



# 16 Zentrale Protokollserver

Sobald viele UNIX- oder Linux-Rechner überwacht werden müssen, stellen viele Administratoren fest, dass eine manuelle Überwachung der Protokolle auf jedem einzelnen Rechner sehr mühsam und fehlerträchtig ist. In diesem Zusammenhang wurden bereits im zweiten Teil des Buches Werkzeuge zur automatischen Protokollanalyse vorgestellt. Diese Werkzeuge analysieren die Protokolle und können regelmäßig einen Bericht versenden oder in besonderen Fällen direkt alarmieren. Dennoch erwartet den Administrator unter Umständen eine tägliche Flut von Berichten in Form von E-Mails. Daher bietet sich eine zentrale Protokollierung und die Erzeugung eines einzigen Berichtes an.

Es gibt aber noch einen weiteren, unter Sicherheitsaspekten wichtigeren Grund für eine zentrale Protokollierung. Wenn tatsächlich ein Einbrecher in der Lage war, in einen Linux-Rechner einzubrechen, so wird er zunächst versuchen, die Spuren, die der Einbruch hinterlassen hat, zu entfernen. Dies betrifft häufig auch entsprechende Einträge in den Protokollen. Werden diese Protokolle lediglich lokal und unverschlüsselt gespeichert, so hat er leichtes Spiel. Er kann mit einem Editor wie zum Beispiel dem *vi* die Protokolldatei editieren. Dieser Editor erlaubt sogar das Editieren der Datei, obwohl der Protokolldienst die Datei weiterhin geöffnet hat, um weitere Meldungen an diese Datei anzuhängen. Viele Systemadministratoren verbannen daher auch den *vi* von sicherheitsrelevanten Rechnern. Diese Maßnahme ist jedoch ähnlich kritisch einzustufen wie die Entfernung eines Compilers. Bei einem manuellen Einbruch ist der Einbrecher meist in der Lage, recht schnell die entsprechenden Programme nachzuinstallieren. Automatische Einbrüche können durch benötigte aber fehlende Bestandteile auf dem System gestoppt werden.

Protokolliert jedoch der Rechner seine Meldungen nicht nur lokal, sondern über das Netzwerk auch zentral auf einem anderen Protokollserver, so kann der Einbrecher zwar die Spuren in den lokalen Protokolldateien entfernen, nicht jedoch die Meldungen auf dem zentralen Protokollserver löschen. Hierzu müsste zunächst wieder ein Angriff auf diesen mit anschließendem Einbruch erfolgen.

Für die Auswertung der Protokolle ist es sehr wichtig, dass die überwachten Rechner eine gemeinsame Zeitbasis verwenden. Ansonsten besteht die Gefahr, dass bei einem Angriff die Meldungen der unterschiedlichen Rechner nicht miteinander korreliert werden können. Die Meldung eines Angriffes durch einen Snort-Sensor und die Modifikation von Dateien und die Installation eines Root-Kits auf einem anderen Rech-

ner können nicht in einen kausalen Zusammenhang gestellt werden, wenn nicht die Zeiten der Meldungen bzw. Modifikationen vergleichbar sind. Im Folgenden wird daher die Installation eines Network Time Protocol (NTP) Zeit-Servers beschrieben.

## 16.1 Einrichtung der zentralen Protokollierung unter Linux

Die meisten Linux-Distributionen verwenden für die zentrale Systemprotokollierung eine Kombination aus zwei Diensten: *Syslogd* und *Klogd*. *Syslogd* ist der zentrale Protokolldienst unter Linux. Dieser Protokolldienst nimmt die Meldungen von verschiedenen anderen Systemen entgegen. Hierbei handelt es sich unter anderem um *Cron*, *E-Mail*, *News*, *Login*, *Lpd* etc. Der *Syslogd* ist typischerweise nicht zuständig für »neue« Dienste wie zum Beispiel einen Apache-Webserver, einen Squid-Proxy oder Samba. Der *Klogd* nimmt vom Kernel die Meldungen entgegen und leitet sie üblicherweise an den *Syslogd* weiter. Damit werden auch Meldungen des Linux-Kernels durch den *Syslogd* protokolliert.

Im Folgenden sollen kurz die wesentlichen Möglichkeiten der Konfiguration des *syslogd* und des *klogd* beschrieben werden. Anschließend wird ein weiterer *Syslogd*-Dämon beschrieben, der ersatzweise eingesetzt werden kann. Dieser modulare *syslogd* ist nicht Bestandteil einer aktuellen Distribution und muss daher manuell installiert werden. Er verfügt jedoch über einige Eigenschaften, die seinen Einsatz auf sicherheitsrelevanten Systemen rechtfertigt.

Der *syslogd* verfügt über lediglich eine Konfigurationsdatei. Dies ist die Datei */etc/syslog.conf*. Sie enthält Informationen darüber, welche Meldungen wo zu protokollieren sind. Hierzu wird die folgende Syntax verwendet:

```
facility.priority           location
```

Der Standard BSD *Syslogd*, der in den meisten aktuellen Linux-Distributionen eingesetzt wird, ist in der Lage, die folgenden *Facilities* zu unterscheiden:

- *auth*
- *authpriv*
- *cron*
- *daemon*
- *kern*
- *lpr*
- *mail*
- *mark* (Lediglich für interne Verwendung)
- *news*
- *security* (identisch mit *auth*, sollte jedoch nicht mehr eingesetzt werden)

- *syslog*
- *user*
- *uucp*
- *local0* bis *local7*

Als *priority* werden die folgenden Werte unterstützt:

- *debug*
- *info*
- *notice*
- *warning, warn* (identisch, sollte jedoch nicht mehr eingesetzt werden)
- *err, error* (identisch, sollte jedoch nicht mehr eingesetzt werden)
- *crit*
- *alert*
- *emerg, panic* (identisch, sollte jedoch nicht mehr eingesetzt werden.)

