4 Filesystem Integrity Checker

Falls Ihnen die Überschrift dieses Kapitels noch nicht allzu viel sagt, wird es definitiv Zeit, dass Sie sich mit dieser Materie auseinander setzen. Bevor ich Ihnen kurz erläutern werde, was *Filesystem Integrity Checker* überhaupt sind, möchte ich Sie bitten kurz über die Schwächen so genannter *Attack-Signature-Scanner* (Anti-Viren-Programme) nachzudenken. Dabei wird Ihnen mit Sicherheit auffallen, dass diese Programme –um einen guten Job machen zu können– stets eine aktuelle Bibliothek benötigen (man könnte auch sagen, dass Sie eine gute Virendatenbank bräuchten), in der Sie Informationen über bekannte Virensignaturen finden, um bereits bekannte, bösartige Fremdprogramme zu erkennen und somit das System vor solchen Eindringlingen schützen können.

In diesem Kapitel werden wir uns mit freier Software beschäftigen, die diesen Makel nicht hat und zusätzlich einen wesentlich größeren Schutz für Ihr System bieten kann. Filesystem Integrity Checker konservieren (stark vereinfach gesagt) einen Systemzustand, der als sicher gilt. Falls nun eine bösartige Software nicht vom Virenscanner erkannt wird und somit zum Beispiel durch eine E-Mail ins System gelangt, kann es mit Hilfe des konservierten Systemzustands gefunden und beseitigt werden.

4.1 Tripwire

Tripwire ist wohl der bekannteste Vertreter der Filesystem Integrity Checker und mit Sicherheit auch eines der besten freien Tools, die sich mit IT-Security einen Namen gemacht haben. Die Entwickler Gene Kim und Eugene Spafford haben es sich zur Aufgabe gemacht, ein Programm zu schaffen, das dabei weitaus mehr bietet als einen einfachen Vergleich des aktuellen Systemzustands mit einer als sicher geltenden Sicherungskopie. Da den Entwicklern bereits sehr bald klar wurde, dass man die Integrität eines Dateisystems anhand einiger, signifikanter Merkmale genauso gut überprüfen kann wie anhand einer exakten Sicherungskopie, bestand die große Aufgabe und Schwierigkeit darin, diese Merkmale auf eine sinnvolle Weise – auch und besonders in Hinblick auf die späteren Überprüfungen – zu extrahieren. Um die Datenmengen, die dabei entstehen, so gering wie möglich zu halten, hat man sich dazu entschlossen, die Informationen mit Algorithmen aus dem Bereich der Nachrichtenverschlüsselung aus dem System zu gewinnen (zum Beispiel MD5).

Bevor wir beginnen, möchte ich an dieser Stelle vorab erwähnen, dass wir mit der weit verbreiteten *Academic-Source-Release* (ASR) arbeiten, aber an den meisten Stellen (Selection Masks und Templates) auch auf die Vorzüge der »anderen« Version eingehen werden. Sie können bereits bei der Installation festlegen, welche Version Sie beziehen. Dies hängt dabei aber zum Großteil davon ab, wie Sie Tripwire nutzen (lizenzpolitisch).

4.1.1 Allgemeine Vorteile

Das Kuriose an Tripwire ist, dass es eigentlich über sein Ziel hinausschießt (im absolut positiven Sinne). Wie wir bereits erfahren haben, war das Ziel der Entwickler ein Programm zu schaffen, das es einem ermöglicht, den aktuellen Sicherheitsstand unseres Systems daran zu erkennen, ob bestimmte (von uns angegebene) Charakteristika verändert wurden oder sich noch im gewünschten Zustand befinden. Dieses System bringt aber noch einige andere Vorteile mit sich. So erleichtert Tripwire das Entfernen schlecht geschriebener Software enorm. Ein weiterer großer Vorteil wird dem Programm im Bereich der aktiven Beweissicherung nachgesagt. Ohne mich zu weit aus dem Fenster lehnen zu wollen (ich muss zugeben, ich hatte nur einige, wenige Vorlesungen während meines Studiums, die sich um Computerrecht oder Ähnliches gedreht haben), könnte die Signatur einer One-Way-Hash-Funktion durchaus als Beweismittel vor Gericht dienen. Da aber zumeist der Angreifer nicht in der Lage ist, den finanziellen Schaden, den er durch seinen Angriff angerichtet hat, zu begleichen, erscheint es wohl sinnvoller, diese Datei der Versicherung zu mailen.

4.1.2 Konfigurationsaufwand

Nachdem wir nun die geplanten und nicht geplanten Features von Tripwire kennen, ist es wohl an der Zeit, den Verwaltungs- und Konfigurationsaufwand dieser Software zu besprechen. Da Tripwire ein statisches System ist, müssen Sie sich jedes Mal mit der Konfiguration beschäftigen, sobald Sie Software hinzufügen oder entfernen. Das heißt, dass auf Testrechnern und ähnlichen Systemen Tripwire nichts zu suchen hat. Da aber die Softwarekonfiguration auf den meisten kritischen Serversystemen nicht wöchentlich geändert wird, sollten Sie sich um den Konfigurationsaufwand keine allzu großen Gedanken machen, da dieser im Vergleich zum Nutzen ein gerne in Kauf genommenes kleines Übel darstellt.

4.1.3 Installation

Wie immer sollten Sie sich die aktuelle Version von Tripwire aus einer sicheren Quelle wie der Buchseite oder bevorzugt von den offiziellen Tripwireseiten, welche Sie in der folgenden Auflistung finden, herunterladen:

- *http://www.cs.purdue.edu/coast*
- *http://www.tripwiresecurity.com*

Nachdem Sie die Echtheit Ihres Pakets überprüft haben, können Sie es ohne Bedenken installieren.

```
Linux:~# rpm –ivh tripwire-version-i.386.rpm
```

Wie ich zu Beginn bereits erwähnt habe, werde ich Ihnen Tripwire anhand der ACR vorstellen. Falls Sie sich schon länger mit Tripwire beschäftigen, werden Sie sicherlich wissen, dass sich in letzter Zeit einiges in Sachen Vermarktung, Open Source etc. getan hat. Wer mehr über die Geschichte von Tripwire wissen möchte, kann dazu mehr unter folgender Adresse finden:

http://www.forbes.com/tool/html/toolbox.htm

Wer daran interessiert ist, an der Entwicklung von Tripwire teilzuhaben, kann sich bei Sourceforge Informationen dazu besorgen:

http://sourceforge.net/projects/tripwire

Wer die aktuellen Binaries beziehen will, kann dies natürlich auch per Internet tun:

http://www.tripwire.org

Jetzt sollten Sie alle wichtigen Adressen und Informationen erhalten haben, um sich die richtige Version zu beschaffen, und wir können guten Gewissens starten.

4.1.4 Konfigurationsdatei

Die Konfigurationsdatei *tw.config* bildet das Herzstück von Tripwire. An sich ist die Syntax leicht zu verstehen und zu erlernen. Falls Sie aber vorhaben *tw.config* für mehrere Hosts, die unterschiedliche Betriebssysteme fahren, einzusetzen, kann die Syntax schnell sehr komplex werden. Aber keine Angst, wir werden mit einer einfachen Konfiguration einsteigen, so dass Sie sich ganz langsam an die Syntax und ihre Tücken gewöhnen können.

