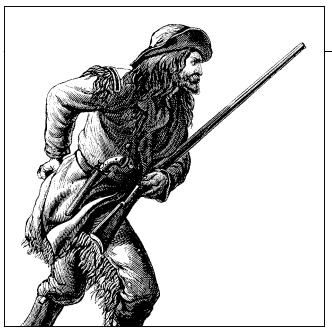

KAPITEL 14

Grundlegende Sicherheitsstrategien

Einen Linux-Computer sicher zu machen, bedeutet sowohl den Blick auf große, globale Zusammenhänge zu richten, als auch auf viele kleine Dinge. Die makroskopische Betrachtungsweise der Sicherheit eines *Linux*-Computer beruht auf seiner Netzfunktionalität und der Firewall. Die mikroskopische Sichtweise zielt auf die einzelne Datei und jedes einzelne Programm. Sowohl Zugriffe auf Dateien, als auch die Verwendung jedes einzelnen Programms können sehr fein eingestellt werden. Der Kernel kann für die gewünschte Verwendung maßgeschneidert werden. Von den zentralen Bibliotheken gibt es mehrere Versionen mit unterschiedlichem Umfang.

Naive Sichtweise auf Basis von Benutzern und Dateien

Für jedes Programm kann festgelegt werden, ob es von keinem, einem bestimmten Benutzer, einer bestimmten Gruppe oder allen Benutzern aufgeführt werden darf.

Der Zugang zum Inhalt einer jeden Datei wird keinem, einem bestimmten Benutzer, einer bestimmten Gruppe oder allen Benutzern gewährt. Es wird dabei jeweils zwischen Lesen und Schreiben unterschieden.

Verzeichnisse sind keinem, einem Benutzer, einer bestimmten Gruppe oder allen Benutzern zugänglich.

Außerdem können für Programme gewisse Vererbungsmerkmale festgesetzt werden, so dass die vorgegebene Benutzeridentität oder Benutzergruppen-Zugehörigkeit genutzt wird. (*seteUID*, *set effective user identity*, *seteGID*, *set effective group identity*). Es kann festgelegt werden, dass ein Programm nach der ersten Ausführung im Auslagerungsspeicher gespeichert wird, so dass ein neuer Aufruf schneller vonstatten geht.

Ein einschaltbares Vererbungsmerkmal kann bei Verzeichnissen bewirken, dass die Eigentümerschaft von Dateien eines Verzeichnis nicht vom Benutzer der Datei eingestellt wird, sondern vom Eigentümer der Datei. Ein anderes Vererbungsmerkmal kann sicherstellen, dass eine Datei von Benutzern geändert, aber nicht gelöscht oder umbenannt werden kann.

Alle Distributionen bringen gewisse Voreinstellungen mit, die auf jeden Fall für einen Arbeitsplatz-*Linux*-Computer ausreichend sind. Einige Distributionen bringen zusätzlich weitere globale Voreinstellungen mit, die für Server ausgewählt werden können.

Gruppierung von Nutzern und Programmen

Eigner- und Zugriffsrechte sind zentrale Punkte der Systemsicherheit. Es ist wichtig, dass Sie diese Berechtigungen richtig vergeben – auch wenn Sie der einzige Benutzer sind –, weil andernfalls merkwürdige Dinge geschehen können. Mit den Dateien, die Benutzer erzeugen und ständig bearbeiten, gibt es in der Regel keine Probleme (trotzdem kann es nicht schaden, wenn Sie die dahinter stehenden Konzepte kennen). Für den Systemverwalter ist die Lage komplizierter. Wenn Sie nur einmal falsche Eigner- oder Zugriffsrechte vergeben, kann das zur Folge haben, dass Sie beispielsweise Ihre E-Mails nicht mehr lesen können. Im Allgemeinen deutet die Meldung

```
Permission denied
```

daraufhin, dass jemand Eigner- und Zugriffsberechtigungen restriktiver als von Ihnen gewünscht vergeben hat.

Die Bedeutung der Eigner- und Zugriffsrechte

Zugriffsrechte geben an, in welcher Weise jemand eine Datei benutzen kann. Unix kennt drei Stufen der Berechtigung:

- *Read* permission (Leseberechtigung) heißt, dass Sie den Inhalt einer Datei ansehen dürfen.
- *Write* permission (Schreibberechtigung) heißt, dass Sie eine Datei ändern und löschen dürfen.
- *Execute* permission (Ausführberechtigung) heißt, dass Sie eine Datei als Programm ausführen dürfen.

Sobald eine Datei erstellt wird, vergibt das System einige Standardberechtigungen, die in den meisten Fällen genügen. So wird zum Beispiel meist die Schreibberechtigung für alle anderen Benutzer gelöscht, so dass Sie die Datei zwar schreiben und lesen können, andere jedoch sie nur lesen können. Falls Sie einen Grund sehen, besonders vorsichtig zu sein, können Sie das System so einrichten, dass andere Leute überhaupt keine Zugriffsrechte bekommen.

Einige Utilitys vergeben Berechtigungen, die von den Voreinstellungen abweichen. Wenn zum Beispiel der Compiler ein ausführbares Programm erzeugt, weist er ihm automatisch die Ausführberechtigung zu.

Es gibt allerdings auch Fälle, in denen die voreingestellten Berechtigungen nicht funktionieren. Wenn Sie zum Beispiel ein Shell-Skript oder ein Perl-Programm schreiben, müssen Sie selbst dafür sorgen, dass das Skript oder Programm ausführbar gemacht wird. Wir werden Ihnen weiter unten in diesem Abschnitt zeigen, wie das geschieht, nachdem wir die grundlegenden Konzepte behandelt haben.

Für Verzeichnisse haben die Zugriffsrechte eine andere Bedeutung:

- Leseberechtigung heißt, dass Sie den Inhalt des Verzeichnisses auflisten dürfen.
- Schreibberechtigung heißt, dass Sie in diesem Verzeichnis Dateien hinzufügen und löschen dürfen.
- Ausführberechtigung heißt, dass Sie Zugriff auf die Dateien haben, solange Sie den Dateinamen kennen.

Unterschiede zwischen Lese- und Ausführberechtigung bei Verzeichnissen können für Aufgaben wie das Nachbilden von Postkästen verwendet werden – die eigene Datei kann abgelegt und bei Bedarf geändert werden, die Daten anderer Benutzer können nicht gesehen werden.

Beachten Sie, dass Benutzer mit Schreibberechtigung für ein Verzeichnis Dateien hinzufügen und auch Dateien löschen dürfen – mit der Vergabe der Schreibberechtigung eröffnen Sie dem Benutzer beide Möglichkeiten. Trotzdem gibt es eine Methode, verschiedenen Benutzern den Zugriff auf ein gemeinsames Verzeichnis zu gestatten, ohne dass jeder die Dateien des anderen löschen kann.

Auf einem *Unix*-System gibt es weit mehr Dateitypen als die einfachen Dateien und Verzeichnisse, die wir bisher besprochen haben. Das sind zum Beispiel Gerätedateien (device files), Sockets, symbolische Links usw. Jeder Dateityp hat seine eigenen Regeln in Bezug auf die Zugriffsrechte, aber es ist nicht notwendig, dass Sie alle Details zu jedem Dateityp kennen.

Eigner und Gruppen

Wem werden eigentlich diese Berechtigungen zugestanden? Damit verschiedene Benutzer auf einem System arbeiten können, unterscheidet *Unix* bei den Berechtigungen drei Benutzergruppen: *Eigner*, *Gruppe* und *Sonstige*. Die Benutzergruppe *Sonstige* umfasst alle Benutzer, die Zugang zum System haben, aber nicht Eigner oder Mitglied der Gruppe sind.

Die Idee hinter der Einrichtung einer Gruppe ist, dass man zum Beispiel einem Team von Programmierern den Zugang zu einer Datei gewährt. So könnte eine Programmiererin sich selbst die Schreibberechtigung für ihren Quellcode erteilen, während die Mitglieder ihres Teams mittels Gruppenberechtigung den lesenden Zugriff bekommen. Die Benutzergruppe *Sonstige* bekommt vielleicht gar keine Zugriffsrechte, damit Leute außerhalb Ihres Teams nicht in Ihrem Quellcode herumschnüffeln können. (Glauben Sie wirklich, dass Ihre Programme *so* gut sind?)

Jede Datei gehört genau einem Eigner und einer Gruppe. Der Eigner ist in der Regel der Benutzer, der die Datei angelegt hat. Jeder Benutzer gehört standardmäßig genau einer Gruppe an, und diese Gruppe wird jeder Datei zugeordnet, die dieser Benutzer erstellt. Sie können beliebig viele Gruppen einrichten, und jeder Benutzer kann beliebig vielen zusätzlichen Gruppen angehören. Indem Sie die Gruppe ändern, die einer Datei zugeordnet ist, können Sie einer beliebig zusammengesetzten Gruppe von Benutzern Zugriff auf die Datei gewähren. Die Wirksamkeit der Zugehörigkeit eines Benutzers zu einer Gruppe wird mit dem Befehl `chgrp` geändert. Es ist immer nur eine Gruppenzugehörigkeit aktiv.

Damit haben wir alle Elemente für unsere Sicherheitsvorkehrungen zusammen: drei Arten der Berechtigung (*lesen*, *schreiben*, *ausführen*) und drei Grade bei den Benutzern (*Eigner*, *Gruppe*, *Sonstige*). Lassen Sie uns einen Blick auf die Berechtigungen für einige typische Dateien werfen.

Folgende Abbildung zeigt ein typisches ausführbares Programm. Wir haben `ls` mit der Option `-l` eingegeben, um diese Anzeige zu erzeugen.

