Study-Guide: Sicherheit

Die letzten drei Kapitel der LPI102 Vorbereitung drehen sich um Fragen der Sicherheit. Ausführung von sicherheitsadministrativen Tätigkeiten Einrichten von Host-Security Einrichten von Sicherheit auf Benutzerebe

Seite:	-= LinuxLernSystem =- (http://www.lpi-test.de)
Kurs:	LPIC-1 [102]
Buch:	Study-Guide: Sicherheit
Gedruckt von:	André Scholz
Datum:	Dienstag, 1 November 2005, 10:52 Uhr

Inhaltsverzeichnis

- 1.114 - Sicherheit
  - 1.114.1 - Ausführung von sicherheitsadministrativen Tätigkeiten
  - 1.114.2 - Einrichten von Host-Security
  - 1.114.3 - Einrichten von Sicherheit auf Benutzerebene

1.114 - Sicherheit

Die letzten drei Kapitel der LPI102 Vorbereitung drehen sich um Fragen der Sicherheit.

- Ausführung von sicherheitsadministrativen Tätigkeiten
- Einrichten von Host-Security
- Einrichten von Sicherheit auf Benutzerebe

1.114.1 - Ausführung von sicherheitsadministrativen Tätigkeiten

Beschreibung: Prüfungskandidaten sollten in der Lage sein, die Systemkonfiguration zu überprüfen, um die Sicherheit des Hosts in Übereinstimmung mit lokalen Sicherheitsrichtlinien sicherzustellen. Dieses Lernziel beinhaltet die Konfiguration von tcpwrappers, das Finden von Dateien mit gesetztem SUID/SGID-Bit, das Überprüfen von Softwarepaketen, das Setzen oder Ändern von Benutzerkennwörtern und des Ablaufs von Kennwörtern, das Aktualisieren von Programmdateien nach Empfehlung von CERT, Bugtraq und/oder Sicherheitswarnungen des Distributors. Ebenfalls enthalten ist ein grundsätzliches Wissen über **ipchains** und **iptables.**

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/proc/net/ip_fwchains`
- `/proc/net/ip_fwnames`
- `/proc/net/ip_masquerade`
- **find**
- **ipchains**
- **passwd**
- **socket**
- **iptables**

Bereits besprochene Techniken

Viele der hier verlangten Techniken sind an anderer Stelle dieses Study-Guides schon besprochen worden. Dazu zählen insbesondere

- Verwaltung der tcpwraper.
- Setzen und Ändern von Passwörtern und Auslauffristen von Passwörtern (Abschnitt 1.111.1)
- Überprüfung von Softwarepaketen (Abschnitt 1.102.6 der Vorbereitung auf die LPI101 Prüfung)

Diese Techniken werden hier nicht näher beschrieben, sondern als Wissen vorausgesetzt.

Auffinden von Programmen mit gesetztem SUID/SGID Bit

Programme, deren SUID-Bit oder SGID-Bit gesetzt sind, und die gleichzeitig als Eigentümer `root` haben, sind in dem Moment eine Sicherheitslücke, in dem es sich um Programme handelt, die einen Zugriff auf eine Shell ermöglichen. Viele Programme benützen dieses Feature um ganz gewöhnliche Aktionen vorzunehmen. Das klassische Beispiel ist das Programm **passwd**. Jeder normale User kann mit diesem Programm sein Passwort ändern. Die Datei, in der das verschlüsselte Passwort gespeichert wird (`/etc/shadow`) ist aber nicht für jeden User beschreibbar. So muß das Programm **passwd** das SUID-Bit gesetzt haben. Nur so kann es - unabhängig davon, wer es aufgerufen hat - die Datei verändern.

Um das ganze System nach Dateien zu durchsuchen, die dieses Bit (oder eines dieser Bits) gesetzt haben, benutzen wir den Befehl find. Mit der Suchoption `-perm` ist es möglich, Dateien zu suchen,

die bestimmte Rechte gesetzt haben. Ein Zugriffsrecht von 4XXX bedeutet, daß das SUID-Bit gesetzt ist, ein 2XXX gibt an, daß das SGID-Bit gesetzt ist. Die Summe davon (6XXX) wären beide Bits gesetzt. Wenn dem angegebenen Zugriffsrecht ein Minuszeichen vorangeht, so ist gemeint, daß mindestens diese Rechte gesetzt sein müssen, es reicht also, nach 4111 oder gar 4000 zu suchen, wenn wir alle Programme mit SUID-Bit suchen.

```
find / -perm -4000 -user root
```

sucht alle Dateien des gesamten Systems, die das SUID-Bit gesetzt haben und dem User root gehören. Das sind zunächst einmal überraschend viele Programme, es wäre also ratsam, die Ausgabe dieses Befehls direkt nach der Installation in eine Datei umzuleiten und regelmäßig den Befehl erneut aufzurufen und mit dieser Datei zu vergleichen.

Das entsprechende zum SGID-Bit wäre der Befehl

```
find / -perm -2000 -group root
```

Die Angabe des Users wurde hier weggelassen, stattdessen suchen wir nach Dateien, die der Gruppe root zugehören. Ohne diese Angabe, würden alle Dateien mit gesetztem SGID-Bit angezeigt.

Grundlegendes zu Firewaltechniken

Der Begriff Firewall überspannt ein weites Feld an Missinterpretationen und falschen Vorstellungen. Es existieren verschiedenste Ansätze, die Sicherheit eines Netzes durch verschiedene Techniken zu verbessern, die alle diesen Namen für sich reklamieren. Die gesamte Thematik umfassend darzustellen würde den Rahmen dieses Kurses bei weitem sprengen, hier geht es nur um das grundlegende Verständnis von Paketfilterfirewalls unter Linux.

Die Firewalltechnik unter Linux basiert auf sogenannten Paketfilterregeln. Ein- und ausgehende Pakete können nach folgenden Kriterien durchgelassen oder abgewiesen werden:

- Sender IP-Adresse
- Empfänger IP-Adresse
- Sender-Port
- Empfänger-Port
- SYN und ACK-Flag
- Verwendete Netzwerkschnittstelle

Die Regeln werden mit einem kommandozeilenorientierten Programm formuliert und stehen dann im Kernelspeicher. Die eigentliche Firewall, also das Programm, das Pakete durchlässt oder abweist, ist kein separates Programm sondern der Kernel selbst.

Dieses Programm, zur Formulierung der Regeln hat zwischen der Kernel-Version 2.2 und 2.4 gewechselt. Das Standard-Programm bei Kerneln der 2.2 Reihe war **ipchains**. Es funktioniert unter gewissen Umständen auch noch bei neueren Kerneln. Seit der Version 2.4 existiert stattdessen das Programm **iptables**, das die selbe Aufgabe hat, sie aber etwas anders löst. Eine Grundkenntnis beider Programme wird für die LPI102 Prüfung vorausgesetzt.

Bevor jedoch die einzelnen Programme besprochen werden, noch etwas Theorie, die für das Verständnis beider Programme unabdingbar ist.