Tripwire bietet eine Vielzahl an Optionen, die es einem ermöglichen, Daten oder ganze Verzeichnisse (auch rekursiv) zu überwachen. Um einen einfachen Weg zu bieten, auch plattformabhängige Charakteristika eines Dateisystems in die Überwachung nahtlos einschließen zu können, hat man sich dazu entschlossen, die Konfigurationsdatei ähnlich einer Liste aufzubauen. Die Grundsyntax hierbei sieht folgendermaßen aus:

```
[!|=] Objekt [Auswahlmaske] [Kommentar]
```

Kommentare

Vorab möchte ich gleich auf die Kommentare eingehen. Wie bei vielen anderen Skriptsprachen und wie bei den meisten Konfigurationsdateien leiten Sie einen Kommentar mit dem #-Zeichen ein, wobei auch hier die Zeichen und Anweisungen hinter dem Kommentarzeichen ignoriert werden.

Objekte und Operatoren

Objekte in Tripwire stellen entweder Dateien oder ganze Verzeichnisse dar. Wenn Sie sich die Syntax von *tw.config* nochmals ansehen, werden Sie bemerken,

dass Sie optional die Operatoren ! und = einsetzen können. Ich möchte Ihnen zuerst den =-Operator anhand eines Beispieles vorstellen:

```
# Beispielfragment aus tw.config
.
= /mail        L
.
```

Das hier aufgeführte Codefragment weist Tripwire an, dass es zwar die Inode des Verzeichnisses */mail* überwachen soll, aber nicht den Inhalt dieses Verzeichnisses. Das heißt, Sie werden zum Beispiel über die Erstellung oder das Löschen von Dateien informiert, aber nicht über die betroffenen Objekte oder die Art der Veränderungen. Dies ist eine äußerst ressourcensparende Option, da sich der Inhalt der Verzeichnisse wie */mail*, */tmp* und */var/spool* im Regelfall sehr oft ändert.

Falls Sie keinen Operator verwenden, werden Sie zusätzlich über die Art der Veränderungen und die betroffenen Objekte informiert.

Das nächste Codefragment zeigt die Verwendung des !-Operators, der es Ihnen erlaubt, bestimmte Objekte aus der Überwachung auszuschließen. Diese Option ist für Verzeichnisse wie */dev* äußerst nützlich. Sie sollten aber grundsätzlich darauf verzichten, da jede Verwendung des Stopp–Operators möglicherweise auf Kosten der Sicherheit geht.

```
# Beispielfragment aus tw.config
.
  !/dev
.
.
```

Bevor wir den Abschnitt über Objekte abschließen, möchte ich Sie noch darauf hinweisen, dass Sie unbedingt die Dateisystemgrenzen beachten müssen. Sind zum Beispiel */joey* und */joey/aga* Mountpunkte für zwei Partitionen und Sie wollen das Verzeichnis */joey* komplett überwachen, so müssen beide Pfade separat aufgeführt werden.

Selection masks / select-flags

Wir haben gesehen, dass Sie mit den verfügbaren Operatoren die Möglichkeit haben, die Überwachung in groben Zügen einzuschränken. Tripwire bietet aber eine weitaus tiefer gehende und feinere Methode, um Abstimmungen über die vorzunehmende Überwachung zu treffen. Mit den so genannten *Selection masks* oder *Select-flags* (hier scheiden sich die Geister, wenn es um die korrekte Betitelung dieser Option geht) haben Sie die Möglichkeit eine konkrete Eigenschaft des zugeordneten Objekts zu kennzeichnen. Welche Eigenschaft Sie dabei wählen können, hängt zum Großteil vom Dateisystem (EXT2 hat sich als Quasistandard etabliert) ab. Welche Selection Masks Ihnen grundsätzlich in Linux unter EXT2

zur Verfügung stehen, sehen Sie in Tabelle 4.1. Über die sinnvolle Implementierung anhand einiger Codefragmente und eventuelle Alternativen werden wir uns im Anschluss Gedanken machen.

Selection Mask	Bericht	Beschreibung
P	st_mode	Ausführungsmodi (SUID-Bit, SGID-Bit und Text-Bit) und Zugriffsrechte
I	st_ino	Inodenummer. Allgemeines zu Inodenummern finden Sie in Kapitel 1. Normalerweise wird durch Schreiben und Lesen einer Datei die Inodenummer nicht verändert. Findet Tripwire dennoch eine Veränderung, so kann man daraus schließen, dass das betroffene Objekt gelöscht oder durch eine Datei mit anderem Inhalt, aber selbem Namen ersetzt wurde.
N	st_nlink	Link-Count. Repräsentiert die Anzahl der Links (keine dynamischen, sondern ausschließlich Hardlinks) und Unterverzeichnisse. Im Falle von Verzeichnissen gibt der Zähler (Counter) die Anzahl der zugehörigen Unterverzeichnisse an. Im Falle einer einfachen Datei die Anzahl der mit der Inode verbundenen Links. Der Zähler erhöht sich bei jedem Hardlink auf die zugehörige Datenzone.
U	st_uid	UID. Benutzer ID
G	st_gid	GID. Gruppen ID
S	st_size	File size. Eine sehr wichtige Mask, da Veränderungen wichtiger Dateien meistens mit einer Änderung der Dateigröße einhergehen.
A	st_atime	Access timestamp. Auf der Jagd nach Crackern erweist sich diese Mask als äußerst hilfreich, da ihr Wert bereits beim Betrachten bzw. beim Einlesen einer Datei geändert wird. Sie sollten diese Datei mit keiner Mask verwenden, die zur Berechnung des Werts die Datei einlesen muss (zum Beispiel Signaturberechnung).
M	st_mtime	Modification-timestamp. Dieses Feld wird geändert, falls eine Datei verändert und mit den Veränderungen gesichert wird.
C	st_ctime	Inode-change-timestamp. Diese Mask gibt den Wert der letzen Statusänderung zurück. Dies entspricht dem letzten Schreibzugriff auf die Inode (zum Beispiel, wenn die Zugriffsrechte einer Datei geändert wurden).
T	Object Type	Dateityp

Tabelle 4.1 Verfügbare Selection-Masks

Selection Mask	Bericht	Beschreibung
D	Device Nr.	Beim Partitionieren sind mit einer speziellen Kennzahl versehen, welche Aufschluss über die Art der Formatierung gibt. Diese Mask stellt sicher, dass diese Kennzahl (neben anderen Merkmalen) in die Referenzdatenbank eingetragen wird, von der die Inode des dazugehörigen Objekts stammt.
I	Size	Vorab ist zu dieser Mask zu sagen, dass sie unter ASR nur in Verbindung mit einer anderen Selection-Mask als Template (werden wir später noch genauer besprechen) zur Verfügung steht. Anders als die Mask s, die bei jeder Veränderung der Dateigröße »anspringt«, erfolgt bei dieser Mask nur eine Meldung, falls eine Datei verkleinert wurde. Diese Mask sollte man auf Logfiles ansetzen, da diese von Eindringlingen gerne verändert werden.
R	File Device Nr.	Diese Mask ist nur für Gerätedateien verfügbar und bezeichnet die Gerätetreibernummer, die mit der dazugehörigen Inode verbunden ist.
B	Blocks	Steht für die Anzahl der Datenblöcke (für EXT2 typischerweise 1024 Bit), die durch die Zonenzeiger der Inode belegt werden.

Tabelle 4.1 Verfügbare Selection-Masks (Forts.)