Bei einer Angabe wie zum Beispiel *cron.info* werden nun alle Meldungen von *cron* mit der Wertigkeit *info* und höher von dieser Regel protokolliert. Sollen spezifisch nur die Meldungen mit der Wertigkeit *info* protokolliert werden, so kann dies mit der Angabe *cron.=info* erfolgen. Außerdem ist die Negation möglich: *cron.!=info*. Zusätzlich besteht die Möglichkeit sowohl für die facility als auch für die priority, den Stern (\*) als Wildcard zu verwenden.

Leider besteht beim Standard BSD Syslog nicht die Möglichkeit, in Abhängigkeit von einem Muster, zum Beispiel einem regulären Ausdruck, die Protokollierung zu konfigurieren.

Nachdem die zu protokollierenden Meldungen in der linken Spalte der */etc/syslog.conf* ausgewählt wurden, muss der Ort der Protokollierung definiert werden. Hier stehen ebenfalls mehrere Möglichkeiten zur Verfügung:

- Normale Datei, z.B. */var/log/messages*
- Named Pipe, z.B. *|/var/log/my\_fifo*. Diese Named Pipe muss zuvor mit dem Befehl *mkfifo* erzeugt werden. Hiermit besteht die Möglichkeit, die Informationen ähnlich einer normalen Pipe an einen anderen Prozess weiterzuleiten.
- Terminal bzw. Konsole, z.B. */dev/tty10*
- Zentraler Protokollserver, z.B. *@logserver.example.com*
- Benutzer, z.B. *root*. Diese Personen erhalten die Meldungen auf ihrem Terminal, falls sie angemeldet sind. Es wird keine E-Mail versandt!
- Alle angemeldeten Benutzer \*

Im Folgenden sollen ein paar Beispiele angegeben werden:

```
# Kritische Kernelmeldungen auf der 10. virtuellen Konsole
# Notfälle direkt an root
```

```

kern.crit                /dev/tty10
kern.emerg               root
# Authentifizierungsvorgänge in einer geschützten Datei und zusätzlich
# zentral
authpriv.*               /var/log/secure
authpriv.*               @remotesyslog
# E-Mail getrennt
mail.*                   /var/log/maillog
# Alles andere in einer Datei
*.info;mail.none;authpriv.none /var/log/messages

```

Der Syslog erhält üblicherweise die zu protokollierenden Meldungen über den UNIX-Socket `/dev/log`. Dieser Socket wird vom Syslog beim Start erzeugt. Dies genügt in den meisten Fällen, da sämtliche lokalen Anwendungen auf diesen Socket zugreifen können. Bei einer Anwendung, die sich in einer Chroot-Umgebung befindet, besteht jedoch das Problem, dass ein derartiger Zugriff auf `/dev/log` nicht möglich ist. Der Syslogd muss dann bei seinem Start einen zusätzlichen UNIX-Socket in der Chroot-Umgebung erzeugen. Dies kann mit der Option `-a socket` erfolgen. Hiermit können bis zu 19 weitere Sockets definiert werden.

Mithilfe dieser UNIX-Sockets ist jedoch der Syslogd nicht in der Lage, Meldungen über das Netzwerk entgegenzunehmen. Hierzu muss zusätzlich ein Internet-Socket geöffnet werden. Die Protokollierung über das Netzwerk erfolgt beim Standard BSD Syslogd an den UDP Port 514. Die Konfiguration dieser Portnummer erfolgt beim Syslogd in der Datei `/etc/services`. Diese Datei enthält hierzu üblicherweise folgenden Eintrag:

```
syslog      514/udp
```

Leider erfolgt die Protokollierung nur mit UDP. Daher kann der Syslog nicht sicherstellen, dass sämtliche Meldungen tatsächlich den Empfänger erreichen. Einzelne Pakete können zum Beispiel aufgrund einer falsch konfigurierten Firewall oder einer Überlastung des Netzwerkes verloren gehen.

Damit der Syslogd nun diesen Port öffnet und Meldungen über diesen Internet-Socket annimmt, muss er mit der Option `-r` gestartet werden. Dann nimmt er jede Meldung, die an diesen Socket gesendet wird, an und protokolliert sie entsprechend seiner eigenen Konfigurationsdatei `/etc/syslog.conf`. Es besteht leider nicht die Möglichkeit, im Syslogd die Clients, die die Meldungen senden, zu authentifizieren oder aufgrund Ihrer IP-Adresse einzuschränken. Dies kann lediglich durch das Setzen intelligenter Paketfilterregeln (iptables) geschehen. Ein Angreifer könnte daher den Protokollserver mit Meldungen (auch mit gefälschten Absenderadressen) bombardieren und so die Protokolle mit unsinnigen Meldungen füllen. Im Weiteren wird der modulare Syslog `msyslog` vorgestellt, der dieses Sicherheitsproblem nicht hat.

Der `klogd` weist keine eigene Konfigurationsdatei auf. Die Meldungen des Kerns werden vom `klogd` an den `syslogd` weitergeleitet, der sie aufgrund seiner Konfigurationsdatei protokolliert. Der `klogd` verfügt jedoch über eine interessante Startoption:

-c X. Diese Option erlaubt die Definition einer Priority X. Kernel-Meldungen, die mindestens diese Wertigkeit aufweisen, werden direkt vom `klogd` auf der Konsole protokolliert. Dies ist jedoch den Geschwindigkeitsbeschränkungen der Konsole unterworfen. Die Konsole wird als serielles Terminal mit einer üblichen Geschwindigkeit von 38.400 Baud emuliert.