Prinzipielles zur TCP/IP Datenübertragung

Um eine Paketfilter-Firewall selbst aufzubauen sind fundierte Kenntnisse der TCP/IP Datenübertragung notwendig. Denn wir müssen ja verschiedenste Regeln formulieren, die sich auf die Datenpakete und deren Protokollheader beziehen. So soll hier nochmal kurz das wichtigste zum Thema Datenübertragung im IP-Netz dargestellt werden.

Der prinzipielle Ablauf einer Datenübertragung

Grundsätzlich müssen wir, angesichts des TCP/IP Schichtenmodells, unterscheiden, was eigentlich wie übertragen werden soll. Uns stehen im Prinzip drei Protokolle zur Verfügung, die vorkommen können:

ICMP
: Das Internet Control Message Protocol arbeitet noch auf der Vermittlungsschicht und dient der Übertragung von Kontrollnachrichten von Rechner zu Rechner. Es benützt selbst keine Ports, aber es werden verschiedene ICMP Nachrichtentypen unterschieden, die in Firewallregeln wie Ports behandelt werden. Ein Beispiel ist Ping, dessen ausgehendes Format vom Typ 8 ist, während das Antwortpaket (pong) den Typ 0 hat.

TCP
: Das Transmission Control Protocol ist das verbindungsorientierte Protokoll auf der Transportschicht. Es benutzt Portnummern, um die jeweiligen Anwendungen auf der obersten Schicht zu adressieren. TCP benutzt einen dreiteiligen Handshake, dessen Ablauf für die Firewallregeln noch Bedeutung haben wird.

UDP
: Das User Datagram Protocol ist das verbindungslose Protokoll auf der Transportschicht. Auch dieses Protokoll benutzt Portnummern, es hat jedoch keinerlei Handshake, also auch keine Flags, die wir bei der Formulierung der Regeln nutzen könnten.

Eine Datenübertragung besitzt - aus der Sicht der Firewall - zwei grundsätzlich unterschiedliche Abläufe. Entweder unser Rechner schickt eine Anfrage an einen Server eines anderen Rechners und bekommt Antwort von ihm, oder ein fremder Client schickt eine Anfrage an einen unserer Server und bekommt Antwort von uns.

Im Prinzip laufen alle Übertragungen nach dem gleichen Muster ab. Ein Client fängt mit einer Anfrage an einen Server an und der antwortet ihm. Dabei spielen die Portnummern eine große Rolle. Um einen Dienst auf einem anderen Rechner anzusprechen, muß der Client also die IP-Adresse des Servers und die Portnummer des gewünschten Dienstes kennen. Damit der Server ihm auch antworten kann, muß der Client auch seine eigene IP-Adresse angeben und eine Portnummer, auf der er selbst lauscht. Da der Client keine fest zugewiesene Portnummer besitzt, wählt er sich einfach eine beliebige Nummer aus dem Pool der unprivilegierten Portnummern (1024 bis 65535) aus.

Wir können also davon ausgehen, daß ein Paket mit einer Anfrage des Clients an den Server folgendes an Header-Information beinhaltet:

- Die IP-Adresse des Servers
- Die bekannte Portnummer des gewünschten Dienstes auf dem Server
- Die IP-Adresse des Clients
- Eine beliebige Portnummer zwischen 1023 und 65535, die sich der Client selbst gewählt hat.

Eine Ausnahme dieses Prinzips haben wir bei ICMP, dort hat der Sender selbst keinen Port bzw.

keinen Pakettyp.

Handshake bei TCP

Bei der Verwendung von TCP als Protokoll gibt es eine weitere Besonderheit. Hier werden noch sogenannte Flags gesetzt, also Optionen, die besondere Steuerfunktionen haben. Eine Verbindung von Client zu Server beginnt bei TCP immer folgendermaßen:

Der Client schickt ein Paket mit folgender Header-Information an den Server:

Empfänger IP:	**Empfänger Port:**	**Sender IP:**	**Sender Port:**	**Flags:**
IP-Adresse des Servers z.B. 123.45.67.89	*Portnummer des gewünschten Dienstes z.B. 80*	*Eigene IP-Adresse des Clients z.B. 111.22.33.44*	*Zufällige Portnummer z.B. 12345*	SYN

Der Client setzt also das SYN-Flag, gleichbedeutend mit der Nachfrage nach einem Verbindungsaufbau (Synchronisation). Der Server schickt jetzt ein Paket zurück, das folgende Information beinhaltet:

Empfänger IP:	**Empfänger Port:**	**Sender IP:**	**Sender Port:**	**Flags:**
IP-Adresse des Clients-111.22.33.44	*Portnummer des Clients-12345*	*IP-Adresse des Servers-123.45.67.89*	*Portnummer des Server-Dienstes-80*	SYN und ACK

Der Server setzt also einerseits das ACK (Acknowledge) Flag um zu zeigen, daß er den Verbindungsaufbau gewährt und setzt zusätzlich nochmal das SYN-Flag um seinerseits die Verbindung anzufordern. Das nächste Paket des Clients enthält jetzt nur noch das ACK-Flag:

Empfänger IP:	**Empfänger Port:**	**Sender IP:**	**Sender Port:**	**Flags:**
IP-Adresse des Servers-123.45.67.89	*Portnummer des Server-Dienstes-80*	*IP-Adresse des Clients-111.22.33.44*	*Portnummer des Clients-12345*	ACK

Von diesem Moment an gilt die Verbindung als etabliert und alle weiteren Pakete haben jetzt grundsätzlich das ACK-Flag und niemals mehr das SYN-Flag gesetzt. Aus dieser Handshake-Methode heraus ist anhand der Flags grundsätzlich festzustellen, ob ein Paket eine Nachfrage eines Clients an einen Server darstellt oder nicht. Zusammengefasst kann man sagen, daß ein Paket immer dann eine Client-Nachfrage beinhaltet, wenn das SYN-Flag gesetzt und das ACK-Flag nicht gesetzt ist.

Diese fünf (TCP) bzw. vier (UDP) Kriterien stehen uns also für die Firewall zur Verfügung. Anhand dieser Eigenschaften müssen wir entscheiden, ob wir ein Paket durchlassen oder nicht.

Architektur der Firewall-Regelketten

Eine Linux-Paketfilter Firewall besteht aus sogenannten Regelketten (chains). Standardmäßig existieren drei solcher Ketten:

- Die input-chain, die alle Pakete betrifft, die den Rechner verlassen
- Die output-chain, die alle Pakete betrifft, die der Rechner empfängt

- Die forward-chain, die alle Pakete betrifft, die der Rechner routen soll.