Nachdem Sie nun einige der Selection-Mask kennen gelernt haben, wollen wir uns mit den verfügbaren Algorithmen beschäftigen. Die Wahl des Algorithmus ist von großer Bedeutung und von vielen Faktoren abhängig. Denn was hilft Ihnen ein absolut sicheres System (was es ja sowieso nicht geben kann, außer Sie schalten den Rechner aus), wenn die Performance so gering ist, dass eine Bedienung nahezu unmöglich ist. In Tabelle 4.2 finden Sie die in Tripwire verfügbaren Algorithmen mit einigen durchaus hilfreichen Bemerkungen. Das Feld »Geschwindigkeit« kann die Werte 1 bis 8 einnehmen, wobei 1 der schnellste Algorithmus ist und 8 der langsamste. Das Feld »Sicherheit« kann zwischen 1 und 5 variieren, wobei Algorithmen, die den Wert 5 zugewiesen bekamen, die sichersten darstellen.

Beachten Sie dabei, dass auch die Auswahl des Algorithmus mit Hilfe der Selection-Masks zu treffen ist.

Mask	Algorithmus	Geschwindigkeit	Sicherheit	Beschreibung
0	-	-	-	null signature
1	MD5	5	5	*Message Digest 5 Algorithmus*, sicherer, aber auch langsamer als sein Vorgänger

Tabelle 4.2 Algorithmen

Mask	Algorithmus	Geschwindigkeit	Sicherheit	Beschreibung
2	Snefru	7	4	Langsamer, aber als sicher geltender Algorithmus
3	CRC-32	4	2	Ursprünglich gedacht zur Erkennung von hardwarebedingten Übertragungsfehlern. Sollte bei sicherheitskritischen Systemen nicht verwendet werden.
4	CRC-16	1	1	Siehe Beschreibung zu CRC-32. Absolut schneller Algorithmus, der in Sachen Geschwindigkeit seinesgleichen sucht! Sollte aber bei sicherheitskritischen Serversystemen nicht eingesetzt werden.
5	MD4	2	3	Sehr beliebt bei RISC Systemen, da er auch sehr schnell ist. Sollte aber auf keinen Fall eingesetzt werden.
6	MD2	8	4	Ist noch auf 8 Bit Prozessortechnik ausgelegt und von daher ungemein langsam. Obwohl er als sicher gilt, sollte er aufgrund der mangelnden Geschwindigkeit nicht eingesetzt werden.
7	SHA	6	5	Ist mit MD4 verwandt. Obwohl – wie Sie vielleicht wissen – einige Gerüchte im Raum stehen, dass dieser Algorithmus eine nicht dokumentierte Sicherheitslücke besitzt, können Sie ihn meiner Ansicht nach bedenkenlos einsetzen (die NASA tut es ja auch).
8	Haval	3	4	Bietet 15 unterschiedliche Varianten für praktische Anwendungen, ist sicher und schnell. Kann durchaus für sicherheitskritische Systeme eingesetzt werden.

Tabelle 4.2 Algorithmen (Forts.)

Verwendung von Select-Masks

Der einfachste Weg, um eine vollständige Beschreibung aller interessanten Eigenschaften eines Objekts zu bekommen, besteht darin, alle dafür nötigen Masks aneinander zu reihen. Dabei haben Sie die Möglichkeit mit zwei weiteren Operatoren (+ und -) bestimmte Masks explizit aus- bzw. einzuschließen. Dabei müssen Sie beachten, dass die von uns verwendete Version ASR alle nicht ausdrücklich ausgeschlossenen Masks (Sie können bestimmte Masks mit Hilfe des --Operators ausschließen) in die Überwachung einschließt. Falls Sie sich an dieser Stelle fragen, wofür dann der +-Operator zu gebrauchen sein sollte, haben Sie bitte noch etwas Geduld. Sie werden es im Abschnitt über Templates erfahren.

Da das eben erwähnte »Feature« in den Manpages nur schlecht bis überhaupt nicht beschrieben ist, werden wir uns anhand einiger Beispiele noch tiefer in die Materie einarbeiten. Im folgenden Listing finden Sie einige Auswahlmasken, die trotz völlig unterschiedlicher Syntax genau dasselbe bewirken.

```
/joey/aj +ug-n
/joey/aj -n
/joey/aj +piuagsmc123456789-n
```

Wie Sie aus dem Listing entnehmen können, ist aus Ihrem Wunsch, nur die UID und GID ohne die Anzahl der Hardlinks zu überwachen, nichts geworden, da Tripwire automatisch alle nicht explizit ausgeschlossenen Masks einbezieht. Um zu erreichen, dass wirklich nur die UID und die GID angezeigt werden, müssen Sie folgende Auswahlmaske benutzen:

```
/joey/aj +ug-pinsamc123456789
```

oder

```
/joey/aj -pinsamc123456789
```

Templates

Im vorhergehenden Abschnitt haben Sie gesehen, wie man eine Auswahlmaske definiert und welche Eigenheiten Sie dabei beachten müssen. Wie Sie sich gut vorstellen können, kann es durchaus vorkommen, dass Sie bei solch langen Masken Fehler machen. Um diese Fehlerquelle teilweise zu beseitigen, haben Sie die Möglichkeit so genannte Templates zu verwenden. Dies sind vordefinierte Auswahlmasken, die es Ihnen ermöglichen, oft genutzte Auswahlschemata mit nur einem Buchstaben zu initialisieren. In Tabelle 4.3 sind die verfügbaren Templates zu sehen.

Template	Äquivalent	Beschreibung
R	+pinugsm12-ac3456789	Read-Only. Dateien sind allgemein verfügbar, dürfen aber nur gelesen werden. (Defaultwert)
L	+pinug-sacm123456789	Log file. Verzeichnisse und Dateien, die sich ständig verändern
N	+pinugsamc123456789	Ignore-Nothing. Nichts ignorieren. Alle Masken sind aktiv.
E	-pinugsamc123456789	Ignore-Everything. Alles ignorieren. Hier werden nur hinzugefügte oder gelöschte Objekte ausgewiesen.
>	+pinug-samc123456789	Growing-File. Gut geeignet für Logdateien, die zwar wachsen dürfen, aber deren Größe nicht abnehmen darf.
Device	+pugsdr-intlbamcCMSH	Dateien, die Tripwire nicht öffnen darf (Gerätedateien)

Tabelle 4.3 Verfügbare Templates

Eine Konfigurationsdatei für das gesamte Netzwerk

Nachdem Sie nun wissen, welche Möglichkeiten beim Designen der *tw.config* zur Verfügung stehen, sollten Sie sich Gedanken machen wie Sie Ihr Netzwerk schützen. Denn – wie wir alle wissen – der Computer ist nichts, da das Netzwerk alles ist. Und da ein solches – von IBM gepriesenes – Netzwerk aus vielen Computern besteht, ist es an der Zeit, über die Möglichkeiten zu sprechen, die Sie haben, um Tripwire zentral zu konfigurieren, das heißt, wie Sie die Konfigurationsdatei verfassen, so dass Sie damit alle Rechner in Ihrem Netzwerk überprüfen und verwalten können. Natürlich können Sie auch auf jedem Rechner eine eigene *tw.config* anlegen, doch dies bedeutet einen enormen Aufwand bei der Verwaltung.

Tripwire bietet uns dafür die Preprocessor-Syntax, deren Befehle mit einem doppelten `@` (`@@`) initialisiert werden. Tabelle 4.4 beinhaltet die verfügbaren Optionen hierfür.

