KAPITEL 9

Das System sicher konfigurieren

Sie haben nunmehr ein funktionierendes Stand-alone-System. Im Grunde fehlt nicht mehr viel, und Sie hätten auch eine funktionierende Internet-Anbindung. Tatsächlich ist dies zum jetzigen Zeitpunkt aber nicht wünschenswert. Je nach verwendeter Distribution klaffen in Ihrem System momentan mehr oder weniger große Sicherheitslöcher, die gestopft werden sollten, bevor Sie eventuellen Angreifern die Gelegenheit geben, ihre Künste an Ihrer Firewall zu erproben.

Hierzu werden wir im folgenden einen kleinen Rundgang durch Ihr System unternehmen und es einmal auf gängige Schwachstellen abklopfen.

Cron

Neben den Diensten, die, sobald sie einmal gestartet wurden, permanent aktiv sind, existieren auch noch Programme, die regelmäßig ausgeführt werden sollten, sich nach Erledigung ihrer Aufgabe dann aber beenden. So wäre es z. B. sinnvoll, täglich ein Backup durchzuführen. Dazu könnte man jede Nacht um drei Uhr morgens ein Backup-Programm starten. Da aber auch ein Operator gelegentlich schlafen muß, ist es sinnvoll, dafür zu sorgen, daß das Programm automatisch zur festgelegten Uhrzeit gestartet wird. Hierzu existiert ein spezieller Dienst namens Cron. Dieser verwaltet eine Reihe von Listen, die *Crontabs*, in denen steht, welches Programm (*Cronjob*) der Rechner wann starten muß.

Der normalerweise unter Linux eingesetzte Cron wurde von Paul Vixie geschrieben und kennt zwei Sorten von Crontabs. Da sind zum einen die benutzerspezifischen Crontabs, die unter */var/spool/cron/tabs/<Benutzerkennung>*[1] zu finden sind. Zusätzlich existiert noch die System-Crontab in Form der Datei */etc/crontab*, die von Cron ebenfalls ausgewertet wird. Eventuell unter */etc/cron.d/* befindliche Dateien werden dabei als Erweiterungen der System-Crontab angesehen. Diese Dateien werden dabei oft für allgemeine Aufräumarbeiten des Systems genutzt. Es existieren aber auch Distributionen, die solche Aufgaben in die benutzerspezifische Crontab von `root` eintragen.

1 Früher wurde */var/cron/tabs/<Benutzerkennung>* verwendet.

Ein Eintrag in einer Crontab hat folgenden Aufbau (vgl. `man 5 crontab`):

<Minute> <Stunde> <Tag d. Monats> <Monat> <Wochentag> [*<Benutzerkennung>*] *<Befehl>*

Die Zeitangaben werden dabei ab 0 durchnumeriert (0 = 1. des Monats, Januar, Sonntag). Eine Benutzerkennung wird nur in der System-Crontab angegeben, Befehle in den benutzerspezifischen Crontabs werden mit den Rechten ihres Besitzers ausgeführt. Will man eines der Zeitfelder nicht definieren, so kann man statt dessen ein »∗« eintragen. Auch die Angabe von Bereichen und symbolischen Namen ist zulässig. Mehr Details dazu finden sich in der oben genannten Manpage.

Neben den auszuführenden Befehlen können auch Umgebungsvariablen in einer Crontab eingetragen werden. Derartige Einträge haben die folgende Form:

<Name> = <Wert>

Interessant ist dabei insbesondere die Variable *MAILTO*. Sie regelt, was mit den Ausgaben der gestarteten Programme geschieht. Ist sie nicht gesetzt, so werden alle Ausgaben per E-Mail an den Besitzer der Crontab geschickt. Ist sie gesetzt, aber leer (*MAILTO=""*), so werden eventuelle Ausgaben ignoriert. Enthält sie schließlich einen Wert, so wird dieser als Empfänger der E-Mail mit den Ausgaben angenommen.

Mit diesem Wissen ausgerüstet, sollten wir beginnen, uns die Crontabs im System einmal genauer anzuschauen. Es gilt zum einen, ein Gefühl dafür zu bekommen, welche Crontabs sich legitimerweise im System befinden. Es ist nämlich durchaus nicht ungewöhnlich, daß ein Angreifer, wenn er einmal Kontrolle über das System erlangt hat, eine Zeitbombe in Form eines Cronjobs einbaut. Sollte sein Zugang zum Rechner jemals gesperrt werden, so könnte ein Skript, das regelmäßig gestartet wird, diesen Zustand wieder rückgängig machen oder dem System – quasi als Vergeltungsmaßnahme – massiven Schaden zufügen.

Ein anderer Grund ist die Tatsache, daß die gängigen Installationen automatisch Cronjobs installieren. Diese löschen z. B. in regelmäßigen Zeitabständen temporäre Dateien oder komprimieren Protokolldateien. Dies ist oft sinnvoll. Es kommt aber manchmal vor, daß zuviel des Guten getan wird. Hier sollten wir prüfen, ob die Meinung der Distributoren, welche Automatismen sinnvoll sind, mit unseren Vorstellungen übereinstimmt.

Untersuchen wir nun ein SuSE 9.3-System, wie es hier beschrieben wurde, so finden wir die folgende Situation vor:

- */var/spool/cron/tabs/* ist leer, d. h., es existieren keine benutzerspezifischen Crontabs.
- */etc/cron.d/* ist ebenfalls leer.
- */etc/crontab* existiert.

Die Datei */etc/crontab* hat folgenden Inhalt:

```
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily, cron.weekly, and cron.monthly
#
-*/15 * * * *   root  test -x /usr/lib/cron/run-crons && /usr/lib/cron/run-crons \
>/dev/null 2>&1
```

Alle Einträge werden mit den Rechten von Root gestartet, der auch benachrichtigt wird, falls die Programme Ausgaben tätigen.

Der erste Eintrag startet ein Skript namens `run-crons`. Dieses tut die folgenden Dinge:

- Es überprüft, ob es auf einem Laptop im Batteriebetrieb läuft und beendet sich gegebenenfalls.
- Es legt ein Verzeichnis */var/spool/cron/lastrun* an.
- Es sorgt dafür, daß Programme in /etc/cron.hourly/, /etc/cron.daily/, /etc/cron.weekly/, /etc/cron.monthly jeweils stündlich, täglich, wöchentlich bzw. monatlich ausgeführt werden. Dabei werden leere Dateien */var/spool/cron/lastrun/* dazu verwendet, festzustellen, ob es schon wieder Zeit ist, die Programme in einem bestimmten Verzeichnis auszuführen.
- Entstehen dabei Hilfsdateien, deren Datum in der Zukunft liegt, so werden sie gelöscht.

Grundsätzlich hat es schon etwas für sich, Verzeichnisse zu schaffen, deren Inhalt regelmäßig ausgeführt wird. Dies ist deutlich einfacher zu handhaben als die etwas kryptische Cron-Syntax. Allerdings müssen wir nun vier weitere Verzeichnisse auf Skripte überprüfen.

Im hier geschilderten Fall enthält nur das Verzeichnis */etc/cron.daily* Skripte. Dabei handelt es sich um die folgenden Skripte:

clean_catman Beim Anzeigen einer Manualseite muß diese zuerst aus einem Quelltext in eine Form gebracht werden, die angezeigt werden kann. Um diesen Prozeß etwas zu beschleunigen, kann man mit dem Programm `catman` einen definierten Satz von Manualseiten einmal formatieren und dann ablegen. Wird nun der `man`-Befehl aufgerufen, kann er auf eine vorformatierte Seite zurückgreifen.

`clean_catman` ist nun dafür zuständig, vorformatierte Seiten zu löschen, auf die lange nicht zugegriffen wurde. Dies spart Speicherplatz, bedeutet aber auch, daß die Anzeige dieser Seiten etwas länger dauert.

Bewertung: Sinnvoll

do_mandb erzeugt eine Indexdatenbank der Manualseiten. Dies erlaubt es, nicht nur nach den Befehlen selbst, sondern auch mit `apropos` und `man -k` nach Stichwörtern in den Kurzbeschreibungen der Befehle zu suchen.

Bewertung: Sinnvoll

logrotate ist ein Dienst, der Logdateien regelmäßig kopiert, das Original löscht und die Archivversion komprimiert. Zu alte Versionen der Logdateien werden endgültig gelöscht.

Bewertung: Grundsätzlich sinnvoll. Sie sollten aber die Konfigurationsdateien des Dienstes überprüfen. Dabei handelt es sich z. B. um */etc/logrotate.conf* und alle Dateien in */etc/logrotate.d*. Achten Sie darauf, wie oft eine Rotation der Dateien stattfindet und wann diese weggeworfen werden. Außerdem sollte es sich tatsächlich um Logdateien handeln. Andernfalls könnte es passieren, daß plötzlich wichtige Systemdateien gelöscht werden. Mehr zu `logrotate` finden Sie in Kapitel 16, Unterabschnitt *logrotate*, ab Seite 503.

suse.de-backup-rc.config überprüft, ob sich der Inhalt der Konfigurationsdatei */etc/rc.config* oder der Dateien in */etc/sysconfig/* geändert hat. Ist dies der Fall, wird ein Archiv der neuen Dateien in einem in der Datei */etc/sysconfig/backup* in der Variablen *RCCONFIG_BACKUP_DIR* festgehaltenen Verzeichnis abgelegt. Die Anzahl der Generationen, die bereitgehalten werden, kann über die Variable *MAX_RCCONFIG_BACKUPS* konfiguriert werden.

Bewertung: Grundsätzlich sinnvoll. Sie sollten aber die Konfigurationsdatei */etc/sysconfig/backup* dahingehend überprüfen, ob die dort eingestellten Werte Ihren Vorstellungen entsprechen.

suse.de-backup-rpmdb ähnelt `suse.de-backup-rc.config`. Hier wird allerdings ein Backup der Datenbank installierter Softwarepakete gemacht. Die Konfiguration erfolgt ebenfalls über die Datei */etc/sysconfig/backup*.

Bewertung: Grundsätzlich sinnvoll. Sie sollten aber die Konfigurationsdatei */etc/sysconfig/backup* dahingehend überprüfen, ob die dort eingestellten Werte Ihren Vorstellungen entsprechen.

suse.de-check-battery überprüft, ob die CMOS-Battery auf dem Motherboard noch in Ordnung ist.

Bewertung: Sinnvoll

suse.de-clean-tmp soll alte temporäre Dateien löschen. Konfiguriert wird dies über die Datei */etc/sysconfig/cron*. Dort sind die folgenden Variablen definiert:

MAX_DAYS_IN_TMP gibt an, nach wie vielen Tagen eine Datei gelöscht werden soll. Ist der Wert `0`, so wird nie gelöscht.

(Vorgabe SuSE 9.3: 0)

TMP_DIRS_TO_CLEAR gibt an, welche Verzeichnisse temporäre Dateien enthalten. Ihre Namen sollten besser keine Leerzeichen enthalten.

(Vorgabe SuSE 9.3: */tmp*)

MAX_DAYS_IN_LONG_TMP entspricht *MAX_DAYS_IN_TMP*, gilt aber für eine zweite Gruppe von Verzeichnissen

(Vorgabe SuSE 9.3: 0)

LONG_TMP_DIRS_TO_CLEAR entspricht *TMP_DIRS_TO_CLEAR*. Allerdings wird die Häufigkeit, mit der diese Verzeichnisse gelöscht werden, durch *MAX_DAYS_IN_LONG_TMP* statt *MAX_DAYS_IN_TMP* bestimmt.

(Vorgabe SuSE 9.3: leer)

OWNER_TO_KEEP_IN_TMP Dateien dieser Nutzer werden nicht gelöscht.

(Vorgabe SuSE 9.3: `root`)

CLEAR_TMP_DIRS_AT_BOOTUP gibt an, ob zusätzlich bei jedem Neustart alle temporären Dateien gelöscht werden sollen. Steht sie auf `yes`, dann werden alle Dateien in *TMP_DIRS_TO_CLEAR* gelöscht. Alternativ kann in der Variable auch eine Liste von Verzeichnissen angegeben werden, die gelöscht werden sollen. Wichtig ist dabei, daß *MAX_DAYS_IN_TMP* und *OWNER_TO_KEEP_IN_TMP* nicht beachtet werden. Dieser Vorgang wird allerdings nicht von `cron` gesteuert, sondern von einem Runlevel-Skript namens `boot.cleanup`.

(Vorgabe SuSE 9.3: `no`)

Bewertung: **Gefährlich**, diese Funktionalität erlaubte es in SuSE 6.3 beliebigen Benutzern, beliebige Dateien zu löschen [25]. Die Art, wie Dateien gesucht werden, ist recht anfällig für Programmierfehler. Meine Empfehlung wäre, die Konfigurationsdatei zu überprüfen, ob die dort eingetragenen Werte tatsächlich den oben angegebenen Vorgabewerten entsprechen, und dann das Skript `suse.de-clean-tmp` zu löschen. Leider kann man `boot.cleanup` nicht einfach löschen, da es noch eine Reihe anderer Aufgaben wahrnimmt.

suse.de-cron-local startet das Skript `/root/bin/cron.daily.local`, falls vorhanden.

Bewertung: Überflüssig. Falls `root` Cronjobs starten will, kann er ebenfalls Skripte in */etc/cron.daily/* ablegen. Dies schafft nur einen Platz mehr, an dem man nach automatisch ausgeführten Skripten suchen muß. Meine Empfehlung ist, das Skript zu löschen.

In einer normalen Installation existiert noch ein weiterer Dienst, der es ebenfalls erlaubt, Programme zu einer bestimmten Uhrzeit zu starten. Im Gegensatz zu einem Cronjob wird es allerdings nur einmalig ausgeführt. Der dafür zuständige Dienst heißt `atd`. Normale Benutzer können ihm mit den Befehlen `at` und `batch` Aufträge erteilen. In unserer speziellen Installation ist er allerdings nicht enthalten oder wird zumindest nicht gestartet.

Der inetd

Der `inetd` ist ein *Metaserver*. Gesteuert durch die Datei */etc/inetd.conf*, wartet er auf bestimmten Ports auf eintreffende Pakete. Wird er nun angesprochen, startet ein externes Programm, das das jeweilige Paket (und eventuell darauf folgende) bearbeitet.

Das externe Programm muß dabei nicht als Netzwerkdienst programmiert sein. Es muß lediglich auf der Standardeingabe Daten entgegennehmen und seine Ausgaben auf die Standardausgabe ausgeben. Damit lassen sich z. B. normale Unix-Befehle in Informationsdienste umwandeln:

```
systat   stream  tcp  nowait  nobody  /bin/ps       /bin/ps -auwwx
netstat  stream  tcp  nowait  root    /bin/netstat  /bin/netstat
```

Hier werden zwei Dienste definiert, die jedem Anfragenden die auf dem Rechner laufenden Prozesse und alle gerade bestehenden Verbindungen auflisten. Es versteht sich wohl von selbst, daß dies eine Konfiguration ist, die auf einem sensitiven Rechner möglichst vermieden werden sollte. Sie zeigt aber auch, wie der `inetd` das Schreiben von Servern vereinfacht.

An diesen Beispielen sehen wir, wie ein Eintrag aufgebaut ist. Er besteht aus fünf Elementen:

Portnummer Im obigen Beispiel sind die Portnummern durch ihre logischen Namen aus der Datei */etc/services* angegeben. Sie könnten aber auch numerisch (z. B. 80 für http, 11 für systat) aufgeführt werden.

Protokollklasse Protokolle können in verschiedene Typen unterteilt werden. Da sind die verbindungsorientierten wie z. B. TCP. Diese werden mit »stream« gekennzeichnet. Verbindungslose Protokolle wie UDP sind dagegen vom Typ »dgram«. Weitere Typen existieren, werden aber nur in Spezialfällen benötigt.