Die unterschiedlichen Distributionen verwalten die Startoptionen des `klogd` und des `syslogd` an unterschiedlichen Positionen. Red Hat verwendet die Datei `/etc/sysconfig/syslog`. SUSE verwendet in Abhängigkeit von der Version entweder die Datei `/etc/rc.config` oder eine Datei im Verzeichnis `/etc/sysconfig`.

Der Standard BSD Syslogd ist also bereits in der Lage, seine Meldungen über das Netzwerk auf einem zentralen Logserver zu protokollieren. Leider weist er weder Mustererkennung mit regulären Ausdrücken noch Authentifizierung oder Verschlüsselung auf. Daher handelt es sich hierbei nicht um einen idealen Protokoll-dienst für sicherheitsrelevante Umgebungen.

## 16.2 Modular Syslog

Der modulare Syslog `msyslog` von der Firma Core Security Technologies (ehemals Core SDI, <http://www.corest.com>) ist ein Ersatz für den Standard BSD Syslog und den `klogd` unter Linux, die von den meisten Linux-Distributionen eingesetzt werden. Der Msyslogd ist komplett rückwärtskompatibel zum Standard BSD Syslog und unterstützt zusätzlich folgende Funktionen:

- Annahme von Meldungen
  - Kernel-Meldungen über das BSD Kernel-Interface und das Linux `/proc/kmsg` Interface.
  - Systemmeldungen über das UNIX-Socket-Interface `/dev/log`
  - Meldungen über eine TCP- oder eine UDP-Verbindung auf einem beliebigen Port
  - Meldungen aus einer beliebigen Datei oder Named Pipe
- Ausgabe von Meldungen
  - kompatibel zum klassischen BSD Syslogd
  - mit einem kryptografischen Hash
  - auf einem anderen Rechner mit TCP oder UDP
  - in Abhängigkeit von einem regulären Ausdruck
  - in einer MySQL- oder PostgreSQL-Datenbank
- Zusätzlich existiert das Werkzeug `peock`, um die Integrität der Protokolle zu testen.

Weitere Programme, die einen möglichen Ersatz des BSD Syslogd darstellen, sind `mbsyslog` (GPL, <http://michael.bacarella.com/projects.shtml>), `Metalog` (GPL, [563](http://meta-</a></p>
</div>
<div data-bbox=)

*log.sourceforge.net/*) und *syslog-ng* (<http://www.balabit.hu/en/downloads/syslog-ng/>). Die wesentlichen Unterschiede zwischen diesen weiteren Programmen und dem modularen Syslogd liegen in den oben genannten Punkten. Keines der weiteren Programme unterstützt alle diese Fähigkeiten. Entweder sind sie nicht kompatibel zum Standard Syslog oder sie unterstützen nicht die automatische Verkettung der Daten mit einem Hash oder sie bieten nicht die Protokollierung über TCP oder in einer Datenbank.

Der Secure Syslogd ist eine Vorgängerversion des Modular Syslogd und wurde auch von Core Security Technologies erzeugt. Er wies auch alle diese Fähigkeiten auf.

### 16.2.1 Installation von msyslogd

Der modulare Syslogd *msyslogd* kann entweder von der Homepage von Core Security Technologies heruntergeladen werden (<http://www.corest.com>) oder von der Sourceforge-Projektseite (<http://sourceforge.net/projects/msyslog/>). Der modulare Syslogd wird unter der BSD-Lizenz vertrieben. Es handelt sich also um freie Open-Source-Software, die sowohl zu privaten als auch zu kommerziellen Zwecken im Rahmen der Lizenz eingesetzt werden darf. Auf der Sourceforge-Projektseite werden üblicherweise sowohl das Quelltextpaket als auch RPM-Pakete angeboten. Die RPM-Pakete sind an die Distribution von Red Hat Linux angepasst. Nach einer leichten Anpassung des im Paket enthaltenen Startskriptes */etc/rc.d/init.d/msyslogd* sollte das RPM-Paket jedoch auch auf SUSE Linux-Distributionen lauffähig sein. Für die Debian-Distribution existiert ebenfalls ein Paket. Eine Übersetzung des Quelltextarchives sollte daher in den seltensten Fällen nötig sein. Falls doch, so genügt meistens ein Aufruf von

```
# ./configure
# make
# make install
```



#### Achtung:

Die RPM-Pakete installieren den *msyslogd* unter dem Namen *msyslogd*. Dieser greift jedoch trotzdem auf die Datei */etc/syslog.conf* zu. Die Manpages wurden jedoch ebenfalls in den RPM-Paketen umbenannt: *msyslogd.8* und *msyslog.conf.5*. Das Quelltextarchiv installiert den *msyslogd* als *syslogd!*. Die ganze mitgelieferte Dokumentation geht davon aus, dass der *msyslogd* unter letzterem Namen zu finden ist. Im Weiteren wird in diesem Buch aber davon ausgegangen, dass das RPM-Paket installiert wurde und der Befehl *msyslogd* lautet.

## 16.2.2 Konfiguration von *msyslogd*

Bei der Konfiguration von *msyslogd* soll zunächst auf die hohe Kompatibilität zum Standard BSD Syslog hingewiesen werden. Das wird im nächsten Abschnitt noch genauer betrachtet.

Der wesentliche Unterschied bei der Verwendung des *msyslogd* liegt in seinem Aufruf. Aufgrund seines modularen Aufbaues ist es erforderlich, alle benötigten Eingabe-Module beim Aufruf zu benennen. Der *msyslogd* verfügt über die folgenden Eingabe-Module: *bsd*, *doors*, *linux*, *streams*, *tcp*, *udp* und *unix*. Des Weiteren besitzt er folgende Ausgabe-Module: *classic*, *mysql*, *peo*, *pgsql*, *regex*, *tcp* und *udp*.