Argument	Beschreibung
@@ifhost hostname	Wahr, wenn hostname mit uname –n oder hostname übereinstimmt. Wenn der Wert wahr zurückgegeben wird, werden alle Anweisungen bis zum nächsten `@@endif` oder `@@else` ausgeführt.
@@ifnhost hostname	Wahr, wenn hostname mit `uname -n` oder hostname nicht übereinstimmt. Wenn der Wert wahr zurückgegeben wird, werden alle Anweisungen bis zum nächsten `@@endif` oder `@@else` ausgeführt.
@@else	Wird ausgeführt, falls `@@ifhost`, `@@ifnhost`, `@@ifdef` oder `@@ifndef` den Wert falsch zurückgeben
@@ifdef Variable	Gibt den Wert `true` zurück, wenn die Variable definiert ist
@@ifndef Variable	Gibt den Wert `true` zurück, wenn die Variable nicht definiert ist
@@endif	Schließt das `@@if*` Statement
@@define Variable String	Weist der Variable einen String zu, falls kein String angegeben wird, so wird der Null-String zugewiesen
@@undef Variable	Variable ist nicht weiter definiert
@@include Pfadname	Zieht die Datei, die unter dem angegebenen Pfadnamen liegt, mit ein. Diese Datei muss dieselbe Konfigurationssyntax haben.
@@Variable	Weist den String, der in Variable gespeichert ist zu.
@@{Variable}	Selbe Beschreibung wie das zuvor aufgeführte Argument
||	Logisches ODER
&&	Logisches UND

Tabelle 4.4 Preprozessoren unter Tripwire

Um Ihnen die Syntax näher zu bringen, habe ich im Folgenden einige Beispiele für Sie vorbereitet. Ich werde die einzelnen Schritte direkt im Code anhand von Kommentaren beschreiben.

```
# Tripwire Konfigurationsdatei mit Preprocessoren
#
# Die erste Zeile aendert das Template READ (R) so ab, dass
anstelle
# der Selection-Masks # 7 und 8 die Masks 1 und 2 verwendet
werden.
@@define READ R+78-12
#
# Da wir ein Netzwerk mit vielen unterschiedlichen
Betriebssystemen
# haben, brauchen wir
# fuer jedes OS eigene Auswahlmasken. Wir werden Tripwire nun
# "sagen" welche Hosts welches Betriebssystem fahren.
# Als Erstes weisen wir den Rechnern joey, roland, grassl das OS
# Linux zu.
#
@@ifhost joey || roland || grassl
@@define LINUX
@@endif
#
# Nun weisen wir den Rechnern aga, aj, bill das OS MS Windows zu.
#
@@ifhost aga || aj || bill
@@define WINDOWS
@@endif
#
# Natürlich haben wir auch einige Solarisserver im Haus
#
@@ifhost angelus || woodstock
@@define SOLARIS
@@endif
#
# Die restlichen Computer in unserem Netzwerk fahren IRIX
#
@@ifndef SOLARIS
@@ifndef LINUX
@@ifndef WINDOWS
@@define IRIX
@@endif
@@endif
@@endif
@@endif
#
# Jetzt kommen wir zum eigentlichen Start von Tripwire
#
# Fuer Linux haben wir eine eigene Konfigurationsdatei unter
# /usr/local/tripwire/etc/tw.config.linux angelegt.
#
@@ifdef LINUX
```

```
@@include /usr/local/tripwire/etc/tw.config.linux
@@else
.
#
# Hier lasse ich einen Großteil der Syntax weg, da Sie die
# Argumente darin bereits kennen.
#
.
@@endif
.
@@ifdef SOLARIS
.
#
# Jetzt definieren wir noch unsere eigenen Variablen und weisen
# ihnen Selection Masks zu
#
@@define privat E
@@define geheim R
/home/joey @@private
/root/online-banking @@geheim
.
.
@@endif
.
.
```

Ich hoffe, dieses Beispiel hat Ihnen weiter geholfen und Sie sehen sich nun in der Lage eine eigene zentrale Konfigurationsdatei zu entwerfen.

4.1.5 Funktionsweise

Bevor wir uns näher mit den einzelnen Befehlen von Tripwire auseinander setzen können, sollten wir uns zuerst mit der Funktionsweise dieses Tools beschäftigen. Man kann diese Prozedur in drei Phasen unterteilen:

1. Initialisierung
2. Integritätstest
3. Updatemodus
4. Wir werden nun kurz auf die einzelnen Phasen eingehen und deren Nutzen und Funktionen erläutern.

Initialisierung

Bei der Initialisierung wird auf Basis der Konfigurationsdatei (*tw.config*) eine Referenzdatenbank erstellt, welche – wie wir ja bereits besprochen haben – eine Beschreibung der sicherheitsrelevanten Objekte des betroffenen Systems bzw.

des betroffenen Netzwerks enthält. Diese Beschreibung ergibt sich aus der Auswahlmaskendefinition, die Sie mit Hilfe der Selection-Masks erstellen.

Die dabei erstellte Referenzdatenbank (welche im ASCII Format erstellt wird) müssen Sie nach der Initialisierung auf jeden Fall noch in ein anderes Verzeichnis verschieben. Genaueres dazu werden Sie bei der Besprechung der einzelnen Argumente für den Tripwire-Befehl finden.

Beachten Sie bitte, dass Ihr System bei der Initialisierung frei von jeglichen bösartigen Fremdkörpern, die nichts in Ihrem System zu suchen haben, ist. Am besten geeignet dafür ist selbstverständlich ein frisch aufgesetztes System; da dies in der Realität natürlich kaum zu bewerkstelligen ist, sollten Sie durch manuelle Überprüfung einiger systemkritischer Dateien (wie *inetd.conf* bzw. *xinetd.conf* oder *passwd*) sicherstellen, dass sich das System in einem sicheren Zustand befindet.

Die Syntax für die Initialisierung werden Sie nach der Besprechung der Funktionsweise erhalten.

Integritätstest

Diese Phase stellt die eigentliche Systemüberprüfung dar. Auch hierbei wird eine Datenbank auf Basis der Konfigurationsdatei erstellt, welche den aktuellen Stand des Dateisystems widerspiegelt. Der erste Vergleich mit der Referenzdatenbank liefert dabei folgende Informationen:

- Anzahl der gescannten Dateien
- Anzahl der hinzugefügten Dateien
- Anzahl der gelöschten Dateien

Der folgende genaue Vergleich wendet dann die von Ihnen definierten Auswahlmasken an und gibt Ihnen die Anzahl der noch übrig bleibenden Dateien zurück. Das heißt: Anfangs werden alle Änderungen im System angezeigt, aber nach Anwendung der Regeln zeigt Ihnen Tripwire, ob bei diesen Änderungen Dateien oder Attribute betroffen sind, die Sie durch Ihre Auswahlmasken spezifiziert haben.

Die Bildschirmausgabe eines Integritätstests, der keine gefährlichen Änderungen an den Dateien, die in *tw.config* spezifiziert sind, ausweist, sieht folgendermaßen aus:

```
Linux:~# tripwire
### Phase 1:    Reading configuration file
### Phase 2:     Generating file list
### Phase 3:     Creating file information database
### Phase 4:    Searching for inconsistencies
###
###        Total files scanned:    231
###            Files added:        0
###            Files deleted:    0
```

```
###            Files changed:    231
###
###        After applying rules:
###            Changes discarded:    231
###            Changes remaining:    0
```

Dem aufgeführten Listing können Sie entnehmen, dass 231 Dateien geändert wurden, aber dass keine dieser Änderungen auf eine Auswahlmaske zutrifft.

Updatemodus

Falls Sie eines Tages bemerken, dass der Integritätstest von Tripwire eine Veränderung anspricht, die von Ihnen gewollt wurde, ist es an der Zeit, die Referenzdatenbank auf den neuesten Stand zu bringen, da Tripwire diesen »Missstand« ansonsten bei jedem Durchgang erneut bemängeln würde.

Diese Option erlaubt es Ihnen aber auch, Ihre Konfigurationsdatei zu verändern, ohne dabei eine komplette Reinitialisierung veranlassen zu müssen. Sie werden zu diesem Befehl im folgenden Abschnitt mehr erfahren.