Protokoll Hier kann jedes in der Datei */etc/protocols* aufgeführte Protokoll stehen, üblich sind aber vor allem »tcp« und »udp«.

Servertyp Es gibt zwei Sorten von Servern. Die einen erzeugen für jedes hereinkommende Paket eines neuen Senders einen neuen Prozeß, der dann nur Pakete dieser einen Gegenstelle bearbeitet. Diese werden als »nowait« eingetragen. Die andere Klasse bearbeitet der Reihe nach alle Pakete, bis eine Zeitlang keine neuen Pakete mehr eintreffen und der Server sich beendet. Diese werden mit »wait« gekennzeichnet.

Die Unterscheidung wird nur für verbindungslose Protokolle getroffen. Verbindungsorientierte Protokolle werden immer als »nowait« konfiguriert.

Es kann auch noch angegeben werden, wie viele Instanzen des Servers innerhalb einer Minute maximal gestartet werden dürfen. Die Voreinstellung liegt bei 40, sie kann aber durch Anhängen eines Punktes und einer Zahl modifiziert werden. »nowait.300« würde also theoretisch bis zu 300 neue Anfragen pro Minute verarbeiten können. Praktisch gibt es allerdings irgendwo eine physikalische Grenze, die nicht überschritten werden kann.

Benutzer Jeder Prozeß muß einem Benutzer zugeordnet sein. Da der `inetd` standardmäßig mit Rootrechten läuft, wäre dies root. Dies ist allerdings nicht immer wünschenswert. Daher wird hier angegeben, mit wessen Rechten der Server tatsächlich laufen soll.

Serveraufruf Der Rest der Zeile wird als Befehl ausgeführt. Dabei ist der erste Eintrag das auszuführende Programm. Die folgenden Einträge sind die Argumente für das Programm. Das erste Argument ist dabei, wie unter Unix üblich, der Name des Programms,[2] bei den restlichen handelt es sich um Parameter.

Standardein- und -ausgabe werden auf eine Netzverbindung umgeleitet.

Zeilen, die mit einem # als erstem Zeichen beginnen, sind Kommentare und werden nicht beachtet. Dies erlaubt es, Einträge, die wir momentan nicht benötigen, zu deaktivieren, ohne die Zeile ganz zu entfernen. Wir sollten von dieser Möglichkeit radikal Gebrauch machen und jeden Server auskommentieren, den wir nicht unbedingt benötigen. Dabei gilt: »Was wir nicht kennen, brauchen wir nicht!« Tatsächlich läßt sich der `inetd` in Firewall- und Desktop-Installationen in der Regel komplett deaktivieren, ohne daß wir dadurch irgendwelche Nachteile hätten. Wie dies geschehen kann, sehen Sie in Kapitel 9, Abschnitt *Entfernen unnötiger Programme, Dienste, Dateirechte und Dateien*, ab Seite 184.

Hat man die Konfigurationsdatei des `inetd` geändert, so muß man dies dem Dienst mitteilen. Dazu ist es nötig, ihm ein Signal SIGHUP zu senden. Signale sind dabei ein Weg, dem Programm mitzuteilen, daß etwas Besonderes passiert ist, das unabhängig vom normalen Programmablauf behandelt werden muß. Übliche Signale sind:

SIGTERM Das Programm soll beendet werden. Es kann aber eine spezielle Routine enthalten, die das Signal bearbeitet und die nicht zur Beendigung des Programms führt. Das Signal kann auch numerisch als 15 angegeben werden.

SIGKILL Das Programm soll beendet werden. Dieses Signal führt immer zum sofortigen Programmabbruch und kann auch numerisch als 9 angegeben werden.

SIGINT Der Anwender hat <Ctrl><C> gedrückt. Das Signal kann vom Programm behandelt werden, führt aber ansonsten zum Programmabbruch. Numerisch wird es als 2 angegeben.

SIGHUP Dient oft dazu, Server dazu aufzufordern, sich neu zu initialisieren. Numerisch wird es als 1 angegeben.

Man kann ein Signal mit dem Befehl `kill` an einen Prozeß senden. Dazu muß man allerdings seine Prozeßnummer (PID) kennen. Diese erhält man, indem man den Befehl `ps` benutzt. Man sollte letzteren mit den Parametern »ax« aufrufen, um alle Prozesse angezeigt zu bekommen:

2 Jedes Programm erhält unter Unix eine Bezeichnung als »nulltes« Argument. Dabei handelt es sich normalerweise um den Namen, unter dem es aufgerufen wurde. Der Aufruf `./proggie 1 2 3` bewirkt also, daß das Programm `proggie` mit den Argumenten `./proggie`, `1`, `2` und `3` aufgerufen wird. Dies gilt allerdings nur, wenn das Programm von der Kommandozeile aus aufgerufen wird. Prinzipiell ist es auch möglich, ein Programm aus einem anderen Programm heraus aufzurufen und dabei ein beliebiges nulltes Argument zu übergeben.

```
# ps ax
PID TTY  STAT TIME COMMAND
1   ?    S    0:05 init
2   ?    SW   0:00 [kflushd]
3   ?    SW   0:00 [kupdate]
4   ?    SW   0:00 [kpiod]
5   ?    SW   0:00 [kswapd]
6   ?    SW   0:00 [md_thread]
71  ?    S    0:00 /usr/sbin/syslogd
75  ?    S    0:01 /usr/sbin/klogd
92  ?    S    0:02 /usr/sbin/inetd
116 ?    S    0:00 sendmail:
124 ?    R    5:04 /usr/sbin/cron
141 tty1 S    0:00 login
142 tty2 S    0:00 /sbin/mingetty
143 tty3 S    0:00 /sbin/mingetty
144 tty4 S    0:00 /sbin/mingetty
145 tty5 S    0:00 /sbin/mingetty
146 tty6 S    0:00 /sbin/mingetty
```

Hier hat der `inetd` die Prozeßnummer 92. Mit dem Befehl

```
# kill -s SIGHUP 92
```

kann er dazu aufgefordert werden, seine Konfigurationsdatei erneut zu lesen.

Einfacher können wir ein Signal mit dem Befehl `killall` senden, der aber nicht in allen Distributionen enthalten ist:

```
# killall -HUP inetd
```

Daß hier `HUP` statt `SIGHUP` steht, ist eine Besonderheit von `killall`. Das `SIG`, das eigentlich alle Signale im Namen tragen, entfällt hier.

Sollten Sie allerdings jemals ein anderes Unix-System außer Linux administrieren, so sollten Sie beachten, daß dort `killall` u. U. eine andere Bedeutung hat. System-V-Systeme kennen ein `killall`, das versucht, alle Prozesse außer den eigenen Elternprozessen zu beenden. Dies wird dort beim Herunterfahren des Rechners benutzt und ist ansonsten eher kontraproduktiv. Unter Linux existiert dieses Kommando als `killall5`.

Stellt man fest, daß man einen oder mehrere der vom `inetd` gestarteten Server benötigt, so muß man überlegen, wie man regelt, wer auf den bereitgestellten Dienst zugreifen darf. Vom `inetd` gestartete Server sind oft recht simpel aufgebaut und verfügen weder über ein adäquates Logging noch über eine Zugriffskontrolle.

Diesen Mangel behebt der `tcpd`. Er wird anstelle des Servers als auszuführendes Programm angegeben. Unser obiges Beispiel wird dabei zu

```
systat   stream  tcp  nowait  nobody  /usr/sbin/tcpd  /bin/ps -auwwx
netstat  stream  tcp  nowait  root    /usr/sbin/tcpd  /bin/netstat
```

Dadurch wird der `tcpd` anstelle der Programme ausgeführt. Anhand seines ersten Arguments erkennt der `tcpd`, welches Programm er starten soll. Ob er dies auch tut, wird über die Dateien */etc/hosts.allow* und */etc/hosts.deny* gesteuert. Trifft eine Regel in */etc/hosts.allow* zu, so wird der Zugriff gestattet. Ist dies nicht der Fall, aber es existiert

eine Regel in */etc/hosts.deny*, so wird der Zugriff verweigert. Im letzten Fall wird der Zugriff gestattet.

Regeln sind folgendermaßen aufgebaut:

<*Dienstliste*> : <*Rechnerliste*>

oder

<*Dienstliste*> : <*Rechnerliste*> : <*Befehl*>

Elemente einer Liste werden dabei durch Leerzeichen oder Kommata getrennt. Eine Regel trifft dann zu, wenn sowohl der Dienst als auch der Rechner, der die Verbindung zu öffnen wünscht, zutreffen. Ist ein Befehl angegeben, so wird er ausgeführt, wenn die Regel zutrifft. Dabei ist zu beachten, daß nach der ersten zutreffenden Regel die Auswertung der Regeln abgebrochen wird.

Die Rechnerlisten können auf mehrere Arten definiert werden. Eine vollständige Liste liefert der Aufruf `man 5 hosts_access`. Lesenswert ist auch die Datei */usr/doc/packages/tcp_wrapper/README*. Für den Hausgebrauch reichen aber die im folgenden vorgestellten Möglichkeiten:

<***Adressangabe***> Natürlich können Adressen normal angegeben werden (z. B. `10.0.0.1` oder `dummy.foo.bar`).

.<log. Domain> Eine Adresse wie `.miscatonic.edu` würde für alle Rechner der Miscatonic Universität in Arkham gelten, genauer für alle Adressen, die auf *miscatonic.edu* enden. Sowohl *cthulhu.miscatonic.edu* als auch *alhazred.miscatonic.edu* würden von dieser Spezifikation erfaßt.

<partielle IP-Addr>. Dies ist das numerische Gegenstück. Hier gibt man den ersten Teil der Adresse an. `10.0.0.` würde auf alle Adressen von *10.0.0.1* bis *10.0.0.255* passen.

n.n.n.n/o.o.o.o Dies ist eine alternative Methode, eine partielle Adresse zu spezifizieren. Man gibt eine vollständige Adresse an, gefolgt von einer Maske, in der die Bits gesetzt sind, die einen an der Adresse tatsächlich interessieren. Das obige Beispiel würde damit zu `10.0.0.0/255.255.255.0`.

ALL Dieser Bezeichner paßt immer. Er kann auch als Serverspezifikation benutzt werden.

LOCAL Dieser Bezeichner paßt auf Rechner, deren Name keinen Punkt enthält.

<$Liste_1$> ***EXCEPT*** <$Liste_2$> Diese Angabe gilt für alle Rechner aus <$Liste_1$>, die nicht auch in <$Liste_2$> stehen.

Auf die Ausführung von Befehlen werde ich hier nicht eingehen. Zwar erlaubt es dieses Feature, »Fallen« einzubauen, die zusätzliche Erkundigungen im Falle eines unerlaubten Zugriffes anstellen, die Gefahr ist aber hoch, dabei versehentlich Schwachstellen ins System einzubauen (siehe Kapitel 4, Unterabschnitt *Unsichere Applikationen*, ab Seite 36). Auf einer Firewall sollte man daher auf Nummer Sicher gehen und die Installation so simpel wie möglich halten.

Nehmen wir einmal an, wir wollen nun den `tcpd` so konfigurieren, daß auf den `systat`-Dienst zugegriffen werden darf, auf alle anderen Dienste aber nicht. Dabei sollen diese Zugriffe auf den lokalen Rechner und den Rechner 10.0.0.1 beschränkt sein. In diesem Falle könnte die Datei */etc/hosts.allow* folgendermaßen aussehen:

```
systat : LOCAL, 10.0.0.1
```

In der Datei */etc/hosts.deny* sollten dagegen alle anderen Dienste generell verboten werden:

```
ALL : ALL
```

Erfolgen nun Zugriffe, so erscheinen sie in jedem Fall im Systemprotokoll. Hier habe ich mit den Regeln herumexperimentiert und den Zugriff mal erlaubt und mal nicht:

```
May 24 21:53:01 host4711 netstat[839]: refused connect from 127.0.0.1 (127.0.0.1)
May 24 22:02:30 host4711 netstat[868]: connect from localhost (127.0.0.1)
```

Entfernen unnötiger Programme, Dienste, Dateirechte und Dateien

Generell ist ein System um so sicherer, je weniger man damit tun kann. Ein absolut sicheres System wäre eines, das keinerlei Funktion erfüllt. Obwohl wir dieses Ideal wohl nicht ganz erreichen werden, sollten wir doch diejenigen Funktionalitäten entfernen, die wir nicht benötigen. Deshalb sollten wir verhindern, daß unnötige Server und Hintergrundprozesse gestartet werden, und darauf achten, daß die Dateirechte möglichst restriktiv gesetzt sind.

Automatisch gestartete Programme

Eine ganze Reihe von Programmen wird automatisch ausgeführt und läuft dann ständig im Hintergrund, ohne daß der Benutzer etwas davon bemerkt. Wir wollen uns im folgenden einmal einen Überblick verschaffen und dann die Dienste deaktivieren, die in unserem System überflüssig sind.

Serverdienste

Obwohl wir in Kapitel 9, Abschnitt *Der inetd*, ab Seite 179 schon viele Serverdienste entfernt haben, könnten noch immer Server durch die Runlevel-Skripte gestartet worden sein. Tatsächlich werden normalerweise nur ziemlich einfache Server durch den `inetd` gestartet.

Einen genauen Überblick, auf welchen Netzwerkports Server aktiv sind, liefert der Befehl `netstat`. Er besitzt mehrere Optionen, über die wir steuern können, welche Informationen wir benötigen. Eine ausführliche Auflistung liefert die Manpage des Befehls, aber für unsere Zwecke reichen uns folgende:

-a zeigt neben aktiven Verbindungen auch die Ports an, auf denen gerade ein Server auf Anfragen wartet.

-n zeigt Adressen und Ports numerisch an. Im Moment haben wir noch keine Namensauflösung konfiguriert, wir wollen also nicht, daß `netstat` versucht, einen DNS-Server zu kontaktieren, um IP-Adressen in logische Adressen umzuwandeln.

-p zeigt zusätzlich an, welches Programm einen Port benutzt oder von unserem Rechner aus an einer Verbindung teilnimmt.

--ip zeigt ausschließlich Informationen für Protokolle der IP-Familie an. Insbesondere werden so Angaben zu Unix-Domain-Sockets unterdrückt. Dieses Konstrukt wird zwar in der Programmierung wie Netzwerk-Sockets angesprochen, kann aber ausschließlich zur Kommunikation auf dem Rechner selbst verwendet werden. Aus diesem Grund sind Unix-Domain-Sockets für uns normalerweise uninteressant.

Leider führt dieser unter bestimmten Umständen dazu, daß nicht nur die Anzeige von Unix-Domain-Sockets unterdrückt wird, sondern auch die von IPv6-Ports. Daher sollten ihn wir auf Systemen, auf denen der Kernel IPv6 unterstützt, nicht einsetzen. Statt dessen können wir Zeilen für Unix-Domain-Sockets mit `grep` ausfiltern.

Wenn wir nun `netstat` einmal auf einem SuSE-9.3-Minimalsystem ausprobieren, so sieht das z. B. folgendermaßen aus:

```
# netstat -anp | grep -v '^unix'
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address  Foreign Address State  PID/Program name
tcp        0      0 *:ssh          *:*             LISTEN 5329/sshd
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
```

Der `sshd` ist ein Serverdienst, der es erlaubt, sich über das Netzwerk auf dem Rechner anzumelden. Wenn Sie die Firewall tatsächlich fernwarten müssen, so ist ein sauber aufgesetzter und sicher konfigurierter SSH-Dienst sicherlich eine sinnvolle Lösung. Wenn es aber möglich ist, so sollten Sie auf die Fernwartung verzichten und sich lieber physisch zum Rechner begeben.