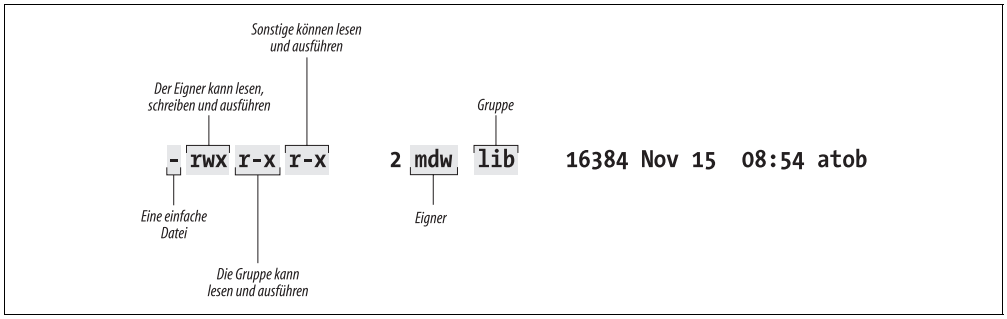

Abbildung 14-1: Eignerschaft und Berechtigungen anzeigen

Zwei nützliche Informationen fallen sofort auf: Der Eigner der Datei ist der Autor dieses Buches und Ihr zuverlässiger Begleiter, *mdw*, und die Gruppe ist *lib* (vielleicht ein Team von Programmierern, die Libraries erstellen). Die wirklich wichtigen Informationen über die Berechtigungen sind allerdings in der Buchstabenfolge links in der Anzeige versteckt.

Das erste Zeichen ist ein Bindestrich und zeigt eine normale Datei an. Die nächsten drei Stellen beziehen sich auf den Eigner – wie man erwarten konnte, hat *mdw* alle drei Berechtigungen. Die folgenden drei Stellen beziehen sich auf die Mitglieder der Gruppe – sie können die Datei lesen (`r`) und ausführen (`x`), aber sie haben keinen schreibenden Zugriff, weil das Feld, das ein `w` enthalten sollte, stattdessen einen Bindestrich aufweist. Die letzten drei Stellen zeigen die Berechtigungen für *Sonstige* – in diesem Fall haben *Sonstige* dieselben Berechtigungen wie die Mitglieder der Gruppe.

Noch ein Beispiel: Angenommen, Sie fordern ein langes Listing einer C-Quelldatei an, dann könnte das so aussehen:

```
$ ls -l
-rw-rw-r--   2 kalle      kalle          12577 Apr  30 13:13 simc.c
```

Der einzige Unterschied hier besteht darin, dass der Eigner Lese- und Schreibrechte (`rw`) hat, genau wie die Gruppe. Alle anderen im System haben nur Leserechte.

Nehmen wir jetzt an, dass wir die Datei in ein ausführbares Programm kompilieren. Der Compiler `gcc` erzeugt die Datei `simc`:

```
$ gcc -osimc simc.c
$ ls -l
total 36
-rwxrwxr-x   1   kalle    kalle    19365    Apr 30 13:14 simc
-rw-rw-r--   1   kalle    kalle    12577    Apr 30 13:13 simc.c
```

Neben den Lese- und Schreib-Bits hat gcc für die ausführbare Datei das Ausführbarkeits-Bit (x) für den Benutzer, die Gruppe und alle anderen gesetzt. Das ist auch sinnvoll so, damit das Programm ausgeführt werden kann:

```
$ ./simc
(Programmausgabe)
```

Als weiteres Beispiel zeigen wir noch ein typisches Verzeichnis:

```
drwxr-xr-x   2 mdw      lib           512 Jul 17 18:23 perl
```

An der Stelle ganz links zeigt jetzt ein d an, dass es sich um ein Verzeichnis handelt. Die Ausführberechtigungen sind wieder da, so dass alle Benutzer sich den Inhalt des Verzeichnisses anzeigen lassen können.

Dateien können noch andere Zustände annehmen, die wir hier nicht besprechen wollen. Lesen Sie die Details in der Manpage zu ls nach. Für uns wird es jetzt Zeit, die Eigner- und Zugriffsrechte zu ändern.

Eigner, Gruppe und Berechtigungen ändern

Wir haben bereits darauf hingewiesen, dass Sie in der Regel mit den Sicherheitsvorkehrungen auskommen, die das System bietet. Allerdings gibt es auch Ausnahmen, insbesondere für Systemverwalter. Ein einfaches Beispiel: Nehmen wir an, dass Sie für einen neuen Benutzer unterhalb von */home* ein Verzeichnis anlegen. Sie müssen diese Aufgabe als *root* erledigen, aber später den Benutzer zum Eigner des Verzeichnisses machen. Wenn Sie das nicht tun, wird der Benutzer nicht in diesem Verzeichnis arbeiten können! (Glücklicherweise sorgt der Befehl adduser automatisch für die richtige Eignerschaft.)

In ähnlicher Weise haben bestimmte Programmsammlungen wie *UUCP* und *News* ihre eigenen Benutzer. Niemand wird jemals als *UUCP* oder *News* einloggen, aber diese Benutzer und Gruppen müssen auf dem System vorhanden sein, damit die Programme dieser Sammlungen sicher funktionieren. Im Allgemeinen besteht der letzte Schritt bei der Installation darin, dass man Eigner, Gruppe und Berechtigungen so anpasst, wie es die Dokumentation vorgibt.

Mit dem Befehl chown ändern Sie die Eignerschaft einer Datei und mit chgrp die Gruppe. Unter *Linux* kann chown nur von *root* verwendet werden, um die Eigentümerschaft einer Datei zu ändern, aber jeder Benutzer kann die Gruppenzugehörigkeit einer Datei von seiner augenblicklichen Gruppe in jede andere Gruppe ändern, zu der er gehört.

Nachdem Sie also zum Beispiel die Software sampsoft installiert haben, können Sie mit folgenden Befehlen sowohl den Eigner als auch die Gruppe auf bin ändern:

```
# chown bin sampsoft
# chgrp bin sampsoft
```

Das geht auch in einem Schritt, wenn die Punkt-Notation verwendet wird:

```
# chgrp bin.bin sampsoft
```

Die Syntax für die Änderung der Berechtigungen ist komplizierter. Man nennt die Berechtigungen auch den *Modus* einer Datei, und diesen ändern Sie mit dem Befehl

chmod. Lassen Sie uns diesen Befehl anhand eines einfachen Beispiels erkunden. Nehmen wir an, dass Sie in *Perl* oder *Tcl* ein nettes kleines Programm namens header geschrieben haben, das Sie anschließend ausführen wollen.

```
$ chmod +x header
```

Das Pluszeichen + bedeutet: *Füge eine Berechtigung hinzu*, und das x zeigt an, welche Berechtigung gemeint ist.

Wenn Sie jemandem die Ausführberechtigung entziehen möchten, setzen Sie statt des Plus- ein Minuszeichen - ein:

```
$ chmod -x header
```

Der eben gezeigte Befehl vergibt die Berechtigung auf allen Ebenen – an den Eigner, die Gruppe und Sonstige. Lassen Sie uns annehmen, dass Sie insgeheim ein Sammler von Software sind, der seine Programme für sich behalten möchte. (Nein, das wäre zu hart. Wir wollen stattdessen annehmen, dass Ihr Skript noch nicht einwandfrei funktioniert und dass Sie andere Leute vor Schaden bewahren möchten, bis das Skript fehlerfrei ist.) Mit folgendem Befehl können Sie die Ausführberechtigung nur für sich selbst vergeben:

```
$ chmod u+x header
```

Alle Angaben vor dem Pluszeichen bezeichnen die Benutzerebene, auf der Sie Berechtigungen vergeben. Die Angaben hinter dem Pluszeichen geben die Art der Berechtigung an. Mit g vergeben Sie Rechte an die Gruppe und mit o an Sonstige (*Others*). Wenn Sie die Ausführberechtigung an sich selbst und die Gruppe erteilen wollen, geben Sie Folgendes ein:

```
$ chmod ug+x header
```

Sie können auch mehrere Berechtigungen gleichzeitig erteilen:

```
$ chmod ug+rwx header
```

Es gibt noch die eine oder andere Abkürzung, die Sie in der Manpage zu chmod nachlesen können, falls Sie jemanden beeindrucken möchten, der Ihnen über die Schulter sieht. Allerdings bleibt die Funktionalität des Befehls auf das beschränkt, was wir hier vorgestellt haben.

Obwohl die Syntax zur Angabe des Dateimodus schon ziemlich obskur ist, gibt es noch eine andere, kompliziertere Syntax. Aus verschiedenen Gründen müssen wir sie hier trotzdem beschreiben. Erstens gibt es Situationen, in denen die gerade vorgestellte Syntax, die *symbolischer Modus* genannt wird, nicht ausreicht. Zweitens benutzen die Leute oft die andere Syntax, die *absoluter Modus* genannt wird, in ihrer Dokumentation. Außerdem könnte es ja passieren, dass Sie die absolute Schreibweise einfach bequemer finden.

Um den absoluten Modus zu verstehen, müssen Sie sich auf die Bitebene und die oktale Schreibweise einlassen. Aber keine Bange – so schwierig wird das nicht. Der typische Modus wird durch drei Zeichen dargestellt, die den drei Benutzerebenen entsprechen (*Eigner*, *Gruppe* und *Sonstige*). Diese Ebenen sind in folgender Abbildung dargestellt. Innerhalb jeder Ebene bezeichnen drei Bits die Berechtigung zum Lesen, Schreiben und Ausführen.