4.1.6 Die Tripwire-Befehle

Dieser Abschnitt beschäftigt sich mit den verfügbaren Tripwire-Befehlen, die Ihnen beim Umgang und bei der Pflege Ihres Systems zur Seite stehen. Bevor wir die einzelnen Optionen dazu besprechen, möchte ich Sie darauf hinweisen, dass das eigentliche Tripwire-Programm (also der Integritätstest) aufgerufen wird, falls Sie der Konsole nur den Programmnamen übergeben.

```
Linux:~# tripwire
```

Sie haben ja bereits bei der Besprechung der Funktionsweise gesehen, wie die Bildschirmausgabe aussehen sollte, falls »nennenswerte« Änderungen auftraten (sprich, wenn Ihre Auswahlmasken nicht berührt wurden). Nun werden wir uns damit beschäftigen, wie Tripwire die Ausgabe gestaltet, falls eine Auswahlmaske betroffen wurde.

Wir haben dazu der Datei */joey/neu/test* eine Zeile hinzugefügt, und da Tripwire in der Konfigurationsdatei die Maske erhielt, dass diese Datei nicht verändert, sondern nur gelesen werden darf, wird der Integritätstest eine Inkonsistenz bemerken und diese auch anzeigen:

```
Linux:~# tripwire
### Phase 1:    Reading configuration file
### Phase 2:     Generating file list
### Phase 3:     Creating file information database
### Phase 4:    Searching for inconsistencies
###
```

```
###        Total files scanned:    232
###            Files added:        0
###            Files deleted:    0
###            Files changed:    232
###
###        After applying rules:
###            Changes discarded:    231
###            Changes remaining:    1
###
changed: -rw-r—r—joey    18 Jun 25 19:50:40 2001 /joey/neu/test
### Phase 5:    Generating observed/expected pairs for changed
files
###
### Attr    Observed (what it is)    Expected (what it should be)
### ======== ========================
============================
/joey/neu/test
st_ino:    137046                136996
st_size:    18                6
st_mtime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
st_ctime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
md5 (sig1):    1oDgYoDDEL.5g2tFr9W.Nm    2kwNorphIYEyAtMnhMr:kIa
snefru (sig2):2.ZirbrL3ZN289TyymBJ5H    27shvz29Wzo.DeyYN38no9
```

Wie Sie aus dem Listing gut entnehmen können, hat Tripwire die Änderung an unserer Datei bemerkt und uns sofort darüber informiert.

Neben dem Integritätsaufruf bietet Tripwire aber noch einige zusätzliche Optionen, welche wir nun besprechen werden:

Initialisierung

Sie kennen diese Option bereits aus der Besprechung der Funktionsweise. Wir werden sie aber an dieser Stelle trotzdem nochmals ausführlich besprechen.

Diese Option werden Sie nicht sehr häufig benutzen, da Sie im späteren Systemlauf durch die Updatefunktion zum Großteil ersetzt werden kann. Sie kommen aber nicht drum herum, die Funktion zum Start Ihres Tripwiresystems zu benutzen, da Sie damit die Referenzdatenbank erstellen. Sie rufen diese Funktion wie folgt auf:

```
Linux:~# tripwire –initialize
```

Nachdem Sie den Befehl eingegeben haben, erhalten Sie folgende Bildschirmausgabe:

```
### Warning:    creating ./databases directory!
###
### Phase 1:    Reading configuration file
```

```
### Phase 2:    Generating file list
### Phase 3:    Creating file information database
###
### Warning:    Database file placed in ./databases/
tw.db_rechnername
###
###          Make sure to move this file and the configuration
###         to secure media!
###
###          (Tripwire expects to find it in '/etc/tw'.)
```

Wie Sie der Bildschirmausgabe entnehmen können, erstellt Tripwire ein Verzeichnis namens *databases* im Verzeichnis, in dem Sie sich beim Aufruf des Befehls befunden haben. Dort speichert es die Datenbank unter dem Namen *tw.rechnername* (wobei *rechnername* durch den Rechnernamen Ihres Systems zu ersetzen ist). Danach werden Sie aufgefordert die Datenbank und die Konfiguration auf ein sicheres Medium zu speichern. Sie erhalten zusätzlich den Hinweis, dass Tripwire die Datenbank in dem Verzeichnis */etc/tw* vermutet. Sie sollten – falls Sie sich dazu entscheiden, die Datenbank in diesem Verzeichnis aufzubewahren – das `Read-Only`-Flag für dieses Verzeichnis setzen. Eine weitaus sicherere Methode wäre es aber, die Datenbank auf eine CD zu brennen, da bei dieser Methode definitiv keine Änderungen daran vorgenommen werden können (Falls ein Angreifer Rootrechte erlangt, könnte er das `Read-Only`-Flag für das Verzeichnis problemlos wieder entfernen.) Vergessen Sie bei dieser Methode aber nicht die CD auf */etc/tw* zu mounten.

Wir werden uns im weiteren Verlauf dieses Kapitels noch intensiver mit der Sicherung der Datenbank beschäftigen. Fürs Erste denke ich aber, dass Sie alles Nötige wissen.

Updatemodus

Auch diese Option kennen Sie bereits aus der Besprechung der Funktionsweise von Tripwire. Um Ihnen die Vorteile und Möglichkeiten diese Option näher zu bringen, werde ich ein Beispiel aus dem alltäglichen Administratorleben benutzen (dieses Beispiel ist zwar etwas stupide, aber es erfüllt seinen Zweck):

Nehmen wir an, Sie bemerken (beziehungsweise der Tripwire-Integritätstest bemerkt dies bei einem seiner Scandurchläufe), dass ein von Ihrem Unternehmen entwickelter Server namens *fishd* als `changed` dargestellt wird (Ursache dafür könnte sein, dass Sie Tripwire gesagt haben, dass die Größe dieser Datei stets gleich bleiben muss). Nachdem Sie sich mit Entwicklern in Ihrem Hause unterhalten haben, kommen Sie zum Ergebnis, dass dieser Vorfall völlig legitim war (ihre Entwickler haben etwas Code hinzugefügt und deshalb wurde die Datei größer). Natürlich sollte man das auch Tripwire mitteilen, da es ansonsten bei jedem Scandurchlauf diesen Missstand anzeigen wird. Ein Weg, um dieses Problem zu lösen, ist eine komplett neue Referenzdatenbank zu erstellen. Das ist aber eher der

»grobe« Weg. Tripwire bietet einen viel eleganteren Modus, um bestimmte Änderungen wirksam zu machen.

Mit dem Updatemodus können Sie schnell und einfach die Änderung des Servers in die Referenzdatenbank aufnehmen, so dass auch Tripwire über die neue Größe der Datei Bescheid weiß.

Da der Server unter */usr/sbin* liegt und die Datei *fishd* heißt, ergibt sich folgender Befehl:

```
Linux:~# tripwire –update /usr/sbin /usr/sbin/fishd
```

Nachdem Sie den Befehl eingegeben haben, werden Sie folgende Bildschirmausgabe erhalten, welche Sie über die Aktualisierung informiert:

```
### Phase 1:    Reading configuration file
### Phase 2:    Generating file list
Updating: update entry: /usr/sbin
Updating: update file: /usr/sbin/fishd
### Phase 3:    Updating file information database
###
### Old database file will be moved to 'tw.rechnername.old'
###        in ./databases
###
### Update database will be stored in './databases/tw.rechnername'
###        (Tripwire expects it to be moved to '/etc/tw'.)
###
```

Wie Sie sehen können, wird die alte Datenbank zu *tw.rechnername.old* umbenannt und eine neue Referenzdatenbank mit den aktualisierten Informationen erstellt, welche Sie wieder in das angegebene Verzeichnis verschieben müssen.