Ein Fernwartungsdienst wie der `sshd` birgt immer die Gefahr, daß Sie plötzlich und unerwartet zusätzliche Administratoren auf Ihrem System bekommen. Es wurden durchaus schon Sicherheitslücken im Dienst gefunden, die es Angreifern erlaubten, die Kontrolle über den Rechner zu übernehmen. Zwar wurden diese recht schnell behoben, aber wer nicht sofort die nötigen Patches einspielte, drohte kurz darauf ein Opfer sich schnell verbreitender Würmer zu werden.

Hintergrundprozesse

Neben den Netzwerkdiensten existieren noch weitere Prozesse, die von Runlevel-Skripten gestartet werden und die dann im Hintergrund aktiv bleiben, bis der Rechner heruntergefahren wird. Um sich einen Überblick zu verschaffen, ist der Befehl `ps` recht nützlich:

```
# ps ax -H
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:00 init [3]
    2 ?        SN     0:00   [ksoftirqd/0]
    3 ?        S<     0:00   [events/0]
    4 ?        S<     0:00   [khelper]
    9 ?        S<     0:00   [kthread]
   19 ?        S<     0:00     [kacpid]
   93 ?        S<     0:00     [kblockd/0]
  133 ?        S      0:00     [pdflush]
  134 ?        S      0:00     [pdflush]
  136 ?        S<     0:00     [aio/0]
  135 ?        S      0:00   [kswapd0]
  728 ?        S      0:00   [kseriod]
 1182 ?        S      0:00   [kjournald]
 2173 ?        S<s    0:00   /sbin/udevd -d
 2214 ?        S      0:00   [khpsbpkt]
 2557 ?        S      0:00   [khubd]
 2588 ?        S      0:00   [kgameportd]
 3794 ?        S<     0:00   [hwscand]
 4537 ?        Ss     0:00   /sbin/resmgrd
 5313 ?        Ss     0:00   /sbin/syslog-ng
 5329 ?        Ss     0:00   /usr/sbin/sshd -o PidFile=/var/run/sshd.init.pid
 5332 ?        Ss     0:00   /sbin/klogd -c 1 -x -x
 5412 ?        Ss     0:00   /usr/sbin/cron
 5420 ?        Ss     0:00   login -- root
 5476 tty1     Ss     0:00     -bash
27228 tty1     R+     0:00       ps ax -H
 5421 tty2     Ss+    0:00   /sbin/mingetty tty2
 5422 tty3     Ss+    0:00   /sbin/mingetty tty3
 5423 tty4     Ss+    0:00   /sbin/mingetty tty4
 5424 tty5     Ss+    0:00   /sbin/mingetty tty5
 5425 tty6     Ss+    0:00   /sbin/mingetty tty6
```

Hier haben wir ein Minimalsystem vor uns, das sechs virtuelle Konsolen besitzt, von denen zur Zeit nur eine benutzt wird. Dies kann man daran erkennen, daß auf tty2 bis tty6 `mingetty`s darauf warten, daß ein Benutzer seinen Namen eingibt. Ist dies geschehen, so wird `login` gestartet, das das Paßwort entgegennimmt. Bei erfolgreicher Überprüfung desselben startet es die Shell des Benutzers (hier: `-bash`). Auch sich selbst zeigt der `ps`-Befehl an (27228).

Zusätzlich zu den benutzten existiert eine Vielzahl weiterer Parameter für `ps`, um z. B. Besitzer, Parameter, Environment, Ressourcennutzung oder Hierarchie der Prozesse anzuzeigen. Probieren Sie einmal `ps aeux -H -w|less -s` (falls die Zeilen abgeschnitten werden, so fügen Sie einfach weitere `-w` an).

Neben dem schon erwähnten `sshd`, `init` (siehe Kapitel 8, Unterabschnitt *Init*, ab Seite 125) und einigen Systemprozessen (erkennbar an den eckigen Klammern) finden sich in unserem Beispielsystem noch die folgenden Dienste:

udevd Dieser Dienst unterstützt das Hotplugging. Damit können automatisch neue Module geladen und Devices angelegt werden, wenn neue Hardware an das System angeschlossen wird.

resmgrd Dies ist ein Dienst, um den Zugriff auf bestimmte Devices zu regeln. Allerdings funktioniert er nur mit Anwendungen, die diesen Dienst unterstützen.

syslog-ng Dieser Dienst führt die Systemprotokolle.

klogd Der `klogd` liest Protokollmeldungen des Kernels aus einem speziellen Speicherbereich aus und übergibt sie an den `syslog-ng`.

cron Der `cron` führt Jobs regelmäßig zu vorher festgelegten Uhrzeiten aus. Dies eignet sich gut, um ständig vorkommende Routineaufgaben zu automatisieren.

Entfernung überflüssiger Dienste

Um einen Dienst zu deaktivieren, gilt es zuerst herauszufinden, welches Skript den Dienst startet. Runlevel-Skripte liegen bei SuSE und Debian unterhalb von */etc/init.d/*.

Wir wollen im folgenden einmal versuchen, den `sshd` zu entfernen. Wir wissen, daß das Programm `sshd` aus einem Skript in */etc/init.d* gestartet wird.

Um nun das Vorkommen von Dateien */etc/init.d* aufzuspüren, die die Zeichenkette `sshd` enthalten, können wir die Befehle `find` und `grep` benutzen.

`find` sucht alle Einträge in allen Unterverzeichnissen eines angegebenen Pfades, die vorher festgelegte Eigenschaften haben. Man kann z. B. reguläre Dateien (`-type f`), Verzeichnisse (`-type d`), Dateien mit SUID-Bit (`-perm 4000`) oder Dateien mit einem bestimmten Namen (`-name` <*Name*>) suchen. Die gefundenen Dateien können dann in verschiedenen Formatierungen ausgegeben werden. Es ist aber auch möglich, mit `-exec` ein Programm aufzurufen, dem man als Parameter den Namen der gefundenen Datei übergibt.

In unserem Fall bietet es sich an, `grep` zu benutzen, um aus den mit `find` gefundenen regulären Dateien unterhalb von */etc/init.d* diejenigen herauszufinden, die die Zeichenkette `master` enthalten. Dazu muß man `grep` als erstes Argument die gesuchte Zeichenkette übergeben, gefolgt von einer Liste der zu durchsuchenden Dateien. In unserem Fall dient dabei `\{\}` als Platzhalter für die jeweils von `find` gefundene Datei. `grep` durchsucht nun die Datei und gibt jede Zeile aus, in der das Schlüsselwort gefunden wurde. Der Parameter `-v` würde dieses Verhalten umkehren und nur die Zeilen anzeigen, in denen der Schlüsselbegriff nicht vorkommt. Hier verwenden wir aber statt dessen `-H`, um `grep` den Namen der jeweiligen Datei anzeigen zu lassen, und `-n`, um auch die Nummer der jeweiligen Zeile zu erfahren. Die Argumente für `-exec` (und damit auch der `grep`-Befehl) werden mit `\;` abgeschlossen.

Wir finden also:

```
# find /etc/init.d -type f -exec grep -n -H sshd \{\} \;
/etc/init.d/.depend.start:1:TARGETS = fbset kbd nfs [...]  sshd [...]
/etc/init.d/.depend.start:22:SuSEfirewall2_setup: [...] sshd autoyast
/etc/init.d/.depend.start:35:sshd: network nfs
/etc/init.d/.depend.stop:1:TARGETS = [...] sshd autoyast
/etc/init.d/.depend.stop:8:network: [...] sshd autoyast
/etc/init.d/.depend.stop:12:nfs: [...] sshd autoyast
/etc/init.d/.depend.stop:15:hotplug: [...] sshd autoyast
/etc/init.d/.depend.stop:20:pcmcia: [...] sshd autoyast
```

```
/etc/init.d/sshd:6:# /etc/init.d/sshd
/etc/init.d/sshd:10:# /usr/sbin/rcsshd
/etc/init.d/sshd:13:# Provides: sshd
/etc/init.d/sshd:18:# Description: Start the sshd daemon
/etc/init.d/sshd:21:SSHD_BIN=/usr/sbin/sshd
/etc/init.d/sshd:28:SSHD_PIDFILE=/var/run/sshd.init.pid
/etc/init.d/sshd:64:    startproc -f -p $SSHD_PIDFILE /usr/sbin/sshd $SSHD_OPTS \
> -o "PidFile=$SSHD_PIDFILE"
/etc/init.d/sshd:74:    killproc -p $SSHD_PIDFILE -TERM /usr/sbin/sshd
/etc/init.d/sshd:100:   echo -n "Reload service sshd"
/etc/init.d/sshd:102:   killproc -p $SSHD_PIDFILE -HUP /usr/sbin/sshd
/etc/init.d/sshd:108:   echo -n "Checking for service sshd "
/etc/init.d/sshd:118:   checkproc -p $SSHD_PIDFILE /usr/sbin/sshd
/etc/init.d/sshd:126:        test /etc/ssh/sshd_config -nt $SSHD_PIDFILE && echo> \
reload
```

.depend.start und *.depend.stop* dienen unter SuSE dazu, die Reihenfolge festzulegen, in der Dienste gestartet werden, es sind aber keine Runlevel-Skripte. Damit bleibt nur `/etc/init.d/sshd` übrig. Eine nähere Untersuchung zeigt, daß es tatsächlich das gesuchte Skript ist. Des weiteren geht aus dem Quelltext hervor, daß sein einziger Zweck ist, den `sshd` zu starten. Dies ist nicht immer so. In manchen Distributionen startet ein Skript gleich mehrere Dienste. Dann wird es notwendig, das Skript zu editieren und den Teil auszukommentieren, der für den fraglichen Server zuständig ist.

Hier reicht es aber, das Skript mit `insserv -r sshd` abzumelden (SuSE) bzw. vorhandene S-Links in K-Links umzubenennen (Debian). Dazu kann man manuell die Verzeichnisse nach Links durchsuchen, man kann aber auch wieder `find` bemühen:

```
# find /etc/init.d/rc*.d -lname "*sshd*" -printf "%p->%l\n"
/etc/init.d/rc3.d/S10sshd->../sshd
/etc/init.d/rc3.d/K12sshd->../sshd
/etc/init.d/rc5.d/S10sshd->../sshd
/etc/init.d/rc5.d/K12sshd->../sshd
```

Hier benutzen wir `-lname`, um Links zu finden, bei denen nicht etwa der Name des Links die Zeichenkette `sshd` enthält, sondern der Name der Datei, auf die der Link verweist. So finden wir auch Links, die nicht nach dem Skript benannt sind. `-printf` erlaubt es uns festzulegen, wie die Ausgabe von `find` formatiert sein soll. Hier geben wir sowohl den Namen des Links aus wie auch den der Datei, auf den er verweist.

Hierbei ist allerdings zu beachten, daß man den Suchpfad entsprechend der verwendeten Distribution angibt (SuSE: */etc/init.d/rc*.d*, Debian: */etc/rc*.d*).

Nach dem nächsten Systemstart sollte man überprüfen, ob die abzustellenden Dienste auch tatsächlich nicht mehr gestartet werden. Als erste Überprüfung kann man den Rechner auch mit `init 1` in den Single User Mode herunterfahren, dort überprüfen, ob der Dienst auch wirklich beendet wurde, und den Rechner dann mit `init 3` wieder herauffahren. Nun sollte der Dienst nicht erneut gestartet worden sein.

Abschließend sind in den Tabellen 9-1 und 9-2 noch einige Runlevel-Skripte aus einer SuSE- und einer Debian-Installation aufgeführt, die einer näheren Betrachtung wert sind.

Tabelle 9-1: Dienste unter SuSE 9.3

Name	Runlevel	Bewertung
S07nfs	3,5	Mountet Netzwerklaufwerke. Firewalls sollten dies allerdings normalerweise nicht tun. *Empfehlung:* Deaktivieren
S10sshd	3,5	Der `sshd` erlaubt einen sicheren Zugriff auf die Firewall von entfernten Rechnern. Um allerdings zu vermeiden, daß auch unautorisierte Dritte Ihren Rechner fernwarten, sollten Sie den Dienst nur aktivieren, wenn Sie wissen, was Sie tun. *Empfehlung:* Im Zweifelsfall deaktivieren.

Tabelle 9-2: Dienste unter Debian 3.1

Name	Runlevel	Bewertung
S15bind9	2, 3, 4, 5	DNS-Server. Sein Einsatz kann auf einer Firewall sinnvoll sein. *Empfehlung:* Bei Bedarf aktiviert lassen.
S20exim4	2, 3, 4, 5	Mailserver. `exim` gilt als deutlich sicherer als `sendmail`. Trotzdem sollte man einen Mailserver nur dann betreiben, wenn man sich vorher gründlich über seine Konfiguration informiert hat. *Empfehlung:* Im Zweifelsfall deaktivieren.
S20ftp-proxy	2, 3, 4, 5	FTP-Proxy. Sein Einsatz kann auf einer Firewall sinnvoll sein. *Empfehlung:* Bei Bedarf aktiviert lassen.
S20inetd	2, 3, 4, 5	Der `inetd` wurde schon in Kapitel 9, Abschnitt *Der inetd*, ab Seite 179 besprochen. Er sollte nur dann aktiviert sein, wenn nach der Entfernung aller überflüssigen Dienste noch Zeilen in der */etc/inetd.conf* übrig sind. *Empfehlung:* Im Zweifelsfall deaktivieren.
S20isdnutils	2, 3, 4, 5	Konfiguration von ISDN. Da wir hierzu eigene Skripte verwenden werden, entstehen Konflikte. *Empfehlung:* Deaktivieren
S20privoxy	2, 3, 4, 5	Webproxy, der sich besonders zur Filterung von Werbebannern und Cookies eignet. *Empfehlung:* Bei Bedarf aktiviert lassen.
S20ssh	2, 3, 4, 5	Der `sshd` erlaubt einen sicheren Zugriff auf die Firewall von entfernten Rechnern. Um allerdings zu vermeiden, daß auch unautorisierte Dritte Ihren Rechner fernwarten, sollten Sie den Dienst nur aktivieren, wenn Sie wissen, was Sie tun. *Empfehlung:* Im Zweifelsfall deaktivieren.
S20squid	2, 3, 4, 5	Cachender Webproxy. Sein Einsatz kann auf einer Firewall sinnvoll sein. *Empfehlung:* Bei Bedarf aktiviert lassen.
S45mountnfs	0,S	Mountet Netzwerklaufwerke. Firewalls sollten dies allerdings normalerweise nicht tun. *Empfehlung:* Deaktivieren
S89atd	2, 3, 4, 5	Dienst, der einen Dienst einmalig zu einer bestimmten Uhrzeit startet. Wird normalerweise nicht benötigt. *Empfehlung:* Deaktivieren

Dateirechte

Haben wir bisher versucht, alle Netzwerkdienste zu finden, mit denen ein Angreifer einen Weg in das System finden könnte, so sollten wir jetzt versuchen, ihm Wege zu verbauen, mit denen er sich nach einem erfolgreichen Angriff höhere Rechte verschaffen könnte.