### Kompatibel zum Standard BSD Syslogd

Von großem Vorteil beim Einsatz des *msyslogd* ist die Tatsache, dass dieser Syslogd abwärtskompatibel zum BSD Syslogd ist. Das bedeutet, dass bei einem Aufruf des Befehls `msyslogd` dieser sich auf einem Linux-System so verhalten wird, wie es von einem Syslogd erwartet wird.

Der Prozess liest automatisch seine Konfigurationsdatei `/etc/syslog.conf` ein und liest die Kernel-Meldungen von `/proc/kmsg` und öffnet den Socket `/dev/log`.

Die Konfigurationsdatei `/etc/syslog.conf` wird vom *msyslogd* identisch dem BSD Syslogd interpretiert. Leider existieren noch einige Fehler in der Implementation der Kompatibilität. Die entsprechenden Voraussetzungen sind jedoch sehr selten gegeben. Lesen Sie die Manpage (*msyslog.conf.5*) für weitere Informationen.

Die Kompatibilität mit den BSD Syslogd-Funktionen wird vom `%classic`-Modul bereitgestellt. Sobald die Funktionalität des *msyslogd* ausreichend getestet wurde, sollte die Konfigurationsdatei an die neue Syntax angepasst werden, so dass anschließend auch die neuen Fähigkeiten (siehe nächster Abschnitt) genutzt werden können. Dazu ist an den entsprechenden Stellen ein Aufruf des `%classic`-Moduls erforderlich. Im nächsten Listing wird die klassische Konfiguration aus dem nachfolgenden Listing mit Modulen gezeigt.

```
# Kritische Kernelmeldungen auf der 10. virtuellen Konsole
# Notfälle direkt an root
kern.crit                %classic  /dev/tty10
kern.emerg              %classic  root
# Authentifizierungsvorgänge in einer geschützten Datei und zusätzlich
# zentral
authpriv.*              %classic  /var/log/secure
authpriv.*              %classic  @remotesyslog
# E-Mail getrennt
mail.*                  %classic  /var/log/maillog
# Alles andere in einer Datei
*.info;mail.none;authpriv.none    %classic  /var/log/messages
```

Um auch den Aufruf des *msyslogd* an seine neuen Fähigkeiten anzupassen, sollten die Eingabe-Module *linux* (für das Kernel-Interface */proc/kmsg*) und *unix* (für die Bereitstellung des */dev/log* Sockets) geladen werden. Erfolgte der Aufruf bisher zum Beispiel mit

```
syslogd -m 0
```

dann sollte nun der Aufruf zusätzlich die entsprechenden Module laden:

```
msyslogd -m 0 -i linux -i unix
```

Bei der Verwendung des RPM-Paketes wird automatisch ein Startskript */etc/rc.d/init.d/msyslog* angelegt, welches die Startoptionen aus der Datei */etc/sysconfig/msyslog* liest. Diese Datei enthält folgende Optionen:

```
CONFIG=""           # example: "-f /etc/syslog.conf"
DEBUG=""           # example: "-d 20"
MARK=""            # example: "-m 20"
IM_BSD=""          # example: "-i bsd"
IM_DOORS=""        # example: "-i doors"
IM_LINUX="-i linux" # example: "-i linux"
IM_STREAMS=""      # example: "-i streams"
IM_TCP=""          # example: "-i tcp accepted.host.com 514"
IM_UDP=""          # example: "-i udp:514"
IM_UNIX="-i unix"  # example: "-i unix"
                   # Look in the im_*.8 man pages for more details on these
                   # options
```

Hier sind bereits standardmäßig die beiden Module eingetragen. Daher genügt es, lediglich den klassischen *syslog*-Dienst zu deaktivieren und den *msyslog* zu aktivieren. Dies erfolgt bei einer Red Hat Linux-Distribution recht einfach mit:

```
# chkconfig --del syslog
# chkconfig --add msyslog
# chkconfig msyslog on
```

Wenn Sie eine SUSE-Distribution verwenden, unterscheidet sich das Vorgehen in Abhängigkeit von der Version. Bitte lesen Sie die entsprechende Besprechung im Kapitel 9.2, »Snort« bzw. im SUSE-Handbuch.

### Nutzung der neuen Fähigkeiten

Wird nur der von den Distributionen verwendete Standard BSD Syslog durch den modularen Syslog ausgetauscht, so ergeben sich noch keine Vorteile. Die zusätzlichen Funktionen des modularen Syslog müssen aktiviert werden, damit sie genutzt werden können. Hierzu sind Anpassungen der Konfigurationsdatei */etc/syslog.conf* und des Aufrufes des Befehls *msyslogd* erforderlich. Zunächst sollen die unterschiedlichen Optionen beim Aufruf erläutert werden, bevor dann die Anpassung der Kon-

figurationsdatei beschrieben wird. Abgeschlossen wird die Betrachtung mit der Integritätsüberprüfung der Dateien mit dem `peochk`-Befehl.