Automatische Aktualisierung

Diese – durchaus brauchbare und praktische – Funktion ist mit Vorsicht zu genießen. Sie haben in den vorhergehenden Abschnitten gesehen, in welchen Schritten ein Integritätstest und ein nötiges Update eines Eintrages ablaufen:

1. Tripwire-Integritätstest
2. Bericht über Änderung an Objekt
3. eventuelle Aktualisierung der Datenbank
4. Im Grunde bietet die Funktion, die wir jetzt besprechen, die Möglichkeit all diese Schritte durch einen einzigen Befehl zu bewerkstelligen. Sie lassen also Tripwire nach Inkonsistenten in Ihrem Dateisystem suchen, erhalten danach einen Bericht über mögliche Treffer und können die betroffenen Objekte und Felder sofort aktualisieren. Auf den ersten Blick eine sehr praktische Funktion. Doch sie hat zwei entscheidende Nachteile:

- Der Administrator muss bei jedem Integritätstest anwesend sein, um mögliche Aktualisierungen vornehmen zu können. Bei einem normalen Test ist dies nicht notwendig. Der Systemverwalter könnte die Bildschirmausgabe zum Beispiel in eine Datei umleiten, die er sich dann automatisch zuschicken lässt, oder er könnte diese Datei mit einem Skript auf Inkonsistenten untersuchen und, falls eine Gefahrenquelle entdeckt wurde, könnte dieses Skript den Administrator automatisch per E-Mail oder SMS (mit Hilfe des Pakets *yaps*) über die Lage benachrichtigen.
- Man wird leichtfertig und vergibt Updates schneller und leichter, als man dies täte, wenn man sie per Hand eingeben müsste.

Sie müssen also selbst entscheiden, was für Sie am besten geeignet ist. Seien Sie aber vorsichtig damit und machen Sie lieber einige Handgriffe mehr.

Nachdem wir jetzt die Funktion kennen, können wir uns mit der Bildschirmausgabe, die wir bei einer gefunden Inkonsistenz (wir benutzen dafür wieder unsere Datei *test* aus dem vorhergehenden Beispiel) erhalten, beschäftigen. Der Aufruf dieser Funktion geschieht über folgenden Befehl:

```
Linux:~# tripwire -interactive
```

Und hier die dadurch entstehende Bildschirmausgabe:

```
### Phase 1:    Reading configuration file
### Phase 2:     Generating file list
### Phase 3:     Creating file information database
### Phase 4:    Searching for inconsistencies
###
###        Total files scanned:    232
###            Files added:        0
###            Files deleted:    0
###            Files changed:    232
###
###        After applying rules:
###            Changes discarded:    231
###            Changes remaining:    1
###
changed: -rw-r—r—joey    18 Jun 25 19:50:40 2001 /joey/neu/test
### Phase 5:    Generating observed/expected pairs for changed
files
###
### Attr    Observed (what it is)    Expected (what it should be)
### ======== ========================
============================
/joey/neu/test
st_ino:    137046               136996
st_size:    18                6
st_mtime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
```

```
st_ctime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
md5 (sig1):    1oDgYoDDEL.5g2tFr9W.Nm    2kwNorphIYEyAtMnhMr:kIa
----> File: '/joey/neu/test'
----> Update entry    [YN(y)nh?]
Updating entry : /joey/neu/test
### Updating database...
###
### Phase 1:    Reading configuration file
### Phase 2:    Generating file list
Updating: update entry: /joey/neu/test
### Phase 3:    Updating file information database
###
### Old database file will be move to 'tw.rechnername.old'
###        in ./databases
###
### Updated database will be stored in './databases/
tw.rechnername'
###        (Tripwire expects it to be moved to '/etc/tw'.)
###
###
### If you changed the tw.config file, remember to run
### 'twdb_check.pl' to ensure database consistency.
### See the README file for details
```

Anhand dieses Listings können Sie gut erkennen, wie einfach es einem gestressten Systemverwalter gemacht wird, bestimmte Einträge zu aktualisieren.

Wie Sie sehen können, läuft der Integritätstest genauso ab wie beim normalen Aufruf von Tripwire. Doch am Ende des Tests werden Sie gefragt, ob Sie die auffälligen Einträge aktualisieren wollen. Dabei stehen Ihnen grundsätzlich drei Optionen zur Verfügung.

1. Nein (`Nn`)
2. Ja (`Yy` und Defaultwert, falls Sie keine Angabe machen)
3. Aufruf einer kleinen Hilfefunktion (`h?`)
4. An Punkt zwei können Sie eine weitere Tücke erkennen. Falls Sie keine Angaben über die Aktualisierung eines Eintrages machen, so kommt der Defaultwert zum Zuge, der ein Update des Eintrages initialisiert.
5. Nachdem Sie Tripwire mitgeteilt haben, welche Einträge Sie aktualisieren möchten, erhalten Sie dieselben Ausgaben wie beim normalen Updatemodus. Mit einem Unterschied:
6. Sie werden darauf hingewiesen, das Skript *twdb_check.pl* aufzurufen, um die Datenbank zu überprüfen. Mit Hilfe dieses Befehls (er sollte sich bei der Installation automatisch nach */usr/sbin* kopieren) können Sie die Integrität der

Datenbank testen. Die Syntax ist –wie Sie sehen– einfach:

```
twdb_check.pl <tw.db_rechnername> <tw.config>
```

Test–tuning

Wenn Sie Tripwire nutzen wollen, um sehr große Datenmengen zu überprüfen, merken Sie bald, dass jeder Test viel Rechenzeit verschlingt und eine gewisse Zeit bis zur Abarbeitung benötigt. Tripwire bietet Ihnen mit der `--ignore`-Funktion ein Hilfsmittel, um schnelle Tests durchzuführen. Doch auch dieses Bonbon ist mit Vorsicht zu genießen, da die Funktion den Performancegewinn durch das Ausblenden bestimmter spezifizierter Auswahlmasken erreicht, was wiederum zu einem enormen Informationsverlust führen kann. Es wäre aber durchaus denkbar, dass Sie zum Beispiel täglich einen kompletten Integritätstest durchführen und sich bei der stündlichen Überprüfung Ihres Systems auf die abgespeckte Variante verlassen. Der Aufruf dieser Funktion erfolgt über folgende Syntax:

```
Linux:~# tripwire --ignore Selection Mask
```

Falls diese Funktion bei Ihnen nicht funktioniert, haben Sie Möglichkeit (außer bei den brandneuen Versionen) auf die Funktion `loosedir` auszuweichen, die grundsätzlich ähnliche Vorteile bietet. Sie können aber dabei (im Gegensatz zu `ignore`) selbst bestimmen, welche Masken Sie ignorieren wollen. Es werden automatisch folgende Flags ausgeblendet:

- st_size
- st_nlinks
- st_mtime
- st_ino

Auch hier erfolgt der Aufruf in gewohnter Manier:

```
Linux: tripwire –loosedir
```

Die Bildschirmausgabe, die Sie dabei erhalten, unterscheidet sich nicht von der eines normalen Integritätstests und wird daher nicht extra aufgeführt.

Alternative Datenbank

Falls Sie mehrere Datenbanken auf Ihrem System gespeichert haben, können Sie auch eine alternative Datenbank spezifizieren. Tripwire wird dann diese anstatt der Datenbank, die sich in dem Defaultverzeichnis befindet, verwenden.