SUID- und SGID-Programme

Als erstes sollten wir überprüfen, welche Programme als SUID (04000) oder SGID (02000) markiert sind. Da diese nicht mit den Zugriffsrechten des Aufrufers, sondern denen ihres Besitzers bzw. denen einer bestimmten Gruppe laufen, kann ein Angreifer sich mit ihrer Hilfe Zugriff auf Dateien oder Ressourcen verschaffen, für die er selbst keine Berechtigung besitzt:

```
# find / -perm +6000 -type f -printf "%-30p\t%u.%g\t%m\n"
/bin/ping                      root.root       4755
/bin/ping6                     root.root       4755
/bin/su                        root.root       4755
/bin/eject                     root.audio      4755
/bin/mount                     root.root       4755
/bin/umount                    root.root       4755
/sbin/unix_chkpwd              root.shadow     2755
/sbin/unix2_chkpwd             root.shadow     2755
/usr/bin/man                   root.root       4755
/usr/bin/mandb                 root.root       4755
/usr/bin/gpg                   root.root       4755
/usr/bin/chage                 root.shadow     4755
/usr/bin/chfn                  root.shadow     4755
/usr/bin/chsh                  root.shadow     4755
/usr/bin/expiry                root.shadow     4755
/usr/bin/gpasswd               root.shadow     4755
/usr/bin/newgrp                root.root       4755
/usr/bin/passwd                root.shadow     4755
/usr/bin/rcp                   root.root       4755
/usr/bin/rlogin                root.root       4755
/usr/bin/rsh                   root.root       4755
/usr/bin/wall                  root.tty        2755
/usr/bin/write                 root.tty        2755
/usr/bin/crontab               root.trusted    4755
/usr/lib/pt_chown              root.root       4755
/usr/lib/mc/cons.saver         root.root       4755
/usr/sbin/utempter             root.tty        2755
/usr/sbin/pppoe-wrapper        root.dialout    4750
/usr/sbin/exim                 root.root       4755
/usr/sbin/pam_auth             root.shadow     2755
```

Nun müssen wir entscheiden, ob wir beim jeweiligen Programm das SUID- bzw. SGID-Bit entfernen können, ohne daß das System unbenutzbar wird. Ist dies der Fall, so sollten wir konsequent handeln. Ist es nötig, dem Programm das jeweilige Bit zu lassen, sollten wir diese Tatsache dokumentieren. Zusätzlich sollten wir eine Liste anlegen, in der alle Dateien mit ihren Rechtebits aufgeführt sind, deren SUID- oder SGID-Bit gesetzt ist. Untersuchen wir dann später ein System darauf, ob es von einem Angreifer verändert wurde, dann wissen wir zumindest, welche Programme rechtmäßigerweise mit erweiterten Rechten laufen.

Um zu einer Entscheidung zu gelangen, gilt es festzustellen, wozu das jeweilige Programm dient. Mehrere Informationsquellen bieten sich an, wenn wir einmal nicht genau wissen, welchem Zweck ein Programm dient.

1. Die Online-Dokumentation (`man` <*Programm*> bzw. `info` <*Programm*>)
2. Die Paketdokumentationen unter */usr/[share/]doc/*<*Paketname*>
3. Die Quelltexte des Programms

Um herauszufinden, zu welchem Paket ein Programm gehört, und gegebenenfalls die Quelltexte nachzuinstallieren, können Sie unter SuSE den Paketmanager `rpm` benutzen:

```
# rpm -q -f /usr/sbin/pam_auth
squid-2.3.STABLE4-51
```

Unter Debian führt Sie dagegen der folgende Aufruf zum Ziel:

```
# dpkg -S /usr/sbin/pam_auth
```

Die gefundenen Programme lassen sich in mehrere Gruppen einteilen:

ping, ping6, mount, umount, at, crontab, mandb, rcp, rlogin, rsh Hier handelt es sich um Standardprogramme, die von normalen Benutzern nur ausgeführt werden können, wenn sie SUID `root` sind. Allerdings ist es auch nicht wirklich nötig, daß normale Benutzer sie ausführen dürfen.

Empfehlung: SUID-Bit entfernen

gpg Mit diesem Programm kann man Dateien und Texte verschlüsseln. Das SUID-Bit ist nötig, um Bereiche im Hauptspeicher dagegen zu schützen, auf die Festplatte ausgelagert zu werden, wo sie dann mit spezieller Software gefunden werden könnten.

Empfehlung: Wenn normale Benutzer `gpg` nicht nutzen sollen, dann sollten Sie das SUID-Bit entfernen, andernfalls in einer speziellen Dokumentation notieren, daß das SUID-Bit bewußt behalten wurde.

eject Dieses Programm kann diverse Wechseldatenträger auswerfen. Wenn der Datenträger gemountet ist, wird ein `umount` ausgeführt. Um auf die jeweiligen Devices zugreifen zu können, benötigt das Programm Rootrechte.

Nun lassen sich bei der Benutzung normaler PC-Hardware die gängigen Wechselmedien wie CDs, Disketten und Zip-Medien leicht durch Drücken eines Knopfes auswerfen.[3] Auch existiert auf einer Firewall kaum ein Grund, warum normale Benutzer überhaupt Wechselmedien benutzen sollten. Auch ein Blick in den Quelltext liefert keinen Grund zur Entwarnung. Das Programm kennt Kommandozeilenargumente, ist 871 Zeilen lang und gibt an keiner Stelle seine Rootrechte auf. Ein Beweis, daß es keine Speicherüberläufe oder andere Probleme mit bösartigen Benutzereingaben gibt, mag möglich sein, ist aber nicht trivial. Angesichts des geringen praktischen Nutzens ist das SUID-Bit eigentlich nicht zu rechtfertigen.

Empfehlung: SUID-Bit entfernen

man Der Befehl `man` generiert Seiten dynamisch, wenn sie gebraucht werden. Damit dies bei häufig benötigten Befehlsbeschreibungen nicht jedesmal aufs neue geschehen muß, werden einmal generierte Seiten eine Weile in einem speziellen Verzeichnis

3 Dies steht im Gegensatz zu anderen Hardwarearchitekturen. Auf SUN-Maschinen war früher durchaus ein Programm wie `eject` nötig, um Disketten wieder aus dem Gerät herauszubekommen.

gelagert. Damit diese auch anderen Benutzern zugute kommen, finden alle Zugriffe als der Pseudonutzer »man« statt. Dies erscheint mir kein allzu großes Risiko.

Empfehlung: In einer speziellen Dokumentation notieren

write, wall Beide Programme schreiben Mitteilungen auf die Konsole eines anderen Benutzers. Dazu benötigen sie natürlich Schreibrecht, das sie erlangen, indem sie mit dem Gruppenrecht `tty` ausgeführt werden. Dies beinhaltet übrigens kein Leserecht, so daß auch Mitglieder der Gruppe `tty` nicht einfach mitlesen können, was jemand tippt oder am Bildschirm sieht. Das SGID-Bit ist hier nicht so gefährlich, aber doch relativ überflüssig, da normale Benutzer in der Regel eher selten von dieser Möglichkeit Gebrauch machen und auf einer Firewall auch normalerweise nicht allzu viele Benutzer gleichzeitig angemeldet sind.

Empfehlung: SGID-Bit entfernen

chage, chfn, chsh, expiry, gpasswd, newgrp Diese Kommandos dienen zur Benutzer- und Gruppenadministration. Normale Benutzer benötigen sie nicht. Darüber hinaus verläßt man sich hier darauf, daß sicherheitskritische Anwendungen selbst entscheiden, welche ihrer Funktionen von normalen Benutzern genutzt werden können.

Empfehlung: SUID/SGID-Bit entfernen

passwd, su Diese Befehle erlauben es normalen Benutzern, sein Paßwort zu ändern und sich unter einer anderen Kennung anzumelden. Es handelt sich um sinnvolle Kommandos, die völlig unbrauchbar werden, wenn man ihr SUID-Bit entfernt.

Empfehlung: In einer speziellen Dokumentation notieren

unix_chkpwd, unix2_chkpwd Hierbei handelt es sich um ein Hilfsprogramm, das mit einer Fehlermeldung abbricht und eine Fehlermeldung in das Systemprotokoll schreibt, wenn es von Hand aufgerufen wird. Es dient dazu, bestimmten Programmen zu ermöglichen, eingegebene Paßwörter zu überprüfen. Als Beispiel werden Bildschirmschoner angegeben.

Diese Möglichkeit scheint kaum genutzt zu werden. Auf meinem Arbeitsrechner ist das SGID-Bit entfernt, ohne daß ich bisher einen Unterschied feststellen konnte.

Empfehlung: SGID-Bit entfernen

ssh `ssh` benötigt aus zwei Gründen Rootrechte. Zum einen benutzt es standardmäßig einen niedrigen Quellport, um bei Bedarf eine unsichere RSH- statt einer sicheren SSH-Verbindung aufbauen zu können. Dies kann aber über einen Kommandozeilenparameter abgeschaltet werden. Zum anderen muß er auf die Datei */etc/ssh_host_key* zugreifen können. Diese ist normalerweise nur mit Rootrechten lesbar. Das ließe sich aber ändern, indem man eine neue Gruppe kreiert, die diese Datei lesen darf und der alle Benutzer angehören, die dieses Programm aufrufen dürfen.

Empfehlung: In einer speziellen Dokumentation vermerken, über weitergehende Maßnahmen gründlich nachdenken.

disable-paste Dieses Programm gehört zum `gpm`. Es führt den Befehl `gpm -A -q` aus. Damit wird der Kopierpuffer des `gpm` gelöscht. Auf diese Weise wird verhindert, daß der zuletzt angemeldete Benutzer einen Befehl im Kopierpuffer speichert, den man dann versehentlich durch Drücken der mittleren Maustaste ausführt. Es bietet sich

an, den Aufruf aus dem Anmeldeskript seiner Shell heraus auszuführen. Dazu sind allerdings Rootrechte erforderlich. Aus diesem Grund existiert dieses Rahmenprogramm, das ausschließlich den besagten Aufruf ausführt und sich dann beendet. Ein Mißbrauch des Programms für Angriffe dürfte schwerfallen, da es weder Parameter kennt, noch Eingaben entgegennimmt. Man kann dem Programm daher seine Rootrechte belassen.

Empfehlung: In einer speziellen Dokumentation notieren

pt_chown Dieses Programm ist eine Hilfsapplikation der `glibc`. Es setzt die Rechte des gerade benutzten Terminals auf Lesen und Schreiben für den aktuellen Benutzer, Schreiben für die Gruppe `tty`. Es wird vom Login-Prozeß benötigt. Ein Blick in den Quelltext zeigt, daß es, sobald es mit Parametern aufgerufen wird, die zusätzlichen Rootrechte aufgibt, bevor die Parameter verarbeitet werden (`setuid(getuid())`). Das SUID-Bit sollte damit keine Gefahr einer Erweiterung der Rechte für normale Benutzer bergen.

Empfehlung: In einer speziellen Dokumentation notieren

cons.saver Dieses Programm gehört zum Midnight Commander. Es handelt sich um einen Bildschirmschoner. Entfernt man das SUID-Bit, so kann es passieren, daß der Bildschirm beim Aufruf des Midnight Commanders durch einen normalen Benutzer schwarz bleibt. Fehlt das Programm allerdings oder ist es nicht ausführbar, so funktioniert alles wie gewohnt.

Empfehlung: SUID-Bit und Ausführungsrechte entfernen oder das Programm löschen.

exim `exim` dient dazu, E-Mails den einzelnen Benutzern zuzustellen. Dazu muß das Programm die jeweilige E-Mail-Datei unter */var/spool/mail/* schreiben können.

Empfehlung: In einer speziellen Dokumentation notieren

pam_auth `pam_auth` wird vom `squid` benutzt, wenn Benutzer authentisiert werden müssen. Allerdings besteht in der hier beschriebenen Konfiguration für den `squid` keine Notwendigkeit dazu.

Empfehlung: SUID-Bit entfernen

Bevor wir nun die speziellen Rechte von einzelnen Programmen entfernen, sollten wir eine gewisse Nachvollziehbarkeit sicherstellen. Falls das System hinterher nicht mehr vernünftig benutzbar ist, so wäre es gut zu wissen, was wir geändert haben, um es gegebenenfalls rückgängig machen zu können. Auch wäre es gut zu wissen, welchen Programmen man die speziellen Rechte gelassen hat. Dies kann bei der Spurensuche nach einem Einbruch sehr hilfreich sein, um neue SUID-Programme des Angreifers zu finden. Es ist daher sinnvoll, alle Änderungen zu dokumentieren und diese Unterlagen der Dokumentation des Rechners hinzuzufügen.

Nun könnte man im Prinzip loslegen. SUID-Bits kann man manuell mit `chmod u-s` <*Dateiname*> entfernen, für SGID-Bits verwendet man `chmod g-s` <*Dateiname*>. Dies ist allerdings ziemlich aufwendig.

Bequemer wird es, wenn wir die Ausgabe unseres `find`-Befehls durch einen Filter leiten, der daraus eine Liste von Anweisungen generiert, welche die Dateien in ihren gegenwärtigen Zustand versetzen. Wenn wir die Ausgabe in eine Datei umlenken und dann die Anweisungen entsprechend unseren Wünschen anpassen, so besitzen wir ein Shellskript, mit dem wir die oben besprochenen Änderungen in einem Rutsch erledigen können.

Für diese Aufgabe bietet sich die Programmiersprache `awk` an. Sie basiert darauf, daß die Eingabe zeilenweise gelesen wird. Man legt nun Muster fest, denen Befehle zugeordnet werden. Paßt ein Muster auf eine Eingabezeile, so werden die zugeordneten Befehle ausgeführt. Das folgende Programm

```
{print "chown " $2 " " $1;
print "chmod " $3 " "  $1;
print ""}
```

legt z. B. kein Muster fest und wird deshalb für jede Zeile ausgeführt. Es besteht aus drei Ausgabebefehlen (`print`), die jeweils eine eigene Zeile ausgeben. Der erste gibt »chown« gefolgt von dem zweiten und dem ersten Wort der Eingabezeile aus. Der zweite Befehl funktioniert entsprechend. Der dritte Befehl erzeugt schließlich eine Leerzeile.

Wir können dieses Programm in eine Datei schreiben, man kann es aber `awk` auch auf der Kommandozeile mitgeben. Damit kommen wir zu dem folgenden Befehl. Da er nicht in eine Zeile paßt, enden die ersten beiden Zeilen auf \. Dies zeigt der Shell, daß noch eine Zeile folgt. Sie fordert daher mit > zur Eingabe des Rests auf:

```
# find /mnt -perm +6000 -type f -printf "%-30p\t%u.%g\t%m\n"| \
> awk '{print "chown " $2 " " $1; print "chmod " $3 " " \
> $1; print ""}'
```

Um daraus ein Shellskript zu machen, muß man das Ganze nur geringfügig verändern.

```
# echo '#!/bin/bash' >setperms
# find /mnt -perm +6000 -type f -printf "%-30p\t%u.%g\t%m\n"| \
> awk '{print "chown " $2 " " $1; print "chmod " $3 " " \
> $1; print ""}' >>setperms
# chmod u+x setperms
# cp setperms setorigperms
```

Will man nun Besitzer oder Gruppe einer Datei ändern, so findet man im Skript `setperms` einen `chown`-Befehl, den man nur anpassen muß. Für Änderungen an Rechten sind die `chmod`-Befehle da. Um ein SUID-Bit zu entfernen, zieht man einfach 4 von der ersten Ziffer ab, für ein SGID-Bit 2. Aus 4755 für ping wird damit 0755. Hat man dabei einen Fehler begangen, so kann man `setorigperms` benutzen, um die Änderungen rückgängig zu machen.

Theoretisch könnten wir nun mit `./setperms` die Änderungen durchführen. Allerdings gibt es da unter SuSE-Linux einen Haken. Auch der `yast` (bzw. das von ihm aufgerufene `SuSEconfig`[4]) überprüft Berechtigungen von Dateien und paßt diese gegebenenfalls an.

4 Ein undokumentiertes Shellskript, das die Datei */etc/sysconfig/security* auswertet und gemäß den darin enthaltenen Variablen das System konfiguriert. Zum Prüfen und Ändern der Dateirechte benutzt es übrigens ein Perlskript namens `chkstat`.

Die vorgesehenen Dateirechte findet er in mehreren Dateien. Grundsätzlich werden */etc/permissions* sowie alle Dateien in */etc/permissions.d/* benutzt. Darüber hinaus bedient er sich der Konfigurationsdatei */etc/permissions.<level>*. Welche Datei(en) er konkret verwendet, hängt von der Variablen *PERMISSION_SECURITY* in der Datei */etc/sysconfig/security* ab. Dort können mehrere durch Leerzeichen getrennte Werte für *<level>* angegeben werden. Erstellt man eine neue Datei */etc/permissions.meine*, so reicht der zusätzliche Eintrag von »`meine`« in die Liste, und die in der neuen Datei eingetragenen Rechte werden ebenfalls abgearbeitet.