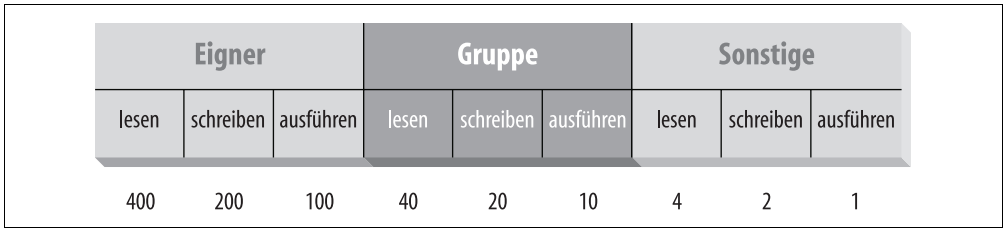

Abbildung 14-2: Die Bits im absoluten Modus

Nehmen wir an, dass Sie sich selbst die Leseberechtigung und niemandem sonst irgendwelche Rechte erteilen wollen. Sie möchten also nur das Bit ansprechen, das mit der Nummer 400 bezeichnet ist. Der `chmod`-Befehl würde dann so aussehen:

```
$ chmod 400 header
```

Um jedermann die Leseberechtigung zu geben, wählen Sie das entsprechende Bit für alle Benutzerebenen: `400` für den Eigner, `40` für die Gruppe und `4` für Sonstige. Der Befehl lautet dann:

```
$ chmod 444 header
```

Das entspricht dem Modus `+r` – abgesehen davon, dass Sie mit diesem Befehl gleichzeitig alle Berechtigungen zum Schreiben und Ausführen aufheben. (Um es genau zu sagen: Es entspricht dem Modus `=r`, den wir weiter oben nicht erwähnt haben. Das Gleichheitszeichen bedeutet: *Vergib diese Berechtigung, und hebe alle anderen Rechte auf.*)

Wenn Sie allen Benutzern die Lese- und Ausführberechtigung zuweisen wollen, müssen Sie die Lese- und Ausführbits addieren. Ein Beispiel: `400` plus `100` ist `500`. Der komplette Befehl lautet also:

```
$ chmod 555 header
```

und das entspricht dem Modus `=rx`. Wenn jemand vollen Zugriff erhalten soll, steht an der entsprechenden Stelle eine `7` – nämlich `4 + 2 + 1`.

Einen Trick wollen wir Ihnen noch verraten, nämlich wie Sie den Modus voreinstellen können, der jeder Datei zugeordnet wird, die Sie erzeugen (mit einem Texteditor, der Umleitung `>` usw.). Führen Sie dazu entweder den Befehl `umask` aus, oder fügen Sie ihn in die Startdatei Ihrer Shell ein. Diese Datei heißt wahrscheinlich *.bashrc*, *.cshrc* oder so ähnlich – je nachdem, mit welcher Shell Sie arbeiten. (Wir werden die Startdateien im nächsten Abschnitt besprechen.)

Der Befehl `umask` bekommt einen Parameter mit auf den Weg, so wie `chmod` den absoluten Modus mitbekommt. Allerdings ist die Bedeutung der Bits gerade umgekehrt. Sie müssen für den Eigner, die Gruppe und Sonstige festlegen, welche Berechtigungen Sie vergeben wollen, und dann jede einzelne Ziffer von `7` subtrahieren. Das Ergebnis ist eine dreistellige Maske.

Nehmen wir an, dass Sie sich selbst alle Rechte zugestehen wollen (`7`), die Gruppe soll Lese- und Ausführberechtigung haben (`5`), und Sonstige erhalten gar keinen Zugriff (`0`).

Ziehen Sie diese Werte von 7 ab, und Sie erhalten 0 für sich selbst, 2 für die Gruppe und 7 für Sonstige. Der Befehl in Ihrer Startdatei muss also lauten:

```
umask 027
```

Eine merkwürdige Vorgehensweise, aber sie funktioniert. Der Befehl `chmod` berücksichtigt die Maske, wenn er Ihren Modus interpretiert. Ein Beispiel: Wenn Sie für eine Datei bei der Erzeugung die Ausführberechtigung vergeben, wird `chmod` Ihnen und der Gruppe die Ausführberechtigung zuteilen, aber alle anderen werden davon ausgeschlossen, weil die Maske für diese Benutzer keine Ausführberechtigung angibt.

Pfade und Privilegien

Unter *Unix*-Derivaten – und damit auch *Linux* – ist ein Befehl einfach eine gewöhnliche Datei. Der Befehl `ls` zum Beispiel ist eine binäre Datei im Verzeichnis *bin*. Statt `ls` einzugeben, könnten Sie also auch den kompletten Pfadnamen eingeben (auch *absoluter Pfadname* genannt):

```
$ /bin/ls
```

Das verleiht *Unix* Flexibilität und macht es zu einem mächtigen Betriebssystem. Ein Systemverwalter, der ein neues Werkzeug bereitstellen möchte, kann dieses einfach in einem der Verzeichnisse installieren, in denen Befehle stehen. Es lassen sich auch verschiedene Versionen eines Befehls installieren – etwa eine neue Version zu Testzwecken in einem bestimmten Verzeichnis, während die alte Version an einer anderen Stelle verbleibt. Die Benutzer können dann selbst entscheiden, mit welcher Version sie arbeiten möchten.

Hier ergibt sich häufig ein Problem: Manchmal geben Sie einen bekannten Befehl ein, aber das System antwortet mit einer Meldung wie `Not found`. Das Problem könnte darin bestehen, dass der Befehl in einem Verzeichnis steht, das von der Shell nicht durchsucht wird. Man nennt die Liste aller Verzeichnisse, in denen die Shell nach Befehlen sucht, den `Pfad` (`PATH`). Mit folgendem Befehl können Sie Ihren Pfad anzeigen lassen (denken Sie an das Dollar-Zeichen, ansonsten bekommen Sie nicht den Inhalt der Umgebungsvariablen zu sehen, sondern ihren Namen, den Sie ohnehin schon kennen!):

```
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/lib/java/bin:\
/usr/games:/usr/bin/TeX:.
```

Hier müssen Sie genau hinsehen. Die Anzeige stellt eine Reihe von Pfadnamen dar, die durch Doppelpunkte getrennt sind. Der erste Pfadname in diesem Fall ist */usr/local/bin*, der zweite */usr/bin* und so weiter. Wenn zwei Versionen eines Befehls vorhanden sind, die in */usr/local/bin* und */usr/bin* stehen, wird der Befehl in */usr/local/bin* ausgeführt. Der letzte Pfadname in diesem Beispiel ist einfach ein Punkt; dieser bezeichnet das aktuelle Verzeichnis. Anders als die Eingabeaufforderung von *Windows* durchsucht *Unix* nicht automatisch das aktuelle Verzeichnis, sondern Sie müssen es ausdrücklich dazu anweisen – so wie wir es hier gezeigt haben. Manche Leute halten das aus Gründen der Systemsicherheit für eine schlechte Idee. (Ein Eindringling, der in Ihren Account vordringt,

könnte ein bösartiges Programm in eines Ihrer Arbeitsverzeichnisse kopieren.) Das betrifft aber hauptsächlich den Systemverwalter; normale Benutzer müssen sich wenig Sorgen darüber machen.

Wenn ein Befehl nicht gefunden wird, müssen Sie herausfinden, wo im System er sich befindet, damit Sie das betreffende Verzeichnis in Ihren Pfad einfügen können. Die Manpage sollte Ihnen sagen, wo der Befehl steht. Nehmen wir an, dass er sich im Verzeichnis */usr/sbin* befindet, wo auch andere Befehle für die Systemverwaltung installiert sind. Ihnen ist klar, dass Sie Zugang zu den Befehlen für die Systemverwaltung brauchen, deshalb geben Sie Folgendes ein (beachten Sie, dass das erste `PATH` ohne, das zweite dagegen mit einem Dollar-Zeichen geschrieben wird):

```
$ export PATH=$PATH:/usr/sbin
```

Dieser Befehl fügt */usr/sbin* zum Suchpfad hinzu, und zwar als das Verzeichnis, das als letztes durchsucht wird. Der Befehl heißt soviel wie: *Definiere meinen Pfad als den alten Pfad plus /usr/sbin*.

Der eben vorgestellte Befehl funktioniert übrigens nicht in allen Shells. Die meisten *Linux*-Benutzer, die mit einer Bourne-kompatiblen Shell wie `bash` arbeiten, sollten keine Probleme damit haben. Wenn Sie allerdings `csh` oder `tcsh` benutzen, müssen Sie stattdessen diesen Befehl eingeben:

```
set path = ( $PATH /usr/sbin )
```

Abschließend wollen wir noch auf ein paar Befehle hinweisen, die nicht als eigenständige Programmdateien existieren. `cd` ist einer davon. Die meisten dieser Befehle wirken sich auf die Shell selbst aus und müssen deshalb von der Shell verstanden und ausgeführt werden. Weil sie ein Teil der Shell sind, nennt man sie *interne Befehle* (*built-in commands*).

Linux entscheidet über die Ausführbarkeit von Dateien nicht durch den Namen oder eine Dateiendung, sondern führt für jede Datei im Dateisystem verankerte Merkmale (Flags) mit. Dabei wird zwischen Ausführungsberechtigungen für den Eigentümer der Datei, für eine Gruppe, der die Datei gehört und alle Benutzer unterschieden. Ausführbare Dateien, das sind Programme, können am Dateianfang spezielle Bytes enthalten, die von der magic-Funktion ausgewertet werden können. Bei Programmen, die im generischen ELF-Binärformat vorliegen, wird von der Shell der Linker aufgerufen, der benötigte Bibliotheken lädt. Im Fall einer Skript-Datei wird der Interpreter der Skript-Sprache – wie *Perl* – aufgerufen. Die `magic`-Funktion kann beispielsweise für `/bin/ls` mit dem Befehl

```
file /bin/ls
```

ausgewertet werden, für das Kommando `ls` ist die Ausgabe

```
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5,
dynamically linked (uses shared libs), stripped
```

Die Ausführbarkeit kann mit `chmod` für Test- und Wartungszwecke gezielt an- und abgeschaltet werden.