Beim Aufruf des `msyslogd` können die folgenden Optionen angegeben werden:

- `-d level`. Debugging-Level
- `-f file`. Konfigurationsdatei (Default: `/etc/syslog.conf`)
- `-m interval`. Intervall in Minuten zwischen `-MARK-` Meldungen (Default: 20, 0 schaltet ab)
- `-u`. Annahme von Meldungen über UDP-Port (ähnlich `-r` bei BSD Syslogd)
- `-i module`. Auswahl der Eingabe-Module

Hierbei stehen nun folgende Eingabe-Module zur Verfügung:

- `bsd`. Kernel-Meldungen eines BSD-Systems
- `doors [pfad]`. Eingabe-Modul für Doors Inter Process Call-(IPC-)Meldungen (Solaris)
- `streams [pfad]`. Eingabe-Modul für Streams Inter Process Call-(IPC-)Meldungen
- `tcp`. Eingabe-Modul für Meldungen über TCP-Verbindungen
- `udp`. Eingabe-Modul für Meldungen über UDP-Verbindungen
  - `-h host`. Nimmt auf der Adresse Anfragen an
  - `-p port`. TCP-Port, der geöffnet wird
  - `-a`. Extrahiert den Rechnernamen (FQDN)
  - `-q`. Extrahiert den Rechnernamen (nicht FQDN)
- `file`. Liest aus einer Datei oder Pipe
  - `-f datei`. Liest aus der angegebenen Datei
  - `-p pipe`. Liest aus der angegebenen Pipe
  - `-n Name`. Legt den Namen fest
- `unix [pfad]`. Liest aus einem UNIX-Socket (Default: `/dev/log`)
- `linux`. Liest die Linux-Kernel-Meldungen
  - `-c level`. Setzt den Log-Level für die Konsole
  - `-C level`. Definiert den Log-Level eines laufenden `msyslog` neu und beendet sich
  - `-k ksym`. Kernel-Symboltabelle (Default: `/proc/ksyms`)
  - `-s`. Verwendet das Syscall-Interface (Default: `/proc/kmsg`)
  - `-x`. Keine Übersetzung der Kernel-Symbole

Um nun den `msyslogd` auf einem lokalen Rechner als Protokolldienst einzusetzen, genügt meist die Angabe von:

```
msyslogd -i unix -i linux
```

Wurde nun eine Anwendung, zum Beispiel ein DNS-Server, in einem Chroot-Verzeichnis installiert, so benötigt dieser Dienst ein UNIX-Socket zur Protokollierung. Er ist nicht in der Lage, aus seinem Chroot-Verzeichnis auf den UNIX-Socket `/dev/log` zuzugreifen. Dann erfolgt der Aufruf als:

```
msyslogd -i unix -i 'unix /chroot_dns/dev/log' -i linux
```

Der *msyslogd* wird dann beim Start den zusätzlichen UNIX-Socket anlegen.

Wenn der *msyslogd* zusätzlich Daten über das Netzwerk entgegennehmen soll, so wird er aufgerufen mit:

```
msyslogd -i unix -i linux -i 'tcp -p 8000'
```

Hierbei ist jedoch zu beachten, dass die Übertragung mit UDP wie auch mit TCP unverschlüsselt und nicht authentifiziert erfolgt. Daher sollte zur eigentlichen Übertragung *stunnel* eingesetzt werden.

```
msyslogd -i unix -i linux -i 'tcp -h localhost -p 8001'
stunnel -d 8000 -r localhost:8001
```

Weitere Informationen über *stunnel* werden im Kapitel über den Netzwerkeinsatz von Snort besprochen (Abschnitt »Stunnel« auf S. 519).

Bei der Anpassung der Konfigurationsdatei `/etc/syslog.conf` an die neuen Fähigkeiten ist sicherlich die Signatur der Protokolle mit dem PEO-Protokoll die wichtigste Funktion. Wir werden zunächst jedoch die weiteren Ausgabe-Module betrachten und zum Schluss die Signatur besprechen.

Zur Kompatibilität mit dem Standard BSD Syslog verfügt der *msyslogd* über das Modul `%classic`. Dieses Modul verfügt über die identischen Fähigkeiten. Es erlaubt die Protokollierung in einer Datei bei Angabe eines Dateinamens (`%classic /var/log/messages`). Die Protokollierung auf den Terminals sämtlicher angemeldeter Benutzer erfolgt mit `%classic *` und über eine kommaseparierte Liste können bestimmte Benutzer ausgewählt werden (z.B. `%classic xapfel,ybirne,zorange`). Eine Protokollierung über das Netz auf einem anderen Rechner:syslog erfolgt mit der Angabe von `%classic @logserver.example.net`.

Zusätzlich ist der *msyslogd* in der Lage, auf einem beliebigen UDP- oder TCP-Port die Protokollierung durchzuführen. Im Zusammenhang mit der Protokollierung über TCP bietet sich die Verschlüsselung mit *stunnel* an (s.o.). Das Modul `%udp` unterstützt hierbei die Angabe des Rechners (`-h`) und des Ports (`-p`). Zusätzlich kann mit der Option `-a` definiert werden, ob der eigene Rechnername in der Meldung übertragen wird. Beispiel:

```
kern.info %udp -a -h logserver.example.net -p 514
```

Das Modul `%tcp` unterstützt zusätzlich zu diesen die weiteren Optionen `-m` und `-s`. Hiermit ist es möglich, eine maximale Zeit für Verbindungswiederholungen in Se-

kunden zu definieren und einen Puffer für die Aufnahme der noch nicht gesandten Meldungen zu generieren.

```
# TCP Verbindung zu logserver, Retry 30 Sekunden, Puffer 16384 Bytes
kern.info    %tcp -a -h logserver.example.net -p 8000 -m 30 -s 16384
```

Bei der Wahl zwischen UDP und TCP ist zu berücksichtigen, dass UDP-Pakete verloren gehen können. Zusätzlich bieten weitere Werkzeuge (z.B. *stunnel*) die Möglichkeit, TCP-Verbindungen mit SSL zu verschlüsseln.

Ein weiteres sehr interessantes Ausgabe-Modul ist *%regex*. Dieses Modul erlaubt, die Ausgabe der Meldungen in Abhängigkeit von einem regulären Ausdruck zu steuern.