Standardmäßig existieren die folgenden Dateien:

/etc/permissions.easy Diese Datei realisiert Standardrechte. Die Einstellungen sind mehr an der Bequemlichkeit als an der Sicherheit orientiert.

/etc/permissions.secure Alle eingetragenen Dateien haben keine SUID/SGID-Bits mehr. Allerdings existieren ein paar SUID/SGID-Programme, die nicht eingetragen sind.

/etc/permissions.paranoid Ein interessanter Versuch, das Konzept auf die Spitze zu treiben. Allerdings sollte diese Datei nicht gedankenlos eingestellt werden. Ohne ein Verständnis für die ihr zugrundeliegenden Ideen wird man sehr schnell Probleme bekommen, das System überhaupt noch zu benutzen.

/etc/permissions.local Diese Datei enthält keine Einträge, sondern ist für eigene Ergänzungen gedacht. SuSE garantiert, daß sie bei Updates nicht überschrieben wird.

Standardmäßig enthält die Variable *PERMISSION_SECURITY* übrigens `easy local`. Es empfiehlt sich, dies auf `secure local` zu ändern.

Nun kann man die Datei */etc/permissions.local* an die eigenen Bedürfnisse anpassen. Zweckmäßigerweise füllt man sie erst einmal mit dem Ist-Zustand:

```
# find / -perm +6000 -type f -printf \
> "%-30p\t%u.%g\t%m\n" >>/etc/permissions.local
```

Nun kann man wie besprochen Rechte in der dritten Spalte und Besitzer in der zweiten Spalte ändern, um dann die Änderungen mit dem Aufruf von `SuSEconfig` in Kraft treten zu lassen.

Veränderbare Dateien

Ein weiteres Problem besteht, wenn normale Benutzer Dateien verändern, die ihnen nicht gehören. Handelt es sich dabei um Programme oder Konfigurationsdateien, so steht dem Erlangen von Rootrechten oft nicht mehr viel im Wege. Folgender Aufruf findet Dateien, deren Inhalt von anderen Personen als dem Besitzer überschrieben werden kann:

```
# find / -perm +022 -type f -printf "%-30p\t%u.%g\t%m\n"
/var/log/wtmp                  root.tty        664
/var/run/utmp                  root.tty        664
```

In diesem Fall wurden zwei Protokolldateien gefunden, die außer durch root auch durch die Gruppe tty schreibbar sind. Dies ist in einem System ohne graphische Oberfläche allerdings unnötig, da hier die Programme, die diese Dateien schreiben (init, login und u. U. mingetty), mit Rootrechten laufen. Lediglich Terminalprogramme unter X (xterm, kvt, ...) werden nur mit dem Gruppenrecht tty ausgeführt. Sollen diese in die erwähnten Protokolldateien schreiben, so muß die Gruppe tty Schreibrecht haben. Hat sie es nicht, so zeigen Befehle wie w und who Benutzer nicht an, die statt virtueller Konsolen Terminalfenster benutzen.

In unserem Fall sollten wir die Rechte auf 644 (nur root darf schreiben) oder 600 (nur root darf Informationen über angemeldete Benutzer schreiben oder lesen) setzen.

Um Dateien verändern zu können, ist es allerdings nicht zwangsläufig nötig, Schreibrecht auf ihnen zu besitzen. Es reicht, Schreibrecht auf einem Verzeichnis zu besitzen, um beliebige Dateien in ihm zu löschen. Diese können dann durch eigene Versionen ersetzt werden. Den einzigen Schutz davor stellt das *Sticky-Bit* dar. Ist es für ein Verzeichnis gesetzt, so können Dateien nur von ihrem Besitzer gelöscht werden. Daher sollte dieses Bit für jedes Verzeichnis gesetzt sein, auf dem mehrere Benutzer Schreibrecht besitzen. Klassische Beispiele für solche Verzeichnisse sind */tmp* und */var/tmp*.

Sehen wir nun in unserer Beispielinstallation nach, ob Verzeichnisse existieren, die nicht unseren Ansprüchen genügen.

```
# find / -type d -perm +022 -not -perm +1000 -printf "%-30p\t%u.%g\t%m\n"
/etc/cups                      lp.lp   775
/var/games                     games.games     775
/var/spool/clientmqueue        mail.mail       770
/var/spool/postfix/maildrop    postfix.maildrop        730
```

Das Verzeichnis */var/games* ist für veränderliche Daten im Zusammenhang mit Spielen gedacht (z. B. Highscores). In unserem Fall sollten allerdings keine Spiele installiert sein. Aus diesem Grund empfehle ich, erst einmal sämtliche Schreibrechte zu entfernen (555), bis das Verzeichnis benötigt wird. Auch das Ändern des Besitzers in root sollte überlegt werden, da der Benutzer games alle Rechte wieder zurücksetzen kann.

Das Verzeichnis */etc/cups* gehört zum Druckdienst cups. Es ist für den Benutzer und die Gruppe lp schreibbar. Normal werden diese Rechte vermutlich zum Drucken benötigt. Wenn wir aber keinen Drucker an die Firewall angeschlossen haben, so ist dies überflüssig. Daher sollten auch hier die Rechte auf 555 gesetzt und das Verzeichnis an root übertragen werden.

Die Verzeichnisse unter */var/spool* dienen der Zustellung von E-Mails. Da Postfix deinstalliert wurde, können wir darüber nachdenken, */var/spool/postfix/maildrop* wie bereits besprochen zu sperren. */var/spool/clientmqueue* sollten wir dagegen lassen, wie es ist.

/etc

In */etc* befinden sich Konfigurationsdateien. Diese sollten normalerweise nur für root schreibbar sein. Es gibt keinen Grund, normalen Benutzern zu erlauben, Systemdienste umzukonfigurieren.

In meinen SuSE-Installationen war das von Anfang an der Fall. Es schadet aber nichts, mit

```
# find /etc -not -uid 0 -type f -printf "%-30p\t%u.%g\t%m\n"
```

selbst nachzusehen.

Spezielle Dateien

Nachdem wir uns bisher mit Programmen beschäftigt haben, wollen wir uns nun Dateien zuwenden, die zwar nicht ausführbar sind, aber dennoch großen Einfluß auf die Sicherheit des Systems haben. Dazu werden wir neben den Devices auch einmal einen Blick auf einige Konfigurationsdateien werfen, die gerne dazu mißbraucht werden, das Verhalten von Programmen im Sinne des Angreifers zu beeinflussen.

Devices

Devices erlauben den direkten Zugriff auf die Hardware des Systems. Es ist daher wichtig zu wissen, welche Devices existieren und wie die Zugriffsrechte für sie gesetzt sind.

Devices befinden sich normalerweise im Verzeichnis */dev/*. Wenn wir also Devices an anderer Stelle finden, so sollte uns das nachdenklich stimmen. Wir finden solche Devices mit folgendem Befehl:

```
# find / -path /dev -prune -o \( -type b -o -type c \) -ls
/var/lib/named/dev/null         root.root       666
/var/lib/named/dev/random       root.root       666
/lib/klibc/dev/console          root.root       600
/lib/klibc/dev/null             root.root       666
```

Werden Devices gefunden, so sollte genau nachgeforscht werden, warum dies der Fall ist. Denkbare Erklärungen wären z. B.:

1. Für einen Dienst ist ein `chroot`-Käfig aufgesetzt worden. Dabei müssen oft auch einige Devices angelegt werden. Wir sehen dies hier am Verzeichnis */var/lib/named/*. Es ist der `chroot`-Käfig für den DNS-Server Bind.
2. Bei der Neukompilation des Kernels soll automatisch eine Initial Ramdisk erzeugt werden. Dies ist ein Minimalsystem, das vor dem eigentlichen Start des Systems aktiv wird und z. B. Module lädt, die nötig sind, um das Wurzelverzeichnis mounten zu können. Das ist nötig, wenn man exotische Hardware benutzt und die nötigen Treiber als Modul kompiliert.

   Hier wird das Verzeichnis */lib/klibc* genutzt, um so ein System zusammenzustellen, wie ein Blick in das Skript `/sbin/mkinitrd` beweist. Dieses Skript wird bei der Installation eines Kernels unter SuSE von `/sbin/installkernel` aufgerufen.
3. Eine Bootdiskette sollte erzeugt werden. Dazu war es nötig, ein komplettes Dateisystem aufzubauen und zu bevölkern.

4. Wer auch immer die Distribution zusammengestellt hat, hat unsauber gearbeitet. Bei namhaften Distributionen ist dies aber eher unwahrscheinlich.
5. Ein erfolgreicher Angreifer hat von kritischen Devices wie z. B. *mem* oder *kmem* versteckte Kopien erzeugt und deren Rechte so verändert, daß auch normale Benutzer auf sie zugreifen dürfen. Das könnte es ihm erlauben, entzogene Rootrechte wiederzuerlangen, indem er z. B. an allen Sicherheitsvorkehrungen vorbei direkt auf den physikalischen Hauptspeicher zugreift.

Andererseits ist es auch verdächtig, falls normale Dateien unter */dev* gefunden werden. Normalerweise ist das Verzeichnis Spezialdateien vorbehalten. Allerdings hat es Fälle gegeben, wo Angreifer nach der Kompromittierung eines Systems einen Sniffer installierten und diesen dazu benutzten, Paßwörter auszuspähen. Die Datei mit den Ergebnissen ihrer Bemühungen speicherten sie dann unter */dev*, weil es auch für den Administrator normalerweise keinen Grund gibt, die Dateien dort regelmäßig zu untersuchen.

Natürlich ist nicht zu erwarten, in einem frisch installierten System solche verdächtigen Dateien zu finden. Trotzdem sollte man ruhig einmal nachsehen. Sollte sich dort doch eine Datei finden, bevor ein Angreifer eine Chance hatte, das System zu infiltrieren, so erspart man sich später Zeit und Nerven, wenn man von ihrer Existenz weiß.

```
# find /dev -type f -ls
178802 4 -rw-r--r-- 1 root root    4 Jul 27 18:49 /dev/.udevdb/udevd.pid
178804 4 -rw-r--r-- 1 root root   52 Jul 26 18:00 /dev/.udevdb/class@mem@mem
178805 4 -rw-r--r-- 1 root root  116 Jul 26 18:00 /dev/.udevdb/block@ram1
178806 4 -rw-r--r-- 1 root root  116 Jul 26 18:00 /dev/.udevdb/block@ram2
[...]
167906 0 -rw-r--r-- 1 root root    0 Jul 27 18:49 /dev/devperms
```

Die hier gefundenen Dateien gehören zum `udevd` und dienen dazu, Devices dynamisch anzulegen. Dieses Feature ist zwar momentan noch experimentell, wird aber von SuSE bereits eingesetzt.

Schließlich sollte man auch noch darauf achten, daß die Devices `root` gehören. Es existieren allerdings Ausnahmen von dieser Regel. Benutzt man ein Terminal (z. B. eine virtuelle Konsole) oder ein Pseudoterminal (z. B. ein `xterm`, `kvt`, Einwahl über eine serielle Schnittstelle), so ist es normal, daß der Besitzer des Devices auf den momentanen Benutzer geändert wird.

Im folgenden Beispiel sind z. B. mehrere Terminals (*/dev/ttyX*) und Pseudoterminals (*/dev/pts/X*) in Betrieb, auf denen ein Benutzer namens »user« angemeldet ist:

```
# find /dev  \( -type b -o -type c \) -not -uid 0  -ls
 96887    0 crw--w----   1 user  users      4,   0 Mar 11 11:11 /dev/tty0
 96899    0 crw--w----   1 user  tty        4,   2 Jun  2 22:44 /dev/tty2
 96908    0 crw--w----   1 user  users      4,   7 Mar 11 11:11 /dev/tty7
     2    0 crw--w----   1 user  tty      136,   0 Jun  2 23:04 /dev/pts/0
     3    0 crw--w----   1 user  tty      136,   1 Jun  2 23:37 /dev/pts/1
     4    0 crw--w----   1 user  tty      136,   2 Jun  2 22:53 /dev/pts/2
     5    0 crw--w----   1 user  tty      136,   3 Jun  2 22:44 /dev/pts/3
     6    0 crw--w----   1 user  tty      136,   4 Jun  2 22:44 /dev/pts/4
```

.exrc

Dateien mit dem Namen »*.exrc*« kommt eine besondere Rolle zu. Manche Editoren (insbesondere diverse `vi`-Varianten, aber je nach Konfiguration z. B. auch der Emacs) führen, wenn sie eine derartige Datei im aktuellen Verzeichnis finden, ihren Inhalt aus. Dies erlaubt es einem Angreifer, beliebige Kommandos mit den Rechten seines Opfers ausführen zu lassen.

Es ist allerdings durchaus normal, derartige Dateien im System vorzufinden:

```
# find / -name ".exrc"
/etc/skel/.exrc
/root/.exrc
/home/user/.exrc
```

In diesem Beispiel befindet sich eine Vorlage unter */etc/skel*, von wo sie in das Heimatverzeichnis eines jeden neuen Nutzers[5] kopiert wird. Dies ist durchaus normal und dient dazu, dem Benutzer eine komfortable Grundkonfiguration seines Editors zur Verfügung zu stellen. Allerdings sollte man sich die gefundenen Dateien einmal genauer ansehen und versuchen, ihren Inhalt zu verstehen.

Findet man allerdings *.exrc*-Dateien in anderen Verzeichnissen, die womöglich noch von diversen Nutzern schreibbar sind (z. B. */tmp*), so ist dies ein Warnsignal, und man sollte den Inhalt besonders genau prüfen.

.rhosts, /etc/hosts.equiv und /etc/hosts.lpd

Diese Dateien konfigurieren den Zugang fremder Rechner auf die R-Dienste und eventuell vorhandene Drucker. Obwohl es sich dabei um zwei verschiedene Dienste handelt, ist das Prinzip in beiden Fällen das gleiche. Über das Netz gestellte Anfragen werden nicht danach zugelassen, ob sich ein bestimmter Benutzer authentisieren kann, sondern es wird nur geprüft, ob der Rechner, der die Anfrage stellt, bekannt ist. Dies birgt die Gefahr, daß ein Angreifer, der einen autorisierten Rechner übernommen hat, damit automatisch Zugriff zu einer Vielzahl anderer Rechner erhält.

In den Dateien *.rhosts* und */etc/hosts.equiv* werden dabei Rechner angegeben, die so vertrauenswürdig sind, daß ihre Benutzer sich ohne Angabe eines Paßwortes mittel Hilfe der R-Dienste am System anmelden dürfen.

Während */etc/hosts.equiv* globale Regeln enthält, kann ein Benutzer auch selbst in seinem Heimatverzeichnis eine Datei namens *.rhosts* anlegen, in der Rechner angegeben sind, von denen aus er sich ohne Angabe eines Paßwortes anmelden möchte.

Die R-Dienste sollten auf einer Firewall unter keinen Umständen installiert sein, weswegen ihre Konfigurationsdateien nicht benötigt werden.

Die Datei */etc/hosts.lpd* dient hingegen dazu, Rechner festzulegen, die auf dem angeschlossenen Drucker drucken dürfen. Unter normalen Umständen sollte auch hierfür kein Grund vorliegen.

5 In dem betrachteten System sind `root` und `user` die einzigen Nutzer, sieht man einmal von einigen Nutzerkonten für Systemdienste ab.

