Hardwareprobleme wie etwa ein nicht angestecktes Netzwerkkabel ausschließen kann).
Wenn Netzwerkpakete nicht an ihrem Zielort ankommen, die Verzögerung der Pakete sehr hoch ist oder man eine Fehlermeldung „No route to host“ erhält, ist es möglich, mittels `traceroute` den Weg eines Paketes vom eigenen Rechner über verschiedene Router zum Ziel zu verfolgen und dabei auch zu sehen, wie viel Zeit das Paket von einem „Hop“ zum nächsten benötigt.
Wenn alle vorherigen Schritte erfolgreich waren, man aber zB bei `ping www.orf.at` die Fehlermeldung „Unknown host www.orf.at“ zurückbekommt, funktioniert das Nameservice nicht richtig. Meist liegt es an der Nichterreichbarkeit eines DNS-Servers, was man leicht durch ein `ping` überprüfen kann. Tools zur Überprüfung der korrekten Funktionsweise (und manuellen *Verwendung*) des Nameservices sind `host` und `nslookup` (veraltet). Gibt man etwa `host www.mat.univie.ac.at` ein erhält man die dem Hostnamen assoziierte IP-Adresse (`131.130.14.152`). Das funktioniert auch umgekehrt und so liefert etwa `host 131.130.1.11` den Hostnamen `ns3.univie.ac.at` zurück.

Interface aktivieren und testen

- (De)Aktivieren
  - `ifconfig device up|down`
  - `ifup device`
  - `ifdown device`
  - `service network start|stop|restart`
- Testen
  - `ping IP-Nummer/IN-Name`
  - `traceroute IP-Nummer/IN-Name`
  - `host IP-Nummer/IN-Name`
  - `route`

Folie 178

ifconfig

```
[root@milkwood /root]# ifconfig
eth0 Link encap:Ethernet  HWaddr 00:90:27:54:80:3A
     inet addr:131.130.1.17 Bcast:131.130.145.255 Mask:255.255.255.0
     UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
     RX packets:41535881 errors:0 dropped:0 overruns:0 frame:0
     TX packets:6051210 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:100
     Interrupt:9 Base address:0x1000

eth1 Link encap:Ethernet  HWaddr 00:00:21:DE:E2:A1
     inet addr:10.10.0.1  Bcast:10.10.0.255  Mask:255.255.255.0
     UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
     RX packets:6877430 errors:3 dropped:0 overruns:0 frame:0
     TX packets:4276200 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:100
     Interrupt:10 Base address:0xfc00
```

Folie 179

```
lo   Link encap:Local Loopback
     inet addr:127.0.0.1  Mask:255.0.0.0
     UP LOOPBACK RUNNING  MTU:3924  Metric:1
     RX packets:64616 errors:0 dropped:0 overruns:0 frame:0
     TX packets:64616 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:0
```

Folie 180

ping

```
[roli@pablo roli]$ ping 131.130.1.11
PING 131.130.1.11 (131.130.1.11) from 192.168.1.110
From 192.168.1.100: Destination Net Unreachable

[root@ken /root]# ping 131.130.1.11
connect: Network is unreachable

[root@ken /root]# ifup eth0
[root@ken /root]# !ping
PING 131.130.1.11 (131.130.1.11) from 62.178.139.47
64 bytes from 131.130.1.11: icmp_seq=0 ttl=249 time=19.606 msec
64 bytes from 131.130.1.11: icmp_seq=1 ttl=249 time=18.998 msec
```

Folie 181

traceroute, host

```
bash-2.04$ /usr/sbin/traceroute www.orf.at
traceroute to www.orf.at 30 hops max, 38 byte
 1  pan.cc.univie.ac.at  1.476 ms  0.956 ms 0.860 ms
 2  iris.cc.univie.ac.at   1.534 ms  1.363 ms 1.760 ms
 3  Vienna-RBS.aco.net   1.804 ms  1.520 ms 1.440 ms
 4  cvix1.apa.net   2.051 ms  1.880 ms  1.543 ms
 5  cvixatm1-l1.apa.net 3.263 ms 3.331 ms 3.445 ms
 6  apain1.apa.at 4.690 ms  5.641 ms  3.988 ms


bash-2.04$ host www.univie.ac.at
www.univie.ac.at. has address 131.130.1.78
bash-2.04$ host 131.130.14.152
152.14.130.131.in-addr.arpa. domain name pointer radon.mat.univie.ac.at.
```

Folie 182

15 Internetdienste und Internetdaemon

Dieses Kapitel beschäftigt sich mit „einfachen“ Netzwerkdiensten und ihrer Konfiguration. Insbesondere besprechen wir den Internetdaemon `(x)inetd`, einen Superserver, der viele der Daemonen, die einfache Netzwerkdienste vermitteln managt. Schließlich erwähnen wir kurz die TCP-Wrapper, die eine einfache Zugangskontrolle zu und damit eine einfache Sicherheitsvorkehrung für Netzwerkdienste ermöglichen.

Die allermeisten Netzwerkdienste sind als Client-Server-Applikationen konzipiert. Der Clientprozess (zB `ssh`) wird vom Benutzer am Clienthost gestartet und versucht eine Verbindung mit dem entsprechenden Serverprozess am Serverhost aufzubauen. Der Serverprozess (zB `sshd`, der Secure Shell Daemon) wird am Serverhost als Daemon meist beim Booten gestartet (Initscript) und läuft mit root-Privilegien oder unter einem eigenen Systemaccount. Er wartet am ihm zugewiesenen Port (hier 22; vgl. Kapitel 13) auf eingehende Anfragen von Clientprozessen von Remotehosts. Geht eine solche Anfrage ein, so wird gegebenenfalls eine Authentifizierung vorgenommen, Logeinträge geschrieben und die Verbindung eröffnet. Meist wird dazu ein weiterer Serverprozess gestartet, der dann nur diese Session managt und nach ihrem Ende ebenfalls beendet wird.

Einfache Netzwerkdienste

- Clientprozess am Clienthost vom Benutzer gestartet
- Serverprozess am Serverhost als Daemon
  - wartet auf eingehende Anfragen
  - root oder andere Privilegien
  - beim Booten gestartet
  - Authentifizierung, Logging
  - forkt und managt Session

Folie 183

15.1 Der Internetdaemon

Um nicht eine zu große Anzahl verschiedener Server laufen zu haben (Performance, Konfigurationschaos, ...) wird oft ein *Internetdaemon* genannter Superserver verwendet – `inetd` oder der neuere `xinetd` („x“ steht hier für e**x**tended und hat nichts mit X zu tun!). Dabei handelt es sich um einen Daemon, der eine Vielzahl von Ports auf eingehende Verbindungsanfragen überwacht und gegebenenfalls den zuständigen Serverdaemon startet.
Services resp. Daemonen die üblicherweise über den Internetdaemon gestartet werden sind `telnetd, ftpd, talkd,` die r-Dienste `rlogind, rshd,...`, `linuxconf-web` (vgl. Abschnitt 2.2) etc. Andere Services werden typischerweise als Standalone Server betrieben; Beispiele dafür sind Secure Shell Server `sshd`, Webserver `httpd`, Mailserver `sendmail,` NIS und NFS Daemonen (siehe Kapitel 16).

Als Internetdaemon wird entweder der `inetd` oder seine Erweiterung der `xinetd` verwendet, deren Konfiguration recht unterschiedlich ist. Der `inetd` wird in einem einzigen File (`/etc/inetd.conf`) konfiguriert, während der `xinetd` sowohl ein globales Konfigurationsfile (`/etc/xinetd.conf`), wie auch ein Konfigurationsfile pro verwalteten Dienst (`/etc/xinetd.d/service`) berücksichtigt. Da in modernen Distributionen hauptsächlich der `xinetd` verwendet wird, beschränken wir uns im Folgenden auf diesen; die Konfiguration ist auf den folgenden Folien erklärt. Weitere Details sind den Manpages `xinetd` und `xinetd.conf` bzw. der Seite `http://www.xinetd.org` zu entnehmen.

Der Internetdaemon

- Prinzipiell für jedes Internetservice ein Daemon
- Prinzipiell jeder bei Systemboot gestartet
- Vereinfachung `(x)inetd` (Internetdaemon)
  - kontrolliert telnetd, ftpd, rlogind,...
  - RedHat < 7.0 und SuSE < 7.3 verwenden einfacheren `inetd`
- trotzdem individuell – als Standalone Server – gestartet:
  sshd, httpd, Nfs, Nis, Samba,...

Folie 184

`xinetd`-Konfiguration

- Konfigurationsdateien `/etc/xinetd.conf` und `/etc/xinetd.d/*`
- für jeden Dienst (service) ist ein Eintrag der Form

```
service name
{
    option1 = value1 value2 ...
    option2 = value1 value2 ...
    ...
}
```

erforderlich, wobei `name` der Name des Dienstes ist (zB: ftp).
Außerdem gibt es die Möglichkeit eines Default-Eintrags.

Folie 185

/etc/xinetd.conf

```
# Simple configuration file for xinetd
# Some defaults, and include /etc/xinetd.d/

defaults
{
        instances       = 60
        log_type        = SYSLOG authpriv
        log_on_success  = HOST PID
        log_on_failure  = HOST
}

includedir /etc/xinetd.d
```

Folie 186

/etc/xinetd.d/telnet

```
# default: on
# description: The telnet server serves telnet
# sessions; it uses unencrypted username/password
# pairs for authentication.
service telnet
{
        flags           = REUSE
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/sbin/in.telnetd
        log_on_failure  = USERID
}
```

Folie 187

`xinetd` und Security

- editieren von `/etc/xinetd.d/*`
  - nicht benötigte Dienste durch die Option `disable = yes` deaktivieren oder zugehöriges Paket deinstallieren
  - Zugang nur für bestimmte Hosts erlauben (siehe nächste Folie)
- `xinetd` neustarten
  - `service xinetd restart`

Folie 188

Zugangskontrolle für den `xinetd`

Folgende Optionen schränken den Zugriff auf einzelne Dienste ein:

- `only_from`: Erlaubt den Zugriff nur für aufgelistete Hosts. Mögliche Werte sind eine beliebige Kombination von (Auswahl)
  - Numerische IP-Adresse, zB: `131.130.14.152` oder Hostname (CNAME), zB: `radon.mat.univie.ac.at`
  - Domainame, zB: `.univie.ac.at`
  - Netzwerkname aus `/etc/networks`
  - Ein Netzwerkbereich der Form `address/netmask`
- `no_access`: Verbietet den Zugriff nur für aufgelistete Hosts. Mögliche Werte sind wie bei `only_from`.
- `access_times`: Erlaubt den Zugriff nur zu bestimmten Zeiten welche in der Form `hh:mm-hh:mm` angegeben werden

Folie 189

Ein typisches Beispiel

```
service ftp
{
        socket_type                = stream
        ...
        only_from       = .mat.univie.ac.at
        no_access       = evil.mat.univie.ac.at
        access_times    = 8:00-16:30
}
```

Diese Konfiguration erlaubt FTP-Verbindungen nur von Rechnern aus der Domäne `mat.univie.ac.at` außer dem Rechner `evil.mat.univie.ac.at` in den Zeiten von 8:00 bis 16:30.

Folie 190

15.2 TCP-Wrapper

Eine weitere, schon etwas ältere Methode den Zugang zu verschiedenen Netzwerkdiensten zu kontrollieren, ist die Verwendung der *TCP-Wrapper*. Diese können auf zwei Arten verwendet werden. Entweder ist der Daemon, der einen Dienst vermittelt mit TCP-Wrapper Unterstützung compiliert (zB: `sshd, sendmail,...`) oder, falls er über keine interne Wrapper Unterstützung verfügt, kann diese extern über den TCP-Wrapper genannten `tcpd`-Daemon implementiert werden. In beiden Fällen werden die Dateien `/etc/hosts.deny` und `/etc/hosts.allow` ausgewertet (deren Syntax auf den Folien erklärt wird) und die darin festgelegten Regeln für den Zugang für die verschiedenen Diensten befolgt.

Für einen Server ohne interne Wrapper Unterstützung kann diese auf folgendem Wege implementiert werden. Der Internetdaemon wird so konfiguriert, dass er bei Anfragen an den entsprechenden Port den `tcpd`-Daemon startet. Dieser übernimmt die Überprüfung der Zugriffsberechtigungen gemäß `/etc/hosts.deny|allow` und startet bei positivem Ausgang den Daemon, der eigentlich den Dienst vermittelt. Diese Vorgangsweise ist beim `xinetd` nicht besonders sinnvoll, da dieser eine interne Zugangskontrolle bereitstellt (siehe oben), ist aber die gängigste Möglichkeit Zugangsbeschränkungen mittels des älteren `inetd` zu realisieren. Beispielkonfigurationen für beide Internetdaemonen finden sich auf den Folien.

TCP-Wrapper

- Zugangsbeschränkung für bestimmte Dienste die mit TCP-Wrapper Support übersetzt wurden (zB `sshd`, `sendmail`)
- Zugangsbeschränkung für Dienste ohne internen TCP-Wrapper Support über `tcpd`
- Konfiguration
  - Service-, Benutzer- und Host-spezifisch
  - `/etc/hosts.allow` (stärker)
  - `/etc/hosts.deny`
- Loggen der Anfragen über `syslogd`
  - `/var/log/messages`
  - `/var/log/secure`

Folie 191

TCP-Wrapper-Konfiguration

- `/etc/hosts.allow(deny)`-Syntax
  - `daemon_list : client_list [ : shell_command ]`
  - `daemon_list : ... user_pattern@host_pattern ...`
- Konfiguration testen mit
  - `tcpdmatch daemon[@host] [user@]host`
  - `tcpdchk`

Folie 192

/etc/hosts.deny

```
bash# cat etc/hosts.deny
#
# hosts.deny    This file describes the names of the hosts
#               which are *not* allowed to use the local
#               INET services, as decided by the
#               '/usr/sbin/tcpd' server.
#

ALL: ALL@ALL : spawn ( /usr/local/sbin/secrep %h %d | \
/bin/mail -s ALKO_secrep_%d roland.steinbauer@univie.ac.at )
```

Folie 193

/etc/hosts.allow

```
[stein@doppler stein]# cat /etc/hosts.allow
#ALL: 131.130.26., 131.130.87., 131.130.11., 131.130.1.26
#      5. stock     akh          ap            mailbox

ALL: 131.130.26., 131.130.87.67, 131.130.87.94,\
   131.130.87.91, 131.130.87.95, 131.130.87.93

in.talkd: 136.142.123.77, 131.130.26.130, 131.130.87.
#           saschas raven.phyast.pitt.edu

sshd2, sshd1, sshd, sshdfwd-X11: ALL EXCEPT .com,\
      .mil, .net, .org, 131.130.38.,  131.130.44.,\
       131.130.39., a-sa7-42.tin.it, \
```

Folie 194

xinetd und TCP-Wrapper

```
$ cat /etc/xinetd.d/telnet

service telnet
{
        flags       = REUSE NAMEINARGS
        protocol    = tcp
        socket_type = stream
        wait        = no
        user        = telnetd
        server      = /usr/sbin/tcpd
        server_args = /usr/sbin/in.telnetd
}
```

Folie 195

inetd und TCP-Wrapper

```
root@dukana# cat /etc/inetd.conf
# inetd.conf
# Echo, discard, daytime, and chargen are used
# primarily for testing.
echo    dgram   udp  wait    root  internal
discard stream  tcp  nowait  root  internal
discard dgram   udp  wait    root  internal
#
# These are standard services.
#
ftp    stream tcp nowait root /usr/sbin/tcpd in.ftpd -l -a
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

Folie 196

Ein typisches Szenario

```
[root@mut /root]# cat /etc/hosts.deny
ALL:ALL
[root@mut /root]# cat /etc/hosts.allow
[root@mut /root]#
    [dylan@milkwood /home/dylan]# telnet mut
    Trying 192.168.0.129...
    Connected to mut.
    Escape character is '^]'.
    Connection closed by foreign host.
    [dylan@milkwood /home/dylan]#
[root@mut /root]# tail -1 /var/log/secure
Mar 29 20:37:15 localhost in.telnetd[1866]: refused
                connect from 192.168.0.1
```

Folie 197

15.3 Einfache Internetdienste

Zum Schluss dieses Kapitels behandeln wir die Internetdienste Telnet, Ftp, die r-Dienste sowie deren „sicheren“ Ersatz die Secure Shell `ssh`.

TELNET ist ein Protokoll im Application Layer der TCP/IP-Protokollfamilie (vgl. Kapitel 13) und dient zum Login auf einem Host über das Netzwerk. Serverseitig – also auf dem Host, auf den man sich übers Netz einloggen will – muss ein Telnetserver laufen; dieser heißt meist `telnetd` oder `in.telnetd` und wird fast immer vom Internetdaemon gestartet. Der Client heißt `telnet`, die Syntax ist `telnet hostname`. Der Loginname wird interaktiv abgefragt; für Details zur Syntax und auch den interaktiven Modus siehe die Manpage bzw. Folie 198.

FTP ist ebenfalls ein Application-Layer-Protokoll und dient dem Transfer von Dateien über ein Netzwerk; FTP steht für File Transfer Protokoll. Unter Linux stehen mehrere Ftp-Server-Pakete zu Verfügung (siehe Abschnitt 17.1), die teilweise über den Internetdaemon oder als Standalone Server betrieben werde. Es existiert eine Vielzahl von Client-Programmen. Unter ihnen das auf jedem System vorhandene, doch recht krude `ftp` und meist auch das komfortable `ncftp`. Letzteres stellt eine Commandline Completion und einen automatischen Anonymen Login zur Verfügung. Anonymes Ftp wird vorallem von großen Ftp-Servern im Internet angeboten, die so den anonymen Download von Daten erlauben. Ist man per Ftp eingeloggt, so können mittels des Kommandos `get remotefile` Dateien herunterladen bzw. mittels `put localfile` upgeloaded werden (falls das vom Server erlaubt wird). Für die Syntax siehe auch Folien 199 - 201 bzw. die Manpages.

Die *r-Dienste* sind eine Familie (älterer) Unix Services, die ein Remotelogin `rlogin, rsh`, Remotecopy `rcp` (in der Syntax ähnlich dem gewöhnlichen `cp`) und Remote Commandexecution `rexec` bereitstellen. Die Server werden über den Internetdaemon gestartet, für die Syntax der Clients siehe die Folie 202 und die Manpages. Da die Authentifizierung sehr unsicher ist (jeder Benutzer kann zB im File `~/.rhosts` Host/Usernamen-Paare angeben, um diesen ein Benutzen der Services *ohne* Passwortabfrage zu ermöglichen!!!) werden diese Services aber kaum noch benutzt.

Telnet

```
[roli@pablo roli]$ telnet merlin.ap.univie.ac.at
Trying 131.130.11.52...
Connected to merlin.ap.univie.ac.at.
Escape character is '^]'.

Compaq Tru64 UNIX V5.1 (Rev. 732) (balin.ap.univie.ac.at) (pts/6)

login: steinbau
Password:

Willkommen auf MERLIN     Mixed Architecture Unix Cluster MERLIN
Aussenstelle Physik            Vienna University Computer Center

You have mail.
--- Diskusage: 56%. ---

bash-2.03$
```

Folie 198

Ftp

```
[roli@pablo roli]$ ftp radon
Connected to radon (131.130.14.152).
220 ProFTPD 1.2.2 Server (Department of Mathematics) [radon.mat.univie.ac.at]
Name (radon:roli): anonymous
331 Anonymous login ok, send your complete email address as your password.
Password:
230 Anonymous access granted, restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (131,130,14,152,9,14).
150 Opening ASCII mode data connection for file list
drwxr-sr-x   5 root     ftp          4096 Nov 11 10:53 pub
drwxrwsrwx   2 martin   ftp          4096 Mar 31 20:54 ttt
226 Transfer complete.
```

Folie 199

```
ftp> get mutt_1.3.25-1.dsc
local: mutt_1.3.25-1.dsc remote: mutt_1.3.25-1.dsc
227 Entering Passive Mode (131,130,14,152,9,22).
150 Opening BINARY mode data connection for mutt_1.3.25-1.dsc (665 bytes).
226 Transfer complete.
665 bytes received in 0.00709 secs (92 Kbytes/sec)
ftp> help
Commands may be abbreviated.  Commands are:
!               debug           mdir            sendport        site
$               dir             mget            put             size
account         disconnect      mkdir           pwd             status
append          exit            mls             quit            struct
ascii           form            mode            quote           system
bell            get             modtime         recv            sunique
binary          glob            mput            reget           tenex
[...]
ftp> quit
[roli@pablo roli]$
```

Folie 200

NcFtp

```
[roli@pablo roli]$ ncftp ftp.univie.ac.at
NcFTP 3.0.3 (April 15, 2001) by Mike Gleason (ncftp@ncftp.com).
Connecting to ftp.univie.ac.at...

                    Welcome to FTP.UNIVIE.AC.AT
                    ===========================
   you are user #46 of 500 simultaneous users allowed.
Anonymous access granted, restrictions apply.
Logged in to ftp.univie.ac.at.
ncftp / > get systems/linux/distributions/mandrake/iso/README
README:                                     2.00 kB   23.18 kB/s
ncftp / > quit
[roli@pablo roli]$
```

Folie 201

Die r-Kommandos

- `in.rlogind`
- benutzerfreundlich, Unix-Standard
  - `rlogin`... analog `telnet`
  - `rsh`... remote Commandexecution
  - `rcp`... Filetransfer, Syntax ähnlich `cp`
- können Passwortauthentifizierung umgehen!
  - `/etc/hosts.equiv` (zuerst gelesen)
  - `~/.rhosts` (per-User Konfiguration)
  - `/.rhosts` (root-Zugang)
  - `/.rhosts`-Syntax: [+|-] hostname username

Folie 202

Telnet Ftp und die r-Services haben ein großes Sicherheitsmanko; sie versenden Klartextpasswörter über das Netz. Gelingt es einem Angreifer (etwa mit einem Portsniffer; siehe Kapitel 18) die Kommunikation „zu belauschen“, so kann er auf einfache Weise Username/Passwort-Paare in Erfahrung bringen.
Einen sicheren Ersatz stellt die *Secure Shell* zur Verfügung. Das Secure-Shell-Protokoll ist ebenfalls ein Application-Layer-Protokoll; sein User-End ist dem von Telnet und Ftp sehr ähnlich. Der große Unterschied besteht darin, dass die gesamte Session verschlüsselt über das Netzwerk übertragen wird. Die Basissyntax ist auf Folie 203 erklärt. Die Konfiguration des (fast immer als Standalone Server betriebenen) Secure Shell Servers erfolgt in der Datei `/etc/ssh/sshd_config`. In den meisten Linux-Distributionen ist der `sshd` mit TCP-Wrapper Unterstützung compiliert, dh. liest die Konfiguration in `/etc/hosts.allow|deny`. Die Secure Shell kennt zwei Protokollvarianten, Version 1 und Version 2; beide bieten die Möglichkeit einer *schlüsselbasierten Authentifizierung* und die Möglichkeit mittels des Authentication Agents (`ssh-agent`) Keys zu verwalten. Mit `ssh-keygen` generiert der Benutzer ein Public/Private Key-Paar, das durch eine Passphrase geschützt ist. Der Private Key liegt in `~/.ssh/` und muss geheim gehalten werden (Mode 400), der Public Key muss auf den Remotehost in die Datei `~/.ssh/authorized_keys` kopiert werden. Auf dem lokalen Rechner muss der `ssh-agent` laufen. Zweckmäßigerweise wird oft die gesamte X-Session vom `ssh-agent` aus gestartet (zB mittels Eintrag `/usr/sbin/ssh-agent /usr/bin/startkde` in `~/.xsession`). Sonst kann zB auch eine Shell vom Agent aus gestartet werden (Eingabe: `ssh-agent bash`). Läuft der Agent so muss mittels `ssh-add` der Private Key geladen werden, wobei die Passphrase abgefragt wird. Jede Secure Shell Verbindung zum Remotehost wird jetzt über das Public/Private-Key-Paar statt das Passwort authentifiziert. Insbesondere und praktisch relevant muss nur einmal (pro lokaler Session) die Passphrase eingegeben werden, alle `ssh`-basierten Verbindungen (auch `scp, sftp`) erfolgen nun ohne weitere Notwendigkeit einer interaktiven Authentifizierung.
Für Details (insbesondere die genauen Dateinamen für die verschiedenen Protokoll und Key-Versionen) siehe Folie 204 und die entsprechenden Manpages.

Zusammengefasst verwendet der sicherheitsbewusste Benutzer `ssh` statt Telnet und `sftp` bzw. `scp` statt Ftp und `rcp`; Ftp sollte nur für anonymen Filetransfer verwendet werden. Der sicherheitsbewusste Administartor verzichtet auf die r-Dienste, den Telnetserver und den nichtanonymen Ftp-Server.

Secure Shell

- Syntax
  - Remote Login: `ssh user@host` oder `ssh -l user host`
  - Filetransfer: `scp user@host:sourcefile destinationfile` oder `scp sourcefile user@host:destinationfile`
  - Filetransfer interaktiv: `sftp user@host`
- Konfiguration
  - Client: `/etc/ssh/ssh_config`, `~/.ssh/`
  - Server: `/etc/ssh/sshd_config`

Folie 203

ssh-Keys

- erstellen: `ssh-keygen` Public/Private Key-Paar mit Passphrase
- Private Key: in `~/.ssh/` am lokalen Host, geheim (Mode 400)
- Public Key: in `~/.ssh/authorized_keys` am Remote Host
- Agent: `ssh-agent command` (zB `startkde, bash`)
- Key laden: `ssh-add` Passphrase wird abgefragt
- alle `ssh`-Verbindungen über Keys authentifiziert

Folie 204

16 NFS und NIS

In diesem Kapitel besprechen wir zwei prominenten Netzwerkdienste, die auf RPC (Remote Procedure Call) aufbauen. Das Network File System (NFS) erlaubt es Dateisysteme über ein Netzwerk zu verteilen; ein Fileserver stellt seine Dateisysteme über das Netz den Clienthosts zur Verfügung. Das „Administrative Database System" NIS (Network Information Service) erlaubt es spezielle Systemdateien wiederum von einem Server über ein Netzwerk an Clienthosts zu übertragen.

16.1 Remote Procedure Call (RPC)

Das *Remote Procedure Call (RPC)*-Protokoll von Sun definiert einen systemunahbängigen Standard für die Kommunikation von Prozessen über ein Netzwerk. Es besteht aus einer Bibliothek, die es Programmen ermöglicht Procedure Calls an entfernte Hosts in einem Netzwerk zu senden. Der Client sendet einen RPC Call an den Server, der eigene Routinen benutzt um die angefragten Operationen auszuführen und schließlich eine Antwort zurück an den Client sendet.

RPC stellt eine weitere Abstraktionsebene über den Application-Layer-Protokollen des TCP/IP-Stacks dar. Um RPC-Protokolle verwenden zu können muss ein eigener Server – der *RPC-Portmapper* meist `portmap` oder `rpcbind` genannt – am System laufen. Startet ein RPC-Server, so teilt er dem Portmapper mit, welche Services – geregelt über die sog. RPC-Servicenummern (analog zu den TCP/IP-Ports) – er anbietet und auf welchen Ports er kontaktiert werden kann. Ein RPC-Client nimmt immer zunächst Verbindung mit dem Portmapper auf, um herauszufinden, wie (d.h. auf welchem Port) er den Server – sprich ein Service bezeichnet durch eine RPC-Servicenummer – kontaktieren kann.
Mit dem Kommando `rpcinfo` kann man abfragen welche RPC-Services ein Host anbietet; siehe dazu Folie 206 bzw. die Manpage.

RPC bringt also eine zusätzliche Protokollebene mit sich. Der Vorteil besteht darin dass RPC-Protokolle unabhängig vom TL-Protokoll funktionieren und so sowohl UDP als auch TCP verwenden können.

Remote Procedure Call (RPC)

- von Sun Microsystems entwickelt
- systemunabhängiges Protokoll für Prozesskommunikation übers Netz
- ermöglicht Programmen Procedure Calls auf entfernten Hosts
- zusätzliche Abstraktionsebene über TCP/IP AL-Protokollen
- Services definiert über RPC-Servicenummern (analog Portnummern)
- benötigt eigenen Dienst: RPC-Portmapper (`portmap`)
  - RPC-Server registriert sich beim Start beim Portmapper
  - Portmapper verbindet RPC-Servicenummern mit TCP- bzw. UDP-Ports
  - RPC-Client kontaktiert zuerst Portmapper

Folie 205

rpcinfo