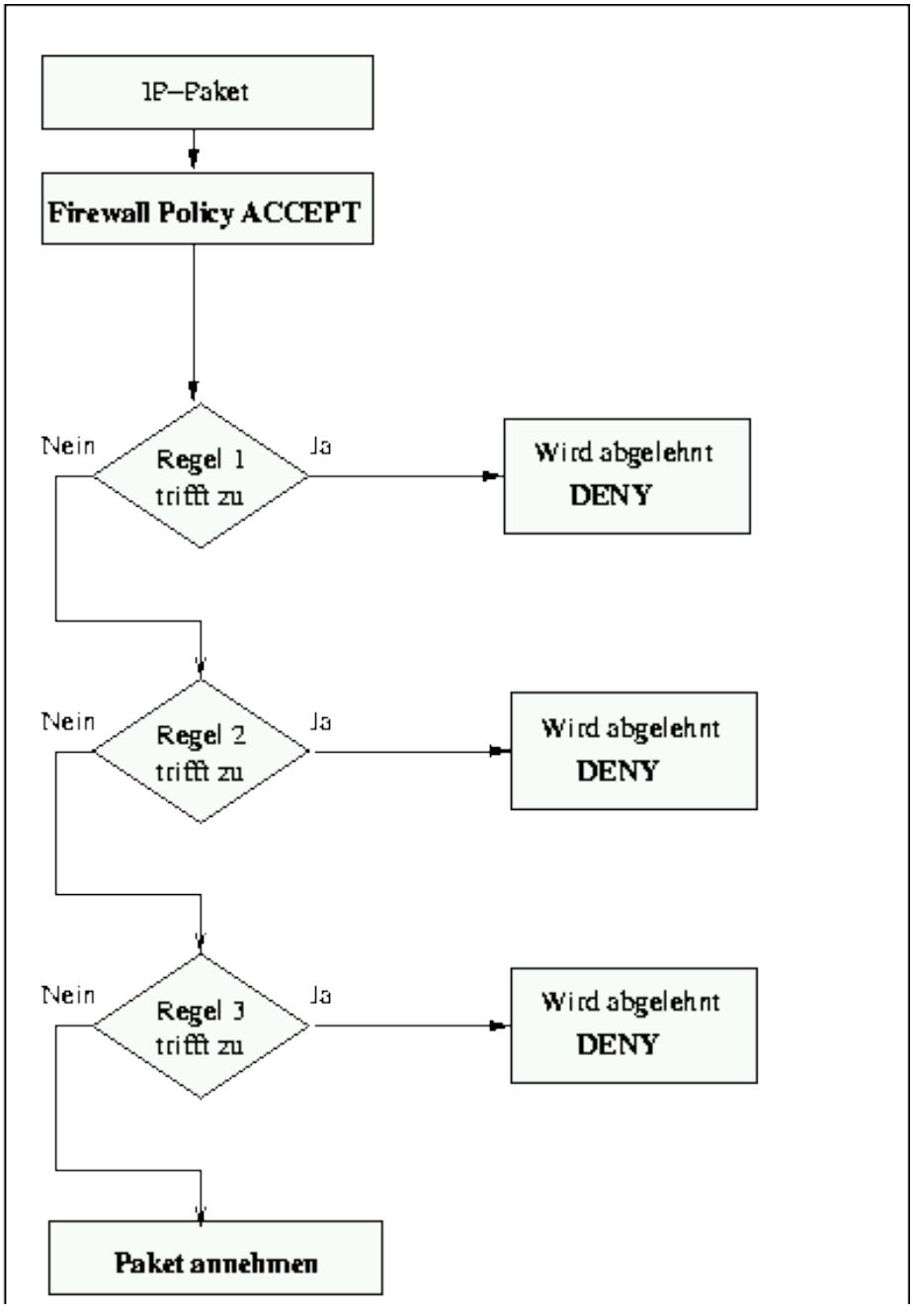
Alle drei Regelketten können unterschiedliche Grundeinstellungen haben, gemeinsam ist ihnen, daß sie eine Sammlung (Verkettung) von Regeln besitzen, die nacheinander abgearbeitet werden, solange bis eine Regel zutrifft oder das Ende der Kette erreicht wurde.

Es stehen drei Grundeinstellungen zur Verfügung:

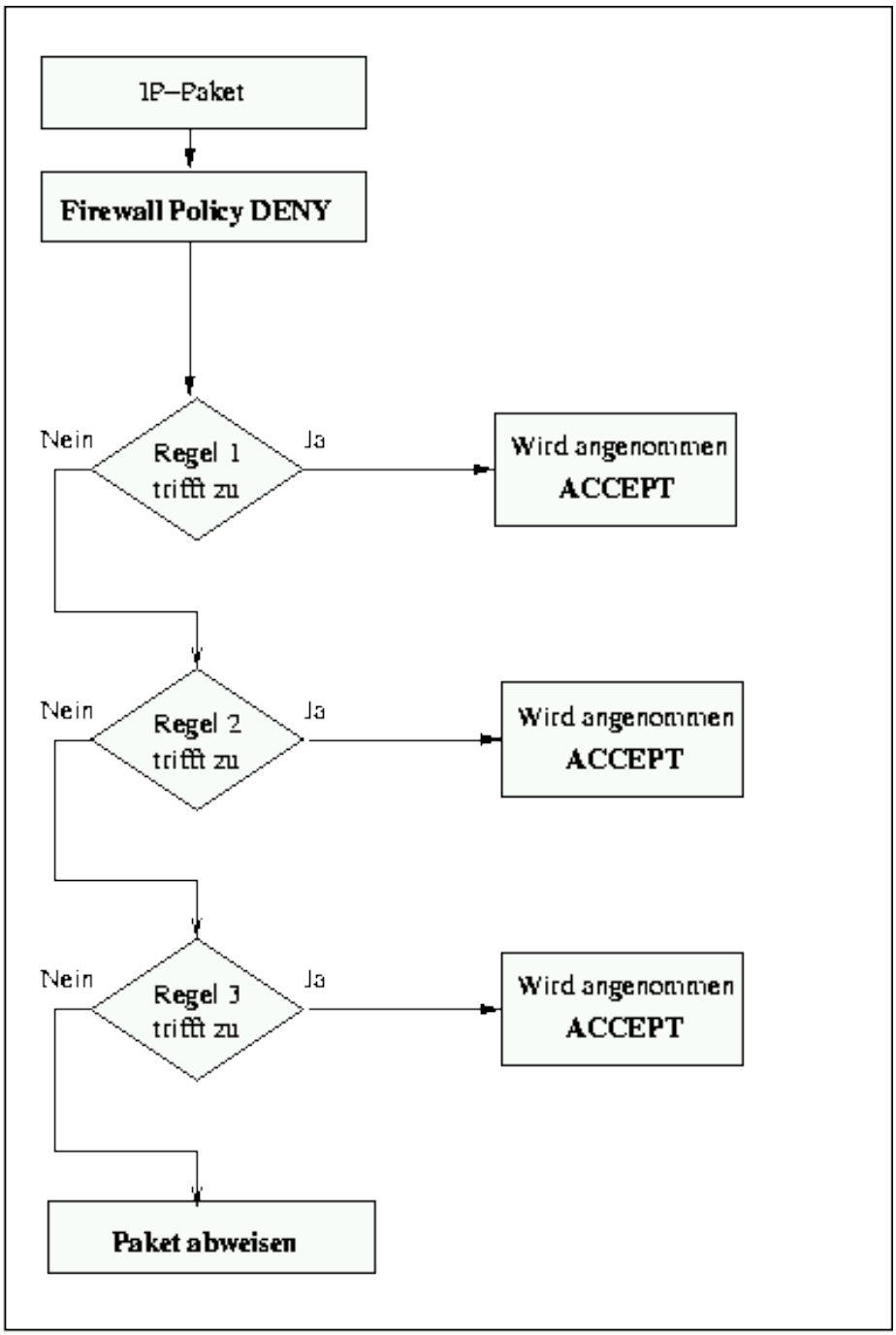
- ACCEPT - Ein Paket wird akzeptiert
- DENY - Ein Paket wird wortlos abgewiesen
- REJECT - Ein Paket wird mit Fehlermeldung abgewiesen.