Der Aufruf erfolgt über folgende Syntax:

```
Linux:~# tripwire --dbfile Datenbank
```

Auch hier führen wir die Alternative für andere Systeme (Versionen) auf:

```
Linux:~# tripwire –d Datenbank
```

Alternative Konfigurationsdatei

Neben einer alternativen Datenbank können Sie bei Tripwire auch eine alternative Konfigurationsdatei angeben, die dann anstelle der Konfigurationsdatei, welche sich im Defaultverzeichnis befindet, verwendet wird.

Der Aufruf verbirgt auch bei dieser Funktion keine großen Überraschungen:

```
Linux:~# tripwire --cfgfile Konfigurationsdatei
```

Auch hier führen wir die Alternative für andere Systeme (Versionen) auf:

```
Linux:~# tripwire –c Konfigurationsdatei
```

Datenbank aus File Descriptor lesen

Eine äußerst praktische und ressourcensparende Methode, ideal für Verschlüsselung und Packen. Die Syntax ist auch hier sehr einfach gehalten:

```
Linux:~# tripwire –dfd Open File Descriptor
```

Konfigurationsdatei aus File Descriptor lesen

Sie haben auch die Möglichkeit, die Konfigurationsdatei aus einem File Descriptor zu lesen. Die Syntax ist den bereits bekannten Befehlen sehr ähnlich:

```
Linux:~# tripwire –cfd Open File Discriptor
```

Variablen definieren

Sie haben ja bereits eine Möglichkeit kennen gelernt, um Variablen zu definieren. Außer dem Preprocessor `@@define` gibt es aber noch einen anderen Weg, das zu erledigen. Sie können mit dem Befehl `–Dvar` eine Variable als String definieren. Die Initialisierung und Bildschirmausgabe können Sie im folgenden Listing ersehen:

```
Linux:~# tripwire –Dvar=string
```

Zum besseren Verständnis werfen Sie einen Blick auf die Bildschirmausgabe:

```
Linux:~# tripwire –DOS=LINUX
### Phase 1:    Reading configuration file
### Phase 2:     Generating file list
### Phase 3:     Creating file information database
### Phase 4:    Searching for inconsistencies
###
###        Total files scanned:    232
###            Files added:        0
###            Files deleted:    0
```

```
###            Files changed:    232
###
###        After applying rules:
###            Changes discarded:    231
###            Changes remaining:    1
###
changed: -rw-r—r—joey    18 Jun 25 19:50:40 2001 /joey/neu/test
### Phase 5:    Generating observed/expected pairs for changed
files
###
### Attr    Observed (what it is)    Expected (what it should be)
### ======== ========================
============================
.
.
.
```

Wie Sie der Initialisierung entnehmen können, haben Sie mit diesem Befehl die Möglichkeit einer Variable einen bestimmten Wert zuzuweisen. Nachdem Sie das getan haben, wird die Initialisierung – mit der neuen Variable – wie gewohnt durchgeführt.

Variablendefinition rückgängig machen

Nachdem Sie nun einen Weg gesehen haben, der es Ihnen erlaubt, Variablen von der Konsole aus zu definieren, werden Sie jetzt das Gegenstück dazu kennen lernen, mit dem Sie die Möglichkeit haben, definierte Variablen zu »un-definieren« (sprich die Definition der Variable rückgängig zu machen):

```
Linux:~# tripwire –Uvar
```

Auch hier ein kleiner Ausschnitt aus der Bildschirmausgabe:

```
Linux:~# tripwire –UOS
### Phase 1:    Reading configuration file
### Phase 2:     Generating file list
### Phase 3:     Creating file information database
### Phase 4:    Searching for inconsistencies
###
###        Total files scanned:    232
###            Files added:        0
.
.
.
```

Genauso wie bei der Definition bestimmter Variablen wird bei diesem Befehl der Integritätstest wie gewohnt durchgeführt.

Signaturtest unterbinden

Es kann durchaus Situationen im Leben eines Administrators geben, bei denen er einen schnellen Integritätstest braucht. Wir haben ja bereits einige Möglichkeiten gesehen, wie man dies erreichen kann. Einer der leichtesten Wege den Test zu beschleunigen ist, nur eine oder wenige Signaturen anfertigen zu lassen (ein stabiler, aussagekräftiger und sicherer Integritätstest benutzt mehrere Algorithmen, um die Signatur zu bestimmen). Das folgende Listing zeigt die Signaturbestimmung unserer beliebten Datei *test* durch die Algorithmen MD5 und snefru.

```
.
.
.
### Attr    Observed (what it is)    Expected (what it should be)
### ======== ========================
============================
/joey/neu/test
st_ino:    137046                136996
st_size:       18                     6
st_mtime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
st_ctime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
md5 (sig1):    1oDgYoDDEL.5g2tFr9W.Nm    2kwNorphIYEyAtMnhMr:kIa
snefru (sig2):2.ZirbrL3ZN289TyymBJ5H    27shvz29Wzo.DeyYN38no9
```

Natürlich braucht der Rechner einige Zeit, bis er die zu bestimmenden Signaturen berechnet hat, wie bereits erwähnt, können Sie Tripwire dazu bringen, bestimmte Signaturberechnungen zu übergehen. Der Befehl hierfür muss lauten:

```
Linux:~# tripwire –i Algorithmen
```

Falls Sie mehrere Algorithmen übergehen möchten, müssen Sie diese durch Kommas voneinander trennen. Das folgende Listing zeigt die Bildschirmausgabe, die wir erhalten, wenn wir wollen, dass nur durch snefru eine Signatur berechnet wird. Die MD5–Berechnung soll also wegfallen.

```
Linux:~# tripwire –i 1
.
.
.
### Attr    Observed (what it is)    Expected (what it should be)
### ======== ========================
============================
/joey/neu/test
st_ino:    137046                136996
st_size:       18                     6
st_mtime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
st_ctime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
snefru (sig2):2.ZirbrL3ZN289TyymBJ5H    27shvz29Wzo.DeyYN38no9
```

Wie Sie sehen können, wurde die MD5–Berechnung nicht ausgeführt und der Vorgang beschleunigt. Falls Sie wollen, dass überhaupt keine Signaturberechnung durchgeführt wird, können Sie dies mit folgendem Befehl verwirklichen:

```
Linux:~# tripwire –i all
```

Die Bildschirmausgabe hierzu lautet:

```
.
.
.
### Attr    Observed (what it is)    Expected (what it should be)
### ======== ========================
============================
/joey/neu/test
st_ino:    137046                136996
st_size:   18                    6
st_mtime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
st_ctime:    Mon Jun 25 19:50:40:2001    Mon Jun 25 19:49:31 2001
```

Durch die Eingabe des oben aufgeführten Befehls wurde keine Signaturberechnung durchgeführt.