Der normale BSD Syslog erlaubt lediglich eine Verwaltung der Protokollmeldungen über die Angabe der *facility* und der *priority*. Eine feinere Verwaltung der Meldungen und ihre Aufteilung auf unterschiedliche Protokolldateien ist nicht möglich. Der *msyslogd* ermöglicht mit dem Modul *%regex* die Verwendung von regulären Ausdrücken als Filter. Diese können getrennt auf die Nachricht (-m), den Rechnernamen (-h, das Datum (-d) und die Uhrzeit (-t) angewendet werden. Zusätzlich kann auch der reguläre Ausdruck mit -v invertiert werden (ähnlich *grep*).

Sollen auf einem zentralen Protokollserver die Protokolle für die verschiedenen Protokollclients (*client1* und *client2*) in unterschiedlichen Dateien abgelegt werden, so kann das durch folgende Einträge erfolgen:

```
*.*          %regex -h 'client1' %classic /var/log/client1.messages
*.*          %regex -h 'client2' %classic /var/log/client2.messages
```

Alle Meldungen, die im Host-Anteil nun den Namen *client1* tragen, werden mit dem *%classic*-Modul in der Protokolldatei */var/log/client1.messages* abgespeichert.

Das Modul *%regex* erlaubt so eine sehr feinfühligke Verteilung der Meldungen auf unterschiedliche Protokolldateien und -server.

Zwei weitere Module, die die Auswertung der Protokollmeldungen vereinfachen können, sind *%pgsql* und *%mysql*. Hiermit ist es möglich, Syslog-Meldungen direkt in einer Datenbank zu protokollieren. Dies erlaubt später eine einfache Suche und Korrelation der Daten. Zusätzlich besteht die Möglichkeit, mit anderen Systemen recht einfach eine Auswertung der Daten (z.B. Apache/PHP-gestützt) vorzunehmen. Im Folgenden soll kurz die Konfiguration der Protokollierung in einer MySQL-Datenbank beschrieben werden.

Um die Protokollierung des *msyslogd* in einer Datenbank zu ermöglichen, ist es zunächst erforderlich, den MySQL-Datenbankserver und -client zu installieren. Dies erfolgt am einfachsten mit den Werkzeugen der Distribution. Nach der Installation und dem Start des MySQL-Datenbankservers müssen Sie das Kennwort des Datenbankadministrators *root* ändern. Dies erfolgt mit dem Befehl *mysqladmin*. Ein Beispielaufwurf ist im Folgenden abgedruckt:

```
# mysqladmin -u root password "n3u3sk3nnw0rt"
```

Nun sollte die Datenbank für die Protokollierung durch *mysyslogd* angelegt werden. Dies erfolgt mit den Befehlen:

```
# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12 to server version: 3.23.49

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE syslog;
Query OK, 1 row affected (0.01 sec)

mysql> use syslog;
Database changed
mysql> CREATE TABLE syslogTB (
  -> facility char(10),
  -> priority char(10),
  -> date date,
  -> time time,
  -> host varchar(128),
  -> message text,
  -> seq int unsigned auto_increment primary key
  -> );
Query OK, 0 rows affected (0.00 sec)
mysql> GRANT INSERT ON syslog.* TO msyslogd IDENTIFIED BY 'kennwort';
Query OK, 0 rows affected (0.00 sec)

mysql> quit;
Bye
```

Nun kann *mysyslogd* in diese Datenbank protokollieren. Hierzu wird die folgende Zeile in der Konfigurationsdatei benötigt:

```
*.*      %mysql -s host:port -u msyslogd -p kennwort -d syslog \
          -t syslogTB -F -P
```

Die beiden Module `%mysql` und `%pgsql` benötigen identische Datenbankschemata und unterstützen auch die gleichen Optionen. Diese Optionen haben hierbei die folgende Bedeutung:

- `-s host:port`. MySQL-Server. Der Port ist optional.
- `-u user`. MySQL-Benutzer. Dieser Benutzer muss das Recht haben, Einträge hinzuzufügen.
- `-p password`. Klartextkennwort. Achtung: Diese Datei sollte anschließend nicht für alle Benutzer lesbar sein!

- `-d database`. Name der Datenbank
- `-t table`. Name der Tabelle in der Datenbank
- `-D`. Verzögerte Einfügungen (**Nur MySQL**: Hierbei erhält der Client sofort eine Bestätigung. Der tatsächliche Insert erfolgt verzögert. Dies beschleunigt die Zusammenarbeit mit MySQL!)
- `-F`. Trage *facility* in einer eigenen Spalte ein.
- `-P`. Trage *priority* in einer eigenen Spalte ein.

Das interessanteste Modul ist jedoch sicherlich das Modul `%peo`. Dieses PEO-Modul verkettet die einzelnen Meldungen in einer Protokolldatei miteinander. Eine Modifikation der Protokolle wird daher später auffallen. Hierzu kann das Modul die Hash-Algorithmen MD5, SHA-1 (Default) oder RipeMD160 einsetzen.

Um diese Verkettung durchzuführen, verknüpft das Modul einen Schlüssel mit der ersten Meldung. Das Ergebnis wird wieder als Schlüssel für die nächste Meldung abgelegt. Bei der Überprüfung der Protokolldatei wird dieser Vorgang wiederholt. Dazu wird der initiale Schlüssel an einem sicheren Ort gespeichert. Die Konfiguration des Moduls `%peo` bietet drei Optionen:

- `-k key`. Schlüssel, der für die Verknüpfung der nächsten Meldung verwendet wird
- `-l`. Line Corruption Mode. Dieser Modus erlaubt es, die Zeile zu erkennen, auf der die erste Modifikation erfolgte. Hierzu wird eine zweite Datei angelegt. Diese Datei trägt denselben Namen wie die Schlüsseldatei mit der zusätzlichen Endung `.mac`.
- `-m hash`. Hash-Methode. Möglich sind `md5`, `sha1` und `rmd160`. Default ist `sha1`.