```
[roli@pablo roli]$ /usr/sbin/rpcinfo -p ken
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100004    2   udp    960  ypserv
    100004    1   udp    960  ypserv
    100004    2   tcp    963  ypserv
    100004    1   tcp    963  ypserv
    100007    2   udp    980  ypbind
    100007    1   udp    980  ypbind
    100007    2   tcp    983  ypbind
    100007    1   tcp    983  ypbind
    100011    1   udp    691  rquotad
    100011    2   udp    691  rquotad
    100011    1   tcp    694  rquotad
    100011    2   tcp    694  rquotad
    100005    1   udp  32769  mountd
    100005    1   tcp  32769  mountd
    100005    2   udp  32769  mountd
    100005    2   tcp  32769  mountd
    100005    3   udp  32769  mountd
    100005    3   tcp  32769  mountd
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100009    1   udp    729  yppasswdd
[roli@pablo roli]$
```

Folie 206

16.2 Network File System (NFS)

Das Network File System ist wahrscheinlich der prominenteste Netzwerkdienst, der RPC verwendet. Ursprünglich 1986 von Sun entwickelt, verbreitete es sich rasch auf alle gängigen Unix-Varianten, da die Protokollspezifikationen von Anfang an frei verfügbar waren. Es wurden auch einige Versuche unternommen, NFS auf anderen Betriebssystemen zu verbreiten (zum Beispiel MS-DOS). Als Standard hielt es sich allerdings nur in der Unix-Welt.
Das NFS-Protokoll hat sich über die Jahre hinweg als äußerst stabil erwiesen. Die erste veröffentlichte Version war Version 2, einige Verbesserungen wurden Anfang der 90er Jahre in die Version 3 implementiert, die aber völlig rückwärtskompatibel ist. Im Februar 2000 schließlich hat Sun die gesamten Quellen unter einer GPL-ähnlichen Lizenz freigegeben.

Einfach gesprochen ermöglicht NFS das Mounten von Dateisystemen über ein Netzwerk. Ein NFS-Client kann ein Dateisystem eines NFS-Servers wie ein lokales Dateisystem mounten (vgl. Abschnitt 8.1). Da (normalerweise) nur root Mounts vornehmen kann, ist unter NFS – anders als in der Windows-Welt (siehe auch Kapitel 19), wo jeder Benutzer Netzwerklaufwerke verbinden kann – das Einrichten von Netzwerkverzeichnissen dem Administrator vorbehalten und wird meist mit einer geeigneten Automatisierung des `mount`-Kommandos (`automount`, `amd`) erledigt.

NFS ist *transparent*, d.h. man kann auf Daten des Servers genauso zugreifen, wie auf lokale Dateien. Das zu Grunde liegende Dateisystem ist dabei nicht von Bedeutung. Das System ist zustandsunabhängig (*stateless*), d.h. der Server speichert keine Zustandsinformationen über die angebundenen Clients. Ist die Netzwerkverbindung unterbrochen, so zieht das den Server überhaupt nicht in Mitleidenschaft (den Client natürlich schon).

NFS besteht aus mehreren Komponenten, darunter ein Mountprotokoll und ein Mountserver `rpc.mountd`, Dämonen, die das Dateiservice selbst vermitteln (`nfs` und andere) und einige Diagnoseprogramme. Ein Großteil dieser sowohl server- als auch clientseitigen Software ist Teil des Kernels und benötigen keinerlei Konfiguration. Allerdings muss man bevor NFS überhaupt verwendet werden kann – sei es als Client oder Server – sicherstellen, dass der NFS-Support im Kernel eincompiliert ist. Das überprüft frau am einfachsten mittels `cat /proc/filesystems`. Erscheint `nfs` in dieser Liste, ist alles in Ordnung. Andernfalls muss man eventuell das entsprechende Modul laden oder den Kernel neu compilieren.

Darüberhinaus ist clientseitig nur die Konfiguration und gegebenenfalls eine geeignete Automatisierung der NFS-Mounts nötig. Serverseitig muss konfiguriert werden, welche Teile des Verzeichnisbaums an welche Clients *exportiert* (also freigegeben) werden. Außerdem müssen sowohl am Client, wie auch am Server der `portmap`-Daemon laufen, am Server noch andere Daemonen, die wir im weiteren Verlauf besprechen.

Was ist NFS?

- Dateien und Verzeichnisse über ein Netzwerk verwenden
- Client-Server Architektur
- Client mountet Dateisysteme vom Server wie lokale Dateisysteme
- transparent und stateless
- Vorteile
  - Gleiche Homedirectories im ganzen Netzwerk
  - Sparen von Plattenplatz
  - zentrale Administration
- entwickelt von SUN, Standard auf allen UNIX-Systemen(!)

Folie 207

Der NFS-Server
Ein NFS-Server exportiert Teile seines Verzeichnisbaums an die NFS-Clients, die diese dann mounten können. Der Prozess am Server, den die Clients beim Mounten kontaktieren ist völlig getrennt vom Prozess, der dann den Zugriff auf die Dateien ermöglicht. Erstere Aufgabe wird vom Mountdaemon, dem `rpc.mountd` wahrgenommen, letztere vom NFS-Daemon `nfsd`. Diese beiden Daemonen müssen also am NFS-Server zusätzlich zum Portmapper `portmap` und *nach* diesem gestartet werden. `rpc.mountd` und `nfsd` werden über ein gemeinsames Initscript (`/etc/init.d/nfs`) mit den passenden Optionen und in ausreichender Zahl (es laufen immer mehrere Kopien von `nfsd`; für Performanceoptimierung siehe die lokale Dokumentation) gestartet.

Die NFS-Exports, also die freigegebenen Verzeichnisse werden in der Datei `/etc/exports` konfiguriert. Hier werden Verzeichnisse und verschiedenen Parameter, die die Zugriffsrechte der Clients regeln eingetragen. Gemäß dieser Einträge entscheidet `rpc.mountd`, ob einkommende Mount-Requests von Clients berechtigt sind oder nicht und lässt die Mounts zu oder lehnt sie ab.
Ist ein NFS-Verzeichnis gemountet, so kann auf alle unterliegenden Dateien und Verzeichnisse zugegriffen werden (sofern in der Konfiguration nicht explizit ausgeschlossen; siehe zB [1], p. 498). Die Zugriffsberechtigungen werden über die ganz normalen Dateiberechtigungen bestimmt, daher ist die beste Regelung wenn zB die UID 615 am Server mit demselben Benutzer assoziiert ist wie am Client. Dies kann durch eine Synchronisierung der Passwortdateien erreicht werden zB durch die Verwendung von NIS (siehe Abschnitt 16.3). Es ist zwar ein Mappen der UIDs möglich (siehe unten) wegen der größeren Kompliziertheit und dem Administrationsaufwand aber nicht empfehlenswert. Einzige Ausnahme ist der root-Account: root am Client wird standardmäßig auf den Gastaccount nobody am Server gemappt. Dies ist aber nur ein sehr schwacher Schutz des Server-Dateisystems vor root am Client, da dieser ja mittels `su` auf jeden Account am Client einloggen und so mit dessen Rechten auf das NFS-Verzeichnis zugreifen kann.

NFS-Daemonen

- `rpc.mountd` (Mountdaemon) überprüft Berechtigung von Mount-Requests
- `nfsd` (eigentlicher NFS-Daemon) vermittelt Dateizugriff

- `rpc.ugidd` übersetzt UIDs zwischen Clients und Server
- `rpc.rquotad` Quotas für NFS-Benutzer
- `rpc.statd` Reboot Notification
- `rpc.lockd` lockt NFS-exportierte Files; meist vom Kernel gestartet

Folie 208

In jeder Zeile von `/etc/exports` steht zuerst das Verzeichnis, das exportiert werden soll, danach ein Leerzeichen, dann die Clients mit den jeweiligen Zugriffsoptionen in Klammern und ohne Leerzeichen zwischen Client und Klammer. Die Syntax von `/etc/exports` ist äußerst heikel und ein Leerzeichen zu viel oder zu wenig stört die ordnungsgemäße Funktion des NFS-Servers! Die Client-Hosts können per Name oder IP-Adresse angegeben werden. Außerdem kann man ganze Subnetze angeben, indem man die entsprechenden Oktette weglässt. Eine Beispieldatei befindet sich auf Folie 209. Wir erläutern einige gängige Optionen für die Datei `/etc/exports`.

- `secure` der Port, von dem die Anfrage kommt, muss < 1024 sein.
- `insecure` Gegenteil von `secure`
- `ro` Nur Lesezugriffe auf das Verzeichnis möglich; Standardeinstellung
- `rw` Schreib- und Lesezugriff
- `root_squash` übersetzt die UID 0 – also root – auf nobody
- `link_relative` verwandelt absolute Links (die mit einem / anfangen) in relative. Das ist aber nur sinnvoll, wenn das gesamte Filesystem exportiert wird. Sonst können die Links ins Nichts führen
- `link_absolute` lässt absolute Links unverändert
- `map_identity` der Server geht davon aus, dass der Client dieselben UIDs verwendet; Standardeinstellung
- `map_daemon` der Server fragt `rpc.uigdd` zur Übersetzung der UIDs (siehe unten)
- `map_static` die UIDs werden in einer Datei statisch übersetzt

/etc/exports

- Zugangsbeschränkung über `/etc/exports`
- Syntax: Directory Host(Optionen)

```
$ cat /etc/exports

/home           131.130.87.(ro) 131.130.145.63(rw)
/usr/local      dukana(insecure,ro)
#/mnt/cdrom     diana(ro,no_root_squash)
/usr/doc        131.130.145.(insecure,ro)
```

Folie 209

NFS-Exports Optionen

- `ro` read-only
- `rw` read-write
- `secure` Anfragen brauchen Port < 1024
- `root_squash` root@client = nobody@server
- `link_relative` absolute Links in relative umwandeln
- `map_*` gibt an, wo der Server das UID Mapping hernehmen soll

Folie 210

Alle Exports resp. NFS-Mounts können (auch Remote) mittels `showmount`-Kommando erfragt werden. Die Option `-a` zeigt alle Verbindungen im *host:verzeichnis* Format. Mit der Option `-d` zeigt `showmount` alle exportierten Verzeichnisse an; siehe auch Folie 211.

Das Kommando `exportfs` dient zur Wartung der NFS-Exports. Wird eine Änderung in `/etc/exports` vorgenommen, so muss diese mit `exportfs -r` an das NFS-System weitergegeben werden, oder alternativ aber weniger empfehlenswert das ganze NFS-Service mittels Initscripts neugestartet werden.

Weitere NFS-Daemonen die (meist) auch von `/etc/init.d/nfs` gestartet werden sind:

- `rpc.ugidd` übersetzt die UIDs zwischen Server und den Clients
- `rpc.quotad` NFS-Quotadaemon; setzt lokale Quotas voraus.
- `rpc.lockd` kümmert sich um das Sperren der vom Netzwerk aus verwendeten Dateien; meist vom Kernel automatisch gestartet.
- `rpc.statd` überwacht *nicht* den Status der Verbindungen sondern verständigt nach einem Reboot des Servers die Clients vom Crash.

Zum Schluss sei für alle weiteren Details auf das ausführliche NFS-HOWTO verwiesen, das auch eine Schritt-für-Schritt Anleitung zum Aufsetzen eines NFS-Servers enthält.

showmount

- `-d` zeigt alle gemounteten Verzeichnisse
- `-a` zeigt alle gemounteten Verzeichnisse und deren Hosts
- `-e` zeigt die exportierten Verzeichnisse an (gem. `/etc/exports`)

```
bash-2.04$ /usr/sbin/showmount -a phobos
All mount points on phobos:
131.130.14.152:/users/neum
131.130.14.152:/users/susanne
131.130.145.102:/dist
131.130.145.108:/dist/redhat-7.1
131.130.145.108:/users/flo
```

Folie 211

NFS-Mounts
Das mounten eines NFS-Verzeichnisses funktioniert analog dem Mounten eines lokalen Filesystems, allerdings mit einer etwas veränderten Syntax. Will man zB auf einer lokalen Workstation das Verzeichnis `/nethome` vom Server `dukana` auf den Mountpoint (= leeres lokales Verzeichnis) `/users` zu mounten, gibt man als root Folgendes ein:
`mount -t nfs dukana:/nethome /users`.
Mount versucht nun Verbindung via RPC zum `rpc.mountd` auf `dukana` herzustellen. Der Mountdaemon am Server überprüft, ob der Client überhaupt berechtigt ist, auf das angeforderte Verzeichnis zuzugreifen und wenn ja, übermittelt er ihm einen sogenannten *File Handle*. Diesen benutzt nun der Client in allen Anforderungen für Dateien unterhalb von `/users`.
Greift nun ein Benutzer (nicht notwendigerweise root) auf eine Datei in `/users` zu, so sendet der Kernel des lokalen Rechners einen RPC Aufruf an `rpc.nfsd` (den NFS-Daemon) auf dem Server. In diesem Aufruf befindet sich der File Handle, der Name der gewünschten Datei und die UID und GIDs des Benutzers als Parameter. Diese werden gebraucht, um die Zugriffsrechte des Files auslesen zu können.

Der NFS-Client
Zunächst klären wir die genaue Syntax des `mount` Kommandos für NFS:

```
mount -t nfs [optionen] nfs_quelle lokales_verzeichnis
```

Für `nfs_quelle` verwendet man `nfs_server:entferntes_verzeichnis`. Da diese Notation eindeutig (gegenüber lokalen Mounts) ist, kann man `-t nfs` auch weglassen.
Es gibt zahlreiche Optionen, die man beim Mounten eines NFS-Verzeichnisses einstellen kann. Man kann sie mittels `-o` an der Kommandozeile übergeben oder für feste Mounts in `/etc/fstab` angeben. In beiden Fällen werden mehrere Optionen mit einem Beistrich getrennt und dürfen kein Leerzeichen enthalten. Die Optionen am Prompt überschreiben immer die in `/etc/fstab`.
Hier ein Beispieleintrag in `/etc/fstab` (vgl. auch Folie 78) :

```
#volume                 mount point     type    options
news:/var/spool/news    /var/spool/news nfs     timeo=42
```

NFS-Mounts

- Syntax, zB:

```
mount -t nfs dukana:/nethome /users
```

- `rpc.mountd`: vergibt File Handles für spätere Verwendung
- Server: `rpc.nfsd` liefert Dateien im per File Handle angegebenen Verzeichnis
- Mit `cat /proc/filesystems` überprüfen, ob der Kernel NFS überhaupt versteht

Folie 212

Alle Optionen von `mount` sind in der Manpage von `nfs` aufgelistet und erklärt. Wir stellen hier die wichtigsten vor.

- *rsize=n* bzw. *wsize=n* geben die Größe des Datenblocks bei Schreib- bzw. Lesevorgängen an, der angefordert wird. Normalerweise 1024 Byte, für optimierte Performance höher.

- *timeo=n* ist die Zeit in Zehntelsekunden, die der Client wartet, bis seine Anforderung erfüllt wird. Was passiert, wenn diese Zeit überschritten wird, hängt von den folgenden Optionen ab.

- *hard* setzt diese Quelle explizit als „hart“ gelinkt. D.h. der Client versucht nach einer Zeitüberschreitung noch einmal, die Anforderung zu versenden (und diesmal rechtzeitig zu empfangen). Es gibt kein Limit, wie oft der Client versucht, eine Antwort zu bekommen.

- *soft* Antwortet der Server hier nicht, gibt `rpc.mountd` dem aufrufenden Programm einen I/O Fehler zurück.

- *intr* Erlaubt es, eine NFS-Aufruf zu unterbrechen. Nützlich, wenn der Server nicht mehr reagiert.

Von `rsize` und `wsize` einmal abgesehen, betreffen alle Optionen das Verhalten des Clients, wenn die Verbindung zum Server abbrechen sollte. Sie arbeiten wie folgt zusammen: Sendet der Client eine Anforderung an den NFS-Server, so erwartet er, dass dieser nach einem gegebenen Zeitraum (mit der `timeo` Option) antwortet. Kommt keine Bestätigung vom Server, so sendet der Client die Anforderung erneut, diesmal mit dem doppelten Timeout. Ist der maximale Wert von 60 Sekunden erreicht, so gibt es ein großes Timeout. Standardmäßig wird dann eine Warnung ausgegeben und die Prozedur fängt von vorne an. Theoretisch geht das unendlich lange so weiter. NFS-Quellen, die ihre Anforderungen stur wiederholen, nennt man hart gemountet. Im Gegensatz dazu liefern weich gemountete Quellen einen Fehler zurück, wenn es ein großes Timeout gibt. Da normalerweise erst in den Buffer geschrieben wird, kommt der Fehler bei einem Schreibbefehl nicht an dem eigentlich auslösenden Prozess an. Daher kann man bei weich gemounteten Quellen nie sicher sein, dass die Schreiboperation auch wirklich erfolgreich war.

Welche dieser beiden Methoden man einsetzt, ist Geschmackssache. Allerdings sollte man Daten, die unbedingt konsistent bleiben müssen, hart einbinden. Wenn man zum Beispiel die X Programme über NFS einbindet, so sollte einem nicht die ganze Sitzung verloren gehen, nur weil die Verbindung kurz unterbrochen ist. Hingegen braucht man das FTP Archiv nur weich einzubinden, da sonst der ganze Prozess hängt, wenn die Verbindung nicht ganz so gut ist. Bei einem Server hinter einem Router sollte man die `timeo` Option unbedingt weiter hinauf setzen.
Hart gemountete Quellen stellen ein Problem dar, weil die Dateioperationen nicht unterbrochen werden können. Der Prozess steht dann, bis der Server wieder erreichbar ist. Der User kann nichts tun, um diesen Vorgang abzubrechen. Ist zusätzlich die `intr` Option gesetzt, so unterbricht der Prozess beim Empfang eines Signals jede Schreiboperation. Die Daten gehen dann aber verloren.

Der Automounter
In mittleren bis großen Netzen ist der Einsatz eines Dienstes, der NFS-Verzeichnisse (zB NFS-exportierte Homedirectories, aber auch lokale CDROMs und Floppies) erst bei Bedarf mountet und nach einem Timeout wieder unmountet besonders vorteilhaft. Das schützt zudem auch vor der Gefahr, die droht, wenn ein Server abstürzt bzw. nicht erreichbar ist, von dem man Verzeichnisse hart gemountet hat. Es gibt zwei Programmpakete, die diesen Dienst bereitstellen, `amd` und `automount`, das ursprünglich von SUN entwickelt wurde. Wir besprechen hier kurz das letztere.
`automount` wird über sogenannte Maps konfiguriert. Die Master Map `/etc/auto.master` gibt die den Mountpoints übergeordneten Directories und die dazugehörigen Maps an. In diesen wiederum sind die Mountpoints und die Devices samt Mountoptionen eingetragen. Beispieldateien finden sich auf der Folie.
Nach dem (Neu-)start des `autofs` Services (Initscript) reicht es dann aus, in ein automount-Verzeichnis (zB `/auto.mnt/floppy`) zu wechseln, um das dorthin gemappte Gerät/Verzeichnis zu mounten. Achtung, das Verzeichnis scheint *vorher* in einem Listing *nicht* auf, da es ja nicht gemountet ist. Dieses Problem kann man aber mittels Setzen symbolischer Links auf die Mountpoints umgehen.

NFS-Mountoptionen

- *rsize=n* und *wsize=n* Größe der Datenblocks beim Lesen/Schreiben
- *timeo=n* Timeout in 1/10-Sekunden angeben
- *hard* warten, bis der Server antwortet
- *soft* gibt nach dem timeout einen Fehler zurück
- *intr* erlaubt es, den Verbindungsaufbau wieder abzubrechen

Folie 213

Automounter

- Mounten nur so lange, wie etwas gebraucht wird
- Remote hart gemountete Verzeichnisse Risiko beim Crash des Servers
- Auch für Floppy, CD-ROM, ...
- Bei Zugriff auf Mountpoint wird Directory gemountet
- einfach konfigurierbar und stabil (Treiber im Kernel)
- `amd` und `automount`

Folie 214

Automounter Maps

```
[roli@pablo roli]$ cat /etc/auto.master
 # Format of this file:
 # mountpoint map options
 /auto.mnt       /etc/auto.mnt           --timeout 5
 /kusers         /etc/auto.kusers        --timeout 300

[roli@pablo roli]$ cat /etc/auto.mnt
 # This is an automounter map and it has the following format
 # key [ -mount-options-separated-by-comma ] location
 redhat   phobos:/dist/redhat-7.0
 cdrom    -fstype=iso9660,ro,exec,nosuid,nodev        :/dev/cdrom
 floppy   -fstype=auto,rw,sync,exec,nosuid,nodev,user :/dev/fd0
[roli@pablo roli]$ cat /etc/auto.kusers
 *       ken.macondo:/users/&
```

Folie 215

16.3 Network Information Service (NIS)

Das *Network Information Service* ist ein „Administrative Database System“ das es ermöglicht, Systemdateien eines Servers über ein Netzwerk für Clients verfügbar zu machen. Meist dient es dazu, die drei Dateien `/etc/passwd`, `/etc/group` und `/etc/shadow` eines Servers über ein Netzwerk zu verteilen und so einheitliche Accounts in einer sogenannten „Nis-Domäne“ (Server+Clients) zu definieren. In Kombination mit NFS gelingt es so, auch große Netzwerke übersichtlich zu verwalten und netzwerkweite Logins mit gleichen Homedirectories bereitzustellen. Solche kombinierten NFS/NIS-Systeme eignen sich zB besonders gut für Instituts/Firmennetze oder PC-Labors.

Entwickelt wurde NIS von SUN in den 80ern unter dem Namen „Yellow Pages“ (der dann aufgrund von Copyright Problemen mit der British Telecom geändert werden musste) und viele der Daemonen und Kommandos haben noch immer ein „yp“ im Namen. NIS ist Standard in der ganzen Unix-Welt und es können – ähnlich zu NFS – gemischte NIS-Domänen mit verschiedenen Unix-Varianten als Server und oder Clients betrieben werden. SUN vertreibt auch ein aus NIS weiterentwickeltes „Administrative Database System“ NIS+ das weniger verbreitet ist und von Linux nicht unterstützt wird. Ein alternatives System ist etwa LDAP (Lightweight Directory Access Protocol); siehe `http://www.openldap.org/`.

Gewöhnliche Anwendungen und Programme am System müssen nichts von NIS wissen, um es zu verwenden. Vielmehr erledigen die Funktionsaufrufe der Unix-C-Bibliothek diese Aufgabe (konfiguriert u.a. in `/etc/nsswitch`).

Funktionsweise
NIS-Dateien (die aus Performancegründen in einem binären Format gespeichert werden; sogenannte NIS-Maps) befinden sich gewöhnlich in `/var/yp/` in einem Unterverzeichnis mit dem Namen der NIS-Domäne. Jedes Mal, wenn man Änderungen an den über NIS-exportierten Dateien vornimmt (zB wenn man einen Benutzer hinzufügt), muss man nach `/var/yp/` wechseln und `make` ausführen. Damit werden die binären Dateien geupdated. Diese Prozedur wird man natürlich durch ein entsprechendes Script automatisieren.

NIS

- Auf RPC basierendes Client/Server System
- Systemdateien eines Servers auf Clients verfügbar machen
- Entwickelt von SUN in den 80ern; Unix-Standard
- Server+Clients in NIS-Domäne zusammengefasst
- Meist zur netzwerkweiten Benutzerverwaltung eingesetzt
- Kombination NIS/NFS ermöglicht netzwerkweite Logins mit gleichem Homedirectory
- Master/Slave-Server: Last verringern, Ausfallssicherheit

Folie 216

NIS-Dämonen Der NIS-Dienst wird server- und clientseitig durch die jeweiligen Daemonen vermittelt. Am Client muss das YP-Bind-Service (`ypbind`) und am Server *zusätzlich* das YP-Server-Service (`ypserv`) gestartet werden. Achtung ebenso wie bei NFS muss sowohl am NIS-Server als auch am NIS-Client der Portmapper laufen.

Der `ypbind` wird über die Datei `/etc/yp.conf` konfiguriert, die in etwa wie folgt aussieht:

```
domain meine.nis.domain server mein.nis.server
```

Außerdem ist es notwendig, dass alle NIS-Server in `/etc/hosts` eingetragen sind und der NIS-Domänenname gesetzt ist. Dieser kann händisch mit dem Kommando `nisdomainname meine.nis.domain` (ohne Argument gibt das Kommando den aktuell gesetzten NIS-Domänenname aus) eingegeben oder dauerhaft (unter RedHat) in `/etc/sysconfig/network` mittels Eintrags `NISDOMAIN=meine.nis.domain` konfiguriert werden.

Der YP-Server wird durch mehrere Dateien konfiguriert. In `/etc/yp.conf` werden einige globale Einstellungen vorgenommen. Meist muss an den Defaulteinstellungen nichts verändert werden – für Details siehe `man ypserv.conf`. In `/var/yp/securenets` ist eingetragen, welche Hosts berechtigt sind die Maps zu verwenden, also in die NIS-Domäne eingebunden sind. Eine Beispieldatei befindet sich auf Folie 217. Für weitere Details siehe die Manpages.

Will man, dass Benutzer ihre NIS-Passwörter – mit dem dafür vorgesehenen Befehl `yppasswd` – ändern können, muss serverseitig der `yppasswdd`-Daemon laufen. Dieser kann einfach mittels Initscripts gestartet werden und muss in den meisten Fällen nicht extra konfiguriert werden.

/etc/securenets

```
#
# securenets    This file defines the access rights to your NIS server
#               for NIS clients. This file contains netmask/network
#               pairs. A clients IP address needs to match with at least
#               one of those.
#               One can use the word "host" instead of a netmask of
#               255.255.255.255. Only IP addresses are allowed in this
#               file, not hostnames.
# Always allow access for localhost
255.0.0.0       127.0.0.0
# This line gives access to everybody. PLEASE ADJUST!
#0.0.0.0                0.0.0.0
# Clients
host            131.130.16.20
host            131.130.16.60
host            131.130.16.24
```

Folie 217

Um die Systemlast auf einem NIS-Server möglichst gering zu halten bzw. um bei Ausfall eines Servers Schäden und Verzögerungen möglichst gering zu halten, gibt es die Möglichkeit, mehrere *NIS-Slave-Server* einzurichten, die die vom *Master-Server* verteilten Dateien mit diesem synchronisieren. Für die Clients spielt es keine Rolle, ob sie die Dateien von einem Master- oder Slave-Server übernehmen.

Aufsetzen einer NIS-Domäne
Zuerst legt man einen NIS-Domainnamen (zB `praxis.compsys`) fest. Diesen teilt man nun dem/den Server(n) und den Clients mittels `nisdomainname praxis.compsys` mit (dauerhaft mittels Eintrag in Netzwerkkonfiguration; siehe oben).

- Master-Server
  Hier initialisiert man die Domäne und erstellt die zur Domäne gehörigen Dateien (Maps) und startet die Server. Welche Maps installiert werden wird in `/etc/ypserv.conf` festgelegt. Diese Datei kann meist in der Defaultform verwendet werden.

  ```
  /usr/lib/yp/ypinit -m
  service portmap start
  service ypserv start
  service ypbind start
  ```

  Dann müssen in der Datei `/var/yp/securenets` die IP-Adressen der Clients eingetragen werden, damit diese vom Server in der Domäne akzeptiert werden (siehe auch Folie 217). Wichtig ist, dass auch am NIS-Server immer der Clientprozess `ypbind` laufen muss, damit auch der Server in die Domäne eingebunden ist. Schließlich sollte der `yppasswdd`-Daemon gestartet werden und das Starten der YP-Daemonen (etwa mittels `chkconfig`) beim Systemstart eingerichtet werden.

- Slave-Server
  Der einzige Unterschied zum Master-Server ist die Option `-s`, der man den Hostnamen des Masters mitangeben muss:

  ```
  /usr/lib/yp/ypinit -s master_der_domain
  service portmap start
  ```

```
service ypserv start
service ypbind start
```

- Clients
  Hier genügt es, die `/etc/yp.conf` wie oben angegeben zu konfigurieren (also Server und Domäne anzugeben, je eine Zeile für den Master und jeden Slave-Server) und dann den `ypbind` zu starten:

  ```
  service portmap start
  service ypbind start
  ```

Wichtige NIS-Kommandos sind auf der letzten Folie des Kapitels zusammengestellt. Schließlich folgt der Verweis auf das NIS-HOWTO für alle weiteren Details.

NIS verwenden

- Dateien in `/var/yp`
- Nach Änderung exportierter Daten dort `make` aufrufen.
- `ypserv`, `ypbind`, `yppasswdd` Services
- Portmapper muss am Client und Server gestartet sein
- Client Konfiguration in `/etc/yp.conf`
- Server Konfiguration in
  - `/etc/ypserv.conf`
  - `/var/yp/securenets`

Folie 218

NIS-Domäne erstellen

- `nisdomainname` setzen
- Master-Server: `ypinit -m`,
  `portmap`, `ypserv`, `ypclient`, `yppasswdd` starten
- Slave-Server: `ypinit -s master`, Services wie oben
- In `/var/yp/securenets` Clients eintragen
- Clients: `/etc/yp.conf` konfigurieren, `portmap` und `ypbind` starten

Folie 219

NIS-Kommandos

- `ypwhich` mit welchem NIS-Server verbunden?
- `ypcat exp` zeigt Inhalt einer exportierten Map
- `yppasswd user` ändert das Passwort von `user`
- `ypchfn user` ändert Informationen über `user`
- `ypchsh user` ändert die Shell von `user`

Folie 220

17 FTP- Web- und Mailserver

In diesem Kapitel besprechen wir drei wichtige Internetservices resp. ihre Server-Implementierungen unter Linux; wir stellen den FTP-Server `proftp`, den Webserver `apache` und den Mailserver `sendmail` vor.

17.1 FTP-Server

FTP (File Transfer Protocol) – obwohl eines der ältesten Internetservices – erfreut sich noch immer großer Beliebtheit. Abgesehen von interner Verwendung (zB als eine zentralisierte Rechner-Installationsmöglichkeit) beschränkt sich der Einsatz heute hauptsächlich auf das sogenannte anonyme FTP, das Benutzern, die keinen Account auf dem jeweiligen Rechner(-netz) besitzen, das Herunterladen bestimmter öffentlicher Daten gestattet. Zu diesen Daten gehören hauptsächlich Programmpakete, Bugfixes, Dokumente oder Musikdateien. Der wesentliche Vorteil gegenüber dem HTTP-Protokoll ist, dass der User selbst mit gewohnten Unix-Befehlen durch den Verzeichnisbaum navigieren darf, wogegen man auf einer Seite am Webserver immer einen Link auf zB neu hinzugekommene Dateien setzen muss, um sie dem Benutzer bekannt zu machen.
Eine besondere Warnung sei hier – noch einmal (vgl. Abschnitt 15.3) – bezüglich des nicht-anonymen (also benutzerbasierten) FTP gegeben: das Passwort, das der User eingibt, geht unverschlüsselt über die Datenleitung und kann mitgesnifft werden. Obwohl viele Seiten im Internet nicht-anonymes FTP zB für Webseitenaktualisierung ermöglichen, raten wir prinzipiell davon ab und wollen uns im Folgenden nur dem anonymen FTP-Server widmen.

FTP

- File Transfer Protocol
- Einer der ältesten Dienste
- Öffentliche Bereitstellung von Daten
- Interne Verwendung für zB Rechner-Installation
- Anonym – nicht-anonym (unsicher!)
- Vom User selbst navigierbar

Folie 221

Verschiedene Server-Pakete

- `wuftpd` Lange Zeit Standard, komfortabel, unsicher
- `ncftpd` Komfortabel, relativ sicher, keine freie Software, keine offenen Sourcen
- `proftpd` Umfangreich konfigurierbar, sicher
- Andere Server (meist nur anonym): `anonftpd`, `vsftpd`, `trollftpd`, ...

Folie 222

Der FTP Server, der früher besonders beliebt war (weil er einfach zu konfigurieren und bei den meisten Distributionen enthalten ist), ist der `wuftpd`, der an der Washington University entwickelt wurde (daher auch der Name) und traditionellerweise vom Internetdämon gemanagt wird. Gerade in letzter Zeit hat man immer wieder große Sicherheitslücken darin entdeckt und auch die ersten Linux-Würmer nutzen unter anderem seine Schwächen aus. Sicherere Alternativen sind der `ncftpd` und der `proftpd`. Da ersterer kommerziell ist, man sich den Sourcecode nicht ansehen darf und er außerdem unter einer besonderen Lizenz steht, die nur für „Educational Use“ freien Einsatz gestattet, beschäftigen wir uns in Folgenden nur mit dem `proftpd`, der noch dazu dem Apache Webserver sehr ähnliche Konfigurationsdateien hat. Neben diesen beiden „Big Players“ stehen noch kleinere spezialisierte FTP-Server zur Verfügung (`oftpd`, `anonftpd`, `trollftpd`, ...), die im Allgemeinen auch relativ sicher sind (einige erlauben sogar überhaupt keine auf Dateien schreibend zugreifende Aktionen und stellen daher vermutlich das Optimum an Sicherheit, jedoch nicht an Komfort dar). Eine neuere Entwicklung ist der `vsftpd`, der schon vom frühesten Stadium an in Hinblick auf höchste Sicherheit programmiert wurde.

Der Proftp-Server

`proftpd` wird bei den meisten Distributionen leider nicht mitgeliefert, sodass wir das entsprechende Paket aus dem Internet herunterladen müssen. Egal, ob man dabei die Sourcen selbst compiliert, das RPM- oder Debian-Paket nimmt, man erhält direkt nach der Installation einen bereits gut vorkonfigurierten FTP Server, dessen Konfigurationsdatei `/etc/proftpd.conf` ist. Genauer beinhaltet das proftpd-standalone Zusatzpaket diese Konfigurationsfile und ein entsprechendes Initscript. Der `proftpd` kann demzufolge sowohl als Standalone Server als auch aus dem Internetdämon heraus verwendet werden.

Im Prinzip reicht es aus, den Server zu starten, um einen voll funktionstüchtigen FTP Server vorzufinden, mit dem man sich sogleich mittels `ftp localhost` verbinden kann:

```
martin@notebook:~$ /usr/bin/ftp localhost
Connected to localhost.
220 ProFTPD 1.2.1 Server (Debian) [notebook]
Name (localhost:martin): anonymous
331 Anonymous login ok, send your complete email
    address as your password.
Password: (Mailadresse eingeben)
230 Anonymous access granted, restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Auffallend ist hier, dass der Server scheinbar in der Standardeinstellung auch nicht-anonymen Zugang (er schlägt als FTP Benutzername `martin` vor) erlaubt. Um sich das lästige Eintippen von `anonymous` und Mailadresse bei jeder Connection zum Server zu ersparen, sollte man übrigens den `ncftp` Client verwenden, der auch ansonsten durch große Benutzerfreundlichkeit glänzt (Tab-Completion, etc., siehe auch Abschnitt 15.3).
Am `ftp>` Prompt kann man nun wie gewohnt Unix-Befehle wie `ls` oder `cd` eingeben, die der FTP-Server im Repository (das die Daten, die man freigibt, enthält und gewöhnlich das Homedirectory des Systembenutzers `ftp` ist, also etwa `/home/ftp`) ausführt. Eine Auflistung aller möglichen Befehle bekommt man durch Eingabe von `help`, spezifische Hilfe zu einem Kommando mit `help kommando` (zB `help mget`).

ProFTPD

- in vielen Distributionen nicht enthalten
- `http://www.proftpd.net`
- Pakete für viele Distributionen verfügbar
- Gut vorkonfiguriert
- Umfassend konfigurierbar (`/etc/proftpd.conf`)
- genau auf Bugs „durchleuchtet“
- Läuft als nicht-privilegierter User

Folie 223

Konfiguration
Wir wollen nun die wichtigsten Optionen der Konfigurationsdatei herausgreifen und Schritt für Schritt durchgehen:

```
ServerName                      "Mein FTP Server"
ServerType                      standalone
```

Die `ServerName` Direktive ist mehr von kosmetischer Natur; sie zeigt beim Einloggen am Server den eingestellten Namen neben der ProFTPD Versionsnummer. Der `ServerType` dagegen bestimmt, ob der Server „standalone“ bzw. über den `(x)inetd` gestartet werden soll.

```
TimeoutNoTransfer               600
TimeoutStalled                  600
TimeoutIdle                     1200
```

Viele Benutzer verbinden mit einem FTP Server, laden etwas herunter und vergessen dann, die Verbindung auch wieder zu trennen. Da es natürlich ein Maximum an gestarteten Serverprozessen geben muss, könnten zB 30 vergessliche User einen FTP Server lahmlegen. Damit es gar nicht soweit kommt, definieren wir Timeouts (in Sekunden) für verschiedene Aktionen, nach denen der Benutzer automatisch einen Disconnect bekommt.

```
DisplayLogin                    welcome.msg
DisplayFirstChdir               .message
LsDefaultOptions                "-l"
```

Wieder mehr kosmetischer Natur sind diese drei Optionen, die jeweils beim Login bzw. beim Wechsel in ein Verzeichnis eine Meldung ausgeben, die in der besagten Datei liegt. Das kann auf großen FTP-Servern mit zB Spiegelungen von Linux-Distributionen bei der Navigation enorm von Vorteil sein. Die letzte Direktive sagt aus, dass der Benutzer auf jeden Fall bei der Eingabe von `ls` ein langes Listing bekommen soll (was im Allgemeinen erwünscht sein dürfte, da man bei FTP sicher gerne die Dateigröße erfährt).

```
Port                            21
```

Dieser Punkt sollte selbsterklärend sein. Eine interessante Möglichkeit, um den Server ein wenig mehr zu schützen, ist, ihn auf einen anderen Port zu legen. Zu viel Steigerung der Sicherheit sollte man sich davon nicht erwarten, denn gegen Tools wie `nmap` ist auch das machtlos. Zumindest aber sichert man sich gegen zufällige Verbindungen von neugierigen Menschen besser ab.

```
MaxInstances                    30
```

Hier kann man die maximale Anzahl an Prozessen für den FTP Server begrenzen. Bei kleineren Servern ist der Defaultwert sicher akzeptabel; für größere sollte man ihn entsprechend erhöhen.

```
User                            nobody
Group                           nogroup
```

Dies sind zwei sicherheitstechnisch wichtige Optionen, die festlegen, unter welchem Benutzer der FTP-Daemon laufen soll. Natürlich muss er zum Starten und Binden auf den (privilegierten) Port 21 root Rechte haben, diese legt er aber danach ab, was ihn auch relativ unempfindlich gegen „Buffer Overflow“ Attacken macht, denn im schlimmsten Fall erhält der Hacker eine nobody Shell statt einer root Shell.
Die generellen Optionen sind damit abgeschlossen, wir wenden uns den verzeichnisspezifischen Direktiven zu und da den Regeln für anonymes FTP.

```
<Anonymous ~ftp>
  User                          ftp
  Group                         nogroup
  UserAlias                     anonymous ftp
```

Die erste Zeile sagt uns, dass die folgenden Regeln bis zum schließenden „Tag“ nur für anonyme Connections und das Homeverzeichnis des ftp Accounts gelten sollen. Und da setzen wir auch gleich den Benutzer auf `ftp`, die Gruppe auf `nogroup`, d.h. wenn sich ein User mit dem FTP-Server anonym verbindet, ist er in Wirklichkeit am Server der Benutzer `ftp` (Gruppe `nogroup`). Außerdem wollen wir erreichen, dass auch der Benutzername `anonymous` am Login Prompt akzeptiert wird, was man mit `UserAlias` bewirkt.

```
<Directory *>
 <Limit WRITE>
   DenyAll
 </Limit>
</Directory>
```

Die obigen Zeilen entziehen dem anonymen FTP-User jedwege Schreibberechtigung, d.h. auch wenn der Benutzer `ftp` irgendwo im über FTP erreichbaren Bereich des Dateisystems Schreibrechte hat, werden diese dem anonymen FTP-User entzogen.

```
</Anonymous>
```

beendet die `Anonymous` Sektion und zugleich die gesamte Konfigurationsdatei.

Nützliche FTP Kommandos

Zum Abschluss dieses Abschnitts stellen wir noch einige nützliche FTP-Befehle (sowohl server- als auch clientseitig) vor (siehe auch Folie 225).

- `ftpwho` zeigt Informationen über die gerade per FTP eingeloggten Benutzer an
- `ftpcount` zählt die gesamten per FTP eingeloggten User
- `ftpshut time [warn]` beendet zur angegebenen Zeit alle FTP Serverprozesse und gibt optional eine Warnung aus (zB `ftpshut +3 Server down for maintainance` deaktiviert den Server in 3 Minuten)

Proftpd Konfiguration

```
ServerName          "Mein FTP Server"
ServerType          standalone
TimeoutNoTransfer   600
TimeoutStalled      600
TimeoutIdle         1200
DisplayLogin        welcome.msg
DisplayFirstChdir   .message
LsDefaultOptions    "-l"
Port                21
MaxInstances        30
User                nobody
Group               nogroup
<Anonymous ~ftp>
  User              ftp
  Group             nogroup
  UserAlias         anonymous ftp
<Directory *>
 <Limit WRITE>
   DenyAll
 </Limit>
</Directory>
</Anonymous>
```

Folie 224

FTP Kommandos

- Serverseitig
  - `ftpwho` Infos über FTP-Benutzer
  - `ftpcount` Anzahl der FTP-User
  - `ftpshut` FTP-Server deaktivieren
- Clientseitig
  - `ftp` Standard, wenig komfortabel
  - `ncftp` Sehr bequem, Filename-Completion
  - `kbear` Komfortabel, Graphisch

Folie 225

17.2 Der Apache Webserver

Das wohl bekannteste Service im Internet ist das World Wide Web (WWW) oder Hyper Text Transfer Protocol (HTTP), das es gestattet, mittels Client, d.h. Webbrowser (Internet Explorer, Konqueror, Opera, Mozilla, Netscape, ...) Text-, Grafik-, Sound-, etc. Files aus dem Netz herunterzuladen und sogenannten Hyperlinks von einem Server zum anderen zu folgen.

Das HTTP-Service wird von Webservern bereitgestellt. Obwohl es mehrere Anbieter gibt, ist der (Open Source) Webserver *Apache* mit ca. 55% Marktanteil der am weitesten verbeitete Server und wird in (fast) jeder Linux-Distribution mitgeliefert; wir konzentrieren uns daher ausschließlich auf ihn.

Der „A PAtCHy sErver“ wurde 1995 aus dem NCSCA (National Center for Supercomputing Applications) HTTPD mittels vieler bereits vorhandener Patches entwickelt; außerdem ist der Name eine bewusste Anspielung auf den indigenen Stamm. Derzeit wird der Server von der Apache Software Foundation, einer Nonprofit Organisation, unter einer GPL-artigen Open Source Lizenz weiterentwickelt; die letzte Version ist 1.3.24. resp. 2.0.35. Für alle Detailinformationen verweisen wir auf die Apache Homepage unter `http://www.apache.org`.

Apache ist nicht nur für alle gängigen Unix-Systeme, sondern auch für MacOS, OS/2 und sogar Windows (95–ME, NT4.0–2000) frei verfügbar und in fast alle Linux-Distributionen sehr gut integriert.

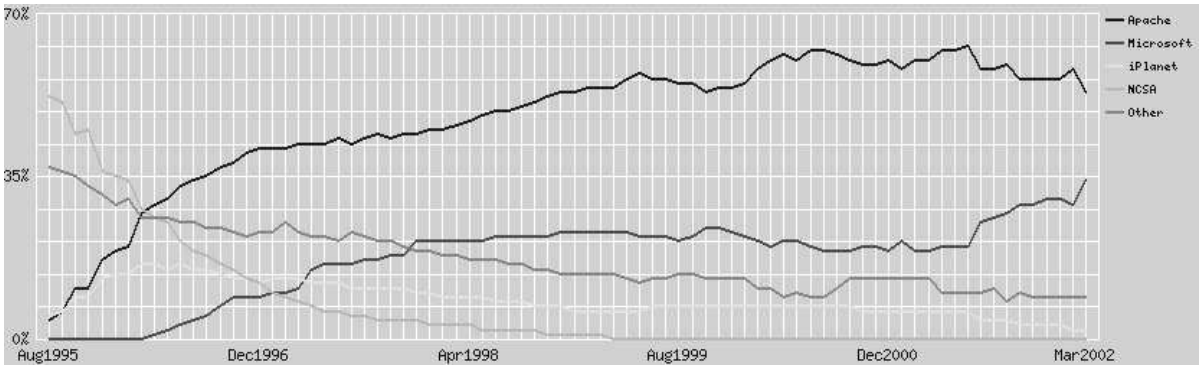
Webserver

- Server für WWW-Hypertext-Dokumente
- verwendet HTTP-Protokoll (Hyper Text Transfer Protocol)
- sendet
  - Text (vor allem HTML)
  - Grafik
  - Sound, ...

  an Client (Browser)
  - Internet Explorer
  - Konqueror
  - Mozilla, ...

Folie 226

HTTP-Server Marktanteile



Folie 227

Der Apache Webserver

- A PAtCHy sErver aufbauend auf NCSA-HTTPD
- opensource, Apache Licence
- für gängige Plattformen (Unix, NT, Mac, OS/2,...)
- eng an HTTP-Standards orientiert
- `http://www.apache.org`
- aktuelle Version: 1.3.24, 2.0.35
- Erweiterungen: PHP-, Perl-Module, Java-Servlets, CGI-Scripts

Folie 228

Apache für (Redhat) Linux

- RPMs erhältlich
  - `apache`
  - `apache-manual`
- Daemon `/sbin/httpd`
- Initscript `/etc/init.d/httpd`
- Konfigurationsdateien `/etc/httpd/conf/`
- Logfiles `/var/log/httpd/`
- Bibliotheken, Module `/usr/lib/apache/`
- Server Root `/var/www/` (früher `/home/httpd/`)

Folie 229

Meist wird der Apache als Standalone Server betrieben und mittels des entsprechenden Initscripts `/etc/init.d/httpd` gestartet; Nach der Installation (mittels `rpm` oder Tarball) steht bereits eine gut dokumentierte Defaultkonfiguration zur Verfügung und wir können den Server starten. Der erste Test ist nun, sich mittels Browser auf das Loopback Device zu verbinden (`http://127.0.0.1/`). Wenn alles geklappt hat, kommt man auf die Default Willkommensseite des lokalen Apache-Servers, wo bereits Links zur mitgelieferten Online Dokumentation (falls installiert) bzw. zur Apache Homepage eingetragen sind.

Wir besprechen im folgenden die grundsätzlichen Konfigurationsoptionen des Apachen und verweisen für Details auf die Homepage bzw. auf die sehr angenehm zu lesende „Apache Administrationsbibel“ [14].
Die wichtigsten Einstellungen für den laufenden Betrieb von Apache werden in der Datei `httpd.conf` gemacht, welche sich je nach Distribution im Verzeichnis `/etc/` oder in einem Apachespezifischen Unterverzeichnis zB `/etc/httpd/conf/` befindet.
Zunächst befassen wir uns mit der Funktionsweise eines Webservers. Ein User gibt eine HTTP-Adresse (zB `http://www.univie.ac.at`) in seinem Browser an. Über einen Nameserver bzw. über eine Hosts-Datei wird die IP-Adresse ermittelt und ein Request an den Server gesendet; (standardmäßig) an Port 80. Dort wartet der Apache auf Anfragen. Im Konfigurationsfile `httpd.conf` müssen wir beides eintragen; den Namen des Servers sowie seine IP-Adresse und den Port.
Erreicht den Server der Request so kommt der Eintrag `DocumentRoot /var/www/html` im Konfigurationsfile zum tragen; er definiert das Haupt(dokumenten)verzeichnis des Webservers. Weiters kann man angeben, welche Datei er aus diesem Verzeichnis an den User schicken soll, wenn nicht explizit eine Datei angegeben wurde. Das ist in den meisten Fällen `index.html`. Es können (optional) auch mehrere Dateien angegeben werden, zB `index.php` etc. Fassen wir die bisherigen Einträge zusammen (Kommentare werden durch „`//`“ markiert).

```
ServerName      www.studentenfutter.at  //ServerName
Listen          08.15.47.11:80          //IP Adresse und Port
DocumentRoot    /var/lib/apache/htdocs
DirectoryIndex index.html index.htm index.php index.shtml
```

Apache Installation

- installieren: `rpm -ihv apache.sowieso.rpm`
- Achtung: trägt sich in `/etc/rc[3,5].d` ein
- starten: `/etc/init.d/httpd start`
- 1. Test: Browser auf `http://127.0.0.1`

  „Willkommensseite“
- Link auf Dokumentation, Info

Folie 230

Außerdem verwendet unser Apache auch einen eigenen Useraccount (vgl. FTP-Server);wir tragen also ein:

```
User apache
Group apache
```

Achtung der Account für den Webserver kann je nach Distribution verschieden heissen; im Normalfall sollte die Einstellung aus dem Defaultkonfigurationsfile verwendet werden.
Wesentlich hierbei ist, dass Apache unter diesen Berechtigungen die Dateien holt resp. liest. Dürfte der User `apache` nicht auf `index.html` zugreifen würde der User am Browser eine Fehlermeldung angezeigt bekommen. In dieser Fehlermeldung erscheint auch die Adresse des Serveradmins; einzutragen mit:

```
Serveradmin ich@bins.net
```

In unserem einfachen Fall hat der User eine HTML-Datei geholt, die als ASCII-Datei auf dem Server liegt. Der Server sendet diese Datei an den Browser und dieser stellt sie dar.