Leistungsbeschränkte Shell-Programme

Die bash-Shell ist ein sehr mächtiges Werkzeug. Der große Leistungsumfang der Shell lässt natürlich auch Missbrauch zu. Unbekannten, nur beschränkt vertrauenswürdigen oder sehr ungeübten Benutzern kann man eine Shell mit beschränktem Leistungsumfang zur Verfügung stellen, die für viele Aufgaben ausreicht. Dies kann durch den Eintrag

```
lieschen:x:1000:100:Lieschen Müller:/home/cep:/usr/bin/rbash
```

in der Datei */etc/passwd* erreicht werden. Die Benutzer der rbash können die meisten kritischen Aktionen nicht ausführen. Eine im Leistungsumfang reduzierte Shell kann auch Programmen, die eine Shell benötigen – wie Web- oder Mail-Server – zugewiesen werden und als Standardshell vorgeschrieben werden. Manche Serverdienste wie Mailserver bringen gleich eine modifizierte Shell mit und sind auf ihre Nutzung voreingestellt.

Auch gibt es von Editoren wie dem vi beschränkte Versionen wie den rvim, die beispielsweise das Ausführen von Shell-Kommandos aus dem Editor heraus verbieten. Durch die Kombination von beschränkter Shell und beschränkten Anwendungsprogrammen kann ein Linux-Computer sehr sicher gemacht werden.

Nur zeitweilig genutzte Dateien

Viele Programme speichern Daten in besonderen Dateien ab, die sie zur Laufzeit anlegen und nach dem Ende der Laufzeit nicht mehr benötigen. So kann bei der Verarbeitung sehr großer Datenmengen kostbarer Hauptspeicher gespart werden. Einige Programme löschen diese Dateien nicht nach dem regulären Programm-Ende. Bei fast allen Programmen, die temporäre Dateien verwenden, bleiben bei einem irregulärem Programm-Ende, einem *Absturz,* die temporären Dateien erhalten. Solche Dateien werden in den Verzeichnissen */tmp* und */var* abgelegt.

Temporäre Dateien können Informationen enthalten, die Missbrauch erlauben. Sie müssen geschützt und nach Benutzung regelmäßig entfernt werden, am besten durch einen cron-Job. Auch während des Zugriffs durch das sie erzeugende Programm können temporäre Dateien Ziel eines Angriffs sein, daher wird über spezielle Funktion mktemp der Zugriff auf das sie erzeugende Programm beschränkt.

Der Inhalt der Verzeichnisse für nur zeitweilig genutzte (temporäre) Dateien */tmp* und */var/tmp* muss bei öffentlich zugänglichen *Linux*-Computern unter sorgfältiger Beobachtung gehalten werden. Bei ausschließlich privat genutzten *Linux*-Computern muss nur der noch verfügbare Platz in regelmäßigen Abständen überwacht werden.

Linux-Computer, die Druckaufgaben und Maildienste wahrnehmen, verwenden das Verzeichnis */var/spool*. Je nach Druck- und Mail-Server bleiben zumindest Protokolldateien zurück. Bei heftigem Druck- und Mailaufkommen besteht die Gefahr, dass der Platz auf dieser Partition oder Platte, die diese Verzeichnisse beherbergt sehr schnell erschöpft ist – dies geschieht natürlich besonders häufig am Wochenende.

Change-root-Umgebung

Eine der ganz großen Stärken des *Linux*-Konzepts ist seine äußerst weit gehende Konfigurierbarkeit. Von einem naiven Standpunkt aus scheint für jede Konfiguration ein eigener *Linux*-Computer notwendig zu sein. Mit dem *Change-Root*-Konzept können auf einem *Linux*-Computer nahezu beliebig viele, einander nicht beeinträchtigende Konfigurationen bereitgestellt werden. Das *Change-Root*-Konzept wird routinemäßig von einigen, besonders bösartigen Angriffen ausgesetzten Serverdiensten wie FTP eingesetzt.

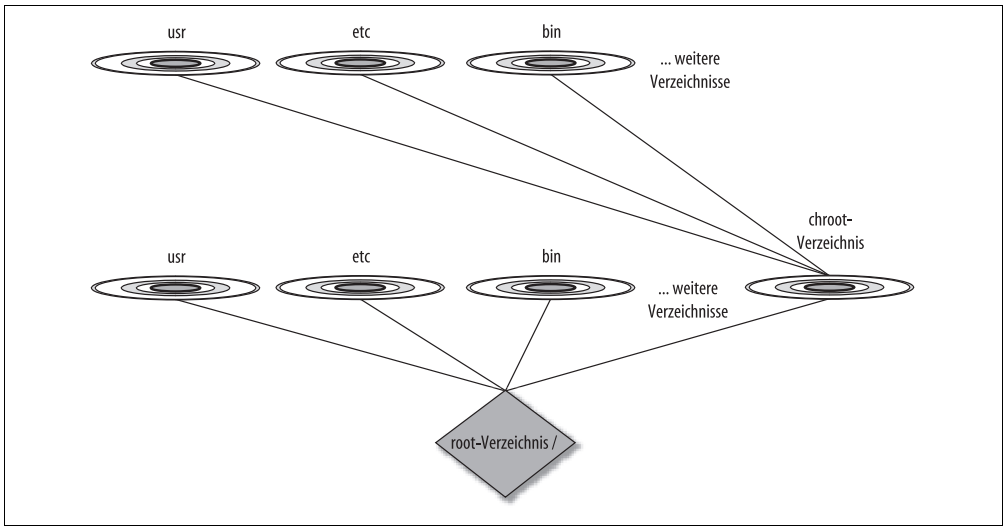

Abbildung 14-3: Struktur einer Change-Root-Umgebung

Vorbereitung einer Change-Root-Umgebung

Eine *Change-Root*-Umgebung ist im Wesentlichen eine *Linux*-Installation ohne die Boot-Mechanismen. So umfasst eine *Change-Root*-Umgebung alle notwendigen Bibliotheken, Programme und Daten einer gewöhnlichen *Linux*-Installation. Von *Linux*-Distributoren wird mit ihren Paketmanagementprogrammen die Option *Installation in ein Verzeichnis* angeboten. So kann ein Standardsystem einfach bereitgestellt werden. Bei der Installation von Hand, die für angepasste Installationen verwendet wird, wird der Paketmanager `rpm` durch

```
rpm -r /chroot -i aaa_base.rpm
```

dazu angewiesen, das Paket *aaa_base.rpm* nicht in das Verzeichnis / zu installieren, wo sein Inhalt normalerweise liegt, sondern in das Verzeichnis */chroot*. Für `dpkg` wird Entsprechendes durch

```
dpkg -root=/chroot -i aaa_base.deb
```

bewirkt. Wenn Pakete mit statisch gebundenen Bibliotheksfunktionen (Static Linked Binary) verfügbar sind, kann bei Verwendung solcher Pakete auf die Installation dyna-

misch gebundener Bibliotheken (Dynamic Linked Libraries) verzichtet werden. Damit kann bei einer Installation mit geringem Umfang an Software die Menge der zu installierenden Pakete stark verringert werden.

Nutzung von Change-Root-Umgebungen

Der Eintritt in eine *Change-Root*-Umgebung erfolgt durch

```
chroot /chroot
```

mit dem Verzeichnis */chroot*, in das ein lauffähiges System installiert wurde. Es wird standardmäßig eine `bash`-Shell gestartet. Soll jedoch ein anderes Programm gestartet werden, so bewirkt der Aufruf

```
chroot /chroot /usr/bin/rvim
```

den Aufruf des leistungsbeschränkten Editor in der geschützten Umgebung. Wenn die Shell oder das Programm beendet werden, hört die *Change-Root*-Umgebung auf zu bestehen. Die *Change-Root*-Umgebung kann benutzt werden, um eine Testumgebung zu schaffen, die das installierte *Linux*-System nicht beschädigen kann – es kann sein Muttersystem nur durch übermäßigen Rechenverbrauch beeinträchtigen. Wenn in der *Change-Root*-Umgebung kein */dev*-Verzeichnis existiert, kann mit Standardprogrammen aus der *Change-Root*-Umgebung nicht ohne weiteres auf die Hardware zugegriffen werden.

User-Mode-Linux

Der Ansatz, eine geschützte Umgebung zu verwenden, kann mit *User-Mode-Linux* noch weiter ausgedehnt werden. *User-Mode-Linux* erlaubt es, nicht nur eine eigene Umgebung wie bei der *Change-Root*-Umgebung zu schaffen, sondern auch einen eigenen Kernel zu benutzen, mit eigener Boot-Umgebung und völlig eigenem Dateisystem.

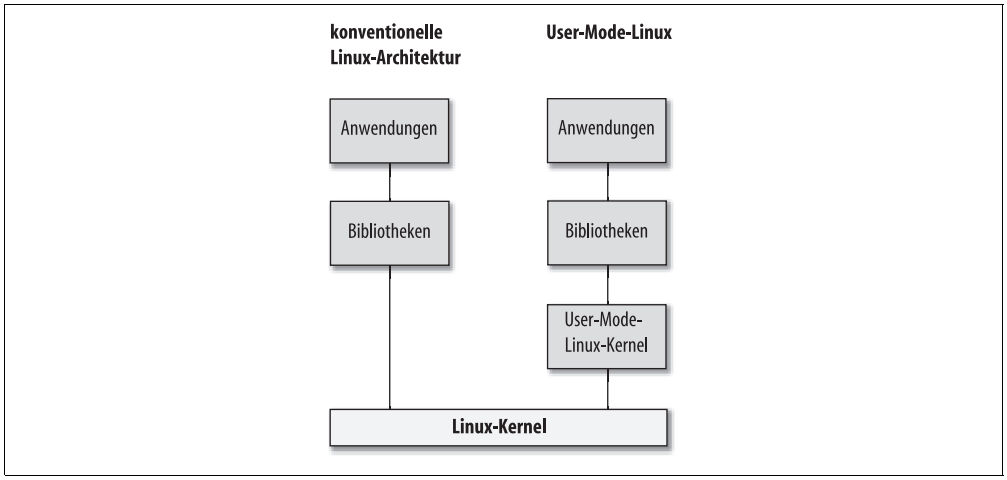

Abbildung 14-4: Architektur des User-Mode-Linux

User-Mode-Linux ist eine gute Testumgebung, eignet sich zur Isolation sicherheitsempfindlicher Anwendungen (Sandbox) und bietet ohne *root*-Rechte auf dem Trägersystem dem Benutzer alle Rechte auf dem von ihm als Prozess gestarteten *User-Mode-Linux*.