Eine Beispielkonfiguration des Moduls `%peo` könnte so aussehen:

```
*.* %peo -l -k /var/ssyslog/.var.log.messages.key %classic /var/log/
└─ messages
```

Damit diese Verkettung erfolgen kann, ist jedoch ein Initialisierungsvektor erforderlich. Die Erzeugung des initialen Schlüssels und auch die Überprüfung der Datei erfolgt mit dem Befehl `peochk`.

Um den initialen Schlüssel zu generieren, ist folgender Aufruf erforderlich:

```
# mkdir /var/ssyslog
# peochk -g -f /var/ssyslog/.var.log.messages.key -i messages.key0
```

Der Befehl `peochk` generiert nun einen Schlüssel und legt diesen in Binärformat in der Datei `/var/ssyslog/.var.log.messages.key` ab. Im ASCII-Format wird der Schlüssel in der Datei `messages.key0` erzeugt. Diese ASCII-Datei sollte kopiert oder ausgedruckt und vom Rechner entfernt werden.

Anschließend muss die Protokolldatei rotiert werden. Das Modul `%peo` funktioniert nur, wenn es mit einer leeren (aber existenten) Protokolldatei beginnt.

Um die Integrität der Datei zu testen, kann nun ebenfalls das `peochk`-Werkzeug verwendet werden. Hierzu muss zunächst die Datei `messages.key0` zur Verfügung stehen. Dies kann zum Beispiel mit einer Diskette erfolgen. Diese Datei sollte auf dem Rechner selbst aus Sicherheitsgründen gelöscht werden.

```
# peochk -f /var/log/messages -i messages.key0 -k /var/ssyslog/
↳ .var.log.messages.key
(0) /var/log/messages file is ok
```

Der Befehl `peochk` unterstützt einige weitere Optionen. Der Vollständigkeit halber werden hier noch einmal alle aufgeführt:

- `-f log`. Protokolldatei. Default ist die Standardeingabe.
- `-g`. Generiere initialen Schlüssel.
- `-i key0`. Initialer Schlüssel in ASCII
- `-k key`. Schlüssel in Binärformat
- `-l`. Prüfe, in welcher Zeile die Korruption auftritt. Erfordert die MAC-Datei, die von dem Modul `%peo` mit der Option `-l` angelegt wird.
- `-m`. Hash-Algorithmus (`md5`, `rnd160` oder `sha1` (Default))
- `-q`. Quiet. Ausgabe einer 0, wenn die Datei nicht verändert wurde. Ansonsten wird eine 1 und die Zeilennummer (`-l`) ausgegeben, auf der die Korruption auftritt.

## 16.3 Zeitsynchronisation

Die zentrale Protokollierung von Meldungen vereinfacht bereits stark die Korrelation von Ereignissen. So können auf dem zentralen Protokollserver die Meldungen unterschiedlichster Rechner in einer zentralen Datei abgelegt werden. Dies vereinfacht sehr stark ihre Zuordnung.

Häufig ist das jedoch aus Sicherheitsgründen nicht möglich. Es wird nicht gewünscht, dass Rechner, die sich in der demilitarisierten Zone (DMZ) einer Firewallstruktur befinden, Netzwerkzugriff auf Rechner hinter der Firewall erhalten, auch wenn es sich nur um zu übertragende Protokollmeldungen handelt. Dann werden gerne die Protokolle lokal abgelegt und in regelmäßigen Abständen von innen abgeholt oder lokal analysiert.

In anderen Fällen müssen die Protokolle mit anderen Informationsquellen, z.B. `tcpdump`-Netzwerkmitschnitten oder Webserver-Protokollen verglichen werden. Der Webserver protokolliert üblicherweise selbstständig und nicht über den System-`syslog`. Daher ist eine zentrale Protokollierung nur aufwändig zu gestalten.

In den genannten Fällen ist es nun für eine Protokollanalyse und einen Vergleich der Systeme unabdingbar, dass diese Systeme eine gemeinsame Zeitbasis nutzen. Hierfür wird unter UNIX und Linux das NTP-Protokoll eingesetzt. Jede Distribution verfügt

über ein Paket, welches in der Lage ist, dieses Protokoll als Client und Zeitserver anzubieten. Dabei handelt es sich um das *ntp*-Paket von <http://www.cis.udel.edu/~ntp>. Dieses Paket wird üblicherweise von den Distributionen als *ntp-\*.rpm* vertrieben.

Im Folgenden wird der Einsatz beschrieben.

### 16.3.1 Erste Synchronisation

Nach Installation des Paketes als RPM-Paket mithilfe des Quelltextpaketes von der oben angegebenen URL sollte der Rechner einmal manuell synchronisiert werden. Wenn die Rechneruhr und die Referenzzeit weit voneinander abweichen, kann es ansonsten zu ungewöhnlichen Nebeneffekten kommen. Die einmalige Synchronisation erfolgte in der Vergangenheit mit dem Befehl `ntpdate`. Dieser Befehl wird jedoch in einer der nächsten *ntp*-Distributionen verschwinden und wurde bereits ersetzt durch den Befehl `ntpd -q`. Mit der Option `-q` emuliert der `ntpd` den Befehl `ntpdate`.