Wenn wir uns nun wieder auf die Suche machen, stellen wir in unserem Beispielsystem fest, daß SuSE standardmäßig die Dateien */etc/hosts.equiv* und */etc/hosts.lpd* installiert hat:

```
# find / -name ".rhosts" -o -name "hosts.equiv" -o -name "hosts.lpd"
/etc/hosts.equiv
/etc/hosts.lpd
```

Wir können sie getrost löschen. Alternativ können wir auch leere Dateien erzeugen, als Besitzer `root` eintragen und jedermann den Zugriff verbieten:

```
# echo > /etc/hosts.equiv
# echo > /etc/hosts.lpd
# echo > /root/.rhosts
# chmod 0 /etc/hosts.equiv /etc/hosts.lpd /root/.rhosts
# chown root.root /etc/hosts.equiv /etc/hosts.lpd /root/.rhosts
```

Versteckte Dateien

Wie DOS kennt auch Unix versteckte Dateien. Während unter DOS ein spezielles Dateiattribut dazu benutzt wird, eine Datei zu verstecken, gilt unter Unix, daß viele Programme Dateien, deren Namen mit einem Punkt beginnen, nur anzeigen, wenn sie explizit dazu aufgefordert werden. So muß `ls` z. B. mit dem Parameter »-a« aufgerufen werden, wenn man die Auflistung derartiger Dateien wünscht.

Es hat nun verschiedentlich Fälle gegeben, wo Angreifer dies nutzten, um Dateien vor den Augen flüchtiger Betrachter zu verbergen. Besonders beliebt sind dabei Dateien wie ».. « (Punkt Punkt Leerzeichen) und »...« (Punkt Punkt Punkt). Aber auch »normale« Namen wie ».mail« oder ».xx« wurden schon benutzt.

Es empfiehlt sich daher, sich öfter einmal einen Überblick über die versteckten Dateien zu verschaffen, um ein Gefühl dafür zu bekommen, was normal ist, und frühzeitig zu erkennen, wenn etwas Ungewöhnliches geschieht.

Der folgende Befehl zeigt versteckte Dateien an. Dabei wird der Dateiname in spitze Klammern eingeschlossen. Sonderzeichen werden als druckbare Zeichen dargestellt:

```
# find / -name ".*" -exec echo \<\{\}\> \; | cat -vT
```

Automatisieren der Suche

Um sich die Sache etwas einfacher zu machen, kann man die Befehle, die wir in den vorangegangenen Unterabschnitten kennengelernt haben, auch zu einem Skript zusammenfassen. Das folgende Skript zeigt nicht nur offene Ports, laufende Prozesse und bedenkliche Dateien und Verzeichnisse an, auch die für die `inetd` konfigurierten Dienste werden dargestellt. Darüber hinaus wird nicht versucht, */proc* zu durchsuchen[6], und `less` wird dazu benutzt, die Ausgabe seitenweise anzuzeigen. Auch ist es so möglich, bei Bedarf wieder zurückzublättern:

6 Dies würde zu Fehlermeldungen führen und den Prozeß unnötig verlangsamen.

```
#!/bin/sh
#########################################################################
#
# confcheck
#
#     Dieses Skript listet sicherheitsrelevante Aspekte der
#     Konfiguration eines Rechners auf.
#
# Copyright (C) 2003 Andreas G. Lessig
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
#
#########################################################################

VERSION="**** CONFCHECK v0.23 ****"

ROOT=""

hiddenfiles(){
while read d
do
    for f in ${d}/.*
    do
        if test -e "$f"
        then
            echo "<${f}>"
        fi
    done | grep -v ".*/\\.>\$" \
    | grep -v ".*/\\.\\.>\$"
done
}


docheck(){
echo -e "\n\n$VERSION\n\n"

echo "---- Dienste des inetd ----"
echo

grep -v "^#" "$ROOT"/etc/inetd.conf

echo
echo "---- Offene Ports ----"
echo

netstat -ap | grep -v '^unix' | \
grep -v '^Active UNIX domain' | \
grep -v '^Proto RefCnt Flags'
```

```
echo
echo "---- Aktive Programme ----"
echo

ps ax -H

echo
echo "---- SUID/SGID - Programme ----"
echo

find "$ROOT"/ \( -path "$ROOT"/proc -prune \) -o  -perm +6000 \
-type f -printf "%-30p\t%u.%g\t%m\n"

echo
echo "---- Veränderbare Dateien ----"
echo

find "$ROOT"/ \( -path "$ROOT"/proc -prune \) -o  -perm +022 \
-type f -printf "%-30p\t%u.%g\t%m\n"

echo
echo "---- Unsichere Verzeichnisse ----"
echo

find "$ROOT"/ \( -path "$ROOT"/proc -prune \) -o  -type d \
-perm +022 -not -perm +1000 -printf "%-30p\t%u.%g\t%m\n"

echo
echo "---- Konfigurationsdateien, die nicht root gehören ----"
echo

find "$ROOT"/etc \( -path "$ROOT"/proc -prune \) -o \
-not -uid 0 -type f -printf "%-30p\t%u.%g\t%m\n"

echo
echo "---- Devices an ungewöhnlichen Orten ----"
echo

find "$ROOT"/ \( -path "$ROOT"/proc -prune \) \
-o \( -path "$ROOT"/dev -prune \) -o \( -type b -o -type c \) \
-printf "%-30p\t%u.%g\t%m\n"

echo
echo "---- Dateien unter /dev, die keine Devices sind ----"
echo

find /dev -type f -ls

echo
echo "---- Devices, die nicht root gehören ----"
echo

find "$ROOT"/dev \( -path "$ROOT"/proc -prune \) -o  \
\( -type b -o -type c \) -not -uid 0 \
-printf "%-30p\t%u.%g\t%m\n"
```

```
echo
echo "---- Spezielle Dateien ----"
echo

find "$ROOT"/ \( -path "$ROOT"/proc -prune \) -o \
\( -name ".exrc" -o -name ".rhosts" -o -name "hosts.equiv" \
-o -name "hosts.lpd" \) -printf "%-3Op\t%u.%g\t%m\n"

echo
echo "---- Versteckte Dateien ----"
echo

find "$ROOT"/ \( -path "$ROOT"/proc -prune \) -o -type d \
-print |hiddenfiles


}

docheck 2>/dev/null | less -s -S
```

Sollten Sie das abtippen, so beachten Sie bitte, daß einige Zeilen zu lang für den Druck waren. In diesen Fällen enden die Zeilen auf »\«. Dieses Zeichen steht als allerletztes in der Zeile. Es folgen keine weiteren Leerzeichen. Die Shell erkennt hieran, daß das Zeilenende von ihr nicht ausgewertet werden soll.

Dieses Skript sollten Sie nicht nur jetzt bei der Einrichtung Ihrer Firewall laufen lassen. Auch die Installation neuer Software oder ein Upgrade auf eine neuere Version Ihrer Linux-Distribution sollten Sie zum Anlaß nehmen, zu überprüfen, ob dadurch neue Risiken in Ihr System gekommen sind.

Es existieren allerdings auch Risiken, die Sie mit `confcheck` nicht entdecken werden. Dies gilt insbesondere, wenn ein Cracker in Ihr System eingedrungen ist und es geschafft hat, ein Rootkit zu installieren. In diesem Fall sind die hier verwendeten Befehle `ps`, `find` und `netstat` durch Versionen ausgetauscht worden, die keine Prozesse, Dateien oder Ports anzeigen, die auf den Angriff hindeuten. Hier hilft nur, ein Rettungssystem zu booten, das zumindest die Befehle `find`, `grep` und `less` enthält. Wenn möglich, sollten Sie das Rettungssystem verwenden, das zu Ihrer Distribution gehört, da ansonsten der einer Datei zugeordnete Benutzer bzw. ihre zugehörige Gruppe falsch benannt werden.

Um `confcheck` mit einer Bootdiskette einzusetzen, sind noch ein paar kleine Änderungen nötig. Als erstes müssen wir berücksichtigen, daß wir das Wurzelverzeichnis auf ein Unterverzeichnis unseres Rettungssystems mounten werden. Dadurch verändern sich alle Pfade. Aus diesem Grund existiert eine Variable, in der wir das Verzeichnis eintragen, auf das wir später die Rootpartition mounten werden. Üblicherweise besitzen Rettungssysteme für solche Zwecke ein Verzeichnis */mnt/*. Hier müssen wir die folgende Änderung vornehmen:

```
ROOT="/mnt"
```

`ps` und `netstat` brauchen wir nicht wirklich, da sie nur den momentanen Zustand des Rettungssystems anzeigen, nicht aber den des Rechners, wenn er normal gebootet wurde. Ihr Aufruf kann deshalb auch auskommentiert werden:

```
echo
echo "---- Offene Ports ----"
echo

# netstat -a | grep -v '^unix' | \
# grep -v '^Active UNIX domain' | \
# grep -v '^Proto RefCnt Flags'


echo
echo "---- Aktive Programme ----"
echo

# ps ax
```

Als nächstes kopieren wir das Skript auf eine Diskette. Dazu können wir den folgenden Befehl benutzen[7]:

```
# mcopy confcheck a:
```

Nun können wir den zu untersuchenden Rechner von der Rettungsdiskette booten. Nachdem das Rettungssystem gestartet ist, mounten wir die Diskette mit dem Skript sowie die Festplatte und führen dann das Skript aus. Achten Sie aber bitte unbedingt darauf, daß Sie die Festplatte schreibgeschützt mounten (ro) sowie die Ausführung von Programmen (noexec) und die Benutzung von Devices (nodev) auf der Festplatte verbieten.

Nehmen wir einmal an, unsere Festplatte besäße nur zwei Partitionen *hda1* und *hda2*, die auf */boot* bzw. */* gemountet würden. Ferner sind die Verzeichnisse */mnt* und */floppy* vorhanden und leer.[8] Dann könnten Sie nach dem Booten des Rettungssystems folgende Befehle benutzen:

```
# mount /dev/fd0 /floppy -t vfat
# mount /dev/hda2 /mnt -o ro,nodev,noexec
# mount /dev/hda1 /mnt/boot -o ro,nodev,noexec
# cd floppy
# ./confcheck
```

Nun können Sie sich zumindest einen gewissen Überblick über den Zustand der Dateien auf der Festplatte und die Rechte dieser Dateien verschaffen. Leider ist es nicht möglich, verläßliche Aussagen über die aktiven Prozesse in einem System zu machen, wenn nicht bekannt ist, ob ein Rootkit installiert wurde oder gar spezielle Module geladen wurden, die es erlauben, Prozesse auf Kernelebene zu verstecken. Immerhin könnte man das System auf offene Ports untersuchen, indem man von einem anderen Rechner aus Port Scans durchführt. Details hierzu finden Sie in Kapitel 15, Abschnitt *Port Scans*, ab Seite 446.

7 Die Benutzung von `mcopy` setzt voraus, daß es sich um eine Diskette handelt, die für das FAT-Dateisystem formatiert wurde (handelsübliche, »IBM formatted« Diskette).

8 Gegebenenfalls können diese auch mit `mkdir` *Verzeichnisname* erzeugt werden.

Das Systemprotokoll

Stößt ein interaktives Programm auf einen Fehler, so gibt es normalerweise eine Meldung auf den Bildschirm aus. Hintergrundprozesse und Systemdienste verfügen nicht über diese Möglichkeit und benutzen daher einen anderen Mechanismus. Sie senden über einen speziellen Aufruf eine Meldung, die dann von einem zentralen Protokolldienst entgegengenommen und in eine Datei abgelegt oder an einen anderen Rechner weitergeleitet wird.

So wie dieser Dienst Protokollmeldungen an andere Rechner weiterleiten kann, so kann er auch Meldungen von anderen Rechnern entgegennehmen. Dann erwartet er auf Port 514 UDP-Meldungen, die er wie ihre lokalen Gegenstücke in das Systemprotokoll einträgt. Unsere Firewall sollte allerdings keine Logmeldungen von anderen Rechnern annehmen, da sonst die Gefahr bestünde, daß ein Angreifer das Systemprotokoll mit unsinnigen Meldungen füllt, so daß sein folgender Angriff unprotokolliert bliebe.

Lokale Meldungen an den Protokolldienst werden nicht über das Netz geschickt, sondern über einen sogenannten Unix Domain Socket. Dieser wird duch eine Art spezieller Datei repräsentiert, die vom Protokolldienst angelegt wird. Üblicherweise wird hier */dev/log* verwendet. Allerdings kann es vorkommen, daß weitere Sockets benötigt werden, wenn z. B. ein Prozess in einen `chroot`-Käfig gesperrt wird und auf den normalen Socket nicht mehr zugreifen kann.

Was die Protokollierung der Meldungen angeht, so verwaltet der Protokolldienst normalerweise nicht nur eine Protokolldatei, sondern mehrere, auf welche die Meldungen je nach Quelle und Wichtigkeit verteilt werden. Dazu kennt das Protokoll vordefinierte Quellen und Prioritäten.

An Quellen existieren u. a.:

authpriv Sicherheitsrelevante Meldungen

cron Meldungen des Cron- oder At-Daemons

kern Kernelmeldungen

mail Mailserver

syslog Meldungen des Protokolldienstes selbst

user Generische Quelle für von normalen Anwendungen generierte Meldungen. Dies ist auch die Vorgabe, wenn beim Einstellen der Meldung nicht explizit eine andere Quelle gewählt wurde.

Dies sind aber nur die Quellen, die für unseren Fall relevant sind. Es existiert noch eine Reihe weiterer Quellen, die man sich mit dem Befehl `man 3 syslog` anzeigen lassen kann.

Neben den einzelnen Quellen unterscheidet der Protokolldienst auch zwischen Meldungen unterschiedlicher Priorität. Diese Prioritätslevel reichen von `debug` (Information zur Suche von Programmierfehlern) über `info` (zur Kenntnisnahme), `notice` (unkritisch, aber von Bedeutung), `warn` (Warnung), `err` (Fehler), `crit` (kritischer Zustand), `alert` (sofortiges Handeln nötig) bis zu `emerg` (das System ist unbenutzbar).

Wie schon bei den Mailservern, so gibt es auch für den Protokolldienst mehrere Implementationen. Debian 3.1 verwendet standardmäßig den klassischen `syslog`. SuSE 9.3 ist dagegen auf den moderneren `syslog-ng` umgestiegen. Für unsere Zwecke sind beide brauchbar. Wir wollen sie uns daher im folgenden beide etwas genauer ansehen.

syslog

Der `syslogd` kennt eine Reihe von Kommandozeilenparametern, von denen für uns dabei insbesondere die folgenden wichtig sind:

-a Socket erlaubt es einen weiteren Socket anzugeben, auf dem der Dienst Meldungen entgegennimmt. Wir werden diesen Parameter später noch brauchen, wenn wir `chroot`-Käfige bauen.

-h schaltet das Weiterleiten von Meldungen an andere Rechner an. Ohne diesen Parameter werden keine Pakete gesendet.

-m Intervall gibt an, alle wieviel Minuten der `syslogd` einen eigenen Eintrag in das Protokoll schreiben soll, der beweist, dass er noch läuft. Dies ist sinnvoll, da man so im Fall eines Rechnerausfalls oder anderer Probleme eine ungefähre Vorstellung hat, wann der Rechner noch lief. Die Voreinstellung ist 20. Wird 0 angegeben, so werden keine zusätzlichen Einträge in das Systemprotokoll geschrieben.

-r aktiviert die Entgegennahme von Nachrichten über das Netzwerk.

Weitere Parameter findet man auf der Manpage des `syslogd`.