Vorbereitung von User-Mode-Linux

Das Dateisystem von *User-Mode-Linux* wird in einer Datei abgespeichert. Um diese Datei zu erzeugen, wird im Arbeitsverzeichnis –vorgeschlagen wird hier */home/uml* – mit

```
dd if=/dev/zero of=root.img bs= 1048576 count=1024
```

eine Datei mit 1 GB Größe bereitgestellt. Ihr Inhalt sind 0-Bytes. In gleicher Weise wird mit

```
dd if=/dev/zero of=swap.img bs= 1048576 count=256
```

eine Swap-Datei mit 256 KB vorbereitet. Beide müssen noch entsprechend vorbereitet werden, das Dateisystem mit

```
mkfs.ext2 -q -b 4096 -F ubd0
```

der Swapmode mit

```
mkswap swap
```

Dann kann das in der Datei *udb0* befindliche Dateisystem mit

```
mount ubd0 /mnt -o loop -t ext2
```

in das vorhandene Dateisystem eingehängt werden. Dann wird die notwendige Software – analog wie bei Change-Root-Umgebung – in das Verzeichnis */mnt* installiert. Nach Abschluss wird die Datei mit

```
sync
umount /mnt
```

ausgehängt und steht dann zur Verfügung. Der Bau eines passenden *Linux*-Kernels und gegebenenfalls einer Init-Ram-Disk wird entweder mit der Anleitung[1] der Programmierer von *User-Mode-Linux* vorgenommen oder es wird eine vorbereitete Kernel-Ramdisk-Kombination verwendet, was zu empfehlen ist. Der Kernel und die Ramdisk kommen in dasselbe Arbeitsverzeichnis wie die zuvor bereitgestellten Dateien. Der modifizierte Kernel kann mit dem Aufruf

```
make menuconfig ARCH=um
```

bereitgestellt werden. Danach ist ein Image mit

```
make bzimage
```

zu erzeugen und im Arbeitsverzeichnis zu speichern.

1 *http://user-mode-linux.sourceforge.net/*

Verwendung von User-Mode-Linux

Im Arbeitsverzeichnis kann dann der *linux* genannte Kernel mit

```
linux ubda=root.img
```

gestartet werden. Eine solche Standardkonfiguration kann dann nur virtuelle Konsolen bedienen. Es kann aber *User-Mode-Linux* zum einen Hardwarezugriff gewährt werden, es kann aber auch über Pseudo-Netzwerkgeräte ein Netzwerkzugriff möglich gemacht werden. Ein besonders hervorzuhebender Einsatz für *User-Mode-Linux* ist der Einsatz in einer komplexen Netzwerkumgebung als Plattform für Firewall, Webserver und andere kritische Serverdienste. Wenn diese nachhaltig gestört werden, bleibt das Hauptsystem unbeeinflusst und kann die *User-Mode-Linux*-Umgebung der Störung angepasst neu starten.

Erhöhung der Sicherheit über den Linux-Kernel

Linux ist von seiner Herkunft aus der *Unix*-Welt ein auf Betriebssicherheit ausgerichtetes System. Die *Linux*-Distributoren stehen aber vor einem Dilemma – einen *Linux*-Kernel auszuliefern, der alle nur denkbaren Anforderungen erfüllt oder eine Vielzahl von auf einen bestimmten Verwendungszeck optimierten Kerneln bereitzustellen. Bei rund 5000 Konfigurationsoptionen – die Gesamtzahl der resultierenden Möglichkeiten übersteigt eine Zahl mit mehr als 1600 Stellen – im *Linux*-Kernel ist kein Distributor in der Lage, die daraus erwachsende gigantische Menge an spezifisch angepassten *Linux*-Kernels bereitzustellen.

Bedarfsanalyse

Linux unterstützt eine sehr große Menge an Protokollen und Geräten. Entbehrliche Gerätetreiber sollten fortgelassen werden. Ein möglicher Programmierfehler in einem nicht verwendeten Gerätetreiber kann sich so nicht auswirken. Ein Standard-*Linux*-Kernel unterstützt gleichzeitig Protokolle, die sich in ihrer Funktion widersprechenden, wie Routing und Bridging. Grundsätzlich ist es möglich, für beide Protokolle gleichzeitig Bedarf zu haben, dies ist aber äußerst selten. So wird man dann auch auf Firewallfunktionalität verzichten, wenn der Linux-Computer ohne Netzwerkanbindung eingesetzt wird oder geschützt hinter einer Firewall liegt. Spezielle Weitverkehrsprotokolle wird nur der einsetzen, der sie für die besondere Hardware auch benötigt

Aufstellung der benötigten Kerneleigenschaften

Die notwendigen Kerneleigenschaften können vom geplanten Verwendungszweck abgeleitet werden.

Ein Datei- und Druckserver benötigt Unterstützung für die verwendeten Netzwerkgeräte, die zu verwendenden Protokolle, die Druckerschnittstellen und die Massenspeicher.

Eine Firewall benötigt Unterstützung für die Netzwerkgeräte und die Paket-Filter. Eine Firewall kommt möglicherweise völlig ohne Massenspeicher aus, wenn sie aus dem Netz gebootet wird und Protokolle auf einen Log-Server im Netz ausgibt.

Ein als isolierte Arbeitsstation verwendeter Linux-Computer mit Anschluss für das Internet benötigt die Unterstützung der Hardware für den Internetzugang, Firewallfunktionalität und – wahrscheinlich – Video und Tonwiedergabe. Verzichten kann dieser Arbeitsplatz-Computer auf SCSI-, Raid-, Routing-, Watchdog- und Kernel-Debugging.

Erzeugen eines minimalen Kernels

Sieben Schritte führen Sie zu einem neuen Kernel, und diese sollten ohne größere Schwierigkeiten durchführbar sein (all diese Schritte werden auf den nächsten Seiten detaillierter beschrieben):

1. Stellen Sie sicher, dass alle notwendigen Werkzeuge und Hilfsprogramme in den korrekten Versionen installiert sind. Die Datei *Documentation/Changes* in den Kernelquellen nennt alle Anforderungen.
2. Rufen Sie `make config` auf, und beantworten Sie eine Reihe von Fragen nach den benötigten Treibern. Sie können auch eine der komfortableren Varianten `make menuconfig` oder (nur wenn Sie das *X-Window*-System installiert haben) `make xconfig` verwenden.
3. Wenn Sie zuvor schon einen Kernel gebaut und dann Patches eingespielt haben, dann können Sie `make oldconfig` verwenden, um die alte Konfiguration weiterzuverwenden, aber nach neuen Optionen, die im alten Kernel noch nicht vorhanden waren, gefragt zu werden. In neueren Kerneln wird mit `make oldconfig` eine Standardkonfiguration erzeugt, wenn dies direkt nach der Installation Kernelquellen der Kernelquelle erfolgt oder nach dem Aufruf von `make mrproper`.
4. Rufen Sie `make dep` auf, um Abhängigkeiten der Quelldateien festzustellen und in die verschiedenen Makefiles einzutragen. Dieser Aufruf ist für 2.6-Kernel nicht mehr erforderlich.
5. Wenn Sie vorher schon von diesem Verzeichnisbaum aus einen Kernel erstellt haben, rufen Sie `make clean` auf, damit alte Objektdateien entfernt werden und eine komplette Neukompilierung stattfindet. Ab Version 2.6 kann die Codeverwaltung des Kernels entscheiden, ob eine Neuübersetzung notwendig ist, der Aufruf kann daher entfallen.
6. Starten Sie die Kompilierung des Kernels mit `make bzImage`.
7. Gehen Sie einen Kaffee trinken (oder zwei; das hängt von der Geschwindigkeit Ihres Rechners und dem verfügbaren Arbeitsspeicher ab).
8. Installieren Sie die neue Kopie des Kernels entweder auf einer Diskette oder mittels *LILO* oder *GRUB*. Mit `make bzDisk` können Sie den Kernel auf eine Boot-Diskette schreiben.

Den Kernel finden Sie als */usr/src/linux/arch/i386/boot/bzImage* für Intel-Architekturen. Alle diese Befehle werden von `/usr/src/linux` aus aufgerufen – außer Schritt 7, den Sie an beliebiger Stelle durchführen können.

Zu den Kernelquelltexten gehört eine *README*-Datei, die auf Ihrem System unter */usr/src/linux/README* stehen sollte. Lesen Sie diese Datei. Sie finden dort aktuelle Hinweise zur Kompilierung des Kernels, die aktueller sein könnten als die Hinweise in diesem Buch. Befolgen Sie die Anweisungen in *README*, und nehmen Sie die Erläuterungen in diesem Buch zu Hilfe.

Der erste Schritt ist der Aufruf von `make config`. Damit wird ein Skript gestartet, das Ihnen eine Reihe von Ja/Nein-Fragen nach den benötigten Treibern stellt. Zu jeder Frage gibt es eine voreingestellte Antwort, aber seien Sie vorsichtig – diese Voreinstellungen entsprechen nicht unbedingt dem, was Sie wollen. Wenn es mehrere Möglichkeiten gibt, wird der Default (Standardvorgabe) als Großbuchstabe wie in `[Y/n]` angezeigt. Ihre Antworten zu jeder der Fragen sind beim nächsten Erzeugen eines Kernels aus diesem Quellenbaum der Default.