Konfiguration testen

Mit der folgenden Funktion haben Sie die Möglichkeit Ihre Konfiguration zu testen, da bei diesem Befehl nur ein »preprocess« der Konfigurationsdatei zu machen und diese dann auf die Standardausgabe zu werfen ist. Der Aufruf dieses Befehls lautet:

```
Linux:~# tripwire –E
```

oder

```
Linux:~# tripwire –preprocess
```

Minimaler Output

Falls Sie das Output eines Integritätstests nicht in eine Datei zur Weiterverarbeitung durch ein Skript geben, kann es sehr lästig sein, wenn viele Anomalien entdeckt wurden, da die Ausgabe dadurch enorm lang wird. Abhilfe schafft der hier vorgestellte Befehl, da er Tripwire veranlasst jeweils nur eine Zeile pro geänderter Datei auszugeben. Der Aufruf erfolgt über folgenden Befehl:

```
Linux:~# tripwire –q
```

Falls das bei Ihnen nicht funktioniert, versuchen Sie bitte folgende Alternative (diese Option hat zwar nicht exakt dieselben Eigenschaften, aber sie schränkt ebenfalls die Ausgabe ein):

```
Linux:~# tripwire –quiet
```

Maximaler Output

Mit dem *Verbose Modus* haben Sie die Möglichkeit alle von Tripwire gescannten Dateien auf die Standardausgabe werfen zu lassen. Falls Ihnen diese Option sinnlos erscheint, bedenken Sie, dass sie bei der Fehlersuche enorm hilfreich sein kann. Stellen Sie sich dazu einfach vor, Sie haben die Vermutung, dass Tripwire – aus irgendwelchen Gründen– die Datei */joey/neu/test2* nicht scannt. Mit dem Verbose Modus können Sie dies leicht überprüfen. Leiten Sie dazu die Ausgabe des Verbose Modus einfach in eine Datei um. Danach gehen Sie in das betreffende Verzeichnis (*/joey/neu*) und geben dort den `find` Befehl ein, dessen Ausgabe Sie wiederum in eine Datei umleiten. Danach benutzen Sie den `diff`-Befehl, um zu sehen, ob Tripwire die Datei *test2* wirklich nicht scannt.

Hilfe über Inodes

Falls Sie mehr über die Inodes erfahren wollen, können Sie diesen Befehl benutzen:

```
Linux:~# tripwire –help
```

Versionsnummer ausgeben lassen

Um sich die Versionsnummer ausgeben zu lassen, benutzen Sie bitte folgenden Befehl:

```
Linux:~# tripwire –version
```

4.1.7 Schutz der Datenbank

Nach der Lektüre des bisherigen Kapitels sollten Sie fit im Umgang mit Tripwire sein. Bevor ich Sie aber »aus diesem Kurs entlassen« kann, möchte ich noch ein Wort über die Datenbank und den Schutz dafür verlieren.

Wir haben bereits bei der Initialisierung gesehen, dass Ihnen Tripwire vorschlägt, die Datenbank auf ein sicheres (`READ-ONLY`) Medium zu verfrachten. Ich möchte Ihnen hierbei auch ans Herz legen, dies ernst zu nehmen. Dabei ist es wichtig, dass das Medium selbst die `READ-ONLY` Flag gesetzt hat, das heißt, die `READ-ONLY`-Flag sollte physikalisch und nicht logisch sein. Denn logischerweise kann jede logische Barriere durch logisches Handeln wieder aufgehoben oder zumindest umgangen werden. Mein Vorschlag (ich weiß, ich wiederhole mich bei diesem Punkt, aber er ist wirklich wichtig) ist daher, die Datenbank auf eine CD zu brennen, diese CD in ein CD-Rom bzw. ein DVD-Rom zu legen und ins Dateisystem einzuhängen. Zusätzlich zu diesem Schutz können Sie mit dem Tripwire–Tool *siggen* die Signatur der Datenbank überprüfen (praktisch, wenn Sie keinen lokalen Zugriff auf die CD-Rom haben und sie theoretisch unbemerkt ausgewechselt werden könnte). Die Handhabung dieses Tools ist einfach.

Nachdem Sie die Datenbank erstellt und auf die CD gebrannt haben, wenden Sie das Tool an, um die Signatur zu berechnen. Danach können Sie ein Skript schreiben, welches durch einen Croneintrag täglich überprüft, ob sich die Signatur und damit die Authentizität der Datenbank geändert hat. Dies mag etwas paranoid erscheinen, bietet aber einen ziemlich sicheren Weg, um Angreifern wieder eine Barriere mehr in den Weg zu stellen. Die Syntax für das *siggen*-Tool lautet wie folgt:

```
Linux:~# siggen –Option Konfigurationsdatei
```

Tabelle 4.5 enthält alle verfügbaren Optionen zu siggen.

Option	Beschreibung
-h	Gibt die Signatur hexadezimal anstatt base 64 aus
-q	Gibt alle Signaturen in einer Zeile aus und lässt somit weitergehende Informationen weg
-a	Es werden alle Signaturen berechnet (Defaultwert)
-v	Verbose Modus (Defaultwert)
-0	Berechnet 0 Signatur
-1	Berechnet MD5 Signatur
-2	Berechnet snefru Signatur
-3	Berechnet CRC 32 Signatur
-4	Berechnet CRC 16 Signatur
-5	Berechnet MD4 Signatur
-6	Berechnet MD2 Signatur
-7	Berechnet SHA Signatur
-8	Berechnet Haval Signatur
-9	Berechnet 0 Signatur (Reserviert für zukünftigen Gebrauch)

Tabelle 4.5 Verfügbare Optionen für siggen

Um den Abschnitt über Tripwire endgültig abzuschließen, können Sie die Verwendung von `siggen` anhand des folgenden Listings einsehen:

```
Linux:~# siggen /etc/tw/tw.linux
sig0: nullsig: 0
sig1: md5    : 1UESQLg04KTxCswAXxnkr
sig2: snefru : 1pdbqHak2P42KbnlsyEcqA
sig3: crc32  : 17UDxV
sig4: crc16  : 000EJQ
sig5: md4    : 0jUn3uxIMxJ2gTe.:pIeoB
sig6: md2    : 1j9rITQBzWsFqgjV.9Qmdq
sig7: sha    : 677qhN468WvtzpuhvW4IZaptiLV
sig8: haval  : 0v5vAv59leWQm4b0UxGMH9
sig9: nullsig: 0
```

4.2 MD5Sum

Wenn es um die Installation eines Tools ging, habe ich im Laufe des Buches sehr oft gesagt, dass Sie die Herkunft des Programms genau überprüfen sollen. Außerdem habe ich Ihnen des Öfteren ans Herz gelegt, dass Sie das frisch heruntergeladene Tool auf Echtheit hin überprüfen.

Zumeist wird dafür auf den Downloadseiten eine MD5-Checksum angeboten. Wir werden in diesem Abschnitt des Buches nun das Tool besprechen, mit dem Sie 128-Bit-Prüfsummen leicht überprüfen können.

4.2.1 Syntax

Da dieses Tool nicht weiter beschrieben werden muss, können wir uns sofort mit der zu verwendenden Syntax beschäftigen.

Aufruf

Der allgemeine Aufruf dieses Programms lautet wie folgt:

```
Linux:~# md5sum [Optionen] Datei
```

Falls Sie keine Datei oder den Platzhalter »-« stattdessen angeben, liest MD5sum von der Standardeingabe (STDIN).

Optionen

Da dieses Tool alles andere als schwer zu bedienen ist, werden wir uns sofort mit den verfügbaren Optionen, die Sie in Tabelle 4.6 erkennen können, beschäftigen.

Option	Beschreibung
-b, --binary	Hiermit werden alle Eingabedateien als Binarydateien behandelt. Für uns Linuxbenutzer spielt diese Option keine große Rolle, da UNIX-Systeme nicht zwischen Textdateien und Binaries unterscheiden.
--status	Diese Option ist nur dann sinnvoll, wenn Sie Checksummen überprüfen.
-t, --text	Hiermit werden alle Binarydateien als Textfiles behandelt. Diese Option ist das Gegenstück zur Option »--binary«.
-w, --warn	Wenn Checksummen überprüft werden, wird bei nicht richtig formatierten MD5-Checksummen gewarnt. Diese Option kann sehr hilfreich sein, wenn alle Checksummen mit einigen wenigen Ausnahmen als gültig erkannt wurden.

Tabelle 4.6 Verfügbare Optionen