Hat eine Kette die Grundeinstellung (Policy) ACCEPT, so bedeutet das, das grundsätzlich alles erlaubt ist, was nicht explizit verboten ist. Hat sie aber DENY oder REJECT als Grundeinstellung, so ist alles verboten, was nicht explizit erlaubt wurde.

Der zweite Fall ist mit Sicherheit die bessere Wahl, wenn es darum geht, ein System richtig abzusichern, auch wenn es zunächstmal mehr Arbeit bedeutet. Schematisch dargestellt könnte man die Grundeinstellung ACCEPT folgendermaßen darstellen:



Die DENY-Grundeinstellung hingegen sähe so aus:



Grundlagen von ipchains

Das Programm, mit dem die Linux-Paketfilterfirewall seine Regeln definiert, heisst bis zur Kernelversion 2.2 **ipchains**. Dieses Programm hat verschiedene Aufrufformen, die alle zu besprechen den Rahmen dieser Darstellung sprengen würde. Ich werde mich hier auf die notwendigen Formen beschränken, die den Umgang mit dem Programm klären.

ipchains formuliert Regeln, die dann direkt in den Speicher geschrieben werden. Das hat zur Folge, daß diese Regeln nach einem Neustart allesamt wieder verschwunden sind. Es empfielt sich also, die Regeln in Form eines Shellscripts zu formulieren.

Grundlegende Parameter von ipchains

Einer der folgenden Parameter muß immer der erste von ipchains sein. Diese Parameter schließen sich aus, es darf in einem Aufruf also immer nur einer der folgenden Parameter auftauchen:

-A *Regelkette ...*
: Eine Regel wird ans Ende der angegebenen Regelkette (input, output oder forward) angehängt. Wird keine Regelkette angegeben, so gilt die Regel für alle Regelketten.

-I *Regelkette ...*
: Eine Regel wird an den Anfang der angegebenen Regelkette (input, output oder forward) eingefügt. Wird keine Regelkette angegeben, so gilt die Regel für alle Regelketten.

-F *Regelkette*
: Alle bestehenden Regeln der angegebenen Kette werden gelöscht. Die Grundeinstellung (Policy) bleibt jedoch erhalten. Wird keine Regelkette angegeben, so werden alle Regeln aller Regelketten gelöscht.

-P *Regelkette Policy*
: Die angegebene Policy (DENY, REJECT oder ACCEPT) wird zur Grundeinstellung der genannten Regelkette.

Um Regeln zu definieren werden wir also mit ipchains -A arbeiten. Der Angabe der Kette folgen dann die entsprechenden Regelformulierungen.

Darstellungsart von Ports und Adressen

Wir werden bei jeder zu formulierenden Regel sowohl mit Portnummern, als auch mit IP-Adressen arbeiten müssen. Dabei stehen uns folgende Darstellungsmöglichkeiten zur Verfügung:

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

```
192.168.100.123/24
```

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

Ports werden als Nummern angegeben. Soll ein ganzer Bereich von gültigen Portnummern angegeben werden, so wird das in der Form

Startport:Endport

dargestellt.

Parameter zur Darstellung der Regeln

Nach der Angabe des Parameters -A (oder -I) und der gewünschten Kette folgen verschiedene Angaben, die alle erfüllt sein müssen, wenn die Regel zutreffen soll. Die wichtigsten Parameter sind:

`-i` *Interface*
: Die Netzwerkschnittstelle für die die Regel gilt. Hier werden die symbolischen Schnittstellennamen wie `eth0, eth1, ppp0, ippp0,..` eingesetzt.

`-p` *Protokoll*
: Hier wird das vom Paket verwendete Protokoll angegeben, also `icmp, tcp` oder `udp`.

`-y`
: Das SYN-Flag einer TCP-Nachricht muß gesetzt, das ACK-Flag darf nicht gesetzt sein. Das Paket ist also das erste eines Verbindungsaufbaues und kommt vom Client.

`! -y`
: Das ACK-Flag einer TCP-Nachricht muß gesetzt sein. Das heißt, das Paket ist entweder der zweite Teil des Verbindungsaufbaues oder, es ist ein Teil einer bestehenden Verbindung. Ist weder -y noch ! -y gesetzt, werden die TCP-Flags nicht überprüft.

`-s` *IP-Adresse* [*Portnummer*]
: Absenderadresse (Source) des Paketes und optional die Absenderprtnummer.

`-d` *IP-Adresse* [*Portnummer*]
: Empfängeradresse (Destination) des Paketes und optional der Empfängerport.

`-j` *Policy*
: Policy dieser Regel. Gültige Policies sind ACCEPT, REJECT und DENY. In der forward-Chain ist auch die Policy MASQ für Masquerading zulässig.

Formulierung von Regeln

Für jede Art von Datenverbindung müssen wir mindestens zwei Regeln formulieren, eine für die input-chain und eine für die output-chain. In der Regel werden diese Formulierungen sich immer an den folgenden Aufbau halten:

```
ipchains -A Regelkette -i Interface -p Protokoll \
         -s Absenderadresse Absenderport \
         -d Empfängeradresse Empfängerport -j Policy
```

In manchen Fällen wird dem Protokoll noch die Angabe `! -y` folgen oder einzelne Ports werden weggelassen. Aber der grundsätzliche Aufbau hält sich immer an diese Struktur.

Aufbau eines eigenen Firewall-Scripts

Um ein komplettes Firewallscript zu schreiben fehlt uns hier der benötigte Rahmen, wir können aber die Regeln für die wichtigsten Dienste hier formulieren und alles weitere kann dann - sofern erwünscht - hinzugefügt werden.

Wir erstellen die Regeln am Besten in einem Shellscript, das wir dann jedesmal einfach aufrufen können, dann stellt sich gar nicht erst die Frage, wie einzelne Regeln gespeichert werden.

Das Script beginnt üblicherweise mit einem ganzen Satz von Variablendefinitionen, damit wir uns später leichter mit der Formulierung der Regeln tun:

```
#!/bin/bash

EXTERN_INTERFACE=eth0    # Unsere Netzschnittstelle
LOOP_INTERFACE=lo        # Das Loopback Interface
IPADDR=10.230.1.100      # Unsere IP-Adresse
ANYWHERE=any/0           # Jede Adresse im Netz
```

```
MYNET=10.230.1.0/24      # Unsere Netzadresse
UNPRIVPORTS=1024:65535   # Die unprivilegierten Ports
```

Als nächstes löschen wir alle bestehenden Regeln. Falls wir das Script einmal mehrfach hintereinander starten kommt es so nicht zu Doppelregeln. Dadurch, daß wir keine Regelkette angeben, werden alle bestehenden Regelketten gelöscht.

```
ipchains -F
```

Alle Ketten sind jetzt leer. Allerdings haben wir noch nicht die Grund-Policies gelöscht bzw. verändert. Sie bleiben auch nach dem Löschen vorhanden. Also stellen wir jetzt diese Policies ein, für jede Kette extra. Dazu stellt **ipchains** den Parameter -P (nicht verwechseln mit -p) zur Verfügung. Wir setzen alle drei Regelketten auf die Grundeinstellung DENY, also ist alles verboten, was nicht explizit erlaubt ist.

```
ipchains -P input    DENY
ipchains -P output   DENY
ipchains -P forward  DENY
```

Ab hier ist jetzt also alles verboten. Es existieren keinerlei Regeln mehr, die Grundeinstellung ist DENY. Wir können uns ab diesem Moment nicht einmal mehr selbst anpingen.