Beantworten Sie einfach die Fragen, indem Sie entweder mit der Eingabetaste die Vorgabe bestätigen oder `y` beziehungsweise `n` (und dann die `Eingabetaste`) eingeben. Nicht alle Fragen akzeptieren eine Ja/Nein-Antwort; eventuell müssen Sie eine Zahl oder einen anderen Wert eingeben. Eine Reihe von Konfigurationsfragen erlauben auch die Antwort `m` neben `y` und `n`. Mit dieser Option wird die entsprechende Kernelfunktion als ladbares Kernelmodul kompiliert, anstatt direkt in das Kernel-Image gelinkt zu werden. Ladbare Module, ermöglichen es, Teile des Kernels (wie zum Beispiel Gerätetreiber) nach Bedarf im laufenden System zu laden und zu entladen. Wenn Sie sich bei einer Option nicht sicher sind, dann geben Sie `?` ein; für die meisten Optionen bekommen Sie dann einen Informationstext zu sehen.

Eine Alternative zur Verwendung von `make config` ist `make xconfig`. Damit wird ein *X-Window*-basiertes Kernelkonfigurationsprogramm kompiliert und ausgeführt. Damit dies funktioniert, muss bei Ihnen das *X-Window*-System laufen, und die dazugehörigen *X11*- und *Tcl/Tk*-Bibliotheken müssen vorhanden sein und so weiter. Anstatt eine Reihe von Fragen zu stellen, können Sie mit diesem Hilfsprogramm Checkboxen verwenden, um die Kerneloptionen auszuwählen, die Sie einschalten möchten. Das System merkt sich Ihre Konfigurationsoptionen jedes Mal, wenn Sie das Programm laufen lassen, so dass Sie nicht erneut alle Optionen eingeben müssen, wenn Sie nur einige Funktionen hinzufügen oder entfernen wollen.

Außerdem gibt es noch `make menuconfig`, das die textbasierte *curses*-Bibliothek verwendet und damit eine ähnliche dialogbasierte Kernelkonfiguration ermöglicht, auch wenn Sie *X* nicht installiert haben. `make menuconfig` und `make xconfig` sind sehr viel komfortabler als `make config`, insbesondere da Sie zu einer Option zurückgehen und diese nachträglich ändern können, bis Sie Ihre Konfiguration abgespeichert haben.

Der folgende Auszug ist ein Teil einer Sitzung mit `make config`. Wenn Sie `make menuconfig` oder `make xconfig` verwenden, werden Sie den gleichen Optionen begegnen, die lediglich

benutzerfreundlicher präsentiert werden (und wir empfehlen Ihnen, wo immer möglich, diese Programme zu verwenden, weil man angesichts der Vielzahl von Konfigurationsoptionen leicht in Verwirrung geraten kann).

```
rm -f include/asm
( cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
#
# Using defaults found in .config
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/n/?]
*
* Loadable module support
*
Enable loadable module support (CONFIG_MODULES) [Y/n/?]
  Set version information on all module symbols (CONFIG_MODVERSIONS) [N/y/?]
  Kernel module loader (CONFIG_KMOD) [Y/n/?]
*
* Processor type and features
*
Processor family (386, 486, 586/K5/5x86/6x86/6x86MX, Pentium-Classic, ...
  defined CONFIG_MPENTIUMIII
Toshiba Laptop support (CONFIG_TOSHIBA) [N/y/m/?]
/dev/cpu/microcode - Intel IA32 CPU microcode support (CONFIG_MICROCODE) [M/n/y/?]
/dev/cpu/*/msr - Model-specific register support (CONFIG_X86_MSR) [M/n/y/?]
/dev/cpu/*/cpuid - CPU information support (CONFIG_X86_CPUID) [M/n/y/?]
High Memory Support (off, 4GB, 64GB) [4GB]
  defined CONFIG_HIGHMEM4G
Math emulation (CONFIG_MATH_EMULATION) [N/y/?]
MTRR (Memory Type Range Register) support (CONFIG_MTRR) [Y/n/?]
Symmetric multi-processing support (CONFIG_SMP) [Y/n/?]
*
* General setup
*
Networking support (CONFIG_NET) [Y/n/?]
...usw. ...
*** End of Linux kernel configuration.
*** Check the top-level Makefile for additional configuration.
*** Next, you may run 'make bzImage', 'make bzdisk', or 'make install'.
```

Wenn Sie sich mit der Hardware Ihres Systems auskennen, sollten die Fragen leicht zu beantworten sein. Wenn Ihnen eine Frage nichts sagt, dann handelt es sich wahrscheinlich um ein hoch spezielles Feature, das Sie nicht benötigen. Die folgenden Fragen stammen aus der Kernelkonfiguration der Version 2.4.4. Wenn Sie andere Patches eingespielt haben oder eine neuere Version des Kernels benutzen, kann es sein, dass weitere Fragen erscheinen. Beachten Sie, dass wir in der folgenden Liste nicht alle Konfigurationsoptionen zeigen; es gibt davon einfach zu viele, und die meisten sind selbsterklärend. Wir haben diejenigen herausgegriffen, bei denen Erklärungsbedarf besteht. Denken Sie daran, dass der Default oft die beste Antwort ist, wenn Sie sich nicht sicher sind, oder probieren Sie es mit ?.

Es sollte hier noch angemerkt werden, dass nicht alle Gerätetreiber in den Kernel gelinkt werden. Einige sind nur als ladbare Module verfügbar und werden getrennt vom Kernel verteilt. (Wie bereits erwähnt, können manche Treiber sowohl in den Kernel gelinkt als auch als Module übersetzt werden.) Ein bekannter Kerneltreiber, der nur als Modul verfügbar ist, ist der *Floppy-Streamer*-Treiber für *QIC-117*-Bandlaufwerke, die am Floppy-Controller angeschlossen werden.

Wenn Sie in der von `make config` präsentierten Liste keinen Treiber für Ihr Lieblingsgerät finden, ist es möglich, dass der Treiber als Modul oder getrennter Kernel-Patch verfügbar ist. Durchsuchen Sie die FTP-Server und die Archiv-CD-ROMs, wenn Sie nicht unmittelbar finden, was Sie suchen.

Prompt for development and/or incomplete code/drivers
: Antworten Sie mit *Ja* (`y`), wenn Sie neue Funktionen ausprobieren wollen, die von den Entwicklern noch nicht für stabil genug gehalten werden. Verwenden Sie diese Option nicht, wenn Sie nicht beim Testen neuer Funktionen mithelfen wollen.

Processor family(386, 486/Cx486, 586/K5/5x86/6x86/6x86MX, Pentium-Classic, Pentium-MMX, Pentium-Pro/Celeron/Pentium-II, Pentium-III/Celeron/Coppermine, Pentium-4, K6/K6-II/K6-III, Ahlon/Duron/K7, Crusoe, Winchip-C6, Winchip2, Winchip-2A, Winchip3, CyrixIII/C3) [Pentium-III/Celeron/Coppermine]
: Hier geben Sie an, welchen CPU-Typ Sie haben. Der Kernel wird dann mit speziellen Optimierungen für diesen Prozessortyp kompiliert. Beachten Sie, dass der Kernel eventuell nicht funktioniert, wenn Sie hier einen höheren Prozessortyp angeben, als Sie eigentlich haben. Der Pentium II MMX ist übrigens eine 686-CPU, keine 586-CPU.

Math emulation
: Antworten Sie mit Ja (`y`), wenn Sie keinen mathematischen Koprozessor in Ihrem Rechner haben. Dies ist notwendig, damit der Kernel einen Koprozessor emulieren kann.

Symmetric multi-processing support
: Hiermit wird die Kernelunterstützung für Systeme mit mehr als einer CPU eingeschaltet. Wenn Sie einen solchen Rechner haben, dann sagen Sie hier Ja (`y`), ansonsten Nein (`n`).

Enable loadable module support
: Schaltet die Unterstützung für dynamisch ladbare Module ein. Sagen Sie hier Ja (`y`).

Set version information on all symbols for modules
: Mit dieser Option ist es möglich, ein Modul, das für eine Kernelversion kompiliert worden ist, mit einer anderen zu verwenden. Das bringt aber eine Reihe von Problemen mit sich, sagen Sie hier deswegen Nein (`n`), wenn Sie nicht genau wissen, was Sie tun.

Kernel module loader
: Wenn Sie diese Option aktivieren, kann der Kernel dynamische Module bei Bedarf automatisch laden und entladen.

Networking support
: Antworten Sie mit Ja (y), wenn Sie irgendeine Form von Netzwerkunterstützung im Kernel haben wollen (einschließlich TCP/IP, SLIP, PPP, NFS usw.).

PCI support
: Schalten Sie diese Option ein, wenn Ihre Hauptplatine einen PCI-Bus verwendet und Sie PCI-Karten in Ihrem Rechner installiert haben. Zur Erkennung und Aktivierung der PCI-Geräte wird das PCI-BIOS verwendet; eine Kernelunterstützung dafür ist für die Verwendung jedes PCI-Gerätes in Ihrem System notwendig.

System V IP
: Mit einem Ja auf diese Frage binden Sie die Unterstützung für die IPC-Funktionen (Inter-Process Communication) von *System V* ein. Dazu gehören zum Beispiel `msgrcv` und `msgsnd`. Einige Programme, die von *System V* portiert wurden, brauchen die IPC-Funktionen; Sie können mit Ja antworten – es sei denn, Sie haben eine starke Aversion gegen IPC.

Sysctl support
: Diese Option weist den Kernel an, das dynamische Ändern von Kernelparametern zu ermöglichen, ohne erneut zu booten. Sie sollten diese Option aktivieren, sofern Sie nicht sehr wenig Speicher haben und die extra 8 KByte nicht entbehren können.

Parallel port support
: Schalten Sie diese Option ein, wenn Sie eine parallele Schnittstelle in Ihrem System haben und von *Linux* aus darauf zugreifen wollen. *Linux* kann die parallele Schnittstelle nicht nur für Drucker, sondern auch für PLIP (ein Netzwerkprotokoll für parallele Leitungen), ZIP-Laufwerke, Scanner und andere Geräte verwenden. In den meisten Fällen brauchen Sie noch einen weiteren Treiber.