Weitere wichtige Konfigurationsdirektiven des Apachen sind

- `UserDir` bezeichnet den Namen des Standard Verzeichnisses, in dem die User des Unix-Systems ihre Homepages ablegen können; default: `public_html`; diese Seiten sind dann unter `http://www. studentenfutter.at/~user` erreichbar.
- `ErrorLog` bezeichnet das Fehlerlogfile; default: `/var/log/httpd /error_log`
- `LogLevel` analog zu den Priorities für den Syslog Daemon
- `CustomLog` gibt das Access-Logfile an (jeder Zugriff wird mitgeschrieben); default: `/var/log/httpd/access_log`

Der Apache kennt auch eigene Berechtigungskonzepte. Man kann in der `httpd.conf` mit

```
<Directory /var/lib/apache/WWW/studentenfresser>
Options
....
</Directory>
```

Zugriffsberechtigungen pro Verzeichnis setzen, eigene Gruppen und User für den Apache einrichten etc. Diese sind unabhängig von den Gruppen und Usern des Betriebssystems. Außerdem können mittels `Options` Direktive pro Directory weitere Optionen angegeben werden.

Bei der Apache Konfiguration ebenfalls sehr wichtig sind die *Module*, die hauptsächlich für dynamische Webseiten eingebunden werden müssen. Der Unterschied ist, dass die HTML-Datei nicht am Server liegt, sondern erst bei einem Request dynamisch erstellt wird, wie ein kleines Programm, das statt purem Text dann eben HTML-Code ausgibt. Beliebte Programmiersprachen dafür sind Perl oder PHP.

Für die weiteren wichtigen und sicherheitsrelevanten Punkte CGI-Scripts (Common Gateway Interface) bzw. SSL (Secure Socket Layer) verweisen wir auf die Online Dokumentation, die einschlägigen HOWTOS und das Buch [9].

Außerdem ist ein GUI zur Konfiguration des Apachen namens Comanche (**Co**nfiguration **Mana**ger for Apa**che**) verfügbar (siehe auch Folie 233).

Apache Konfiguration

- Online-Dokumentation, Homepage
- gut dokumentierte Beispielfiles
- `/etc/httpd/conf/httpd.conf` Basiskonfiguration
- `/etc/httpd/conf/access.conf` (veraltet)
- `/etc/httpd/conf/srm.conf` (veraltet)
- GUI Comanche (**Co**nfiguration **Mana**ger for Apa**che**)

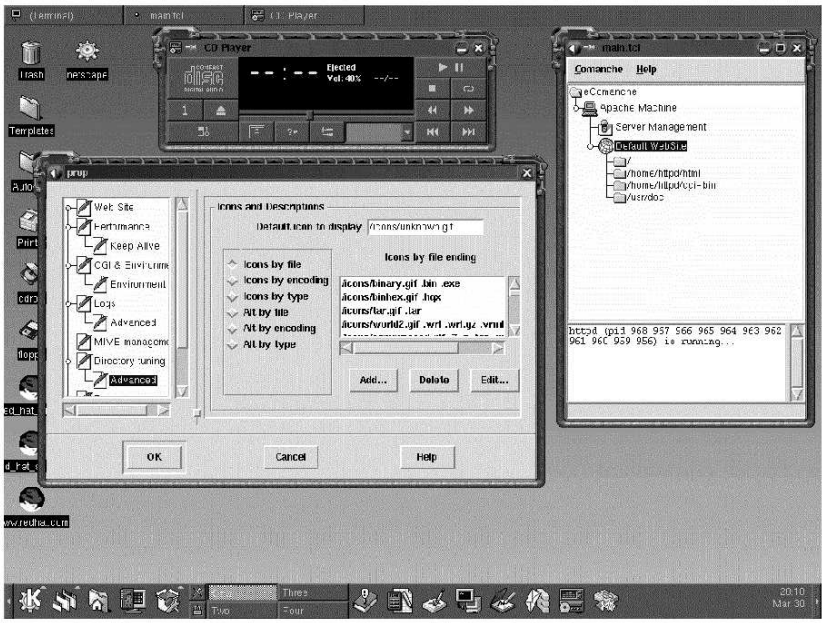
Folie 231

/etc/httpd/conf/httpd.conf

```
ServerType standalone
Port 80
HostnameLookups off
User nobody
Group nobody
ServerAdmin root@localhost
ServerRoot /etc/httpd
ErrorLog logs/error_log
LogLevel warn
LoadModule env_module  modules/mod_env.so
LogFormat "%h %l %u %t %>s %b" common
Timeout 300
KeepAlive On
MaxKeepAliveRequests 100
StartServers 10
MaxClients 150
MaxRequestsPerChild 100
```

Folie 232

Comanche



Folie 233

17.3 Der Sendmail Mailserver

Neben dem Surfen im World Wide Web ist das Kommunikationsmittel Email der am häufigsten genutzte Dienst im Internet. Laut verschiedener Schätzungen liegt die weltweite Anzahl versendeter Emails bei weit über fünf Milliarden täglich bzw. einem Drittel des gesamten Datenverkehrs des Internet.

Aus diesen Daten wird der besondere Anspruch an einen Mailserver (und nicht zuletzt an seinen Administrator, der auch Postmaster genannt wird) sichtbar. Als besonders stabil, performant und flexibel hat sich der Mailerdämon `sendmail` erwiesen, der heute einen Anteil von über 60% aller im Internet laufenden Mailserver innehält. Dieser Dämon läuft am Server auf Port 25 und nimmt Daten von einem Netzwerkclient entgegen, die dem Format des SMTP (Simple Mail Transfer Protocol) entsprechen (für Wissbegierige: Die Funktionsweise von SMTP wird in RFC 821 festgelegt, das etwa über `http://www.sendmail.org/rfc/0821.html` erhältlich ist). Im gängigen Administratorjargon wird ein Maildämon übrigens häufig MTA (Mail Transfer Agent) - im Gegensatz zum MUA (Mail User Agent), der Client, mit dem der Benutzer seine Email-Nachrichten verfaßt - genannt. Bevor wir zur Konfiguration von `sendmail` kommen, muß aber die Funktionsweise des Mailverkehrs erläutert werden.

Mailserver - `sendmail`

- über 5 Milliarden Emails täglich
- sendmail über 60% Marktanteil
- `http://www.sendmail.org`
- stabil, performant, flexibel
- lauscht auf Port 25
- kommuniziert über SMTP
- SMTP definiert in RFC 821
- Mail Transfer Agent

Folie 234

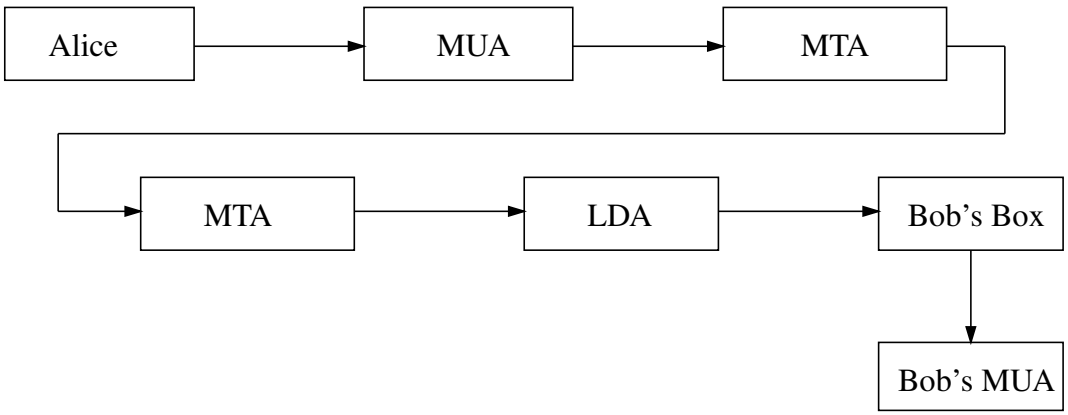
Funktionsweise

Ohne uns in die Untiefen der Protokollebene zu stürzen, wollen wir hier kurz den Weg einer Email-Nachricht vom Verfasser (Alice) zum Empfänger (Bob) nachvollziehen.

Alice öffnet ihren MUA (zB mutt, kmail oder pine), tippt dort ihre Nachricht und eine Empfängeradresse ein und teilt dem Programm mit, daß sie die Nachricht nun versenden möchte. Dieses verpackt den Text zusammen mit einem Header, der unter anderem Sender, Empfänger, Betreff, Datum und einige andere Informationen enthält, in das, was man gewöhnlich eine Email nennt. Sodann wird, je nach Konfiguration des MUA, auf unterschiedliche Weise - über den lokalen MTA oder über einen Zwischenmailserver (Smarthost) - eine Netzwerkverbindung auf Port 25 des Empfängers geöffnet und nach einer Anmeldeprozedur die Email auf den Zielrechner übertragen. Dort übergibt sie der MTA dem LDA (Local Delivery Agent), der die Email in das System-Mailverzeichnis legt (gewöhnlich `/var/spool/mail/`). Ein sehr populärer und mächtiger LDA ist `procmail`. Er kann auch zusätzlich verschiedene Dinge überprüfen, zB ob die empfangene Mail Spam ist, ob Mailanhänge Viren enthalten o.ä.. Bob loggt sich nun zum Lesen seiner Emails entweder direkt am Empfängerrechner ein oder lädt Mails mittels des POP3- oder IMAP-Protokolls vom Server auf seinen Rechner zuhause herunter.

Der Weg vom Sender-MTA zum Empfänger-MTA ist übrigens wesentlich komplexer als es hier den Anschein hat, der Pfad einer Email geht oft auch über mehr als 3-4 Rechner, etwa wenn der Zielrechner hinter einer Firewall steht und der Firewallserver SMTP-Anfragen von außen nach innen weiterleiten muß. Eine ähnliche Situation liegt vor, wenn man einen Rechner (MX-Host genannt) definiert, der Mails für ein Rechnernetz empfangen soll (was auch immer er dann genau tut, zB ob vom Senderrechner überhaupt Emails akzeptiert werden sollen), bevor sie am Zielort angelangen. Im allgemeinen sind aber solche Zwischenschritte für den Sender oder Empfänger transparent und durch das Ansehen der Email-Header nachvollziehbar.

Funktionsweise von SMTP



Folie 235

Konfiguration
`sendmail` war bis vor einigen Jahre für seine sehr komplexe Konfiguration über die berühmt-berüchtigte Datei `/etc/sendmail.cf` gefürchtet (zur „Abschreckung“ findet sich auf Folie 237 ein Auszug einer `/etc/sendmail.cf` eines Produktionsservers). Heute wird üblicherweise nicht mehr diese Datei direkt bearbeitet, sondern mittels der Makrosprache `m4` aus dem File `/etc/mail/sendmail.mc`, das ein erheblich vereinfachtes Format hat, eine `sendmail.cf` erstellt.

Im Verzeichnis `/etc/mail/` liegen gewöhnlich noch einige andere Dateien, von denen der Großteil aber nur für fortgeschrittene Aufgaben eingesetzt wird. Lediglich `local-host-names`, in die alle Hostnamen des Mailservers eingetragen werden sollten, für die man zusätzlich zum eigentlichen Hostnamen Mail empfangen will (zB weil am Rechner ein Virtual Web Host definiert ist), und `access`, um bestimmten Rechnern oder Benutzern Zugang zum `sendmail`-Dämon zu ermöglichen oder verbieten.
Schließlich ist noch das File `/etc/aliases` zu nennen, in dem Empfängernamen definiert werden können, die an andere Email-Adressen weitergeleitet werden sollen (zB werden so gewöhnlich Emails für `webmaster` an `root` geschickt). Dies ist auch dann nützlich, wenn Benutzer ihren Account am Server aufgegeben haben und ihre Emails an eine neue Adresse weitergeleitet werden sollen. Auf diesem Wege kann man auch Mails für einen Empfänger auf mehrere Email-Adressen weiterleiten und so einfach Mailinglisten erstellen.

Achtung: Wenn man Änderungen an der `aliases` vornimmt, muß man das Programm `newaliases` ausführen, damit diese aktiv werden. Generell sollte man vor Änderungen in allen Konfigurationsdateien die jeweilige Manpage lesen und alle Modifikationen genau überprüfen, da `sendmail` bei ungültigen Einträgen in einem seiner Konfigurationsfiles meist den Start verweigert und das gerade bei einem so wichtigen Dienst vermieden werden muß.

`sendmail`-Konfiguration

- `/etc/sendmail.cf` komplex, undurchschaubar
- `/etc/mail/sendmail.mc` einfacher, übersichtlicher
- `/etc/mail/local-host-names` zusätzliche Hostnamen des Rechners, für die Mail empfangen werden soll
- `/etc/mail/access` Zugriffssteuerung auf Benutzer/Rechner-Ebene
- `/etc/aliases` Empfänger-Aliases

Folie 236

Beginn einer `sendmail.cf`

```
Cwlocalhost
Fw/etc/mail/local-host-names
CP.
DS
CO @ % !
C..
C[[
C{Accept}OK RELAY
Kresolve host -a<OK> -T<TEMP>
```

Folie 237

Beispiel `sendmail.mc`

```
include(‘../m4/cf.m4’)
VERSIONID(‘M4 Example Configuration’)dnl
OSTYPE(linux)dnl
FEATURE(nouucp)dnl
FEATURE(always_add_domain)dnl
define(‘LUSER_RELAY’,‘local:postmaster’)dnl
define(‘SMART_HOST’, mail.domain.at)dnl
MAILER(local)dnl
MAILER(smtp)dnl
```

Folie 238

Das `sendmail`-Paket enthält auch verschiedene Clientprogramme. Eines davon ist das eigentliche `sendmail`-Binary, das zum Versenden von Mails verwendet wird. Gleichzeitig enthält es, wenn bestimmte Optionen übergeben werden, auch den Programmcode für den Server-Dämon. Man kann übrigens mit `sendmail` Mails von der Kommandozeile aus verschicken, indem man die Ausgabe eines anderen Programms nach `sendmail` pipet und als Argument (mindestens) eine Mailadresse übergibt; also etwa:

```
echo "Hallo" | sendmail martin@mat.univie.ac.at
```

Ein besonders bei Problemen nützliches Hilfsprogramm ist `mailq`, mit dem man die Warteschlange für noch zu versendende Emails einsehen kann. Meist bleiben dort Mails hängen, bei denen zB der DNS-Name des Zielsystems ungültig ist. Diese Mails werden gewöhnlich nach einer bestimmten Wartezeit, in der alle paar Stunden der Versand wieder versucht wird, aus der Warteschlange entfernt und der Absender benachrichtigt.

Für eine genauere Beschreibung von `sendmail` verweisen wir angehende Postmaster auf jeden Fall zumindest an die diversen Manpages, die `sendmail`-Dokumentation, das Mail-Administrator-HOWTO und die ausgezeichneten Bücher [10] und [13] um den Überblick über dieses doch einigermaßen komplexe Programm zu behalten und bei Problemen „das richtige“ zu tun. `sendmail` führt auch eine hilfreiche Log-Datei, die bei Redhat Linux in `/var/log/maillog` zu finden ist und bei vielen Fehlern Aufschluß über die Ursachen bringen kann.

18 Netzwerksicherheit

Häufig hört man als Antwort auf Kritik über mangelnde Sicherheit von Computersystemen die rethorische Frage: „Was haben wir schon für wichtige Daten?“ oder: „Wir haben ohnehin eine Firewall!“ Im folgenden soll darauf eingegangen werden, warum diese Aussagen unüberlegt sind und wovor man sich schützen soll.

18.1 Allgemeines

Jedes Computersystem hat einen gewissen Erhaltungswert. Dieser kann sehr hoch sein, wie zB im Fall von gespeicherten sensiblen Daten oder Systemen deren Ausfall Geld oder Leben kostet, oder sehr niedrig sein. Er ist aber fast nie gleich null. Die Wiederherstellung eines Systems kostet zumindest ein bißchen Zeit. Des weiteren werden durch fahrlässig administrierte Computer oft andere Teile eines Computernetzes in Mitleidenschaft gezogen. Es gibt in Zeiten von automatischen Angriffsmechanismen (wie etwa den Internetwürmern Codered und Nimda) auch keinen Computer im Internet der für niemanden interessant ist. Automaten unterscheiden das nicht! Schlecht administrierte Computer könnten in Zukunft das Internet in seiner jetzigen Form unbenutzbar machen.
Hinter der Absicherung und dem Angriff auf ein Computersystem stehen in keiner Weise mystische Geheimnisse, wie oft suggeriert wird. Ganz im Gegenteil: Ein klares technisches Verständnis der Vorgänge und ein eher tabellarisches Wissen von bekannten Angriffspunkten verschiedener Betriebssysteme und Server ist von Nöten.

Allgemeines

Warum Sicherheit?

- Erhaltungswert des Systems
- Schutz vor Angriffsautomaten
- Andere nicht in Mitleidenschaft ziehen

Was ist bei der Absicherung wichtig?

- Ordentliche Administration
- Testen auf Sicherheitslöcher
- Gutes Verständnis der Netzwerktechnik

Folie 239

18.2 Updates

Typischerweise nutzen Angreifer aus dem Netz bekannte Fehler in Serversoftware aus. Dazu braucht es oft nur wenige bis keine besonderen Fähigkeiten. Wer sucht, findet im Internet Angriffswerkzeuge, die selbsttätig bekannte Fehler ausnutzen. Oft muß man solchen Angriffswerkzeugen nur sein Ziel sagen. Es soll schon vorgekommen sein, dass ein Administrator eines Unixsystems bei der Aufarbeitung eines Angriffs in den Logfiles die hilflosen Versuche eines - offenbar nur auf Windowssystemen versierten - Einbrechers vorgefunden hat, mit dem Befehl `dir` wenigstens ein sinnvolles Kommando abzugeben.
Weiters gibt es das Phänomen von Internetwürmern. Das sind autark handelnde Programme, die versuchen mit einprogrammierten Angriffen Computer zu knacken, sich auf diesen einzunisten und von dort aus weiterzuverbreiten.
Gegen derartige Angriffe gibt es nur ein Mittel: Das regelmäßige Updaten des Systems. Dies passiert am besten per täglichem Cronjob. Die häufig geäusserte Regel „Never change a running system“ ist hierbei ein unangebrachter Zugang. Auch Firewalls können einem Administrator das Update nicht ersparen.

Updates

- Distributoren beheben sicherheitskritische Fehler
- Schutz vor Angriffsautomaten und Script-Kiddies
- Am besten täglich und automatisch
- „Never change a running system“?

Folie 240

18.3 Sniffer

Das beste Passwort ist vollkommen wertlos, wenn man es im Klartext (also unverschlüsselt) über das Netz verschickt. Leider gibt es viele Programme bzw. Protokolle, die noch immer Passwörter im Klartext verschicken und trotzdem verwendet werden. Einige prominente Vertreter sind

- Telnet
- FTP (authentifiziert)
- POP
- HTTP (Logins auf Webseiten)
- SMB (Windows Freigaben)

Der letzte Punkt ist nur bedingt richtig, da in den neueren Versionen von Windows und Samba eine verschlüsselte Authentifizierung stattfindet kann. Für alle anderen Programme gibt es entweder ein Ersatzprogramm mit Verschlüsselung (Telnet $\leftrightarrow$ Ssh, FTP $\leftrightarrow$ SFTP, HTTP $\leftrightarrow$ HTTPS) oder die Möglichkeit die Programme über sogenannte Ssh-Tunnel zu betreiben (POP).
Der Inhalt jedes Datenpaketes kann zumindest auf jedem Knoten (Router/Gateway) über den es läuft gelesen werden. Das können sehr viele Stationen sein (siehe Folie 241).
Im Beispielfall gehen also beim Klicken des „Senden und Empfangen“-Buttons des Mailprogramms der Username, das Passwort und der Inhalt aller übertragenen Mails im Klartext über 16 Computer.
Doch das sind bei weitem nicht die einzigen Computer, die mithören können. Ist ein Netzwerk per BNC-Kabel oder mit einem Hub zusammengeschlossen, so kann jeder Computer im Netzwerk den Datenverkehr aller anderen abhören. Und selbst in Netzwerken, die über einen Switch verbunden sind, kann jeder Computer im Netzwerk mit ein wenig mehr Aufwand den gesamten Verkehr abhören. (ARP spoofing, ARP table overflow am Switch, etc.)
Oft installiert ein Cracker, der sich Zugang zu einem Computer eines Netzes verschafft hat auf diesem einen Sniffer, der Benutzername/Passwort-Paare in eine Datei mitloggt. Diese Datei kann er entweder bei Zeiten abholen oder sich automatisch,

anonym und verschlüsselt in irgend ein öffentliches Forum im Internet posten. Mit dieser Methode kann man in kürzester Zeit viele Logins von vielen Computern - auch außerhalb des kompromitierten Netzwerks - beschaffen. Und wenn ein Angreifer einen Account eines Systems kennt, dann ist es für ihn schon sehr viel leichter. Hätte in diesem Fall ein Administrator auf den root-Account über ein unverschlüsseltes Protokol zugegriffen, so wäre der Angriff schon perfekt. Deshalb sollte ein Administrator niemals unverschlüsselt über das Netz auf den root-Account zugreifen. Ein `su -` innerhalb einer Telnetsession ist dabei natürlich um nichts besser.
Wir werden uns nun am einem Beispiel ansehen, wie einfach es ist, Klartextpasswörter mitzulesen. Das Programm, das wir dazu verwenden heißt `ngrep`. Wir lesen damit den Benutzernamen und das Passwort aus einer authentifizierten FTP-Session aus (siehe Folie 242).

traceroute www.gmx.de

```
1  center2-sued-wien.chello.at (10.34.11.2)  17.078 ms  19.695..
2  213.47.218.249 (213.47.218.249)  8.873 ms  38.098 ms  3.737..
3  at-vie-rd-03-ge-2-1.chellonetwork.com (213.46.173.21)  4.13..
4  at-vie-rc-01-ge-2-1.chellonetwork.com (213.46.160.129)  26...
5  * 213.46.160.113 (213.46.160.113)  27.846 ms  8.483 ms
6  uk-lon-rc-01-pos-5-0.chellonetwork.com (213.46.160.117)  62..
7  213.46.160.137 (213.46.160.137)  86.378 ms  83.870 ms  67.9..
8  de-fra-rc-02-pos-30.chellonetwork.com (213.46.160.90)  78.4..
9  de-fra-rc-01-pos-4-0.chellonetwork.com (213.46.160.85)  60...
...
14  atm-10102.gw-backbone-a.muc.schlund.net (212.227.112.118) ..
15  to-gmx.schlund.net (212.227.112.42)  139.813 ms  87.667 ms..
16  www.gmx.net (213.165.65.100)  86.669 ms  97.269 ms *
```

Folie 241

ngrep -qd lo port 21

```
T 192.168.0.10:21 -> 192.168.0.10:41833 [AP]
220 ProFTPD 1.2.4 Server (Debian) [carla.doppler]..

T 192.168.0.10:41833 -> 192.168.0.10:21 [AP]
USER ferdinand..

T 192.168.0.10:21 -> 192.168.0.10:41833 [AP]
331 Password required for ferdinand...

T 192.168.0.10:41833 -> 192.168.0.10:21 [AP]
PASS w8h7RLNZ..

T 192.168.0.10:21 -> 192.168.0.10:41833 [AP]
230 User ferdinand logged in...
```

Folie 242

18.4 Portscanner

Ein wichtiger Punkt für die Sicherheit eines Systems ist es, keine nicht benötigten - und damit meist nicht administrierten - Dienste laufen zu lassen. Eine Methode das festzustellen, ist direkt am untersuchten Computer mit `netstat` oder `lsof` von Servern belegte Ports aufzulisten (siehe Folie 243).
Das heißt aber nicht unbedingt, dass ein anderer Rechner die gleichen Services sieht. Einerseits können Ports für andere Computer mit dem TCP-Wrapper oder einem Paketfilter unsichtbar gemacht werden, andererseits könnte - im schlimmsten Fall - der Computer bereits gecrackt sein und das `netstat`-Kommando verändert worden sein.
Daher sollte man offene Ports auch mit einem Portscanner von einem anderen Computer aus überprüfen. Ein Portscanner versucht im wesentlichen nichts anderes, als sich auf allen möglichen Ports mit seinem Ziel zu verbinden und meldet dann, auf welchen Ports dies möglich war. Oftmals hinterlegt ein erfolgreicher Angreifer ein Backdoor, um zu einem späteren Zeitpunkt wieder leicht in das System einsteigen zu können. Oft sind Backdoors ganz einfache Server, die auf einem hohen Port auf eine Verbindung warten und gegebenenfalls eine root-Shell liefern. Auch zum finden solcher Backdoors eignen sich Portscanner.
WARNUNG! Portscans sollte man nur auf die Rechnern loslassen, die man selbst administriert, um Sicherheitslücken aufzuspüren. Nur böse Buben scannen fremde Computer, um Angriffe vorzubereiten. Ein Portscan ist auch sehr leicht am angegriffenen Rechner nachzuweisen und unter normalen Bedingungen lässt sich auch der Angreifer feststellen.
Wir schauen uns nun einen der verbreitetsten Portscanner, nämlich `nmap`, an. Wir werden nur ein elementares Beispiel sehen. Die lesenswerte Manpage von `nmap` lässt aber schon ahnen, dass das Spiel hier noch lange nicht am Ende ist (siehe Folie 244).
Ein weiteres sehr interessantes Werkzeug zur Untersuchung von Computern und deren Services ist `nessus`. Es versucht nicht nur eine Verbindung zu allen Ports zu öffnen, sondern auch festzustellen, welcher Dienst hinter diesem Port zu finden ist. Am Ende der Prozedur wird ein Bericht erstellt, der auf Sicherheitsmängel hinweist (siehe Folie 245). Zu Testzwecken haben wir in diesem Fall einen Ssh-Server anstatt wie üblich auf Port

22, auf Port 9876 laufen lassen. An diesem Beispiel erkennt man auch leicht, dass das „Verstecken“eines Services auf einem untypischen Port keinerlei Sicherheit bringt.

netstat -l

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address     Foreign Address  State
tcp        0      0 *:pop3           *:*              LISTEN
tcp        0      0 *:auth           *:*              LISTEN
tcp        0      0 zulu:domain      *:*              LISTEN
tcp        0      0 localhost:domain *:*              LISTEN
tcp        0      0 *:ssh            *:*              LISTEN
tcp        0      0 *:ipp            *:*              LISTEN
tcp        0      0 *:smtp           *:*              LISTEN
udp        0      0 *:1037           *:*
udp        0      0 *:10000          *:*
udp        0      0 zulu:domain      *:*
udp        0      0 localhost:domain *:*
```

Folie 243

nmap 192.168.0.13

```
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on zulu.doppler (192.168.0.13):
(The 1541 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     open        smtp
53/tcp     open        domain
110/tcp    open        pop-3
113/tcp    open        auth
631/tcp    open        cups
10000/tcp  open        snet-sensor-mgmt

Nmap run completed -- 1 IP address (1 host up) scanned in 1 secon
```

Folie 244

nessus

Problems regarding : unknown (9876/tcp)
Security holes :

```
You are running a version of OpenSSH which is older than 3.0.2.