Konfiguriert wird der `syslogd` über die Datei */etc/syslog.conf*. Ein anderer Name kann im Bedarfsfall mit der Option »-f« angegeben werden. Diese Datei enthält Einträge der Art:

Quelle.Level[;Quelle.Level] Ziel

Eine typische Konfiguration, wie sie mit einer Distribution mitkommt, sieht z. B. folgendermaßen aus:

```
kern.warn;*.err;authpriv.none    /dev/tty10
kern.warn;*.err;authpriv.none   |/dev/xconsole
*.emerg                          *
news.crit                       -/var/log/news/news.crit
news.err                        -/var/log/news/news.err
news.notice                     -/var/log/news/news.notice
mail.*                          -/var/log/mail
*.=warn;*.=err                  -/var/log/warn
*.crit                           /var/log/warn
*.*;mail.none;news.none         -/var/log/messages
```

Hier sind auch die meisten der möglichen Konstrukte verwendet worden. Beginnen wir mit den Möglichkeiten, eine Quelle zu spezifizieren. Hier wurden `kern`, `authpriv`, `news` und `mail` verwendet. Will man Nachrichten aller Quellen spezifizieren, kann man dies mit »*« tun.

Neben den einzelnen Quellen unterscheidet der `syslogd` auch zwischen Meldungen unterschiedlicher Priorität. Alle Meldungen der angegebenen oder einer höheren Priorität

werden protokolliert. Auch hier kann mit »*« angegeben werden, daß alle Meldungen unabhängig von ihrer Priorität gemeint sind. Ist der Priorität ein Gleichheitszeichen vorangestellt, so bezieht sich die Angabe ausschließlich auf Meldungen der vorgegebenen Priorität. So existiert im oben dargestellten Beispiel eine Regel, wonach Meldungen der Prioritäten `warn` und `err` in */var/log/warn* protokolliert werden sollen. Meldungen der Priorität `crit` und höher werden von dieser Regel nicht erfaßt, sondern erst in der nächsten Stufe der Prioritäten behandelt.

Eine Sonderrolle spielt dabei die Priorität `none`. Sie dient dazu, Meldungen einer bestimmten Quelle nicht an das angegebene Ziel zu senden. Im obigen Beispiel wird in der letzten Zeile z. B. angegeben, daß alle Nachrichten nach */var/log/messages* geschrieben werden sollen, mit Ausnahme der Meldungen von Mail- und Newsserver.

Als letztes wird schließlich angegeben, wohin eine Nachricht protokolliert werden soll. Dabei kann es sich um eine normale Datei handeln (z. B. */var/log/messages*). Es ist aber auch möglich, eine Named Pipe anzugeben, die durch das Voranstellen eines »|« gekennzeichnet wird. Soll die Nachricht an einen anderen Rechner weitergeleitet werden, so wird dieser in der Form *@Rechner.Domäne* angegeben.

Ein vorangestelltes Minuszeichen bewirkt, daß nach dem Schreiben in die Datei nicht automatisch die Systemfunktion `sync()` aufgerufen wird. Damit wird das eigentliche Schreiben der Daten auf die Platte verzögert. Hierdurch steigt die Geschwindigkeit, mit der die Protokollierung erfolgt, da weniger der langsamen Festplattenzugriffe nötig sind. Allerdings bedeutet dies auch, daß im Falle eines plötzlichen Rechnerausfalls noch nicht gesicherte Daten verlorengehen. Man sollte daher sorgfältig abwägen, ob für den betrachteten Meldungstyp ein solches Vorgehen sinnvoll ist.

Wird als Ziel ein »*« angegeben, so wird die Meldung auf den Konsolen aller angemeldeten Benutzer angezeigt.

Wollen wir nun das Logging auf unserer Firewall konfigurieren, so ist obiges Beispiel eigentlich zu kompliziert. Wir brauchen weder mehrere Protokolldateien für einen Mail- oder Newsserver noch ein Logging auf eine X-Console. Auch sollte nach Warn- und Fehlermeldungen grundsätzlich ein `sync()` durchgeführt werden.

Eine sinnvolle Konfiguration könnte daher so aussehen:

```
kern.warn;*.err;authpriv.none    /dev/tty10
*.emerg                          *
*.warn                           /var/log/warn
*.*                              /var/log/messages
#*.*                             @loghost.mydomain
```

Hier werden wie wieder besonders wichtige Meldungen auf die zehnte virtuelle Konsole ausgegeben. Letzte Meldungen vor dem Exitus des Systems werden zusätzlich auf allen benutzten Konsolen angezeigt. Eine spezielle Datei */var/log/warn* erhält alle Meldungen, die über die normale Routine hinausgehen. In */var/log/messages* werden schließlich grundsätzlich alle Nachrichten gesammelt. Sollen die Meldungen an einen speziellen Protokollrechner geschickt werden, so brauchen Sie nur die letzte Zeile auszukommentieren und den Platzhalter durch den Namen des Protokollrechners zu ersetzen.

Um die geänderten Einstellungen wirksam werden zu lassen, sollten Sie dem `syslogd` das Signal `SIGHUP` senden (siehe Kapitel 9, Abschnitt *Der inetd*, ab Seite 181).

syslog-ng

Im Gegensatz zum `syslog` hat der `syslog-ng` eine viel ausgefeiltere Konfiguration, die es erlaubt detailliert zu filtern. Leider bedeutet dies auch, daß jeder Versuch, die Konfiguration vollständig zu erläutern, den Rahmen dieses Buches sprengen würde. Unter SuSE finden Sie aber das recht umfangreiche Handbuch des `syslog-ng` unter */usr/share/doc/packages/syslog-ng/*.

Eine recht gute Ergänzung dazu ist das Kapitel über die Konfiguration des Systemprotokolls in [10]. Dieses Kapitel kann auch als Probekapitel unter *http://www.oreilly.com/catalog/linuxss2/* heruntergeladen werden.[9] Dort werden einige Details erklärt, die im Handbuch der Software eindeutig zu kurz kommen oder fehlen.[10]

Konfiguration

Wir wollen uns hier auf eine simple Konfiguration beschränken, die der schon für den `syslog` besprochenen entspricht. Sie ist deutlich simpler als die standardmäßig mit SuSE mitgelieferte. Die Standardkonfiguration kennt eine Vielzahl von Logdateien für alle Arten von Meldungen. Im Gegensatz dazu wollen wir hier sicherstellen, daß alle Meldungen in einer Datei landen, damit wir später nicht erst lange nach der richtigen Logdatei suchen müssen.

Bevor wir aber anfangen, müssen wir noch eine Besonderheit unter SuSE beachten. Während man normalerweise direkt die Datei */etc/syslog-ng/syslog-ng.conf* editieren würde, wird diese Datei unter SuSE von `SuSEconfig` automatisch aus der Quelldatei */etc/syslog-ng/syslog-ng.conf.in* generiert.

Normalerweise merkt `SuSEconfig`, wenn wir die Datei direkt editieren, und überschreibt sie nicht. Wir sollten aber nichts riskieren und ihm explizit den Zugriff auf die Datei verbieten. Dies geschieht in der Datei */etc/sysconfig/syslog*. Hier müssen wir die Variable *SYSLOG_NG_CREATE_CONFIG* ändern, so daß dort steht:

```
SYSLOG_NG_CREATE_CONFIG="no"
```

Da die SuSE-Datei für unsere Zwecke zu kompliziert ist, fangen wir von Grund auf an und benennen die Datei *syslog-ng.conf* um. Nun öffnen wir eine neue Datei, die wir ganz nach unseren Wünschen gestalten können.

Als erstes können wir einige Optionen konfigurieren, die bestimmen, wie der Dienst sich verhält:

9 Sie können natürlich auch das ganze Buch kaufen . . . `;-)`. Im Ernst, es ergänzt dieses Buch recht gut, und das sage ich nicht nur, weil mein Buch im selben Verlag erschienen ist.

10 Zum Beispiel was die Option `chain_hostname` macht, oder daß man den Dienst in einem `chroot`-Käfig betreiben kann.

```
options {
        chain_hostnames(no);
        sync(0);
        perm(0640);
        create_dirs(no);
        stats(3600);
};
```

Die Optionen bedeuten dabei im einzelnen:

chain_hostnames gibt an, ob bei einer Nachricht, die über mehrere Rechner weitergeleitet wurde, jeweils die bekannten Zwischenstationen angegeben werden sollen. Das bedeutet, wenn ein Rechner in der Nachricht als Quelle angegeben ist, die Meldung aber über das Netz von einem anderen Rechner empfangen wurde, so wird der Netzwerkname des Senders durch einen Schrägstrich getrennt angehängt. Passiert eine Nachricht mehrere `syslog-ng`-Server, die jeweils die Meldung an einen anderen Server weiterreichen, dann erhält man mit dieser Option eine Meldung der Art

Label@Rechner1/Rechner2/...

Label ist hierbei die Bezeichnung, die wir für unsere Quelle gewählt haben. Was das genau ist, dazu kommen wir gleich.

Wir werden auf der Firewall keine Meldungen aus dem Netzwerk annehmen, aus diesem Grund benötigen wir diese Option nicht und haben sie durch Angabe von `no` ausgeschaltet.

Früher hieß diese Option übrigens `long_hostnames`. Diese Form funktioniert immer noch, sollte aber nicht mehr benutzt werden.

sync gibt an, wieviele Zeilen zwischengespeichert werden sollen, bevor sie tatsächlich in die Zieldatei geschrieben werden. `0` sorgt hier dafür, daß keine Zwischenspeicherung erfolgt.

perm gibt die Berechtigungen vor, die benutzt werden, wenn `syslog-ng` eine Datei anlegt. Grundsätzlich legt der `syslog-ng` Dateien immer dann an, wenn sie als Ziele angegeben sind, im System aber noch nicht existieren.

create_dirs gibt an, ob neben Dateien auch fehlende Verzeichnisse angelegt werden sollen. Dies ist aber in unserem Fall unnötig. Wir werden unsere Dateien direkt unter */var/log* anlegen.

stats legt die Anzahl der Sekunden fest, nach denen der `syslogd` einen Eintrag mit statistischen Informationen erzeugt. In der Regel steht dort nur `STATS: dropped 0`. Der Eintrag kann aber dabei helfen, herauszufinden, wann der Rechner im Fall eines Hardware-Defekts noch funktioniert hat. Man kann die Meldung also als »Ich lebe noch!« interpretieren.

Lassen wir den `syslog-ng` anlegen, so können wir auch dafür Rechte angeben. Läuft er mit `root`-Rechten, so können wir darüber hinaus festlegen, daß er die Dateien mit einem anderen Besitzer und einer anderen Gruppe als der aktuellen anlegen soll.

Dazu dienen die folgenden Optionen:

dir_owner erhält als Argument die UID neu angelegter Verzeichnisse, also die numerische Angabe des gewünschten Besitzers.

dir_group erhält die GID der Gruppe, der neu angelegte Verzeichnisse gehören sollen.

dir_perm entspricht *perm*, gilt aber für Verzeichnisse.

owner gibt die UID für neu angelegte Dateien vor.

group bestimmt, welcher GID eine Datei zugeordnet sein soll.

Nun müssen wir festlegen, aus welchen Quellen wir Meldungen entgegennehmen wollen. Normalerweise sieht dies folgendermaßen aus:

```
source src {
        internal();

        unix-dgram("/dev/log");
        #unix-dgram("/var/lib/named/dev/log");
        #unix-dgram("/var/lib/ftp-proxy/rundir/dev/log");

        #udp(ip("0.0.0.0") port(514));

        #file("/proc/kmsg" log_prefix("kernel: "));
};
```

Hier definieren wir eine Gruppe von Quellen mit der Bezeichnung `src`. Dies ist auch die Bezeichnung, die bei der Verwendung von `chain_hostnames` im Systemprotokoll erscheint.

Diese Gruppe faßt eine ganze Reihe von Quellen zusammen:

internal wird für die Meldungen benutzt, die vom `syslog-ng` selbst ausgehen.

unix-dgram gibt einen klassischen Unix Domain Socket für Meldungen an, die lokal auf dem Rechner gesendet werden. Verwenden Sie noch einen 2.2er Kernel, so sollten Sie statt dessen `unix-stream` benutzen.

Hier ist sowohl ein Eintrag für den normalen Socket */dev/log* vorhanden als auch auskommentierte Einträge für Sockets in `chroot`-Käfigen für den DNS-Server Bind und den `ftp-proxy`.

udp liest Meldungen aus dem Netz. Ohne Parameter nimmt er Pakete von jedem Rechner entgegen und bindet sich an Port 514. Mit `ip` kann man gezielt die Adresse eines Netzwerk-Interfaces angeben, auf dem Nachrichten angenommen werden sollen. Mit `port` kann man den zu verwendenden Port vorgeben. Hier wurden allerdings nur die Standardvorgaben noch einmal explizit aufgeschrieben, so daß man ebenso einfach `udp()` schreiben könnte.

file Ist für die Datei gedacht, in die der Kernel seine Meldungen schreibt. Unter Linux ist dies */proc/kmsg*. Allerdings läuft unter SuSE ein Dienst namens `klogd`, der diese Datei ausliest und normal über */dev/log* an den `syslog-ng` weiterleitet. Da macht es natürlich keinen Sinn, wenn sich `syslog-ng` und `klogd` in die Quere kommen. Des-

halb ist dieser Eintrag auskommentiert. Installiert man den `syslog-ng` unter Debian, so wird der `klogd` deinstalliert. Hier müssen Sie das Kommentarzeichen entfernen.

Die Option `log_prefix` erlaubt es, der Meldung einen beliebigen Text voranzustellen. Hier nutzen wir diese Funktion, um Meldungen des Kernels kenntlich zu machen. Die Option kann aber mit jeder Quelle verwendet werden.

Normale Dateien darf man hier übrigens nicht angeben. Diese Quelle dient nur dazu, Kernelmeldungen zu lesen.

Nun können wir Filter definieren, die auf bestimmte Nachrichten passen. Wollen wir einfach nach angegebener Quelle und Priorität filtern, so sieht das folgendermaßen aus:

```
filter red_alert {
        priority(emerg);
};
filter error {
        priority(err .. emerg);
};
filter warning {
        priority(warn) or filter(error) ;
};
filter interesting {
        facility(kern) and filter(warning) or
        not facility(authpriv) and filter (error);
};
```

Die Fähigkeiten von `syslog-ng` gehen aber weit darüber hinaus, nur nach Quellen und Prioritäten zu filtern. Zum Beispiel bezeichnet der folgende Ausdruck Nachrichten, die von `iptables` erstellt wurden:

```
filter f_iptables {
        facility(kern) and match("IN=") and match(ÖUT=");
};
```

Es gibt noch weitere Filterfunktionen. Hier eine kleine Aufstellung:

facility prüft gegen die angegebene Quelle. Mehrere Werte können mit Kommata getrennt angegeben werden, z. B. `facility(mail, news)`.

level spezifiziert eine Priorität. Mehrere Prioritäten können mit Kommata getrennt angegeben werden, z. B. `priority(warn, err, crit)`. Man kann aber auch Bereiche angeben, indem man den niedrigsten und höchsten Wert durch zwei Punkte getrennt angibt, z. B. `priority(warn .. crit)`.

priority ist identisch zu `level`.

program erhält als Argument einen regulären Ausdruck, der auf das Feld für den Programmnamen passen muß.

host enthält einen regulären Ausdruck für den angegebenen Rechnernamen, der die Nachricht gesendet hat.

match gibt einen regulären Ausdruck für den Inhalt der Nachricht an.

filter erlaubt es, einen bereits definierten Filter zu benutzen.

Die Nachrichten müssen nun aber auch noch ausgegeben werden. Dazu müssen wir Dateien oder andere Rechner angeben, die unsere Nachrichten erhalten sollen.

```
destination logconsole {
        file("/dev/tty10" group(tty) perm(620));
};
destination root_tty    {
        usertty(root);
};
destination warnings   {
       file("/var/log/warn");
};
destination stdlog {
       file("/var/log/messages");
#      udp("loghost.mydomain" port(514));
};
```

Hier haben wir nur Destinations vom Typ `file` oder `udp` verwendet. Es gibt aber tatsächlich noch eine ganze Reihe zusätzlicher Methoden, Nachrichten auszugeben:

file gibt eine Datei oder ein Terminal an. Es existiert eine ganze Reihe von Makros, die es z. B. erlauben, Logdateien anzulegen, in deren Namen das aktuelle Datum vorkommt.[11] Auf diese Weise hat man jeden Tag ein neues Logfile, das automatisch bei Bedarf angelegt wird. Eine genaue Beschreibung der Makros würde allerdings den Rahmen sprengen. Ich möchte daher auf die eingangs erwähnten Dokumentationen hinweisen.