Um die Synchronisation durchzuführen, wird eine Zeitquelle benötigt. Diese kann ganz unterschiedlicher Natur sein. Es existieren Treiber für DCF-77-Uhren, GPS-Karten usw. Die Konfiguration der einzelnen Hardware-Zeitquellen ist jedoch für eine Besprechung zu aufwändig. Daher wird im Weiteren davon ausgegangen, dass als Zeitquelle mehrere NTP-Server im Internet zur Verwendung kommen. Es existieren eine ganze Reihe von öffentlichen kostenlosen NTP-Servern. Eine Liste freier NTP-Server finden Sie unter <http://www.eecis.udel.edu/~mills/ntp/clock1.htm>.

Solange der `ntpdate`-Befehl noch zur Verfügung steht, kann folgender Befehl für eine erste Synchronisation eingegeben werden (Der Befehl `ntpd -q` benötigt eine Konfigurationsdatei, s.u.):

```
# ntpdate <server> # z.B. ntp0.uni-erlangen.de
22 Jul 13:47:10 ntpdate[5752]: adjust time server 131.188.3.220 offset
↳ 0.018978 sec
# ntpd -q # Die Konfigurationsdatei muss existieren
ntpd: time slew 0.012685s
```

### 16.3.2 Konfiguration und Einsatz

Der Aufruf von `ntpdate` führt jedoch nur eine erste Synchronisation durch. Anschließend werden die beiden Uhren (lokale Rechneruhr und Zeitquelle) wieder auseinander laufen. Um dies zu vermeiden, ist es sinnvoll, den Zeitserver zu installieren. Der Zeitserver (`ntpd`) ist in der Lage, sich selbst in regelmäßigen Abständen mit einer Zeitquelle zu synchronisieren und diese Information auch weiteren Clients zur Verfügung zu stellen.

Die Konfiguration des Zeitserverns ist sehr einfach. Er benötigt eine Konfigurationsdatei `/etc/ntp.conf`. Im Wesentlichen müssen in dieser Datei die Zeitquellen aufgeführt werden. Dies können lokale GPS-Empfänger, aber auch andere Zeitserver im Internet

sein. Die Zeitquellen werden auf einzelnen Zeilen mit dem Schlüsselwort `server` definiert:

*Listing 16.1 Konfigurationsdatei /etc/ntp.conf*

```
server ntp0.uni-erlangen.de # Universität Erlangen
server ntp0.fau.de         # Universität Erlangen
server ptbtime1.ptb.de    # Physikalisch-Technische Bundesanstalt
↳ Braunschweig
server chronos.cru.fr     # Universität in Rennes, Frankreich
driftfile /etc/ntp/drift
```

Nach dem Start des NTP-Servers `ntpd` kann die Synchronisation verfolgt werden:

```
# service ntpd start # SUSE rcntpd start
# ntpq -p
      remote           refid      st t when poll reach  delay  offset  jitter
=====
ntp0-rz.rrze.un 0.0.0.0      16 u   -   64   0   0.000   0.000 4000.00
ntp0-rz.rrze.un .GPS.        1 u   1   64   1  68.798  25.557   0.008
ntp1.ptb.de     .PTB.        1 u   3   64   1  55.806  28.373   0.008
chronos.cru.fr  .GPS.        1 u   4   64   3  56.858  30.467  11.900
```

Die Aktualisierung der Anzeige benötigt jedoch nach dem Start des NTP-Servers einige Sekunden bis zu einigen Minuten.

Dieser NTP-Server kann nun von anderen Clients im Netzwerk verwendet werden, um eine Zeitsynchronisation durchzuführen. Generell sollten pro Netzwerk zwei interne Zeitserver zur Verfügung stehen, die sich mit externen Quellen synchronisieren. Alle weiteren Rechner können dann diese beiden Zeitserver kontaktieren.

### 16.3.3 Sicherheit von ntpd

NTP ist ein integraler Dienst, wenn er verwendet wird. Eine Unterwanderung dieses Dienstes oder gar die Möglichkeit, falsche Zeiten an diesen Dienst melden zu können, würde die Zeitsynchronisation vereiteln und eine Korrelation der Ereignisse vollkommen unmöglich gestalten. Daher ist es wichtig, dass die Sicherheit dieses Dienstes gewährleistet ist. Dazu sind einige weitere Einträge in der Konfigurationsdatei nötig.

*Listing 16.2 Sichere Konfiguration von NTP*

```
# Standard: Vertraue niemanden
restrict default notrust nomodify

# Vertraue der Zeit der folgenden vier Server
restrict ntp0.uni-erlangen.de mask 255.255.255.255 nomodify notrap noquery
restrict ntp0.fau.de         mask 255.255.255.255 nomodify notrap noquery
restrict ptbtime1.ptb.de    mask 255.255.255.255 nomodify notrap noquery
```

```
restrict chronos.cru.fr      mask 255.255.255.255 nomodify notrap noquery

server ntp0.uni-erlangen.de # Universität Erlangen
server ntp0.fau.de          # Universität Erlangen
server ptbtime1.ptb.de     # Physikalisch-Technische Bundesanstalt
↳ Braunschweig
server chronos.cru.fr      # Universität in Rennes, Frankreich

# Erlaube interne Clients
restrict 192.168.115.0     mask 255.255.255.0 nomodify notrust notrap
```

NTP verwendet als Server den UDP-Port 123. Meist verwendet der Client auch diesen Port. Es existieren jedoch auch einige Clients, die einen unprivilegierten Port für die Verbindung mit dem Server verwenden. Das bedeutet, dass natürlich die zwei internen NTP-Zeitserver über diesen Port externe Systeme kontaktieren müssen. Eine Firewall muss diesen Zugang erlauben. Alternativ bietet sich auch der Kauf von DCF77- oder GPS-Hardware an, die vom System direkt ausgelesen und per NTP im Netzwerk zur Verfügung gestellt wird.