Damit das Loopback-Interface wieder funktioniert erlauben wir in den nächsten beiden Regeln grundsätzlich alles auf diesem Interface.

```
ipchains -A input  -i $LOOP_INTERFACE -j ACCEPT
ipchains -A output -i $LOOP_INTERFACE -j ACCEPT
```

Jetzt erlauben wir alle ICMP-Pakete auf der Netzschnittstelle. Wir könnten zwar hier verschiedene Einschränkungen machen, aber das ist langwierig und kompliziert. Viele Programme benötigen ICMP-Nachrichten um z.B. eine Nichtverfügbarkeit oder ähnliche Fehlermeldungen zu übertragen.

```
ipchains -A input -i $EXTERN_INTERFACE -p icmp -j ACCEPT
ipchains -A output -i $EXTERN_INTERFACE -p icmp -j ACCEPT
```

Jetzt funktioniert wenigstens ein Ping schonmal wieder. Allerdings nur mit numerischen IP-Adressen. Denn wir haben noch keinerlei Zugriff auf Nameserver. DNS funktioniert in den meisten Fällen über UDP auf Port 53. Wir können jetzt die erste vollständige Regel anwenden, indem wir den Zugriff auf Nameserver freischalten. Da uns über Port 53 wenig Gefahr droht geben wir ihn für alle frei, nicht nur für einen bestimmten Nameserver. Falls das nicht gewünscht wöre, müsste der Eintrag $ANYWHERE durch die IP-Adresse des Nameservers ersetzt werden.

```
ipchains -A output -i $EXTERN_INTERFACE -p udp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 53 -j ACCEPT

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

Jetzt können wir anfangen, die wichtigen Dienste einzeln zu erlauben. Zunächst einmal HTTP (TCP Port 80) und HTTP mit SSL (TCP Port 443) Wir erlauben nur Antworten eines Servers an uns.

```
ipchains -A output -i $EXTERN_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 80 -j ACCEPT

ipchains -A input -i $EXTERN_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 80 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

ipchains -A output -i $EXTERN_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS \
         -d $ANYWHERE 443 -j ACCEPT

ipchains -A input -i $EXTERN_INTERFACE -p tcp ! -y\
         -s $ANYWHERE 443 \
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

Sind wir selbst auch Webserver, dann müssen wir auch Zugriffe fremder Clients auf unseren Server zulassen und unsere Antwortpakete an diese Clients. Das könnte mit der Einschränkung passieren, daß wir nur Clients aus dem lokalen Netz ($MYNET) akzeptieren.

```
ipchains -A input -i $EXTERN_INTERFACE -p tcp\
         -s $MYNET $UNPRIVPORTS \
         -d $IPADDR 80 -j ACCEPT

ipchains -A output -i $EXTERN_INTERFACE -p tcp ! -y\
         -s $IPADDR 80 \
         -d $MYNET $UNPRIVPORTS -j ACCEPT
```

Schwieriger wird es mit FTP. Hier existieren zwei Portnummern (20 und 21), eine für den FTP-Befehlskanal und eine für den Datenkanal. Zudem existieren zwei Modi, der sogenannte aktive Modus und der passive Modus, der von den meisten Browsern verwendet wird. Der Unterschied liegt darin, daß der passive Modus beim Datenkanalaufbau auf beiden Ports (Sender und Empfänger) unprivilegierte Portnummern verwendet. Wir brauchen also 6 Regeln um als Client an einen FTP Server zu kommen:

- Anfrage des lokalen Clients beim fremden Server
- Antwort des fremden Servers
- Datenkanalaufbau durch den fremden Server (aktiv)
- Antwort auf Datenkanalaufbau durch den lokalen Client (aktiv)
- Datenkanalaufbau des Clients zum fremden Server (passiv)
- Antwort des fremden Servers auf Datenkanalaufbau (passiv)

In der oben genannten Reihenfolge sieht das also folgendermaßen aus:

```
ipchains -A output -i $EXTERN_INTERFACE -p tcp \
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE 21 -j ACCEPT

ipchains -A input -i $EXTERN_INTERFACE -p tcp ! -y \
         -s $ANYWHERE 21\
         -d $IPADDR $UNPRIVPORTS -j ACCEPT
```

```
ipchains -A input -i $EXTERN_INTERFACE -p tcp\
         -s $ANYWHERE 20\
         -d $IPADDR $UNPRIVPORTS -j ACCEPT

ipchains -A output -i $EXTERN_INTERFACE -p tcp ! -y\
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE 20 -j ACCEPT

ipchains -A output -i $EXTERN_INTERFACE -p tcp\
         -s $IPADDR $UNPRIVPORTS\
         -d $ANYWHERE $UNPRIVPORTS -j ACCEPT

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

Das Gleiche in umgekehrter Reihenfolge wäre dann für einen eigenen FTP-Server nötig.

Dabei belassen wir es einmal, die Grundzüge des Aufbaus der Firewallscripts sind damit eigentlich geklärt. Jedes verwendete Protokoll benötigt also einen Regelsatz, der sowohl die eingehenden, als auch die ausgehenden Pakete berücksichtigt.

Unsere Firewall schützt bisher natürlich nur uns selbst, Aber die Mechanismen sind natürlich auch bei einer Firewall mit zwei Netzwerkkarten im Grunde die selben.

Grundlagen von iptables

Der grundsätzliche Aufbau von **iptables** ist sehr ähnlich dem von **ipchains**. Nur werden die Regelketten jetzt in Tabellen verwaltet, die jeweils einzelne oder mehrere Regelketten enthalten können. Das sollte zu einer größeren Übersichtlichkeit führen.

Drei Standard-Tabellen stehen zur Verfügung:

nat
: Diese Tabelle ist zuständig für das IP-Masquerading oder Network Address Translation (NAT). Datenpakete, die durch einen Rechner geroutet werden, passieren diese Tabelle nur einmal. Die Regeln dieser Tabelle werden nur auf das erste Paket eines Datenstroms angewandt. Wenn dieses Paket passieren darf, so werden alle weiteren Pakete des gleichen Datenstroms automatisch maskeriert, ohne daß die Regeln jedesmal überprüft werden. Die Tabelle enthält drei Regelketten (chains), `PREROUTING, OUTPUT` und `POSTROUTING`.

mangle
: Diese Tabelle ist dazu gedacht, einzelne Elemente des Headers zu verändern. Beispiele wären die Veränderung der Felder `TTL, TOS` oder `MARK`. Sie hat zwei Regelketten, `PREROUTING` und `OUTPUT`. Die erstere wird verwendet, um Pakete zu verändern, in dem Moment, in dem sie die Firewall erreichen, die zweite verändert Pakete, die innerhalb der Firewall erzeugt werden, bevor die Routing-Entscheidungen anstehen.

filter
: Diese Tabelle entspricht im Wesentlichen der Aufgabenstellung des alten **ipchains**. Hier werden die Firewall-Regeln erstellt. Die drei bekannten Regelketten `INPUT, FORWARD` und `OUTPUT` sind hier eingebaut.

Der Befehl **iptables** erhält die Information, welche Tabelle er bearbeiten soll durch den Parameter `-t Tabelle`. Wird diese Angabe weggelassen, so wird standardmäßig die Tabelle **filter** angenommen.

Die eigentliche Formulierung der Regeln selbst entspricht ziemlich genau der von **ipchains**.

Sicherheitswarnungen

In der Welt der Informatik gibt es zwei wesentliche Philosophien, wie mit Sicherheitslücken in Programmen umgegangen werden soll. Die eine geht davon aus, daß Sicherheitslücken am Besten geheim bleiben sollen und stillschweigend - etwa mit Hilfe eines Service-Packs - beseitigt werden sollten. Die typische Vertreterin dieser These ist die Firma Microsoft. Die zweite Philosophie geht genau andersherum vor. Ihre These ist, daß Sicherheitslücken sofort öffentlich gemacht werden müssen, um es den entsprechenden Systemverwaltern sofort zu ermöglichen, Maßnahmen zu ergreifen, die die Löcher schließen können. Linux-Software wird in der Regel mit dieser zweiten Philosophie arbeiten.

Es gibt zwei große Einrichtungen, die zentral solche Sicherheitslücken aufdecken und veröffentlichen: BUGTRAQ und CERT.

BUGTRAQ ist eine moderierte Mailingliste, die für die detailierte Diskussion und Veröffentlichung von Sicherheitsproblemen zuständig ist. Hier wird beschrieben, was Sicherheitslücken genau sind, wie herausgefunden wird, ob sie ein bestimmtes System betreffen und wie sie beseitigt werden können.

Um sich an dieser Mailingliste zu beteiligen - lesend oder schreibend - muß eine Mail an `LISTSERV@SECURITYFOCUS.COM` geschickt werden, die im Mailkörper - nicht in der Überschrift - die folgende Anweisung besitzt:

```
SUBSCRIBE BUGTRAQ Nachname, Vorname
```

Man erhält anschließend eine Bestätigungsmail, die man beantworten muß. Sobald diese Antwort auf die Bestätigungsmail eingegangen ist, ist man Mitglied der Liste und erhält regelmäßig die Nachrichten über neu entdeckte Sicherheitslücken.

Auf der Webseite von www.securityfocus.com können die Archive der Mailingliste eingesehen werden.

Die zweite große Institution für die Veröffentlichung von Sicherheitslücken ist CERT. Auf der Webseite von www.cert.org werden regelmäßig neue Sicherheitslücken und Lösungen dazu veröffentlicht.

Weitere wichtige Adressen im Internet sind

- www.ciag.org
- www.sans.org

Zusätzlich veröffentlichen die Linux-Distributoren selbst Nachrichten über Sicherheitslücken und liefern etsprechende Bug-Fixes. Ein paar Beispiele wären:

- SuSE-Linux Updates und Bugfixes
- RedHat Security Alerts and Bugfixes
- Debian Sicherheits-Informationen

Diese Informationsquellen sollten regelmäßig konsultiert werden, um über neue Sicherheitsprobleme informiert zu sein und auftretende Schlupflöcher zu schließen.

Das Programm socket

Das Programm socket(1) erlaubt es, auf beliebigen Ports entweder Client- oder Serversockets aufzubauen, deren Datenströme mit der Standardein- und -ausgabe verbunden werden. Dieses Programm kann dazu benutzt werden, bestimmte Ports des eigenen Rechners zu überprüfen, ob von außen dort noch Zugriffe möglich sind.

Selbst wenn alle möglichen Dienste, die von außen zugreifbar sind, geschlossen wurden, so existiert mit **socket** eine Möglichkeit, Angriffe von innen heraus zuzulassen. Das Programm sollte also mit Vorsicht benutzt werden und sicherheitshalber nicht für alle User zugänglich sein.

1.114.2 - Einrichten von Host-Security

Beschreibung: Prüfungskandidaten sollten über das Einrichten einer grundlegenden Host-Security Bescheid wissen. Diese Tätigkeiten enthalten die Konfiguration von syslog, Shadow-Paßwörter, das Einrichten eines Mail-Alias für die Mails von root und das Abdrehen der nicht verwendeten Netzwerkdienste.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/inetd.conf` oder `/etc/init.d/*`
- `/etc/nologin`
- `/etc/passwd`
- `/etc/shadow`
- `/etc/syslog.conf`

Auch für dieses Kapitel gilt, daß die meisten der hier geforderten Techniken bereits an anderer Stelle beschrieben wurden. Aus diesem Grund werden neben den Verweisen auf die bereits besprochenen Techniken nur noch einige Details geklärt.

Syslog Konfigurieren

Die Konfiguration des Syslog-Daemons wurde im Abschnitt 1.111.3 bereits ausführlich beschrieben. Sicherheitsrelevante Systemmeldungen können mit Hilfe dieser Technik in eine spezielle Datei geschrieben werden, die dann entsprechend beobachtet werden muß.

Typische Herkünfte von sicherheitsrelevanten Meldungen sind:

kern
: Meldungen der Firewalls

auth und **authpriv**
: Meldungen und vertrauliche Meldungen des Sicherheitsdienstes. Hier sind die logins verzeichnet.

Neben der Ausgabe in Dateien sei hier auch nochmal an die Möglichkeit erinnert, die Meldungen auch auf das Terminal bestimmter User (vorzugsweise des **root**-Users) zu schreiben.

Shadow-Passwörter

Alle modernen Linux-Distributionen arbeiten heute standardmäßig mit den Shadow-Passwörtern. Die genauen Beschreibungen der Formate der Shadow-Dateien sind in Abschnitt 1.111.1 nachlesbar.

Um ein System mit dem alten Passwortsystem in ein Shadow-Passwort-System zu konvertieren, existieren die Programme pwconv und grpconv. Umgekehrt können moderne Shadow-Systeme mit pwunconv und grpunconv wieder zurück in das alte Format verwandelt werden.

Logins zeitweise verbieten

Wenn der Systemverwalter für eine bestimmte Zeit verbieten will, daß sich User auf dem Rechner einloggen, so kann er einfach eine Datei `/etc/nologin` anlegen. Diese Datei kann einen kurzen Text enthalten, der erklärt, warum gerade kein Login möglich ist.

Sowohl das Programm **login**, als auch Netzwerkdienste wie **ssh, telnet, rlogin** oder **rsh** verweigern den Login, wenn diese Datei existiert. Statt dessen zeigen sie dem User den Inhalt der Datei `/etc/nologin` an.

Diese Beschränkung gilt nicht für den **root**-User.

Mail Alias von root

Normalerweise sollte der Systemverwalter nicht immer unter seinem **root**-login arbeiten, außer er arbeitet tatsächlich an der Verwaltung. Wenn er jedoch normale Arbeiten am Computer erledigt, sollte er mit einem Normallogin arbeiten, wie alle anderen auch. Das hat mehrere Gründe. Zum einen können fatale Fehler im Normalbetrieb nicht stattfinden, weil ein Normaluser diese Fehler nicht machen kann. Will ein User etwa ein Verzeichnis löschen und vertippt sich, so daß ein Leerzeichen in die Befehlszeile rutscht:

```
rm -r / foo
       ^
      /|\
       |
   unbeabsichtigtes Leerzeichen
```

so wäre das der Befehl, das Wurzelverzeichnis und alle darunterliegenden Verzeichnisse zu löschen. Das kann aber nur der Systemverwalter (root). Arbeitet **root** ständig mit seinem Systemverwalter-Account, so gibt es keine Instanz, die ihn hindern würde, solche Fehler zu machen. Arbeitet er als Normaluser, so werden solche Fehler mangels Berechtigung nicht ausgeführt.

Damit aber **root** trotzdem seine Mails bekommt, die an **root@localhost** gesendet wurden, kann er einen Mailalias einrichten. Diese Technik wurde bereits im Abschnitt 1.113.2 besprochen. Der Eintrag in `/etc/aliases` müsste z.B. lauten:

```
root:  hans
```

Nach dieser Veränderung müßte der Befehl

```
newaliases
```

ausgeführt werden, damit **sendmail** den Alias benutzen kann.

Eine andere Möglichkeit wäre die Weiterleitung der Mails von root an einen anderen User über die Benutzung der Datei `~/.forward` im Heimatverzeichnis von **root**

Nicht benötigte Netzwerkdienste abschalten

Netzwerkdienste können unter Linux grundsätzlich auf zwei verschiedene Arten gestartet werden. Entweder sie sind grundsätzlich schon im Arbeitsspeicher geladen und warten auf Aufträge, die sie

abarbeiten sollen, oder sie werden - jeweils wenn sie gebraucht werden - vom **inetd** aufgerufen. Beide Techniken haben ihre Vor- und Nachteile die Fall für Fall abgewogen werden sollten, um zu entscheiden, auf welche Weise ein bestimmter Dienst gestartet werden soll. Meist ist die Voreinstellung der Distributionen ein guter Anhaltspunkt, um zu entscheiden, ob ein Dienst als Stand-Alone oder inetdbasierter Dienst laufen soll.

Die Voreinstellungen der verschiedenen Distributionen ermöglichen oft Zugriffe, die in der Praxis nicht gewünscht sind. Um solche Dienste abzuschalten, muß entsprechend erstmal klar sein, wie er gestartet wird. Die beiden folgenden Abschnitte gehen die beiden möglichen Methoden durch und zeigen Möglichkeiten der Abschaltung.

Inetd-basierte Dienste abschalten

Wenn ein Dienst über den **inetd** (siehe auch Abschnitt 1.113.1) gestartet wird, so ist er in der Datei `/etc/inetd.conf` eingetragen. Wurde stattdessen der **xinetd** benutzt, liegen die Einstellungen in `/etc/xinetd.conf`. Einträge in diesen Dateien können einfach auskommentiert werden, indem ihnen ein Kommentarzeichen am Zeilenanfang vorangestellt wird.

Das alleinige Auskommentieren dieser Zeile reicht aber nicht aus, denn der **inetd** ließt diese Datei nur beim Start. Um einem laufenden **inetd** diese Veränderungen bekannt zu geben, wird ihm ein **HUP**-Signal geschickt.

In vielen Fällen ist es überhaupt nicht erwünscht, daß der **inetd** läuft. Sollte das der Fall sein, so kann dafür gesorgt werden, daß dieser Dienst selbst gar nicht gestartet wird. Der **inetd** selbst ist ein Stand-Alone-Dienst, dessen Abschaltung im nächsten Abschnitt beschrieben wird.

Stand-Alone Dienste abschalten

Stand-Alone Dienste werden in den modernen Linux-Distributionen alle auf eine gemeinsame Art und Weise gestartet. Im Verzeichnis `/etc/init.d` liegen Shellscripts, die die einzelnen Dienste starten und stoppen. Diese sogenannten Init-Scripts verstehen jeweils mindestens die Parameter **start** und **stop**. Häufig gibt es noch weitere Parameter wie **status** (gibt eine Ausgabe aus, ob der entsprechende Dienst läuft oder nicht), **reload** (zwingt den Dienst seine Konfigurationsdatei neu einzulesen) oder **restart** (Kombination aus **stop** und anschließendem **start**).

Jedesmal, wenn ein neuer Dienst installiert wird, installiert er ein entsprechendes Init-Script in das Verzeichnis `/etc/init.d`. Jeder Dienst kann jetzt mit Hilfe dieser Scripts gestartet werden. Installieren wir den Dienst **foo**, dann existiert also in diesem Verzeichnis ein Script, wahrscheinlich ebenfalls mit dem Namen `/etc/init.d/foo`. Um den Dienst von Hand zu starten, wird dieses Script mit dem Parameter **start** aufgerufen:

```
/etc/init.d/foo start
```

entsprechend kann ein laufender Dienst mit dem Script wieder heruntergefahren werden:

```
/etc/init.d/foo stop
```

Die alleinige Existenz dieser Scripte legt aber noch nicht fest, daß diese Dienste auch automatisch gestartet werden. Für diese Einstellungen sind die sogenannten Runlevel-Verzeichnisse gedacht.

Im Verzeichnis `/etc` (manchmal auch in `/etc/init.d` liegen für jeden Runlevel ein weiteres

Verzeichnis. Diese Runlevelverzeichnisse haben immer die Namen

rc*Runlevelnummer*.d

also etwa rc0.d, rc1.d, rc2.d oder rcS.d (Single User Mode). Innerhalb dieser Verzeichnisse finden sich symbolische Links auf die Init-Scripts, die in dem jeweiligen Runlevel ausgeführt werden sollen. Der Namen dieser Links beginnt entweder mit einem S (Start) oder mit einem K (Kill). Dem folgt eine zweistellige Nummer, die angibt, in welcher Reihenfolge die Scripts ausgeführt werden sollen. Anschließend folgt der Name des Scripts. Die Nummer ist kein absoluter Wert, sondern nur eine Hilfe für die alphabetische Sortierung der Scriptnamen für die Ausführung.

Um unser **foo**-Script in /etc/init.d automatisch im Runlevel 5 zu starten, genügt also die Erstellung eines symbolischen Links im Verzeichnis /etc/rc5.d in der Form:

```
lrwxrwxrwx 1 root root   13  5. Sep 17:22 S20foo -> ../init.d/foo
```

Die 20 bedeutet nicht, daß dieser Dienst an 20. Stelle gestartet wird. Alle Links in diesem Verzeichnis, die mit einem S beginnen, werden beim Eintritt in diesen Runlevel alphabetisch sortiert ausgeführt. Die Nummer dient also dazu, die Reihenfolge anzugeben, nicht die absolute Position. Es ist ohne Problem möglich, daß mehrere Links mit der selben Nummer arbeiten, sie werden dann hintereinander (S20bar vor S20foo) aufgerufen. Wird ein Script durch einen Link aufgerufen, dessen Namen mit einem S beginnt, so wird das entsprechende Script mit dem Parameter **start** aufgerufen

Analog dazu werden die Scripts, deren Namen mit K beginnen, beim Verlassen des Runlevels ausgeführt, wieder in alphabetischer Reihenfolge. Nur diesmal wird das Script, auf das sie verweisen mit dem Parameter **stop** aufgerufen.

Um also Dienste grundsätzlich abzuschalten und zu verhindern, daß diese Dienste bei einem Neustart wieder automatisch geladen werden, muß im entsprechenden Runlevelverzeichnis der entsprechende Link einfach gelöscht werden.

1.114.3 - Einrichten von Sicherheit auf Benutzerebene

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Sicherheit auf Benutzerebene zu implementieren. Die Tätigkeiten beinhalten das Verwalten von Beschränkungen auf Benutzerlogins, Prozesse und Speichernutzung.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **quota**
- **usermod**

Die Verwaltung von Einschränkungen einzelner Useraccounts ist ein vielfältiger Bereich, der in einzelnen Fällen schon in anderen Abschnitten beschrieben wurde. Die Haupt-Möglichkeiten der Einschränkungen liegen in einer durchdachten Organisation der Dateien und Programme hinsichtlich ihrer Gruppenzugehörigkeit. Sind bestimmte Programme nur von Mitgliedern einer bestimmten Gruppe ausführbar, so kann z.B. die Einstellung, welcher User ein solches Programm ausführen darf einfach über die Gruppenmitgliedschaft des Users verwaltet werden. Diese Techniken sind im Abschnit 1.111.1 bereits beschrieben worden.

Auch die Möglichkeit, die Gültigkeit eines Accounts und andere einen User betreffende Einstellungen mit usermod zu verändern wurde dort bereits ausführlich besprochen.

Die Einschränkungen des Festplattenplatzes, den einzelne User zur Verfügung haben mittels diskquotas wurde bereits bei der Vorbereitung auf die Prüfung LPI101 im Abschnitt 1.104.4 ausführlich besprochen und wird daher hier nicht weiter behandelt.

Einschränkungen mit ulimit

Eine Sache, die noch keine Beschreibung erfahren hat, für userbezogene Einstellungen aber sehr wichtig ist, ist der Umgang mit dem Shellbefehl **ulimit**. Es handelt sich hier um ein eingebautes (internes) Shellkommando, das der Shell selbst Einschränkungen hinsichtlich der von ihr benötigten Ressourcen macht. In der Regel werden diese Einstellungen in systemweiten Startdateien der Shell (wie etwa `/etc/profile`) gemacht. Einstellungen in usereigenen Startdateien sind in der Regel sinnlos, weil jeder User diese Einschränkungen selbst wieder aufheben könnte.

ulimit kennt verschiedene Möglichkeiten für Beschränkungen von Ressourcen, die für die Shell und alle Prozesse, die von ihr gestartet werden gelten. Die Syntax ist:

> **ulimit**[-SHacdflmnpstuv [*Limit*]]

Die Optionen `-H` und `-S` spezifizieren jeweils den Hard- bzw. Softlimit einer Ressource. Ein Hardlimit kann nicht mehr erhöht werden, sobald er einmal gesetzt ist. Ein Softlimit kann bis zum Wert des Hardlimits erhöht werden. Werden beide Angaben weggelassen, so werden sowohl Hard-, als auch Softlimit gesetzt.

Die Angabe des *Limit* kann entweder eine Zahl sein, in der für die angegebene Ressource gültigen Einheit, oder einer der Spezialwerte **hard, soft** oder **unlimited**. Diese Spezialwerte bezeichnen die

für diese Ressource eingestellten Hard- und Softlimits bzw. keinen Limit.

Ist kein *Limit* angegeben, so wird der gegenwärtige Wert des Softlimits der entsprechenden Ressource ausgegeben. Ist zusätzlich ein `-H` angegeben, so wird stattdessen der Hardlimit angezeigt.

Die einzelnen Ressourcen werden über Parameter eingestellt und haben die folgenden Bedeutungen:

-a
: Alle aktuellen Limits werden dargestellt

-c
: Stellt die maximale Größe von Core-Dateien ein. Das sind Dateien, die erstellt werden, wenn ein Programm abstürzt. Diese Dateien enthalten dann den Daten- und Programmbereich des abgestürzten Programms für die Fehlersuche mit einem Debugger. Ein Wert von 0 schaltet Core-Files grundsätzlich ab.

-d
: Die maximale Größe des Datensegments eines Prozesses.

-f
: Die maximale Größe von Dateien, die die Shell anlegt.

-l
: Die maximale Speichergröße, die gesperrt werden darf

-m
: Die maximale Größe von residentem Speicher

-n
: Die maximale Anzahl gleichzeitig geöffneter Dateien

-p
: Die Größe von Pipes in 512 Byte Blöcken

-s
: Die maximale Stack-Größe

-t
: Der maximale Wert von CPU-Zeit in Sekunden

-u
: Die maximale Anzahl von Prozessen für einen einzelnen User

-v
: Die maximale Größe von virtuellem Speicher, der für die Shell verfügbar ist.

Ist *Limit* angegeben, so bezeichnet er den neuen Wert für die angegebene Ressource (außer bei `-a`, das nur Ausgaben macht). Wird keine Ressource angegeben, so wird `-f` angenommen. Die Werte beziehen sich in der Regel auf 1024 Byte Blöcke, außer bei der Angabe von `-t` (Sekunden), `-p` (512-Byte Blöcke) und `-n` und `-u`, die keinen Multiplikator haben sondern genau so gewertet werden, wie angegeben.

Ein typischer Eintrag in `/etc/profile` wäre z.B.

```
ulimit -Sc 0        # Der Soft-Limit für Core-Dateien wird
                    # auf 0 gesetzt
                    # Es werden also keine Corefiles angelegt.
ulimit -d 15000     # Hard- und Softlimit für die Datengröße
                    # eines Programmes werden auf 15 MByte
                    # gesetzt (15000*1024)
ulimit -s 15000     # Hard- und Softlimit für die Stack-Größe
```

```
                    # eines Programmes werden auf 15 MByte
                    # gesetzt.

# Die folgende Konstruktion gilt nur für den User hans.
# Er darf nicht mehr als 10 Prozesse gleichzeitig laufen haben.
if [ $LOGNAME = "hans" ]
then
  ulimit -u 10
fi
```

Mit diesen Einstellungen ist es möglich, die einzelnen User in ihrem Ressourcenverbrauch einzuschränken. Diese Ressourcen beziehen sich aber nur auf eingeloggte User, nicht auf Netzwerkdienste. Genau besehen sind es ja Einstellungen der Shell. Also greifen diese Einstellungen nur da, wo die Shell aktiv ist.