Normal floppy disk support
: Antworten Sie hier mit Ja (y), es sei denn, Sie wollen keine Unterstützung für Diskettenlaufwerke (das kann auf Rechnern, auf denen Diskettenunterstützung nicht notwendig ist, ein bisschen Speicher sparen).

Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support
: Antworten Sie hier mit Ja (y), es sei denn, Sie benötigen keine Unterstützung für IDE/MFM/RLL-Festplattenlaufwerke. Wenn Sie mit Ja geantwortet haben, werden Sie nach den Gerätetypen (Festplatten, CD-ROM-Laufwerke, Bandlaufwerke und Diskettenlaufwerke) gefragt, die über den IDE-Treiber angesprochen werden sollen. Wenn Sie keine IDE-Festplatten, ATAPI-CD-ROM-Laufwerke und sonstige ATAPI-Geräte, sondern nur SCSI-Geräte haben, kann diese Option abgeschaltet werden.

XT harddisk support
: Antworten Sie hier nur mit Ja, wenn Sie einen älteren XT-Festplatten-Controller haben (und ihn auch mit Ihrem *Linux*-System verwenden wollen).

Parallel port IDE device support
: Diese Option schaltet die Unterstützung für Geräte mit IDE-Befehlssatz ein, die an die parallele Schnittstelle angeschlossen werden. Dazu gehören beispielsweise portable CD-ROM-Laufwerke.

Networking options
: Wenn Sie Netzwerkunterstützung (siehe oben) gewählt haben, werden Ihnen hier eine Reihe von Fragen über die gewünschten Netzwerkoptionen gestellt. Wenn Sie keine speziellen Bedürfnisse haben (in diesem Fall werden Sie ohnehin wissen, wie Sie die Fragen zu beantworten haben), verwenden Sie die Default-Antworten. Eine Reihe von Fragen sind etwas esoterisch (zum Beispiel `IP: Disable Path MTU Discovery`), und die Defaults sollten in fast allen Fällen verwendet werden.

SCSI support
: Wenn Sie irgendeinen SCSI-Controller in Ihrem System haben, antworten Sie mit Ja. Sie werden eine Reihe von Fragen zu den SCSI-Geräten in Ihrem System gestellt bekommen; klären Sie vorab, welche Hardware Sie installiert haben. All diese Fragen drehen sich um bestimmte SCSI-Controller-Chips und -Platinen. Wenn Sie sich nicht sicher sind, was für eine Art von SCSI-Controller Sie haben, lesen Sie in der Hardwaredokumentation nach oder befragen Sie die *Linux-HOWTO*-Dokumente. Sie werden auch gefragt, ob Sie Unterstützung für SCSI-Festplatten, -Bandlaufwerke, -CD-ROMs und andere Geräte haben wollen; geben Sie hier die richtigen Optionen für Ihre Hardware an. Ohne SCSI-Hardware oder der Notwendigkeit einer SCSI-Emulation sollten Sie mit Nein antworten; dadurch wird der kompilierte Kernel erheblich kleiner.

Network device support
: Hier kommen eine Reihe von Fragen über spezielle von *Linux* unterstützte Netzwerkkarten. Wenn Sie eine Ethernetkarte (oder eine andere Netzwerkkarte) haben, schalten Sie die Optionen für Ihre Hardware ein. Wie bei den SCSI-Geräten sollten Sie Ihre Hardwaredokumentation oder die *Linux-HOWTO*-Dokumente (wie zum Beispiel das *Ethernet-HOWTO*) lesen, um herauszufinden, welcher Treiber zu Ihrer Netzwerkkarte passt.

Amateur Radio support
: Diese Option schaltet die grundlegende Unterstützung für Netzwerkverbindungen über öffentliche Funkfrequenzen wie etwa CB ein. Wenn Sie die entsprechende Ausrüstung haben, dann aktivieren Sie diese Option. Lesen Sie auch die *AX25-* und *HAM-HOWTOs*.

ISDN subsystem
: Wenn Sie ISDN-Hardware in Ihrem System haben, dann aktivieren Sie diese Option und wählen den passenden Hardwaretreiber für Ihre Hardware aus. In den meisten Fällen sollten Sie dann auch *Support synchronous PPP* auswählen. (Siehe hierzu auch *PPP über ISDN-Leitungen*.)

Old CD-ROM drivers
: Hier folgen eine Reihe von Fragen zu den speziellen CD-ROM-Laufwerken, die der Kernel unterstützt. Dazu gehören zum Beispiel das *Sony-CDU31A/33A*, *Mitsumi*, CD-ROM-Laufwerke von *SoundBlaster Pro* und so weiter. Wenn Sie einen SCSI- oder IDE-CD-ROM-Controller haben (und die Unterstützung für diesen eingeschaltet ist), müssen Sie keine dieser Optionen aktivieren. Einige CD-ROM-Laufwerke haben eigene Schnittstellenkarten, für die mit diesen Optionen die Treiber aktiviert werden.

Character devices
: *Linux* unterstützt eine Reihe von speziellen Zeichen-Geräten wie serielle und parallele Schnittstellen-Controller, *QIC-02*-Bandlaufwerke sowie Mäuse mit proprietären Schnittstellen (also keine Mäuse, die an die serielle Schnittstelle angeschlossen werden, wie etwa die *Microsoft Serial Mouse*). In diesem Abschnitt finden Sie auch die Unterstützung für Joysticks und die *Video for Linux*-Treiber, die Video- und Framegrabber-Hardware unterstützen. Aktivieren Sie die für Ihre Hardware passenden Optionen.

`Filesystems`
: Es folgen eine Reihe von Fragen zu den Dateisystemen, die der Kernel unterstützt. Im Abschnitt *Mit Dateisystemen arbeiten* sind wir auf die Dateisysteme eingegangen, die *Linux* unterstützen kann. Hier können Sie bestimmen, für welche Dateisysteme Sie die Unterstützung einbinden möchten. Die meisten Systeme sollten die Dateisysteme *Second Extended* und */proc* unterstützen. Wenn Sie direkt von *Linux* aus auf Ihre *DOS*-Daten zugreifen wollen, sollten Sie die Unterstützung für das *DOS*-Dateisystem einbinden. Für den Zugriff auf CD-ROM-Daten brauchen Sie das ISO-9660-Dateisystem (das die meisten CD-ROMs benutzen).

`Console drivers`
: Stellen Sie sicher, dass Sie hier zumindest `VGA text console` angewählt haben, ansonsten können Sie Ihr *Linux*-System nicht von der Konsole aus benutzen.

`Sound card support`
: Wenn Sie auf diese Frage mit Ja antworten, werden Ihnen mehrere Fragen zu Ihrer Soundkarte gestellt sowie zu den Treibern, die Sie installieren möchten, ferner zum IRQ und zur Adresse Ihrer Soundkarte.

`Kernel hacking`
: Dieser Abschnitt enthält Optionen, die nur dann nützlich sind, wenn Sie vorhaben, selbst am *Linux*-Kernel zu arbeiten. Dazu gehört auch die Fehlerdiagnose. Wenn Sie das nicht vorhaben, sagen Sie hier Nein (`n`).

Nachdem Sie `make config` (oder eines der anderen Programme) beendet haben, werden Sie aufgefordert, das *Top-Level-Makefile*, also */usr/src/linux/Makefile*, zu editieren. In den meisten Fällen wird das gar nicht notwendig sein. Wenn Sie vorhaben, Compiler-Optionen für den Kernel oder die Voreinstellungen für das *Root*-Verzeichnis oder den Videomodus zu ändern, können Sie das *Makefile* entsprechend anpassen.

Wenn Sie eine komplette Neukompilierung des Kernels erzwingen möchten, sollten Sie an dieser Stelle `make clean` aufrufen. Damit werden alle Objektdateien aus diesem Verzeichnisbaum entfernt, die bei einer früheren Kompilierung erzeugt wurden. Wenn Sie von diesem Verzeichnisbaum aus noch nie einen Kernel erstellt haben, können Sie sich diesen Schritt wohl ersparen (obwohl damit kein Schaden angerichtet wird). Falls Sie den Kernel nur an wenigen Stellen geändert haben, können Sie diesen Schritt auslassen, damit nur die geänderten Dateien neu kompiliert werden. Auf jeden Fall stellen Sie mit dem Aufruf von `make clean` einfach sicher, dass der ganze Kernel von Grund auf neu kompiliert wird. Falls Sie irgendwelche Zweifel haben, geben Sie diesen Befehl ein, um sicherzugehen.

Jetzt sind Sie soweit, dass Sie den Kernel kompilieren können. Rufen Sie dazu `make bzImage` auf. Es empfiehlt sich, den Kernel zu erstellen, wenn die Systemlast nur gering ist, so dass der größte Teil des Arbeitsspeichers für die Kompilierung genutzt wird. Falls weitere Benutzer auf dem System arbeiten oder Sie eine große Anwendung (wie das *X-Window*-System oder einen anderen Compiler-Lauf) starten, kann die Kernelkompilierung bis zum Kriechgang abgebremst werden. Das Zauberwort heißt hier Arbeitsspeicher. Wenn ein System wenig Speicher hat und anfängt, Seiten auszulagern, dann wird die Kompilation auf jeden Fall langsam, egal, wie schnell der Prozessor ist.

Die Zeit für die Kernelkompilierung kann irgendwo zwischen einigen Minuten und einigen Stunden betragen – das hängt von Ihrer Hardware ab. Der Kernel umfasst eine ganze Menge Code – mittlerweile mehr als 30 MByte –, seien Sie also nicht überrascht. Langsamere Systeme mit vier MByte an RAM (oder weniger) müssen mehrere Stunden für die komplette Kompilierung ansetzen; schnellere Rechner mit mehr Speicher können nach weniger als einer Viertelstunde fertig sein. Aktuelle Systeme brauchen nur ein paar Minuten. Hier zeigen sich die Unterschiede in der Leistungsfähigkeit zwischen den verschiedenen Generationen der Systeme.