Versions prior than 3.0.2 are vulnerable to an enviroment
variables export that can allow a local user to execute
command with root privileges.
This problem affect only versions prior than 3.0.2, and when
the UseLogin feature is enabled (usually disabled by default)

Solution : Upgrade to OpenSSH 3.0.2 or apply the patch for prior
...
```

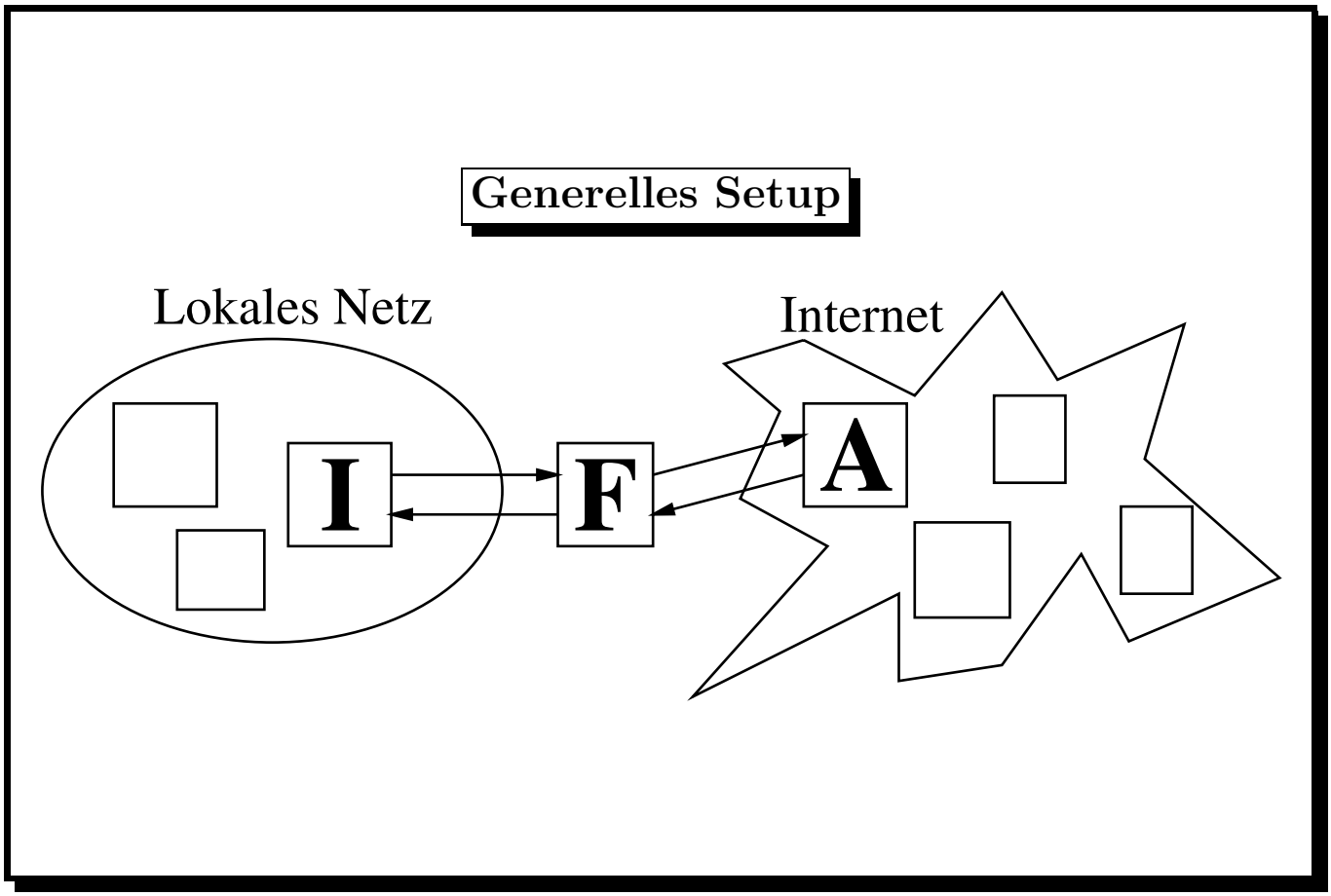
Folie 245

18.5 Firewalls

Eine Firewall ist ein Mechanismus zum Schutz eines oder mehrerer Computer vor unerwünschten Zu- und Angriffen aus dem Netz. Es gibt zwei Grundtypen von Firewalls:

- Proxy Server
- Paketfilter

Wie auch beim Begriff „Server“ wird der Begriff „Firewall“ sowohl für Hardware als auch für Software verwendet. Oft ist eine Firewall ein eigener Computer, der einerseits im internen Netz und andererseits im Internet hängt. Die Firewall soll vordefinierte Verbindungen zwischen Computern innerhalb und außerhalb der Firewall ermöglichen, unerwünschte Zugriffe aber unterbinden. In Weiteren wird **I** einen Computer innerhalb, **A** einen Computer außerhalb und **F** die Firewall selbst bezeichnen.



Folie 246

18.5.1 Proxy Server

Bevor wir uns den Paketfiltern zuwenden, soll hier kurz auf die Funktionsweise und die Arten von Proxy Servern eingegangen werden. Es ist also jetzt **F** ein Proxy Server. Ist ein Netz via Proxy Server an das Internet angebunden, so besteht für einen Computer **I** keine Möglichkeit eine direkte Verbindung zu einem Computer **A** aufzubauen. Anstelle einer solchen Verbindung treten zwei Verbindungen, einerseits von **I** nach **F** und andererseits von **F** nach **A**. Die Software am Rechner **I** muß für die Verwendung des Proxy Servers konfiguriert sein und stellt damit Anfragen nie an **A** selbst, sondern an **F**. **F** verbindet sich mit **A** und liefert das Resultat an **I** zurück.
Die Sicherheitsmechanismen in Proxy Servern erlauben es, Verbindungen auf einige Dienste einzuschränken (zB Web, SSH,...), eine Benutzerauthentifizierung durchzuführen, sämtlichen Datenverkehr mitzuloggen und vieles mehr.
Proxy Server sind in zwei Typen unterteilt:

- Application Proxy
- SOCKS Proxy

Application Proxies (zB *squid*) müssen jedes Application Protocol, dass sie unterstützen verstehen. So kann *squid* nur Web- und FTP-Traffic verarbeiten. Eine Secure Shell-Verbindung kann man über *squid* nicht weiterleiten. Der Vorteil solcher Proxies ist, dass sie sich auch zum Cachen oder zum Filtern von Inhalten eignen.
SOCKS Proxies (zB *dante*) arbeiten auf der Schicht von TCP und UDP. Diese verstehen also die Protokolle des Application Layers nicht und können damit auch nicht zu Caching Zwecken verwendet werden. Der Vorteil ist, dass (fast) beliebige TCP- und UDP-Anwendungen verarbeitet werden können.

18.5.2 Paketfilter

Ein völlig anderer Zugang ist es, auf dem Computer **F** einen Paketfilter zu verwenden. Hierbei werden Verbindungen von **A** nach **I** - wie wenn **F** ein gewöhnlicher Router wäre - direkt durchgeführt. Das hat auch den Vorteil, dass keinerlei Konfiguration des Rechners **A** vonnöten ist. Der Paketfilter auf **F** liest

den Datenverkehr auf verschiedenen Protokollschichten mit, und entscheidet ob das Datenpaket weitergeleitet oder verworfen wird. Im Folgenden eine unvollständige Auflistung von Eigenschaften von Paketen verschiedener Schichten, welche Paketfilter kennen (siehe Folie 247).
Seit Version 2.0 ist im Linux Kernel ein Paketfilter eingebaut. Der diesbezügliche Code wurde seither in jedem neuen Majorrelease geändert. Dementsprechend unterscheiden sich auch die Commandlinetools der verschiedenen Versionen.

Kernel	Komando
2.0	`ipfwadm`
2.2	`ipchains`
2.4	`iptables`

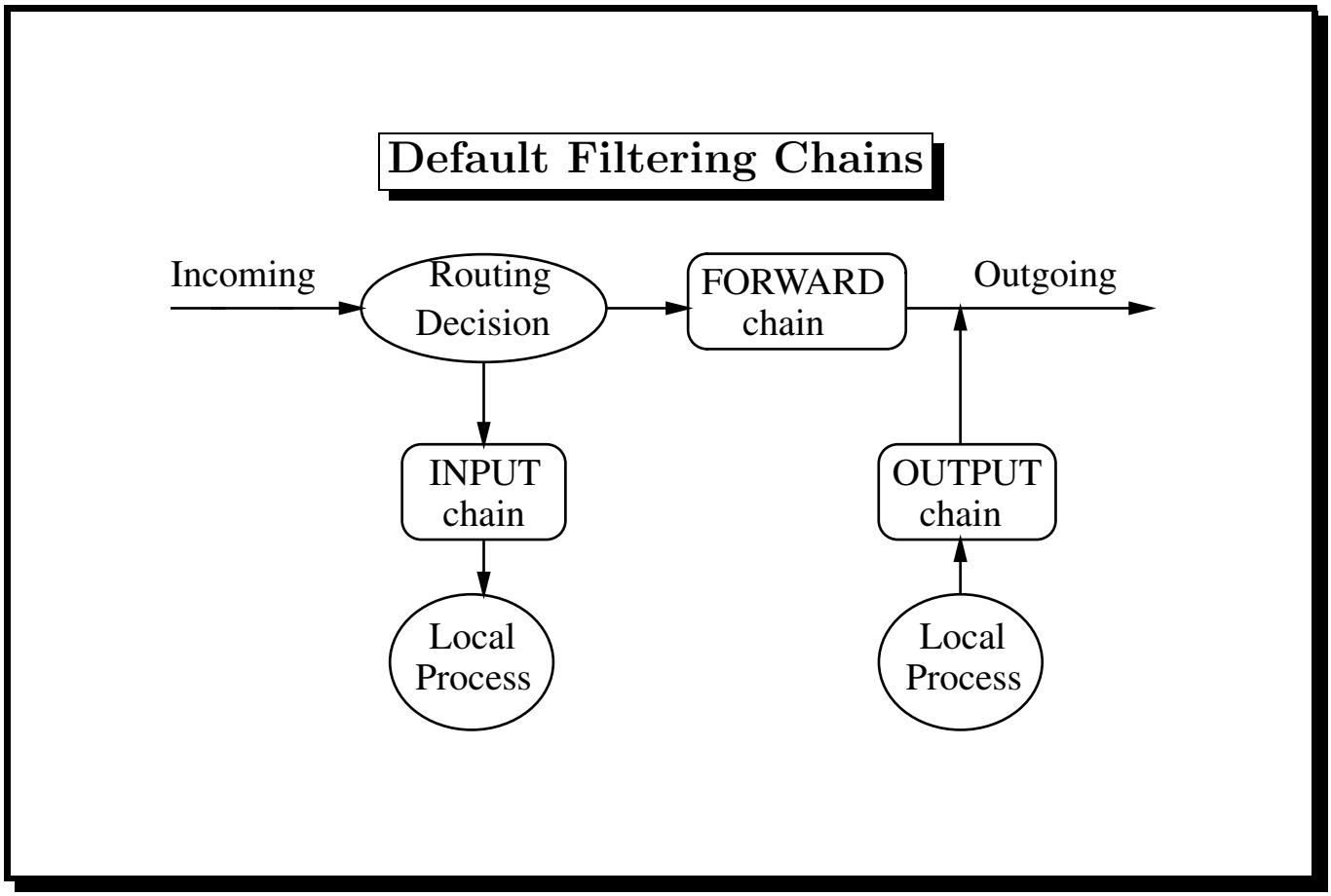
Wir wenden uns nun der Verwendung der iptables zu. Ein Paket durchläuft verschiedene Chains (Ketten) von Regeln (Rules). Drei solcher Chains sind immer vorhanden: INPUT, OUTPUT und FORWARD Chain. Wir betrachten im folgenden die Chains unserer Firewall **F**. Ein Paket, das vom Rechner **I** oder **A** kommt und zum Rechner **F** will, durchläuft die INPUT Chain. Ein Paket, dass am Weg von **I** nach **A** die Firewall **F** trifft, durchläuft die FORWARD Chain. Ein Paket, dass von **F** nach **I** oder **A** geht, durchläuft die OUTPUT Chains. Die Situation auf der Firewall **F** wird am besten durch folgendes Diagramm beschrieben, welches für die Kernelserie 2.4 gilt (siehe Folie 248).
Wir betrachten nun den Werdegang eines Paketes, wenn es eine Chain durchläuft. Der erste Befehl, den wir kennenlernen, listet alle Regeln der INPUT Chain auf (siehe Folie 249).
Dies zeigt uns nun eine Liste aller Regeln, die die INPUT Chain bilden. Ein Paket, das versucht diese Chain zu durchlaufen wird der Reihe nach mit diesen Regeln verglichen. Entspricht ein Paket in allen Punkten einer Regel, so wird das zu dieser Regel gehörige Target auf das Paket angewendet. Targets können einerseits selbstdefinierte Chains sein oder aber eingebaute Targets, von welchen wir im folgenden einige anführen (siehe Folie 250).
In obigem Listing ist das immer ACCEPT. Das Paket darf also passieren, wenn eine der Regeln zutrifft. Wichtig ist zu bemerken, dass, sobald eine Regel zutrifft und ein Target auf das Paket angewendet wird, keine weitere Regel in der Chain mit

dem Paket verglichen wird. Trifft keine der Regeln einer Chain auf das Paket zu, so wird die Policy auf dieses angewendet. In unserem Fall wird also jedes Paket, auf das keine Regel zutrifft, abgelehnt. Einen etwas wortreicheren Output liefert folgendes Kommando (siehe Folie 251).
Hier sehen wir nun auch Zähler für die Anzahl bzw. die Datenmenge der Pakete, auf die die entsprechende Regel zugetroffen hat. Diese Zähler sind einerseits praktisch zum Messen des Datenverkehrs, andererseits lässt sich anhand dieser Zähler oft auch herausfinden, wo - und damit warum - ein Paket nicht gewünschten Weg ging.
Zum Einfügen einer neuen Regel ans Ende einer Chain verwendet man den Befehl
`# iptables -A` *Chain* `....`
Da die Reihenfolge der Regeln relevant ist, gibt es auch die Möglichkeit mit
`# iptables -I` *Chain* `[`*number*`] ...`,
eine Regel in eine bestimmte Zeile einzufügen. Die oben beschriebene Policy einer Chain wird mit dem Befehl
`# iptables -P` *Chain target*
gesetzt (siehe Folie 252).
Wir listen nun einige der Optionen von `iptables` auf, insbesondere einige, welche die Paketeigenschaften in der Tabelle auf Folie 247 betreffen (siehe Folie 253).
Besonders hervorheben möchten wir die `--state` Option. Mit dieser Option kann man Pakete behandeln, die zu einer existierenden Verbindung gehören. Diesen Umstand nutzt auch „Rusty's Really Quick Guide to Packet Filtering“, den wir hier in leicht modifizierter Form wiedergeben möchten (siehe Folie 254). Rusty Russel ist der Linux IP Firewall Maintainer.
Unter der Voraussetzung, dass `eth0` (also die erste Ethernetkarte im Computer) mit dem Internet verbunden ist, lässt diese Firewall jede Verbindung vom lokalen Netz ins Internet zu, verbietet aber im Gegenzug die umgekehrte Richtung vollständig.

Eigenschaften von Datenpaketen

Layer	Protocol	Eigenschaft
Network Access	Ethernet	MAC Adresse
Internet	IP	Source IP, Destination IP
		Incoming-/Outgoing-Interface
	ICMP	ICMP Type
Transport	TCP	Port, Flags (zB SYN),
		Connection Tracking
	UDP	Port, Connection Tracking
Application	FTP	Zugeordnete Datenverbindung

Folie 247


Default Filtering Chains
Incoming
Routing Decision
FORWARD chain
Outgoing
INPUT chain
OUTPUT chain
Local Process
Local Process

Folie 248

Listing einer Chain

```
[root@milch root]# iptables -L INPUT
Chain INPUT (policy DROP)
target     prot opt source     destination
ACCEPT     all  --  anywhere   anywhere
ACCEPT     all  --  anywhere   anywhere         state RELATED,ESTABLISHED
ACCEPT     all  --  phobos     milch
ACCEPT     tcp  --  anywhere   milchtcp dpt:ssh
```

Folie 249

Targets

Target	Beschreibung
ACCEPT	Das Paket wird akzeptiert.
DROP	Das Paket wird verworfen.
REJECT	Das Paket wird verworfen, dem Sender wird ein ICMP 'port unreachable' Paket zurückgesandt.
LOG	Ein Eintrag, der das Paket beschreibt, wird in ein Logfile geschrieben.
MASQUERADE	siehe 18.5.4

Folie 250

Langes Listing einer Chain

```
[root@milch root]# iptables -v -L INPUT
Chain INPUT (policy DROP 8180 packets, 2125K bytes)
 pkts bytes target  prot opt in   out  source    destination
 3662  468K ACCEPT  all  --  lo   any  anywhere  anywhere
 130K   67M ACCEPT  all  --  any  any  anywhere  anywhere   state REL...
    0     0 ACCEPT  all  --  any  any  phobos    milch
    1    60 ACCEPT  tcp  --  any  any  anywhere  milchtcp dpt:ssh
```

Folie 251

Neue Regel Einfügen

```
[root@milch root]# iptables -A INPUT -p tcp -s 131.130.16.0/24 --sport
     1024:65535 -d milch --dport finger -i eth0 -j ACCEPT
[root@milch root]# iptables -v -L INPUT
Chain INPUT (policy DROP 8180 packets, 2125K bytes)
.. target  prot opt in   out  source    destination
.. ACCEPT  all  --  lo   any  anywhere  anywhere
.. ACCEPT  all  --  any  any  anywhere  anywhere   state REL...
.. ACCEPT  all  --  any  any  phobos    milch
.. ACCEPT  tcp  --  any  any  anywhere  milchtcp dpt:ssh
.. ACCEPT  tcp  --  eth0 any  131.130.16.0/24
     milchtcp spts:1024:65535 dpt:finger
```

Folie 252

`iptables Optionen (Auszug)`

`--mac-source` *MAC*	MAC Quelladresse
`-s` *IP*	IP Quelladresse
`-d` *IP*	IP Zieladresse
`-i` *Interface*	Eingehendes Interface
`-o` *Interface*	Ausgehendes Interface
`-p` *Protocol*	`tcp, udp, icmp` oder `all`
`--icmp-type` *Typname*	ICMP Typ
`--source-port` *Port*	TCP- bzw. UDP-Quellports
`--destination-port` *Port*	TCP- bzw. UDP-Zielports
`--syn`	TCP-Verbindungaufbau
`--state`	Status einer Verbindung

Folie 253

Rusty's Really Quick Guide to Packet Filtering

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m state --state NEW -i ! eth0 -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEP
iptables -A FORWARD -m state --state NEW -i ! eth0 -j ACCEPT
```

Folie 254

18.5.3 Paketfilter stoppen und starten

Nachdem der Aufruf des Kommandos `iptables` direkt in die Chains im Kernel schreibt, gehen diese bei einem Reboot verloren. Um die Chains bei einem Reboot wiederherzustellen, sollte man die `iptables` Kommandos in ein Startscript geschrieben werden. Wir werden hier den Standardweg von Redhat durchgehen.
Hierbei schreibt man ein Shellscript, das die Firewall in den gewünschten Zustand bringt und speichert es in `/etc/sysconfig/iptables.sh`. Am Ende des Scripts sollte die Zeile

```
iptables-save > /etc/sysconfig/iptables
```

stehen. Beim Ausführen des Scripts bewirkt diese Zeile, dass die aktuellen Regeln in die Datei `/etc/sysconfig/iptables` gespeichert werden, von wo sie das Script `/etc/init.d/iptables` beim Booten einliest.

Redhat iptables Startscripts

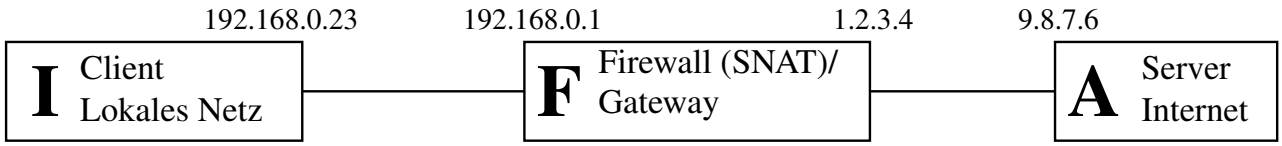
```
# vim /etc/sysconfig/iptables.sh
# chmod 755 /etc/sysconfig/iptables.sh
# /etc/sysconfig/iptables.sh
# chkconfig --add iptables
# /etc/init.d/iptables restart
Flushing all current rules and user defined chains:   [  OK  ]
Clearing all current rules and user defined chains:   [  OK  ]
Applying iptables firewall rules:                     [  OK  ]
```

Folie 255

18.5.4 Masquerading (Source NAT)

Oft ist man vor das Problem gestellt, mehreren Computern einen Internetzugang zu ermöglichen, obwohl man nur eine offizielle IP Adresse zur Verfügung hat (zB via ein Modem). Das lässt sich erreichen, indem man in unserem Standardbeispiel am Rechner **F** einen Proxy Server oder aber IP-Masquerading bzw SNAT (Source Network Address Translation) betreibt. **F** wird dann oft als masquerading Gateway bezeichnet.
Dabei werden bei allen IP-Paketen, die von einem internen Computer **I** über den Gateway **F** ins Internet gehen, die IP-Quelladressen auf die im Internet gültige IP-Adresse des Gateways umgeschrieben, sodass der Server **A** glaubt, mit **F** zu kommunizieren. Bei den Antwortpaketen von **A** schreibt der Gateway **F** die IP-Zieladressen wieder auf die des Computers **I** um, sodass dieser die Antwort erhält. Für den Computer **I** ist das Verfahren transparent, das heißt er glaubt direkt mit **A** zu sprechen (siehe Folie 256).
Hier hat Rusty Russel sich selbst übertroffen und die Minimalkonfiguration des Paketfilters auf der Firewall **F** extrem kurz beschrieben (siehe Folie 257). Es wird davon ausgegangen, dass die Firewall über ein Modem (`ppp0`) ans Internet angebunden ist. Diese Minimalkonfiguration beinhaltet keinerlei Filterregeln zum Verbot bestimmter Verbindungen. Um die Firewall gegen äußere Rechner **A** abzuschotten, sollte man dieses Kochrezept mit dem der Folie 254 verbinden. Die Computer innerhalb der Firewall genießen bei dieser Technik automatisch einen recht hohen Schutz, da sie keine im Internet gültigen IP-Adressen haben und somit aus dem Internet nicht erreichbar sind.
Durch laden des Kernelmoduls `iptable_nat` stehen drei weitere Chains zur Verfügung: PREROUTING, POSTROUTING und OUTPUT. Diese lassen sich mit dem `iptables`-Kommando unter Angabe der Option `-t nat` bedienen. Eine Auflistung dieser drei Chains erhält man also mit
`# iptables -t nat -L`.
Für Masquerading ist ausschließlich die POSTROUTING Chain von Belang.