Die Optionen `owner`, `group`, `perm`, `create_dirs`, `dir_perm`, `dir_owner` und `dir_group`, die wir als globale Optionen kennengelernt haben, können auch zusammen mit `file` verwendet werden. Eine solche spezielle Angabe hat dabei Vorrang vor einer global unter `options` definierten.

Im Falle eines Terminals benötigen wir insbesondere `group` und `perm`, da `syslog-ng` beim Schreiben auf ein Terminal grundsätzlich die Rechte neu setzt. Ein Terminal hat aber standardmäßig besondere Rechte. Es ist der Gruppe `tty` zugeordnet und ist nur für den Besitzer lesbar und schreibbar. Die Gruppe `tty` darf schreiben, aber nicht lesen, der Rest der Welt hat keine Rechte. Ohne die expliziten Angaben im obigen Beispiel würde `syslog-ng` das Device der Gruppe `root` zuordnen und dieser Gruppe Lese- statt Schreibrecht geben.

pipe erlaubt es, in eine Named Pipe zu schreiben. Das prominenteste Beispiel für diesen Dateityp ist */dev/xconsole*. Diese Pipe kann dann mit einem anderen Programm ausgelesen und weiterverarbeitet werden.

unix-stream, unix-dgramm erlauben es, Unix Domain Sockets zu verwenden.

udp, tcp sorgen dafür, daß die Nachricht an den angegebenen Protokollserver weitergesandt wird. Mit der Option `port` kann ein Port angegeben werden. Wird dieser ausgelassen, so wird Port 514 (Syslog) verwendet.

11 Man könnte etwa */var/log/messages-$YEAR-$MONTH-$DAY* benutzen. Auch Konstruktionen mit `$FACILITY` oder `$LEVEL` bzw. `$PRIORITY` könnten im Einzelfall Sinn machen.

usertty erhält einen Benutzernamen als Argument. Meldungen werden auf alle Terminals gesendet, die dem angegebenen Benutzer zugeordnet sind.

program hat nur einen Parameter. Dabei handelt es sich um den Namen eines Programms, das gestartet wird. Die Meldungen erhält es über seine Standardeingabe. Das Programm wird unter Zuhilfenahme der aktuellen Shell gestartet, so daß es möglich ist, Wildcards und Ein-/Ausgabe-Umlenkungen zu benutzen. Beachten Sie aber, daß das Programm nur einmal gestartet wird und dann aktiv bleibt, bis `syslog-ng` beendet wird oder das Signal `HUP` erhält.

Schließlich müssen wir noch die einzelnen Bausteine zusammensetzen. Dies geschieht mit `log`:

```
log { source(src); filter(interesting); destination(logconsole); };
log { source(src); filter(red_alert); destination(root_tty); };
log { source(src); filter(warning); destination(warnings); };
log { source(src); destination(stdlog); };
```

Hierbei werden jeweils alle Nachrichten, die aus einer der angegebenen Quellen empfangen wurden und auf die alle Filter passen, an alle angegebenen Ziele geschickt. Sind keine Filter angegeben, so wird jede Nachricht weitergeleitet.

Grundsätzlich wird jede passende `log`-Zeile verwendet. Eine Meldung kann daher auch mehrfach an das gleiche Ziel gesendet werden, falls es in mehreren Zeilen vorkommt.

Wollen wir dies nicht, so können wir die Option `flags` verwenden. Wollen wir etwa, daß Meldungen mit der Priorität `warning` und höher nur in */var/log/warn* protokolliert werden, nicht aber in *var/log/messages*, so könnten wir bestimmen, daß keine weiteren Regeln betrachtet werden, falls die Regel für Warnings greift:

```
log { source(src); filter(warning); destination(warnings); flags(final)};
```

Man kann auch mehrere durch Kommata getrennte Flags angeben. Die folgenden Flags kennt `syslog-ng`:

final Greift die Regel, so werden für diese Meldung keine weiteren `log`-Zeilen betrachtet.

fallback Diese `log`-Zeile wird nur beachtet, wenn keine der `log`-Zeilen paßt, die nicht als `fallback` markiert sind.

catchall Welche Quellen angegeben sind, spielt keine Rolle. Meldungen aus allen Quellen werden betrachtet.

flowcontrol Wenn dieses Flag aktiv ist, hört der `syslog-ng` auf, Meldungen anzunehmen, falls das Ziel sie nicht schnell genug abarbeiten kann. Ohne dieses Flag werden die Nachrichten entgegengenommen, gehen aber verloren, wenn das Ziel zu langsam ist.

Hier nun noch einmal das komplette Skript:

```
options {
        chain_hostnames(no);
        sync(0);
        perm(0640);
        create_dirs(no);
        stats(3600);
};

source src {
        internal();

        unix-dgram("/dev/log");
        #unix-dgram("/var/lib/named/dev/log");

        #udp(ip("0.0.0.0") port(514));

        #file("/proc/kmesg" log_prefix("kernel: "));
};

filter red_alert {
        priority(emerg);
};
filter error {
        priority(err .. emerg);
};
filter warning {
        priority(warn) or filter(error) ;
};
filter interesting {
        facility(kern) and filter(warning) or
        not facility(authpriv) and filter (error);
};

destination logconsole {
        file("/dev/tty10" group(tty) perm(620));
};
destination root_tty    {
        usertty(root);
};
destination warnings   {
        file("/var/log/warn");
};
destination stdlog {
        file("/var/log/messages");
#       udp("loghost.mydomain" port(514));
};

log { source(src); filter(interesting); destination(logconsole); };
log { source(src); filter(red_alert); destination(root_tty); };
log { source(src); filter(warning); destination(warnings); };
log { source(src); destination(stdlog); };
```

chroot-Käfig

Auch wenn wir den `syslog-ng` nicht so konfigurieren, daß er Verbindungen von anderen Rechnern annimmt, so kann doch zumindest jeder angemeldete Benutzer beliebige Meldungen an ihn schicken[12]. Gelingt es dabei, einen Programmierfehler auszunutzen, dann bestünde die Möglichkeit, Befehle mit den Rechten des Dienstes abzusetzen. Da er standardmäßig mit `root`-Rechten läuft, könnte so ein Angriff dazu führen, daß ein Angreifer mit normalen Benutzerrechten plötzlich zu `root` wird.

Um solche Angriffe zu verhindern, kann man den `syslog-ng` anweisen, nach dem Lesen der Konfigurationsdatei und dem Öffnen der Quellen alle Rechte abzugeben und nur noch unter einem normalen Benutzerkonto ohne besondere Rechte zu arbeiten. Darüber hinaus sperrt sich der Dienst in einen `chroot`-Käfig ein. D. h., für ihn wird ein Unterverzeichnis zum neuen Wurzelverzeichnis. Alle Dateien außerhalb dieses Verzeichnisses existieren für ihn nicht mehr. Erst danach nimmt er Meldungen entgegen und öffnet Zieldateien.

Wir müssen uns aber auch im klaren sein, daß dieses Mehr an Sicherheit auch mit Einschränkungen in der Funktion erkauft wird. So funktioniert `usertty` nicht mehr als Ziel, da der Dienst nicht feststellen kann, welches Terminal welchem Benutzer zugeordnet ist.

Auch können wir den Dienst nicht einfach durch Senden eines `HUP`-Signals dazu auffordern, seine Konfigurationsdatei neu einzulesen. Zwar können wir die Konfigurationsdatei in seinen `chroot`-Käfig kopieren, aber einige der darin aufgeführten Quellen liegen außerhalb des Käfigs und können daher nicht mehr geöffnet werden. Aus diesem Grund müssen wir den Dienst wirklich beenden und neu starten.

Wenn Sie diese Aussichten nicht schrecken, dann können wir jetzt damit anfangen, die nötigen Vorbereitungen zu treffen.

Zuerst brauchen wir eine Gruppe und einen Benutzer, mit deren Rechten der Dienst arbeiten wird:

```
# groupadd syslogng
# useradd -c "Syslog NG Daemon" \
> -d /var/lib/syslog-ng \
> -g syslogng -G '' \
> -s /bin/false \
> syslogng
```

Dieser Benutzer ist nur der Gruppe `syslogng` zugeordnet und hat als Shell `/bin/false` eingetragen. Letzteres wird wahrscheinlich dazu führen, daß `program` als Zielangabe nicht mehr funktioniert. Bei dieser Art von Ziel versucht der Dienst, seine Shell aufzurufen, startet statt dessen aber immer ein Programm, das nichts anderes tut, als sich sofort zu beenden. Dabei liefert es einen Rückgabewert, der anzeigt, daß ein Fehler aufgetreten ist.

Benötigen Sie ein Ziel vom Typ `program`, so müssen Sie `/bin/false` durch `/bin/sh` ersetzen. Sie sollten dies aber nur tun, wenn es unbedingt notwendig ist.

12 Dazu dient zum Beispiel das Programm `logger`.

Weiterhin benötigen wir ein Unterverzeichnis, in dem wir den syslog-ng einsperren können. Wir nehmen einmal */var/lib/syslog-ng*. Unter diesem Verzeichnis müssen wir die Teile des Dateisystems nachbilden, die für syslog-ng relevant sind.

Beginnen wir mit dem Verzeichnis für die Logdateien:

```
# cd /var/lib
# mkdir -m 0755 -p syslog-ng/var/log
# cd syslog-ng
# chown root.syslogng var/log
# chmod g+w var/log
```

Durch den Parameter -p legt mkdir nicht nur das Verzeichnis *log* an, sondern auch alle fehlenden übergeordneten Verzeichnisse.

Das eigentliche Verzeichnis *log* ordnen wir der Gruppe syslogng zu. Diese Gruppe hat Schreibrecht auf dem Verzeichnis. Dieses Vorgehen ist notwendig, damit syslog-ng in dem Verzeichnis seine Ausgabedateien anlegen kann. Zu dem Zeitpunkt, wo er zu schreiben versucht, hat er seine root-Rechte bereits aufgegeben.

Um die Übersicht nicht zu verlieren, sollten wir noch einen symbolischen Link aus */var/log* auf unser neues Log-Verzeichnis anlegen:

```
# ln -s /var/lib/syslog-ng/var/log /var/log/syslog-ng
```

Wenn wir Ausgaben auf die virtuellen Konsolen machen wollen, dann müssen wir die entsprechenden Devices anlegen:

```
# mkdir dev
# for n in 1 2 3 4 5 6 7 8 9 10
> do
> mknod dev/tty$n c 4 $n
> chmod 0620 dev/tty$n
> chown root.syslogng dev/tty$n
> done
```

Schließlich müssen wir noch das Verzeichnis *etc* anlegen und mit zusätzlichen Konfigurationsdateien füllen. Zuerst einmal legen wir das Verzeichnis selbst an:

```
# mkdir etc
```

Als nächstes wollen wir, daß der Dienst weiß, in welcher Zeitzone er sich befindet. Nur so können wir sicherstellen, daß wir halbwegs sinnvolle Zeitangaben in unseren Protokollen finden. Hierzu müssen wir ihm die Datei */etc/localtime* zugänglich machen.

Unter SuSE ist dies eine normale Datei, die wir einfach kopieren können:

```
# cp -p /etc/localtime etc/
```

Unter Debian ist es ein symbolischer Link:

```
# ls -l /etc/localtime
lrwxrwxrwx 1 root root 33 Feb 17 21:26 /etc/localtime ->
/usr/share/zoneinfo/Europe/Berlin
```

Dieser Link zeigt nach */usr/share/zoneinfo*. Wir kopieren also als erstes den Link, wobei wir darauf achten, cp mit dem Parameter -d anzuweisen, den Link als Link zu kopieren und nicht etwa aufzulösen:

```
# cp -dfp /etc/localtime etc
```

Nun müssen wir nur noch die eigentliche Datei kopieren:

```
# mkdir -p -m0755 usr/share/zoneinfo/Europe
# cp -dfp /usr/share/zoneinfo/Europe/Berlin usr/share/zoneinfo/Europe/
```

Wollen wir auch Nachrichten über das Netzwerk annehmen, so sollten wir noch einige Dateien kopieren, die es dem Dienst erlauben, Namen aufzulösen. Nehmen wir nur Nachrichten über */dev/log* entgegen, so ist dieser Schritt nicht nötig:

```
# cp -dfp /etc/resolv.conf etc
# cp -dfp /etc/hosts etc
# cp -dfp /etc/services etc
# cp -dfp /etc/protocols etc
```

Damit die Namensauflösung aber auch wirklich funktioniert, werden noch einige Bibliotheken benötigt, die erst geladen werden, wenn das erste Mal versucht wird, einen Rechnernamen oder eine IP-Adresse aufzulösen:

```
# mkdir -m0755 lib
# cp -dfp /lib/ld* lib
# cp -dfp /lib/libnss_dns* lib
# cp -dfp /lib/libnss_files* lib
# cp -dfp /lib/libresolv* lib
```

Schließlich müssen wir noch dem Dienst beim Start die nötigen Parameter mitgeben, damit er auch den chroot-Käfig benutzt.

Unter SuSE machen wir dies, indem wir die Parameter in der Datei */etc/sysconfig/syslog* in die Variable SYSLOG_NG_PARAMS eintragen:

```
SYSLOG_NG_PARAMS="-C /var/lib/syslog-ng -u syslogng -g syslogng"
```

Unter Debian müssen wir dagegen das Runlevel-Skript /etc/init.d/syslog-ng anpassen. Der Dienst wird in der Unterfunktion syslogng_start() gestartet:

```
syslogng_start() {
    echo -n "Starting system logging: syslog-ng"
    start-stop-daemon --start --quiet --exec "$SYSLOGNG" --pidfile
"$PIDFILE" -- -p "$PIDFILE" \
      || { echo " start failed."; return 1; }
    echo "."
    return 0
}
```

Dabei steht alles zwischen start-stop-daemon und "$PIDFILE" \ in einer Zeile. Der Umbruch ist durch den Druck bedingt.

Der Befehl `start-stop-daemon` wird unter Debian dazu benutzt, Dienste zu starten. Dabei stehen alle Optionen für `start-stop-daemon` vor `--`, während dahinter die Optionen stehen, die unverändert an den Dienst weitergegeben werden. `--pidfile` ist damit eine Option von `start-stop-daemon`, während `-p` eine Option des `syslog-ng` ist.

Wir müssen also nun die Optionen `-C /var/lib/syslog-ng -u syslogng -g syslogng` hinter `-p` anfügen. Außerdem war ich so frei, einen zusätzlichen Zeilenumbruch einzufügen. Das Zeilenende mußte dabei mit `\` auskommentiert werden:

```
syslogng_start() {
    echo -n "Starting system logging: syslog-ng"
    start-stop-daemon --start --quiet --exec "$SYSLOGNG" \
    --pidfile "$PIDFILE" -- -p "$PIDFILE" \
    -C /var/lib/syslog-ng -u syslogng -g syslogng \
      || { echo " start failed."; return 1; }
    echo "."
    return 0
}
```

Neustart des Dienstes

Wenn wir nun den `syslog-ng` neu starten, tritt die neue Konfiguration in Kraft. Unter SuSE können wir dies mit dem folgenden Befehl tun:

```
# /etc/init.d/syslog restart
```

Unter Debian dagegen mit

```
# /etc/init.d/syslog-ng restart
```