Wenn während der Kompilierung Fehler oder Warnungen auftreten, können Sie nicht davon ausgehen, dass der kompilierte Kernel korrekt funktioniert. In den meisten Fällen wird die Kompilierung beim Auftreten eines Fehlers abgebrochen. Fehler können nach verkehrt eingespielten Patches, bei Problemen mit `make config` oder aufgrund von echten Fehlern im Code auftreten. In den *sicheren* Versionen des Kernels treten Programmierfehler nur äußerst selten auf, während die Entwicklerkernel und neue Treiber im Teststadium durchaus fehlerhaft sein können. Im Zweifelsfall sollten Sie den Kernelverzeichnisbaum komplett löschen und noch einmal von vorn beginnen.

Nach der Kompilierung finden Sie die Datei *bzImage* im Verzeichnis */usr/src/linux/arch/i386/boot*. Diese Kopie des Kernels wird so benannt, weil sie ein ausführbares Abbild des Kernels ist, das intern mit `gzip2` komprimiert wurde. Beim Booten wird der Kernel sich selbst in den Arbeitsspeicher entpacken – Sie können nicht `bzip2` oder `bunzip2` von Hand anwenden! Auf diese Weise belegt der Kernel erheblich weniger Speicherplatz, und Kopien davon passen vielleicht auf eine Diskette. In früheren Kernelversionen wurden die beiden Kompressionsalgorithmen `gzip` und `bzip2` unterstützt; ersterer führte zu einer Datei namens *zImage*. Aber weil `bzip2` bessere Kompressionswerte

erreicht, sollte `gzip` nicht mehr verwendet werden; die damit komprimierten Kernel sind heutzutage meistens zu groß, um installiert zu werden.

Wenn Sie zu viele Funktionen ausgewählt haben, können Sie nach dem Kompilieren die Meldung `kernel too big` bekommen. Das passiert nur selten, weil Sie normalerweise auf einem Rechner nur sehr wenig Hardwaretreiber benötigen, aber es kann passieren. In diesem Fall gibt es nur einen Ausweg: Kompilieren Sie einen Teil der Kernelfunktionen als Module.

Anschließend sollten Sie mit dem neuen Kernel `rdev` aufrufen, um sicherzustellen, dass die Einstellungen für das Root-Dateisystem, den Videomodus und andere Parameter korrekt sind.

Wenn der neue Kernel erstellt ist, können Sie ihn zum Booten vorbereiten. Dazu müssen Sie entweder das Kernelabbild auf eine Boot-Diskette kopieren oder *LILO* so einrichten, dass der Kernel von der Festplatte bootet. Machen Sie den Kernel mit einer dieser beiden Methoden bootfähig, wenn Sie damit arbeiten wollen, und starten Sie das System neu.

Eine Warnung: Sie sollten immer einen bekanntermaßen lauffähigen Kernel zum Booten bereithalten. Entweder lassen Sie einen früheren Kernel in der Auswahl von *LILO*, oder Sie testen neue Kernel zunächst von Diskette. Ansonsten kann es passieren, dass Ihr System nicht mehr gebootet werden kann, wenn Sie einen fatalen Fehler begehen, beispielsweise einen sehr wichtigen Treiber im neuen Kernel vergessen.

Modularer gegenüber monolithischem Kern

Ein modularer Kernel kann Bestandteile bei Bedarf nachladen. Es können auch von vorn herein im Boot-Prozess Bestandteile geladen werden. Während das Nachladen von Gerätetreibern nur selten ein Sicherheitsrisiko darstellt, ist das Laden oder Entladen von Protokoll-Modulen ein nicht vernachlässigbares Sicherheitsrisiko. Ein denkbarer Angriff, der die Firewallfunktionen entlädt, aber die Routing-Funktionen aufrechterhält, macht eine Firewall funktionslos. Daher ist es empfehlenswert, auf gefährdeten Linux-Computern monolithische Kernel einzusetzen, oder zumindest die für die Sicherheit bedeutsamen Funktionen fest einzubinden.

Die Übersetzung der Module erfolgt analog zur Übersetzung des Kernels mit dem Aufruf

```
make modules
```

wodurch alle Module übersetzt werden. Die Module werden durch den Aufruf

```
make modules_install
```

in die richtigen Verzeichnisse installiert. Danach wird mit

```
depmod –a
```

die notwendige Datenbank erstellt.

Die modernen Kernel der 2.6-er-Reihe bieten eine spezielle Option, genannt *Extra Version*, die es erlaubt, die zu installierenden Module für verschieden Konfigurationen auseinander zu halten. Es werden Unterverzeichnisse in */lib/modules* erzeugt, deren Name aus der Kernelversionsnummer mit dem Zusatz aus *Extra Version* besteht.

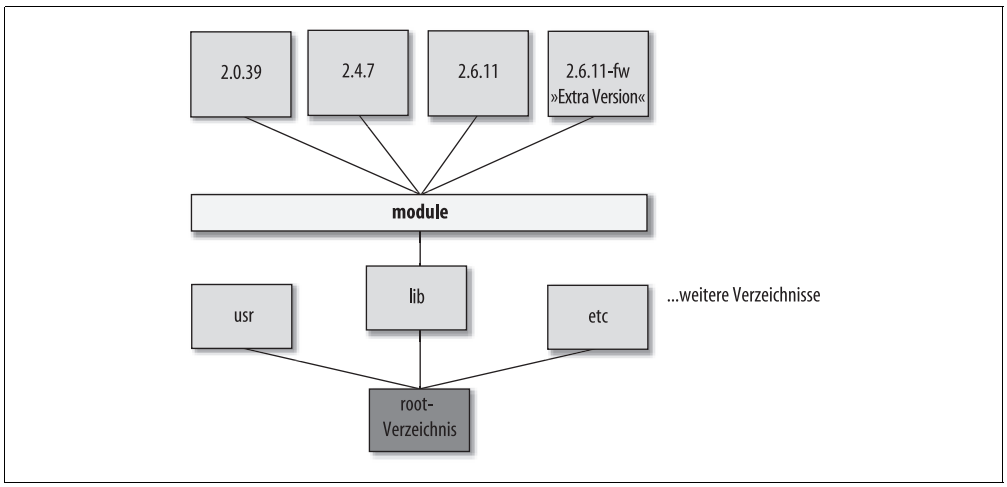

Abbildung 14-5: Verzeichnis der Module

Bereitstellung spezieller, normalerweise nicht vorhandener Eigenschaften

Der *Linux*-Kernel stellt ausgefeilte Sicherheitsmechanismen zur Verfügung, die in den häufigen Verwendungen von Linux-Computern keine Rolle spielen und auch einigen Aufwand beim Einsatz erfordern. Daher sind die von den Distributoren bereitgestellten Kernel nicht mit diesen speziellen Verfahren ausgerüstet. Wenn diese Möglichkeiten bereitgestellt werden, müssen sie meist beim Booten aktiviert werden, da sich sonst *Linux* nicht wie erwartet verhält, so bei *SELinux* mit dem Parameter

```
selinux=1
```

Verschlüsselung

Die aktuellen Kernelversionen der 2.6-er-Familie bieten diverse Verschlüsselungsalgorithmen. Alle diese Verfahren verwenden endlich lange Schlüssel und sind daher vom Prinzip her nicht vollkommen sicher. Der Aufwand zum Brechen ist allerdings sehr hoch. Die angebotenen Algorithmen werden beispielsweise bei IP-Sec-Verbindungen verwendet. Entsprechende Anwendungen müssen nicht die Verschlüsselung bereitstellen, sondern können auf Kernelfunktionen zurückgreifen. Sobald Kryptographie-Hardware allgemein verbreitet ist, kann der Kernel so eingerichtet werden, sie zu nutzen.

Verschlüsselte Dateisysteme

Linux kann verschlüsselte Dateisysteme anlegen und verwalten. Im Gegensatz zu gewöhnlichen Dateisystemen, die durch Manipulation an der Hardware – Starten von CD aus – zugänglich gemacht werden können, bieten verschlüsselte Dateisysteme auch hardwareorientierten Angriffen erheblichen Widerstand. Der Zugriff kann nur durch Eingabe eines Schlüssels erfolgen. Zuerst wird eine leere Datei angelegt

```
root# dd if=/dev/urandom of=~/.crypto bs=1024k count=10
```

die hier 10 MB groß ist. Danach wird die Datei verschlüsselnd eingebunden

```
root# losetup -e AES /dev/loop0 ~user/.crypto
password:
```

und ein Passwort vergeben. Diese Passwortvergabe wird nicht, wie sonst üblich, wiederholt. Eingabefehler werden also nicht bemerkt! Dann ein Dateisystem erzeugt:

```
root# mke2fs /dev/loop0
```

und eingehängt

```
root# mount -t ext2 /dev/loop0 ~user/cryptomount
```

Damit ist das verschlüsselte Dateisystem transparent zugänglich. Nach Gebrauch wird es abgekoppelt

```
root# umount /dev/loop0
```

und abgetrennt

```
root# losetup -d /dev/loop0
```

Danach kann nur noch mit

```
root# losetup -e AES /dev/loop0 ~user/.crypto
password:
root# mount -t ext2 /dev/loop0 ~user/cryptomount
```

auf die Daten zugegriffen werden.

Firewall

Die einfache Firewall-Standardfunktionalität ist bei allen von den Distributoren ausgelieferten *Linux*-Kerneln aktiviert. Die Funktionalität ist bei allen größeren Revisionen des *Linux*-Kernels wesentlich geändert worden. Wenn umfangreiche Firewall-Konfigurationsskripte vorhanden sind, können diese weiterbenutzt werden, wenn die älteren Mechanismen wie `iptables` in einem Kernel aktiviert werden, der schon `ipchains` beherrscht.