Masquerading



IP-Header: Paket von I nach A	S: 192.168.0.23 D: 9.8.7.6	S: 1.2.3.4 D: 9.8.7.6
IP-Header: Paket von A nach I	S: 9.8.7.6 D: 192.168.0.23	S: 9.8.7.6 D: 1.2.3.4

Folie 256

Rusty's Three Line Guide to Masquerading

```
modprobe iptable_nat
iptables -t nat -A POSTROUTING -o ppp0 -j MASQERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Folie 257

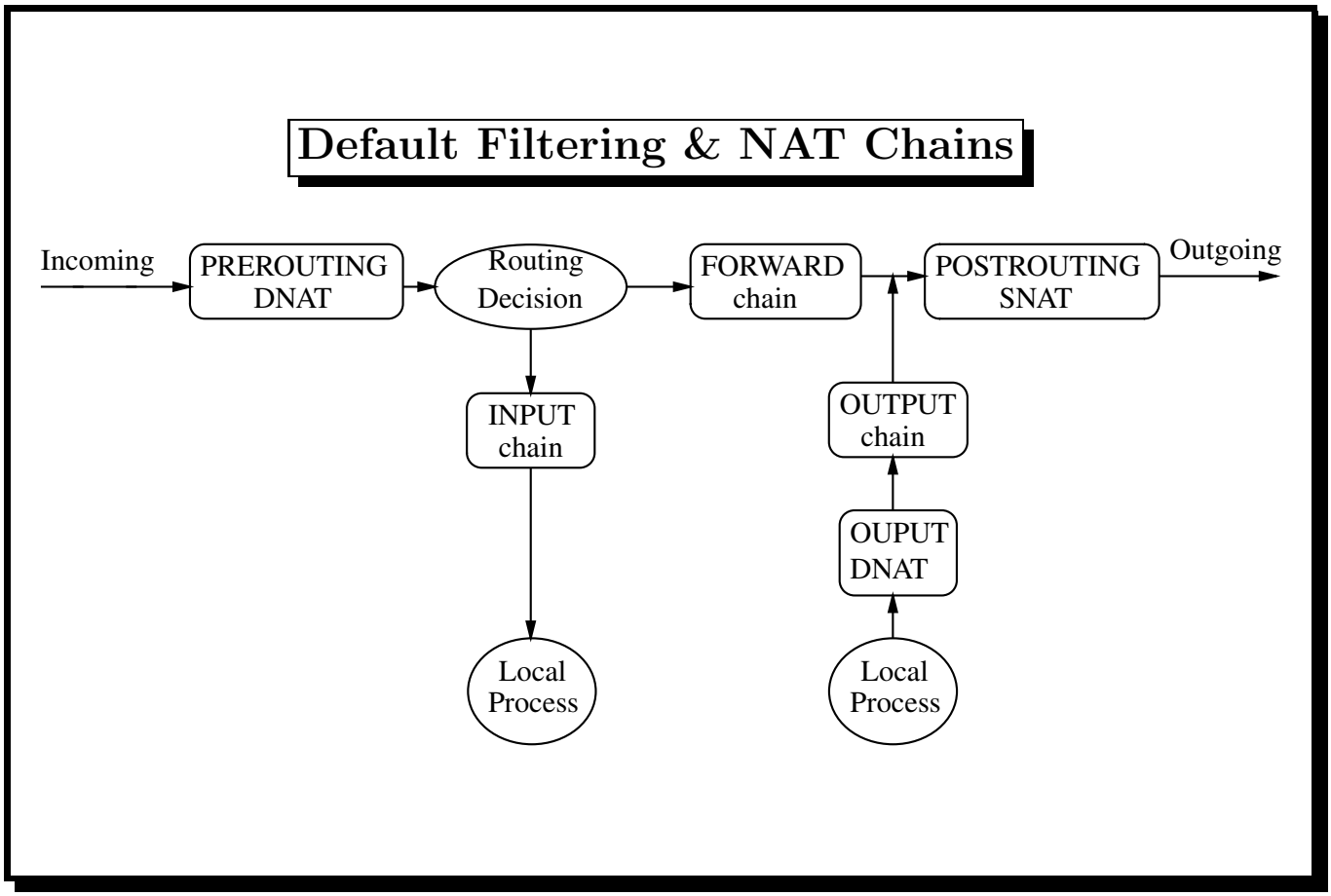
18.5.5 Portforwarding (Destination NAT)

Wie zuvor besprochen kann ein Computer **I** im lokalen Netz von aussen nicht erreicht werden. Was aber, wenn man zB einen Webserver betreiben will. Man könnte diesen auf der Firewall **F** betreiben. Das wird meist aber nicht gemacht, da es die Sicherheit der Firewall unter Umständen kompromitiert. Meist geht man folgenden Weg: Auf der Firewall laufen keinerlei Dienste ausser Ssh zu administrativen Zwecken. Der Webserver läuft am Rechner `I` im masquierten Netz. Kommt nun eine Anfrage an die Firewall **F** auf Port 80, so wird diese zum Computer `I` weitergeleitet. Die Antwort von `I` leitet **F** zurück an `A`. Dieses Verfahren nennt man Portforwarding oder DNAT (Destination Network Address Translation). Der anfragende Computer `A` hat dabei den Eindruck mit der Firewall `F` zu kommunizieren.
Wir werden uns nun die Regel für den beschriebenen Fall anschauen. Portforwarding findet in der PREROUTING Chain statt (siehe Folie 258).
Zum Abschluß eine Übersicht über die Filter- und NAT-Chains und wie sie durchlaufen werden (siehe Folie 259).

Webserver Portforwarding

```
modprobe iptable_nat
iptables -t nat -A PREROUTING -p tcp --dport www -j DNAT
   --to-destination I
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Folie 258


Default Filtering & NAT Chains
Incoming
PREROUTING
DNAT
Routing
Decision
FORWARD
chain
POSTROUTING
SNAT
Outgoing
INPUT
chain
OUTPUT
chain
OUPUT
DNAT
Local
Process
Local
Process

Folie 259

18.5.6 Wovor schützt eine Firewall?

Nachdem wir nunmehr einen technischen Überblick über die Arten und Vorgehensweisen von Firewalls gegeben haben, können wir die Schutzwirkung abschätzen. Grundsätzlich ist anzumerken, dass eine Firewall in keiner Weise ein angebotenes Service schützt. Wird ein Webserver betrieben, so muß er auch erreichbar und damit von der Firewall uneingeschränkt sein. Der Webserver selbst (das Programm auf Port 80) ist in diesem Fall mit genauso angreifbar wie ohne Firewall. Auch vor Viren, die Personen im lokalen Netz auf ihre Computer per Email oder Web herunterladen, bringt eine Firewall keinerlei Schutz.
Eine Firewall kann aber ein Service, das nur im lokalen Netz angeboten werden soll, vor Zugriffen von aussen schützen. Auch kann damit verhindert werden, dass Computer im lokalen Netz unerwünschter Weise Services (oder auch Backdoors) nach aussen anbieten. Und man kann damit auch Teile des Internets für die Benutzer im lokalen Netz sperren.
Der Aufwand des Administrator steigt aber exponentiell mit der Komplexität der Firewall. Starke Einschränkungen führen häufig auch zu unerwünschten Behinderungen. Wenn keine besonderen Anforderungen an die Firewall gestellt werden, sollte man das Regelwerk so schmal wie möglich halten.

Wovor schützt eine Firewall?

JA

- Unerwünschte Server im lokalen Netz
- Zugriff auf Services für den internen Bereich
- Unerwünschte Zugriffe von Benutzern auf Internetdienste
- ...

NEIN

- Angriffe auf angebotene Services
- Viren per Email oder Web
- ...

Folie 260

19 Samba

Das Samba Paket bietet eine Implementation des SMB (Server Message Block) Protokolls für Unix und andere Plattformen, die so in eine MS-Windows Netzwerkumgebung eingebunden werden können. Wir besprechen hier detailliert sowohl das SMB-Protokoll, als auch die Konfiguration von Samba unter Linux.

Im folgenden wird der Begriff Windows NT (oder kurz NT) synonym für die Familie der aus Windows NT abgeleiteten Betriebsysteme verwendet und steht somit für Windows NT, Windows 2000 und Windows XP.

Was ist Samba?
Samba integriert einen Unix Rechner in die Windows Netzwerkumgebung (durch Implementation des SMB (Server Message Block) Protokolls). Genauer gesagt lässt Samba jeden Unixrechner in der Netzwerkumgebung von Windows erscheinen und erlaubt Datei- und Druckfreigaben zu erstellen. Das heißt, unter Unix vorhandener Plattenplatz kann ganz normal unter Windows genutzt werden, und unter Unix vorhandene Drucker kann man als Netzwerkdrucker unter Windows ansteuern. Darüber hinaus bietet Samba viele Dienste, die sonst nur von Windows NT geleistet werden. Dazu gehören:

- **WINS Server** (Windows Namensauflösung)
- **Computersuchdienst**
- **Logon Server** für Windows 95/98
- **PDC** (NT Primary Domain Controller)

Um Samba zu verstehen, müssen wir zunächste einige Netzwerkfunktionalitäten und Netzwerkprotokolle unter Windows erkären.

Was ist Samba?

- Der Linux-Rechner „erscheint“ in der Netzwerkumgebung der Windows Maschinen
- Linux kann als Fileserver bequemen Zugriff von Windows aus zulassen
- Der/die Drucker des Linux Rechners können von Windows aus benutzt werden
- Verhält sich wie ein echter Windows NT Server

Folie 261

19.1 NetBIOS

Sobald Windowsrechner Dateisysteme austauschen, sich gegenseitig in der Netzwerkumgebung sehen oder Drucker freigeben, funktioniert die Kommunikation über NetBIOS. NetBIOS ist eine reine Softwareschnittstelle zur Kommunikation von Rechnern, die Programmen unterschiedliche Dienste zur Kommunikation zur Verfügung stellt.
NetBIOS wurde entworfen, um in kleinen, lokalen Netzen Kommunikation zu ermöglichen. Dabei lag der Schwerpunkt des Entwurfs auf der Einfachheit der Anwendung. Auf Skalierbarkeit und die Andwendung in Weitverkehrsnetzen wurde beim Design nicht geachtet. Die Kommunikation mit NetBIOS wurde in drei Teilbereiche aufgeteilt, in den Namensdienst, den Datagramm- und den Sitzungsdienst.

- **Namensdienst** Im Rahmen des Namensdienstes sind die Rechner in der Lage, sich gegenseitig im Netz zu identifizieren. Es sei an dieser Stelle betont, dass der NetBIOS-Namensdienst nichts mit der Anzeige in der Netzwerkumgebung zu tun hat. Der Computersuchdienst, der für die Netzwerkumgebung zuständig ist, hängt jedoch sehr stark von einem korrekt funktionierenden Namensdienst ab.

- **Datagrammdienst** Betrachtet man die Rechnerkommunikation auf dem Netz, so sieht man, dass die versendeten Daten in einzelne Pakete aufgeteilt werden. Diese einzelnen Pakete werden dann vom Netz nach bestem Bemühen an einen Zielrechner ausgeliefert. Geht ein Paket verloren, kann man nichts machen, man bekommt unter Umständen nicht einmal eine Benachrichtigung darüber, dass etwas nicht stimmt. Aufeinander folgende Pakete können in vertauschter Reihenfolge beim Empfänger ankommen. Es kann sogar sein, dass Pakete auf dem Weg dupliziert werden, also mehrfach ankommen. Diese Unzuverlässigkeit des Netzes wird durch den Datagrammdienst an die Benutzerprogramme weitergegeben. Der Datagrammdienst hat jedoch nicht nur Nachteile. Zwei Vorteile sind der geringe Aufwand, mit dem Pakete verschickt werden können, und die Möglichkeit, ein Datagramm an mehrere Rechner gleichzeitig zu verschicken. Die Anwendung muss selbst entscheiden, wie sie mit der Unzuverlässigkeit des Dienstes klarkommt.

- **Sitzungsdienst** Die Unzuverlässigkeit des Netzes ist für bestimmte Applikationen wie Dateitransfer oder Terminalanwendungen nicht akzeptabel. Wenn man eine Datei überträgt, möchte man sicher sein, dass die Datei komplett und korrekt übertragen wurde. Für diese höheren Anforderungen wurde der Sitzungsdienst entworfen. Zwei Rechner vereinbaren eine NetBIOS-Sitzung. Die Daten, die über diese Verbindung übertragen werden, kommen auf jeden Fall an, und zwar in der richtigen Reihenfolge. Können Daten einmal nicht übertragen werden, so erhält die versendende Applikation eine Fehlermeldung. Die Applikation kann nun versuchen, die abgebrochene Sitzung neu aufzubauen. Dieser Zuverlässigkeit steht ein erhöhter Aufwand beim Sitzungsauf- und -abbau gegenüber.

Zwei Rechner, die kommunizieren wollen, müssen sich zunächst gegenseitig identifizieren. NetBIOS sieht hierfür bis zu 16 Zeichen lange Namen vor. Jede Applikation kann für sich beliebig viele Namen reservieren und unter einem dieser Namen Verbindungen aufbauen und Daten austauschen. Diese Reservierung von Namen gilt sowohl für Server, die vom Netz aus erreichbar sein müssen, als auch für Clients, die Server im Netz erreichen wollen, da Server wissen müssen, wohin die Antworten gehen müssen.
Zwei Anwendungen wollen nun per NetBIOS miteinander kommunizieren. Dazu muss zunächst der Server seine Bereitschaft kundtun, Verbindungen entgegenzunehmen. Dazu meldet er sich im Netz mit seinem Namen an. Diese Anmeldung geschieht per Broadcast, so dass alle im Netz mithören können. Jeder Rechner ist frei, beliebige Namen im Netz für sich zu beanspruchen, sofern diese noch nicht belegt sind. Eine Reservierung geschieht also, indem ein Rechner per Broadcast ankündigt, dass er unter einem bestimmten Namen erreichbar ist. Dann wartet er auf Protest. Beklagt sich niemand, schickt er eine zweite Reservierung und wartet wieder. Nach der dritten Reservierung ist der Rechner ausreichend sicher, dass kein anderer den Namen bereits für sich eingenommen hat, und sieht ihn als für sich reserviert an.
Wenn nun ein Client mit einem Server reden möchte, dann muss er sich wie der Server einen eindeutigen Namen „ausdenken“ und im Netz reservieren. Das Verfahren dazu ist identisch. Zusätzlich

muss der Client jedoch die MAC-Adresse des Servers herausbekommen. Die Mechanismen, wie dies geschieht, hängen davon ab, wie NetBIOS implementiert ist.
NetBIOS kann mit unterschiedlichen Protokollen implementiert werden. NetBEUI, IPX und TCP/IP sind drei heute verwendete Protokolle, wobei für Neuinstallationen TCP/IP das bevorzugte Protokoll sein sollte. (Samba ist ausschließlich in der Lage, NetBIOS über TCP/IP zu benutzen.) Der Ablauf der Namensauflösung soll an einem Beispiel verdeutlicht werden.
Auf einem Client soll eine Verbindung zu dem Server `pdc` aufgebaut werden. Direkt erreicht man dies, indem man in der Taskleiste Start → Ausführen → \\`pdc` eingibt.
Bei TCP/IP muss der Client die IP-Adresse des Servers herausfinden. Dies geschieht wie bei den anderen Protokollen per Broadcast im lokalen Netz. IP-Router können nicht angewiesen werden, die Anfragen per Broadcast in alle angeschlossenen Netze weiterzuleiten. Aus diesem Grund gibt es hier andere Mechanismen, die im Folgenden beschrieben werden. Nachdem die IP-Adresse herausgefunden wurde, kommen die bekannten Mechanismen von IP zum tragen.

NetBIOS

- NetBIOS ist Software-Schnittstelle, entworfen für kleine Netze
- Verwendet eigenen Namensraum (NetBIOS Name $\neq$ DNS Name)
- beliebig viele Namen pro Rechner
- Kommunikation über Broadcasts
- Samba funktioniert nur über TCP/IP

Folie 262

19.2 Bestandteile von Samba

Das Programmpaket Samba besteht aus mehreren Teilen, von denen einige der Serverseite und andere der Clientseite zugeordnet werden können. Die Servertools:

- `smbd` ist der zentrale Serverprozess, der für die eigentlichen Datei- und Druckdienste zuständig ist. Sie werden mehrere smbds im System finden. Einer dieser Prozesse lauscht auf dem TCP-Port 139, und nimmt neue Verbindungen entgegen. Jede neue Verbindung stößt einen neuen Prozess smbd an. Samba ist im Hauptspeicherverbrauch recht sparsam. Jeder aktive Client benötigt etwa 1 MB Hauptspeicher auf dem Server. Clients, die gerade nicht aktiv Dateien mit dem Sambaserver austauschen, benötigen praktisch überhaupt keine Resourcen. Viel Hauptspeicher kann von Samba selbstverständlich gut als Cache genutzt werden.

- `nmbd` ist für die NetBIOS Namens- und Datagrammdienste zuständig. Dieser Prozess reserviert beim Start von Samba die entsprechenden NetBIOS-Namen, er ist WINS-Server und für den Computersuchdienst zuständig.

- `testparm` Mit diesem Programm kann man die smb.conf auf syntaktische Korrektheit prüfen.

- `smbpasswd` wird zur Pflege der verschlüsselten Passwörter auf Serverseite verwendet (siehe unten).

Die Clients sind:

- `smbclient` Mit dem Programm smbclient kann man auf Freigaben von NT-Rechnern zugreifen. Man kann auf von NT zur Verfügung gestellten Druckern drucken und man kann NT-Freigaben in tar-Dateien sichern. Weiterhin wird mit smbclient die Liste der Server im Netz erfragt, analog zu der Netzwerkumgebung unter Windows.

- `nmblookup` ist ein Diagnosewerkzeug für die NetBIOS-Namensauflösung. Wenn zwei Computer mit Windows sich nicht finden können, kann man mit `nmblookup` deren Versuche, sich gegenseitig zu finden, genau nachstellen. Ebenso können WINS-Server befragt werden und ein NetBIOS Node Status abgefragt werden. Das entsprechende Programm

auf Seiten von Windows ist das Kommandozeilenprogramm `nbtstat`.

- `smbmount` erlaubt das Mounten von SMB Shares.
- `smbprint` ermöglicht das Drucken auf SMB Druckern.

Auf der Serverseite finden sich noch weitere Komponenten:

- `smb.conf` ist die zentrale Konfigurationsdatei von Samba; meits in `/etc/`.
- `/var/lock/samba` für temporäre Lockdateien und Datenbanken
- `/etc/smbpasswd` ist die Passwortdatenbank von Samba, sofern mit verschlüsselten Passwörtern gearbeitet wird in `/usr/local/ samba/private/`

Bestandteile von Samba

Server:

- `smbd` ist der eigentliche Daemon für die Datei- und Druckerfreigabe.
- `nmbd` ist für die NetBIOS Namensauflösung zuständig.
- `testparm` überprüft zentrale Konfigurationsdatei.
- `smbpasswd` ist für die Passwörter der Windows Benutzer zuständig.

Client:

- `smbclient` erlaubt Zugriff auf Windows (und Samba) Shares.
- `smbmount` erlaubt das mounten solcher Shares.
- `smbprint` erlaubt das Drucken auf Printer Shares.

Folie 263

19.3 NetBIOS-Konfiguration mit Samba

Als erstes soll eine minimale Konfiguration von Samba erreicht werden, mit der jeder Rechner in der Netzwerkumgebung zu sehen ist. Dazu sollte die Datei `smb.conf` folgendermaßen aussehen:

```
[global]
workgroup = arbeitsgruppe
interfaces = <IP-Adresse>/<Netzmaske>
```

Der grundsätzliche Aufbau der `smb.conf` gleicht dem Aufbau der .INI-Dateien von Windows 3. Die Datei ist in mehrere Abschnitte unterteilt, die jeweils durch einen Abschnittsnamen eingeleitet werden. Dieser Abschnittsname selbst wird in eckige Klammern gesetzt. Der Inhalt jedes Abschnitts besteht nun aus Parameterzuweisungen. Im Beispiel gibt es nur den Abschnitt global. In diesem werden Festlegungen getroffen, die den Server als ganzes betreffen. Wenn später Freigaben erstellt werden, geschieht dies durch Anlegen von weiteren Abschnitten. Mit dem Parameter `workgroup` wird die Arbeitsgruppe festgelegt, in der sich der Server befinden soll. Der Parameter `interfaces` gibt das Netzwerkdevice an. Zusätzlich kann auch die Broadcastadresse angegeben werden.

Mit diesen beiden Einstellungen wird man direkt den Sambarechner in der Netzwerkumgebung sehen. Zur Vereinfachung sollten noch zwei weitere Parameter gesetzt werden, die später erklärt werden:

```
security = share
encrypt passwords = yes
```

Nun kann Samba (`smb` und `nmbd`) mittels des Initscripts `/etc/init.d/smb` gestartet werden. Meist wird Samba als Standalone Server verwendet. Es ist aber möglich den `smb` (aber nur diesen) über den Internetdaemon zu betreiben.

/etc/samba/smb.conf

- einzelne Teile werden mit [...] getrennt
- `[global]` ist der Hauptteil und zwingend erforderlich
- `workgroup` legt fest, in welcher Arbeitsgruppe/Domäne sich der Rechner anmelden soll
- `interfaces` gibt die Netzwerkadressen an, unter denen der Rechner erreichbar sein soll

Folie 264

19.4 Einfache Freigaben

Wenn der Sambarechner nicht nur im Netz gesehen werden soll, sondern auch sinnvolle Dinge tun soll, muss man Freigaben zur Verfügung stellen. Dies tut man, indem man einfach einen neuen Abschnitt beginnt. Um eine Freigabe vollständig zu machen, muss man mit dem Parameter `path` angeben, welches Verzeichnis man freigeben möchte. Eine für alle Zugriffe offene Freigabe des Verzeichnisses `/cdrom` erreicht man mit folgendem Eintrag in `smb.conf`:

```
[cdrom]
path = /cdrom
guest ok = yes
```

Damit entsteht auf dem Server eine Freigabe namens `CD`, die das Verzeichnis `/cdrom` im Netz für alle zum Lesen zur Verfügung stellt.
Achtung: Es findet hier *keine* Überprüfung der Zugriffsrechte statt. Um diese Überprüfung zu ermöglichen, sollte zunächst einmal der Aufbau einer Verbindung zu einer Freigabe genauer beleuchtet werden.
Um *Drucker* unter Samba zur Verfügung zu stellen, müssen diese zunächst unter Unix funktionieren. Alle Drucker, die via LPD, LPRng oder CUPS installiert sind, können als Netzwerkdrucker für Windowsclients freigegeben werden. Druckertreiber unter Windows gehen vom Windows-Metafile-Format aus, und wandeln dies entsprechend um. Das Windows-Metafile-Format enthält Aufrufe an die Graphische Komponente von Windows, das GDI. Wenn man einen Drucker, der über Unix angesprochen wird, von Windows aus nutzen möchte, muss man planen, wo die Aufbereitung in das druckereigene Format geschehen soll. Zwei Wege sind denkbar.

Dateifreigaben

- Dateien müssen zuerst freigegeben werden, um von Windows aus sichtbar zu sein
- Jede Freigabe ist ein eigener Abschnitt (zB: [cdrom]) in `/etc/samba/smb.conf`
- `path` gibt das Verzeichnis an, das freigegeben werden soll

Folie 265

Druckerfreigabe

- Funktioniert genauso wie Dateifreigabe
- `printable` gibt an, dass es sich um einen Drucker handelt
- `printer` gibt die Druckerwarteschlange unter Unix an
- `path` braucht eigenes Spool-Verzeichnis!

Folie 266

- Auf den Arbeitsplätzen wird ein generischer Postscripttreiber installiert. Die Clients müssen nicht wissen, welches Druckermodell sich hinter einer Freigabe verbirgt. Die Umwandlung findet auf dem Druckerserver mittels `ghostscript` statt.

- Auf den Arbeitsplätzen werden für jeden Netzdrucker die korrekten Treiber installiert.

Beide Wege haben Vor- und Nachteile. Im ersten Fall hat man weniger Aufwand mit der Administration auf Clientseite. Man muss den korrekten Druckertreiber nur einmal definieren, am Druckerserver. Beim zweiten Weg kann man die bessere Unterstützung der Druckerhersteller für die Windowsplattformen nutzen. Druckertreiber für Windows bieten in der Regel die Möglichkeit, Sonderfunktionen wie die Auswahl des Papierschachtes zu nutzen. Dieser erhöhte Komfort zieht jedoch nach sich, dass auf jedem Client der korrekte Druckertreiber installiert ist.
Eine Druckfreigabe wird genau wie eine Dateifreigabe in einem eigenen Abschnitt erstellt, wobei für die Druckfunktion drei Optionen notwendig sind:

```
[deskjet]
printable = yes
printer = lp
path = /tmp
```

Zu einer Druckfreigabe wird die Definition durch die Angabe `printable = yes`. Mit der Option `printer =` wird festgelegt, welche Druckerwarteschlange unter Unix angesprochen werden soll. Die Option `path =` legt einen Spoolbereich fest. Ein Druckjob, den ein Windowsrechner an Samba schickt, muss zunächst in einer Datei abgespeichert werden. Wenn diese Datei geschlossen wird, teilt der Client dem Server mit, dass diese nun zum Drucker geschickt werden soll. Samba realisiert dies, indem das Programm `lpr` mit der Druckdatei als Argument aufgerufen wird. Samba muss also für sich die Möglichkeit haben, Druckjobs in Dateien zu speichern, bevor sie an den `lpd` übergeben werden. Dies sollte nicht das Spoolverzeichnis sein, das der `lpd` selbst für den Drucker vorsieht.

19.5 Die Netzwerkumgebung

Die Netzwerkumgebung ist einer der instabileren Aspekte von Windows. Hiermit kann man sich, sofern alles funktioniert, alle Rechner in einer Arbeitsgruppe anzeigen lassen. Dabei dauert es mitunter geraume Zeit, bis ein Rechner in einer Anzeige erscheint, und es dauert unter Umständen noch länger, bis er wieder verschwindet. Eine naive Implementation könnte funktionieren, indem jeder Rechner, der Serverdienste anbietet, dieses regelmäßig per Broadcast im Netz mitteilt. Ein solches Vorgehen hat jedoch mehrere Nachteile. Erstens würde die Last im Netz mit jedem zusätzlichen Rechner stark ansteigen. Zweitens muss jeder Rechner, der die Netzwerkumgebung anzeigen will, relativ komplexe Software laufen lassen. Und drittens scheitert dieses Schema auf jeden Fall an Subnetzgrenzen, die für Broadcasts eine Grenze darstellen. Aus diesen Gründen ist man einen anderen Weg gegangen.
Der *Lokale Master Browser* (im Folgenden auch LMB genannt) ist ein Rechner, der im Netz die Netzwerkumgebung pflegt. Dieser Rechner wird nirgendwo zentral bestimmt, sondern er wird gewählt. Diese Wahl findet immer dann statt, wenn einer der beteiligten Rechner feststellt, dass es im Moment keinen solchen Lokalen Master Browser gibt. Beispielsweise kann der Explorer von Windows eine solche Wahl anstoßen. Wenn Windows 95 die geschwenkte Taschenlampe anzeigt, wird der LMB gesucht. Ist keiner vorhanden, wird eine Wahl angestoßen.
Die Wahl erfolgt mit Datagrammen an den Gruppennamen `arbeits gruppe`. Ein Rechner verschickt ein Datagramm an diesen Namen. Jeder Rechner, der diesen Namen reserviert hat, hört dieses Datagramm und entscheidet, wie er selbst vorgehen soll. In dem Datagramm sind verschiedene Kriterien zur Wahl enthalten, beispielsweise das Betriebssystem des versendenden Rechners.
Empfängt beispielsweise eine Windows NT Workstation ein Paket von einem Windows NT Server, so entscheidet sie, dass sie die Wahl verloren hat. Damit wird sie selbst nicht mehr aktiv. Kommt dieses Paket jedoch von einem Windows 95 Rechner, so hält sie sich selbst für geeigneter, den Lokalen Master Browser zu übernehmen. Dann wird sie selbst ein solches Wahlpaket mit ihren Parametern versenden. Der Windows 95 Rechner empfängt dies, und sieht, dass er verloren hat. Auf diese Weise schaukelt

sich die Wahl hoch, bis der beste Rechner die Wahl gewinnt.
Wenn es nun mehrere Windows NT Workstations im Netz gäbe, dann wäre die Wahl unentschieden. An dieser Stelle kommt die *Uptime* der Rechner ins Spiel. Der Rechner, der am längsten läuft, gewinnt die Wahl. Nun kann es sein, dass nach einem Stromausfall zwei Rechner genau die gleiche Uptime haben. Dann kommt als letztes und eindeutiges Entscheidungskriterium der NetBIOS-Name des Rechners zum Zug. Der alphabetisch vorne stehende Rechner gewinnt. Mit diesen drei Kriterien ist eine eindeutige Wahl gesichert.
Samba ordnet sich in der Standardeinstellung zwischen Windows 95 und Windows NT ein, das heißt, gegen Windows 95 gewinnt Samba die Wahl, überlässt jedoch Windows NT Rechnern den Lokalen Master Browser.
Drei Parameter in der `smb.conf` bestimmen das Verhalten von Samba in der Wahl zum Lokalen Master Browser:

- **os level** Damit wird die Einordnung von Samba in die unterschiedlichen Betriebssysteme geregelt. Diese haben für die Betriebssystemstufe folgende Werte:

Windows for Workgroups	0
Windows 95/98	1
Windows NT Workstation	16
Windows NT Server	32

  Diese Werte sind nicht als fest anzusehen. Wenn ein neues Service Pack für ein Betriebssystem herausgegeben wird, ist es möglich, dass in der Software für den Lokalen Master Browser Fehler bereinigt wurden. Dann ist es sinnvoll, dass diese neue Software die Rolle des LMB übernimmt. Der einfachste Weg ist, den `os level` einfach hochzusetzen. Samba hat hier einen Vorgabewert von 20.
  Der Parameter `os level` kann Werte von 0 bis 255 annehmen. Setzt man ihn auf 255, wird nach einer erfolgreichen Wahl niemand mehr Local Master Browser werden können.

- **local master** Möchte man auf keinen Fall den LMB auf einem Sambarechner haben, so setzt man den Parameter `local master = no`. Dann nimmt Samba an keiner Wahl teil.

- **preferred master** Mit der Standardeinstellung `preferred master = no` sucht Samba beim Start nach einem LMB. Findet er einen, meldet er sich dort. Findet er keinen LMB, bleibt Samba passiv. Jemand anders muss eine Wahl anstoßen. Wenn dann eine Wahl stattfindet, nimmt Samba teil und ordnet sich anhand seines os level ein. Wenn man sicher gehen möchte, dass Samba auf jeden Fall nach dem Start den LMB übernimmt, dann muss man den `os level` hoch genug setzen, und den Parameter `preferred master = yes` setzen. Damit wird Samba beim Start des `nmbd` auf jeden Fall eine Wahl anstoßen und sie dann unter Umständen gewinnen.

Mit den Einstellungen
```
[global]
os level = 66
preferred master = yes
```
kann man sicher sein, dass der Sambarechner immer den LMB innehat. Es sei denn, ein anderer Administrator von Samba kommt auf die Idee, einen noch höheren Wert für den `os level` zu benutzen.
Ein Primary Domain Controller kann unter Umständen erheblich gestört werden, wenn er in seinem Subnetz nicht der LMB ist.

Netzwerkumgebung

- Ein *Local Master Browser* sammelt die Broadcasts des Subnetzes; wird gewählt
- `os level` gibt die „Chance“ an, mit der Samba die Wahl gewinnt
- `local master = no` verhindert, dass Samba an der Wahl teilnimmt
- `preferred master = yes` Samba fängt eine Wahl an, falls er nicht LMB ist

Folie 267

19.6 NetBIOS über Subnetzgrenzen

Wird die Namensreservierung und -auflösung ausschließlich per Broadcast durchgeführt, kann man Rechner, die hinter Routern liegen, nicht erreichen. Broadcasts verbleiben in den Subnetzen, in denen sie ausgesendet wurden.
Sind zum Beispiel zwei Netze über einen Router verbunden, so reserviert jeder der beiden Rechner seinen Namen in dem ihm zugeordneten Subnetz. Die Workstation `WKS` schickt ihre Reservierungen per Broadcast im Subnetz 1, und der Server `SERVER` wird seinen Namen im Subnetz 2 reservieren. Der Router zwischen beiden bekommt diese Reservierungen zwar mit, wird sie aber nicht in das jeweils andere Subnetz weiterleiten. Wenn nun `WKS` ihren Server `SERVER` sucht, geschieht dies ebenfalls per Broadcast im Subnetz 1. Diese Anfrage bleibt in ihrem Subnetz und erreicht `SERVER` gar nicht, so dass dieser auch nicht antworten kann.
Der einfachste Weg, die Namensauflösung über Subnetzgrenzen hinweg zu realisieren, geht über eine statische Tabelle. Unter Windows liegt diese in der Datei `LMHOSTS`. Sie liegt abhängig von der Windowsversion in unterschiedlichen Verzeichnissen und lässt sich am einfachsten mit der Suchfunktion des Desktops finden. Diese Datei ist ähnlich aufgebaut wie die Datei `/etc/hosts` unter Unix. Ein Beispieleintrag ist der folgende:

```
192.168.1.5 samba
```

Die Einträge in der `LMHOSTS` können durch den Zusatz `♯PRE` ergänzt werden. Dieser Zusatz legt fest, in welcher Reihenfolge die Namensauflösung vorgenommen wird. Ist kein `♯PRE` vorhanden, so wird zunächst eine konventionelle Namensauflösung per Broadcast versucht. Erst, wenn diese fehlschlägt, wird in der `LMHOSTS` nachgeschaut. Ist der Zusatz vorhanden, so wird ohne Namensauflösung direkt der Wert in der `LMHOSTS` verwendet.
Die zweite Möglichkeit, das Problem zu lösen, ist eine zentrale Datei `LMHOSTS`. Dazu gibt es den WINS-Server. Ein solcher Server ist ein Rechner, bei dem sich jede Applikation im Netz mit ihren Namen anmeldet. Die IP-Adresse dieses Servers muss jedem Rechner mitgeteilt werden. Bei Windows geschieht dies in den Eigenschaften des TCP/IP Protokolls im Reiter WINS-Adresse. Setzt man DHCP-Server ein, kann man ebenfalls den WINS-Server festlegen. Samba bekommt die Adresse mit dem Parameter `wins server = <ip-adresse>` im Ab-

schnitt `[global]` der `smb.conf` mitgeteilt. Sobald ein Client die IP-Adresse des WINS Servers kennt, ist es völlig gleichgültig, ob sich dieser im gleichen Subnetz befindet oder nicht.
Die Namensreservierung erfolgt nicht mehr per Broadcast, sondern mit einem gerichteten UDP-Paket an den WINS-Server. Gerichtete Pakete leitet der Router wie jedes andere Paket an den WINS-Server weiter. Dieser sieht in seiner Tabelle nach, ob der Name bereits reserviert ist. Ist das nicht der Fall, so wird er spontan eine Bestätigung der Reservierung zurückschicken. Diese Reservierung gilt nun für eine bestimmte Zeit und muss rechtzeitig erneuert werden.
Ist der Name bereits reserviert, wird der WINS-Server den bisherigen Besitzer befragen, ob er den Namen noch benötigt. Bekommt er keine Antwort, wird er dem neuen Besitzer ebenfalls eine Bestätigung schicken. Möchte der alte Besitzer den Namen noch verwenden, so wird der Anfragende eine Ablehnung der Reservierung erhalten. Diese Nachfrage ist notwendig, um einem abgestürzten Rechner das spontane Booten zu ermöglichen, da bei einem Absturz keine Freigabe der Namensreservierung erfolgen kann.
Die Namensanfrage, die den Server nicht erreichte, weil der Router keine Broadcasts weitergibt, wird nun direkt an den WINS-Server gerichtet, der in seiner Tabelle nachsehen kann.
Samba kann ganz normal als WINS-Server konfiguriert werden, indem der Parameter `wins support = yes` gesetzt wird. Ist diese Parameter gesetzt, kann Samba nach einem Neustart bei allen Clients und allen sonstigen Servern als WINS-Server eingetragen werden. Werden diese dann neu gestartet, melden sie sich beim WINS-Server an.
Wenn nun ein Rechner mit Samba als WINS-Server konfiguriert ist, und sich die anderen Rechner dort anmelden, werden diese in der Datei `/var/lock/samba/wins.dat` abgelegt. Der `nmbd` pflegt diese Datei dynamisch, je nach Reservierungen und Abmeldungen. Die Datei `wins.dat` wird in regelmäßigen Abständen geschrieben. Wenn es notwendig sein sollte, den wirklich aktuellen Stand unabhängig von diesem Zeitintervall zu erhalten, so kann man dem `nmbd` das HANGUP-Signal durch den Befehl `killall -HUP nmbd` senden. Außerdem wird die `wins.dat` beim Beenden des `nmbd` geschrieben.
Diese Datenbank wird auf Festplatte gehalten, damit die Da-

ten einen Neustart von Samba überleben. Jeder Rechner, der einen Namen für sich reserviert hat, hat diese Reservierung für einen bestimmten Zeitraum ausgesprochen. Wenn Samba jetzt neu gestartet werden sollte, und dadurch die Datenbank verloren ginge, wäre der gesamte NetBIOS-Namensraum nicht mehr verfügbar. Außerdem kann ein WINS-Server die angeschlossenen Clients weder von sich aus finden, noch sie darum bitten, sich erneut zu registrieren. Daher ist die WINS Datenbank über Neustarts von Samba hinaus zu erhalten.
Die Anfrage, die die Workstation `WKS` absetzt, wird nun nicht mehr per Broadcast gestellt, sondern mit einem gerichteten Paket an den WINS-Server, bei dem sich alle Rechner angemeldet haben.
WINS hat gegenüber der broadcastbasierten Namensreservierung einige Vorteile. Namensreservierung per Broadcast erfolgt durch Wartezeiten. Es wird die Reservierung angekündigt, es wird gewartet, die Reservierung wird erneut angekündigt, und es wird wieder gewartet. Dieses Spiel wiederholt sich mehrfach, bis der Rechner sicher sein kann, dass ein eventueller Vorbesitzer des Namens genug Zeit hatte, sich zu beklagen. Beim Einsatz von WINS entfallen diese Wartezeiten, da hier ein einziger Rechner sämtliche reservierte Namen registriert und in seiner Tabelle nachschauen kann. Daher ist die Reservierung per NetBIOS deutlich schneller, und auch weniger netzbelastend. Selbst wenn man also nur ein einziges Subnetz hat, sollte man zur Reduzierung der Netzlast den Einsatz eines WINS-Servers in Erwägung ziehen.
Zusätzlich sei hier angemerkt, dass es netzwerkweit nur einen einzigen WINS-Server geben darf. Selbst wenn es unterschiedliche Arbeitsgruppen oder Domänen gibt, darf es nicht mehr als einen WINS-Server geben. Setzt man mehrere WINS-Server ein, hat man getrennte Namensräume. Rechner im einen Namensraum können mit Rechnern, die an einem anderen WINS-Server angeschlossen sind, nicht kommunizieren. Es kann trotzdem zu Kollisionen kommen, da Windowsrechner bestimmte Namen unabhängig von WINS-Einstellungen ausschließlich per Broadcast reservieren. Unter Windows NT kann man mehrere WINS-Server einsetzen, die sich gegenseitig abstimmen. Diese WINS-Server treten gegenüber den Clients als ein einziger Server auf, unabhängig von ihrer Anzahl.

Die Abfrage eines WINS Servers durch `nmblookup` erfolgt beispielhaft folgendermaßen:

```
nmblookup -R -U 192.168.1.5 samba
```

Hiermit wird der WINS Server, der auf dem Rechner 192.168.1.5 liegt, nach dem Namen `samba` befragt.

Samba kennt zwei zusätzliche Funktionen, die es im Zusammenhang mit WINS interessant machen. Einerseits kann Samba als WINS Proxy eingerichtet werden, indem `wins proxy = yes` gesetzt wird. Ist diese Einstellung aktiv, dann wird Samba sämtliche Reservierungen und Anfragen, die es aus dem lokalen Netz per Broadcast erhält, an den mit `wins server =` konfigurierten WINS-Server weiterleiten. Damit kann man einen Samba-Server in ein Subnetz stellen. Sämtliche Rechner in diesem Netz werden nun beim WINS angemeldet, und nutzen diesen auch. Dies ist auch dann der Fall, wenn sie entweder selbst keinen WINS-Server ansprechen können oder nicht dafür konfiguriert sind. Man sollte jedoch in jedem Fall eine echte Konfiguration des WINS Servers auf dem Client vorziehen. Ein WINS-Proxy kann nur eine Behelfslösung sein, da man sich damit auf einen weiteren Rechner verlässt.

Unter Windows kann man statische Einträge im WINS vornehmen. Dies geht so direkt unter Samba nicht. Man muss hierzu den Parameter `dns proxy = yes` auf dem WINS-Server setzen. Empfängt der WINS-Server nun eine Anfrage, die er nicht aus seiner Datenbank beantworten kann, wird er eine ganz normale Unix-Hostnamenanfrage machen. Typischerweise wird er in der `/etc/hosts` nachschauen und danach dann das DNS anhand der Konfiguration in der Datei `/etc/resolv.conf` befragen. Damit ist es durch einen Eintrag auf dem WINS Server möglich, den gesamten DNS-Namensraum auch in der NetBIOS-Namenswelt zur Verfügung zu stellen.

WINS Server

- Namensreservierung funktioniert nur im eigenen Subnetz
- `LMHOSTS` nur statische Möglichkeit, einen Server außerhalb zu erreichen
- WINS Server verwaltet alle Rechner im Netzwerk (d.h. Domäne/Arbeitsgruppe)
- `wins server` gibt Samba den WINS Server bekannt
- `wins proxy = yes` lässt Samba die Datenbank des WINS Servers spiegeln
- `wins support = yes` Samba wird selbst WINS Server

Folie 268

19.7 SMB-Sitzungen

Wird am Client eine Verbindung zu einer Freigabe auf einem SMB-Server aufgebaut, so müssen mehrere Schritte durchlaufen werden.

- **NetBIOS-Namensauflösung** Zu einem Rechnernamen muss eine IP-Adresse herausgefunden werden. Dies wurde bereits eingehend behandelt.

- **TCP-Verbindung** Wenn die IP-Adresse klar ist, wird eine TCP-Verbindung zu Port 139 des Servers aufgebaut. Um vorhandene TCP-Verbindungen anzuzeigen, gibt es sowohl auf Unix- als auch auf Windowsrechnern das Werkzeug `netstat`.

- **NetBIOS-Sitzung** Auf einem Serverrechner arbeiten unter Umständen mehrere Applikationen, die Namen für sich reserviert haben. Diese sind alle unter der IP-Adresse des Rechners und dem TCP-Protokoll auf Port 139 erreichbar. Anhand des TCP-Verbindungsaufbaus ist nicht klar, welche Serverapplikation angesprochen werden soll. Die Unterscheidung wird durch den Servernamen getroffen, der in der TCP-Verbindung als erstes übertragen wird.

  Dass der Servername übertragen wird, kann man ganz einfach mit Hilfe des Programms `smbclient` sehen. Man versucht, sich die Liste der Freigaben eines realen Windowsrechners geben zu lassen, indem man Folgendes aufruft:

  ```
  smbclient -L smallwin
  ```

  Damit wird zunächst eine NetBIOS-Namensanfrage ausgelöst, und dann eine Verbindung zum entsprechenden Server ausgelöst. smbclient hat jedoch die Möglichkeit, einen Server unter einem anderen Namen anzusprechen, indem man `smbclient -L test -I ip-adresse` eingibt. `smbclient` wird zunächst versuchen, eine Verbindung zum NetBIOS-Namen `test` aufzubauen, und zwar ohne dass eine NetBIOS-Namensanfrage ausgelöst wird. Stattdessen wird die angegebene IP-Adresse auf Port 139 direkt angesprochen, und der Name `test` als Servername angegeben. Windows merkt, dass das nicht stimmen kann und verweigert den Verbindungsaufbau mit einer Fehlermeldung. Erst im

zweiten Versuch wird es `smbclient` gelingen, eine Verbindung aufzubauen, da diese Verbindung zum allgemeinen Namen `*smbserver` aufgebaut wird. Auch der Clientname wird in der Verbindung übergeben. Dies testet man am besten mit

```
smbclient //win/c\$ -n blafasel
```

und schaut sich die Verbindungstabelle auf der Windowsmaschine mit `nbtstat -s` an.

Mit dem übergebenen Servernamen kann man sehr nette Tricks anstellen. Man stelle sich vor, dass einige Freigaben nur für bestimmte Clientrechner sichtbar sein sollen. Dies ist mit Bordmitteln von Samba so nicht möglich. Man kann zwar mit dem Parameter `browseable` festlegen, ob bestimmte Freigaben in der Netzwerkumgebung erscheinen. Dieser Parameter hat aber zwei Nachteile. Erstens sind die Freigaben nur unsichtbar geworden, darauf zugreifen kann man immer noch. Zweitens kann man Freigaben nur für alle Rechner verstecken oder freigeben.

Samba bietet die Option, unter zwei oder mehreren verschiedenen Namen in der Netzwerkumgebung zu erscheinen. Mit dem Parameter `netbios name` gibt man einen Namen für den Server an. Zusätzliche Namen kann man mit `netbios aliases` vergeben. Mit

```
netbios name = fichte
netbios aliases = birke eiche kiefer buche
```

handelt man sich einen ganzen Wald in der Netzwerkumgebung ein. Klickt man auf die einzelnen Server, sieht man überall die gleichen Freigaben und Zugriffsrechte. Nun kann man für jeden dieser virtuellen Rechner eine eigene Konfigurationsdatei anlegen. Beispielsweise kann man sie `/etc/samba/smb.conf.birke`, `/etc/samba/smb.conf.eiche` und so weiter nennen. Die Datei `/etc/ smb.conf` ist für den Rechner `fichte` zuständig und enthält neben den Einstellungen für `fichte` den Parameter

```
config file = /etc/samba/smb.conf.%L
```

Dabei steht `%L` für den Servernamen, unter dem Samba angesprochen wird. Wenn es eine passende Datei gibt, dann

bewirkt der Parameter `config file`, dass die komplette Konfiguration neu eingelesen wird. Existiert keine passende Datei, so wird der Parameter einfach ignoriert. Um nun den Zugriff nur für einzelne Clients zu erlauben, kann bei den einzelnen virtuellen Servern mit den Parametern `hosts allow` und `hosts deny` der Zugriff geregelt werden.

- **Negotiate Protocol** Die NetBIOS-Sitzung ist nun aufgebaut, und es können Daten übermittelt werden. Innerhalb dieser NetBIOS-Sitzung wird eine SMB-Sitzung schrittweise aufgebaut. SMB ist ein Protokoll, bei dem im Prinzip der Client jede Aktion durch eine Anfrage anstößt, und der Server diese beantwortet.

  Die erste Anfrage, die der Client an den Server schickt, ist ein *Negotiate Protocol Request*. In dieser Anfrage schickt der Client an den Server eine Liste der Protokollvarianten, die er beherrscht. Der Server wählt nun aus dieser Liste der Protokolle eines aus, und schickt eine entsprechende Antwort zurück. Die verschiedenen Protokolle bauen aufeinander auf. Daher kann man mit dem Parameter `protocol` das höchste Protokoll festlegen, mit dem Samba arbeiten soll. In der Antwort auf diese erste Anfrage werden zwei weitere Einstellungen verschickt, die Teile des weiteren Ablaufs festlegen.

  In der Antwort auf diese erste Anfrage werden zwei weitere Einstellungen verschickt, die Teile des weiteren Ablaufs festlegen.

  Der Server entscheidet, ob er die Zugriffssteuerung auf Benutzer- oder auf Freigabeebene regeln möchte. Damit wird festgelegt, zu welchem Zeitpunkt der Benutzer ein Passwort liefern muss. Entweder kann es beim direkt folgenden *Session Setup* erfolgen, oder erst beim *Tree Connect* danach.

  Der Parameter `security` legt fest, welche Art der Zugriffssteuerung gewählt wurde. Mit `security = share` wird die Freigabeebene eingestellt, `security = user` legt die Clients auf die Benutzerebene fest.

  Sichtbar wird diese Unterscheidung in der Windowswelt nur bei Windows 95 und Windows 98. Diese Betriebssysteme

beherrschen zunächst einmal nur die Zugriffssteuerung auf Freigabeebene, da sie nicht über eine Benutzerdatenbank verfügen. Es ist nicht möglich, einzelnen Benutzern den Zugriff auf Freigaben zu gewähren oder zu verweigern. Um trotzdem benutzerbasiert Zugriffssteuerung zu ermöglichen, muss ein Server angegeben werden, der für Windows die Benutzerdatenbank pflegt. Damit können Passwörter benutzerbasiert überprüft werden.

Weiterhin gibt der Server dem Client vor, ob Klartextpasswörter verwendet werden sollen, oder ob die Passwörter verschlüsselt werden. Wenn der Server festlegt, dass verschlüsselte Passwörter verwendet werden, wird zusätzlich die Herausforderung für das *Challenge Response* Verfahren mitgeschickt.

Die Entscheidung über Klartextpasswörter muss also getroffen werden, ohne dass der Server den Benutzernamen, der sich anmelden will, kennt. Es ist also nicht möglich, für einige Benutzer Klartextpasswörter und für andere Benutzer verschlüsselte Passwörter zu verwenden.

- **Session Setup** Nachdem die Protokollversion ausgehandelt ist, wird vom Client ein *Session Setup* verschickt. In diesem Session Setup schickt der Client seinen Benutzernamen an den Server. Sofern dieser `security = user` verlangt hat, wird an dieser Stelle das Passwort mitgeschickt. Damit ist der Server in der Lage, die Identität des Benutzers festzustellen. Wenn `security = share` vereinbart wurde, dann ignoriert der Server ein hier eventuell mitgeschicktes Passwort.

- **Tree Connect** Als letztes legt der Client fest, welche Freigabe er ansprechen will. Der entsprechende Aufruf heißt *Tree Connect*. Sofern `security = share` vereinbart wurde, wird an dieser Stelle das Passwort überprüft. Der Benutzername kann in diesem Fall nicht zur Zugriffsregelung verwendet werden. Dieser wurde unter Umständen gar nicht übermittelt, da der Client den Session Setup komplett auslassen darf. Andererseits hat er bei einem durchgeführten Session Setup kein Passwort angeben müssen, anhand dessen die Identität des Benutzers zweifelsfrei hätte festgestellt werden können.

weitere Optionen

- `smbclient -L <Rechnername>` zeigt die Freigaben am angegebenen Rechner an
- `netbios name` setzt den NetBIOS Namen von Samba (muss nicht der FQDN sein)
- `netbios aliases` weitere Namen, die Samba für sich reserviert
- `config file` weitere Konfigurationsdatei, die eingebunden wird (am besten mit % L verwenden)
- `security = share` setzt die Sicherheit auf Freigabeebene (alle dürfen zugreifen); andere Parameter: `user, server, domain`

Folie 269

19.8 Zugriffsrechte

Bei Windows NT kann man mit zwei unterschiedlichen Mechanismen Rechte vergeben. An einer Freigabe kann man über Schreib- und Lesezugriff entscheiden. Innerhalb des Dateisystems kann man detailiert Rechte vergeben.
Ist bei Samba `security = user` gesetzt, so hat der Server die Möglichkeit, anhand des angemeldeten Benutzers Zugriffsrechte zu vergeben oder zu verweigern. Wenn bezüglich der Zugriffsrechte bei einer Freigabe nichts gesagt wird, hat jeder korrekt angemeldete Benutzer Leserecht. Man kann auch Gastbenutzern Leserecht geben, indem man `guest ok = yes` setzt.
Mit den Optionen zur Rechtevergabe an Freigaben hat man die Möglichkeit, einzelnen Benutzern und ganzen Unixgruppen Rechte zu geben oder zu nehmen. Die Möglichkeiten sind hier deutlich weiter gehend als die Semantik, die Unix mit den Rechtemasken für den Dateibesitzer, die besitzende Gruppe und den Rest der Welt bereit stellt. Von den möglichen Anwendungen sollen hier drei häufig benötigte Fälle dargestellt werden.

- **Alle Benutzer haben gleichen Zugriff**

  ```
  [projekt]
  path = /data/projekt
  ```

  Bei dieser Freigabe bekommen alle Benutzer, die sich mit Namen und Passwort am Server angemeldet haben, *Leserecht* auf die Freigabe. Schreibrecht vergibt man, indem man den Parameter `writeable = yes` setzt:

  ```
  [projekt]
  path = /data/projekt
  writeable = yes
  ```

- **Einige Benutzer haben gleichen Zugriff**

  Will man den Zugriff auf einige Benutzer einschränken, erstellt man eine Liste `valid users` auf:

  ```
  [projekt]
  path = /data/projekt
  valid users = mueller, meier
  ```

  Zu dieser Freigabe haben die Benutzer `mueller` und `meier` Lesezugriff. Sollen diese Benutzer Schreibzugriff bekom-

men, so ist wie im vorangegangenen Beispiel der Parameter `writeable = yes` zu setzen:

```
[projekt]
path = /data/projekt
valid users = mueller, meier
writeable = yes
```

Für den Parameter `valid users` spielt der Benutzer `root` keine besondere Rolle. Das heißt, dass er auf die Freigabe `projekt` keinen Zugriff hat. Soll er Zugriff bekommen, muss man ihn wie jeden anderen Benutzer in die Liste `valid users` mit aufnehmen.

Der Parameter `valid users` gibt die Möglichkeit, ganze Unixgruppen in den Zugriff mit aufzunehmen. Um dies zu erreichen, muss man das `@`-Zeichen voranstellen:

```
[projekt]
path = /data/projekt
valid users = root, @users
writeable = yes
```

Mit dieser Einstellung haben alle Benutzer, die in der Unixgruppe users sind, Schreibzugriff auf die Freigabe. Zusätzlich kann der Benutzer root schreiben.

- **Einige Benutzer haben Leserecht, andere Schreibrecht**

  Will man differenziert Rechte vergeben, so muss man sämtliche Benutzer, die überhaupt Zugriff auf die Freigabe bekommen sollen, in die Liste `valid users` aufnehmen, und mit `writeable = no` nur Leserechte vergeben. Die Benutzer, die über diese Standardeinstellung hinaus Schreibrecht bekommen sollen, müssen in die `write list` aufgenommen werden.

```
[projekt]
path = /data/projekt
valid users = @users, @admins
writeable = no
write list = @admins
```

Mit diesen Einstellungen haben die Benutzer der Gruppe users Leserecht, und die Benutzer der Gruppe `admins` haben Schreibrecht.

Zugriffsrechte

- ohne weitere Optionen darf jeder auf die Freigabe zugreifen
- `writable` allgemeiner Schreibschutz
- `valid users` diese User dürfen auf die Freigabe zugreifen
- `write list` diese User dürfen Schreiben (auch wenn `writable =no`

Folie 270

Unix-Zugriffsrechte
Unter Windows NT gibt es zwei Möglichkeiten, Zugriff auf Dateien zu gewähren. Über eine Freigabe kann ein Lese- oder ein Schreibrecht vergeben werden. Im zweiten Schritt können dann über eine Rechtevergabe im Dateisystem weitere Rechte vergeben werden. Samba regelt die Zugriffskontrolle ebenfalls in zwei Schritten. Die freigabebezogenen Rechte werden über Parameter wie `valid users` und `write ok` geregelt. Die Zugriffsrechte innerhalb des Dateisystems regelt Samba nicht selbst, sondern verlässt sich hierfür auf das darunterliegende Betriebssystem Unix.
Zwischen Unix und DOS bestehen große Unterschiede. DOS und alle seine Nachfolger sind Einzelbenutzersysteme, Unix ist von Anfang an als Multiusersystem entworfen worden. Diese Unterschiede werden besonders deutlich, wenn man die Attribute betrachtet, die auf Dateien vergeben werden. DOS kennt vier Attribute:

- **Read-Only** Der Inhalt dieser Datei kann nur gelesen, aber nicht geschrieben werden. Die Datei kann nicht gelöscht werden.
- **System** Diese Datei ist für spezielle Betriebssystemzwecke vorgesehen.
- **Hidden** Diese Datei wird mit dem Kommando „DIR“ nicht angezeigt.
- **Archiv** Das Archivbit wird bei jedem Schreibzugriff gesetzt. Backupprogrammen ist es freigestellt, dieses Bit zurückzusetzen. Damit kann eine inkrementelle Sicherung ermöglicht werden.

Diese Bits können vom Benutzer frei gesetzt und wieder zurückgesetzt werden. Sie bieten also keinen echten Zugriffsschutz, sondern nur eine gewisse Sicherung gegen Fehlbedienung. Unter DOS werden Ausführungsrechte nicht verwendet. Sie stehen für Samba zur Verfügung, um die DOS-Attribute im Unix-Dateisystem abzubilden. Das Schreibschutzbit unter DOS hat mit dem Schreibrecht des Dateibesitzers unter Unix eine Entsprechung. Bis auf die Umsetzung des Schreibschutzbits kann die Umsetzung der Attribute unter Samba mit den entsprechenden

Parametern `map <xxx>` gesteuert werden, wobei das Archivbit ohne Zusatzangabe umgesetzt wird, die anderen beiden Attribute nicht. Die Attribut-Umsetzung geschieht an Hand der folgenden Tabelle:

DOS-Attribut	Unix-Recht	Maske	Parameter	Standard
Schreibschutz	rw-******	200	-	immer
Archiv	r-x******	100	`map archive`	`yes`
System	***r-x***	010	`map system`	`no`
Versteckt	******r-x	001	`map hidden`	`no`

Samba muss nun diese beiden Dateiattribute ineinander überführen. Samba muss neu erstellten Dateien Unixrechte zuordnen. Wird eine Datei neu erstellt, dann gibt der Client dem Server die DOS-Attribute mit, mit der er die Datei erstellt haben möchte. Daraus formt Samba einen Satz von Unix-Zugriffsrechten. Diese Rechte werden vom Parameter create mask eingeschränkt. Die Standardvorgabe für die `create mask` ist gleich `744`, was der Rechtemaske `rwxr-r-` entspricht. Der Dateieigentümer hat Schreib- und Leserecht, alle anderen haben reines Leserecht. Samba schränkt die Rechte ein, indem der gewünschte Satz an Rechten mit einer logischen UND-Operation mit der `create mask` verknüpft wird. Nur die Rechte, die in der `create mask` gesetzt sind, können möglicherweise in der neu erzeugten Datei auftauchen. In einem weiteren Schritt setzt Samba explizit gewünschte Zugriffsrechte anhand des Parameters `force create mode`, dessen Standardwert auf `000` steht. Dies geschieht durch eine ODER-Verknüpfung mit diesem Wert.
Diese Zusammenhänge werden an einem Beispiel deutlicher. Es kann gewünscht sein, dass auf neu erstellten Dateien nur der Dateibesitzer und die Gruppe Leserecht haben sollen. Der Rest der Welt soll diese Dateien nicht lesen können. Das wird dadurch erreicht, dass man die `create mask = 740` setzt, also das Leserecht für den Rest der Welt ausmaskiert. Es kann darüber hinaus gewünscht sein, dass die besitztende Gruppe ein Schreibrecht eingeräumt bekommt. Das kann man durch `force create mode = 020` erreichen. Tabellarisch dargestellt heißt dies:

Wunsch			rw-r--r--
create mask	740	UND	rw-r----- rw-r-----
force create mode	020	ODER	----w----
Ergebnis			rw-rw----

Die Ausführungsrechte auf Dateien werden unter DOS nicht verwendet, sie können also verwendet werden, um DOS-Attribute im Unix-Dateisystem abzulegen. Ausführungsrechte auf Dateiverzeichnissen wirken sich jedoch auf das Verhalten von Samba aus, da durch sie der Zugriff zu den Verzeichnissen geregelt wird. Daher kann es wünschenswert sein, dass die Rechtezuweisung auf Dateien und Verzeichnissen unterschiedlich geregelt wird. Die Parameter `create mask` und `force create mode` wirken daher nur auf neu angelegte Dateien. Für Verzeichnisse sind die Parameter `directory mask` und `force directory mode` verantwortlich. Der Vergabewert für `directory mask` ist hierbei `755`, um den Zutritt für die Gruppe und den Rest der Welt zu ermöglichen, die Vorgabe für `force directory mode` besetzt mit dem Wert `000` kein zusätzliches Recht.

Unix-Zugriffsrechte

- `create mask` gibt die Zugriffsrechte im `chmod` Stil für neu erstellte Dateien an
- `force create mode` gibt zusätzliche Zugriffsrechte an, die immer angehängt werden
- `directory mask` gibt die Zugriffsrechte für neu erstellte Verzeichnisse an
- `force directory mode` analog zu `force create mask` für Verzeichnisse

Folie 271

Passwörter
Im SMB-Protokoll wird zur Authentifizierung ein Challenge-Response Verfahren eingesetzt. Der Server verschickt an den Client eine Zufallszahl, die sogenannte Herausforderung. Der Client kennt das Benutzerpasswort und verschlüsselt die Herausforderung mit dem Passwort als Schlüssel. Diesen verschlüsselten Wert verschickt der Client anstelle des Passworts. Der Server kennt das Benutzerpasswort ebenfalls, und kann den verschlüsselten Wert entschlüsseln. Entsteht bei der Entschlüsselung wieder die Herausforderung, so hat der Benutzer die Herausforderung offensichtlich mit dem korrekten Passwort verschlüsselt. Kommt etwas anderes heraus, war das Passwort nicht richtig.
Authentifizierung unter Unix setzt voraus, dass der Client dem Server das Klartextpasswort präsentiert. Der Server kann daraus den Hashwert berechnen, und mit dem gespeicherten Wert vergleichen. Leider verfügt er nicht über das Klartextpasswort des Benutzers, um das Challenge-Response Verfahren durchführen zu können. Daher muss unter Samba für die Passwortverschlüsselung eine zweite Passwortdatenbank gepflegt werden, die Datei `smbpasswd`.
Auch in der Datei `smbpasswd` stehen keine Klartextpasswörter. Bevor die Herausforderung mit dem Passwort verschlüsselt wird, wird das Passwort unter Windows ebenfalls durch eine Hashfunktion geschickt. Von dieser Hashfunktion gibt es zwei Varianten, die beide nicht mit den unter Unix verwendeten Funktionen übereinstimmen. Das heißt, dass man mit den dort enthaltenen Werten so direkt nicht mehr anfangen kann als mit den Werten aus der Datei `/etc/shadow` unter Unix, denn wenn man sie als Passwort eingeben würde, würde Windows sofort wieder den Hash darauf anwenden, und einen anderen, also falschen Wert daraus errechnen. Das Programm `smbclient` muss diese Operation ebenfalls durchführen, nur hat man hierzu den Quellcode und kann die entsprechenden Stellen auskommentieren. So hat man die Möglichkeit, sich anhand der Werte in der `smbpasswd` ohne Einsatz von crack bei einem NT-Rechner anzumelden.
Alles nicht dramatisch, sagt Microsoft. Das Äquivalent zur Datei `smbpasswd` liegt unter NT verschlüsselt vor. Diese Verschlüsselung muss jedoch reversibel sein, um das Challenge-Response Verfahren durchführen zu können. Ein Teil der Sicherheitsargumentation liegt darin, dass dieses Verschlüsselungsver-

fahren nicht offengelegt wurde. Das Verfahren war solange geheim, bis Jeremy Allison das Programm `pwdump` veröffentlicht hat. Dieses Programm extrahiert aus der Benutzerdatenbank von NT eine Datei, die direkt als `smbpasswd` verwendet werden kann.

Samba als Logon-Server

Wenn sich in einem Netz Windows 95/98 Clients befinden, kann es wünschenswert sein, dass sich die Benutzer dieser Arbeitsplätze nur mit einem Passwort anmelden können, das zentral auf einem Server vorgehalten wird. Dazu muss der entsprechende Server spezielle Aufrufe von Clients entgegennehmen und korrekt beantworten. In der reinen Windowswelt ist dazu ein Windows NT Server notwendig, der als sogenannter Primary Domain Controller (PDC) installiert ist. Samba ist ebenfalls in der Lage, dies zu tun. Dazu ist im Abschnitt `[global]` der Parameter `domain logons = yes` zu setzen. Die Implementation, die Microsoft gewählt hat, um Domänenanmeldungen zu ermöglichen, erzwingt zusätzlich, dass der Domain Master Browser auf dem gleichen Rechner liegt wie der Logon Server. Das heißt, man benötigt für Domänenanmeldungen die folgenden Parameter:

```
[global]
workgroup = samba
domain logons = yes
domain master = yes
```

Hat man diese Parameter gesetzt, kann man in den Eigenschaften des Clients für Microsoft-Netzwerke einstellen, dass der Client sich an der Domäne `samba` anmelden soll. Hat man verschlüsselte Passwörter aktiviert, kann man vom Client aus sein SMB-Passwort ändern, indem man das entsprechende Kontrollfeld in der Systemsteuerung von Windows benutzt.

Samba in Windows NT Domänen

Die Domänenanmeldung unter Windows 95/98 ist eine relativ einfache Sache, da es sich dabei praktisch nur um eine Überprüfung der Benutzerpasswörter handelt. So etwas wie Benutzer kennt Windows 95 praktisch nicht, jeder Benutzer hat vollen Zugriff auf das gesamte System.

Microsoft unterscheidet verschiedene Netzwerkmodelle. Das Peer-To-Peer Netz ist das Modell, das auch Unix zu Grunde

liegt. Hier hat jeder beteiligte Rechner eine eigene Benutzerdatenbank, eigene Passwörter und eigene Rechtezuordnungen. Das Domänenmodell ist das Modell, das sich signifikant von Unix unterscheidet. Mit dem Domänenmodell wird eine Workstation in die Lage versetzt, mehr als eine Benutzerdatenbank zu benutzen. Neben der eigenen Benutzerdatenbank, die jede Workstation hat, kann sie eine Benutzerdatenbank von einem anderen Rechner importieren. In einer Windows NT Domäne gibt es einen Rechner, der seine eigene Benutzerdatenbank anderen zur Verfügung stellt, den sogenannten Primary Domain Controller. Dieser reserviert für sich spezielle NetBIOS-Namen, um sich den Workstations als Logonserver anzubieten. Eine Workstation befragt den Primary Domain Controller nach allen relevanten Daten zu den Benutzern, die sich bei ihr anmelden wollen, und die Rechte auf der Workstation wahrnehmen können.
Es muss jede Workstation explizit in die Domäne aufgenommen werden.
Bei Samba ist es so, dass es zu jedem Benutzer, der ein Passwort in der `/etc/smbpasswd` hat, einen Benutzer im System geben muss. Der zu einer Workstation gehörende Benutzer muss den NetBIOS-Namen der Workstation, ergänzt um ein $-Zeichen, haben. Man benötigt also zwei Schritte, um eine Workstation in die Domäne aufzunehmen. Im ersten Schritt wird der Unixbenutzer angelegt. Dies geschieht in vielen Linuxsystemen mit dem Kommando `useradd -m < user >`. Der angelegte Benutzer benötigt im Unixsystem weder ein Passwort noch ein Heimatverzeichnis. Er ist notwendig, da die Workstation in der Domäne eine eigene SID bekommt, die aus der Unix userid berechnet wird. Dann muss die Workstation ein Passwort in der `/etc/smbpasswd` bekommen, und zwar mit dem Befehl `smbpasswd -a -m <name>`. Ein Beispiel sieht folgendermaßen aus:

```
root@diana: useradd -m wks\$
root@diana: smbpasswd -a -m wks
```

Man beachte, dass beim Befehl `useradd` ein Dollarzeichen, maskiert durch den Backslash, hinzugefügt wurde. Der Befehl `smbpasswd` fügt diesen bei Verwendung des Parameters `-m` selbst hinzu.
Samba als Domänenmitglied

Mit dem Parameter `security` kann man den Zeitpunkt steuern, zu dem das Benutzerpasswort geprüft wird. `security = share` legt fest, dass die Prüfung beim Tree Connect stattfindet, das heißt, wenn die Freigabe angesprochen wird. Ist `security = user` angegeben, wird das Passwort bereits einen Schritt vorher, also beim Session Setup geprüft. Bei `security = user` wird also die Kombination von Benutzer und Passwort geprüft bei `security = share` die Kombination Freigabe und Passwort.
Der Parameter `security` kann noch zwei weitere Werte annehmen: `server` und `domain`. Bei beiden Einstellungen verhält sich Samba gegenüber dem Client genau wie bei `security = user`, der Benutzer muss sich unter seinem Namen beim Server authentifizieren. Die Unterschiede liegen in der Art und Weise, wie das Passwort überprüft wird.

- `security = user` Die Überprüfung findet anhand einer lokalen Datenbank statt. Werden Klartextpasswörter verwendet (`encrypt passwords = no`), so wird die lokale Unix-Passwortdatenbank in `/etc/passwd, /etc/shadow` oder die entsprechende NIS-Tabelle herangezogen. Bei verschlüsselten Passwörtern mit wird die Samba-eigene Passwortdatenbank in der Datei `smbpasswd` zur Überprüfung herangezogen.

- `security = server` Bei dieser Einstellung bekommt der Sambaserver vom Client einen Benutzernamen und ein Passwort präsentiert. Er versucht daraufhin, sich mit diesem Passwort bei einem weiteren Server anzumelden. Funktioniert dies, hat der Benutzer sein Passwort offensichtlich richtig eingegeben. Schlägt dies fehl, wird auch dem Client des Sambaservers der Fehler mitgeteilt und der Zugriff verweigert. Der Passwortserver, der zur Überprüfung herangezogen wird, muss mit seinem NetBIOS-Namen im Parameter `password server` angegeben werden.

- `security = domain` Auch hierbei wird die Überprüfung einem Passwortserver überlassen. Dieser muss jedoch ein Primary Domain Controller sein, der den Sambaserver in die Domäne aufgenommen hat. Der Hauptvorteil gegenüber `security = server` besteht in einer deutlich reduzierten Last auf dem Passwortserver und einer verschlüsselten Kommunikation zwischen Samba und Passwortserver.

Um einen Windowsrechner dazu zu bringen, für einen Sambaserver die Passwortüberprüfung zu Übernehmen, muss man nur `security = server` und den `password server` passend setzen. Dabei übernimmt der Server ausschließlich die Überprüfung der Passwörter. Bei verschlüsselten Passwörtern können Benutzer nur dann in die `smbpasswd` aufgenommen werden, wenn sie in der Unix-Benutzerdatenbank existieren. Genau so verhält es sich bei `security = server`. Benutzer können auf Samba nur dann zugreifen, wenn sie als normale Unixbenutzer existieren.
`security = server` ist nicht die optimale Lösung für die Überprüfung von Passwörtern durch einen weiteren Rechner.
Um die Vorteile der Domänenmitgliedschaft zu nutzen, ist etwas mehr Aufwand notwendig. Mitglied einer Domäne zu sein heißt, mit dem Primary Domain Controller über einen verschlüsselten Kanal kommunizieren zu können. Diese Verschlüsselung wird verwendet, um Benutzerinformationen verdeckt austauschen zu können. Als Verschlüsselungsverfahren kommt ein symmetrisches oder auch secret key Verfahren zum Einsatz. Um ein symmetrisches Verfahren anwenden zu können, müssen sich beide Partner über ein gemeinsames Geheimnis, den secret key einig sein. Ein solches gemeinsames Geheimnis muss regelmäßig geändert werden, um einer großen Klasse von kryptographischen Angriffen auszuweichen. Eine solche Änderung darf selbstverständlich nicht abgehört werden können, da ein Zuhörer damit die gesamte Kommunikation abhören kann. Für die Änderung eines Geheimnisses gab es bereits vor der Implementation des Domänenprotokolls ein fertiges Protokoll, das man direkt verwenden konnte: Die Möglichkeit, Benutzerpasswörter über das Netz zu ändern, war mir einem gesicherten Protokoll implementiert. Um dieses Protokoll zur verschlüsselten Kommunikation zwischen einer Workstation oder einem Mitgliedsserver und dem Domänencontroller nutzen zu können, muss es für jedes Domänenmitglied ein Benutzerkonto geben. Genau dies wird auf dem Domänencontroller erstellt, wenn man eine Workstation oder einen Server mit dem Servermanager in die Domäne aufnimmt. Betritt man danach mit der Workstation die Domäne, wird als erstes das Passwort des Computerkontos geändert.
Um einen Sambaserver in eine Domäne aufzunehmen, sind zwei Schritte notwendig.

- Auf dem Server muss der Sambaserver mit seinem NetBIOS-

Namen in die Domäne aufgenommen werden.

- Der Sambaserver selbst muss darüber informiert werden, dass er sich in der Domäne befindet, und er muss sein Passwort ändern. Dies geschieht mit dem Befehl

  ```
  smbpasswd -j DOM -r PDC
  ```

  Dabei steht DOM für die Domäne, die betreten wird. Mit PDC wird der NetBIOS-Name des Domänencontrollers der Domäne benannt.

Mit diesem Kommando wird das Maschinenpasswort auf dem PDC auf einen neuen, zufälligen Wert geändert. Dieses neue Maschinenpasswort für den Samba Server wird in einer Datei im gleichen Verzeichnis wie die Datei `smbpasswd` abgespeichert und hat folgenden Namen:
`<NT DOMAENENNAME>.<Samba Servername>.mac`
Die Endung .mac steht für *Machine ACcount* Passwortdatei. Im obigen Beispiel würde die Datei also DOM.SERV1.mac heißen. Diese Datei wird von root erstellt und ist für keinen anderen Benutzer lesbar. Sie ist der Schlüssel zu Ihrer Domänensicherheit und sollte genau so vorsichtig behandelt werden wie die Datei `/etc/shadow`.
Nach diesen beiden Schritten kann man mit `security = domain`, `password server = PDC BDC1 BDC2` und `encrypt passwords = yes` die Passwortüberprüfung an einen der Domänencontroller delegieren. Dies sind die Primären und Backup Domänencontroller, die Samba der Reihe nach kontaktieren wird, um Benutzer zu authentifizieren. Samba wird sie in der aufgeführten Reihenfolge ansprechen. Sie können also die Reihenfolge verändern, um eine günstigere Lastverteilung zu erreichen. Eine weitere Option ist die Angabe `password server = *`. Damit sucht Samba mit den Standardmethoden von Windows NT nach einem Domänencontroller und befragt die Server, die es bei dieser Anfrage herausbekommen hat.
Warum ist `security = domain` besser als `security = server`? Der Vorteil der Domänensicherheit ist, dass Samba die Authentifizierung über einen gesicherten RPC Kanal schickt, genau wie ein Windows NT Server es tun würde. Das heißt, dass Samba nun genau wie ein Windows NT Server an einer Vertrauensstellung teilnehmen kann. Das heißt, Sie können einen Samba Server in eine Resourcendomäne aufnehmen, und Sie können

die Authentifizierung via Resourcen PDC vom PDC der Benutzerdomäne vornehmen lassen.
Zusätzlich muss in der Einstellung `security = server` der Samba Daemon eine Verbindung zum Authentifizierungsserver während seiner gesamten Laufzeit offenhalten. Dies kann die Anzahl der offenen Verbindungen auf einem Windows NT Server in die Höhe treiben, so dass dieser keine Verbindungen mehr annimmt. Mit `security = domain` verbinden sich die Samba Daemonen nur so lange mit dem PDC, wie es für die Benutzerauthentifizierung notwendig ist. Danach wird die Verbindung wieder abgebaut, so dass die Verbindungen wieder anderweitig verwendbar sind.
Und nicht zuletzt bekommt der Samba Server als Teil der Antwort auf die Authentifizierungsanforderung Informationen über den Security Identifier, die Gruppenzuordnungen und andere Informationen über den Benutzer. Alle diese Informationen werden Samba zukünftig erlauben, in einem sogenannten Appliance Modus zu laufen. In diesem Modus wird kein manuell angelegter Unixbenutzer mehr notwendig sein. Samba wird Unix Benutzer und Gruppen aus der Authentifizierungsantwort des PDC erzeugen. Damit wird Samba wirklich ein Plug and Play Mitglied einer Domäne.
Dieser Appliance Modus kann heute schon annähernd erreicht werden, indem bei Samba der Parameter `add user script` angegeben wird. In diesem Parameter wird ein Unixprogramm angegeben, das dynamisch einen Unixbenutzer erzeugen muss, nachdem ein Passwortserver die Korrektheit eines Passworts bestätigt hat. Ein Beispiel kann sein:

```
add user script = /usr/bin/useradd -m %U
```

Damit wird einfach ein Benutzer hinzugefügt, wenn er noch nicht existiert, aber der PDC das Passwort bestätigt hat.